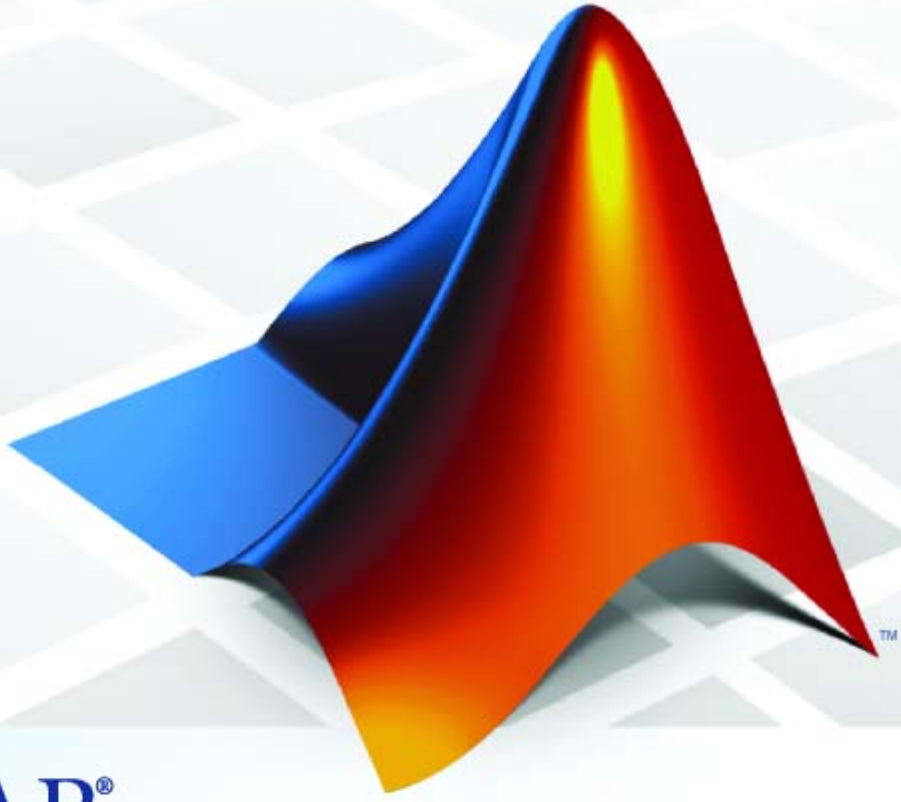
# Simulink® Response Optimization™ 3

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® Response Optimization™ User's Guide*

**Revision History**

| | | |
|---|---|---|
| June 2004 | Online only | New for Version 2.0 (Release 14) (Renamed from *Nonlinear Control Design Blockset User's Guide*) |
| October 2004 | Online only | Revised for Version 2.1 (Release 14SP1) |
| March 2005 | Online only | Revised for Version 2.2 (Release 14SP2) |
| September 2005 | Online only | Revised for Version 2.3 (Release 14SP3) |
| March 2006 | Online only | Revised for Version 3.0 (Release 2006a) |
| September 2006 | Online only | Revised for Version 3.1 (Release 2006b) |
| March 2007 | Online only | Revised for Version 3.1.1 (Release 2007a) |
| September 2007 | Online only | Revised for Version 3.1.2 (Release 2007b) |
| March 2008 | Online only | Revised for Version 3.1.3 (Release 2008a) |
| October 2008 | Online only | Revised for Version 3.2 (Release 2008b) |

# Contents

## 1 Approaching Response Optimization in Simulink Models

## 2 Response Optimization in a SISO Design Task



## 3 Function Reference

1

# Approaching Response Optimization in Simulink Models

# Choosing Signals to Constrain

**In this section...**



## Constraining Signals in Simulink Models

Simulink® Response Optimization™ software works by adjusting parameters in a Simulink® model so that chosen response signals within the system behave in a specified way. You choose the signals that you want to shape or constrain by attaching Signal Constraint blocks to them. The constraints on the behavior of the response signals and the tuned parameters are set within the Signal Constraint blocks.

The first step in the response optimization process is to choose which signals in your Simulink model you would like to constrain and to attach Signal Constraint blocks to these signals.

## Attaching Signal Constraint Blocks

Once you have selected signals to constrain, you need to attach a Signal Constraint block to each of these signals. You can find the Signal Constraint block under Simulink Response Optimization category within the Simulink Library Browser. Alternatively, you can open Simulink Response Optimization library by typing `srolib` at the MATLAB® prompt.

To attach a Signal Constraint block to a signal in your model, drag the block from the block library into the model and join the signal line to the inport of the Signal Constraint block. A model can include multiple Signal Constraint blocks, and you can attach the Signal Constraint block to any signal, including signals within subsystems of your model.

**Note** The Signal Constraint block is not an outport block of the system and does not interfere with a linearization of your model (as opposed to blocks in the Nonlinear Control Design Blockset, the previous name for this product, which were outport blocks).

# Creating a Response Optimization Project

Double-click a Signal Constraint block to open the Signal Constraint window associated with it. Within this window you can specify the constraints imposed on the signal, along with other information needed for the response optimization project.

Although you must specify the constraints for each signal individually within each Signal Constraint block, you only need to set the remaining settings such as tuned parameters and optimization settings within one Signal Constraint window as they apply to the whole project.

Opening a Signal Constraint window, automatically creates a response optimization project. The project consists of the following information:

- Constraints on all signals that have Signal Constraint blocks attached
- Tuned parameters in the system and specifications for these parameters such as initial guesses and maximum and minimum values
- Uncertain parameters in the system and specifications for these parameters
- Optimization and simulation setup options

A response optimization project exists within a single model; there are no cross-model projects. Additionally, although you can create different sets of constraints and tuned parameters and save these as different response optimization projects, you can only associate one project with the model at any time.

The remaining steps involved in specifying the settings of a response optimization project are discussed in the following sections:

- "Specifying the Desired Response" on page 1-8
- "Specifying Tuned Parameters in the Model" on page 1-25

To save the project for use in a later session, see "Saving and Reloading Response Optimization Projects" on page 1-5.

# Saving and Reloading Response Optimization Projects

**In this section...**



## Saving Response Optimization Projects

Saving a response optimization project allows you to reuse your settings during a later session. These settings include constraint bounds, tuned and uncertain parameters, and settings for optimization and simulation. Additional settings such as the position of the Signal Constraint window, axis limit settings, and the name and location of the project are saved with the Simulink *model*.

To save the constraints and data of the response optimization project to a workspace variable or a file, select **File > Save** from a Signal Constraint window in the model. Within the Save Project dialog box, you can save the project as a

- **MATLAB workspace variable**: Enter the name of a MATLAB workspace variable, and then click **OK** to save the project. This is obviously a temporary solution as the project will no longer exist once you terminate your MATLAB session.
- **Model workspace variable**: Enter the name of a model workspace variable, and then click **OK** to save the project. This method is convenient as the project is stored with the model and you do not need to worry about keeping a separate file or variable available.
- **MAT-file**: Enter a filename, and then click **OK** to save the project. Alternatively, you can save the project to an existing file by clicking the button to the right of **MAT-file** and selecting a file from the directory. Saving the project as a MAT-file is convenient when you want to save multiple projects for a single model.

To automatically reload the project when reopening the Simulink model, select the **Save and reload project with Simulink model** check box at the

bottom of the window. The Simulink model stores the location and name of the project. Hence, when you change the location or filename for the project, you must also resave the Simulink model.

When reopening a model in which a response optimization project has been previously saved and the **Save and reload project with Simulink model** check box was selected, the model searches for the variable or file containing the project and automatically loads it into the model. If the file cannot be found, a warning dialog appears.

Although the save command is issued from a single Signal Constraint window, the constraints and data from *all* Signal Constraint windows are saved as a single project that includes signal constraints, tuned parameters, uncertain parameters, and setup options.

To save the constraints and data of the response optimization project under a new name, select **File > Save As** from a Signal Constraint window in the model, and then follow the preceding instructions.

## Saving Additional Settings

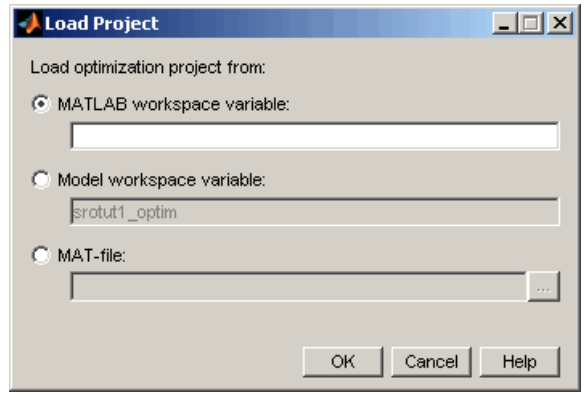
In addition to settings that you save with the response optimization project, there are several other settings that you save with the *model*. These settings include the following:

- The position of the Signal Constraint window on the screen
- The axis limit settings within the Signal Constraint window
- The location where the response optimization project, either a workspace variable or a MAT-file, is saved
- The name of the response optimization project

When you modify one or more of these attributes, you must resave the Simulink model to retain the settings when reloading the model in a subsequent session. To save the Simulink model, select **File > Save** within the model window.

## Reloading Response Optimization Projects

To reload a response optimization project from the MATLAB workspace, model workspace, or a file, select **File > Load** from a Signal Constraint window in the model. In the Load Project dialog box (shown next), enter the name of the MATLAB workspace variable, model workspace variable, or MAT-file that contains the project, and then click **OK**. Alternatively, you can load the project from an existing file by clicking the button to the right of **MAT-file** and selecting a file from the directory.




Although the load command is issued from a single Signal Constraint window, the constraints are loaded into *all* Signal Constraint blocks in the model. Additionally, tuned parameters, uncertain parameters, and optimization and simulation setup options are loaded into the model.

**Note** Loading a project cannot be undone.

# Specifying the Desired Response

**In this section...**



## Enforcing Signal Bounds

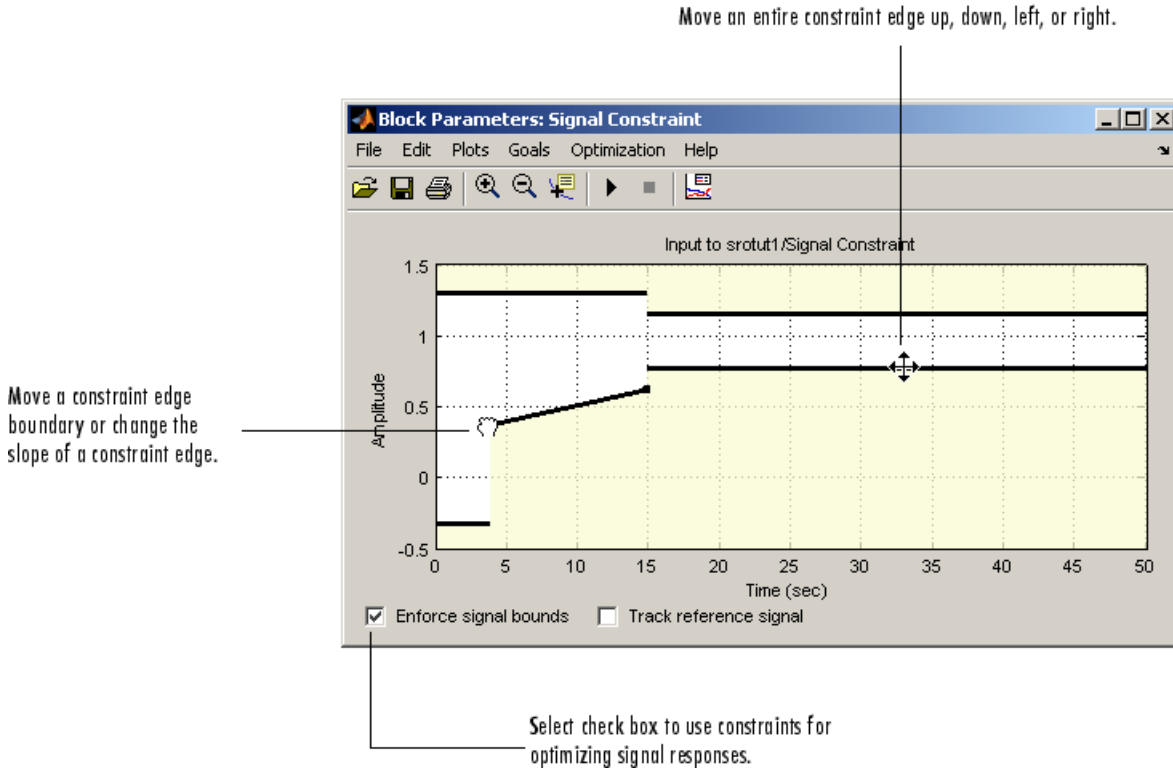
You can specify the desired response of a signal by enforcing signal bounds or by tracking a reference signal. To enforce signal bounds, select this option at the bottom of the Signal Constraint window, and then position time-domain-based constraint bound segments in the Signal Constraint window. To track a reference signal, select this option at the bottom of the Signal Constraint window, and then plot the signal in the Signal Constraint window. This section provides further details on both methods as well as instructions for editing the figure axes and plotting additional responses.

To specify the desired response signal using time-domain-based constraints, first select the **Enforce signal bounds** option at the bottom of the Signal Constraint window. Then, constrain the response signal by positioning the constraint bound segments within the figure axes using the following techniques.

## Moving Constraints

Constraint-bound segments define the time-domain constraints you would like to place on a particular signal in your model. To position these segments, which appear as a yellow shaded region bordered by a black line, use the

mouse to click and drag segments within the Signal Constraint window as shown in the following figure.




- To move a constraint segment boundary or to change the slope of a constraint segment, position the pointer over a constraint segment endpoint, and press and hold down the left mouse button. The pointer should change to a hand symbol. While still holding the button down, drag the pointer to the target location, and release the mouse button. Note that the segments on either side of the boundary might not maintain their slopes.
- To move an entire constraint segment up, down, left, or right, position the mouse pointer over the segment and press and hold down the left mouse button. The pointer should change to a four-way arrow. While still holding the button down, drag the pointer to the target location, and release the

mouse button. Note that the segments on either side of the boundary might not maintain their slopes.

---

**Tip** To move a constraint segment to a perfectly horizontal or vertical position, hold down the **Shift** key while clicking and dragging the constraint segment. This causes the constraint segment to *snap* to a horizontal or vertical position.

---

To use these constraints to optimize signal responses, make sure that the **Enforce signal bounds** check box is selected at the bottom of the window.

---

**Note** It is possible to move a lower bound constraint segment above an upper bound constraint segment, or vice versa, but this produces an error when you attempt to run the optimization.

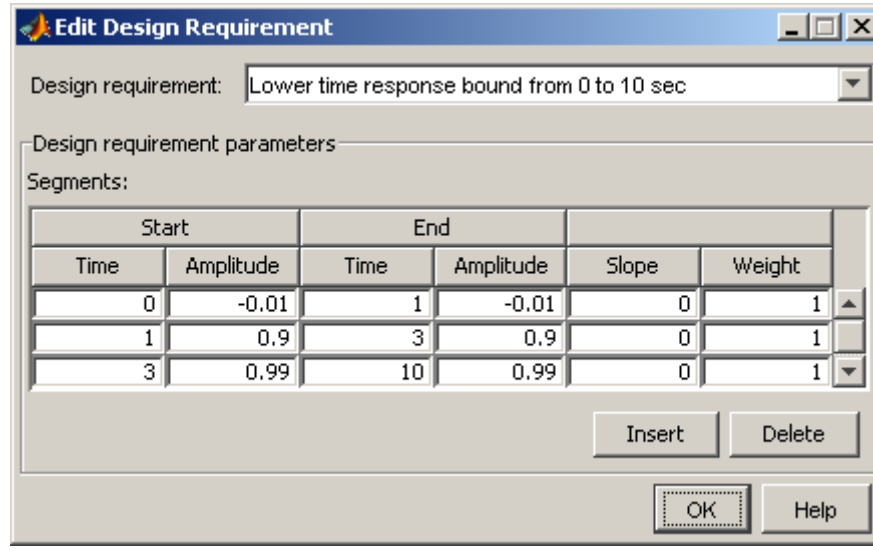
---

## Including Gridlines on the Axes

When moving constraint bound segments in the Signal Constraint window, it is sometimes helpful to display gridlines on the axes for careful alignment of the constraint bound segments. To turn the gridlines on or off, right-click within the axes of the Signal Constraint window and select **Grid**.

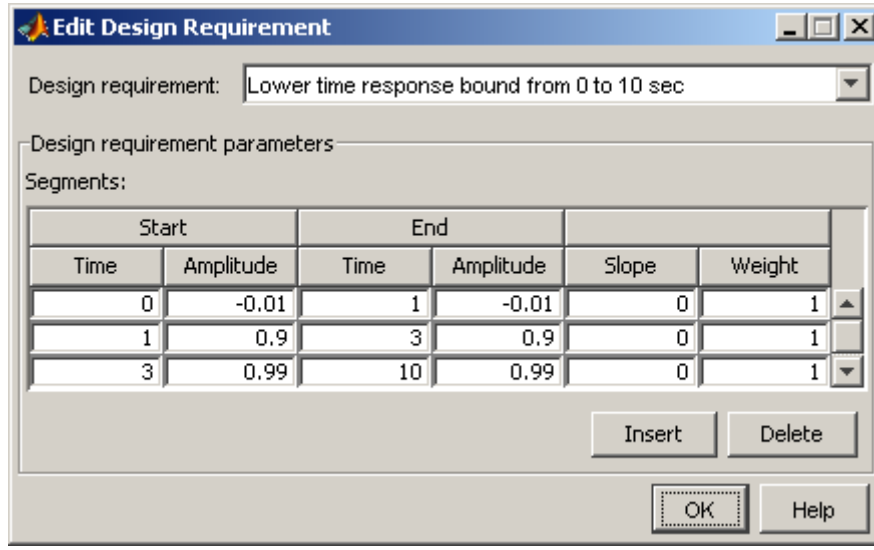
## Positioning Constraints Exactly

To position a constraint segment exactly, position the pointer over the segment you want to move and press the right mouse button. Select **Edit** from the menu to open the Edit Design Requirement dialog box, shown next. For information on using the Edit Design Requirement dialog box, see “Edit Design Requirement Dialog Box” on page 1-12.




## Adjusting Constraint Weightings

To change the weight of a constraint segment, position the pointer over the segment you want to weight and click the right mouse button. Select **Edit** from the menu to open the Edit Design Requirement dialog box, shown next. For information on using the Edit Design Requirement dialog box, see “Edit Design Requirement Dialog Box” on page 1-12.




## Edit Design Requirement Dialog Box

The Edit Design Requirement dialog box allows you to exactly position constraint segments and to edit other properties of these constraints. The dialog box has two main components:

- An upper panel to specify the constraint you are editing
- A lower panel to edit the constraint parameters

The upper panel of the Edit Design Requirement dialog box resembles the image in the following figure.


In the context of the SISO Tool in Control System Toolbox™ software, **Design requirement** refers to both the particular editor within the SISO Tool that contains the requirement and the particular requirement within that editor. To edit other constraints within the SISO Tool, select another design requirement from the drop-down menu. In the context of the Signal Constraint block, the constraints are always time-bound constraints.

### Edit Design Requirement Dialog Box Parameters

The particular parameters shown within the lower panel of the Edit Design Requirement dialog box depend on the type of constraint/requirement. In some cases, the lower panel contains a grid with one row for each segment and one column for each constraint parameter. The following table summarizes the various constraint parameters.

**Edit Design Requirement Dialog Box Parameters**

| Parameter | Found in | Description |
|---|---|---|
| **Time** | Upper and lower time response bounds on step and impulse response plots | Defines the time range of a segment within a constraint/requirement. |
| **Amplitude** | Upper and lower time response bounds on step and impulse response plots | Defines the beginning and ending amplitude of a constraint segment. |
| **Magnitude** | SISO Tool Open-Loop Bode Editor, Prefilter Bode Editor | Defines the beginning and ending amplitude of a constraint segment. |
| **Weight** | Upper and lower time response bounds on step and impulse response plots, SISO Tool Open-Loop Bode Editor, Prefilter Bode Editor, Root Locus Editor, Open-Loop Nichols Editor | Defines the weight of a segment within a constraint/requirement. The weight is a measure of the relative importance of this constraint segment when used in a response optimization project. Weights can vary between 0 and 1, where 0 implies that the constraint segment is disabled and does not have to be satisfied, and 1 implies that the constraint segment must be satisfied. The weight of a constraint segment is graphically represented by the thickness of the black constraint line. An invisible constraint segment represents a weight of 0, and a thick constraint segment represents a weight of 1. |

**Edit Design Requirement Dialog Box Parameters (Continued)**

| Parameter | Found in | Description |
|---|---|---|
| **Frequency** | SISO Tool Open-Loop Bode Editor, Prefilter Bode Editor | Defines the frequency range of an edge within a constraint. |
| **Slope (dB/decade)** | SISO Tool Open-Loop Bode Editor, Prefilter Bode Editor | Defines the slope, in dB/decade, of a constraint segment. It is an alternative method of specifying the magnitude values. Entering a new **Slope** value changes any previously defined magnitude values. |
| **Final value** | Step response bounds | Defines the input level after the step occurs. |
| **Rise time** | Step response bounds | Defines a constraint segment for a particular rise time. |
| **% Rise** | Step response bounds | The percentage of the step's range used to describe the rise time. |
| **Settling time <** | SISO Tool Root Locus Editor | Defines a constraint segment for a particular settling time. |
| **Settling time** | Step response bounds | |
| **% Settling** | Step response bounds | The percentage of the final value that defines the settling region used to describe the settling time. |
| **Percent overshoot <** | SISO Tool Root Locus Editor | Defines the constraint segments for a particular percent overshoot. |
| **% Overshoot** | Step response bounds | |
| **% Undershoot** | Step response bounds | Defines the constraint segments for a particular percent undershoot. |
| **Damping ratio >** | SISO Tool Root Locus Editor | Defines the constraint segments for a particular damping ratio. |

**Edit Design Requirement Dialog Box Parameters (Continued)**

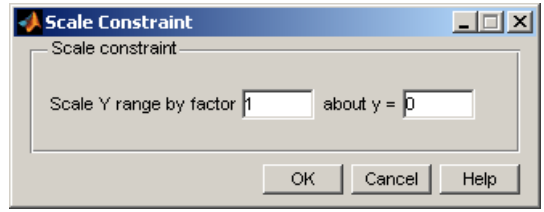
| Parameter | Found in | Description |
|---|---|---|
| **Natural frequency** | SISO Tool Root Locus Editor | Defines a constraint segment for a particular natural frequency. To specify the constraint, choose `at least` or `at most` from the menu, and then specify the natural frequency of interest. |
| **Real** | SISO Tool Root Locus Editor | Defines the beginning and end of the real component of a pole-zero region constraint. |
| **Imaginary** | SISO Tool Root Locus Editor | Defines the beginning and end of the imaginary component of a pole-zero region constraint. |
| **Phase margin >** | SISO Tool Open-Loop Nichols Editor | Defines a constraint segment for a minimum phase margin. The phase margin specified should be a number greater than 0. |
| **Located at** | SISO Tool Open-Loop Nichols Editor | Defines the center, in degrees, of the constraint segment defining the phase margin, gain margin, or closed-loop peak gain. The location must be -180 plus a multiple of 360 degrees. If you enter an invalid location point, the closest valid location is selected. |
| **Gain margin >** | SISO Tool Open-Loop Nichols Editor | Defines a constraint segment for a particular gain margin. |
| **Closed-Loop peak gain <** | SISO Tool Open-Loop Nichols Editor | Defines a constraint segment for a particular closed-loop peak gain. The specified value can be positive or negative in dB. The constraint follows the curves of the Nichols plot grid, so we recommend that you have the grid on when using this feature. |

**Edit Design Requirement Dialog Box Parameters (Continued)**

| Parameter | Found in | Description |
|---|---|---|
| **Open loop phase** | SISO Tool Open-Loop Nichols Editor | Defines the beginning and end of the open loop phase component of a gain-phase constraint segment. |
| **Open loop gain** | SISO Tool Open-Loop Nichols Editor | Defines the beginning and end of the open loop gain component of a gain-phase constraint segment. |

## Scaling Constraints

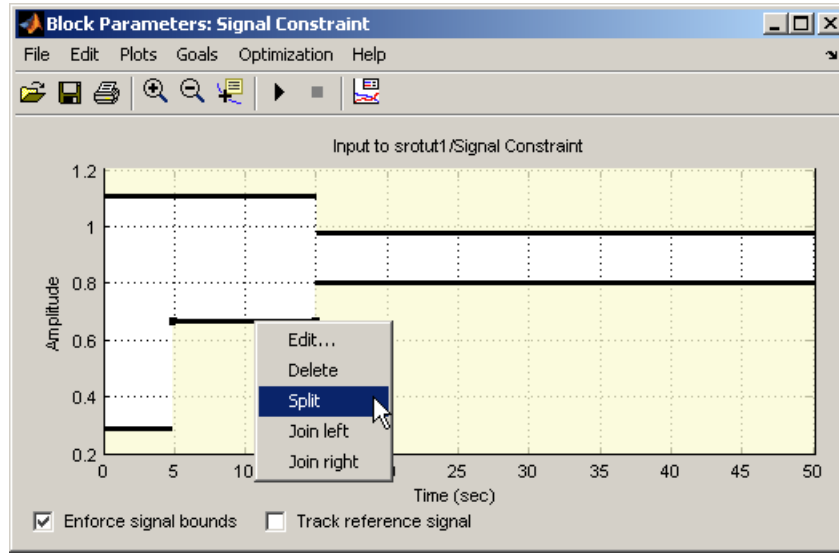
Instead of clicking and dragging the constraints to their new positions, you can scale the constraints. To scale the constraints, select **Edit > Scale Constraint** in the Signal Constraint window. This displays the Scale Constraint dialog box.




Enter the amount by which you want the constraints to scale and the point about which you want to scale them, and then click **OK**.

## Splitting and Joining Constraints

To split a constraint segment, position the pointer over the segment to be split, and press the right mouse button. Select **Split** from the context menu. The segment splits in half. You can now manipulate each segment individually.

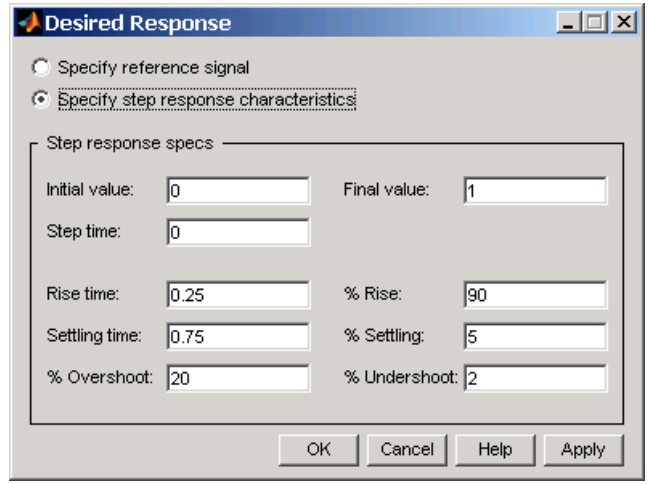



To join two neighboring constraint segments, position the pointer over one constraint segment, and press the right mouse button. Select **Join left** or **Join right** from the menu to join the segment to the left or right respectively.

## Choosing Step Response Specifications

When you are optimizing the step response of your system, an alternative method of positioning the constraint bound segments is to specify the desired step response characteristics such as rise time, settling time, and overshoot.

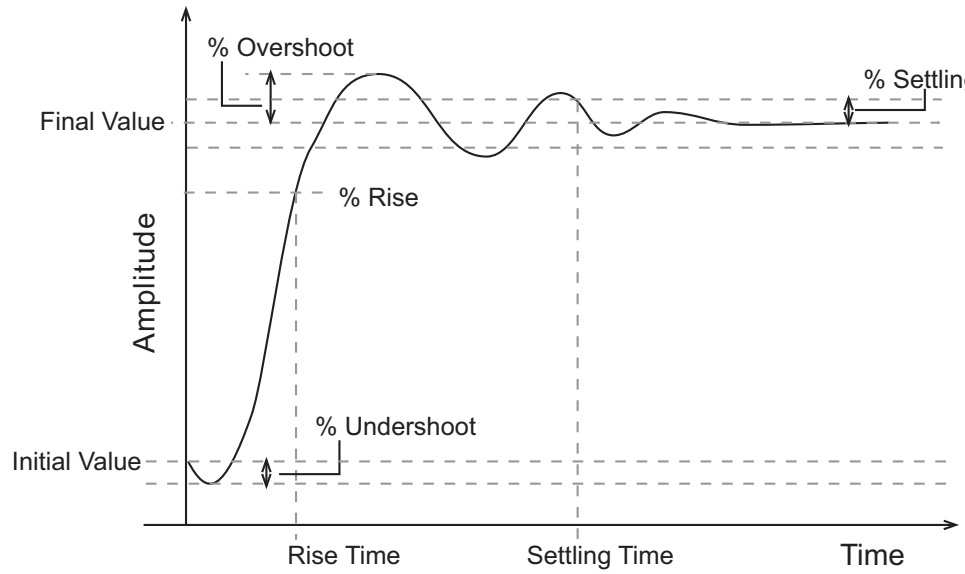
To specify step response characteristics, select **Goals > Desired Response** in the Signal Constraint window or right-click in the white space of the figure window and select **Desired Response** from the context menu. This displays the Desired Response dialog box. Select **Specify step response characteristics** to display the step response specifications as shown in the following figure.




The top three options specify the details of the step input:

- **Initial value**: Input level before the step occurs
- **Step time**: Time at which the step takes place
- **Final value**: Input level after the step occurs

The remaining options specify the characteristics of the response signal. Each of the step response characteristics is illustrated in the following figure.




- **Rise time**: The time taken for the response signal to reach a specified percentage of the step's range. The step's range is the difference between the final and initial values.
- **% Rise**: The percentage used in the rise time.
- **Settling time**: The time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
- **% Settling**: The percentage used in the settling time.
- **% Overshoot**: The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step's range. The step's range is the difference between the final and initial values.
- **% Undershoot**: The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step's range. The step's range is the difference between the final and initial values.

Enter values for the response specifications in the Response Specifications dialog box, based on the requirements of your model, and then click **OK**. The constraint segments now reflect the constraints specified.

# Tracking Reference Signals

**In this section...**

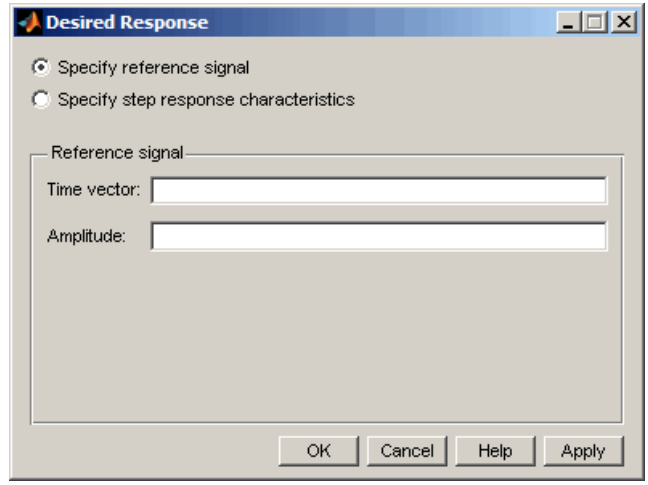


## Specifying a Reference Signal

You can specify the desired response as an ideal or *reference* trajectory. First, select the **Track reference signal** option at the bottom of the Signal Constraint window. Then, plot the reference signal within the figure axes using the following techniques. You can use this reference signal in addition to, or instead of, enforcing signal bounds.

## Plotting the Reference Signal

Plot a reference signal by selecting **Goals > Desired Response** in the Signal Constraint window or by right-clicking in the white space of the figure window and selecting **Desired Response** from the context menu. This displays the Desired Response dialog box. Select the radio button labeled **Specify reference signal** to display the reference signal setup as shown in the following figure.



Define the reference signal by entering vectors, or variables from the workspace, for the time and amplitude of the signal, and then clicking **OK**. To turn the reference signal on or off, right-click in the white space of the figure window and select **Show > Reference Signal**.

# Plotting Responses in the Signal Constraint Window

| In this section... |
| --- |
|  |
|  |
|  |
|  |
|  |

## Types of Response Plots

You can choose to plot several different signals in the Signal Constraint window, including reference signals, initial response signals, and response signals generated during the optimization.

## Reference Signals

To plot a reference signal, use the methods in "Plotting the Reference Signal" on page 1-21.

## Current Response

To display the current response signal, based on the current parameter values, right-click within the white space of the Signal Constraint window and select **Plot Current Response**. The current response appears as a thick white line.

## Initial Response

To turn the display of the initial response signal on or off, right-click within the white space of the Signal Constraint window and select **Show > Initial Response**. The initial response is the response of the signal based on parameter values in place before the optimization is run. The initial response appears as a blue line.

## Intermediate Steps

To turn on, or off, the display of the response signal at intermediate steps during the optimization, right-click within the white space of the Signal Constraint window and select **Show > Intermediate Steps**. The response signal at an intermediate step is based on parameter values at an intermediate point in the optimization.

# Specifying Tuned Parameters in the Model

**In this section...**



## Defining Tunable Parameters

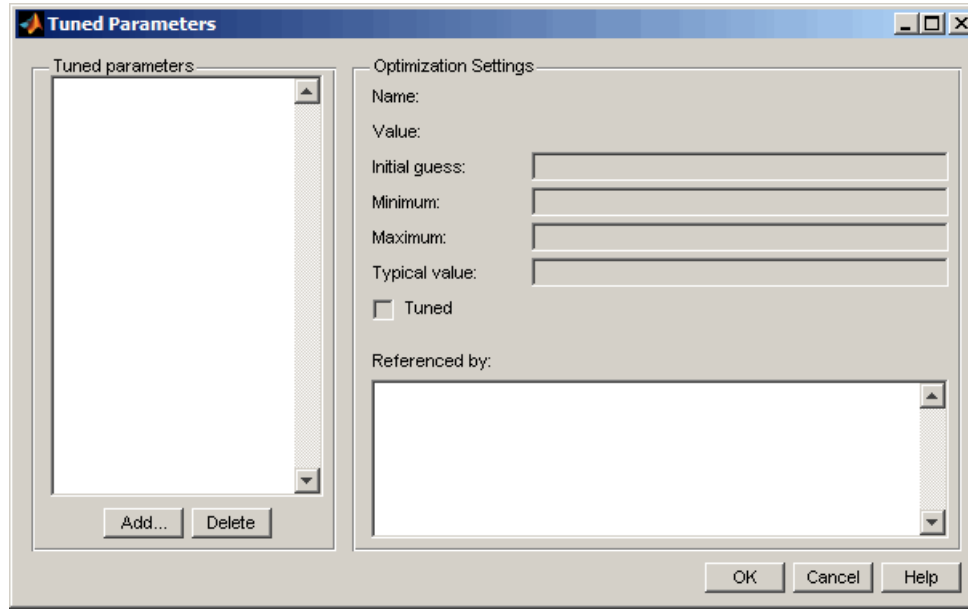
Before running the optimization, you need to define which system parameters are tunable. By tuning these parameters, Simulink Response Optimization software makes the response signal meet the imposed constraints. In addition, you can define other parameters to account for plant uncertainty in your response optimization project.

Simulink Response Optimization software optimizes the response signals of the model by varying the model's tuned parameters so that the response signals lie within the constraint bound segments or closely match a specified reference signal. You can specify these tuned parameters by selecting **Optimization > Tuned Parameters** in a Signal Constraint window.

---

**Note** When you have more than one Signal Constraint block in your model, you need to specify the tuned parameters in only one window as these settings apply to all constrained signals within the model.

---




## Adding Tuned Parameters

Within the Tuned Parameters dialog box, the tuned parameters are shown in a list on the left. To add a tuned parameter to your response optimization project, click the **Add** button. This displays the Select Parameters dialog box listing all model parameters of the model currently available in the MATLAB workspace. (If a parameter is already listed in the tuned parameters list, it does not appear in the Select Parameters dialog box.)

Select the parameters that you want to tune, then click **OK** to add them to the list of tuned parameters. To delete a parameter from the tuned parameters list, select the parameter you want to delete and click **Delete**.

## Changing Tuned Parameter Specifications

To display the settings for a particular tuned parameter, select it within the **Tuned Parameters** list. Its settings appear on the right under **Optimization Settings**, as listed in the following table.

| Setting | Description | Default |
|---|---|---|
| **Name** | The name of the parameter. | Not an editable field |
| **Value** | The current value of the parameter. | Not an editable field |
| **Initial guess** | The initial value used by the optimization algorithm. A well-chosen initial guess can speed up the optimization and help keep the solution away from undesirable local minima. You can edit this field with numbers, variables, or expressions to provide an alternate initial guess. | The current value of the parameter |
| **Minimum** | The minimum value, or lower bound, that you would like the parameter to take. You can edit this field to provide an alternate minimum value. | `-Inf` |
| **Maximum** | The maximum value, or upper bound, that you would like the parameter to take. You can edit this field to provide an alternate maximum value. | `Inf` |
| **Typical value** | The tuned parameters are scaled, or normalized, by dividing their current value by a typical value. You can edit this field to provide an alternate scaling factor. | The initial value of the parameter |
| **Tuned** | This check box indicates whether this parameter is tunable. Select it if you want this parameter to be tuned during the optimization. Unselect if you do not want this parameter to be tuned during the optimization but you would like to keep it on the list of tuned parameters (for a subsequent optimization). | Selected |
| **Referenced by** | A list of all blocks this parameter appears in. | Not an editable field |

After selecting the tuned parameters for the project and editing their optimization settings, click **OK** to save your changes and exit the Tuned Parameters dialog box.

# Including Uncertainty in Parameter Values

**In this section...**



## What Are Uncertain Parameters

As discussed in "Simple Control Design Example" in Simulink Response Optimization Getting Started Guide, a precise plant model might not be known for your particular problem. Instead, you might know what the nominal plant should be and have some idea of the uncertainty inherent in various components of the plant. For example, in "Adding Uncertainty", the plant parameter `zeta` varies up to 5% about its nominal value and `w0` varies between 0.7 and 1.45.

You can incorporate uncertainty into your design in two different ways:

- Passive mode: Optimize the signals based on the nominal parameter values only. Use the uncertain parameter values to validate the results by plotting responses based on these perturbed values.
- Active mode: Optimize signals based on both nominal parameter values as well as perturbed (uncertain) parameter values. This mode is more time consuming.

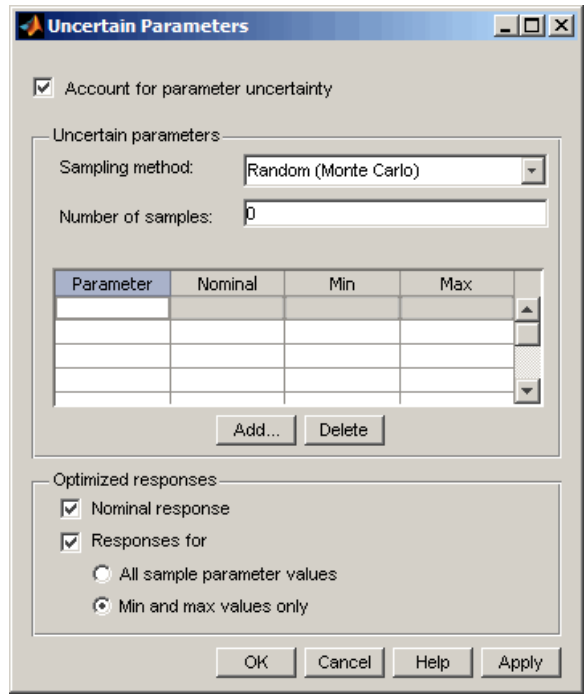
## Adding Uncertain Parameters

To specify uncertainty in parameters, select **Optimization > Uncertain Parameters** from the Signal Constraint window.

---

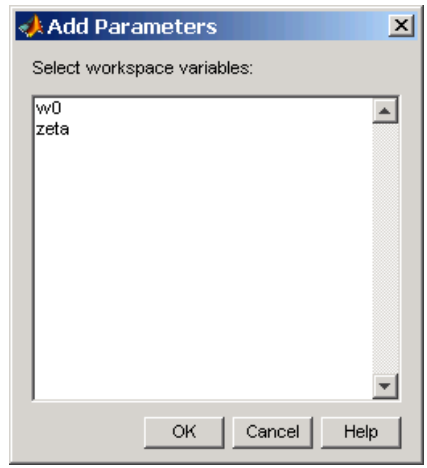
**Note** When you have more than one Signal Constraint block in your model, you need to specify the uncertain parameters in only one window, as these settings apply to all constrained signals within the model.

---




By default, the **Account for parameter uncertainty** check box is selected when you open the Uncertain Parameters dialog box. This indicates that you want to include parameter uncertainty in your optimization. By clearing this option, you can turn off parameter uncertainty without deleting information you have already entered in the **Uncertain parameters** list.

To add a new uncertain parameter to the **Uncertain parameters** list, click the **Add** button. This displays the Select Parameters dialog box listing all model parameters currently available in the MATLAB workspace. (If a parameter is already listed in either the tuned parameters or uncertain parameters list, it will not appear in the Select Parameters dialog box.)




Select the parameters that you want to add uncertainty to, and then click **OK** to add them to the list of uncertain parameters. To delete a parameter from the **Uncertain parameters** list, select the parameter you want to delete and click **Delete**.

## Changing Uncertain Parameter Specifications

There are two sampling methods that you can use to investigate the uncertain parameters. Both involve using several sample parameter values within the range of uncertainty.

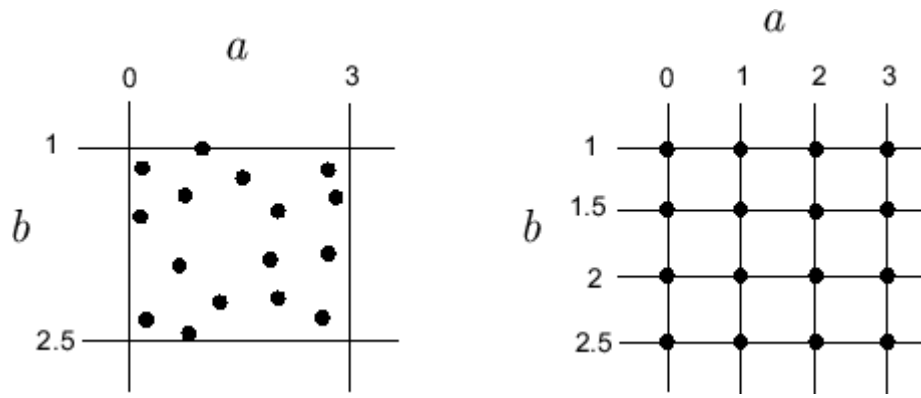
- **Random (Monte Carlo)** computes the optimization at several random parameter values within the range of uncertainty. When this method is selected, you must also enter a value for **Number of samples**, which indicates the number of random parameters that Simulink Response Optimization software uses. For each parameter in the uncertain parameters list, you can change the nominal value as well as the range of uncertainty indicated by the maximum and minimum values the parameter can take.

  When more than one parameter contains uncertainty, random parameter combinations are chosen within the hyper-rectangle defined by the minimum and maximum values of all parameters. For example, in the case of two uncertain parameters $a$ and $b$, with values ranging from 0 to 3 and from 1 to 2.5 respectively, the sample values, represented by black

dots, are scattered randomly within the rectangle shown on the left of the following figure.

- **Grid** computes the optimization at several specified parameter values within the range of uncertainty. For each parameter in the uncertain parameters list, you can change the nominal value as well as specify a vector of sample parameter values. **Number of samples** is computed from the sample values specified in the list.

  When more than one parameter contains uncertainty, the sample values form a grid of parameter combinations. For example, in the case of two uncertain parameters $a$ and $b$, with sample values `[0 1 2 3]` and `[1 1.5 2 2.5]`, the sample values, represented by black dots, form the grid of parameter combinations shown on the left of the following figure.




To increase the speed of the computation, you can choose not to use all sample parameter values in the optimization. To include the nominal parameter values in the optimization, select the **Nominal response** check box. To include parameter values other than the nominal value in the optimization, select the **Response for** check box and then select either **All sample parameter values** or **Min and max values only**. These options include either all sample parameter value combinations or all combinations of minimum and maximum parameter values, respectively. Only the optimized responses are used to adjust the tuned parameters. Responses based on other sample parameter values may still be plotted in the Signal Constraint window.

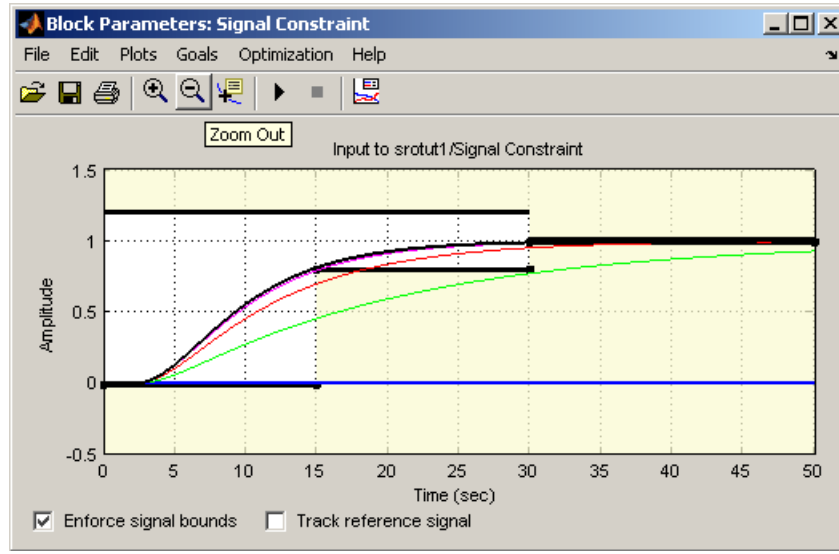
# Running the Optimization

After you have specified constraints and set the tuned and uncertain parameters, you can run the optimization. If the optimization does not converge the first time, it often converges after adjusting the constraints or tuned parameter characteristics, or choosing different options. The latter sections of this chapter give advice on how set optimization options and options for the simulations used in the optimization.

Simulink Response Optimization software uses optimization algorithms to find parameter values that allow a feasible solution, or best fit in the case of reference tracking, to the given constraints. Once the appropriate signals have been constrained with signal bounds or by tracking a reference signal, the tuned parameters set, and (optionally) any uncertain parameters and optimization settings specified, you are ready to run the optimization.

Run the optimization by selecting **Optimization > Start** in the Signal Constraint window, or click the **Start** button, which is the small triangle located on the control panel below the menus.

Simulink Response Optimization software begins by plotting the initial response in blue in the Signal Constraint window. During the optimization, intermediate responses are also plotted in various colors. The final response is plotted in black. If uncertainty is included in the optimization, the uncertain response signals are plotted as dashed lines, along with the nominal response as a solid line.



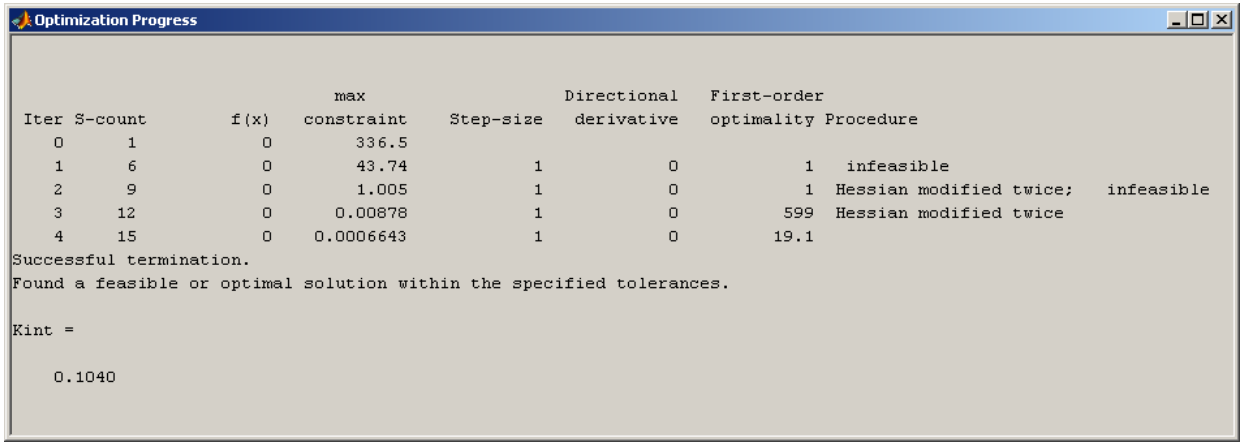

Simulink Response Optimization software changes the values of the tuned parameters within the MATLAB workspace and displays the final value in the Optimization Progress window. Alternatively, you can enter a parameter name at the MATLAB prompt to see its final value.

---

**Note** After the optimization, the values of the tuned parameters are changed to the new optimized values. This means that if you want to run another optimization, it uses these tuned values of the parameters as initial values, unless you specify alternative initial values in the Tuned Parameters dialog box. To revert to the unoptimized parameter values, select **Edit > Undo Optimize Parameters** from the Signal Constraint window.

---

The Optimization Progress window displays numerical output. The form of this output depends on the optimization algorithm being used. To learn more, see “Selecting Optimization Methods” on page 1-36 and the discussion of **Display level** in “Selecting Additional Optimization Options” on page 1-37.




**Note** The `Gradient descent` optimization algorithm may violate the bounds on parameter values when it cannot satisfy the signal constraints specified in the Signal Constraint block and the bounds on parameter values simultaneously. To learn how to troubleshoot this problem, see the "Questions" in the Troubleshooting section.

# Tuning the Optimization Results

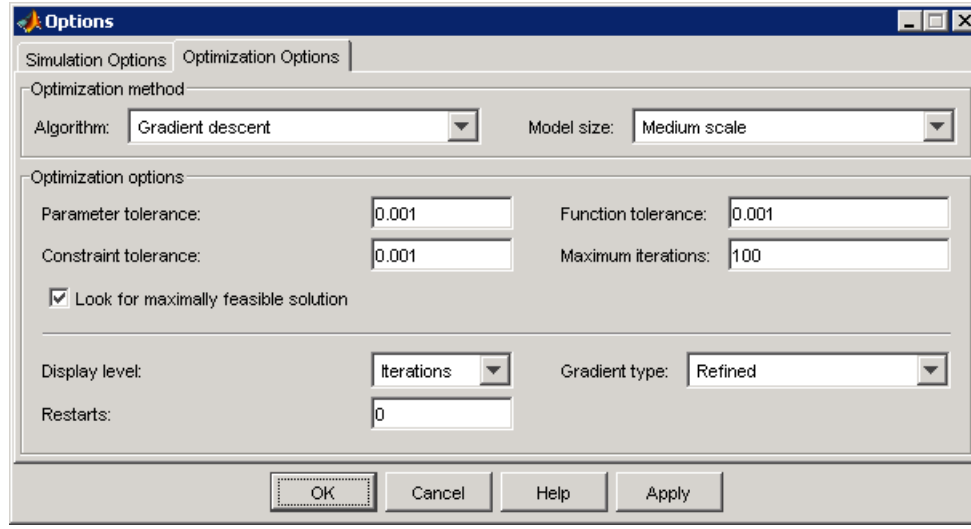
**In this section...**



## Accessing Optimization Options

Several options can be set to tune the results of optimization. These options include the optimization algorithm and the tolerances the algorithms use.

To set options for optimization, select **Optimization > Optimization Options** in the Signal Constraint window. This opens the Options dialog box.



**Note** If the optimization fails, a good first work-around is to change the **Gradient-type** to `Refined`. For more information on this option, refer to "Selecting Additional Optimization Options" on page 1-37.

## Selecting Optimization Methods

Both the algorithm and model size define the optimization method. Use the **Optimization Options** panel in the Options dialog box to set algorithm and the model size.


For the **Algorithm** parameter, the three options are

- `Gradient descent` — Uses the Optimization Toolbox™ function `fmincon` to optimize the response signal subject to the constraints.
- `Pattern search` — Uses the Genetic Algorithm and Direct Search Toolbox™ function `patternsearch`, an advanced direct search method, to optimize the response. This option requires the Genetic Algorithm and Direct Search Toolbox.
- `Simplex search` — Uses the Optimization Toolbox function `fminsearch`, a direct search method, to optimize the response. `Simplex search` is most useful for simple problems and is sometimes faster than `Gradient descent` for models that contain discontinuities.

By default, the **Model Size** parameter is set to `Medium scale`. When the model is very large and `Gradient descent` is selected as the optimization algorithm, you can change **Model Size** to `Large scale` to increase computation speed. For more information about the optimization models, see the Optimization Toolbox documentation or the Genetic Algorithm and Direct Search Toolbox documentation.

## Selecting Optimization Termination Options

Use the **Optimization options** panel to specify when you want the optimization to terminate.


- **Parameter tolerance**: When using the `Simplex search` algorithm, the optimization terminates when successive parameter values change by less than this number. For more details, refer to the discussion of the parameter `TolX` in the reference page for the Optimization Toolbox function `fmincon`.
- **Constraint tolerance**: This number represents the maximum relative amount by which the constraints can be violated and still allow a successful convergence.
- **Function tolerance**: The optimization terminates when successive function values are less than this value. Changing the default **Function tolerance** value is only useful when you are tracking a reference signal or using the `Simplex search` algorithm. For more details, refer to the discussion of the parameter `TolFun` in the reference page for the Optimization Toolbox function `fmincon`.
- **Maximum iterations**: The maximum number of iterations allowed. The optimization terminates when the number of iterations exceeds this number.
- **Look for maximally feasible solution**: When selected, the optimization continues after it has found an initial solution, until it finds a maximally feasible, optimal solution. When this option is unselected, the optimization terminates as soon as it finds a solution that satisfies the constraints and the resulting response signal sometimes lies very close to the constraint segment. In contrast, a maximally feasible solution is typically located further inside the constraint region.

By varying these parameters you can force the optimization to continue searching for a solution or to continue searching for a more accurate solution.

## Selecting Additional Optimization Options

At the bottom of the **Optimization Options** panel is a group of additional optimization options.


## Display Level

The **Display level** option specifies the form of the output that appears in the Optimization Progress window. The options are `Iterations`, which displays information after each iteration, `None`, which turns off all output, `Notify`, which displays output only if the function does not converge, and `Termination`, which only displays the final output.

For more information on the type of iterative output that appears for the algorithm you selected using the **Algorithm** option, see the discussion of output for the corresponding function.

| Algorithm | Function | Output Information |
|---|---|---|
| `Gradient descent` | `fmincon` | `fmincon` section of "Function-Specific Output Headings" in the Optimization Toolbox documentation |
| `Simplex search` | `fminsearch` | `fminsearch` section of "Function-Specific Output Headings" in the Optimization Toolbox documentation |
| `Pattern search` | `patternsearch` | "Display to Command Window Options" in the Genetic Algorithm and Direct Search Toolbox documentation |

## Restarts

In some optimizations the Hessian may become ill conditioned and the optimization does not converge. In these cases it is sometimes useful to restart the optimization after it stops, using the endpoint of the previous optimization as the starting point for the next one. To automatically restart the optimization, indicate the number of times you want to restart in this field.

## Gradient Type

When using `Gradient descent` as the optimization algorithm, Simulink Response Optimization software calculates gradients based on finite

difference methods. The default method for computing the gradients is `Basic`. The `Refined` method offers a more robust and less noisy gradient calculation method than `Basic`, although it is sometimes more expensive and does not work with certain models such as SimPowerSystems™ models. If the optimization fails, a good first work-around, before changing solvers or adding parameter bounds, is to change **Gradient type** to `Refined`.

# Setting the Simulation Options

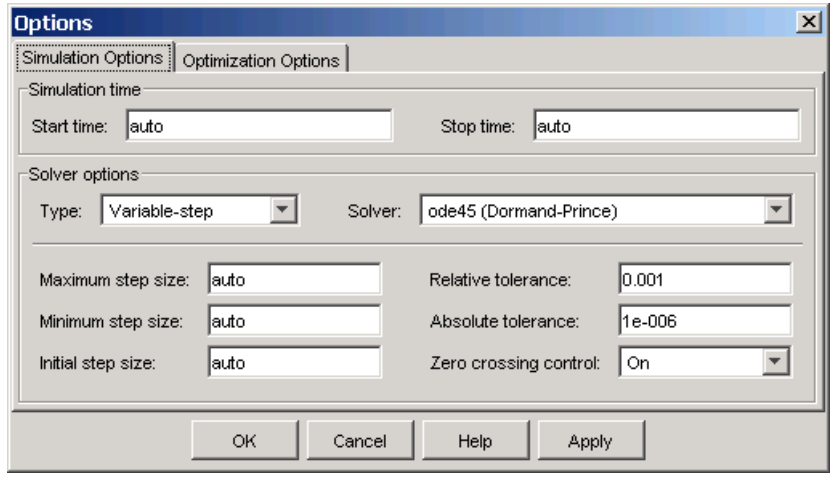
**In this section...**



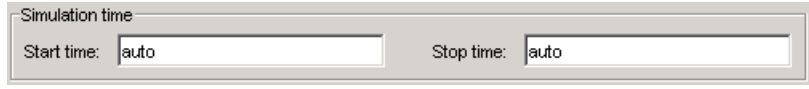
## Accessing Simulation Options

To optimize the response signals of a model, Simulink Response Optimization software runs simulations of the model.

You can set options for these simulations by selecting **Optimization > Simulation Options** in the Signal Constraint window. This opens the Options dialog box.




## Selecting Simulation Time



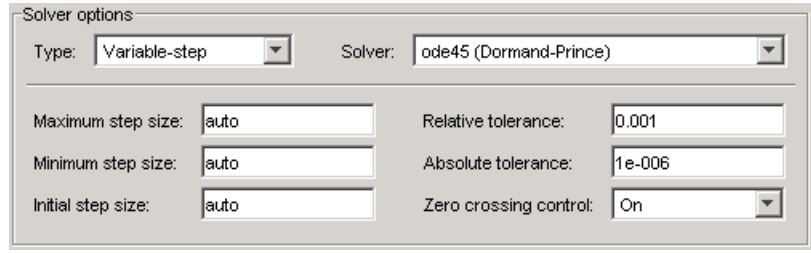
By default, the **Start time** and **Stop time** are automatically set to the model's start and stop times. To specify alternative start and stop times for the response optimization project, enter them under **Simulation time**.

---

**Note** Simulink Response Optimization software automatically replaces the stop-time value in **Stop time** with the largest time value in the constraints. This prevents the software from entering an infinite loop.

---

## Selecting Solvers




When running the simulation, Simulink software solves the dynamic system using one of several solvers. You can specify several solver options under **Solver options** in the Options dialog box. The type of solver can be variable-step or fixed step. Variable step solvers keep the error within specified tolerances by adjusting the step size the solver uses. Fixed-step solvers use a constant step-size. When your model's state's are likely to vary rapidly, a variable-step solver is often faster.

### Variable-Step Solvers

When you select `Variable-step` as the solver **Type**, you can choose any of the following as the **Solver**:

- `discrete (no continuous states)`
- `ode45 (Dormand-Prince)`
- `ode23 (Bogacki-Shampine)`
- `ode113 (Adams)`
- `ode15s (stiff/NDF)`

- `ode23s (stiff/Mod.  Rosenbrock)`
- `ode23t (Mod.  stiff/Trapezoidal)`
- `ode23tb (stiff/TR-BDF2)`

See the Simulink documentation for information on these solvers.

## Variable-Step Solver Options

When you select `Variable-step` as the Simulink solver **Type**, you can also set several other parameters that affect the step size of the simulation:

- **Maximum step size**: The largest step size solver can use during a simulation
- **Minimum step size**: The smallest step size solver can use during a simulation
- **Initial step size**: The step size solver uses to begin the simulation
- **Relative tolerance**: The largest allowable relative error at any step in the simulation
- **Absolute tolerance**: The largest allowable absolute error at any step in the simulation
- **Zero crossing control**: Set to `on` for the solver to compute exactly where the signal crosses the *x*-axis. This is useful when using functions that are nonsmooth and the output depends on when a signal crosses the *x*-axis, such as absolute values.

By default, the values for these options are automatically chosen. To choose your own values, enter them in the appropriate fields. For more information on these options, and the circumstances in which to use them, see the Simulink documentation.

## Fixed-Step Solvers

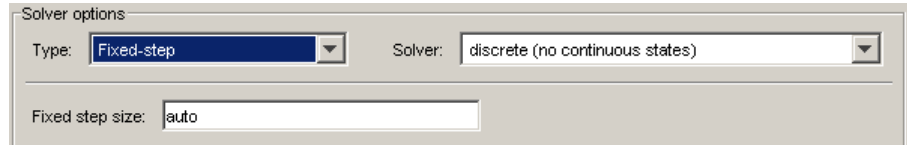
When you select `Fixed-step` as the solver **Type**, you can choose any of the following as the **Solver**:

- `discrete (no continuous states)`

- `ode5 (Dormand-Prince)`
- `ode4 (Runge-Kutta)`
- `ode3 (Bogacki-Shanpine)`
- `ode2 (Heun)`
- `ode1 (Euler)`

See the Simulink documentation for information on these solvers.

When you select `Fixed-step` as the solver **Type**, you can also set **Fixed step size**, which determines the step size the solver uses during the simulation. By default, Simulink automatically chooses a value for this option.



# Speeding Up Response Optimization Using Parallel Computing

**In this section...**



## When to Use Parallel Computing for Response Optimization

You can use Simulink Response Optimization software with Parallel Computing Toolbox™ software to speed up the time-domain response optimization of a Simulink model. Using parallel computing may reduce your model's optimization time in the following cases:

- The model contains a large number of tuned parameters, and the `Gradient descent` algorithm is selected for optimization.
- The `Pattern search` algorithm is selected for optimization.
- The model contains a large number of uncertain parameters and uncertain parameter values.
- The model is complex and takes a long time to simulate.

When you use parallel computing, Simulink Response Optimization software distributes independent simulations to run them in parallel on multiple MATLAB sessions, also known as *workers*. Distributing the simulations significantly reduces the optimization time because the time required to simulate the model dominates the total optimization time. For more information on how the software distributes the simulations and the expected

speedup, see “Understanding Parallel Computing in Simulink® Response Optimization Software” on page 1-45.

The following sections describe how to configure your system, and use parallel computing:



## Understanding Parallel Computing in Simulink Response Optimization Software

You can enable parallel computing with the `Gradient descent` and `Pattern search` optimization algorithms in the Simulink Response Optimization software. When you enable parallel computing, Simulink Response Optimization software distributes independent simulations during optimization on multiple MATLAB sessions. The following sections describe which simulations are distributed and the expected speedup using parallel computing:



### Parallel Computing with the `Gradient descent` Algorithm

When you select `Gradient descent` as the optimization algorithm, the model is simulated during the following computations:

- Constraint and objective value computation — One simulation per iteration
- Constraint and objective gradient computations — Two simulations for every tuned parameter per iteration
- Line search computations — Multiple simulations per iteration

The total time, $T_{total}$, taken per iteration to perform these simulations is given by the following equation:

$$T_{total} = T + (N_p \times (2 \times T)) + (N_{ls} \times T) = T \times (1 + (2 \times N_p) + N_{ls})$$

where $T$ is the time taken to simulate the model and is assumed to be equal for all simulations, $N_p$ is the number of tuned parameters, and $N_{ls}$ is the number of line searches.

When you use parallel computing, Simulink Response Optimization software distributes the simulations required for constraint and objective gradient computations. The simulation time taken per iteration when the gradient computations are performed in parallel, $T_{totalP}$, is approximately given by the following equation:

$$T_{totalP} = T + \left(ceil\left(\frac{N_p}{N_w}\right) \times 2 \times T\right) + (N_{ls} \times T) = T \times \left(1 + 2 \times ceil\left(\frac{N_p}{N_w}\right) + N_{ls}\right)$$

where $N_w$ is the number of MATLAB workers.

---

**Note** The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

---

The expected speedup for the total optimization time is given by the following equation:

$$\frac{T_{totalP}}{T_{total}} = \frac{1 + 2 \times ceil\left(\frac{N_p}{N_w}\right) + N_{ls}}{1 + (2 \times N_p) + N_{ls}}$$

For example, for a model with $N_p$=3, $N_w$=4, and $N_{ls}$=3, the expected speedup equals $\frac{1 + 2 \times ceil\left(\frac{3}{4}\right) + 3}{1 + (2 \times 3) + 3} = 0.6$.

For a demo on the performance improvement achieved with the `Gradient descent` algorithm, see Improving Optimization Performance Using Parallel Computing in the **Demos** tab.

## Parallel Computing with the `Pattern search` Algorithm

The `Pattern search` optimization algorithm uses search and poll sets to create and compute a set of candidate solutions at each optimization iteration.

The total time, $T_{total}$, taken per iteration to perform these simulations, is given by the following equation:

$$T_{total} = (T \times N_p \times N_{ss}) + (T \times N_p \times N_{ps}) = T \times N_p \times (N_{ss} + N_{ps})$$

where $T$ is the time taken to simulate the model and is assumed to be equal for all simulations, $N_p$ is the number of tuned parameters, $N_{ss}$ is a factor for the search set size, and $N_{ps}$ is a factor for the poll set size.

When you use parallel computing, Simulink Response Optimization software distributes the simulations required for the search and poll set computations, which are evaluated in separate parfor loops. The simulation time taken per iteration when the search and poll sets are computed in parallel, $T_{totalP}$, is given by the following equation:

$$\begin{aligned} T_{totalP} &= (T \times ceil(N_p \times \frac{N_{ss}}{N_w})) + (T \times ceil(N_p \times \frac{N_{ps}}{N_w})) \\ &= T \times (ceil(N_p \times \frac{N_{ss}}{N_w}) + ceil(N_p \times \frac{N_{ps}}{N_w})) \end{aligned}$$

where $N_w$ is the number of MATLAB workers.

---

**Note** The equation does not include the time overheads associated with configuring the system for parallel computing and loading Simulink software on the remote MATLAB workers.

---

The expected speed up for the total optimization time is given by the following equation:

$$\frac{T_{totalP}}{T_{total}} = \frac{ceil(N_p \times \frac{N_{ss}}{N_w}) + ceil(N_p \times \frac{N_{ps}}{N_w})}{N_p \times (N_{ss} + N_{ps})}$$

For example, for a model with $N_p$=3, $N_w$=4, $N_{ss}$=15, and $N_{ps}$=2, the expected speedup equals $\frac{ceil(3 \times \frac{15}{4}) + ceil(3 \times \frac{2}{4})}{3 \times (15 + 2)} = 0.27$ .

---

**Note** Using the `Pattern search` algorithm with parallel computing may not speed up the optimization time. To learn more, see "Why do I not see the optimization speedup I expected using parallel computing?" in the "Troubleshooting" section.

---

For a demo on the performance improvement achieved with the `Pattern search` algorithm, see Improving Optimization Performance Using Parallel Computing in the **Demos** tab.

## Configuring Your System for Parallel Computing

To use parallel computing, you must first configure your system as described in the following sections:

- "Configuring Parallel Computing on Multicore Processors" on page 1-48
- "Configuring Parallel Computing on Multiprocessor Networks" on page 1-49

After you configure your system for parallel computing, you can use the GUI or the command-line functions to optimize the model's response using parallel computing.

### Configuring Parallel Computing on Multicore Processors

With a basic Parallel Computing Toolbox license, you can establish a pool of up to four parallel MATLAB sessions in addition to the MATLAB client.

To start a pool of four MATLAB sessions in local configuration, type the following at the MATLAB prompt:

```
matlabpool open local
```

To learn more, see the matlabpool reference page in the Parallel Computing Toolbox documentation.

### Configuring Parallel Computing on Multiprocessor Networks

To use parallel computing on a multiprocessor network, you must have the Parallel Computing Toolbox software and the MATLAB® Distributed Computing Server™ software. To learn more, see the Parallel Computing Toolbox and MATLAB Distributed Computing Server documentation.

To configure a multiprocessor network for parallel computing:

1. Create a user configuration file to include any model file dependencies, as described in "Defining Configurations" and FileDependencies reference page in the Parallel Computing Toolbox documentation.

2. Open the pool of MATLAB workers using the user configuration file, as described in "Applying Configurations in Client Code" in the Parallel Computing Toolbox documentation.

   Opening the pool allows the remote workers to access the file dependencies included in the user configuration file.

## Checking Model Dependencies

Model dependencies are files, such as referenced models, data files and S-functions, without which a model cannot run. When you use parallel computing, Simulink Response Optimization software helps you identify model path dependencies. To do so, the software uses the Simulink Manifest Tools. The dependency analysis may not find all the files required by your model. To learn more, see the "Dependency Analysis" in the Simulink documentation.

If your model has dependencies that the software cannot detect automatically, you must add the dependencies before you start the optimization using parallel computing:

1. Add the path dependencies using the GUI or at the command line, as described "How to Use Parallel Computing in the GUI" on page 1-50, and "How to Use Parallel Computing at the Command Line" on page 1-54.

2. Add the file dependencies, as described in "Configuring Parallel Computing on Multiprocessor Networks" on page 1-49.

When you use parallel computing, verify that the remote MATLAB workers can access all the model dependencies.

---

**Note** The optimization errors out if all the remote workers cannot access all the model dependencies.

---

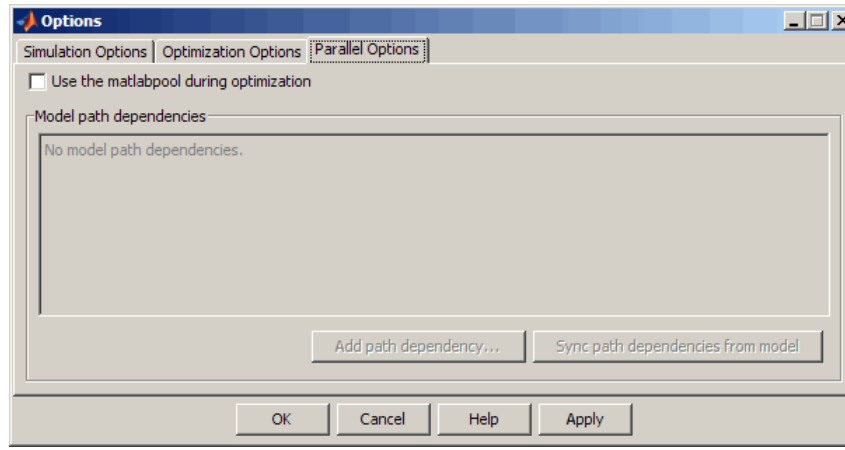
## How to Use Parallel Computing in the GUI

After you configure your system for parallel computing, as described in "Configuring Your System for Parallel Computing" on page 1-48, you can use the GUI to optimize your model's response:

1. Open the model that you want to optimize.

2. Configure the response optimization project for your model, as described in "Simple Control Design Example" in the *Simulink Response Optimization Getting Started Guide*.

**3** In the Signal Constraint block, select **Optimization > Parallel Options** to open the **Parallel Options** tab.



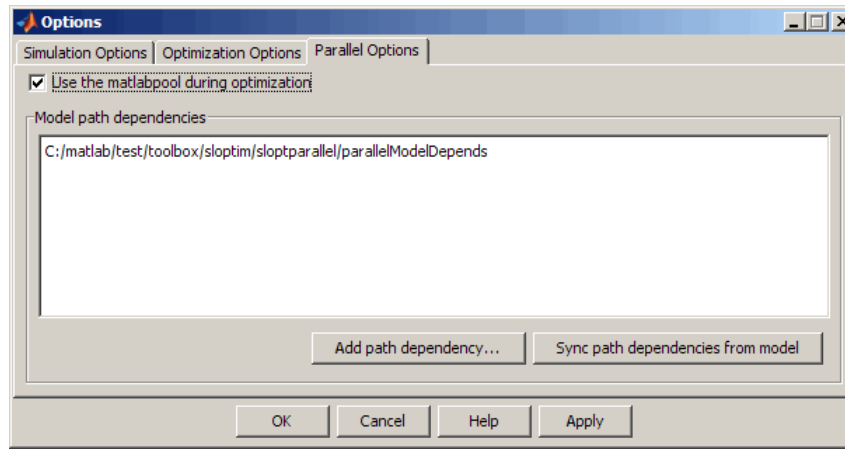
4 Select the **Use the matlabpool during optimization** option.

This action checks for model path dependencies in your Simulink model and displays the path dependencies in the **Model path dependencies** list box.

---

**Note** As described in "Checking Model Dependencies" on page 1-49, the automatic path dependencies check may not detect all the path dependencies in your model.

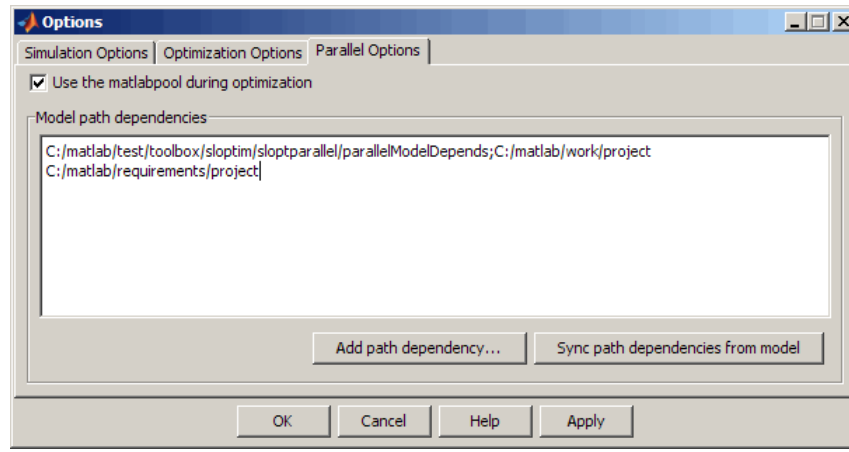
---



5 (Optional) Add the path dependencies that the automatic check does not detect to the response optimization project.

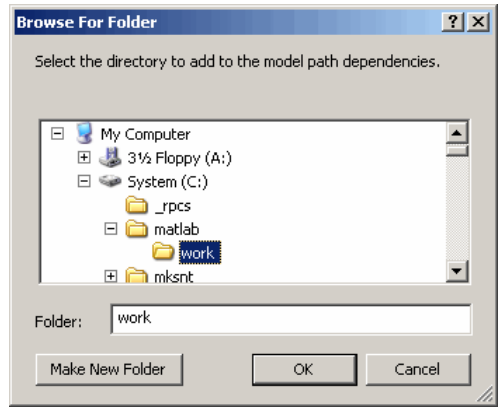
a Specify the paths in the **Model path dependencies** list box.

You can specify the paths separated with a semicolon, or on a new line.




b Click **Apply** to include the new paths in the response optimization project.

Alternatively, you can click **Add path dependency** to open a Browse For Folder dialog box where you can select the directory to add.



6. (Optional) If you modify the Simulink model such that it introduces a new path dependency, then you must resync the path dependencies. Click **Sync path dependencies from model** in the **Parallel Options** tab to rerun the automatic dependency check for your model.

   This action updates the **Model path dependencies** list box with any new path dependency found in the model.

7. Click **OK**.

8. In the Signal Constraint block window, select **Optimization > Start** to optimize the model response using parallel computing.

For more information on how to troubleshoot problems that occur during optimization using parallel computing, see the "Questions" in the "Troubleshooting" section.

## How to Use Parallel Computing at the Command Line

After you configure your system for parallel computing, as described in "Configuring Your System for Parallel Computing" on page 1-48, you can optimize your model's response using the command-line functions:

1. Open the model that you want to optimize.

2. Configure a response optimization project, `proj`, as described in "Control Design Example Using Functions" in the *Simulink Response Optimization Getting Started Guide*.

3. Enable the parallel computing option in the response optimization project by typing the following command:

   ```
   optimset(proj,'UseParallel','always');
   ```

4. Find the model path dependencies by typing the `finddepend` command.

   ```
   dirs=finddepend(proj)
   ```

   This command returns the model path dependencies in your Simulink model.

**Note** As described in "Checking Model Dependencies" on page 1-49, the `finddepend` command may not detect all the path dependencies in your model.

5. (Optional) Modify `dirs` to include the model path dependencies that `finddepend` does not detect.

```
dirs=vertcat(dirs,'\\hostname\C$\matlab\work)
```

6. Add the updated path dependencies to the response optimization project, `proj` by typing the following command:

```
optimset(proj,'ParallelPathDependencies',dirs)
```

7. Run the optimization by typing the following command:

```
optimize(proj)
```

For more information on how to troubleshoot problems that occur during optimization using parallel computing, see the "Questions" in the "Troubleshooting" section.

# Accelerating Model Simulations During the Optimization

**In this section...**



## About Accelerating Optimization

You can accelerate the response optimization computations by changing the simulation mode of your Simulink model. Simulink Response Optimization software supports `Normal` and `Accelerator` simulation modes. For more information about these modes, see "Accelerating Models" in the Simulink documentation.

The default simulation mode is `Normal`. In this mode, Simulink uses interpreted code, rather than compiled C code during simulations.

In the `Accelerator` mode, Simulink Response Optimization software runs simulations during optimization with compiled C code. Using compiled C code speeds up the simulations and reduces the time to optimize the model response signals.

## Limitations

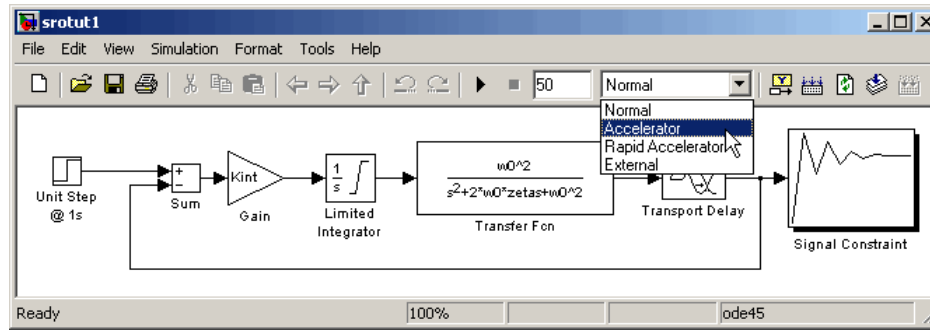
You cannot use the `Accelerator` mode if your model contains algebraic loops. If the model contains MATLAB function blocks, you must either remove them or replace them with Fcn blocks.

If the model structure changes during optimization, the model is compiled to regenerate the C code for each iteration. Using the `Accelerator` mode increases the computation time. To learn more about code regeneration, see "Code Regeneration in Accelerated Models" in the Simulink documentation.

## Setting Accelerator Mode for Response Optimization

To set the simulation mode to `Accelerator`, open the Simulink model window and perform one of the following actions:

- Select **Simulation > Accelerator**.
- Choose `Accelerator` from the drop-down list as shown in the next figure.




**Tip** To obtain the maximum performance from the `Accelerator` mode, close all Scope blocks in your model.

# Response Plots Property Editor

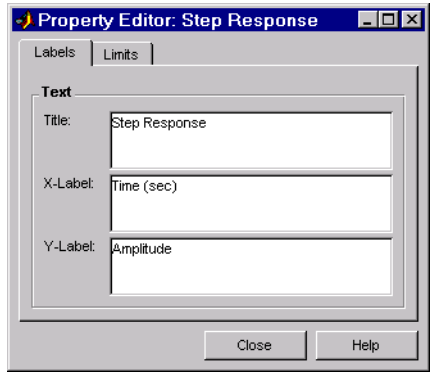
| In this section... |
| --- |
|  |
|  |
|  |

## Modifying Properties of Response Plots

This section discusses how you can change the properties of response plots.

**Note** Click the tabs to get help on panes in the Property Editor.

This figure shows the Property Editor dialog box for a step response.




In general, you can change the following properties of response plots.

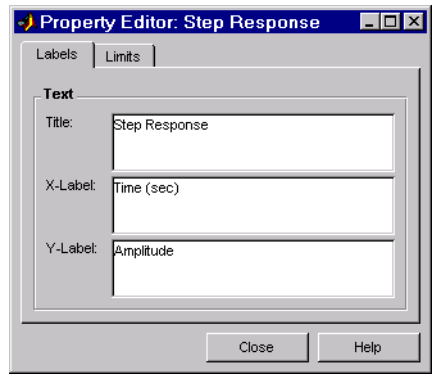
- **Labels** – Titles and X- and Y-labels
- **Limits** – Numerical ranges of the *x*- and *y*- axes

As you make changes in the Property Editor, they display immediately in the response plot. Conversely, if you make changes in a plot using right-click

menus, the Property Editor for that plot automatically updates. The Property Editor and its associated plot are dynamically linked.

## Labels Pane

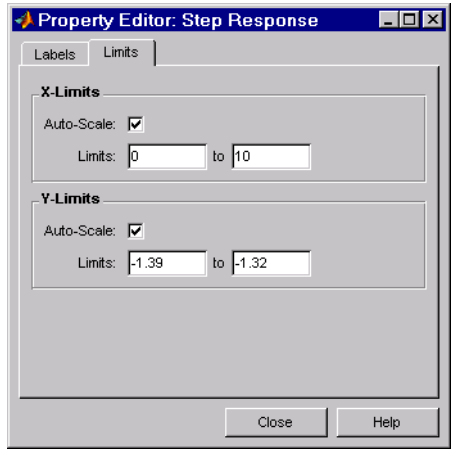
**Note** Click the tabs below to get help on the Property Editor.




To specify new text for plot titles and axis labels, type the new string in the field next to the label you want to change. The label changes immediately as you type, so you can see how the new text looks as you are typing.

## Limits Pane

**Note** Click the tabs to get help on the Property Editor.




Default values for the axes limits make sure that the maximum and minimum $x$ and $y$ values are displayed. If you want to override the default settings, change the values in the **Limits** pane fields. The **Auto-Scale** check box automatically clears if you click a different field. The new limits appear immediately in the response plot.

To reestablish the default values, select the **Auto-Scale** check box again.

# Including Independent Parameters

**In this section...**



## Basic Steps for Tuning Independent Parameters

Sometimes parameters in your model depend on independent parameters that do not appear in the model. The following steps give an overview of how to tune and include uncertainty in these independent parameters. example follows in the next section:

1. Add the independent parameters to the model workspace (along with initial values).
2. Define a Simulation Start function that runs before each simulation of the model. This Simulation Start function defines the relationship between the dependent parameters in the model and the independent parameters in the model workspace.
3. The independent parameters now appear in the Add Parameters dialog box when you select **Tuned parameters** or **Uncertain parameters**. Add these parameters to the list of tuned parameters to tune them during the response optimization.

---

**Caution** Avoid adding independent parameters together with their corresponding dependent parameters to the lists of tuned and uncertain parameters. Otherwise, the optimization could give incorrect results. For example, when a parameter `x` depends on the parameters `a` and `b`, avoid adding all three parameters to the lists of tuned and uncertain parameters.

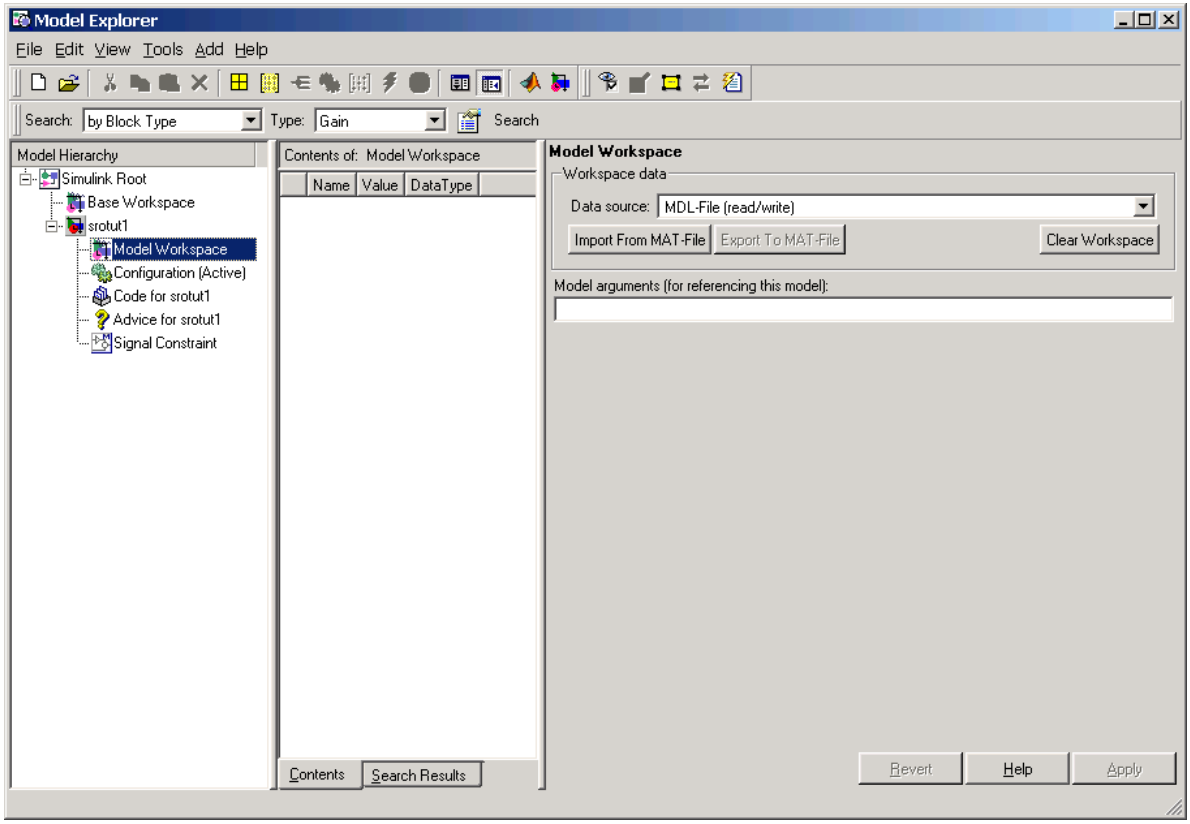
---

## Example

Assume that the parameter `Kint` in the model `srotut1` is related to the parameters `x` and `y` according to the relationship `Kint=x+y`. Also assume that the initial values of `x` and `y` are 1 and -0.7, respectively. To tune `x` and

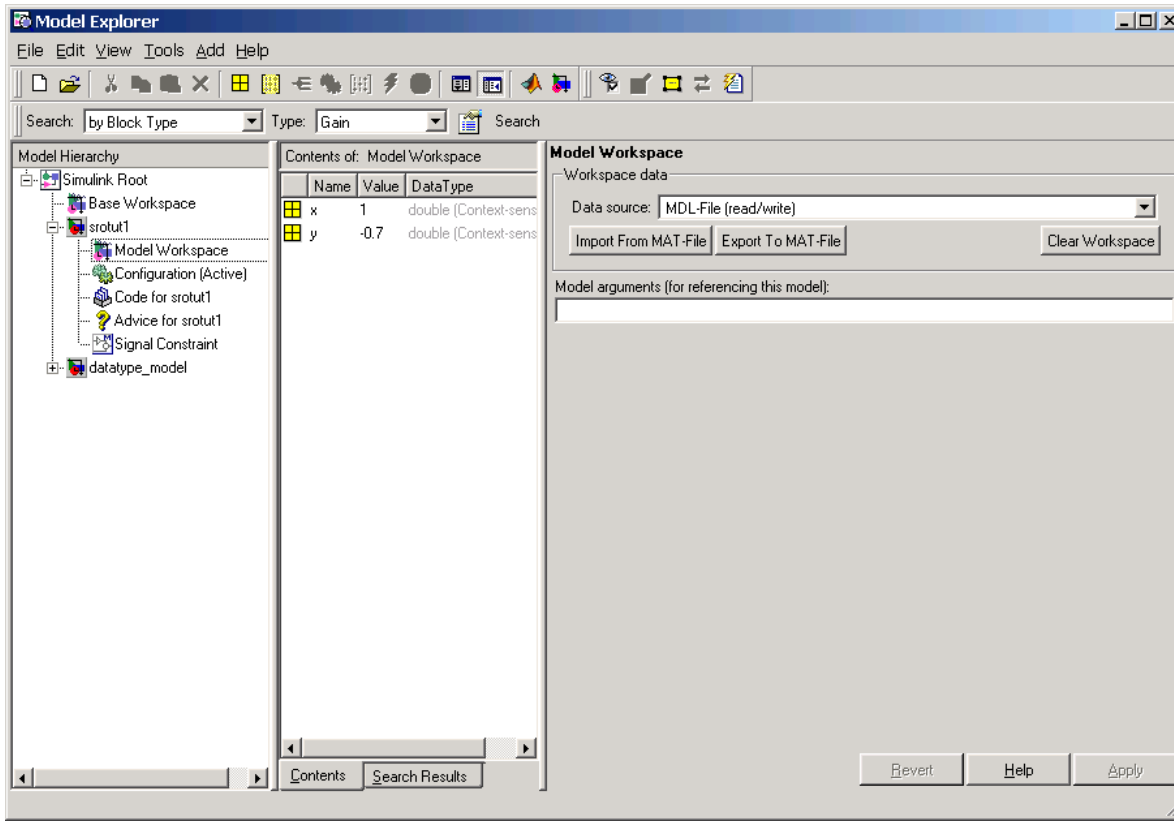
y instead of `Kint`, first define these parameters in the model workspace. To do this,

1. Select **View > Model Explorer** from the `srotut1` window.
2. Select **Model Workspace** under the **srotut1** node in the tree browser within the Model Explorer window.




3. Select **Add > MATLAB Variable** within the Model Explorer to add a new variable to the model workspace. A new variable appears within the pane labeled **Contents of: Model Workspace**. Change the variable name to `x` and the initial value to `1`.

**4** Repeat step 3 to add a variable `y` with an initial value of `-0.7`. The Model Explorer window should now look like the following figure.




**5** To add the Simulation Start function defining the relationship between `Kint` and the independent parameters `x` and `y`, select **File > Model Properties** in the `srotut1` window, and then select **Callbacks** in the Model Properties dialog box.

**6** Under **Simulation start function**, enter the name of a new M-file, for example, `srotut1_start`.

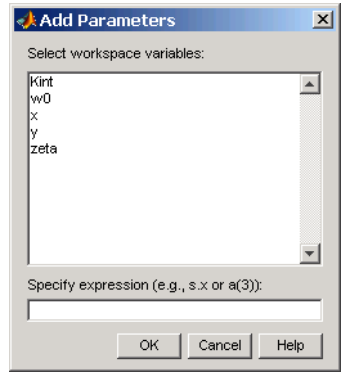
**7** Create a new M-file with this name. The contents of the M-file should define the relationship between the parameters in the model and the

parameters in the workspace. For this example, the M-file should look something like the following:

```
wks = get_param(gcs, 'ModelWorkspace')
x = wks.evalin('x')
y = wks.evalin('y')
Kint = x+y;
```

**Note** You must first use the `get_param` function to get the variables `x` and `y` from the model workspace before you can use them to define `Kint`.

8 When you add a new tuned or uncertain parameter, `x` and `y` should now appear in the Add Parameters dialog box.



2

# Response Optimization in a SISO Design Task

# Response Optimization in a SISO Design Task

**In this section...**



## Tuning within a SISO Design Task

When you have Control System Toolbox software installed, you can design compensators for control systems by tuning compensator elements or parameters within a SISO Design Task in the Control and Estimation Tools Manager. You can tune any elements or parameters, such as poles, zeros, and gains, within any compensators in the system to optimize the responses of both open and closed loops.

Optimize the responses of systems in the SISO Design Task to meet both time- and frequency-domain performance requirements by graphically constraining signals:

- Add frequency-domain design requirements to plots such as root-locus, Nichols, and Bode in the SISO Design Task graphical tuning editor called SISO Design Tool.
- Add time-domain design requirements to plots such as step or impulse response (when displayed within the LTI Viewer as part of a SISO Design Task).

You can use response optimization within a SISO Design Task in the Control and Estimation Tools Manager to tune both command-line LTI models as well as Simulink models:

- Create an LTI model using the Control System Toolbox command-line functions and use the `sisotool` function to create a SISO Design Task for the model.
- Use a Simulink Compensator Design task (from Simulink® Control Design™ software) to automatically analyze the model and then create a SISO Design Task for a linearized version of the model. You can then use

the response optimization tools within the SISO Design Task to tune the response of the linearized Simulink model.

When using response optimization within a SISO Design Task you cannot add uncertainty to system parameters.

Simulink Response Optimization Getting Started Guide provides detailed instructions for optimizing response in a SISO Design Task. A detailed example guides you through the following process:

1. Create and import a linear model into a SISO Design Task. You can either create an LTI model at the MATLAB command line, or use a Simulink Compensator Design Task from Simulink Control Design software to automatically linearize the system and create a SISO Design Task. See "Creating an LTI Plant Model" for more information.

2. Under **Automated Tuning** select `Optimization based tuning` as the **Design Method** and then click the **Optimize Compensators** button to create a **Response Optimization** task within the Control and Estimation Tools Manager. See "Creating a Response Optimization Task" for more information.

3. Within the **Response Optimization** node, select the **Compensators** pane to select and configure the compensator elements you want to tune during the response optimization. See "Selecting Tunable Compensator Elements" for more information.

4. Under **Design requirements** in the **Response Optimization** node, select the design requirements you want the system to satisfy. See "Adding Design Requirements" for more information.

5. Click the **Start Optimization** button within the **Response Optimization** node. The optimization progress results appear under **Optimization**. The **Compensators** pane contains the new, optimized compensator element values. See "Optimizing the System's Response" for more information.

## Supported SISO Tool Requirements

This section lists the SISO Tool requirements that can be optimized using Simulink Response Optimization software.

## Root Locus Diagrams

**Settling Time.** If you specify a settling time in the continuous-time root locus, a vertical line appears on the root locus plot at the pole locations associated with the value provided (using a first-order approximation). In the discrete-time case, the constraint is a curved line.

It is required that $\mathrm{Re}\{pole\} < -4.6/T_{settling}$ for continuous systems and $\log(\mathrm{abs}(pole))/T_{discrete} < -4.6/T_{settling}$ for discrete systems. This is an approximation of the settling time based on second order dominant systems.

**Percent Overshoot.** Specifying percent overshoot in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the percent value (using a second-order approximation). In the discrete-time case, the constraint appears as two curves originating at (1,0) and meeting on the real axis in the left-hand plane.

The percent overshoot ( $p.o.$ ) constraint can be expressed in terms of the damping ratio, as in this equation:

$$p.o. = 100e^{-\pi\zeta/\sqrt{1-\zeta^2}}$$

where $\zeta$ is the damping ratio.

**Damping Ratio.** Specifying a damping ratio in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the damping ratio. In the discrete-time case, the constraint appears as curved lines originating at (1,0) and meeting on the real axis in the left-hand plane.

The damping ratio defines a requirement on $-\mathrm{Re}\{pole\}/\mathrm{abs}(pole)$ for continuous systems and on

$$r = \text{abs}(pSys)$$
$$t = \text{angle}(pSys)$$
$$c = -\log(r) / \sqrt{(\log(r))^2 + t^2}$$

for discrete systems.

**Natural Frequency.** If you specify a natural frequency, a semicircle centered around the root locus origin appears. The radius equals the natural frequency.

The natural frequency defines a requirement on $\text{abs}(pole)$ for continuous systems and on

$$r = \text{abs}(pSys)$$
$$t = \text{angle}(pSys)$$
$$c = \sqrt{(\log(r))^2 + t^2} / Ts_{model}$$

for discrete systems.

**Region Constraint.** Specifies an exclusion region in the complex plane, causing a line to appear between the two specified points with a shaded region below the line. The poles must not lie in the shaded region.

## Open-Loop and Prefilter Bode Diagrams

**Gain and Phase Margins.** Specify a minimum phase and or a minimum gain margin.

**Upper Gain Limit.** You can specify an upper gain limit, which appears as a straight line on the Bode magnitude curve. You must select frequency limits, the upper gain limit in decibels, and the slope in dB/decade.

**Lower Gain Limit.** Specify the lower gain limit in the same fashion as the upper gain limit.

## Open-Loop Nichols Plots

**Phase Margin.** Specify a minimum phase amount.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to phase margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

**Gain Margin.** Specify a minimum gain margin.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to gain margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

**Closed-Loop Peak Gain.** Specify a peak closed-loop gain at a given location. The specified value can be positive or negative in dB. The constraint follows the curves of the Nichols plot grid, so it is recommended that you have the grid on when using this feature.

While displayed graphically at only one location around a multiple of -180 degrees, this requirement applies to gain margin regardless of actual phase (i.e., it is interpreted for all multiples of -180).

**Gain-Phase Requirement.** Specifies an exclusion region for the response on the Nichols plot. The response must not pass through the shaded region.

This only applies to the region (phase and gain) drawn.

## Step/Impulse Response Plots

**Upper Time Response Bound.** You can specify an upper time response bound.

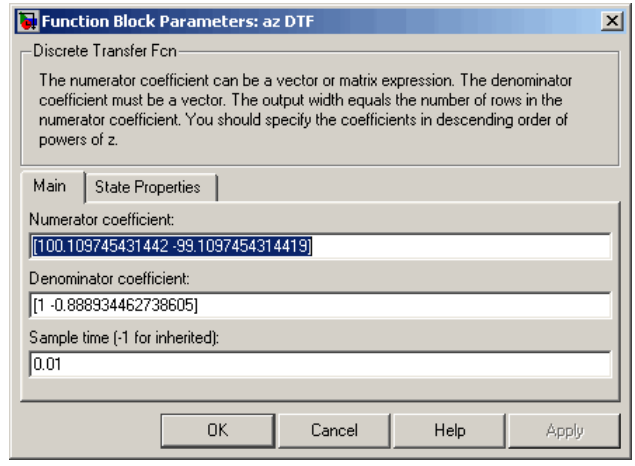
**Lower Time Response Bound.** You can specify a lower time response bound.

# Response Optimization Using Simulink Control Design Software

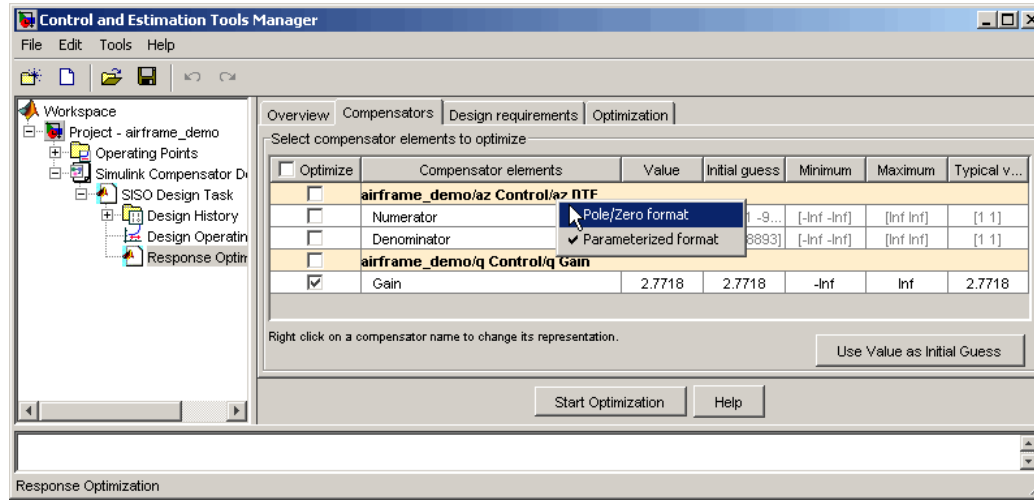
Users with Control System Toolbox and Simulink Control Design software can perform frequency domain based optimization of Simulink models.

You can use Simulink Control Design software to configure SISO Tool with compensators, inputs, outputs, and loops computed from a Simulink model (see “Designing Compensators” in Simulink Control Design User’s Guide). Once SISO Tool is configured with data from a Simulink model, you can use Simulink Response Optimization software to optimize SISO Tool based models.

There is only one difference when tuning compensators derived from Simulink Control Design software: The tuning of compensators from a Simulink model is done through the masks of the Simulink blocks representing each compensator. When selecting parameters to optimize, users can tune the compensator in the pole, zero, or gain format, or in a format consistent with the Simulink block mask as shown in the following figure. Changing the compensator format is not possible when optimizing pure SISO Tool models (those not derived using Simulink Control Design software).




**Mask of a Simulink® compensator block**




**Response optimization compensators pane**

To walk through a Simulink Response Optimization example using Simulink Control Design software, and the SISO Tool, see the tutorial “Enforcing Time and Frequency Requirements on a Single Loop Controller Design ” and Simulink Response Optimization demo, `airframe_demo` (both in the Demos pane of the Help browser), which is based on a SISO Tool session started using Simulink Control Design software.

3

# Function Reference

# Response Optimization Projects

| | |
|---|---|
| `finddepend` | Find model path dependencies |
| `getsro` | Response optimization project for given Simulink model |
| `ncdupdate` | Upgrade models with old Nonlinear Control Design Blockset blocks |
| `newsro` | Create default Simulink Response Optimization project |
| `optimize` | Run response optimization project |

# Constraints and Parameters

| | |
|---|---|
| `findconstr` | Find constraints on given response |
| `findpar` | Find specifications for given tuned parameter |
| `gridunc` | Construct N-D grid of uncertain parameter values |
| `initpar` | Initialize tuned parameters |
| `randunc` | Randomly sample uncertain parameters |
| `setunc` | Specify parameter uncertainty in optimization project |

# Optimization and Simulation Settings

| | |
|---|---|
| `optimget` | Current optimizer settings |
| `optimset` | Modify optimization settings |
| `simget` | Current simulation settings |
| `simset` | Modify simulation settings |

4

# Functions — Alphabetical List

# findconstr

## Purpose

Find constraints on given response

## Syntax

```
constraints=findconstr(proj,'blockname')
```

## Description

`constraints=findconstr(proj,'blockname')` returns a constraint object for the Signal Constraint block, `blockname`, within the response optimization project, `proj`. This object contains all the data defining the desired response, including the positions of the constraint bound segments as well as the reference signals. The constraints are used in a response optimization project to define the region in which the response signal must lie.

Modify the constraint object properties `UpperBoundX`, `UpperBoundY`, `LowerBoundX`, and `LowerBoundY` to specify new constraint bound segments on the signals. These properties define the *x* and *y* values for the beginning and ending points of each constraint segment.

Modify the constraint object properties `ReferenceX` and `ReferenceY` to specify a new reference signal to track. These properties contain the vectors of *x* and *y* data defining the reference signal.

## Example

Open the model `srotut1` by typing

```
srotut1
```

Create a response optimization project.

```
proj=newsro('srotut1','Kint');
```

Find the constraints object for this project.

```
constraint=findconstr(proj,'srotut1/Signal Constraint')
```

This returns

```
          ConstrEnable: 'on'
            isFeasible: 1
            CostEnable: 'off'
                Enable: 'on'
```

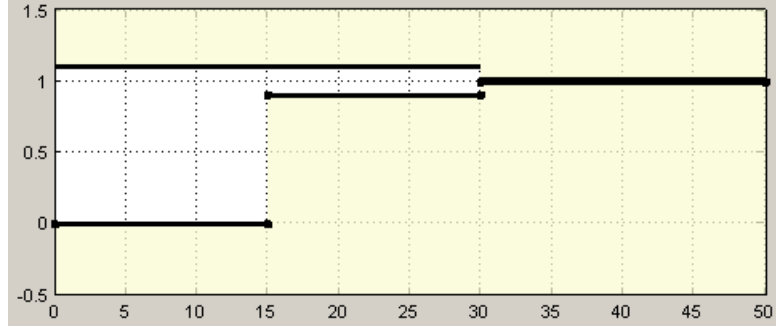
```
                Name: 'Signal Constraint'
          SignalSize: [1 1]
         LowerBoundX: [3x2 double]
         LowerBoundY: [3x2 double]
    LowerBoundWeight: [3x1 double]
         UpperBoundX: [2x2 double]
         UpperBoundY: [2x2 double]
    UpperBoundWeight: [2x1 double]
          ReferenceX: []
          ReferenceY: []
     ReferenceWeight: []

  Signal Constraint.
```

Change the positioning of the constraint bounds by editing the upper and lower bound matrices.

```
constraint.UpperBoundY=[1.1 1.1;1.01 1.01];
constraint.LowerBoundY=[0 0;0.9 0.9;0.99 0.99];
constraint.UpperBoundX=[0 30;30 50];
constraint.LowerBoundX=[0 15;15 30;30 50];
```

These bounds define the constraints given in the following figure.

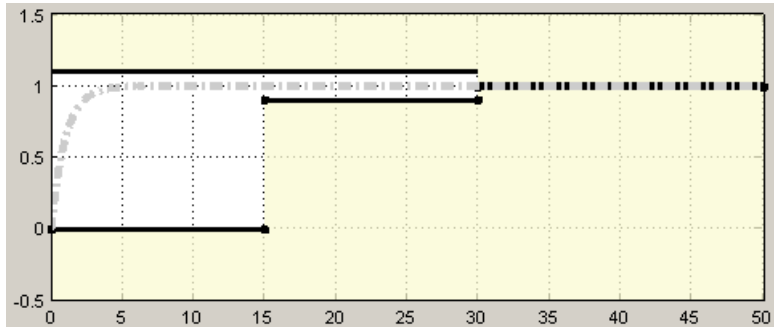



Include a reference signal with the following commands:

```
constraint.ReferenceX=linspace(0,50,1000);
```

```
constraint.ReferenceY=1-exp(-linspace(0,50,1000));
```

This defines the reference signal shown in the following figure.




## See Also

`getsro`, `newsro`, `optimize`

## Purpose

Find model path dependencies

## Syntax

```
dirs=finddepend(proj)
```

## Description

`dirs=finddepend(proj)` returns a cell array of directories that contain model dependencies. The directories or model path dependencies are required to optimize the model response using parallel computing.

---

**Note** `finddepend` may not detect all path dependencies for your model.

---

`finddepend` returns an empty cell array in the following cases:

- The model does not have any dependencies.
- The model has dependencies that cannot be detected.

  For more information, see "Analysis Limitations" in the *Simulink User's Guide*.

You must modify `dirs` to include additional path dependencies for your model:

- Paths that `finddepend` cannot detect

  For example, directories that contain M-code for your model or block callback

- Paths detected by `finddepend` that the workers cannot access directly

  For example, path dependencies on your local drive

  For more information on modifying `dirs`, see "How to Use Parallel Computing at the Command Line" on page 1-54 in the *Simulink Response Optimization User's Guide*.

Use the `optimset` command to add the model path dependencies to the response optimization project.

## Example

Open the `pidtune_demo` model.

```
pidtune_demo
```

Extract the response optimization project from this model.

```
proj=getsro('pidtune_demo');
```

Enable the parallel computing option in the response optimization project.

```
optimset(proj,'UseParallel','always');
```

Find the model path dependencies.

```
dirs=finddepend(proj)
```

This command returns an empty cell array because the `pidtune_demo` model does not have any path dependencies.

To add model path dependencies, use the `optimset` command.

```
optimset(proj,'ParallelPathDependencies',dirs)
```

## See Also

"Model Dependencies" in the Simulink documentation, `optimget`, `optimset`

## Purpose

Find specifications for given tuned parameter

## Syntax

```
p=findpar(proj,'param')
```

## Description

`p=findpar(proj,'param')` returns a tuned parameters object for the parameter with the name `param` within the response optimization project, `proj`. The tuned parameters object defines specifications for each tuned parameter that the response optimization algorithm uses, such as initial guesses, lower bounds, etc.

The properties of each tuned parameter object are

| | |
|---|---|
| `Name` | A string giving the parameter's name. |
| `Value` | The current value of the parameter. This changes during the optimization. |
| `InitialGuess` | The initial guess for the parameter value for the optimization. |
| `Minimum` | The minimum value this parameter can take. By default, it is set to `-Inf`. |
| `Maximum` | The maximum value this parameter can take. By default, it is set to `Inf`. |
| `TypicalValue` | A value that the tuned parameter is scaled by during the optimization. |
| `ReferencedBy` | The block, or blocks, in which the parameter appears. |
| `Description` | An optional string giving a description of the parameter. |
| `Tuned` | Set to `1` or `0` to indicate if this parameter is to be tuned or not. |

Edit these properties to specify additional information about your parameters.

## Example

Create a response optimization project for `srotut1`.

```
proj=newsro('srotut1','Kint');
```

Find the tuned parameters object for the parameter `Kint`.

```
p=findpar(proj,'Kint')
```

This returns

```
              Name: 'Kint'
             Value: 0
      InitialGuess: 0
           Minimum: -Inf
           Maximum: Inf
      TypicalValue: 0
      ReferencedBy: {0x1 cell}
       Description: ''
             Tuned: 1

  Tuned parameter.
```

Change the initial guess to `0.5`, and the minimum value to `0` with the `set` function.

```
set(p,'InitialGuess',0.5,'Minimum',0)
```

## See Also

`getsro`, `newsro`, `optimize`

## Purpose

Response optimization project for given Simulink model

## Syntax

```
proj=getsro('modelname')
```

## Description

`proj=getsro('modelname')` returns the response optimization project, `proj`, currently associated with the Simulink model with name, `modelname`. The model should be open and contain Simulink Response Optimization blocks. Use the project with the `optimize` function to optimize response signals in the model by tuning specified parameters.

## Example

Open the model `pidtune_demo` by typing

```
pidtune_demo
```

Extract the response optimization project from this model

```
proj=getsro('pidtune_demo')
```

This returns

```
              Name: 'pidtune_demo'
        Parameters: [3x1 ResponseOptimizer.Parameter]
      OptimOptions: [1x1 ResponseOptimizer.OptimOptions]
             Tests: [1x1 ResponseOptimizer.SimTest]
             Model: 'pidtune_demo'

Simulink Response Optimization Project.
```

Use the `findpar` and `findconstr` functions to specify signal constraints and tuned parameters.

## See Also

`findconstr`, `findpar`, `newsro`, `optimize`

# gridunc

## Purpose

Construct N-D grid of uncertain parameter values

## Syntax

```
uset=gridunc('P1',Values1,'P2',Values2,...)
```

## Description

`uset=gridunc('P1',Values1,'P2',Values2,...)` takes vectors (for scalar-valued parameters) or cell arrays of values `Values1`, `Values2`,... for the uncertain parameters `P1`, `P2`,... and constructs `uset`, an object containing a multidimensional grid of all parameter value combinations.

Optimize the responses based on uncertain parameter values by setting the `Optimized` property of the uncertain parameter object, `uset`, to `true`. (By default, this value is set to `false`.)

Use the `setunc` function to set the uncertain parameter values within the response optimization project.

## Example

Create a grid of uncertain parameter values for the parameters `P`, `I`, and `D`.

```
uset=gridunc('P',[1,2,3,4],'I',[0.1,0.2,0.3],'D',[30,35,40])
```

This returns

```
    Optimized: [4x3x3 logical]
            P: [4x3x3 double]
            I: [4x3x3 double]
            D: [4x3x3 double]

4x3x3 grid of parameter vectors.
```

View the data in detail using dot notation. For example:

```
uset.P

ans(:,:,1) =

     1     1     1
     2     2     2
```

```
     3     3     3
     4     4     4

ans(:,:,2) =

     1     1     1
     2     2     2
     3     3     3
     4     4     4

ans(:,:,3) =

     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

To optimize responses based on all the parameter combinations within `uset`, enter the following command:

```
uset.Optimized(1:end)=true;
```

## See Also

`randunc`, `setunc`

# initpar

## Purpose

Initialize tuned parameters

## Syntax

```
initpar(proj)
```

## Description

`initpar(proj)` sets the `InitialGuess` value of tuned parameters in the response optimization project, `proj`, with the values of the parameters that are currently in the model or base workspace.

## See Also

`findpar`

## Purpose

Upgrade models with old Nonlinear Control Design Blockset blocks

## Syntax

```
ncdupdate('modelname')
```

## Description

`ncdupdate('modelname')` searches the Simulink model specified by the string `'modelname'` for Nonlinear Control Design Blockset Outport blocks and replaces them by the equivalent Signal Constraint block from Simulink Response Optimization library. The model must be open prior to calling `ncdupdate`. Nonlinear Control Design Blockset software is the version of Simulink Response Optimization software that existed before Release 14.

When your model automatically loads its Nonlinear Control Design Blockset settings from an `ncdStruct` variable, this variable changes in the workspace during the update so that it is compatible with Simulink Response Optimization software. Make sure to resave this variable after the update so that the correct settings load with your model.

When your Nonlinear Control Design Blockset settings are stored in an `ncdStruct` variable, but do not automatically load with the model, first load the `ncdStruct` variable into the workspace before calling `ncdupdate`, and then resave the variable afterwards.

To retain the upgraded blocks, make sure you also save the model after running `ncdupdate`.

## See Also

`slupdate`

## Purpose

Create default Simulink Response Optimization project

## Syntax

```
proj=newsro('modelname',params)
```

## Description

`proj=newsro('modelname',params)` creates a new response optimization project, `proj`, for the Simulink model with name `modelname`. The tuned parameters are specified by the cell array of strings, `parameters`. The specified model should contain at least one block from Simulink Response Optimization library. Type `srolib` to open the library. Use the project with the `optimize` function to optimize response signals in the model by tuning specified parameters.

## Example

Create a project, `proj`, for the model `pidtune_demo` with the tuned parameters `Kp`, `Ki`, and `Kd`.

```
proj = newsro('pidtune_demo',{'Kp' 'Ki' 'Kd'})
```

This returns

```
              Name: 'pidtune_demo'
        Parameters: [3x1 ResponseOptimizer.Parameter]
      OptimOptions: [1x1 ResponseOptimizer.OptimOptions]
             Tests: [1x1 ResponseOptimizer.SimTest]
             Model: 'pidtune_demo'

Simulink Response Optimization Project.
```

Use the `findpar` and `findconstr` functions to specify signal constraints and tuned parameters.

## See Also

`findconstr`, `findpar`, `getsro`, `optimize`

## Purpose

Current optimizer settings

## Syntax

```
opt_settings=optimget(proj)
```

## Description

`opt_settings=optimget(proj)` returns the current optimization settings object, `opt_settings`, for the response optimization project `proj`.

Use `optimset` to modify the optimization options.

For more information on the settings and their possible values, see the `optimset` reference page.

## Example

Create a new default response optimization project for the model `srotut1`.

```
proj=newsro('srotut1','Kint');
```

Get the optimization settings for this project.

```
opt_settings=optimget(proj)
```

This command returns the following list of optimization settings and their current values.

```
                    Algorithm: 'fmincon'
                      Display: 'iter'
                 GradientType: 'basic'
            MaximallyFeasible: 0
                      MaxIter: 100
                       TolCon: 1.0000e-003
                       TolFun: 1.0000e-003
                         TolX: 1.0000e-003
                     Restarts: 0
                  UseParallel: 'never'
    ParallelPathDependencies: {0x1 cell}
                SearchMethod: []
```

## See Also

optimset, simget, simset

## Purpose

Run response optimization project

## Syntax

```
result=optimize(proj)
```

## Description

`result=optimize(proj)` optimizes the responses specified in the response optimization project, `proj`, with the constraints, parameters, and settings. The response optimization results are displayed after each iteration. The tuned parameters are changed in the workspace. Enter the parameter name at the MATLAB prompt to see its new value.

A results object, `result`, is also returned. The properties of this object are

- `Cost`: The final value of the cost function.
- `ExitFlag`: `1` if the optimization terminated successfully, `0` if it did not.
- `Iteration`: The number of iterations.

For more information on the results properties, see the reference pages for the Optimization Toolbox functions `fmincon` and `fminsearch` and the Genetic Algorithm and Direct Search Toolbox function `patternsearch`.

## Example

Open the `pitchrate_demo` model.

```
pitchrate_demo
```

Create a response optimization project based on the current settings in the model.

```
proj=getsro('pitchrate_demo');
```

Run the optimization with the following command.

```
results=optimize(proj)
```

The expected results are displayed as follows.

```
                          max              Directional  First-order
Iter  S-count  f(x)   constraint  Step-size  derivative   optimality  Procedure
   0        1     0         1803
   1       14     0          160     0.0287           0       0.0152
   2       21     0       0.2607     0.0327           0      0.00598   Hessian modified
   3       28     0      0.04203      0.071           0       0.0122   Hessian modified
   4       35     0     0.001894     0.0164           0      0.00112   Hessian modified
   5       42     0   7.631e-006   0.000804           0    5.01e-006   Hessian modified
Successful termination.
Found a feasible or optimal solution within the specified tolerances.

k1 =

    0.8674


k2 =

   -0.1513


k3 =

   -0.5003


results =

         Cost: 0
            X: [4x1 double]
     ExitFlag: 1
    Iteration: 5
```

The results can differ due to changes in MATLAB numerics, Simulink solvers, optimization algorithms, and running on different platforms.

## See Also

findconstr, findpar, getsro, newsro, optimget, optimset

## Purpose

Modify optimization settings

## Syntax

```
optimset(proj,'Property1',Value1,'Property2',Value2,...)
```

## Description

`optimset(proj,'Property1',Value1,'Property2',Value2,...)` modifies the optimization settings within the response optimization project, `proj`. The value of the optimization setting, `Property1`, is set to `Value1`, `Property2` is set to `Value2`, etc.

| Property | Description | Possible Settings |
|---|---|---|
| `Algorithm` | The optimization algorithm used. The following algorithms are available:<br>• `fmincon` — Optimization Toolbox function `fmincon`<br>• `patternsearch` — Genetic Algorithm and Direct Search Toolbox function `patternsearch`<br>• `fminsearch` — Optimization Toolbox function `fminsearch` | `{'fmincon'} \| 'patternsearch' \| 'fminsearch'` |
| `Display` | The level of information that the optimization displays:<br>• `off` — No output<br>• `iter` — Output at each iteration<br>• `final` — Final output only<br>• `notify` — Output only if the function does not converge | `'off' \| {'iter'} \| 'final' \| 'notify'` |

| Property | Description | Possible Settings |
|---|---|---|
| `GradientType` | Method used to calculate gradients when using `'fmincon'` as the `Algorithm`. Use one of the following finite difference methods for gradient calculation:<br>• `basic` — Default method for computing the gradients<br>• `refined` — Offers a more robust and less noisy gradient calculation method than `'basic'`<br>The `refined` method is sometimes more expensive, and does not work with certain models such as SimPowerSystems models. | {`'basic'`} \| `'refined'` |

| Property | Description | Possible Settings |
|---|---|---|
| `MaximallyFeasible` | Option to specify that the optimization algorithm continue after an initial solution has been found:<br>• `0` — Terminate the optimization as soon as an initial solution that satisfies the constraints is found. The resulting response signal may lie very close to the constraint segment.<br>• `1` — Continue the optimization after an initial solution is found. The optimization can continue to search for a maximally feasible solution that is typically located further inside the constraint region. | {0} \| 1 |
| `MaxIter` | Maximum number of iterations allowed | Positive integer value |
| `TolCon` | Termination tolerance on the constraints | Positive scalar value |
| `TolFun` | Termination tolerance on the function value | Positive scalar value |
| `TolX` | Termination tolerance on the parameter values | Positive scalar value |

| Property | Description | Possible Settings |
|---|---|---|
| `Restarts` | In some optimizations, the Hessian may become ill-conditioned, and the optimization does not converge. In these cases, it is sometimes useful to restart the optimization after it stops, using the endpoint of the previous optimization as the starting point for the next one. To automatically restart the optimization, use this option to indicate the number of times you want to restart. | Nonnegative integer value |

| Property | Description | Possible Settings |
|---|---|---|
| `UseParallel` | Parallel computing option for the following optimization algorithms:<br>• `fmincon`<br>• `patternsearch`<br><br>**Note** Parallel Computing Toolbox software must be installed to enable parallel computing for the optimization algorithms.<br><br>When set to `'always'`, the algorithms compute the following in parallel:<br>• `fmincon` — Computes finite difference gradients<br>• `patternsearch` — Performs population evaluation<br><br>Disable the option by setting to `'never'`. | `'always' \| {'never'}` |
| `ParallelPathDependencies` | Option to store model path dependencies when using parallel computing | Cell array of strings |
| `SearchMethod` | Search options for use with the `patternsearch` algorithm | See "Search Options" in the Genetic Algorithm and Direct Search Toolbox documentation. |

For more information on the possible settings and the values they can take, see the reference page for `optimset` in the MATLAB documentation.

## Example

Create a default response optimization project for the model `srotut1`.

```
proj=newsro('srotut1','Kint');
```

Get the optimization settings for this project.

```
opt_settings=optimget(proj)
```

This command returns the following list of optimization settings and their current values.

```
                    Algorithm: 'fmincon'
                      Display: 'iter'
                 GradientType: 'basic'
            MaximallyFeasible: 0
                      MaxIter: 100
                       TolCon: 1.0000e-003
                       TolFun: 1.0000e-003
                         TolX: 1.0000e-003
                     Restarts: 0
                  UseParallel: 'never'
    ParallelPathDependencies: {0x1 cell}
                SearchMethod: []
```

Use `optimset` to change the maximum number of iterations to 150.

```
optimset(proj,'MaxIter',150)
```

To view the changes to `opt_settings`, enter the variable name at the MATLAB prompt.

```
opt_settings
```

This command returns

```
                    Algorithm: 'fmincon'
                      Display: 'iter'
                 GradientType: 'basic'
            MaximallyFeasible: 0
                      MaxIter: 150
                       TolCon: 1.0000e-003
                       TolFun: 1.0000e-003
                         TolX: 1.0000e-003
                     Restarts: 0
                  UseParallel: 'never'
ParallelPathDependencies: {0x1 cell}
                 SearchMethod: []
```

## See Also

optimget, simget, simset

## Purpose

Randomly sample uncertain parameters

## Syntax

```
uset=randunc(N,'P1',Range1,'P2',Range2,...)
```

## Description

`uset=randunc(N,'P1',Range1,'P2',Range2,...)` generates random values for the parameters `P1`, `P2`, ... subject to the range constraints `Range1`, `Range2`, ...

The parameters `P1`, `P2`, ... are uncertain parameters in a response optimization project. Each range constraint specifies lower and upper bounds for the uncertain parameter value.

For a scalar-valued parameter, `p`, specify the range as `[Min,Max]` or `{Min,Max}`. The interpretation is then

```
Min <= p <= Max
```

For vector- or matrix-valued parameters, specify the range as `{Min,Max}` where `Min` and `Max` are commensurate vectors or matrices. The interpretation is then

```
Min(i,j) <= p(i,j) <= Max(i,j)
```

The set of uncertain parameter values consists of

- All vertices of the parameter box specified by the upper and lower bounds ($2^S$ values if there are $S$ parameters).
- `N` randomly picked points inside the parameter box, where `N` is the first input argument to `randunc`.

To optimize the responses based on uncertain parameter values, set the `Optimized` property of the uncertain parameter object, `uset`, to `true`. By default, this value is set to `false`.

Use the `setunc` function to set the uncertain parameter values within the response optimization project.

## Example

Create a set of 12 randomly generated uncertain parameter values for the parameters `P`, `I`, and `D`.

```
uset=randunc(4,'P',{1,4},'I',{0.1,0.3},'D',{30,40})
```

This returns

```
    Optimized: [12x1 logical]
            P: [12x1 double]
            I: [12x1 double]
            D: [12x1 double]

Scattered set with 12 parameter vectors.
```

View the data in detail using dot notation. For example:

```
uset.P

ans =

    1.0000
    4.0000
    1.0000
    4.0000
    1.0000
    4.0000
    1.0000
    4.0000
    2.2372
    2.8691
    2.6946
    1.1896
```

To optimize responses based on all the parameter combinations within `uset`, enter the following command.

```
uset.Optimized(1:end)=true
```

## See Also

gridunc, setunc

# setunc

## Purpose

Specify parameter uncertainty in optimization project

## Syntax

```
setunc(proj,unc_settings)
```

## Description

`setunc(proj,unc_settings)` sets the parameter uncertainty specifications for the response optimization project, `proj`. Use the function `gridunc` or `randunc` to specify the uncertainty settings, `unc_settings`.

## Example

Create a response optimization project.

```
proj=newsro('srotut1','Kint');
```

Specify uncertain parameter settings using `gridunc`.

```
uset=gridunc('zeta',[0.9,1,1.1],'w0',[0.95,1,1.05]);
```

Set the uncertain parameters in the project.

```
setunc(proj,uset)
```

## See Also

`gridunc`, `randunc`

## Purpose

Current simulation settings

## Syntax

```
simoptions=simget('proj')
```

## Description

`simoptions=simget('proj')` returns a object containing the current simulation options, `simoptions`, used by the response optimization project, `proj`. To modify the project's simulation settings, use the function `simset`.

For a detailed list of simulation options and the possible values they can take, see the reference page for the Simulink function `simset`. The default values of the simulation options for the project are the same as those used by the Simulink model the project is associated with. Changes that are made to the project's simulation settings are only used during simulations that are run as part of the optimization, and they do not affect the simulation settings for the model.

## Example

Create a response optimization project for the `srotut1` model.

```
proj=newsro('srotut1','Kint');
```

Get the simulation settings for this project

```
simoptions = simget(proj)
```

This returns

```
simoptions =
         AbsTol: 1.0000e-006
      FixedStep: 'auto'
    InitialStep: 'auto'
        MaxStep: 'auto'
        MinStep: 'auto'
         RelTol: 1.0000e-003
         Solver: 'ode45'
      ZeroCross: 'on'
      StartTime: '0.0'
       StopTime: '50'
```

## See Also

optimget, optimset, simset

## Purpose

Modify simulation settings

## Syntax

```
simset(proj,'setting1',value1,'setting2',value2,...)
```

## Description

`simset(proj,'setting1',value1,'setting2',value2,...)` modifies the simulation settings within the response optimization project, `proj`. The value of the simulation setting, `setting1`, is set to `value1`, `setting2` is set to `value2`, etc.

For a detailed list of simulation options and the possible values they can take, see the reference page for the Simulink function `simset`. The default values of the simulation options for the project are the same as those used by the Simulink model the project is associated with. Changes that are made to the project's simulation settings are only used during simulations that are run as part of the optimization, and they do not affect the simulation settings for the model.

## Example

Create a response optimization project for the `srotut1` model.

```
proj=newsro('srotut1','Kint');
```

Get the simulation settings for this project.

```
simoptions = simget(proj)
```

This returns

```
simoptions =
         AbsTol: 1.0000e-006
      FixedStep: 'auto'
    InitialStep: 'auto'
        MaxStep: 'auto'
        MinStep: 'auto'
         RelTol: 1.0000e-003
         Solver: 'ode45'
      ZeroCross: 'on'
      StartTime: '0.0'
       StopTime: '50'
```

Use `simset` to change the solver type to `ode23` and the absolute tolerance to `1e-7`.

```
simset(proj,'Solver','ode23','AbsTol',1e-7)
```

Check the new values:

```
sim_settings=simget(proj);
sim_settings.Solver
```

This shows that the solver is now set to `ode23`.

```
ans =
ode23
```

Check the absolute tolerance:

```
sim_settings.AbsTol
```

This value is now set to `1e-7`.

```
ans =
  1.0000e-007
```

## See Also

`optimget`, `optimset`, `simget`

5

# Block Reference

# CRMS

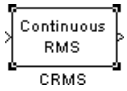
## Purpose

Compute continuous-time, cumulative root mean square (CRMS) of signal

## Library

Simulink Response Optimization

## Description




Attach the CRMS block to a signal to compute its continuous-time, cumulative root mean square value. Use in conjunction with the Signal Constraint block to optimize the signal energy.

The continuous-time, cumulative root mean square value of a signal $u(t)$, is defined as

$$R.M.S = \sqrt{\frac{1}{T}\int_0^T \|u(t)\|^2 \, dt}$$

The R.M.S value gives a measure of the average energy in the signal.

## See Also

DRMS, Signal Constraint

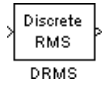
## Purpose

Compute discrete-time, cumulative root mean square (DRMS) of signal

## Library

Simulink Response Optimization

## Description




Attach the DRMS block to a signal to compute its discrete-time, cumulative root mean square value. Use in conjunction with the Signal Constraint block to optimize the signal energy.

The discrete-time, cumulative root mean square value of a signal $u(t_i)$, is defined as

$$R.M.S = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left\|u(t_i)\right\|^2}$$

The R.M.S value gives a measure of the average energy in the signal.

## See Also

CRMS, Signal Constraint

# Signal Constraint

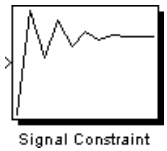
## Purpose

Specify desired signal response

## Library

Simulink Response Optimization

## Description




Attach a Signal Constraint block to the signal to optimize its response to known inputs. Simulink Response Optimization software tunes parameters in the model to meet specified constraints. The constraints include bounds on signal amplitudes and matching of reference signals. The constraints are applicable to vector- and matrix-valued ports, in which case the signal bounds and reference signals apply to all entries of the signal/matrix.

For a complete discussion of the use of this block, see the Simulink Response Optimization documentation starting with “Product Overview” and Chapter 1, “Approaching Response Optimization in Simulink Models”.

## See Also

CRMS, DRMS

A

# Examples

See “Response Optimization Tutorial” in the Getting Started guide.

# Index

## R



## S



## T



## U