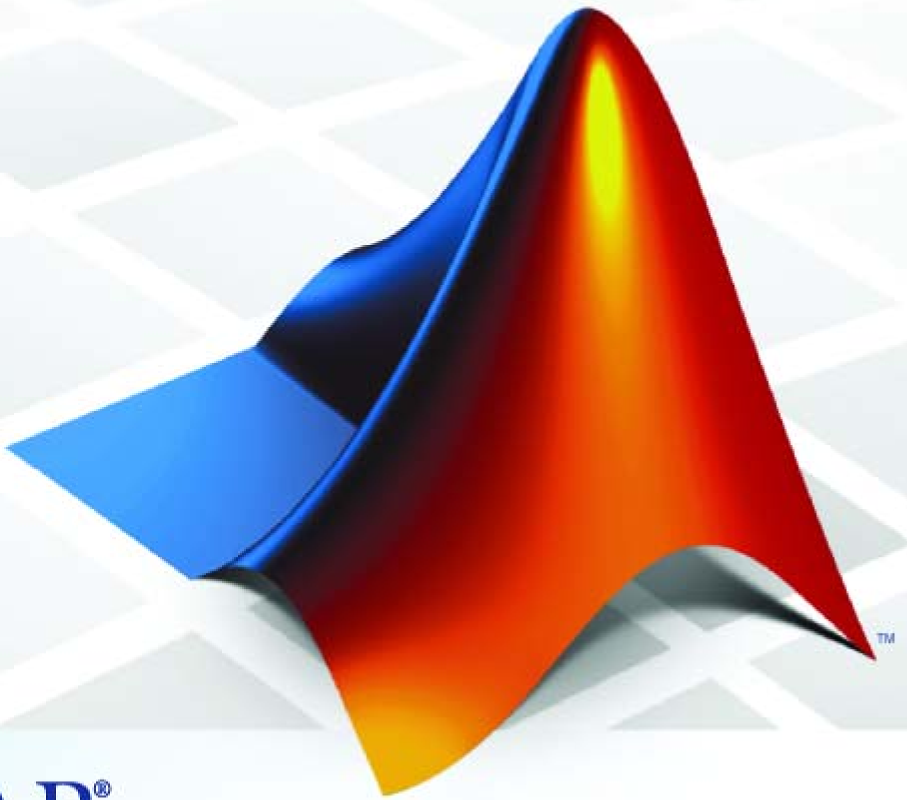
Simulink® Control Design™ 2
User's Guide



MATLAB®
& SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

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)
September 2006	Online only	Revised for Version 2.0.1 (Release 2006b)
March 2007	Online only	Revised for Version 2.1 (Release 2007a)
September 2007	Online only	Revised for Version 2.2 (Release 2007b)
March 2008	Online only	Revised for Version 2.3 (Release 2008a)
October 2008	Online only	Revised for Version 2.4 (Release 2008b)

Contents

1 Working with Simulink® Control Design Projects

2 Specifying Operating Points

5 Specifying Operating Points Using Functions

6 Linearizing Models Using Functions

7 Understanding Analysis in the Simulink® Control Design Software

Working with Simulink Control Design Projects

Beginning a Project

In this section...

Creating a Simulink Control Design Project

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

To open a new Simulink Control Design project:

1 Select **Start > Simulink > Simulink Control Design > Linearization Task** or select **Start > Simulink > Simulink Control Design > Simulink Compensator Design Task.**

2 Enter a project name, select a model to analyze, and choose the tasks you want to perform. Click **OK** to close the dialog box and open the new project.

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

Creating an Operating Points Task

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

Creating a Linearization Task

To create a linearization task in the Control and Estimation Tools Manager, use one of the methods in "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 the Simulink Control Design getting started documentation for more information.

Saving Projects

See “Saving Projects” in the Simulink Control Design getting started documentation for more information.

Specifying Operating Points

Creating Operating Points

In this section...

Role of Operating Points in Linearization

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

Creating Operating Points from Specification

You can specify target values or constraints on a subset of the model's inputs, outputs, and states (see "Creating Operating Points from Specifications" in the Simulink Control Design getting started documentation). The software uses numerical optimization 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
- Your initial guess for the states in the Controller is 0

then, you can compute an operating point that closely matches the specifications.

Creating Operating Points from Known Values

You can completely specify all inputs and states in the operating point (see "Creating Operating Points from Known Values" in the Simulink Control Design getting started 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
- You also know the values of the states in the Controller

then, this information completely specifies the operating point.

Creating Operating Points From Simulation

You can use the following methods for creating an operating point from a simulation of your model:

- "Creating Operating Points at Specified Simulations Times" on page 2-3
- "Creating Operating Points at Simulation Events" on page 2-4

Creating Operating Points at Specified Simulations Times

You can extract an operating point at specified times during a simulation of the model (see "Creating Operating Points from Simulation" in the Simulink Control Design getting started 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 approach is especially useful when the simulation has reached a steady state.

Creating Operating Points at Simulation Events

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

- Trigger-based events
- Function-call events

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

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

To create operating points at one or more simulation events:

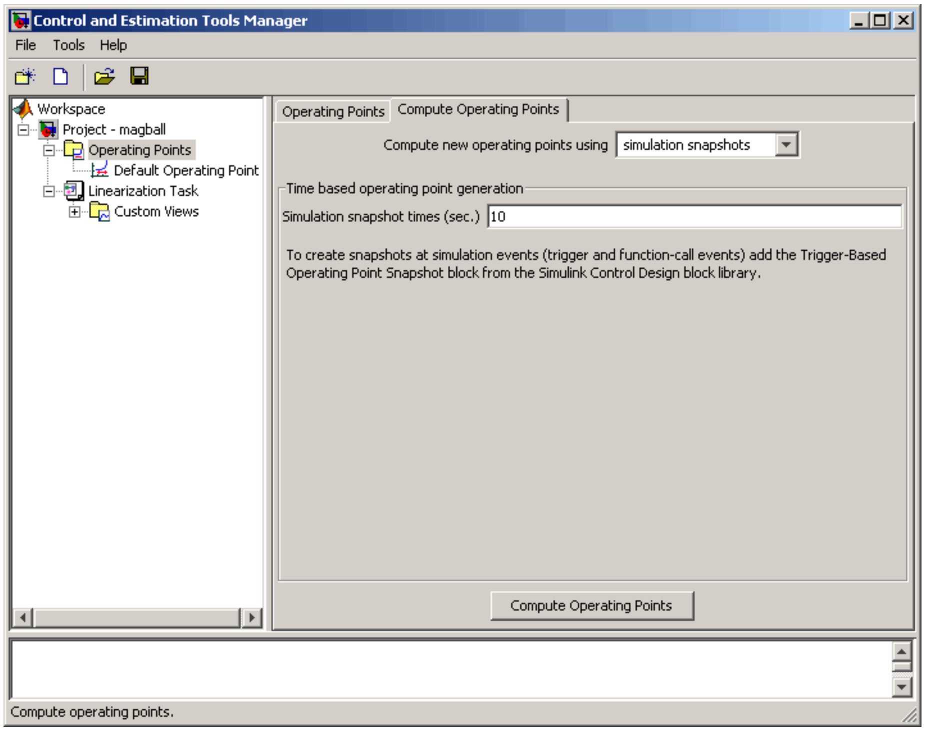
1. Add a Trigger-Based Operating Point Snapshot block to your model. This block is in the Simulink Control Design block library.

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

2. Select the **Compute Operating Points** tab in the **Operating Points** node.

3. From the **Compute new operating points using** list, select `simulation snapshots`.

4. Enter a scalar value that specifies the simulation end time in the **Simulation snapshot times (sec.)** field, shown in the following figure.



Compute new operating points using: simulation snapshots

Time based operating point generation

Simulation snapshot times (sec.): 10

To create snapshots at simulation events (trigger and function-call events) add the Trigger-Based Operating Point Snapshot block from the Simulink Control Design block library.

Compute operating points.

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

Using the Default Operating Point

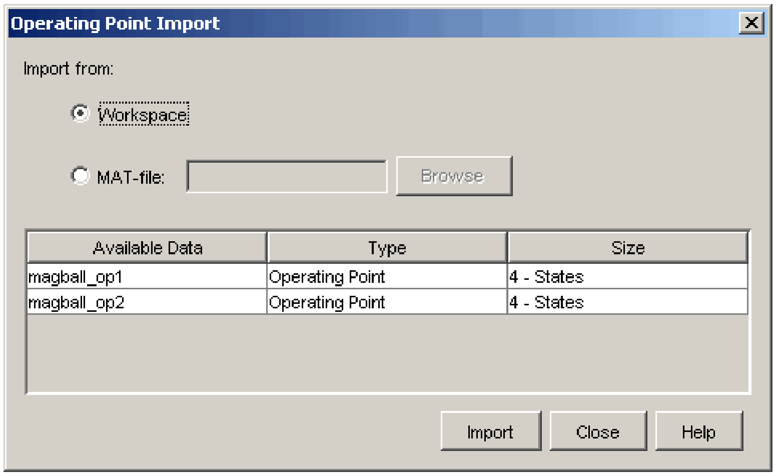
You can choose to accept the default operating point in the Simulink Control Design software. The initial values of the states, inputs, and outputs, define this operating point. For more information on using the Default Operating Point see “Simulink Control Design Default Operating Point” in the Simulink Control Design getting started documentation. Only use the default operating point when the initial values are very close to the operating point of interest.

Importing Operating Points

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

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

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



Available Data	Type	Size
magball_op1	Operating Point	4 - States
magball_op2	Operating Point	4 - States

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.

Computing Equilibrium Operating Points

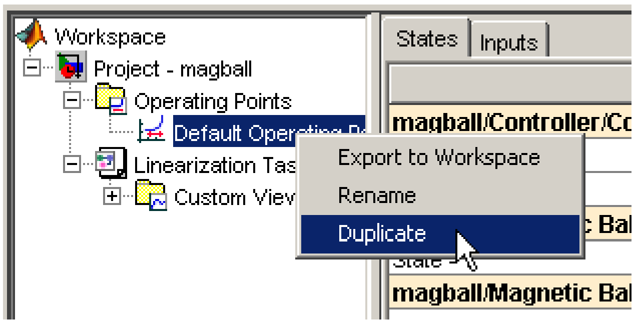
You can use the software to compute equilibrium operating points. Follow the basic instructions in “Creating Operating Points from Specifications” in the Simulink Control Design getting started documentation. When you enter specifications in the **States** pane, select the **Steady State** check box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

Working with Operating Points

In this section...

Copying Operating Points

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

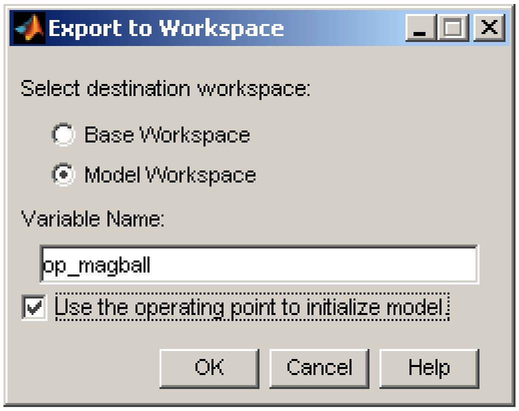


The new operating point appears beneath the original one in the tree. Click the new operating point to display its contents in the pane on the right. To change state or input values in the duplicated operating point, edit the values in the right pane. To change the name of the new operating point, right-click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

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

Exporting Operating Points

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



Select destination workspace:

( ) Base Workspace

(●) Model Workspace

Variable Name: op_magball

☑ Use the operating point to initialize model

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

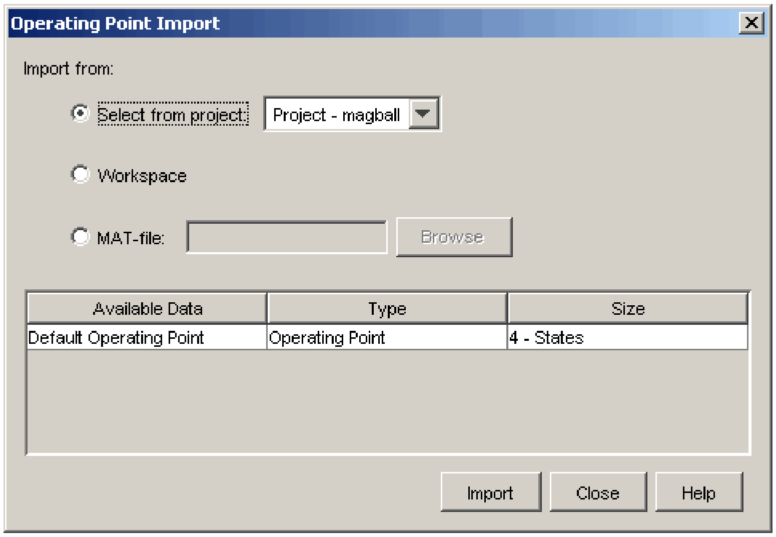
- **Model Workspace** to export the operating point to the Model workspace where you can save it with the model for future use.

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.

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.


Operating Point Import

Import from:

(●) Select from project: Project - magball

( ) Workspace

( ) MAT-file:

Available Data	Type	Size
Default Operating Point	Operating Point	4 - States

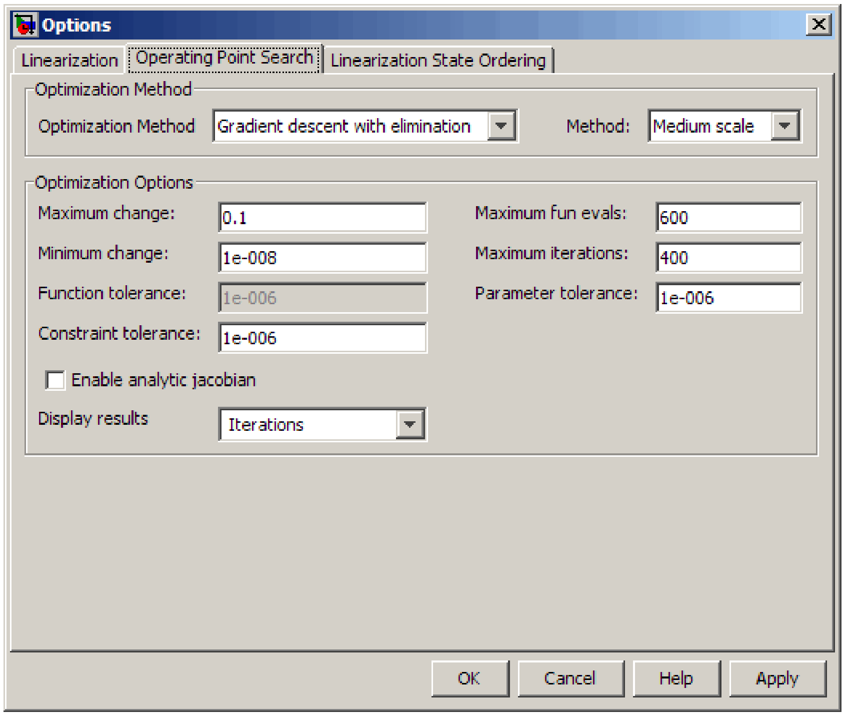
Select where to import the initial values from (a project, the workspace, or a file), then select the operating point from the list of available operating points below (or in the case of MAT-files, browse for a file). Click **Import** to import the initial values from the selected operating point into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

Constraining Outputs

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

Changing Optimization Settings

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



Optimization Method

Optimization Method: Gradient descent with elimination
Method: Medium scale

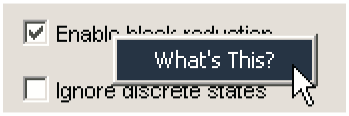
Optimization Options

Maximum change:	0.1	Maximum fun evals:	600
Minimum change:	1e-008	Maximum iterations:	400
Function tolerance:	1e-006	Parameter tolerance:	1e-006
Constraint tolerance:	1e-006		

☐ Enable analytic jacobian

Display results: Iterations

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



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

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

The methods `Gradient descent with elimination`, `Simplex search`, and `Nonlinear least squares` refer to the optimization methods `fmincon`, `fminsearch`, and `lsqnonlin` respectively. The method `Gradient descent` refers to the optimization method `graddescent`, described in the `linoptions` reference page. The reference page for the Optimization Toolbox function `optimset` contains documentation for the following operating point search settings (the corresponding `optimset` parameter values are given in parentheses):

Operating Point Search Option	Parameter in `optimset`
Large Scale	`LargeScale` set to `'on'`
Medium Scale	`LargeScale` set to `'off'`
Maximum change	`DiffMaxChange`
Minimum change	`DiffMinChange`
Function tolerance	`TolFun`
Constraint tolerance	`TolCon`
Maximum fun evals	`MaxFunEvals`
Maximum iterations	`MaxIter`
Parameter tolerance	`TolX`

Enable analytic jacobian	`Jacobian`. When this option is selected, the Jacobian is computed at each iteration by linearizing the model about the current operating point. This option does not work with models that contain references to other models using the Model block or with models from products based on the Simscape™ platform that are in Trimming mode.
Display results	Display information contained in the output variable of the optimization functions, such as number of iterations, stepsize, etc.

Computing Operating Points for Blocks with Special Behavior

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

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".

Creating Linearized Models

Linearization Background

In this section...

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:

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

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

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

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

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

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

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

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

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

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

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

where $k_1, \ldots, k_\mathrm{m}$ 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 in the Control and Estimation Tools Manager

In this section...

Linearizing at an Operating Point

You can use linearize around operating points. Refer to "What Are Operating Points?" in the Simulink Control Design getting started documentation for more information on the role of operating points in linearization.

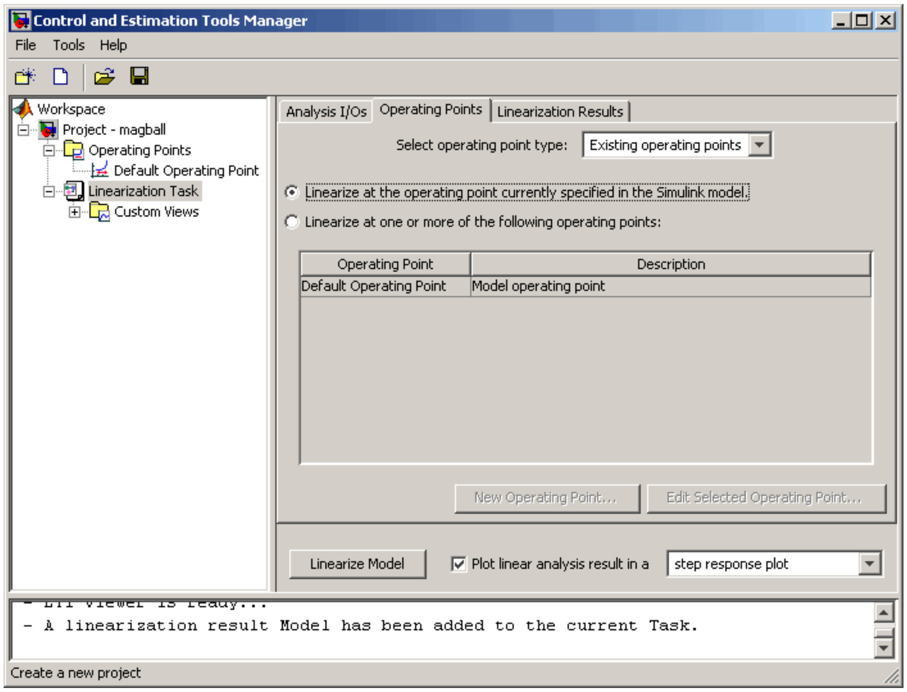
This section contains the following topics:

Linearizing at a Simulink Model Operating Point

To linearize around the operating point in the Simulink model:

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

2 Select the **Linearize at the operating point currently specified in the Simulink model** option button. This button is selected by default.



Select operating point type: Existing operating points

(●) Linearize at the operating point currently specified in the Simulink model.

( ) Linearize at one or more of the following operating points:

Operating Point	Description
Default Operating Point	Model operating point

[✓] Plot linear analysis result in a step response plot

```
- LTI Viewer is ready...
- A linearization result Model has been added to the current Task.
```

Create a new project

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

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

Linearizing at Captured Operating Points

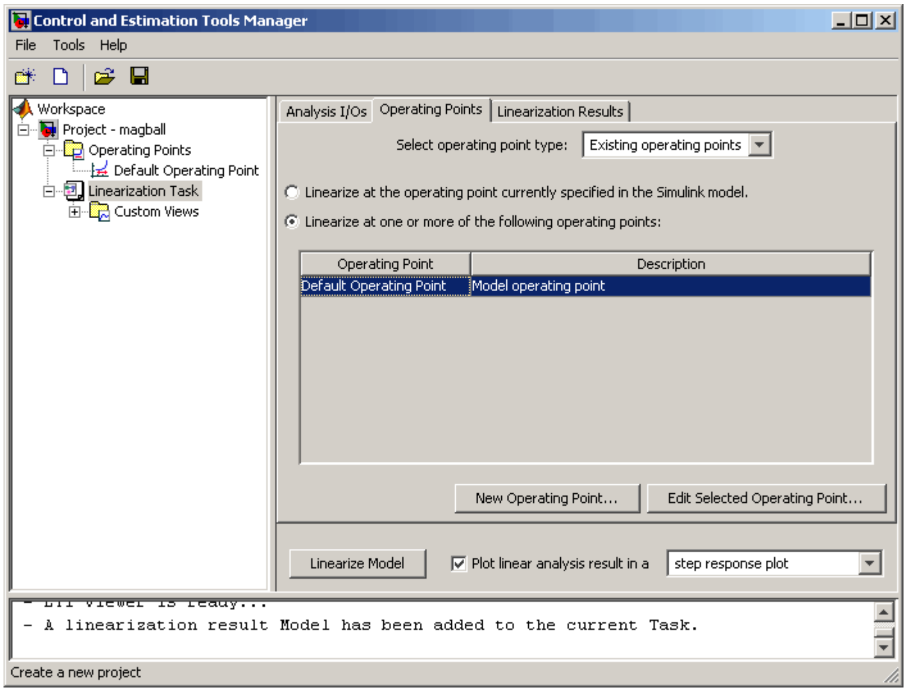
You can linearize around operating points that you captured in the **Operating Points** node.

To linearize around one or more operating points:

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

2 Select the **Linearize at one or more of the following operating points** option button.

3 In the **Operating Point** list, select one or more operating points around which to linearize the model.



Select operating point type: Existing operating points

( ) Linearize at the operating point currently specified in the Simulink model.

(●) Linearize at one or more of the following operating points:

Operating Point	Description
Default Operating Point	Model operating point

☑ Plot linear analysis result in a step response plot

```
- LTI Viewer is ready...
- A linearization result Model has been added to the current Task.
```

4 Click **Linearize Model**. The software linearizes around these operating points and adds the linearization result, labeled **Model**, to the **Linearization Task** node.

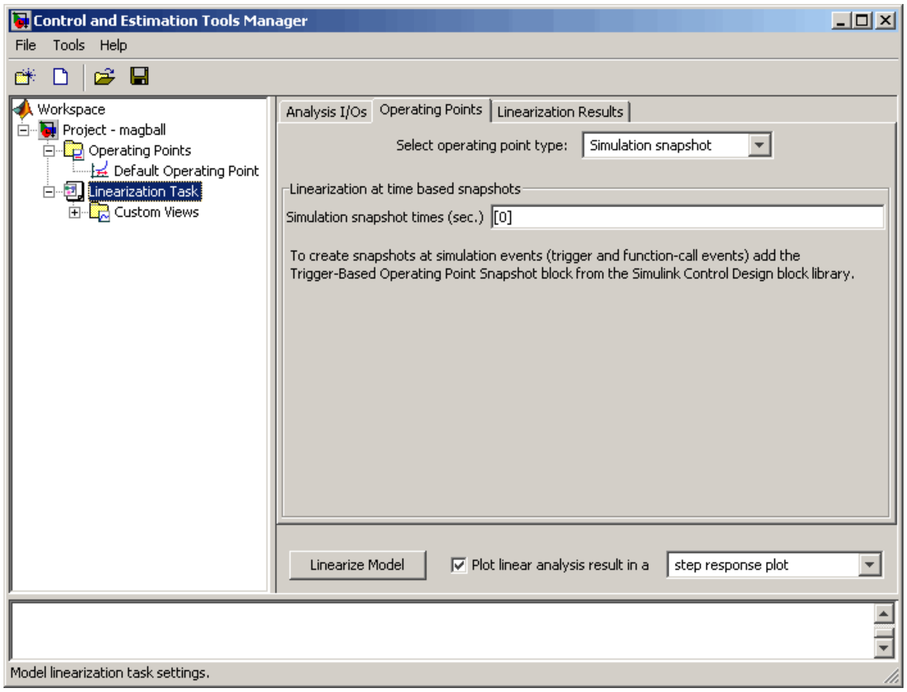
Linearizing at Specified Simulation Times

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

To linearize around one or more simulation times:

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

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



Select operating point type: Simulation snapshot

Linearization at time based snapshots

Simulation snapshot times (sec.) [0]

To create snapshots at simulation events (trigger and function-call events) add the Trigger-Based Operating Point Snapshot block from the Simulink Control Design block library.

☑ Plot linear analysis result in a step response plot

Model linearization task settings.

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

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

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

Linearizing at Simulation Events

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

- Trigger-based events
- Function-call events

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

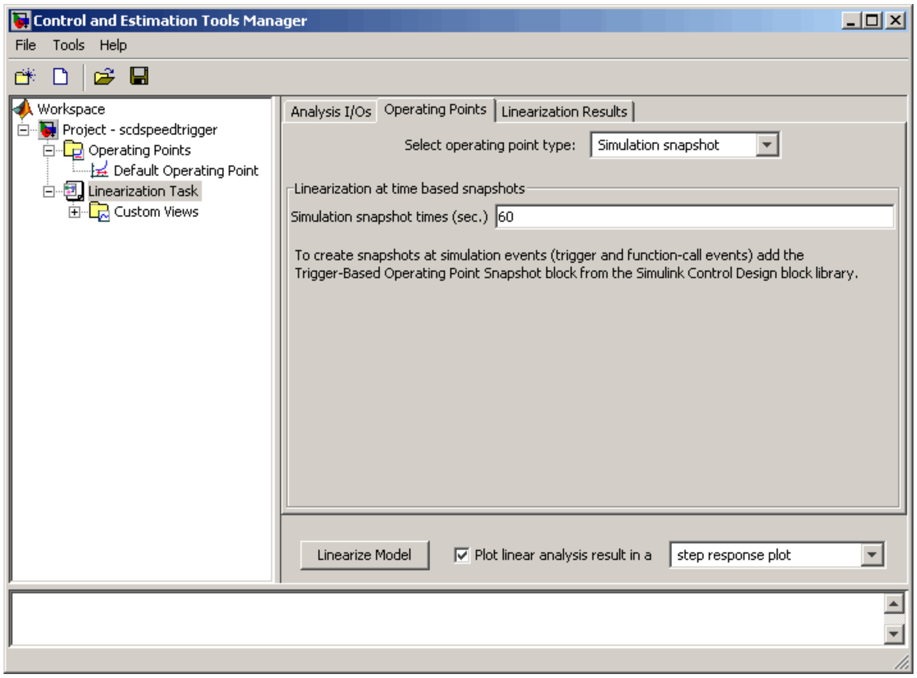
The Simulink Control Designn software linearizes around the operating points of all simulation events within a specified simulation time. The linearization takes into account all states in the model operating point.

To linearize around one or more simulation events:

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

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

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



Select operating point type: Simulation snapshot

Linearization at time based snapshots

Simulation snapshot times (sec.) 60

To create snapshots at simulation events (trigger and function-call events) add the Trigger-Based Operating Point Snapshot block from the Simulink Control Design block library.

☑ Plot linear analysis result in a step response plot

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

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

Analyzing Linearization Results

For information on analyzing the linearization results, see “Viewing Linearization Results” in the Simulink Control Design getting started documentation.

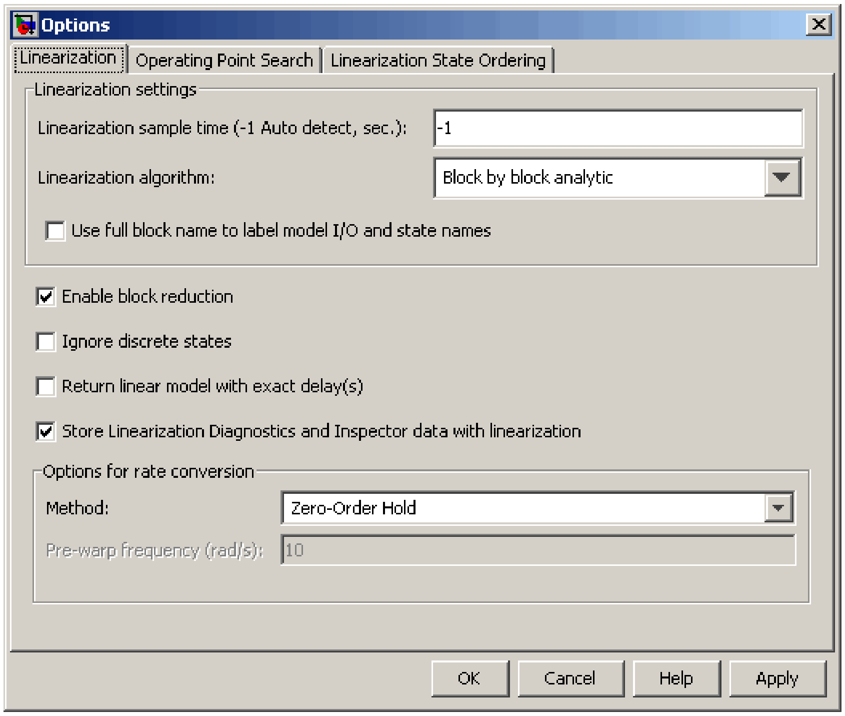
Changing Linearization Options

You have many options for controlling and modifying the results of a linearization. This section describes these options in the following topics:

- "Changing Linearization Settings and Algorithms" on page 3-12
- "Changing State Ordering in the Linearized Model" on page 3-13

Changing Linearization Settings and Algorithms

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



Linearization settings

Linearization sample time (-1 Auto detect, sec.): -1

Linearization algorithm: Block by block analytic

[ ] Use full block name to label model I/O and state names

[✓] Enable block reduction

[ ] Ignore discrete states

[ ] Return linear model with exact delay(s)

[✓] Store Linearization Diagnostics and Inspector data with linearization

Options for rate conversion

Method: Zero-Order Hold

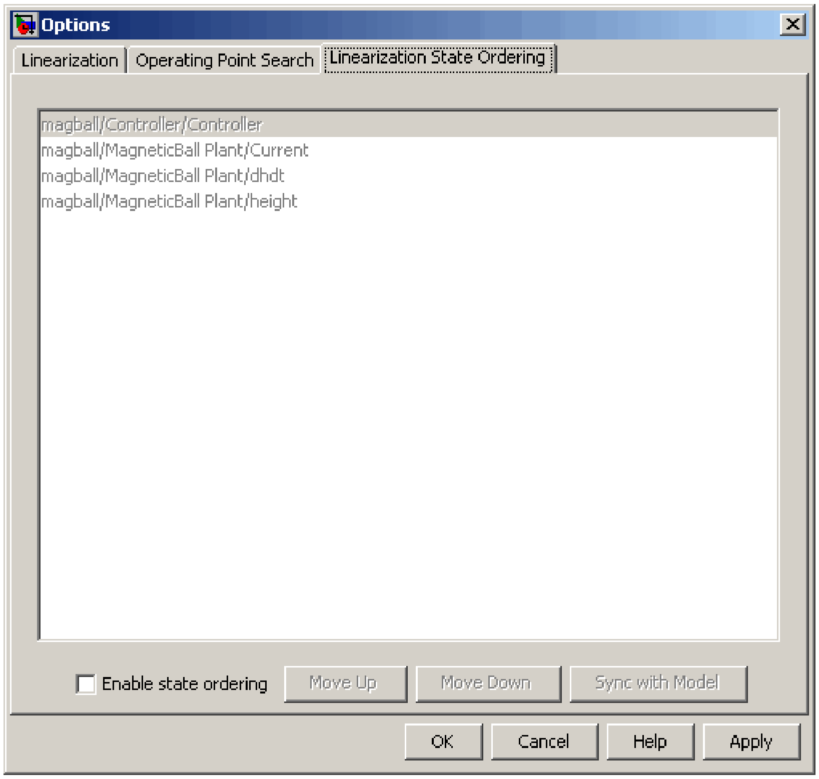
Pre-warp frequency (rad/s): 10

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

For more information on these settings, refer to the `linoptions` reference page. For information about numerical-perturbation linearization, which is used when you select `Numerical perturbation` as the **Linearization algorithm** parameter, see "Numerical-Perturbation Linearization" on page 7-4.

Changing State Ordering in the Linearized Model

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


Options
Linearization
Operating Point Search
Linearization State Ordering
magball/Controller/Controller
magball/MagneticBall Plant/Current
magball/MagneticBall Plant/dhdt
magball/MagneticBall Plant/height
Enable state ordering
Move Up
Move Down
Sync with Model
OK
Cancel
Help
Apply

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

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

Creating Other Types of Linear Models

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

- **Input-Output Point**, an input point immediately followed by an output point. This is useful for measuring sensitivity to output disturbances
- **Output-Input Point**, an output point immediately followed by an input point. This is useful for robust control. Use the resulting transfer function in mu analysis of your system.
- **Open Loop**, discussed in "Performing Open-Loop Analysis"
- **Output Constraint**, discussed in "Constraining Outputs" on page 2-11

Linearizing Discrete-Time and Multirate Models

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

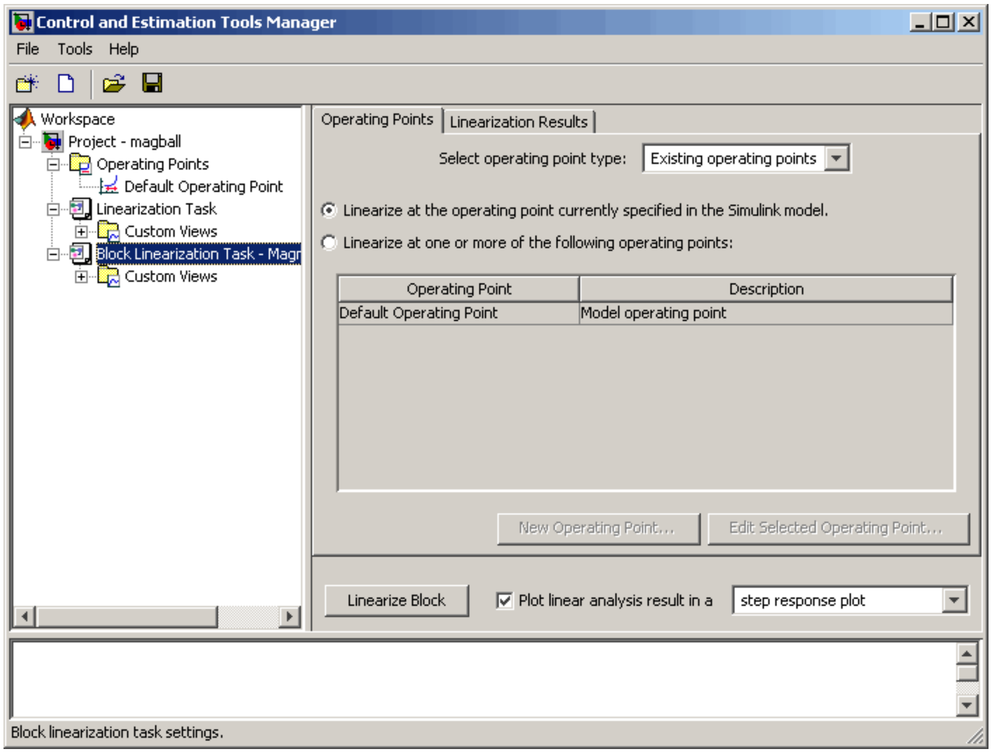
To change the method that Simulink Control Design software uses for converting a multirate model to a single-rate model, change the **Rate conversion method** in the Options dialog box.

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

Linearizing a Block

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

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



Select operating point type: Existing operating points

(●) Linearize at the operating point currently specified in the Simulink model.

( ) Linearize at one or more of the following operating points:

Operating Point	Description
Default Operating Point	Model operating point

[✓] Plot linear analysis result in a step response plot

Block linearization task settings.

To complete the linearization, specify an operating point in the same way as when linearizing models. See "Linearizing Models in the Control and Estimation Tools Manager" on page 3-6 for details. Then, click the **Linearize Block** button in the **Operating Points** pane of the **Block Linearization Task** node. You do not need to choose linearization input and output points because the inports and outports of the block are used. If you do have linearization input and output points in your model, they will be ignored. You cannot linearize an individual block using numerical-perturbation linearization (when `Numerical perturbation` is selected as the **Linearization Algorithm** parameter).

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

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 software 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 the 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 the Simulink Compensator Design Task Pane

In this section...

Tasks in the Simulink Compensator Design Task Pane

The Simulink Compensator Design Task pane lets you

- Configure blocks to tune
- Select the closed-loop response
- 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.

What Blocks Are Tunable?

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

- Gain
- PID Controller
- PID Controller (with Approximate Derivative)
- LTI System
- 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 Simulink Control Design getting started documentation.

Creating Custom Configuration Functions

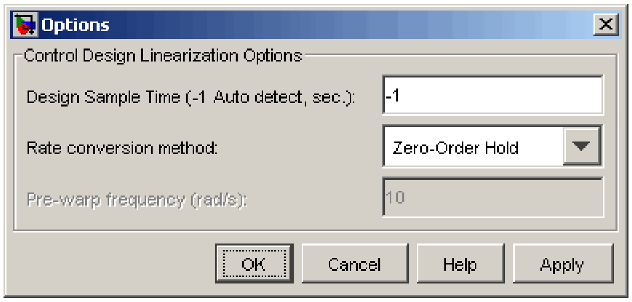
When you have masked subsystems that you want to tune in your model, they will not automatically appear in the list of tunable blocks. For them to appear in the list, you need to create a custom configuration function for the masked subsystem. The custom configuration function serves the following functions:

- It informs the Simulink Control Design software that you want this block to be available for tuning.
- It determines how you want the SISO Design Task to treat the block; it describes the relationship between the block mask parameters and the poles and zeros of the transfer function.

To learn how to create a custom configuration function, see the Simulink Control Design demo "Tuning Custom Masked Subsystems".

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.



Control Design Linearization Options

Design Sample Time (-1 Auto detect, sec.): -1

Rate conversion method: Zero-Order Hold

Pre-warp frequency (rad/s): 10

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.

Enhanced SISO Design Task

In this section...

Tools for Compensator Design in the Enhanced SISO Design Task

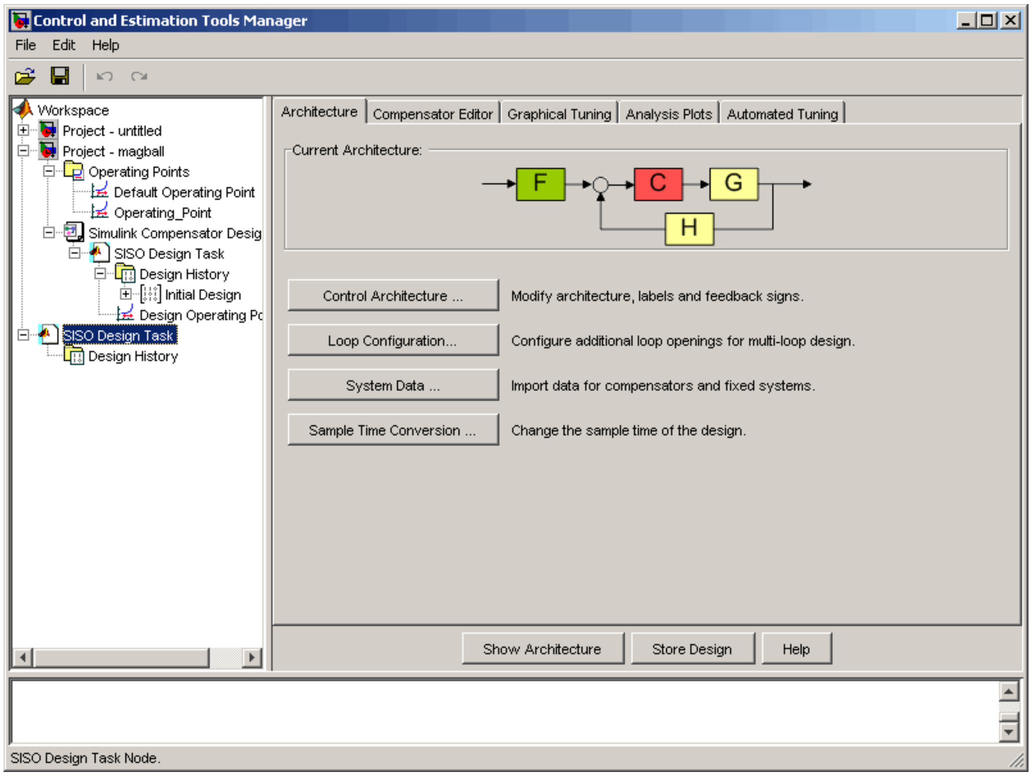
The enhanced SISO Design Task lets you tune compensators using functionality from the Control System Toolbox software and the Simulink Response Optimization software. It includes several tools for tuning the response of SISO systems:

- 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 software)
- A response optimization tool that automatically tunes the system to satisfy design requirements (available when you have the Simulink Response Optimization product)

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

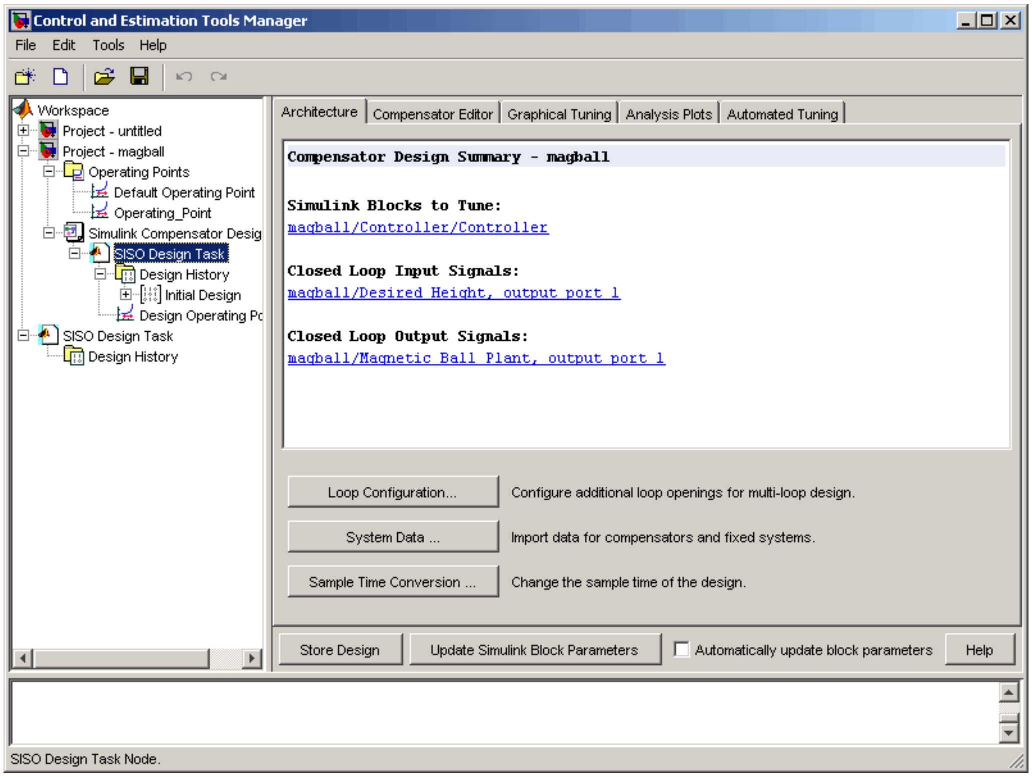
The SISO Design Task is a graphical user interface (GUI) that simplifies the task of designing controllers. This section describes the similarities and differences between the SISO Design Task, which is available in the Control System Toolbox product, and the enhanced SISO Design Task, which is available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

The following figure shows the SISO Design Task as it appears in the Control and Estimation Tools Manager.



Control Architecture ...	Modify architecture, labels and feedback signs.
Loop Configuration...	Configure additional loop openings for multi-loop design.
System Data ...	Import data for compensators and fixed systems.
Sample Time Conversion ...	Change the sample time of the design.

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.



Compensator Design Summary - magball

Simulink Blocks to Tune:
magball/Controller/Controller

Closed Loop Input Signals:
magball/Desired Height, output port 1

Closed Loop Output Signals:
magball/Magnetic Ball Plant, output port 1

Configure additional loop openings for multi-loop design.

Import data for compensators and fixed systems.

Change the sample time of the design.

SISO Design Task Node.

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 Simulink Control Design getting started 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”. • When you are satisfied with your system’s performance, the enhanced SISO Design Tool lets you click **Update Simulink Block Parameters** to write the parameters back to your Simulink model.

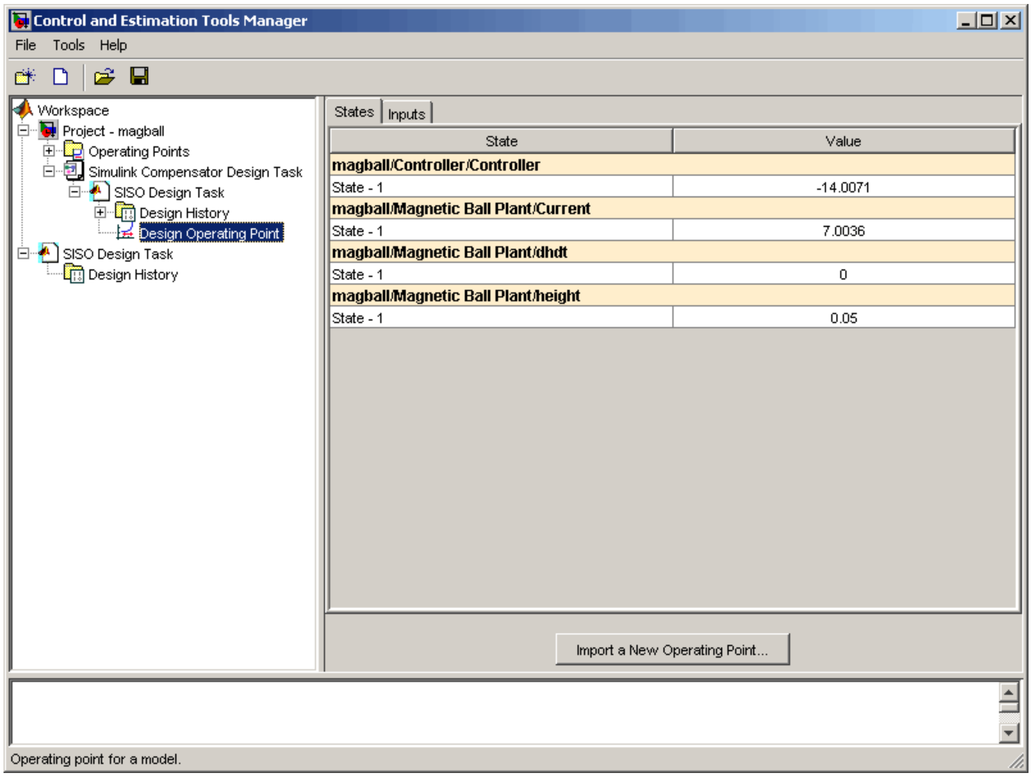
For additional information, see:

- “Creating a SISO Design Task” in the Simulink Control Design getting started documentation
- “Designing Compensators” in the Control System Toolbox getting started documentation

- "SISO Design Tool" in the Control System Toolbox getting started documentation

Design Operating Point Node

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



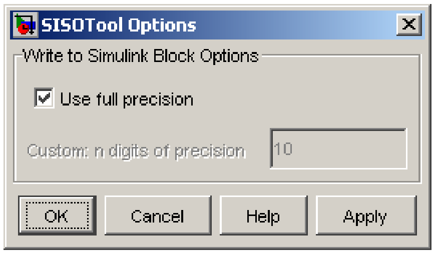
State	Value
magball/Controller/Controller	
State - 1	-14.0071
magball/Magnetic Ball Plant/Current	
State - 1	7.0036
magball/Magnetic Ball Plant/dhdt	
State - 1	0
magball/Magnetic Ball Plant/height	
State - 1	0.05

Operating point for a model.

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.


SISOTool Options

Write to Simulink Block Options

☑ Use full precision

Custom: n digits of precision 10

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 Simulink Control Design getting started documentation.

Specifying Operating Points Using Functions

Overview

This section describes how to specify operating points for a model using functions in the MATLAB command window. Use the functions when you want to create M-files to automate the linearization process, or when you want to use an operating point to initialize a Simulink model. For a description of how to use the graphical interface for this task, see Chapter 2, "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 Simulink Control Design getting started 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.

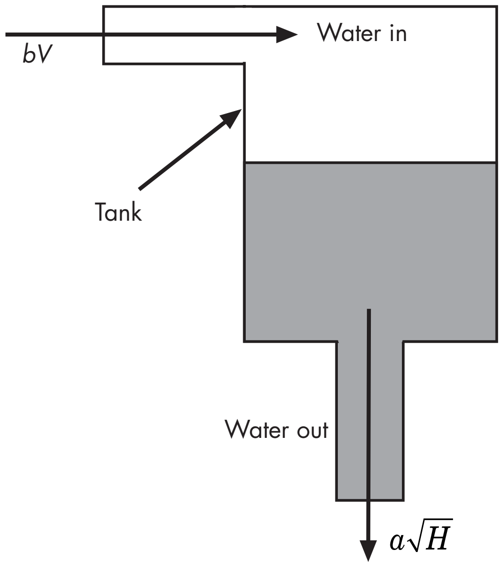
Example: Water-Tank System

In this section...

Water-Tank System

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

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



Schematic Diagram for the Water-Tank System

Model Equations

This section describes the model equations for the example started in the previous section "Example: Water-Tank System" on page 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 simplifies the analysis of this model. For information on the linearization process, see "Linearizing Models" in the Simulink Control Design getting started documentation.

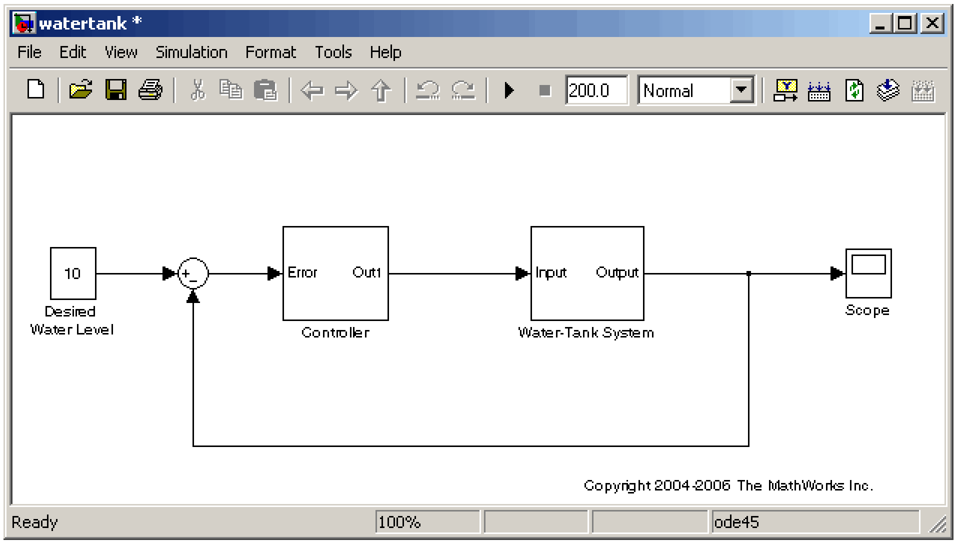
Creating or Opening a Simulink Model

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.

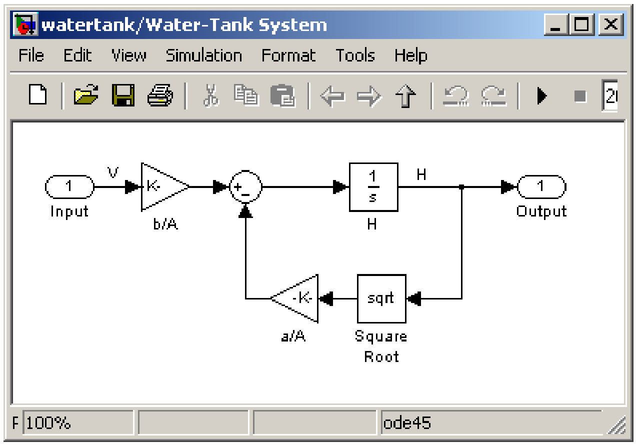


Simulink® Model of the Water-Tank System

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.


watertank/Water-Tank System
File Edit View Simulation Format Tools Help
1
Input
V
K-
b/A
+
-
1/s
H
H
1
Output
-K-
a/A
sqrt
Square Root
100%
ode45

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·V).

Computing Operating Points from Specifications

In this section...

Workflow for 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 Using Functions" 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 software 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 5-11, use the following command:

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

This adds a constraint on the signal between the Sum block and the integrator block, H, within the Water-Tank System block. The constraint is associated with an 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

In this section...

Workflow for Specifying Completely Known Operating Points

To use functions to specify completely known operating points

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

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

Creating an Operating Point Object

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

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

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

MATLAB software displays

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

```
States:
----------
(1.) watertank/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 Using Functions” on page 6-3 for more information on creating linearization point objects.

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

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

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

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

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

States:
----------
(1.) watertank/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.

Linearizing Models Using Functions

Overview

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

1. Creating or opening a Simulink model
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 Versus Command-Line Functions” in the Simulink Control Design getting started documentation. A particular advantage of the linearization functions is the ability to write scripts to automate the linearization process or perform *batch linearization*.

Configuring the Model for Linearization Using Functions

In this section...

Workflow for 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 Simulink Control Design getting started 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 Simulink Control Design getting started 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.

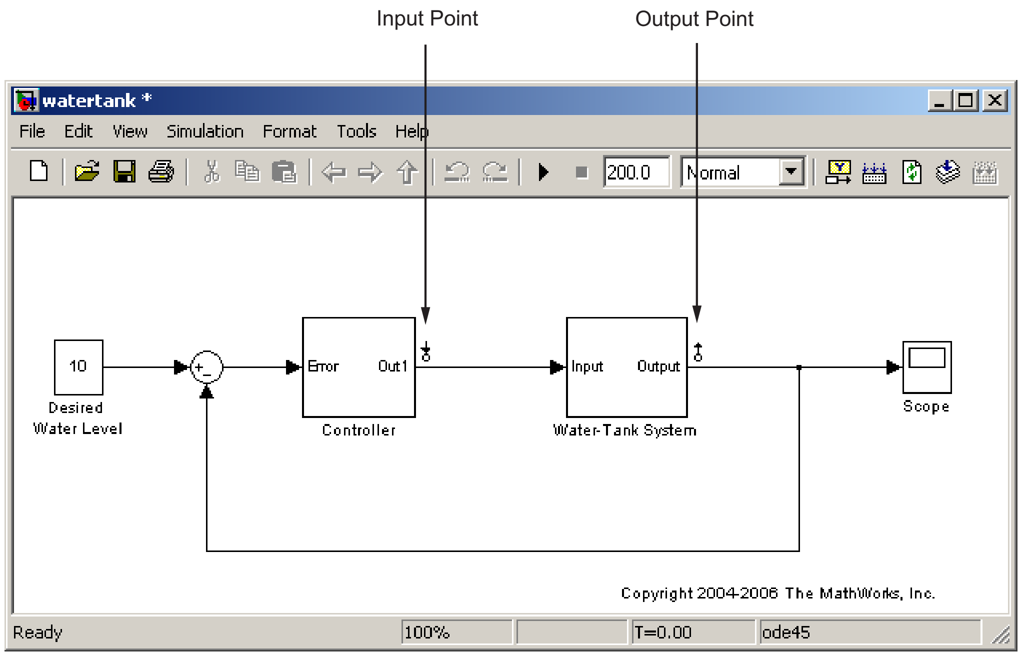
```
Linearization IOs:
--------------------------
Block watertank/Controller, Port 1 is marked with the following
properties:
 - No Loop Opening
 - An Input Perturbation
 - No signal name. Linearization will use the block name

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 when they have been selected using the right-click menus described in “Inserting Linearization Points”. 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: ''
```

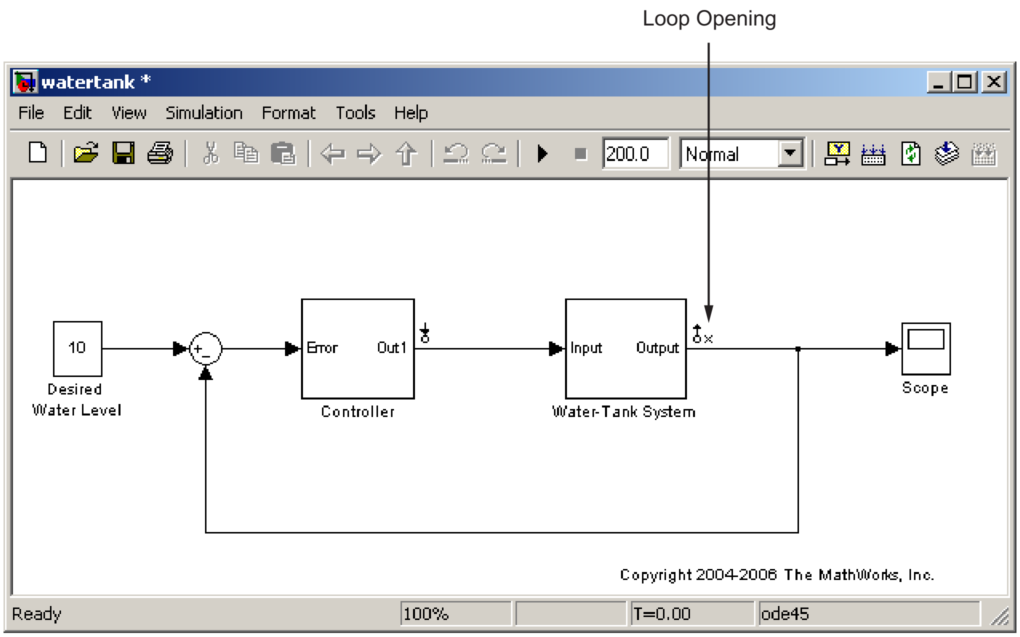
You can edit this object to make any necessary changes. For example, to make this linearization point a loop opening as well, type

```
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 Simulink Control Design getting started 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 6-7.

Linearizing the Model Using Functions

In this section...

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 Simulink Control Design getting started 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 returns the matrices `a`, `b`, `c`, and `d` of a linear, time-invariant, state-space model that approximates your nonlinear system in a region around the operating point.

```
a =
                H
   H  -0.01582


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 the Simulink Control Design software will find the slowest sample rate in the model to use for the sample rate of the linearized model. To create a linearized model with different sample time, specify a new parameter value before linearizing the model. A value of `0` will give a continuous-time model. For more information, see the Simulink Control Design demo “Linearization of Multirate Models”.

Analyzing the Results Using Functions

In this section...

Options for Analyzing the Results

This section describes how to analyze the linearization results using functions. For a description of how to use the graphical interface for this task, see "Viewing Linearization Results" in the Simulink Control Design getting started documentation.

You can analyze the linearized model by

- Using Control System Toolbox functions at the MATLAB prompt.
- Displaying it in the LTI Viewer.
- Incorporating the results into a block in a Simulink model.

  For methods on simulating the linearized model for comparison with the original model, refer to "Validating Linearization Results" on page 7-15

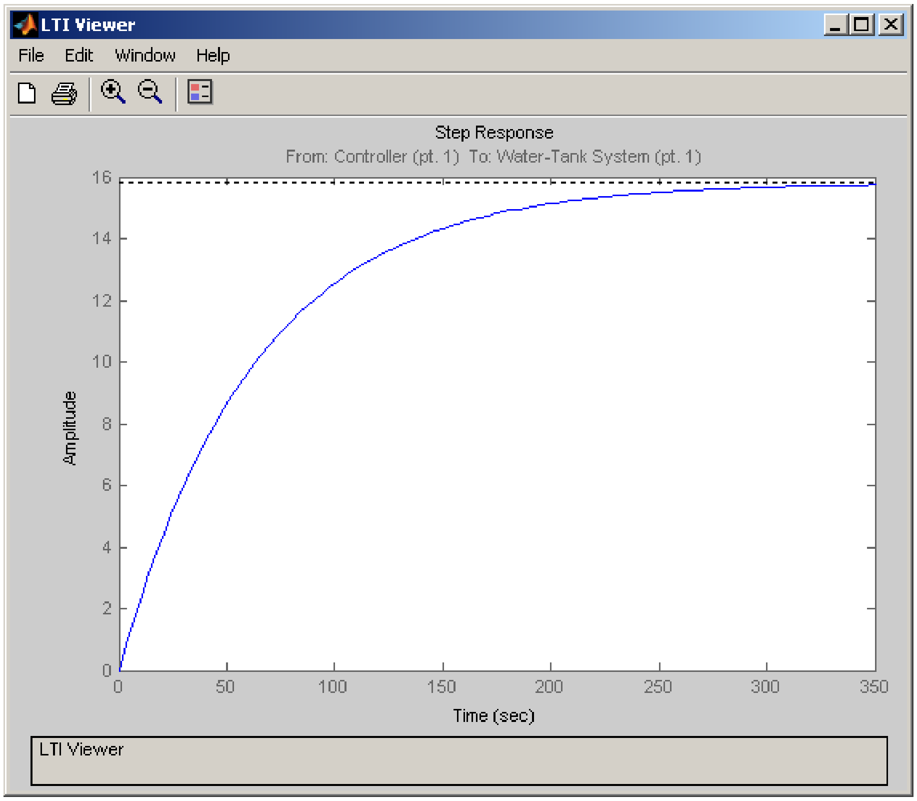
Using the LTI Viewer

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 Using Functions" on page 6-10 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 Using Functions" on page 6-10 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.

Understanding Analysis in the Simulink Control Design Software

Choosing a Linearization Algorithm Method

In this section...

Options for Linearization Algorithm Method

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

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

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

Advantages of Block-by-Block Analytical Linearization

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

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

- You can compute linearized models with exact representations of continuous time delays.

For more information, see .

Advantages and Disadvantages of Numerical-Perturbation Linearization

Numerical-perturbation linearization linearizes the *whole system* by numerically perturbing the system's inputs and states around the operating point. The advantage of this method is that it is quick and simple, especially for large or complicated systems. However, there are also several disadvantages with this method:

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

For more information, see "Numerical-Perturbation Linearization" on page 7-4.

Numerical-Perturbation Linearization

In this section...

What is Numerical-Perturbation Linearization?

An alternative linearization method available for use in the Simulink Control Design software is numerical-perturbation linearization, which computes state-space matrices for the linearized model by numerical perturbation of the *whole system*. The method is relatively quick and simple, although as mentioned in "Choosing a Linearization Algorithm Method" on page 7-2, 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 7-7

Invoking Numerical-Perturbation Linearization

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

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

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

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

To linearize the model, type

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

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

Perturbation Algorithm

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

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

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

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

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

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

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

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

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

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.

- $u_{\mathrm{p,i}}$ 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 $x_{\mathrm{p,i}}$, u_{o}.
- $\dot{x}|_{u_{p,i}}$ is the value of $\dot{x}$ at $u_{\mathrm{p,i}}$, 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 Simulink Control Design getting started 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 Simulink Control Design getting started 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:

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

  When using the Control and Estimation Tools Manager graphical interface, select **Tools > Options** to open the Options dialog, and then select the **Linearization** tab-pane. Within the **Linearization** pane, make sure that you have selected `Numerical perturbation` as the **Linearization algorithm** and then enter a value for **Relative Perturbation level** under **Options for numerical perturbation algorithm**.
- You can provide individual perturbation levels for each state and inport block. These values override the values computed using the `NumericalPertRel value`. Set the perturbation levels using the `linoptions` function to edit the linearization properties `NumericalXPert` and `NumericalUPert`. To specify the absolute perturbation levels for `NumericalXPert` and `NumericalUPert`, you can use the `operpoint` function to create an operating point object and then edit the operating point values using dot-notation or the `set` function.

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

Example: Linearizing a Model Using Numerical-Perturbation at the MATLAB Command Line

The following example illustrates how to linearize a model at the MATLAB command line using numerical perturbation.

1 Open the model.

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

At the MATLAB command line, enter

```
scdairframe_reference
```

to open this model.

2 Set Inport and Outport blocks.

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

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

3 Create an operating point object for the model.

There are several possible methods for creating an operating point object. Which one you use depends on the model you are using and the information you have about the operating point. For more information on creating operating points, see "Specifying Operating Points" in the Simulink Control Design getting started documentation.

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

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

4 Specify the linearization algorithm.

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

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

5 Set the perturbation levels.

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

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

and

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

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

a Create an operating point object, and edit the state values within this object

b Then, assign these values to the `NumericalXPert` option, using the following commands:

```
state_pert=operpoint('scdairframe_reference');
```

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

6 Linearize the model.

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

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

Example: Linearizing a Model Using Numerical-Perturbation in the GUI

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

1 Open the model.

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

At the MATLAB command line, enter

```
scdairframe_reference
```

to open this model.

2 Set Inport and Outport blocks.

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

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

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

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

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

4. Create an operating point object for the model.

   There are several possible methods for creating operating point objects. Which one you use depends on the model you are using and the information you have about the operating point. For more information on creating operating points, see "Specifying Operating Points" in the Simulink Control Design getting started documentation.

   This example uses the default operating point for the linearization.

5. Specify the linearization algorithm.

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

6. Set the perturbation levels.

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

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

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

---

7 Linearize the model:

   a Select **Linearization Task** in the pane on the left of the Control and Estimation Tools Manager.

   b Select the **Operating Points** pane on the right.

   c Within the **Operating Points** pane, select the operating point that you want to use for the linearization. For this example, there should be only one choice, the default operating point.

   d Click the **Linearize Model** button to linearize the model around this operating point. The results are plotted in the LTI Viewer.

Handling Special Blocks

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

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

Handling Feedback Loops

"Performing Open-Loop Analysis" in the Simulink Control Design getting started 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.

Validating Linearization Results

Comparing the Linearized and Original Nonlinear Models

Workflow for Comparing the Linearized and Original Nonlinear Models

To ensure the accuracy of linearized models, you should validate the linearization results. One way to validate the linearization results is to compare a simulation of the linearized and original nonlinear models, both at the linearization operating point, to determine if they behave in a similar way.

To compare the linearized and original nonlinear models, perform the following steps:

1. Insert the linearized subsystem into a copy of the original nonlinear model.

2. Configure the states, inputs, and outputs of the linearized model and original nonlinear model to match the operating point used for linearization.

   For more information, see "Impact of Operating Point on Comparison of Linearized and Original Nonlinear Models" on page 7-16.

3. Compare output signals from a transient simulation of the two models.

   Use a perturbation input, for example a step response, that is small enough to keep the operating point of your models in the range of accuracy of the linear model.

Note If the linear and nonlinear models you are comparing are unstable, add a feedback system with a stabilizing controller to both models before comparing them.

For an example of comparing a linearized model to the original nonlinear model, see "Example of Comparing Unstable Open-Loop Models Using Simulation" on page 7-17.

Impact of Operating Point on Comparison of Linearized and Original Nonlinear Models

When you compare models, you must configure the states, inputs, and outputs of the linearized model and the original nonlinear model to match the operating point used for linearization.

The following state-space equations describe the linearized model:

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

The states, inputs, and outputs of the linearized model are defined about an operating point of the original nonlinear model, using the following variables:

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

When the original nonlinear model is at the linearization operating point $x(t)=x_0$, $u(t)=u_0$, $y(t)=y_0$, the linearized model is at the operating point $\delta x(t)=0$, $\delta u(t)=0$, $\delta y(t)=0$.

To compare the models accurately at the linearization operating point, do the following:

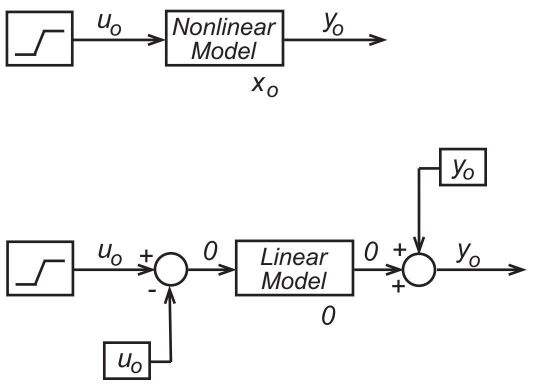
- Initialize the original nonlinear model with the linearization operating point.

- In the linearized model, subtract u_0 from input values and add y_0 to the output signal.

  This manipulation results in an input of zero and an output of y_0 for the linearized system.

Note If you linearize only a portion of the entire model, you also need to initialize the states of the parts of the model that you did not linearize. These states must match the linearization operating point.

The following schematic shows the nonlinear and linear models you compare.



Example of Comparing Unstable Open-Loop Models Using Simulation

In this example, you compare the following models:

- `magball_validate_nonlinear` — the original nonlinear model, which contains a nonlinear plant, named Magnetic Ball Plant, in a single-loop stabilizing control system. This model has been initialized with the operating point used for linearization, `Model_op`.
- `magball_validate_linear` — the linearized model, which is a new version of `magball_validate_nonlinear` that you create in this example. You replace the nonlinear Magnetic Ball Plant with an existing linearized

version of the plant, named `Model_sys`. This plant was linearized about the operating point `Model_op`.

The Magnetic Ball Plant is unstable. Thus, you must compare the entire linearized model against the entire original nonlinear model, both of which include stabilizing control systems.

To compare the linearized and original nonlinear models, complete the following tasks:

Note If you want to skip to step 4 where you simulate the linearized and original nonlinear models, you can open preconfigured models by typing the following at the MATLAB command prompt:

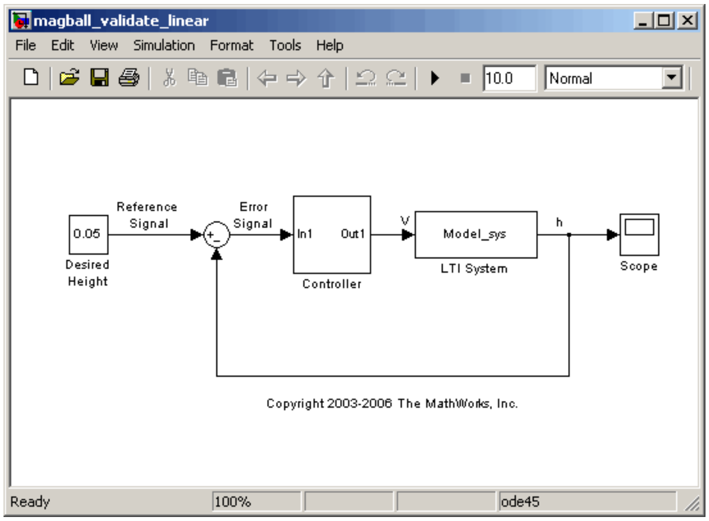
```
magball_validate_nonlinear
magball_validate_linear
```

---

1. Create a new version of the `magball_validate_nonlinear` model with a linearized plant.
   - **a** Open the original nonlinear model by typing the following at the MATLAB command prompt:

     ```
     magball_validate_nonlinear
     ```

   - **b** Open a new Simulink window and paste a copy of the original `magball_validate_nonlinear` model into this window.
   - **c** Delete the Magnetic Ball Plant subsystem in the new model and replace it with an LTI System block (located in the Control System Toolbox category of the Simulink Library Browser).
   - **d** In the **LTI system variable** field in the LTI System block parameter window, enter `Model_sys` to import the linearized model into this block. `Model_sys` is an existing linearized version of the Magnetic Ball Plant.
   - **e** Save the new model as `magball_validate_linear`.



2 Initialize the states in the controller to match the linearization operating point, `Model_op`.

a Create a new operating point object for the system by typing the following:

```
op=operpoint('magball_validate_linear')
```

b Change the states of the operating point for the Controller in `op` to match the states in `Model_op` by typing the following:

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

Note In both operating points, the `States` field of the first element of the operating point vector represents the controller.

---

c Create a Simulink structure from this operating point using the `getstatestruct` function by typing the following:

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

The structure contains the operating point values in a format that Simulink can use to set initial values.

d Use the values in `x_struct` as initial values for `magball_validate_linear`.

   i Select **Simulation > Configuration Parameters** in the `magball_validate_linear` model window.

   ii Click the **Data Import/Export** tab.

   iii Select the check box, and enter `x_struct` in the **Initial State** field.

   iv Click **OK**.

Note You keep the operating point state values in the linearized model at zero because this subsystem was linearized about the existing operating point values.

Note Typically, you also initialize the nonlinear model with the linearization operating point. However, for this example, the `magball_validate_nonlinear` model is already initialized with the linearization operating point `Model_op`.

---

3 Configure the input and output of the new model `magball_validate_linear` to ensure that the entire model is at the linearization operating point.

Note You use the operating point values of 14 and 0.05 for the input and output, respectively. These values can be found using a Scope block to measure steady-state signal levels in the original nonlinear model.

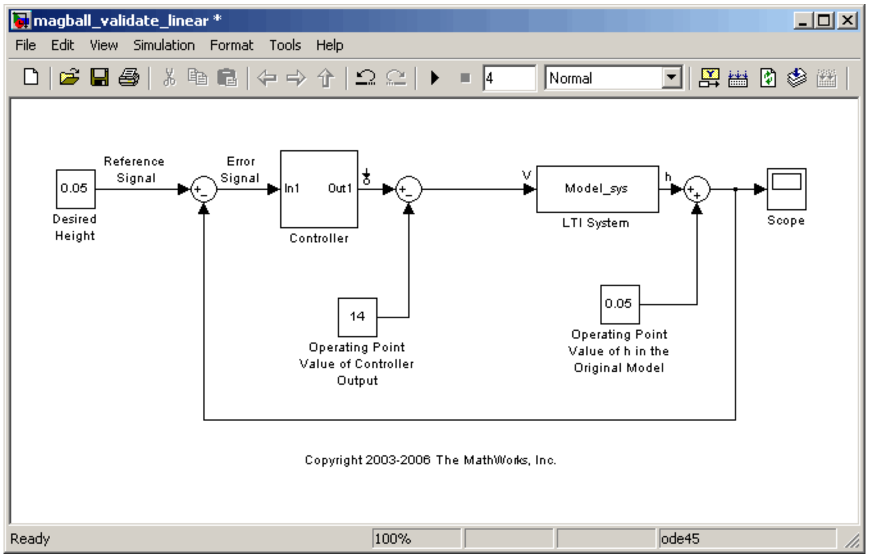
---

a Add a `Constant` block with a value of `14` and a Sum block to subtract this value from the input signal of the linearized system.

   Subtracting the input value from the original nonlinear model ensures that the input to the linearized portion of the model is zero.

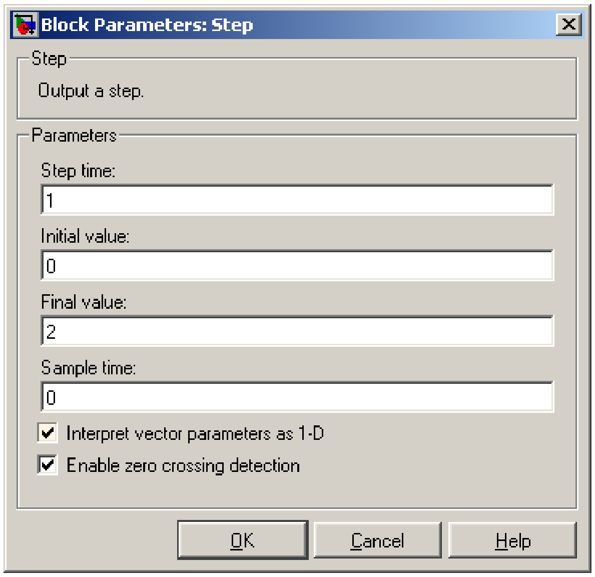
b Add a Constant block with a value of `0.05` and a Sum block to add this output value to the output of the linearized model, which is zero.

After configuring the inputs and outputs, the model should resemble the following figure.


magball_validate_linear *
File Edit View Simulation Format Tools Help
4
Normal
0.05
Desired Height
Reference Signal
Error Signal
In1
Out1
Controller
V
Model_sys
LTI System
h
Scope
14
Operating Point Value of Controller Output
0.05
Operating Point Value of h in the Original Model
Copyright 2003-2006 The MathWorks, Inc.
Ready
100%
ode45

4 Compare transient simulations of the two models.

a Add a Step block with the parameter values shown in the following figure to the input of the plant in both the original and new models.


Block Parameters: Step

Step

Output a step.

Parameters

Step time:
1

Initial value:
0

Final value:
2

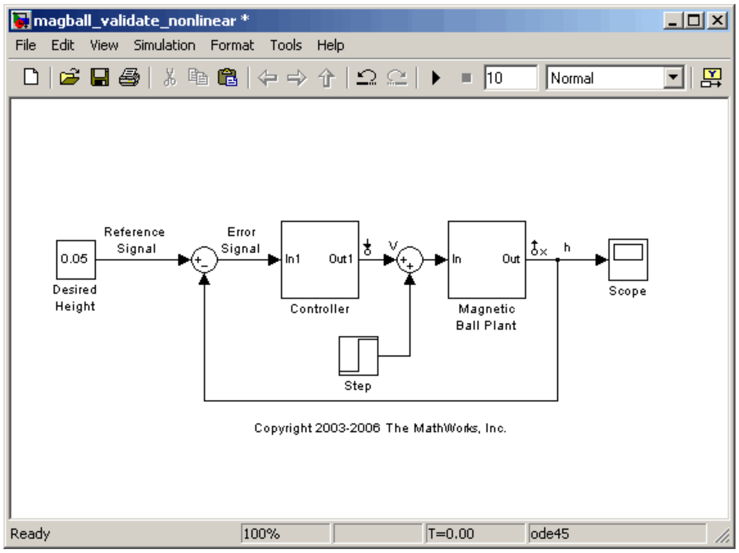
Sample time:
0

✓ Interpret vector parameters as 1-D

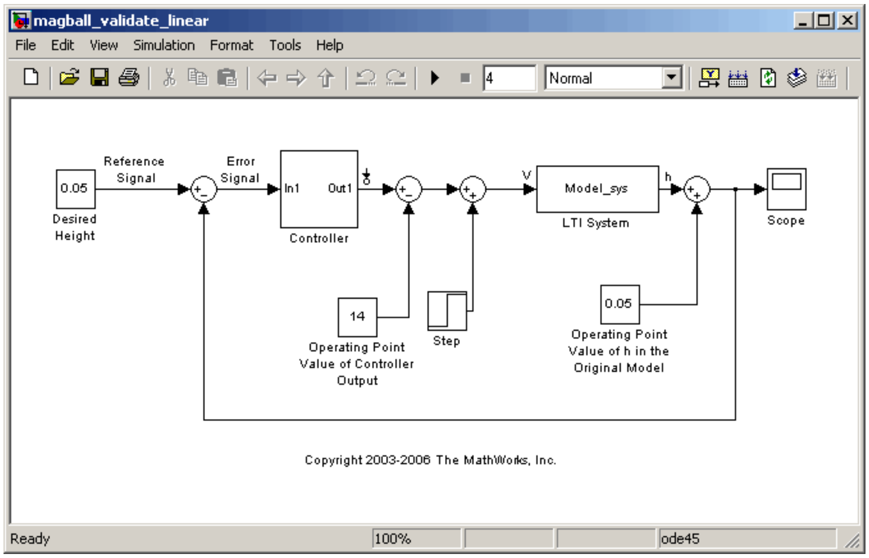
✓ Enable zero crossing detection

Parameter Values for Step Block

The model diagrams now resemble the following figures.



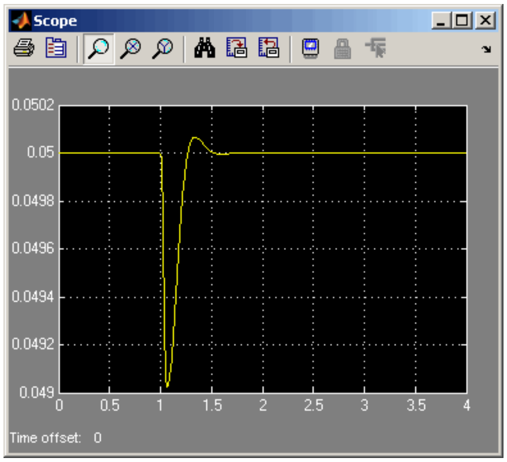
Nonlinear Magball Model with a Step Input


magball_validate_linear
File Edit View Simulation Format Tools Help
4
Normal
0.05
Desired Height
Reference Signal
Error Signal
In1
Out1
Controller
u
V
Model_sys
LTI System
h
Scope
14
Operating Point Value of Controller Output
Step
0.05
Operating Point Value of h in the Original Model
Copyright 2003-2006 The MathWorks, Inc.
Ready
100%
ode45

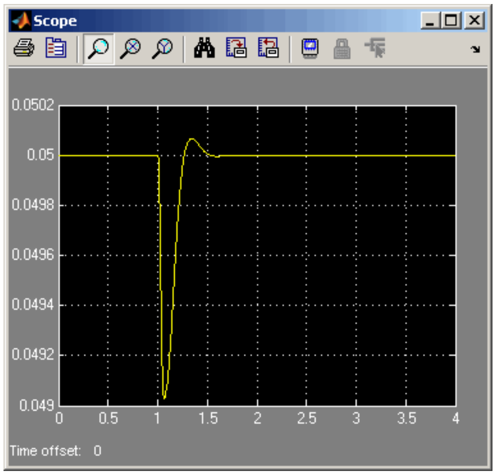
Linear Magball Model with a Step Input

The addition of the step input allows you to observe the response of the models to a perturbation. This step input is small enough to keep the operating point of the models within the range of accuracy of the linear model.

b Run a simulation for each model in their respective Simulink windows. The following figures show the Scope block output signals from the original and linearized models.



Original Nonlinear Model



Linearized Model

The linearized and original nonlinear models react to the step input in a similar way. This shows that the linearized model provides a good approximation of the nonlinear model.

Troubleshooting Linearization Results

In this section...

Diagnosing Blocks

Blocks in Your Model That Impact Linearization Results

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

You can locate the blocks that impact your linearization results by highlighting the blocks in the linearization path. For instructions about highlighting blocks in the linearization path, see "Highlighting Blocks in the Linearization".

Note If one of the blocks in the linearization path of your model does not highlight, it might have a linearization result of zero. You can use the Simulink Control Design diagnostic messages to determine if other blocks in your model are causing this block to linearize to zero. For information on how to view diagnostic messages, see "Using Diagnostics Messages to Find Problematic Blocks in Your Model" on page 7-28.

Which Blocks Linearize Correctly?

Nearly all core Simulink blocks give accurate linearization results. The exceptions are the blocks that are not supported for linearization. See "Unsupported Blocks for Linearization" on page 7-27 for information on how to locate these blocks.

In some cases, the accurate linearization results you obtain might not be the result you expect. If you encounter unexpected linearization results, you can use the Simulink Control Design diagnostic messages to identify which blocks are causing problems in your linearization. For information on how to view diagnostic messages, see "Using Diagnostics Messages to Find Problematic Blocks in Your Model" on page 7-28.

Note Blocks with discontinuities, such as relay and Boolean logic, linearize to zero or infinity. These blocks do not display in the diagnostic messages. If your model contains these blocks, verify that they linearize in the way that you expect. For information on how to find these blocks in your model, see "Troubleshooting Your Model at the Subsystem and Block Level" on page 7-29 and "The Model Explorer". For more information on blocks with discontinuities, see .

Blocks with Configuration Warnings

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

To locate blocks in your model with configuration warnings and to read the warning messages, view the list of block with warning in the Simulink Control Design **Diagnostics Messages** tab. For information on how to view diagnostic messages, see "Using Diagnostics Messages to Find Problematic Blocks in Your Model" on page 7-28.

Unsupported Blocks for Linearization

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

You can find a list of unsupported blocks for linearization in the Simulink Control Design **Diagnostics Messages** tab. For information about viewing diagnostic messages, see "Using Diagnostics Messages to Find Problematic Blocks in Your Model" on page 7-28.

Blocks That Automatically Linearize Using Numerical Perturbation

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

- Blocks that are located near discontinuous regions

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

- Blocks that have nondouble inputs and states

  These blocks linearize to zero.

To locate blocks in your model that automatically linearize using numerical perturbation, click the hyperlink in the **blocks without pre-programmed exact Jacobian (linearized using numerical perturbation)** section of the Simulink Control Design **Diagnostics Messages** tab. For information about viewing diagnostic messages, see "Using Diagnostics Messages to Find Problematic Blocks in Your Model" on page 7-28.

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

For more information about numerical perturbation linearization and setting numerical perturbation levels, see "Numerical-Perturbation Linearization" on page 7-4.

Using Diagnostics Messages to Find Problematic Blocks in Your Model

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

- "Unsupported Blocks for Linearization" on page 7-27
- "Blocks with Configuration Warnings" on page 7-26
- "Blocks That Automatically Linearize Using Numerical Perturbation" on page 7-27

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

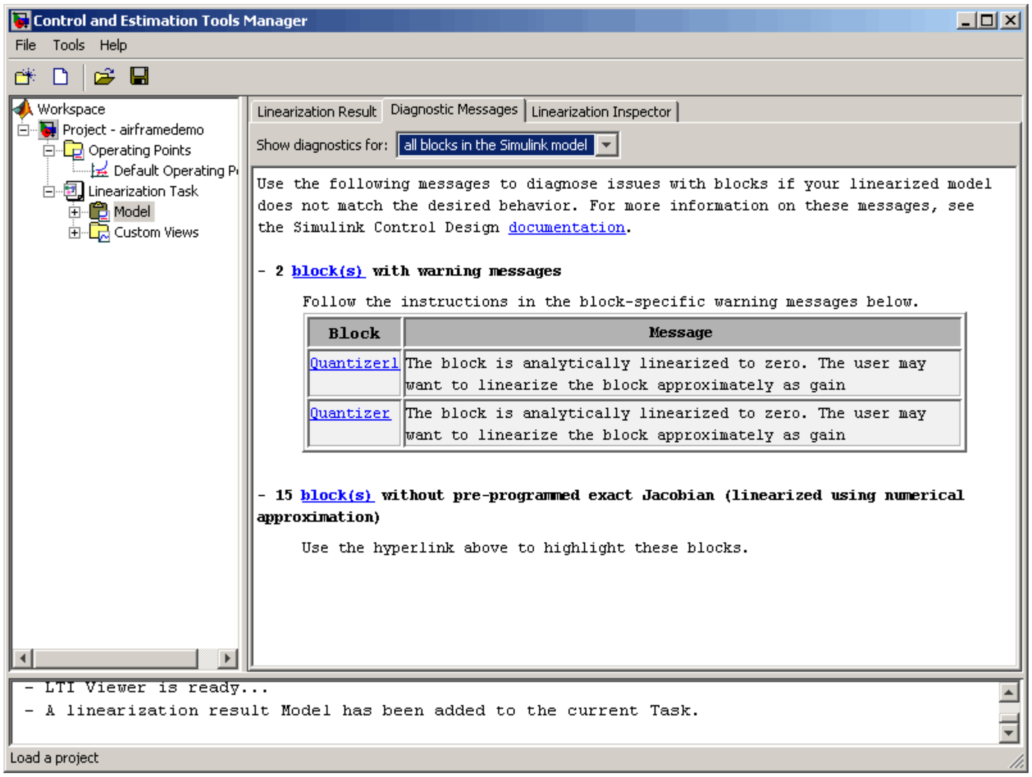
1 Select the **Model** node in the project tree for your linearized model.

2 Select the **Linearization Diagnostic Messages** tab.

3 In the **Show diagnostics for** drop-down list, choose one of the following options:

   - **Blocks in the linearization path**
   - **All blocks in the Simulink model**

Your result resembles the following figure.



Show diagnostics for: all blocks in the Simulink model

Use the following messages to diagnose issues with blocks if your linearized model does not match the desired behavior. For more information on these messages, see the Simulink Control Design documentation.

- 2 block(s) with warning messages

Follow the instructions in the block-specific warning messages below.

Block	Message
Quantizer1	The block is analytically linearized to zero. The user may want to linearize the block approximately as gain
Quantizer	The block is analytically linearized to zero. The user may want to linearize the block approximately as gain

- 15 block(s) without pre-programmed exact Jacobian (linearized using numerical approximation)

Use the hyperlink above to highlight these blocks.

```
- LTI Viewer is ready...
- A linearization result Model has been added to the current Task.
```

Load a project

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

Troubleshooting Your Model at the Subsystem and Block Level

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

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

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

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

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

  For instructions on using the linearization inspector, see "Inspecting the Linearization Results Block by Block".
- Linearize individual subsystems and blocks in your model.

  This approach is an effective way to debug the linearization of a large model because it allows you to quickly narrow down the problem. For instructions on linearizing subsystems and blocks, see "Linearizing a Block" on page 3-16.

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

Troubleshooting Linearization Settings

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

- Rate Conversion Method

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

- Return model with exact delay

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

For more information about these settings, see “Changing Linearization Settings and Algorithms” on page 3-12 in the Simulink Control Design documentation and the `linoptions` reference page.

Troubleshooting Models with Events-Based Subsystems

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

For more information about linearizing event-based subsystems, see .

Troubleshooting Your Operating Point

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

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

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

- How the operating point you choose affects your linearization results — See the “Why Are Operating Points Important?” section of the Simulink Control Design getting started documentation.
- How to creating accurate operating points — See the “Recommendations for Computing Operating Points” on page 7-32 section of the *Simulink Control Design User’s Guide.*

Recommendations for Computing Operating Points

In this section...

How to Create Accurate Operating Points

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

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

Impact of Blocks on the Simulink Model Operating Point

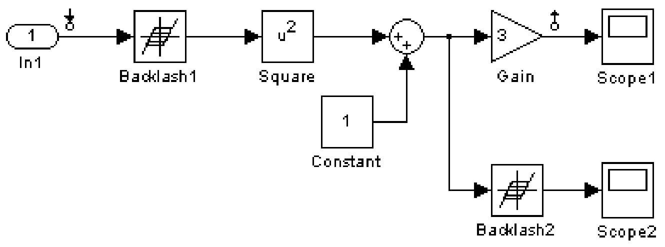
The full operating point in a Simulink model is specified in a number of ways by the blocks in the model:

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

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

Example of the Impact of Blocks with Internal States

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



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

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

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

When you linearize this model, the linearization is performed around the full model operating point, which includes the two states. For the input and output points specified in this model, the second backlash block is not in the linearization path and thus its state does not impact the linearization result.

Types of Blocks with Internal States

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

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

Finding Blocks with Internal States in Your Model

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

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

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

Working with Models Containing Blocks with Internal States

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

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

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

- "Memory Blocks" on page 7-35
- "Transport Delay and Variable Transport Delay Blocks" on page 7-37
- "Backlash Block" on page 7-37

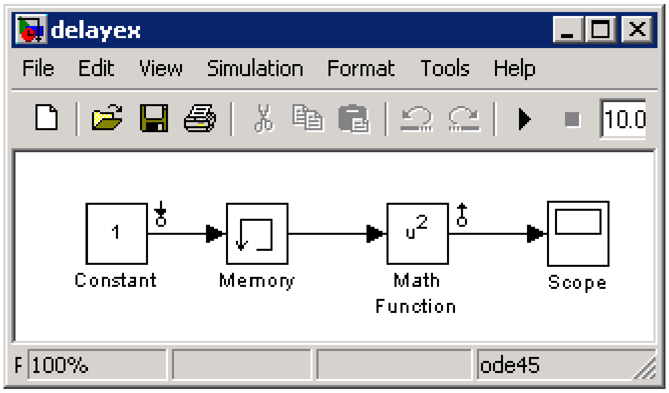
For other blocks with internal states, you should consider their impact on the analysis tools in the Simulink Control Design software in the following ways:

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

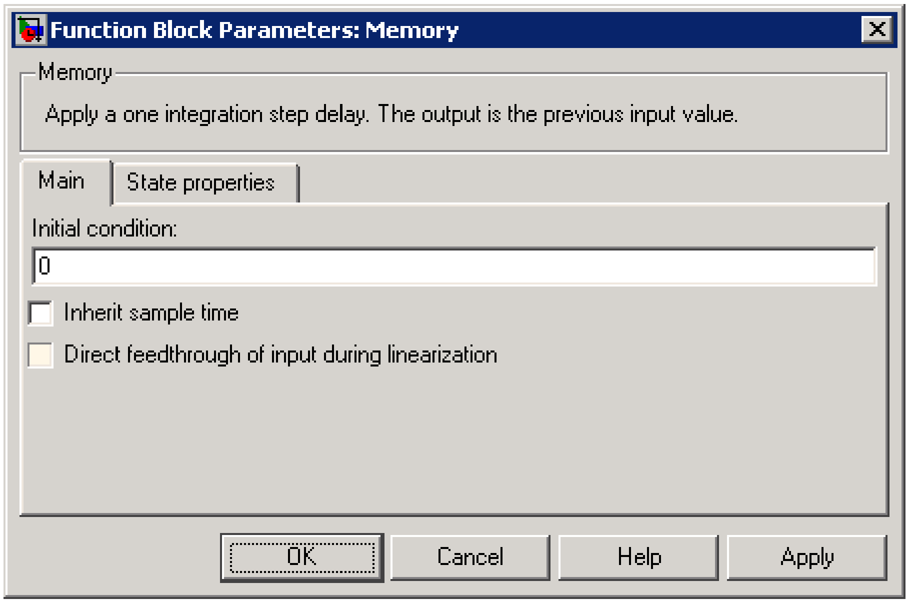
If the block does have impact, consider replacing it using a configurable subsystem when searching for an operating point and linearizing.

In many cases, performing a linearization using linearization snapshots avoids the challenges associated with blocks with internal states. You can linearize your model at steady state using linearization snapshots as described in "Linearizing at Specified Simulation Times" on page 3-8 and "Linearizing at Simulation Events" on page 3-10.

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



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


Function Block Parameters: Memory

Memory

Apply a one integration step delay. The output is the previous input value.

Initial condition:

0

☐ Inherit sample time

☐ Direct feedthrough of input during linearization

Transport Delay and Variable Transport Delay Blocks. When you have Transport Delay or Variable Transport Delay blocks in your model, you can properly configure the initial outputs of these blocks so that operating point searches or linearization uses the correct output value at steady state condition. The discussion in "Memory Blocks" on page 7-35 applies to configuring the initial outputs of the Transport Delay and Variable Transport Delay blocks.

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

Computing Operating Points for SimMechanics Models

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

1. Locate and open the machine environment (Env) block for the system.

2. From the **Parameters** pane, set **Analysis mode** to `Trimming`. Click **OK** to close the block dialog box. This will create an output port in the model that contains constraints related to errors in the system that must be set to zero for a steady state operating point.

3. To set these constraints to zero within a project for the model in the Control and Estimation Tools Manager, select **Operating Points** in the pane on the left, and then select **Compute Operating Points > Outputs**. Within this pane, set all constraints to `0`.

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

Choosing Initial Values for Computing Operating Points

When you compute an operating point from design specifications (trimming), it is often important to begin with a set of state and input values that are close to the actual steady state operating point values that you are trying to compute. To do this you can simulate the model for a specified period of time and then take a *snapshot* of the state and input values at that time. You can do this using either the Control and Estimation Tools Manager (see "Creating Operating Points from Simulation" in the Simulink Control Design getting started 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 2-6.

Function Reference

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

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 2-11.

Description

opnew=addoutputspec(op,'block',portnumber) adds an output specification for a Simulink model to an existing operating point specification, op, created with operspec. The signal being constrained by the output specification is indicated by the name of the block, 'block', and the port number, portnumber, that it originates from.

You can edit the output specification within the new operating point specification object, opnew, to include the actual constraints or specifications for the signal. Use the new operating point specification object with the function findop to find operating points for the model.

This function automatically compiles the Simulink model, given in the property Model of op, to find the block’s output portwidth.

Example

Create an operating point specification for the model magball.

```
op=operspec('magball')
```

This specification returns the object op. Note that there are no 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)
```

This function adds the output specification is added to the operating point specification object.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:             0
      spec:  dx = 0,  initial guess:             0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:             7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:             0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:          0.05

Inputs: None

Outputs:
-----------
```

```
(1.) magball/Controller
      spec:  none
```

Edit the output specification to constrain this signal to be 14.

```
newop.Outputs(1).Known=1, newop.Outputs(1).y=14
```

The final output specification is displayed.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:             0
      spec:  dx = 0,  initial guess:             0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:             7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:             0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:          0.05

Inputs: None

Outputs:
-----------
(1.) magball/Controller
      spec:  y = 14
```

See Also

findop, operspec, operpoint

Purpose

Copy operating point or operating point specification

Syntax

```
op_point2=copy(op_point1)
op_spec2=copy(op_spec1)
```

Description

`op_point2=copy(op_point1)` returns a copy of the operating point object `op_point1`. You can create `op_point1` with the function `operpoint`.

`op_spec2=copy(op_spec1)` returns a copy of the operating point specification object `op_spec1`. You can create `op_spec1` with the function `operspec`.

Note The command `op_point2=op_point1` does not create a copy of `op_point1` but instead creates a pointer to `op_point1`. In this case, any changes made to `op_point2` are also made to `op_point1`.

Examples

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

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

The operating point is displayed.

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
```

```
      x: 0.05

Inputs: None
```

Create a copy of this object, `opp`.

```
new_opp=copy(opp)
```

An exact copy of the object is displayed.

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

See Also

`operpoint`, `operspec`

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 Simulink Control Design getting started 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 you work 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.

Note For all syntaxes, `findop` automatically uses the following properties in the Simulink model:

- `BufferReuse = 'off'`
- `RTWInlineParameters = 'on'`
- `BlockReductionOpt = 'off'`

Simulink restores the original property values after finding the operating point.

The output of `findop` is always an operating point object. Use this object with the function `linearize` to create linearized models of Simulink models. The operating point object has the following properties:

- "Model" on page 9-9
- "States" on page 9-9
- "Inputs" on page 9-9
- "Time" on page 9-10

Model

`Model` specifies the name of the Simulink model to which this operating point object refers.

States

`States` describes the operating points of states in the Simulink model. The `States` property is a vector of state objects that contains the operating point values of the states. There is one state object per block that has a state in the Simulink model. The `States` object has the following properties:

`Nx`	Number of states in the block. This property is read-only.
`Block`	Block with which the states are associated.
`x`	Vector containing the values of states in the block.
`Ts`	Vector containing the sample time and offset for the state.
`SampleType`	Set this value to `CSTATE`, for a continuous state, or `DSTATE`, for a discrete state.
`inReferencedModel`	Set this value to `1`, when the state is inside a referenced model, or `0`, when it is not.
`Description`	Text string describing the block.

Inputs

`Inputs` is a vector of input objects that contains the input levels at the operating point. There is one input object per root-level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	Inport block with which the input vector is associated
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	Text string describing the input

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

The operating point report object, returned when finding operating points from specifications, has the following properties:

- `Model`
- `Inputs`
- `Outputs`
- `States`
- `Time`
- `TerminationString`
- `OptimizationOutput`

Of these properties, `Model`, `Inputs`, `Outputs`, `States`, and `Time` contain the same information as the operating point specification object, with the addition of `dx` values for the `States` and `yspec` values, or desired `y` values, for the `Outputs`. The `TerminationString` contains the message that `findop` displays after terminating the optimization. The `OptimizationOutput` property contains the same properties returned in the output variable of the Optimization Toolbox functions `fmincon`, `fminsearch`, and `lsqnonlin`. See the Optimization Toolbox documentation for more information. If you do not have Optimization Toolbox software, you can access the documentation at:

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

Examples

Example 1

Create an operating point specification object for the model `magball` with the `operspec` function.

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

Edit the operating point specification object to reflect any specifications on the operating points such as minimum and maximum values, initial guesses, and known values. This example uses the default specifications in which `SteadyState` is set to `1` for all states, specifying that an equilibrium operating point is desired.

Find the equilibrium operating points with the `findop` function.

```
op_point=findop('magball',op_spec)
```

This function returns an operating point object, `op_point`.

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

```
Inputs: None
```

The MATLAB window displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the operating point values of the states. In this example, there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` field although there are no inputs in this model. To view the properties of `op_point` in more detail, use the `get` function.

MATLAB also displays the operating point report object.

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

Operating condition specifications were 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 preceding report, the `dx` values are all small and close to the desired `dx` values of `0` indicating that an equilibrium or steady-state value was found.

Example 2

To extract an operating point from a simulation at the times 10 and 20 using `findop`, enter the following:

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

This function returns the message:

```
There is more than one operating point.  Select an element
in the vector of operating points to display.
```

To display the first operating point, enter the command

```
op_point(1)
```

This command should display:

```
 Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=10)

States:
----------
(1.) magball/Controller/Controller
      x: -4.82e-010
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 2.6e-006
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

To display the second operating point, enter:

```
op_point(2)
```

This function returns:

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

States:
----------
(1.) magball/Controller/Controller
      x: -5.5e-010
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 2.54e-006
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

See Also

`operspec`, `linearize`

Purpose

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 Simulink Control Design getting started 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
```

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

```
W=op.States
```

This notation returns a vector of objects containing information about the states in the operating point.

```
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

Use `get` to view details of `W`. For example:

```
get(W(2),'x')
```

returns

```
ans =
    7.0036
```

See Also `findop`, `getlinio`, `linio`, `operpoint`, `operspec`, `set`

getinputstruct

Purpose

Input structure from operating point

Syntax

```
in_struct = getinputstruct(op_point)
```

Description

`in_struct = getinputstruct(op_point)` extracts a structure of input values, `in_struct`, from the operating point object, `op_point`. The structure, `in_struct`, uses the same format as Simulink software which allows you to set initial values for inputs in the model within the **Data Import/Export** pane of the Configuration Parameters dialog box.

Example

Create an operating point object for the `f14` model:

```
op_f14=operpoint('f14');
```

Extract an input structure from the operating point object:

```
inputs_f14=getinputstruct(op_f14)
```

This extraction returns

```
inputs_f14 =

       time: 0
    signals: [1x1 struct]
```

To view the values of the inputs within this structure, use dot-notation to access the `values` field:

```
inputs_f14.signals.values
```

In this case, the value of the input is `0`.

See Also

`getstatestruct`, `getxu`, `operpoint`

Purpose

Linearization I/O settings for Simulink model

Syntax

```
io = getlinio('sys')
```

Graphical Interface

As an alternative to the `getlinio` function, view linearization I/Os in the **Analysis I/Os** pane of the **Linearization Task** node within the Simulink Control Design GUI. See “Inspecting Analysis I/Os”.

Description

`io = getlinio('sys')` finds all linearization annotations in the Simulink model, `sys`, and returns a vector of objects, `io`. Each object represents a linearization annotation in the model and is associated with an output port of a Simulink block. Before running `getlinio`, use the right-click menu to insert the linearization annotations, or I/Os, on the signal lines of the model diagram.

Each object within the vector, `io`, has the following properties:

`Active`	Set this value to `'on'`, when the I/O is used for linearization, and `'off'` otherwise
`Block`	Name of the block the with which I/O is associated
`OpenLoop`	Set this value to `'on'`, when the feedback loop at the I/O is open, and `'off'`, when it is closed
`PortNumber`	Integer referring to the output port with which the I/O is associated

`Type`	Choose one of the following linearization I/O types: • `'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 syntax 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)
- 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, you find the operating point from simulation of the model.

```
op=findop('magball',20);
```

Next, compute the open-loop model seen by the block `magball/Controller`, with the `getlinplant` function.

```
[sysp,sysc]=getlinplant('magball/Controller',op)
```

The output variable `sysp` gives the open-loop plant model as follows:

```
a =
                   magball/Magn  magball/Magn  magball/Magn
    magball/Magn           -100             0             0
    magball/Magn         -2.798             0         195.7
    magball/Magn              0             1             0

b =
                   magball/Cont
    magball/Magn             50
    magball/Magn              0
```

```
   magball/Magn             0

c =
                  magball/Magn  magball/Magn  magball/Magn
   Controller (             0             0            -1

d =
                  magball/Cont
   Controller (             0

Continuous-time model.
```

See Also `findop`, `linoptions`, `operpoint`, `operspec`

Purpose

State structure from operating point

Syntax

```
x_struct = getstatestruct(op_point)
```

Description

`x_struct = getstatestruct(op_point)` extracts a structure of state values, `x_struct`, from the operating point object, `op_point`. The structure, `x_struct`, uses the same format as Simulink software which allows you to set initial values for states in the model within the **Data Import/Export** pane of the Configuration Parameters dialog box.

Example

Create an operating point object for the `magball` model:

```
op_magball=operpoint('magball');
```

Extract a state structure from the operating point object:

```
states_magball=getstatestruct(op_magball)
```

This extraction returns

```
states_magball = 

       time: 0
    signals: [1x4 struct]
```

To view the values of the states within this structure, use dot-notation to access the `values` field:

```
states_magball.signals.values
```

This dot-notation returns

```
ans =

     0
     0
```

```
ans =

    7.0036


ans =

     0


ans =

    0.0500
```

See Also `getinputstruct`, `getxu`, `operpoint`

getxu

Purpose

States and inputs from operating points

Syntax

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

Description

`x = getxu(op_point)` extracts a vector of state values, `x`, from the operating point object, `op_point`. The ordering of states in `x` is the same as that used by Simulink software.

`[x,u] = getxu(op_point)` extracts a vector of state values, `x`, and a vector of input values, `u`, from the operating point object, `op_point`. States in `x` and inputs in `u` are ordered in the same way as for Simulink.

`[x,u,xstruct] = getxu(op_point)` extracts a vector of state values, `x`, a vector of input values, `u`, and a structure of state values, `xstruct`, from the operating point object, `op_point`. The structure of state values, `xstruct`, has the same format as that returned from a Simulink simulation. States in `x` and `xstruct` and inputs in `u` are ordered in the same way as for Simulink.

Example

Create an operating point object for the `magball` model by typing:

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

To view the states within this operating point, type:

```
op.States
```

which returns

```
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
```

```
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
```

To extract a vector of state values, with the states in an ordering that is compatible with Simulink, along with inputs and a state structure, type:

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

This syntax returns:

```
x =
    0.0500
         0
         0
    7.0036
         0

u =
     []

xstruct =
       time: 0
    signals: [1x4 struct]
```

View `xstruct` in more detail by typing:

```
xstruct.signals
```

This syntax displays:

```
1x4 struct array with fields:
    values
    dimensions
    label
    blockname
```

View each component of the structure individually. For example:

```
xstruct.signals(1).values

ans =

     0
     0
```

or

```
xstruct.signals(2).values

ans =

    7.0036
```

You can import these vectors and structures into Simulink as initial conditions or input vectors or use them with `setxu`, to set state and input values in another operating point.

See Also `operpoint`, `operspec`

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 Simulink Control Design getting started 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, you can save `xstruct` 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 syntax returns the following operating point:

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

States:
----------
(1.) magball/Controller/Controller
      x: 5.28e-009
      x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
      x: 6.99
(3.) magball/Magnetic Ball Plant/dhdt
      x: -2.62e-005
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

Use these operating point values as initial values in an operating point specification object.

```
opspec=operspec('magball');
newopspec=initopspec(opspec,oppoint)
```

The new operating point specification object is displayed.

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:     5.28e-009
      spec:  dx = 0,  initial guess:    -2.56e-006
```

```
(1.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:          6.99
(1.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:    -2.62e-005
(1.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:          0.05

Inputs: None

Outputs: None
```

You can now use this object to find operating points by optimization.

See Also

`findop`, `getstatestruct`, `getxu`, `operpoint`, `operspec`

linearize

Purpose

Create linearized model from Simulink model

Syntax

```
lin=linearize('sys',io)
lin=linearize('sys',op,io)
lin=linearize('sys',op,io,options)
lin_block=linearize('sys',op,'blockname')
lin=linearize('sys',op)
lin=linearize('sys',op,options)
[lin,op] = linearize('sys',snapshottimes);
lin = linearize('sys','StateOrder',stateorder)
```

Graphical Alternative

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

Description

`lin=linearize('sys',io)` takes a model name, `'sys'`, and an I/O object, `io`, as inputs and returns a linear time-invariant state-space model, `lin`. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. `io` must be associated with the Simulink model, `sys`.

`lin=linearize('sys',op,io)` takes a model name, `'sys'`, an operating point object, `op`, and an I/O object, `io`, as inputs and returns a linear time-invariant state-space model, `lin`. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. Both `op` and `io` must be associated with the same Simulink model, `sys`.

`lin=linearize('sys',op,io,options)` takes a model name, `'sys'`, an operating point object, `op`, an I/O object, `io`, and a linearization options object, `options`, as inputs. It returns a linear time-invariant state-space model, `lin`. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. Both `op` and `io` must be associated with the same Simulink model, `sys`. The linearization options object is created with the function `linoptions` and contains several options for linearization.

`lin_block=linearize('sys',op,'blockname')` takes a model name, `'sys'`, an operating point object, `op`, and the name of a block in the model, `'blockname'`, as inputs and returns `lin_block`, a linear time-invariant state-space model of the named block. The operating point object is created with the function `operpoint` or `findop`. Both `op` and `'blockname'` must be associated with the same Simulink model, `sys`. You can also supply a fourth argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

`lin=linearize('sys',op)` creates a linearized model, `lin`, of the system `'sys'` at the operating point, `op`. Root-level inport and outport blocks in `sys` are used as inputs and outputs for linearization. The operating point object, `op`, is created with the function `operpoint` or `findop`. You can also supply a third argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

`lin=linearize('sys',op,options)` is the form of the `linearize` function that is used with numerical-perturbation linearization. The function returns a linear time-invariant state-space model, `lin`, of the entire model, `sys`. The operating point object, `op`, is created with the function `operpoint` or `findop`. The `LinearizationAlgorithm` option must be set to `'numericalpert'` within `options` for numerical-perturbation linearization to be used. Create the variable `options` with the `linoptions` function. The function uses inport and outport blocks in the model as inputs and outputs for linearization.

`[lin,op] = linearize('sys',snapshottimes);` creates operating points for the linearization by simulating the model, `'sys'`, and taking snapshots of the system's states and inputs at the times given in the vector `snapshottimes`. The function returns `lin`, a set of linear time-invariant state-space models evaluated and `op`, the set of operating point objects used in the linearization. You can specify input and output points for linearization by providing an additional argument such as a linearization I/O object created with `getlinio` or `linio`, or a block name. If an I/O object or block name is not supplied the linearization uses root-level inport and outport blocks in the model. You can also supply an additional argument, `options`, to provide options for the linearization. Create `options` with the function `linoptions`.

`lin = linearize('sys','StateOrder',stateorder)` takes the Simulink model `'sys'` and creates a linear-time-invariant state-space model, `lin`, whose states are in a specified order. Specify the state order in the cell array `stateorder` by entering the names of the blocks containing states in the model `'sys'`. For all blocks, you can enter block names as the full block path. For continuous blocks, you can alternatively enter block names as the user-defined unique state name.

Note For all syntaxes, `linearize` automatically uses the following properties in the Simulink model:

- `BufferReuse = 'off'`
- `RTWInlineParameters = 'on'`
- `BlockReductionOpt = 'off'`

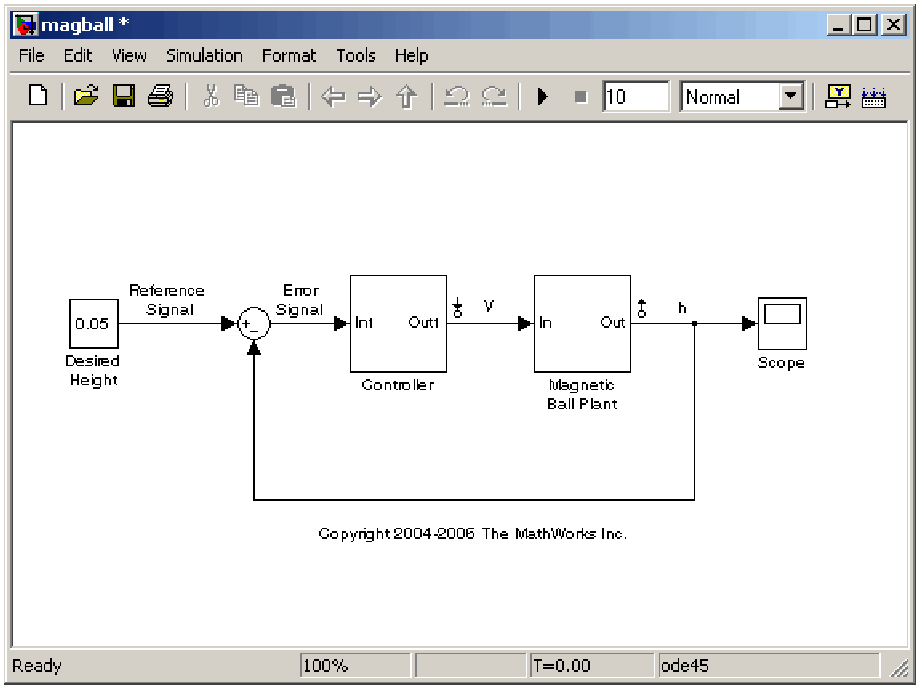
Simulink restores the original property values after creating the linearized model.

Algorithms

The function `linoptions` sets the linearization algorithm options and then passes them to the function `linearize` as an optional argument.

Examples

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



Create an I/O object based on the linearization annotations, create an operating point specification object for the model, and then find the operating point using `findop`.

```
io=getlinio('magball');
op=operspec('magball');
op=findop('magball',op);
```

Compute a linear model of the `magball` system, based on the linearization I/Os, `io`, and defined about the operating point, `op`, with the command

```
lin=linearize('magball',op,io)
```

which returns

```
a =
                 Controller       Current          dhdt         height
   Controller             0             0             0             -1
   Current              -50          -100             0              0
   dhdt                   0        -2.801             0          196.2
   height                 0             0             1              0


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


c =
               Controller     Current      dhdt      height
   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:

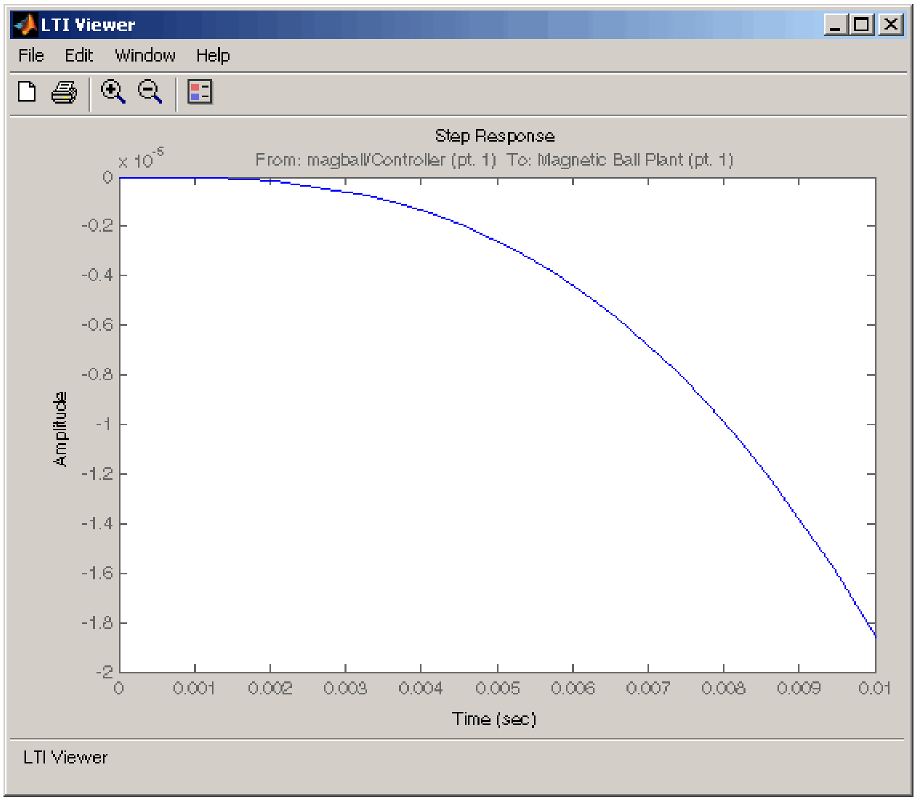
$$\dot{x}(t) = ax(t) + bu(t)$$
$$y(t) = cx(t) + du(t)$$

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, by typing:

```
ltiview(lin)
```

which produces the following plot.



See Also

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

linio

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 syntax displays:

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

By default, this I/O is an input point. Create a second I/O setting within the object, `io`. This I/O originates from the Magnetic Ball Plant block, is an output point and is also an open-loop point.

```
io(2)=linio('magball/Magnetic Ball Plant',1,'out','on')
```

The new object, `io`, is displayed as follows:

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

Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
```

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

See Also

`getlinio`, `linearize`, `setlinio`

Purpose

Set options for linearization and finding operating points

Syntax

```
opt=linoptions
opt=linoptions('Property1','Value1','Property2','Value2',
   ...)
```

Graphical Interface

As an alternative to the `linoptions` function, set options for linearization and finding operating points with the Simulink Control Design GUI.

Description

`opt=linoptions` creates a linearization options object with the default settings. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

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

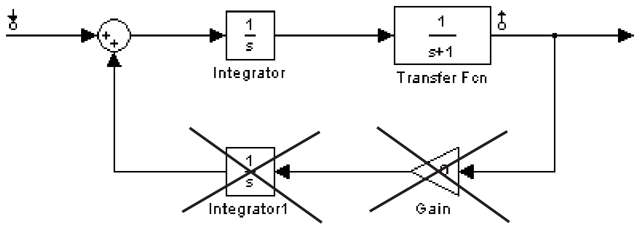
The following options can be set with `linoptions`:

`LinearizationAlgorithm`	Set to `'numericalpert'` (default is `'blockbyblock'`) to enable numerical-perturbation linearization (as in Simulink 3.0 software) where root-level inports and states are numerically perturbed. Linearization annotations are ignored and root-level inports and outports are used instead.
`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 those blocks that are not in the path of the linearization. Block reduction eliminates the states of blocks in dead linearization paths from your linearization results. Some examples of dead linearization paths are linearization paths that include:

- Blocks that linearize to zero
- Switch blocks that are not active along the path
- Disabled subsystems
- Signals marked as open-loop linearization points

The linearization result of the model shown in the following figure includes only two states. It does not include states from the two blocks outside the linearization path. These states do not appear because these blocks are on a dead linearization path with a block that linearizes to zero (the zero gain block).



Set to `'off'` to return a linearized model that includes all of the states of the model.

`IgnoreDiscreteStates`	Set to `'on'` when performing continuous linearization (SampleTime set to 0) to remove any discrete states from the linearization and accept the D value for all blocks with discrete states. Set to `'off'` (default) to include discrete states.

`RateConversionMethod`	When you linearize a multirate system, set this option to one of the following rate conversion methods:

- `'zoh'` (default) to use the zero order rate conversion method
- `'tustin'` to use the Tustin (bilinear) method
- `'prewarp'` to use the Tustin approximation with prewarping
- `'upsampling_zoh'` to upsample discrete states when possible and to use `'zoh'` otherwise
- `'upsampling_tustin'` to upsample discrete states when possible and to use `'tustin'` otherwise
- `'upsampling_prewarp'` to upsample discrete states when possible and to use `'prewarp'` otherwise

---

Note When you select `'prewarp'` or `'upsampling_prewarp'`, set the `PreWarpFreq` option to the desired prewarp frequency.

Note You can only upsample when you convert discrete states to a new sample time that is an *integer-value-times faster* than the sampling time of the original system.

For more information, and examples, on methods and algorithms for rate conversions and linearization of multirate models, see:

- "Linearization of Multi-Rate Models" and "Rate Conversion Method Selection for Linearization" demos listed under the Simulink Control Design Demos in the demos browser.
- "Continuous/Discrete Conversions of LTI Models" and "Resampling of Discrete-Time Models" in the Control System Toolbox documentation.

`PreWarpFreq`	The critical frequency Wc (in rad/sec) used by the `'prewarp'` option when linearizing a multirate system.
`UseExactDelayModel`	Set to `'on'` to return a linear model with an exact delay representation. Set to `'off'` (default) to return a model with approximate delays. For more information, see .
`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: $\mathrm{NumericalPertRel} + 10^{-3} \times \mathrm{NumericalPertRel} \times \lvert x \rvert$ The perturbation of the system's inputs is specified by: $\mathrm{NumericalPertRel} + 10^{-3} \times \mathrm{NumericalPertRel} \times \lvert u \rvert$
`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 software, 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 software is installed. The available optimizer types are:

- `graddescent_elim`, the default optimizer, enforces an equality constraint to force the time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and the output signals to be equal to their specified 'Known' value. The optimizer fixes the states, `x`, and inputs, `u`, that are marked as 'Known' in an operating point specification and then optimizes the remaining variables.
- `graddescent`, enforces an equality constraint to force the time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and the output signals to be equal to their specified 'Known' value. `findop` also minimizes the error between the states, `x`, and inputs, `u`, that are marked as 'Known' in an operating point specification. If there are not any inputs or states marked as 'Known', `findop` attempts to minimize the deviation between the initial guesses for `x` and `u` and their trimmed values.
- `lsqnonlin` fixes the states, `x`, and inputs, `u`, that are marked as 'Known' in an operating point specification and optimizes the remaining variables. The algorithm then tries to minimize both the error in the time derivatives of the states (`dx/dt=0`, `x(k+1)=x(k)`) and the error between the outputs and their specified 'Known' value.
- `simplex` uses the same cost function as `lsqnonlin` with the direct search optimization routine found in `fminsearch`.

See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox software, you can access the documentation at `http://www.mathworks.com/support/`.

`DisplayReport`	Set to `'on'` to display the operating point summary report when running `findop`. Set to `'off'` to suppress the display of this report.

See Also `findop`, `linearize`

Purpose

Create operating point for Simulink model

Syntax

```
op = operpoint('sys')
```

Graphical Interface

As an alternative to the `operpoint` function, create operating points in the **Operating Points** node of the Simulink Control Design GUI. See "Specifying Operating Points" in the Simulink Control Design getting started 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 9-47
- "States" on page 9-47
- "Inputs" on page 9-48
- "Time" on page 9-48

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 with which the states are associated.
`x`	Vector containing the values of states in the block.
`Ts`	Vector containing the sample time and offset for the state.
`SampleType`	Set this value to `CSTATE`, for a continuous state, or `DSTATE` for a discrete state.
`inReferencedModel`	Set this value to `1`, when the state is inside a referenced model, or `0`, when it is not.
`Description`	Text string describing the block.

Inputs

`Inputs` is a vector of input objects that contains the input levels at the operating point. There is one input object per root-level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	Inport block with which the input vector is associated
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	Text string describing the input

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

Example

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

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

which returns the following:

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

States:
----------
(1.) magball/Controller/Controller
      x: 0
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05

Inputs: None
```

MATLAB software displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the values of the states at the operating point. In this example there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

See Also

`get`, `linearize`, `operspec`, `set`, `update`

Purpose

Create operating point specifications for Simulink model

Syntax

```
op_spec = operspec('sys')
```

Graphical Alternative

As an alternative to the `operspec` function, create operating point specifications in the **Operating Points** node of the Simulink Control Design GUI. See “Creating Operating Points from Specifications” in the Simulink Control Design getting started 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:

- “Model” on page 9-50
- “States” on page 9-50
- “Inputs” on page 9-52
- “Time” on page 9-52
- “Outputs” on page 9-53

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

Model

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

States

`States` describes the operating point specifications for states in the Simulink model. The `States` property is a vector of state objects that each contain specifications for particular states. There is one state specification object per block that has a state in the model. The `States` object has the following properties:

`Block`	Block with which the states are associated.
`x`	Vector containing values of states in the block. Set the corresponding value of `Known` to `1` for values that are known operating point values. Set the corresponding value of `Known` to `0` for values that are initial guesses for the operating point values. The default value of `x` is the initial condition value for the state.
`Nx`	Number of states in the block. This property is read-only.
`Ts`	Vector containing the sample time and offset for the state.
`SampleType`	Set this value to `CSTATE`, for a continuous state, or `DSTATE`, for a discrete state.
`inReferencedModel`	Set this value to `1`, when the state is inside a referenced model, or `0`, when it is not
`Known`	Vector of values set to `1`, for states whose operating points are known exactly, and set to `0`, for states whose operating points are not known exactly. Set the operating point values in the `x` property.
`SteadyState`	Vector of values set to `1`, for states whose operating points should be at equilibrium, and set to `0` for states whose operating points are not at equilibrium. The default value is `1`.
`Min`	Vector containing the minimum values of the corresponding state's operating point.

`Max`	Vector containing the maximum values of the corresponding state's operating point.
`Description`	Text string describing the block.

Inputs

`Inputs` is a vector of input specification objects that contains specifications for the input levels at the operating point. There is one input specification object per root-level inport block in the Simulink model. The `Inputs` object has the following properties:

`Block`	The inport block with which the input vector is associated.
`PortWidth`	Width of the corresponding inport.
`u`	Vector containing values of inputs. Set the corresponding value of `Known` to `1`, for values that are known operating point values. Set the corresponding value of `Known` to `0`, for values that are initial guesses for the operating point values.
`Known`	Vector of values set to `1`, for inputs whose operating points are known exactly, and set to `0`, for inputs whose operating points are not known exactly. Set the operating point values in the `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`	Text string describing the input.

Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

Outputs

`Outputs` is a vector of output specification objects that contains the specifications for the output levels at the operating point. There is one output specification object per root-level 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 with which the output vector is associated.
`PortWidth`	Width of the corresponding outport.
`PortNumber`	Port number with which the output is associated.
`y`	Vector containing values of outputs. Set the corresponding value of `Known` to `1`, for values that are known operating point values. Set the corresponding value of `Known` to `0` for values that are initial guesses for the operating point values.
`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 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`	Text string describing the output.

Example

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

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

which returns the following:

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

States:
----------
(1.) magball/Controller/Controller
      spec:  dx = 0,  initial guess:            0
      spec:  dx = 0,  initial guess:            0
(2.) magball/Magnetic Ball Plant/Current
      spec:  dx = 0,  initial guess:            7
(3.) magball/Magnetic Ball Plant/dhdt
      spec:  dx = 0,  initial guess:            0
(4.) magball/Magnetic Ball Plant/height
      spec:  dx = 0,  initial guess:         0.05

Inputs: None

Outputs: None
```

The following is displayed:

- the name of the model
- the time at which any time-varying functions in the model are evaluated
- the names of blocks containing states
- default operating point values and initial guesses (based on initial conditions of the states)
- steady-state specifications

In this example, there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. By default, MATLAB software sets the `SteadyState` property to `1` and the upper and lower bounds on the operating points to `Inf` and `-Inf` respectively. MATLAB also displays the `Inputs` and `Outputs`, although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

See Also

`addoutputspec`, `findop`, `get`, `operspec`, `linearize`, `set` , `update`

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 Simulink Control Design getting started 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 function returns the properties of `op_cond`.

```
ans =
     Model: {}
    States: {}
    Inputs: {}
      Time: {}
```

To set the value of a particular property of `op_cond`, provide the property name and the desired value of this property as arguments to `set`. For example, to change the name of the model associated with the operating point object from `'magball'` to `'Magnetic Ball'`, type:

```
set(op_cond,'Model','Magnetic Ball')
```

To view the property value and verify that the change was made, type:

```
op_cond.Model
```

which returns

```
ans =
Magnetic Ball
```

Because `op_cond` is a structure, you can set any properties or fields using dot-notation. First, produce a list of properties of the second `States` object within `op_cond`, as follows:

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

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

```
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`. These settings appear as 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

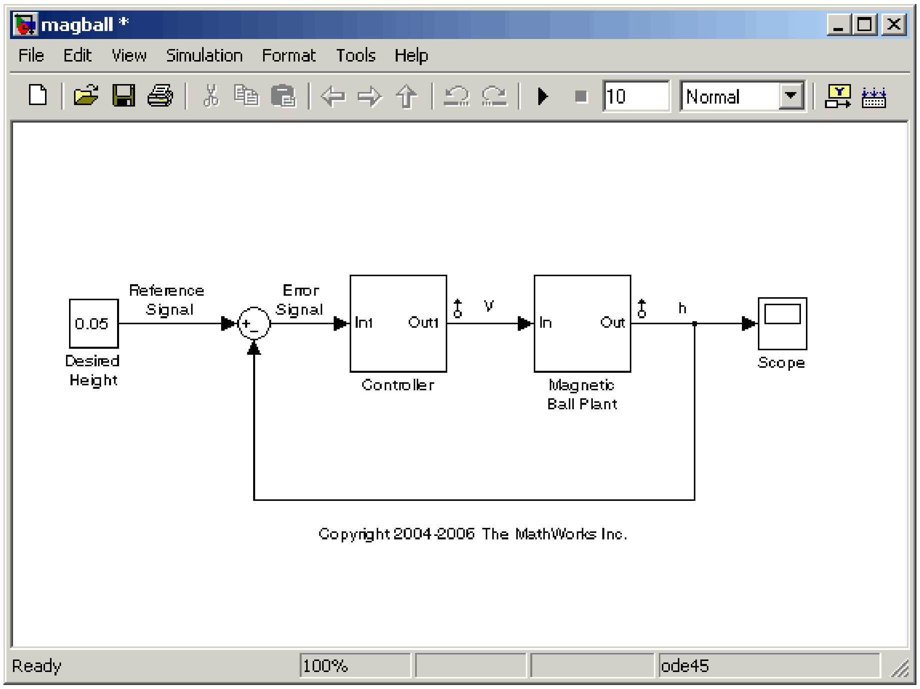
Before assigning I/O settings to a Simulink model using `setlinio`, you must create a vector of I/O objects representing linearization annotations, such as input points or output points, on a Simulink model.

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. Notice a small arrow pointing away from a small circle just above the signal line. This arrow 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 similar to 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:

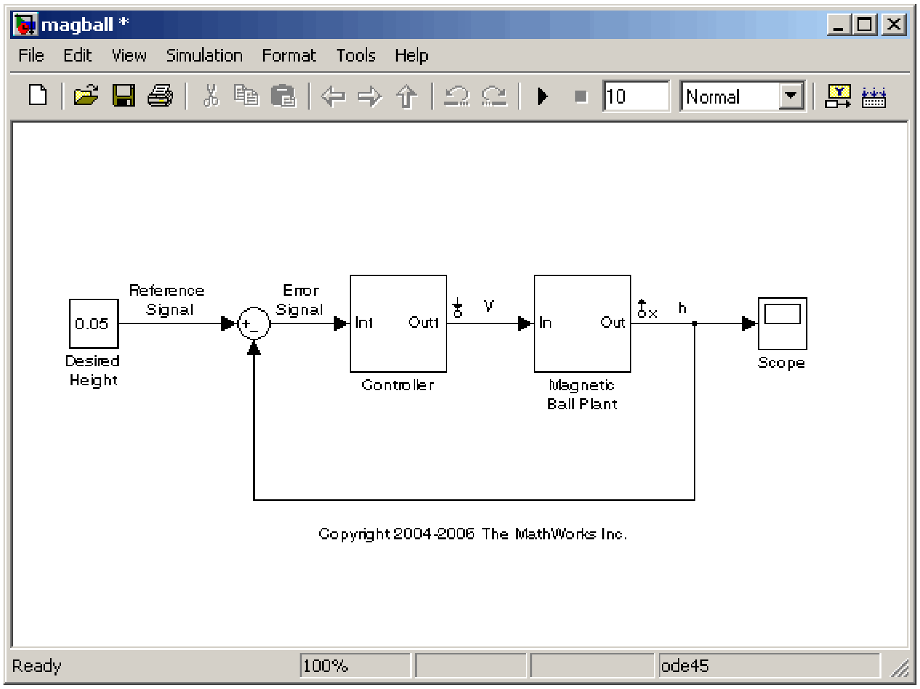
```
oldio=setlinio('magball',io)
```

This assignment returns the old I/O settings (that have been replaced by the settings in `io`).

```
    Linearization IOs:
--------------------------
Block magball/Controller, Port 1 is marked with the following
properties:
 - An Output Measurement
 - No Loop Opening
 - No signal name. Linearization will use the block name

Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
 - An Output Measurement
 - No Loop Opening
 - No signal name. Linearization will use the block name
```

The model diagram should now look similar to that in the following figure:



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 2-6 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 selection saves 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')
```

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 as follows:

```
 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 Simulink Control Design getting started 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')
```

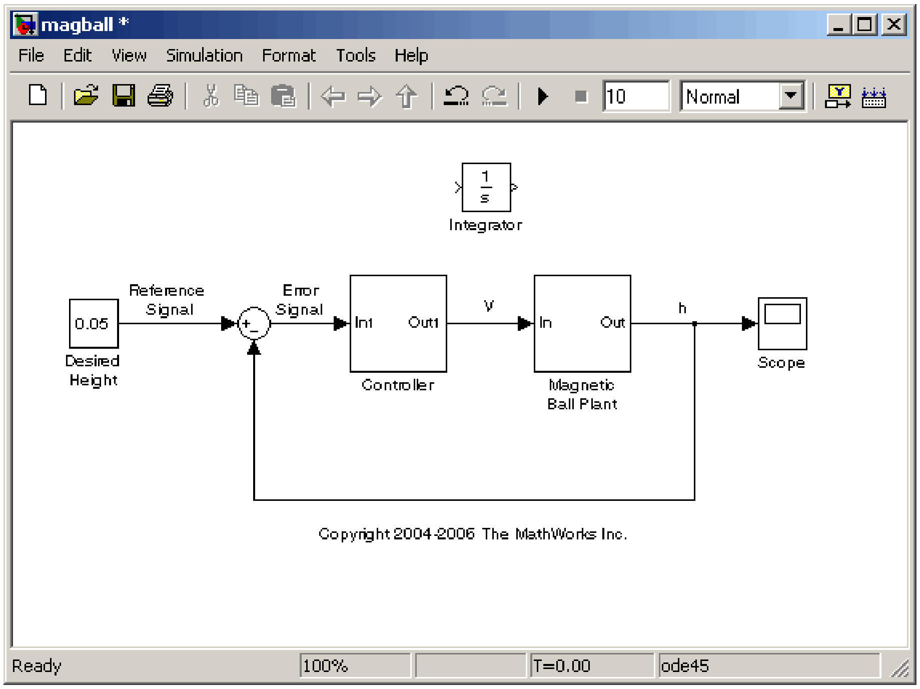
This syntax returns:

```
 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 appears:

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

States:
----------
```

```
(1.) magball/Controller/Controller
      x: 0
(2.) magball/Magnetic Ball Plant/Current
      x: 7
(3.) magball/Magnetic Ball Plant/dhdt
      x: 0
(4.) magball/Magnetic Ball Plant/height
      x: 0.05
(5.) magball/Integrator
      x: 0

Inputs: None
```

See Also operpoint, operspec

10

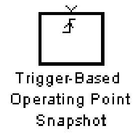
Block Reference

Trigger-Based Operating Point Snapshot

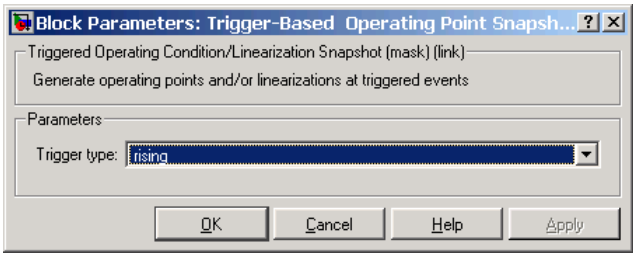
Purpose Generate operating points, linearizations, or both at triggered events

Library Simulink Control Design

Description Attach this block to a signal in a model when you want to take a snapshot of the system's operating point at triggered events such as when the signal crosses zero or when the signal sends a function call. You can also perform a linearization at these events. To extract the operating point or perform the linearization, you need to simulate the model using either the `findop` or `linearize` functions or the `simulation snapshots` option in the Control and Estimation Tools Manager.



Choose the trigger type in the Block Parameters dialog box, as shown in the following figure.


Block Parameters: Trigger-Based Operating Point Snapsh...

Triggered Operating Condition/Linearization Snapshot (mask) (link)

Generate operating points and/or linearizations at triggered events

Parameters

Trigger type: rising

The possible trigger types are

- `rising`: the signal crosses zero while increasing.
- `falling`: the signal crosses zero while decreasing.
- `either`: the signal crosses zero while either increasing or decreasing.
- `function-call`: the signal send a function call.

Note The Simulink Control Design demo called "Trigger-Based Operating Point Snapshot" illustrates how to use this block.

See Also `findop`, `linearize`

Examples

Use this list to find examples in the documentation.

Linearization Example Using Functions

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

Index