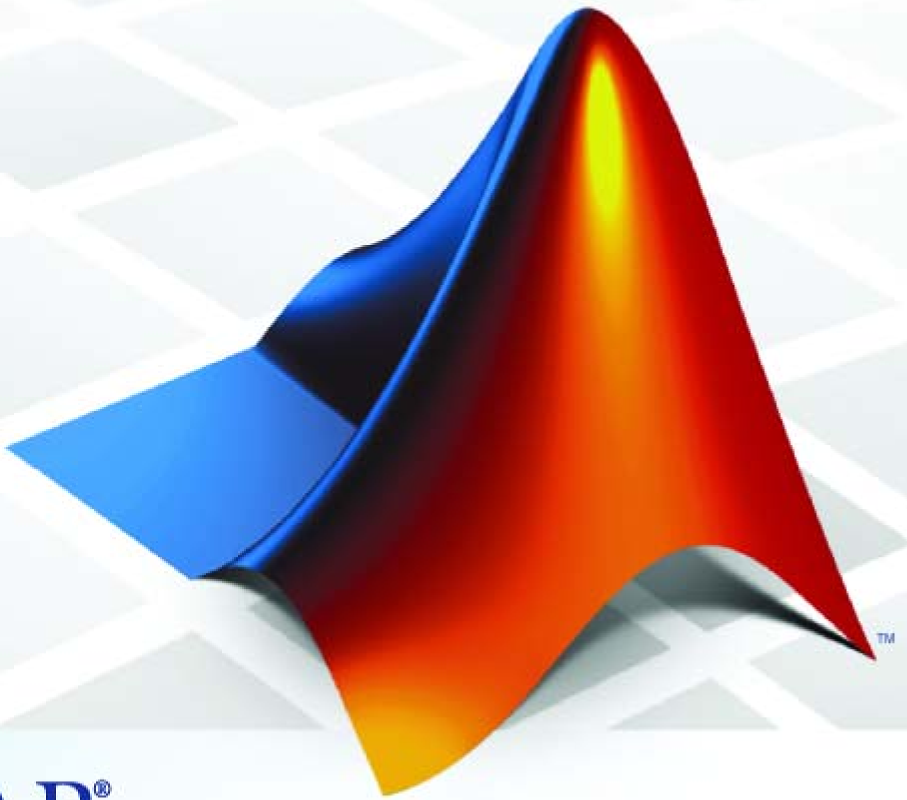
Simulink® Design Optimization™ 1
User's Guide



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® Design Optimization™ User's Guide

© COPYRIGHT 1993–2009 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

March 2009	Online only	New for Version 1 (Release 2009a)

Contents

3 Optimizing Model Parameters

5 Modeling Systems Using Lookup Tables

6 Function Reference

Data Analysis and Processing

Configuring a Model for Importing Data

Before you can analyze and preprocess the estimation data, you must assign the data to the model's channels. In order to assign the data, the Simulink® model must contains one of the following elements:

- Top-level Inport block

Note You do not need an Inport block if your model already contains a fixed input block, such as a Step block.

- Top-level Outport block
- Logged signal. The logged signal can be a top-level signal in the model or a signal in the model subsystem.

  For more information about the blocks and logged signals, see the Inport and Outport block reference pages and "Logging Signals" in the Simulink documentation.

In the Control and Estimation Tools Manager GUI, the rows in the **Input Data** tab correspond to the model's top-level Inport blocks. Similarly, the rows in the **Output Data** tab correspond to either the top-level Outport blocks or logged signals in the model.

Adding an Inport or Outport block or marking a signal for logging creates a new row in the corresponding **Input Data** or **Output Data** tab. You can use the new row to import estimation data for the corresponding signal. To view the new row, click **Update Task** in the **Estimation Task** node of the Control and Estimation Tools Manager GUI.

Creating an Estimation Project

Before you begin data import, you must create and set up an estimation project by configuring the appropriate parameters, solvers, and cost functions. Simulink® Design Optimization™ software provides a Graphical User Interface (GUI) that makes setting up the estimation project quick and easy.

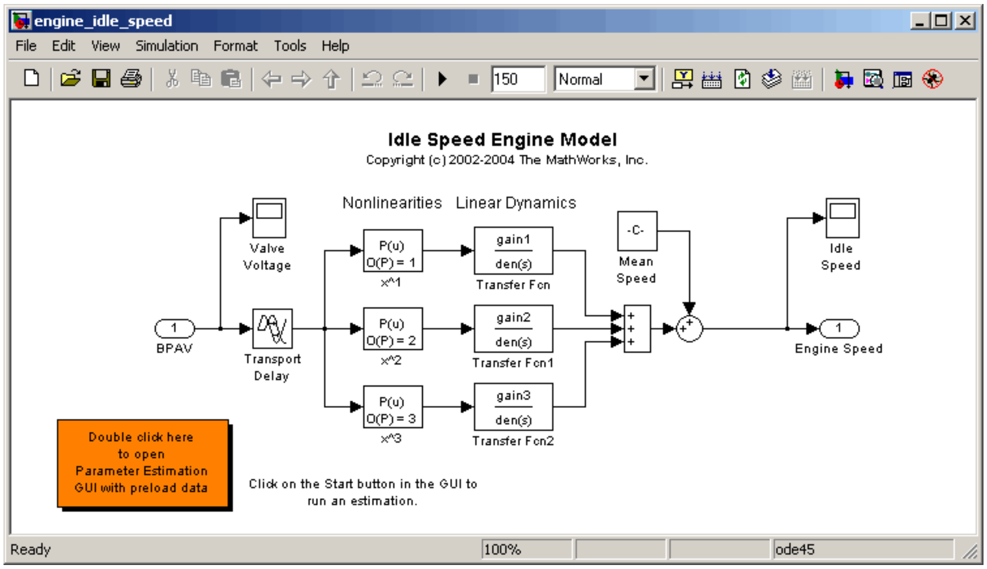
To create an estimation project:

1. Open the nonlinear idle speed model of an automotive engine by typing :

   ```
   engine_idle_speed
   ```

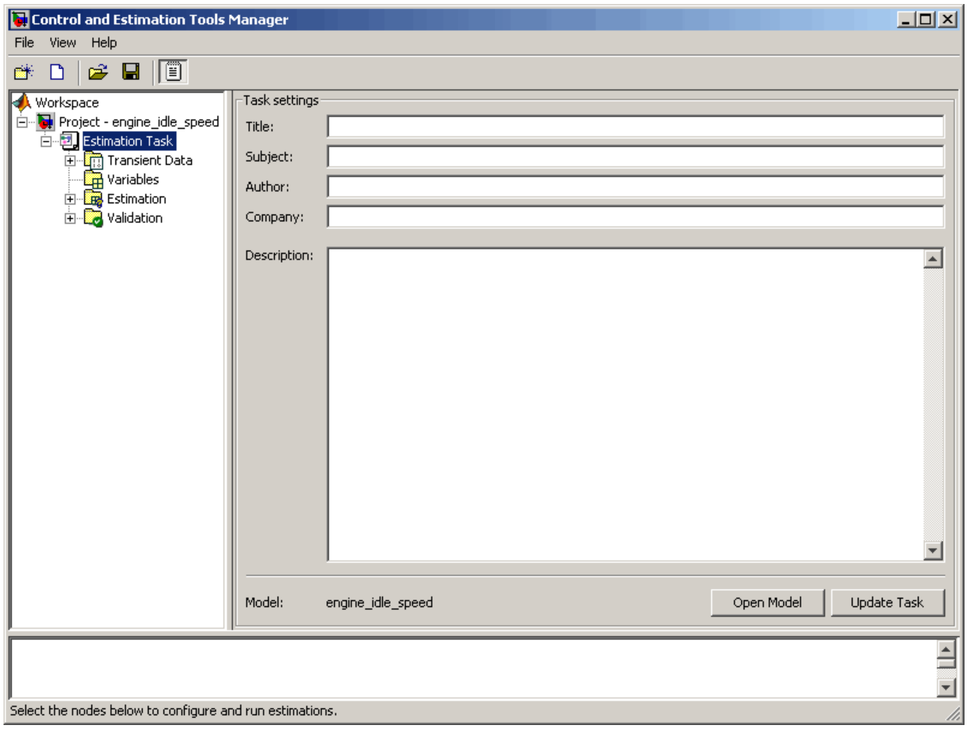
   at the MATLAB® prompt.

   The model appears as shown next.

   The model contains the Inport block `BPAV` and Outport block `Engine Speed` for importing input and output data, respectively. To learn more, see "Configuring a Model for Importing Data " on page 1-2.

2 Open the Control and Estimation Tools Manager GUI by selecting **Tools > Parameter Estimation** in the Simulink model window.



Control and Estimation Tools Manager GUI

The project tree displays the project name **Project - engine_idle_speed**. Estimation tasks are organized inside the **Estimation Task** node.

Importing Data into the GUI

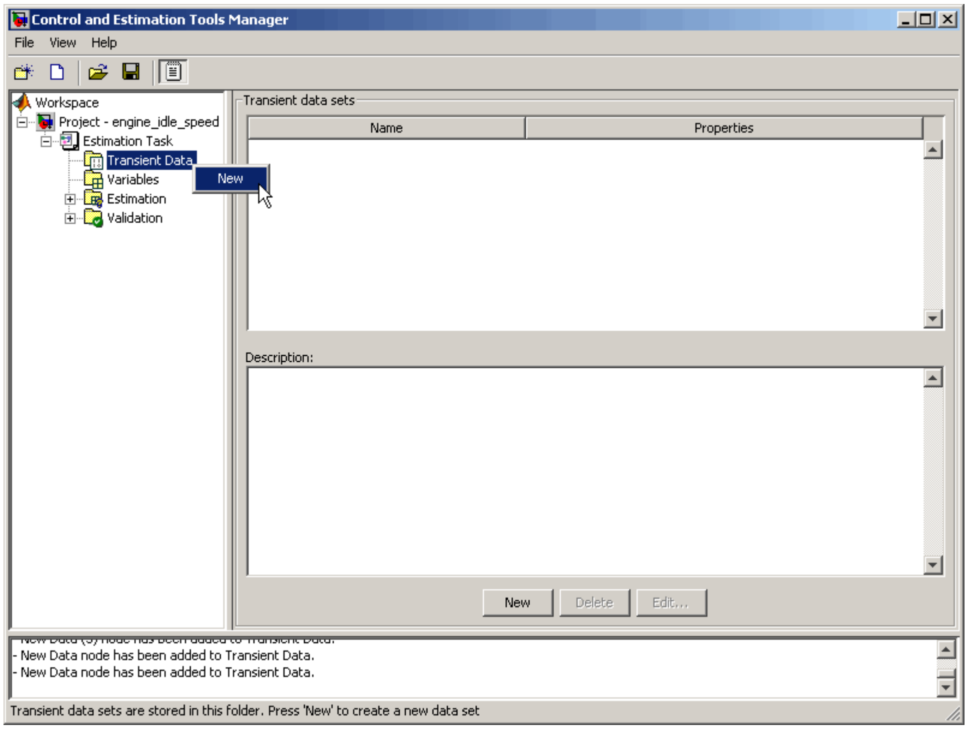
How To Import Time-Domain Data into the GUI

After you create an estimation project, as described in “Creating an Estimation Project” on page 1-3, you can import the estimation data into the GUI. To learn more about the types of data for parameter estimation, see “Types of Data for Parameter Estimation” in the *Simulink Design Optimization Getting Started Guide*.

To import *transient* (measured) data for your dynamic system:

1 In the Control and Estimation Tools Manager, select **Transient Data** under the **Estimation Task** node of the **Workspace** directory tree.

2 Right-click **Transient Data** and select **New** to create a **New Data** node. Alternatively, you can use the **New** button to create this node.

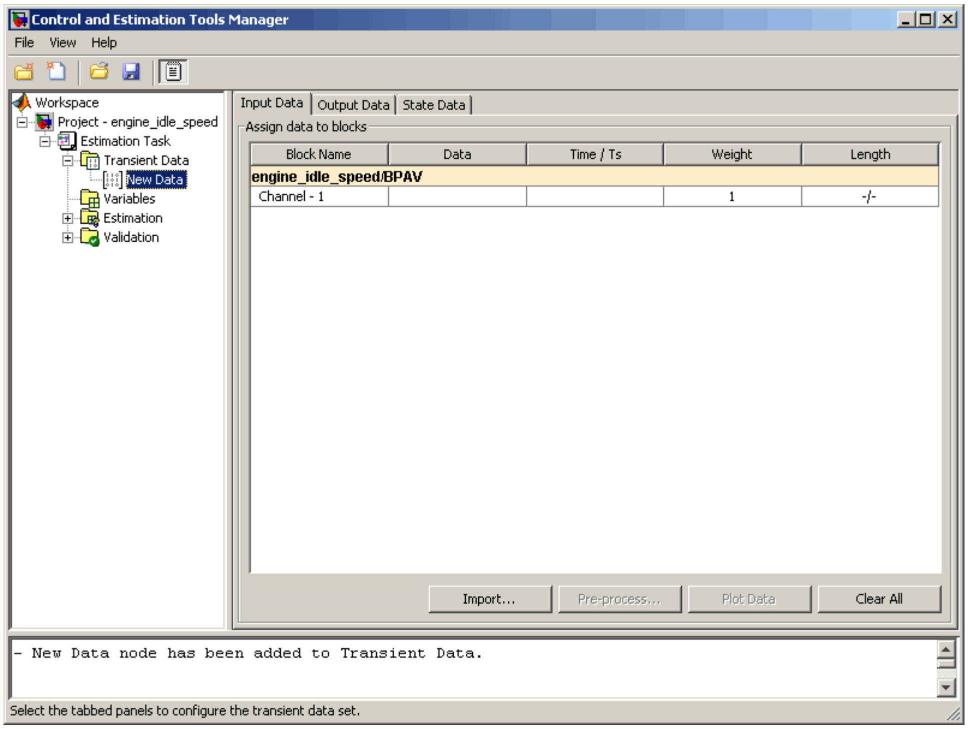


- New Data node has been added to Transient Data.
- New Data node has been added to Transient Data.

Transient data sets are stored in this folder. Press 'New' to create a new data set

3 Select the **New Data** node under the **Transient Data** node.

The Control and Estimation Tools Manager GUI now resembles the next figure.



Block Name	Data	Time / Ts	Weight	Length
engine_idle_speed/BPAV				
Channel - 1			1	-/-

```
- New Data node has been added to Transient Data.
```

Select the tabbed panels to configure the transient data set.

Import Data into the Control and Estimation Tools Manager

The table rows in the **Input Data** tab corresponds to the Inport block `BPAV` in the `engine_idle_speed` model. Similarly, the rows in the **Output Data** tab corresponds to the Outport block `Engine Speed`.

Note The Simulink model must contain an Inport or Outport block or logged signals to enable importing data. For more information, see "Configuring a Model for Importing Data " on page 1-2.

The idle-speed model of an automotive engine contains the measured data stored in the `iodata` array. The array contains two columns: the first for input data, and the second for output data. You must import both the input and the output data, as described in the following sections:

- "Importing Input Data and Time Vector" on page 1-8

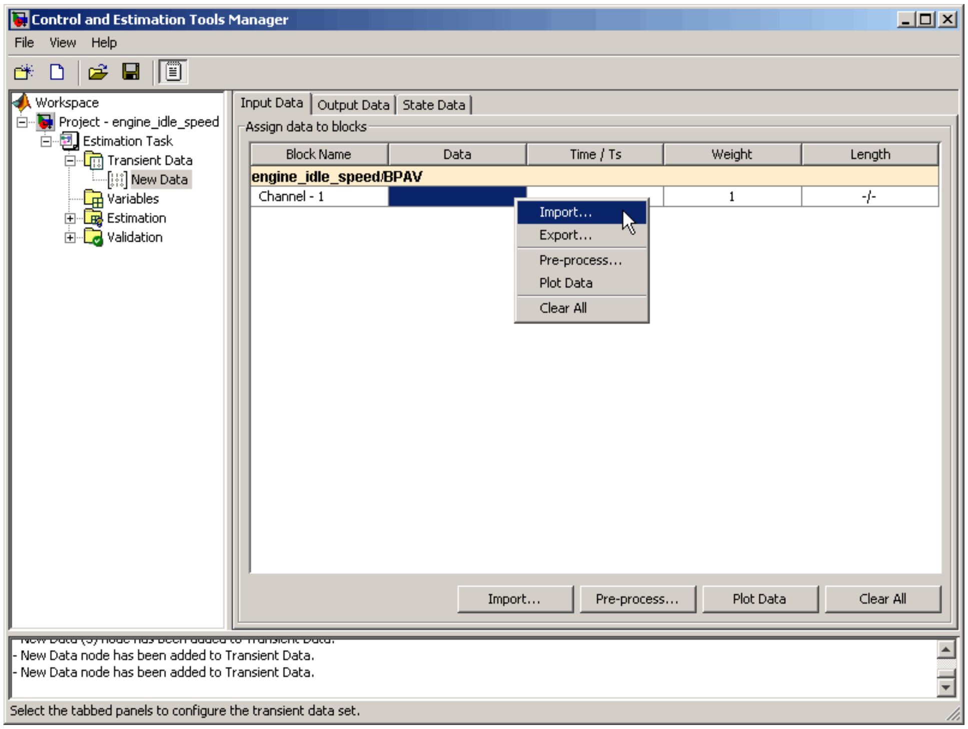
- "Importing Output Data and Time Vector" on page 1-9

Importing Input Data and Time Vector

To import the input data for the port BPAV:

1 In the **New Data** node, click the **Input Data** tab.

2 Right-click the **Data** cell and select **Import** to open the Data Import dialog box. Alternatively, you can use the **Import** button to open this dialog box.

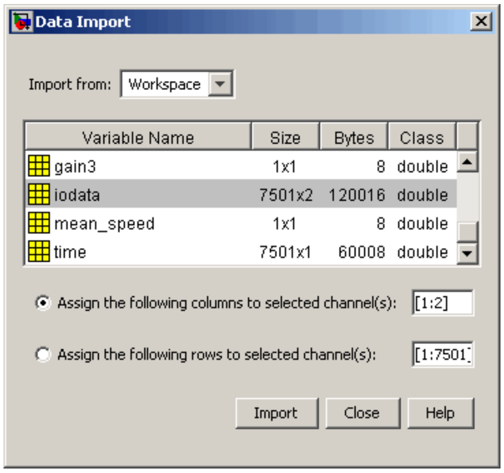


Block Name	Data	Time / Ts	Weight	Length
engine_idle_speed/BPAV				
Channel - 1			1	-/-

```
- New Data node has been added to Transient Data.
- New Data node has been added to Transient Data.
```

Select the tabbed panels to configure the transient data set.

3 In the Data Import dialog box, select `iodata` from the list of variables.



Variable Name	Size	Bytes	Class
gain3	1x1	8	double
iodata	7501x2	120016	double
mean_speed	1x1	8	double
time	7501x1	60008	double

● Assign the following columns to selected channel(s): [1:2]

○ Assign the following rows to selected channel(s): [1:7501]

4 Enter 1 in the **Assign the following columns to selected channel(s)** field, and then click **Import**.

5 In the **Input Data** tab, select the **Time/Ts** cell.

6 Select `time` in the Data Import dialog box.

7 Click **Import** to import the time vector for the input data.

8 Click **Close** to close the Data Import dialog box.

Importing Output Data and Time Vector

To import the output data for the port `Engine Speed`:

1 In the **New Data** node, select the **Output Data** tab.

2 Right-click the **Data** cell and select **Import** to open the Data Import dialog box.

3 In the Data Import dialog box, select `iodata` from the list of variables.

4 Enter 2 in the **Assign the following columns to selected channel(s)** field to use the second column of `iodata`, and then click **Import**.

5 In the **Output Data** tab, select the **Time/Ts** cell.

6. Select `time` in the Data Import dialog box.
7. Click **Import** to import the time vector for the output data.
8. Click **Close** to close the Data Import dialog box.

Plotting and Analyzing Data in the GUI

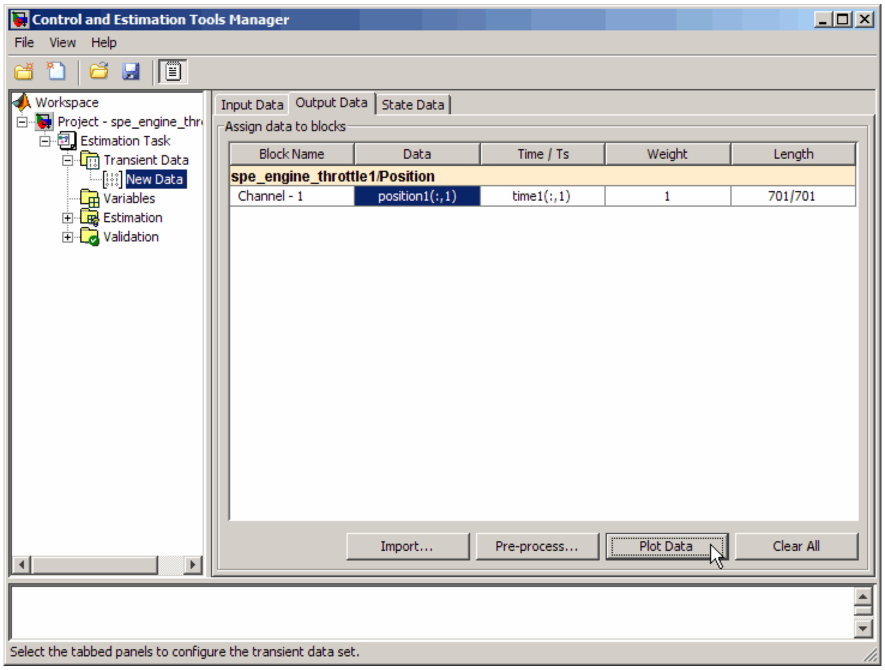
In this section...

Why Plot the Data Before Parameter Estimation

After you import the estimation data, as described in "Importing Data into the GUI" on page 1-5, it is useful to remove outliers, smooth, detrend, or otherwise treat the data to make it more tractable for analysis and estimation purposes. To view and analyze the data characteristics, you must plot the data on a time plot.

How To Plot Data in the GUI

To plot a data set, select the **Data** cell that you want to plot in the **Transient** Data node of the Controls and Estimation Tools Manager GUI, and click **Plot Data**.

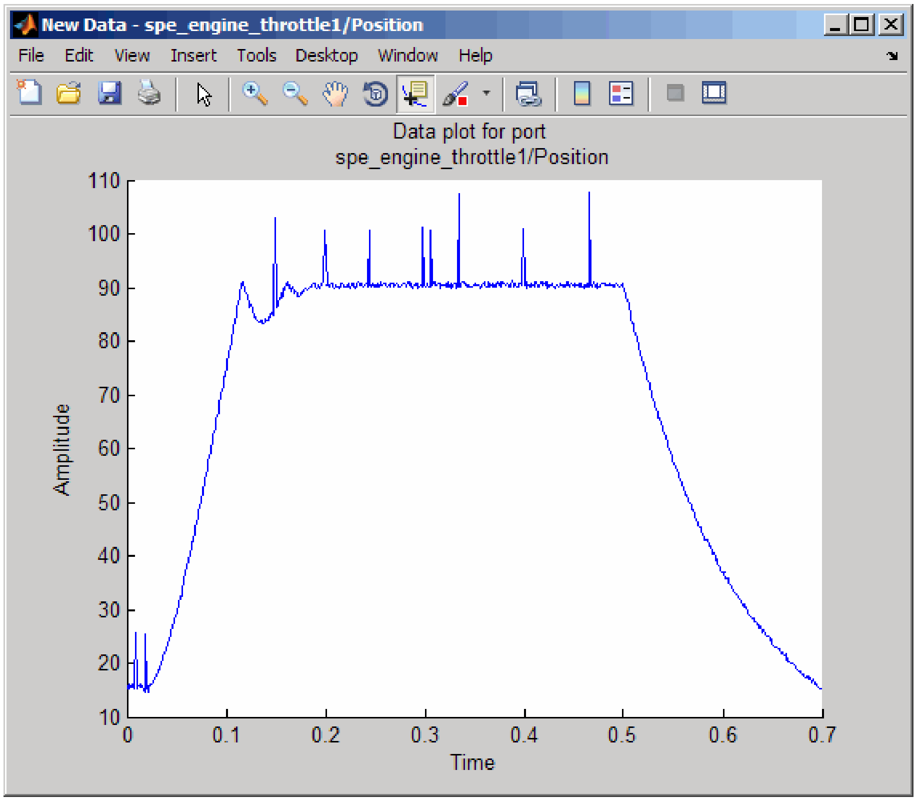


Assign data to blocks

Block Name	Data	Time / Ts	Weight	Length
spe_engine_throttle1/Position				
Channel - 1	position1(:,1)	time1(:,1)	1	701/701

Select the tabbed panels to configure the transient data set.

The data is plotted on a time plot, as shown in the next figure.


New Data - spe_engine_throttle1/Position
File Edit View Insert Tools Desktop Window Help
Data plot for port
spe_engine_throttle1/Position
Amplitude
Time
110
100
90
80
70
60
50
40
30
20
10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Using the time plot, you can examine the data characteristics such as noise, outliers and portions of the data to use for estimating parameters. After you analyze the data, you the preprocess the data as described in "Preprocessing Data in the GUI" on page 1-14.

Preprocessing Data in the GUI

In this section...

Ways to Preprocess Data Using the Data Preprocessing Tool

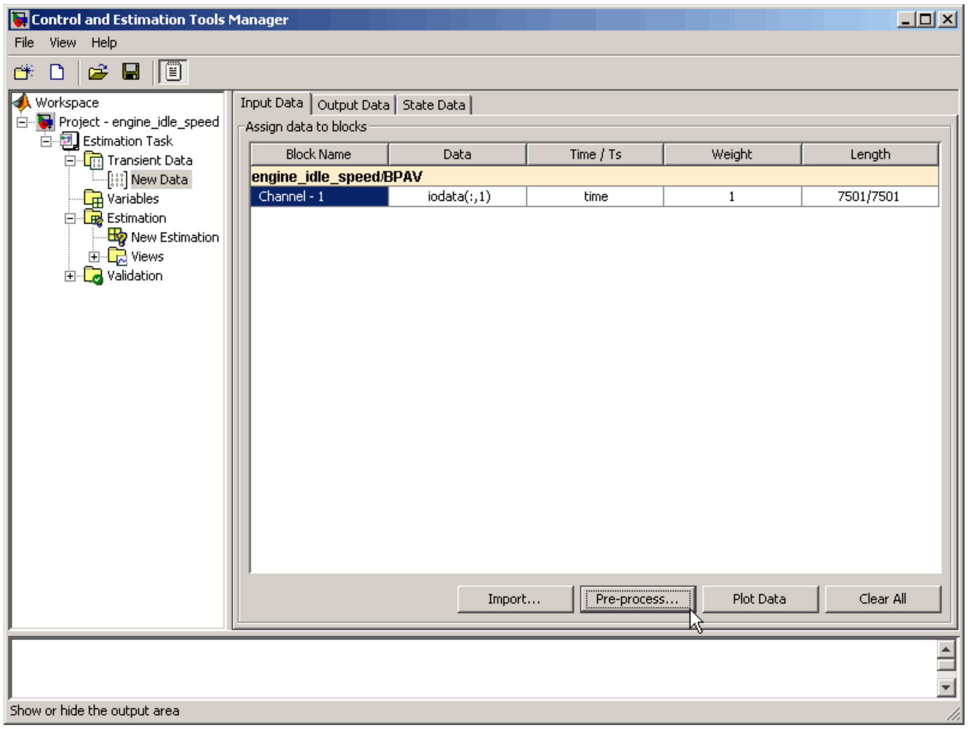
After you import the estimation data, as described in “Importing Data into the GUI” on page 1-5, you can perform the following preprocessing operations using the Data Preprocessing Tool in Simulink Design Optimization software:

- Exclusion — Exclude a portion of the data from the estimation process. You can exclude data by:
  - Selecting it with your mouse.
  - Graphically by selecting regions on a plot.
  - Using rules, such as upper or lower bounds.
- Handle missing data — Remove missing data, or compute missing data using interpolation.
- Handle outliers — Remove outliers.
- Detrend — Remove mean values or a straight line trend.
- Filter — Smooth data using a first-order filter, an arbitrary transfer function, or an ideal filter.

Opening the Data Preprocessing Tool

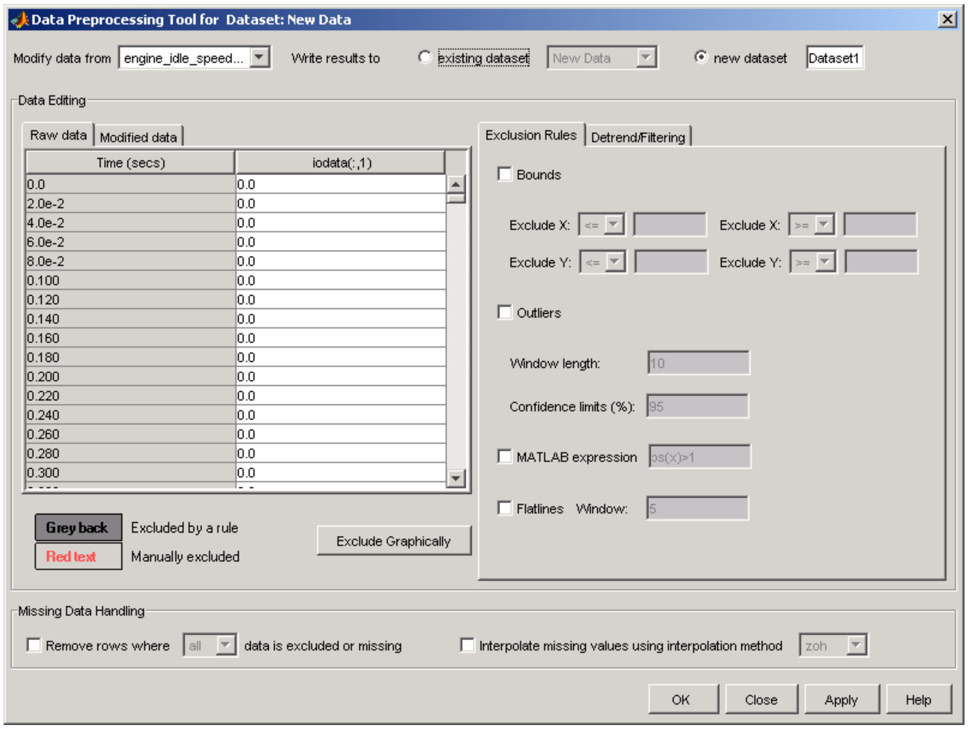
To open the Data Preprocessing Tool:

1 In the Control and Estimation Tools Manager GUI, select the **Transient Data** node under the **Estimation Task** node, and then choose the data you want to preprocess either in the **Input Data**, or **Output Data** tab. This enables the **Pre-process** button.



Block Name	Data	Time / Ts	Weight	Length
engine_idle_speed/BPAV				
Channel - 1	iodata(:,1)	time	1	7501/7501

2 Click **Pre-process** to open the Data Preprocessing Tool.


Data Preprocessing Tool for Dataset: New Data

Modify data from: engine_idle_speed... Write results to ( ) existing dataset New Data (●) new dataset Dataset1

Data Editing

Raw data | Modified data

Time (secs)	iodata(:,1)
0.0	0.0
2.0e-2	0.0
4.0e-2	0.0
6.0e-2	0.0
8.0e-2	0.0
0.100	0.0
0.120	0.0
0.140	0.0
0.160	0.0
0.180	0.0
0.200	0.0
0.220	0.0
0.240	0.0
0.260	0.0
0.280	0.0
0.300	0.0

Grey back: Excluded by a rule
Red text: Manually excluded

Exclusion Rules | Detrend/Filtering

☐ Bounds
Exclude X: <= Exclude X: >=
Exclude Y: <= Exclude Y: >=

☐ Outliers
Window length: 10
Confidence limits (%): 95

☐ MATLAB expression abs(x)>1

☐ Flatlines Window: 5

Missing Data Handling

☐ Remove rows where all data is excluded or missing ☐ Interpolate missing values using interpolation method zoh

---

Tip When you have multiple data sets, select the data set that you want to preprocess from the **Modify data from** drop-down list in the Data Preprocessing Tool.

In this section, the sample data set imported for preprocessing is the same as used in the `engine_idle_speed` Simulink model. For an overview of creating estimation projects and importing data sets, see “Configuring a Model for Importing Data ” on page 1-2, and “Creating an Estimation Project” on page 1-3.

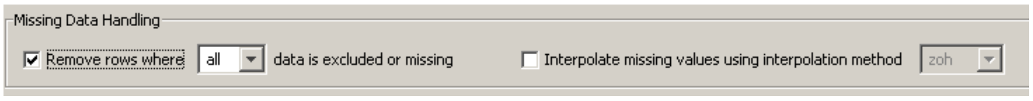
Handling Missing Data

- “Removing Missing Data” on page 1-17

- “Interpolating Missing Data” on page 1-17

Removing Missing Data

Rows of missing or excluded data are represented by `NaN`s. To remove the rows containing missing or excluded data, select the **Remove rows where** check box in the **Missing Data Handling** area of the Data Preprocessing Tool GUI.

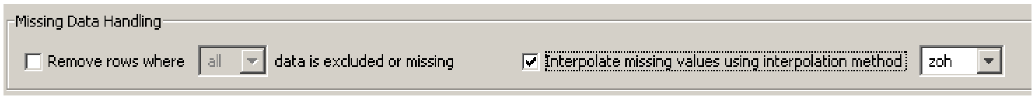

Missing Data Handling
☑ Remove rows where
all
data is excluded or missing
☐ Interpolate missing values using interpolation method
zoh

When the data set contains multiple columns of data, select `all` to remove rows in which all the data is excluded. Select `any` to remove any excluded cell. In the case of one-column data, `any` and `all` are equivalent.

Tip You can view the modified data in the **Modified data** tab of the Data Preprocessing Tool GUI.

Interpolating Missing Data

The interpolation operation computes the missing data values using known data values. When you select the **Interpolate missing values using interpolation method** check box in the **Missing Data Handling** area of the Data Preprocessing Tool GUI, the software interpolates the missing data values.


Missing Data Handling
☐ Remove rows where
all
data is excluded or missing
☑ Interpolate missing values using interpolation method
zoh

You can compute the missing data values using one of the following interpolation methods:

- Zero-order hold (`zoh`) — Fills the missing data sample with the data value immediately preceding it.
- Linear interpolation (`Linear`) — Fills the missing data sample with the average of the data values immediately preceding and following it.

By default, the interpolation method is set to `zoh`. You can select the `Linear` interpolation method from the **Interpolate missing values using interpolation method** drop-down list.

Tip You can view the results of interpolation in the **Modified data** tab of the Data Preprocessing Tool GUI.

Handling Outliers

Outliers are data values that deviate from the mean by more than three standard deviations. When estimating parameters from data containing outliers, the results may not be accurate.

To remove outliers, select the **Outliers** check box to activate outlier exclusion. You can set the **Window length** to any positive integer, and use confidence limits from 0 to 100%. The window length specifies the number of data points used when calculating outliers.

Removing outliers replaces the data samples containing outliers with `NaN`s, which you can interpolate in a subsequent operation. To learn more, see "Interpolating Missing Data" on page 1-17.

Detrending Data

To detrend, select the **Detrending** check box. You can choose constant or straight line detrending. Constant detrending removes the mean of the data to create zero-mean data. Straight line detrending finds linear trends (in the least-squares sense) and then removes them.

Filtering Data

- "Types of Filters" on page 1-18
- "How to Filter Data " on page 1-19

Types of Filters

You have these choices for filtering your data:

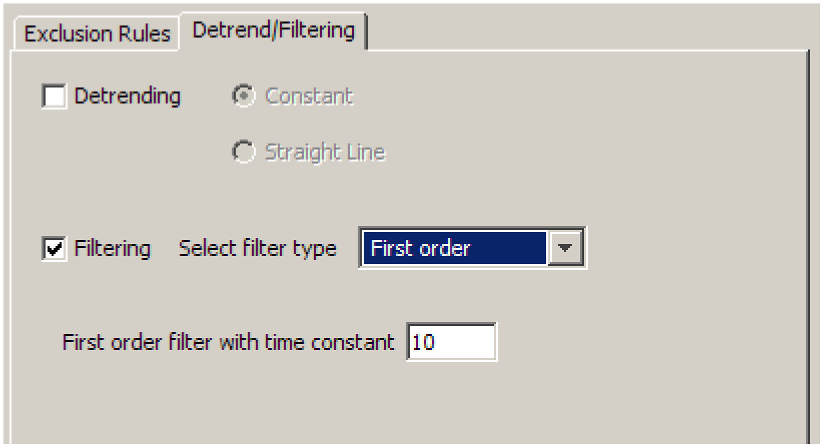
- `First order` — A filter of the type $\frac{1}{\tau s+1}$
  where τ is the time constant that you specify in the associated field.
- `Transfer function` — A filter of the type

  $$\frac{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_0}{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_0}$$

  where you specify the coefficients as vectors in the associated **A coefficients** and **B coefficients** fields.
- `Ideal` — An idealized (noncausal) filter, either stop or pass band. Specify either filter as a two-element vector in the **Range (Hz)** field. These filters are ideal in the sense that there is no finite rolloff or ripple; the ends of the ranges are perfectly horizontal in the frequency domain.

How to Filter Data

To filter the data to remove noise, select the **Detrend/Filtering** tab in the Data Preprocessing Tool GUI. Select the **Filtering** check box, and choose the type of filter from the **Select filter type** drop-down list.



Filtering ✓ Select filter type: First order

First order filter with time constant: 10

Selecting Data

Techniques for Excluding Data in the Data Preprocessing Tool

You can use the Data Preprocessing Tool to select a portion of the data to be excluded from the estimation process. You can choose one of the following techniques:

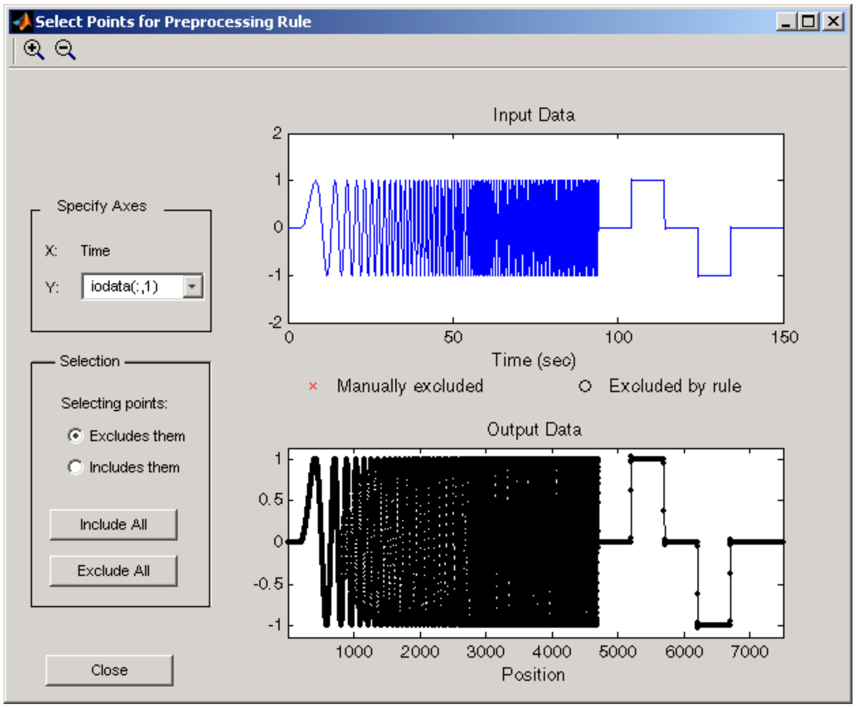
- Selecting data from the Data Editing Table.
- Selecting data from a plot of the data.
- Specifying a rule.

You accomplish the first two manually, and for the last you specify a rule. When you exclude data using manual selection, the excluded data is shown as red. When you exclude data using a rule, the background color of the cell becomes gray. When a portion of the data is excluded both manually and by a rule, the data is red, and the background is gray.

Note Changes in data are visible everywhere. When you use the **Data Editing** table, you can view the results in the data plot.

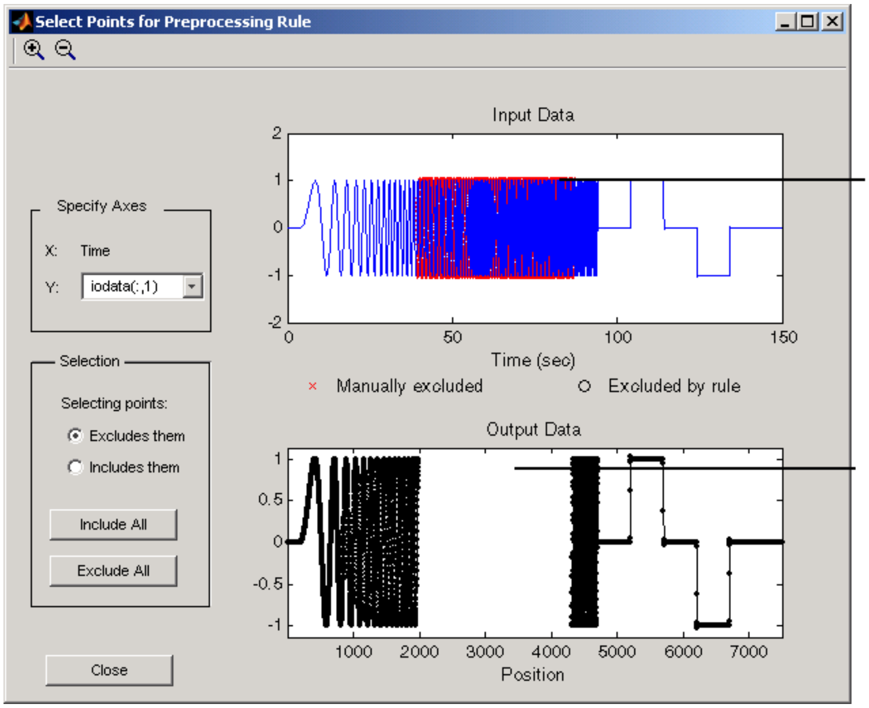
Graphically Selecting Data

You can exclude data graphically. Click **Exclude Graphically** to open the Select Points for Preprocessing Rule window.



The way you exclude data is similar to the way you select a region for zooming: place your cursor in the **Input Data** plot and drag the mouse to draw a region of exclusion.

This figure shows an example of resulting data exclusion in the input data.



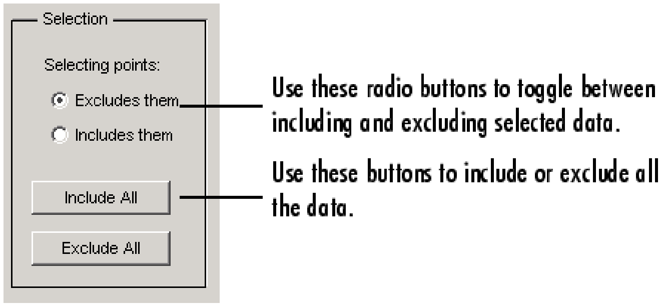
The excluded area is red in the Input Data plot.

By default, In the Output Data plot, the excluded input data produces a blank area.

In the **Output Data** plot, the excluded input data produces a blank area by default. This corresponds to the NaNs that now represent excluded data. If you choose to interpolate or remove the excluded data, the output data shows the interpolated points.

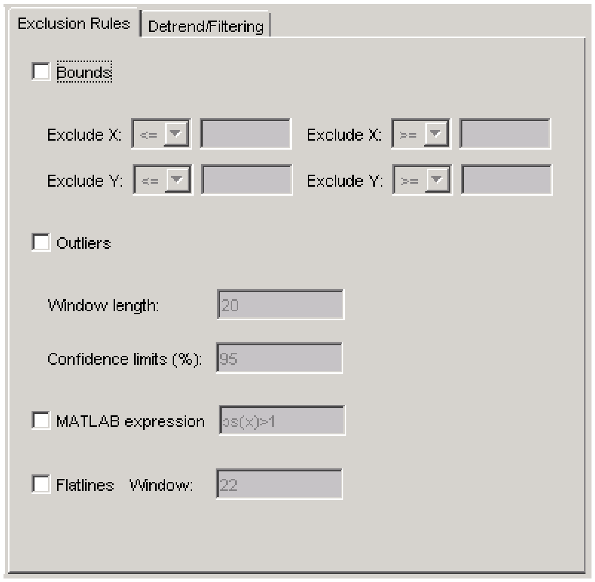
When you make changes in the Select Points for Preprocessing Rule window, they immediately appear in the **Data Editing** pane, and vice versa.

Selection Pane. By default, any box that you draw with your mouse selects data for exclusion, but you can toggle between exclusion and inclusion using the **Selection** pane on the left side of the Select Points for Preprocessing Rule window.


Selection
Selecting points:
Excludes them
Includes them
Include All
Exclude All
Use these radio buttons to toggle between including and excluding selected data.
Use these buttons to include or exclude all the data.

Using Rules to Select Data Samples

A more precise way to exclude data is to use mathematical rules. The **Exclusion Rules** pane in the Data Preprocessing Tool allows you to enter customized rules for excluding data.



These are the rules you can use to exclude data:

- "Upper and Lower Bounds" on page 1-24
- "MATLAB Expressions" on page 1-24

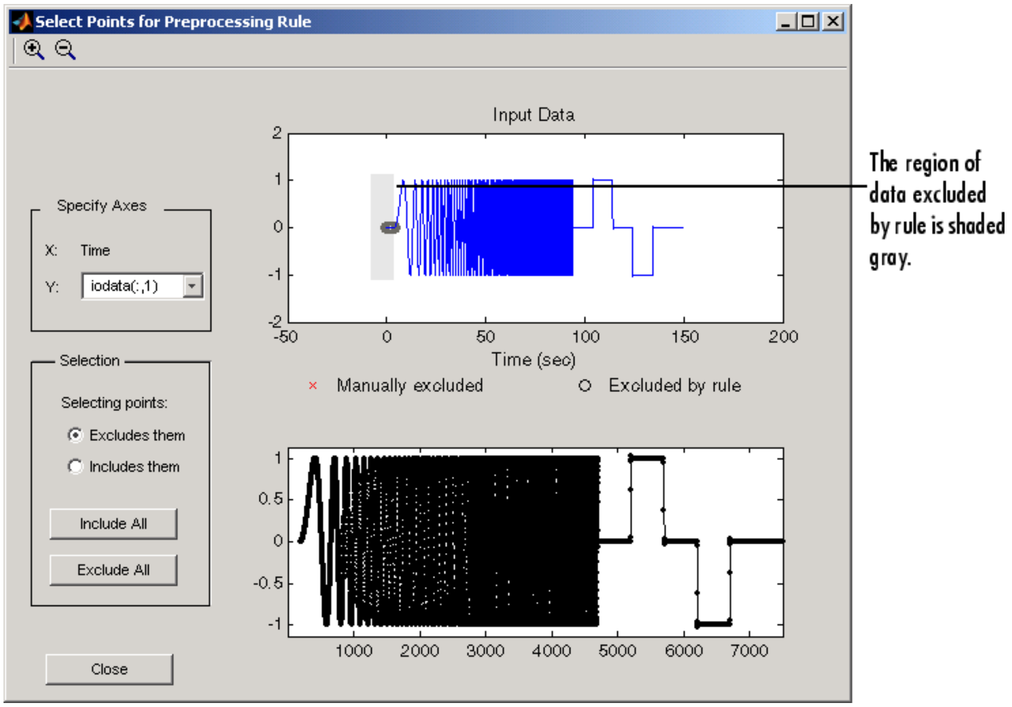
- "Flatlines" on page 1-24

Upper and Lower Bounds. Select the **Bounds** check box to activate upper and lower bound exclusion. Enter numbers in the **Exclude X** and **Exclude Y** fields for upper and lower bound exclusion. By default, the exclusion rule is to include the boundary values, but you can use the menu to exclude the boundaries as well.

MATLAB Expressions. Use the **MATLAB expression** field to enter any mathematical expression using MATLAB code. Use `x` as the variable name in your expression for the data being tested.

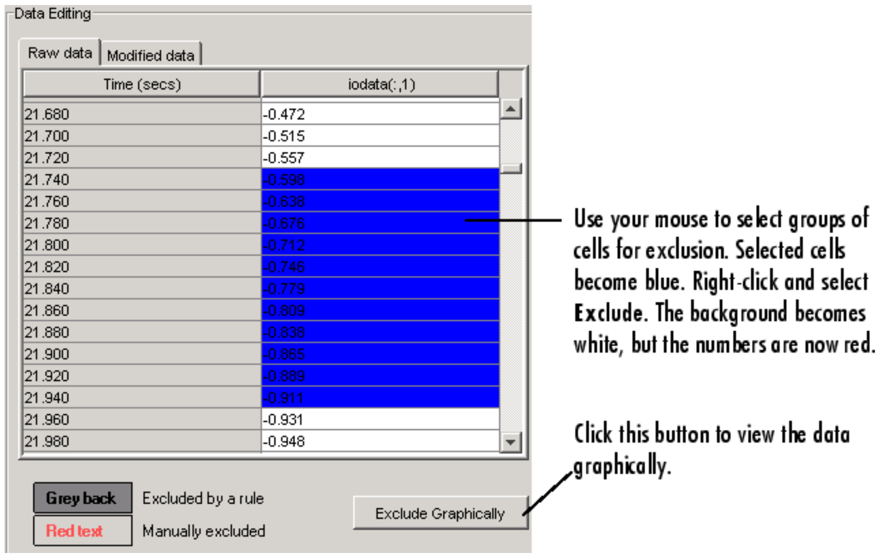
Flatlines. If you have areas of your data set where the data is constant, providing no new information, then you can choose to exclude those data points as flatlines. The **Window length** field sets the minimum number of constant data points required to define the area as a flatline.

Example of Rule Exclusion. This figure shows data with a region of the *x*-axis excluded.



Using the Data Table to Select Data Samples

The **Data Editing** table lists both the raw data set and the modified data that you create.

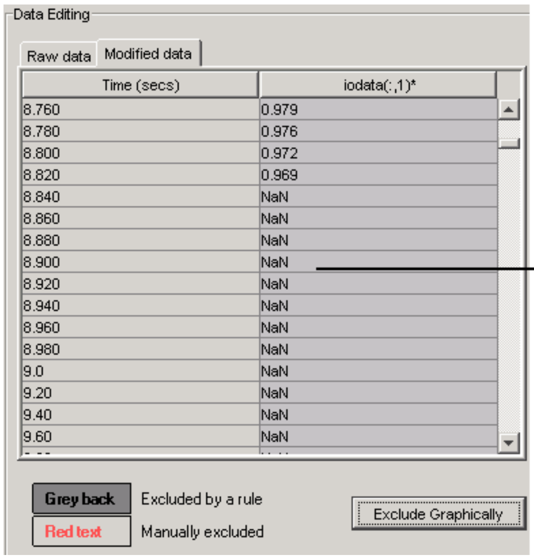


Time (secs)	iodata(:,1)
21.680	-0.472
21.700	-0.515
21.720	-0.557
21.740	-0.598
21.760	-0.638
21.780	-0.676
21.800	-0.712
21.820	-0.746
21.840	-0.779
21.860	-0.809
21.880	-0.838
21.900	-0.865
21.920	-0.889
21.940	-0.911
21.960	-0.931
21.980	-0.948

Use your mouse to select groups of cells for exclusion. Selected cells become blue. Right-click and select **Exclude**. The background becomes white, but the numbers are now red.

Click this button to view the data graphically.

There are two tabs in the **Data Editing** pane: **Raw data** and **Modified data**. The **Raw Data** pane shows the working copy of the data. For example, if you exclude rows of data in the **Raw data** pane, the corresponding rows of numbers become red in this table. By default the **Modified data** pane represents the rows you removed by inserting NaNs.



Data Editing

Raw data | Modified data

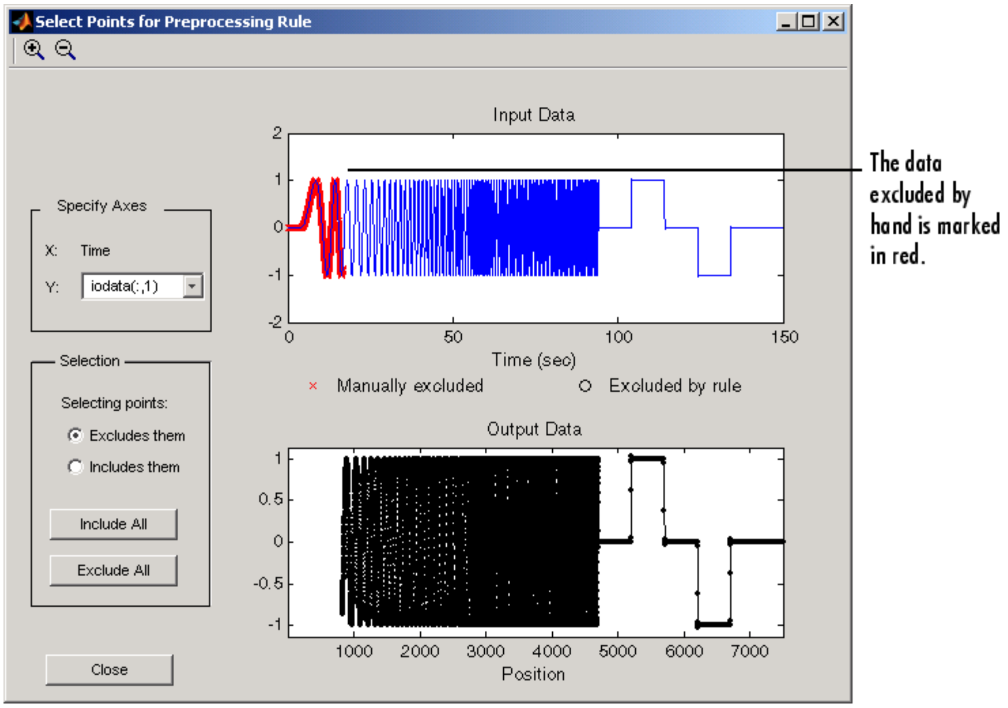
Time (secs)	iodata(:,1)*
8.760	0.979
8.780	0.976
8.800	0.972
8.820	0.969
8.840	NaN
8.860	NaN
8.880	NaN
8.900	NaN
8.920	NaN
8.940	NaN
8.960	NaN
8.980	NaN
9.0	NaN
9.20	NaN
9.40	NaN
9.60	NaN

Grey back	Excluded by a rule
Red text	Manually excluded

By default, data that you excluded from the **Raw Data** table is represented by NaNs in the **Modified Data** table. If you choose to interpolate or remove missing data, the results of that action are shown in the **Modified Data** table.

In the **Modified data** pane, you can choose to remove the excluded data completely or interpolate it. See "Handling Missing Data" on page 1-16 for more information.

After you select data for exclusion, you can view it graphically by clicking **Exclude Graphically**.



As you make changes in the **Data Editing** pane, they immediately appear in the Select Points for Preprocessing Rule window, and vice versa.

Adding Preprocessed Data Sets to an Estimation Project

After you preprocess the data using the techniques described in "Ways to Preprocess Data Using the Data Preprocessing Tool" on page 1-14, you can add the data set to an estimation project either by overwriting an existing data set or creating a new data set.

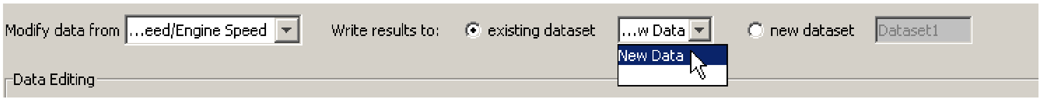
- "Overwriting an Existing Data Set" on page 1-29
- "Creating a New Data Set" on page 1-30

Overwriting an Existing Data Set

To overwrite an existing data set with the preprocessed data:

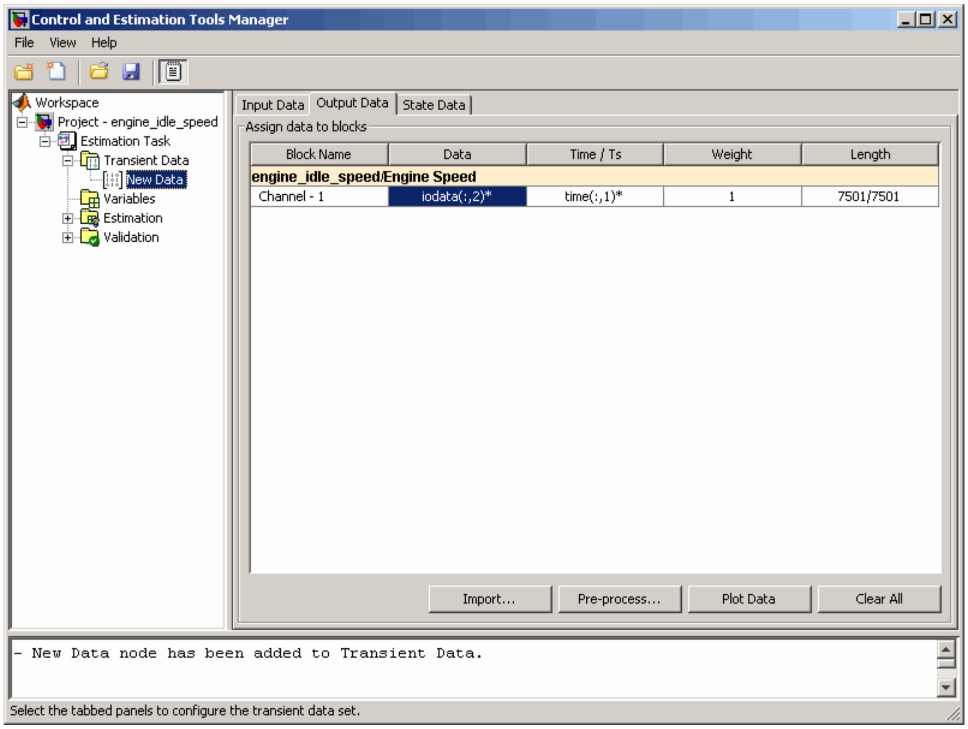
1. In the **Write results to** area of the Data Preprocessing Tool GUI, select the **existing dataset** option.

2. Choose the data set you want to overwrite from the drop-down list.


Modify data from ...eed/Engine Speed
Write results to: existing dataset ...w Data
New Data
new dataset Dataset1
Data Editing

3. Click **Add**.

   This action overwrites the selected data set with the modified data in the Control and Estimation Tools Manager GUI.



Block Name	Data	Time / Ts	Weight	Length
engine_idle_speed/Engine Speed				
Channel - 1	iodata(:,2)*	time(:,1)*	1	7501/7501

```
- New Data node has been added to Transient Data.
```

Select the tabbed panels to configure the transient data set.

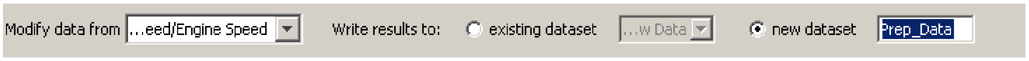
Tip You can export the preprocessed data to the MATLAB Workspace, as described in "Exporting Prepared Data to the MATLAB Workspace" on page 1-31.

Creating a New Data Set

If you do not want to overwrite an existing data set with the preprocessed data, as described in "Overwriting an Existing Data Set" on page 1-29, you can create a new data set for the preprocessed data:

1 In the **Write results to** area of the Data Preprocessing Tool GUI, select the **new dataset** option.

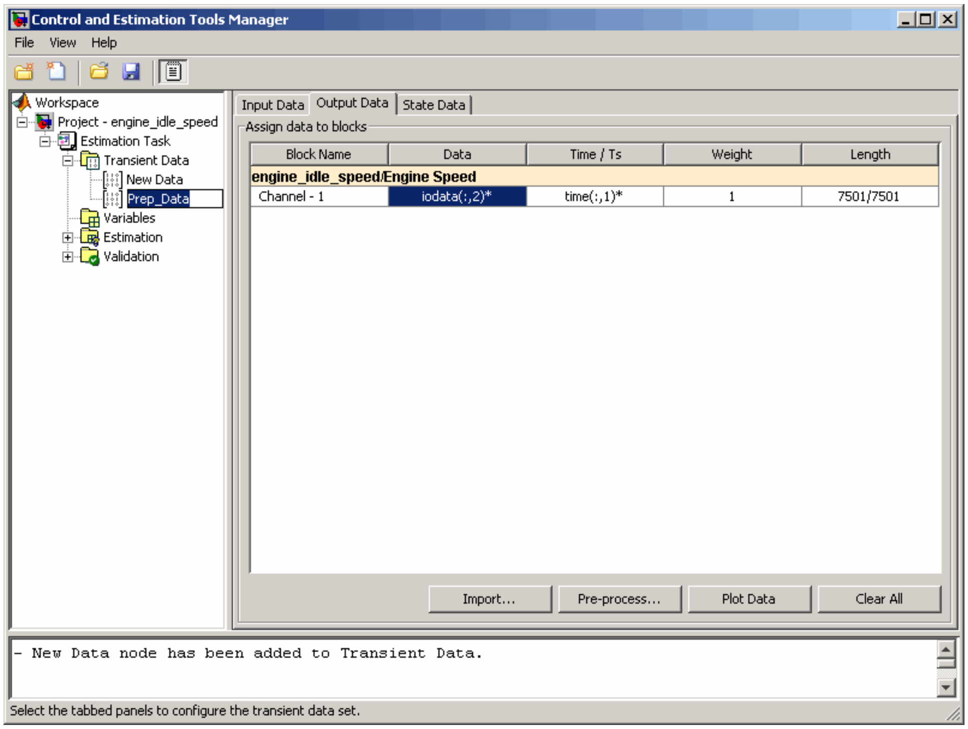
2 Specify the name of the data set in the adjacent field.



Modify data from ...eed/Engine Speed   Write results to: existing dataset ...w Data   [✓ new dataset] Prep_Data

3 Click **Add**.

This action adds a new data node in the Control and Estimation Tools Manager GUI containing the modified data.



Block Name	Data	Time / Ts	Weight	Length
engine_idle_speed/Engine Speed				
Channel - 1	iodata(:,2)*	time(:,1)*	1	7501/7501

```
- New Data node has been added to Transient Data.
```

Select the tabbed panels to configure the transient data set.

Tip You can export the preprocessed data to the MATLAB Workspace, as described in "Exporting Prepared Data to the MATLAB Workspace" on page 1-31.

Exporting Prepared Data to the MATLAB Workspace

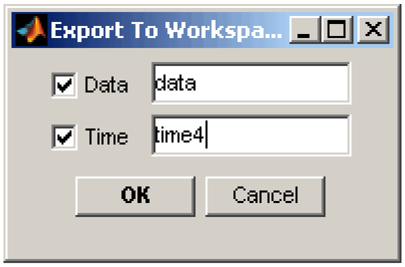
After you add the preprocessed data to an estimation project, as described in "Adding Preprocessed Data Sets to an Estimation Project" on page 1-28, you can export the data set to the MATLAB Workspace. You can use the data to further prepare it or estimate parameters using the data.

1 In the **Transient Data** node of the Control and Estimation Tools Manager GUI, select the node containing the prepared data set.

2 Right-click the table **Data** cell containing the data that you want to export, and select **Export**.

The Export to Workspace dialog box opens.

3 Specify the MATLAB variable names for the prepared data and the corresponding time vector in the **Data** and **Time** fields, respectively.



☑ Data: data
☑ Time: time4

4 Click **OK**.

The resulting MATLAB variables `data` and `time4` appear in the MATLAB Workspace browser.

Estimating Model Parameters

Overview of Parameter Estimation

When you estimate model parameters, Simulink Design Optimization software compares the measured data with data generated by a Simulink model. Using optimization techniques, the software estimates the parameter and (optionally) initial conditions of states to minimize a user-selected cost function. The cost function typically calculates a least-square error between the empirical and model data signals.

After you import and preprocess the estimation data, as described in "Importing Data into the GUI" on page 1-5 and "Preprocessing Data in the GUI" on page 1-14, follow these steps to estimate model parameters:

1. "Creating an Estimation Task" on page 2-3
2. "Specifying Data for Parameter Estimation" on page 2-4
3. "Specifying Parameters to Estimate" on page 2-6
4. "Specifying Initial States" on page 2-16
5. "Selecting Views for Plotting" on page 2-18
6. "Specifying Estimation Options" on page 2-22
7. "Performing Estimation" on page 2-33
8. "Performing Validation" on page 2-40

Configuring Parameter Estimation in the GUI

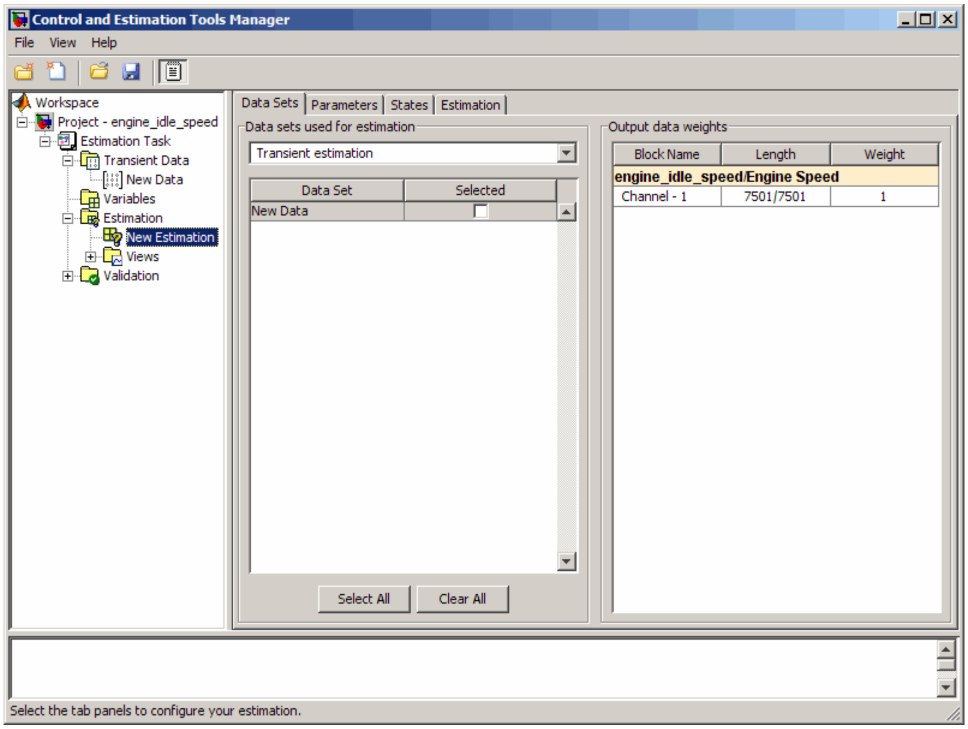
In this section...

Creating an Estimation Task

This section describes how to use the GUI to estimate parameters. After you import the transient data, as described in "Importing Data into the GUI" on page 1-5, you must create an estimation task and configure the estimation settings. To create a container that stores the estimation settings:

1. In the Control and Estimation Tools Manager, right-click the **Estimation** node in the workspace directory tree and select **New**.
2. Select the **New Estimation** node.

   The Control and Estimation Tools Manager now resembles the next figure.



Data sets used for estimation: Transient estimation

Data Set	Selected
New Data	☐

Output data weights

Block Name	Length	Weight
engine_idle_speed/Engine Speed		
Channel - 1	7501/7501	1

Select the tab panels to configure your estimation.

Specifying Data for Parameter Estimation

- “Prerequisite for Specifying Data” on page 2-4
- “How to Specify Data in the GUI” on page 2-5

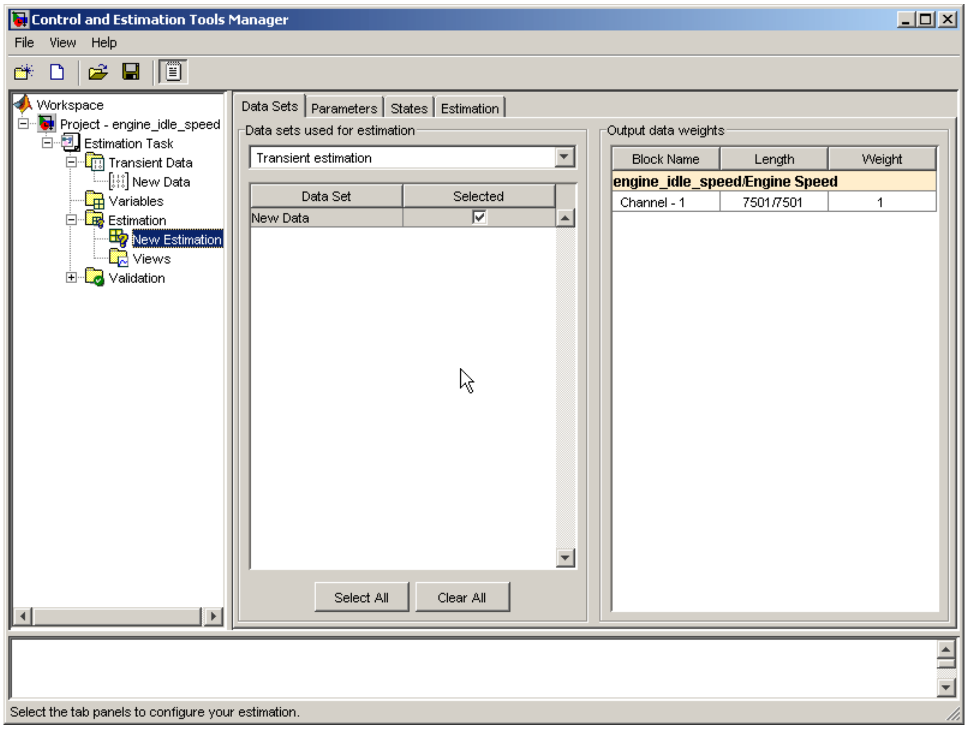
Prerequisite for Specifying Data

To specify a data set for estimation, you must have already imported the data in the GUI and created an Estimation Task, as described in “Creating an Estimation Task” on page 2-3. If your data contains noise or outliers, you must also preprocess the data, as described in “Preprocessing Data in the GUI” on page 1-14.

How to Specify Data in the GUI

After you select the **New Estimation** node, the **Data Sets** tab appears. Here you select the data set that you want to use in the estimation.

Select the **Selected** check box to the right of the **New Data** data set.



Data Set	Selected
New Data	✓

Block Name	Length	Weight
engine_idle_speed/Engine Speed		
Channel - 1	7501/7501	1

Select the tab panels to configure your estimation.

Note If you imported multiple data sets, you can select them for estimation by selecting the check box to the right of each desired data set. When using several data sets, you increase the estimation precision. However, you also increase the number of required simulations: for N parameters and M data sets, there are M*(2N+1) simulations per iteration.

Then, specify the weight of each output from this model by setting the **Weight** column in the **Output data weights** table.

The relative weights are used to place more or less emphasis on specific output variables. The following are a few guidelines for specifying weights:

- Use less weight when an output is noisy.
- Use more weight when an output strongly affects parameters.
- Use more weight when it is more important to accurately match this model output to the data.

Specifying Parameters to Estimate

Choosing Which Parameters to Estimate First

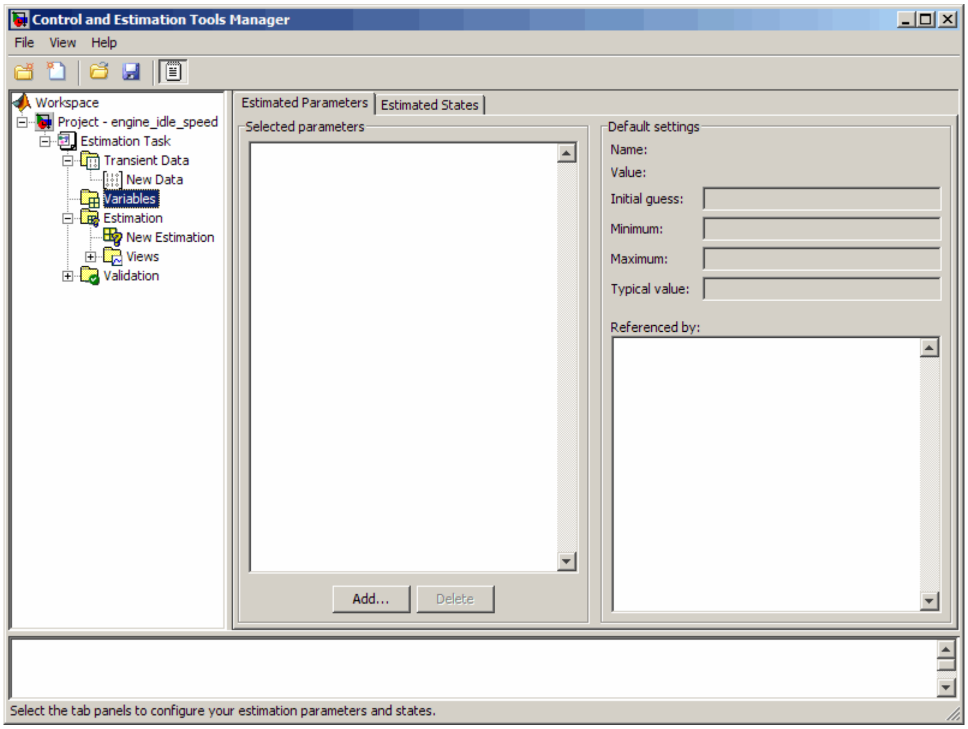
Estimating model parameters is an iterative process. Often, it is more practical to estimate a small group of parameters and use the final estimated values as a starting point for further estimation of parameters that are trickier. When you have a large number of parameters to estimate, select the parameters that influence the output the most to be estimated first. Making these sorts of choices involves experience, intuition, and a solid understanding of the strengths and limitations of your Simulink model.

After you estimate a subset of parameters and validate the estimated parameters, select the remaining parameters for estimation.

How to Specify Parameters for Estimation in the GUI

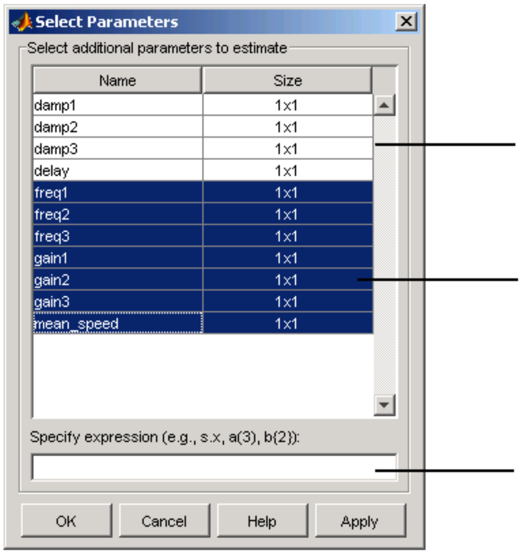
To select parameters for estimation:

1. In the Control and Estimation Tools Manager, select the **Variables** node in the workspace directory tree to open the **Estimated Parameters** pane.



Select the tab panels to configure your estimation parameters and states.

2 In the **Estimated Parameters** pane, click **Add** to open the Select Parameters dialog box.


Select Parameters
Select additional parameters to estimate
Specify expression (e.g., s.x, a(3), b{2}):
OK
Cancel
Help
Apply

Name	Size
damp1	1x1
damp2	1x1
damp3	1x1
delay	1x1
freq1	1x1
freq2	1x1
freq3	1x1
gain1	1x1
gain2	1x1
gain3	1x1
mean_speed	1x1

By default, the Select Parameters dialog box looks at all variables in the model workspace and the MATLAB workspace that are used by the model.

List of parameters

Use your mouse to select data. To select adjacent parameters, hold down the **Shift** key while clicking the first and last parameter in the selection. To select nonadjacent parameters, hold down the **Ctrl** key while clicking each parameter.

Use the text field to get the parameters contained in either a Simulink parameter object, MATLAB array, structure, or cell array. Note that you cannot use mathematical expressions such as `x + 5`.

The dialog box lists all the variables in the model workspace and the MATLAB workspace that the model uses. You can use the mouse to select the parameters to estimate.

You can also enter parameters, separated by commas, in the **Specify expression** field of the Select Parameters dialog box. The parameters can be stored in one of the following:

- Simulink software parameter object

  Example: For a Simulink parameter object `k`, type `k.value`.

- Structure

  Example: For a structure S, type S.*fieldname* (where *fieldname* represents the name of the field that contains the parameter).

- Cell array

  Example: Type `C{1}` to select the first element of the C cell array.

- MATLAB array

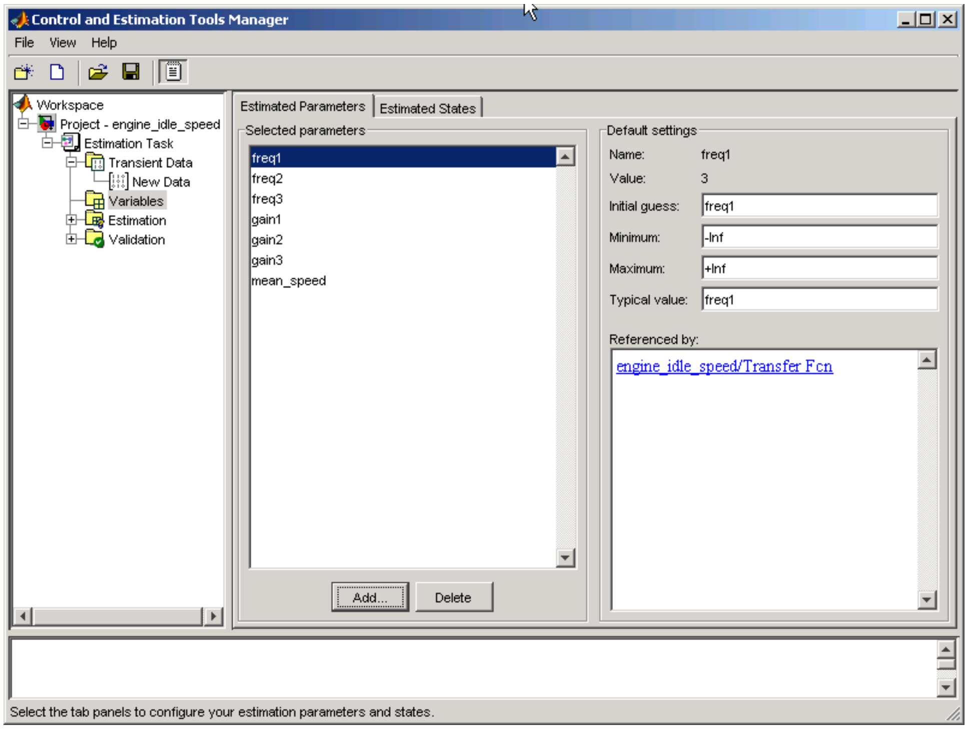
  Example: Type `a(1:2)` to select the first column of a 2-by-2 array called `a`.

Sometimes, models have parameters that are not explicitly defined in the model itself. For example, a gain `k` could be defined in the MATLAB workspace as `k=a+b`, where `a` and `b` are not defined in the model but `k` is used. To add these independent parameters to the Select Parameters dialog box, see "Specifying Parameter Dependency" on page 2-13.

3. Select the last seven parameters: `freq1`, `freq2`, `freq3`, `gain1`, `gain2`, `gain3`, and `mean_speed`, and then click **OK**.

Note You need not estimate the parameters selected here all at once. You can first select all the parameters that you are interested in, and then later select the ones to estimate as described in the next step.

The Control and Estimation Tools Manager now resembles the next figure.



Selected parameters
freq1
freq2
freq3
gain1
gain2
gain3
mean_speed

Default settings

Name: freq1
Value: 3
Initial guess: freq1
Minimum: -Inf
Maximum: +Inf
Typical value: freq1

Referenced by:
engine_idle_speed/Transfer Fcn

Select the tab panels to configure your estimation parameters and states.

To learn how to specify the settings in the **Default settings** area of the pane, see "Specifying Initial Guesses and Upper/Lower Bounds" on page 2-11.

4 In the **New Estimation** node of the Control and Estimation Tools Manager GUI, select the **Parameters** tab . In this pane, you select which parameters to estimate and the range of values for the estimation.

   a Select the parameters you want to estimate by selecting the check box in the **Estimate** column.

   b Enter initial values for your parameters in the **Initial Guess** column.

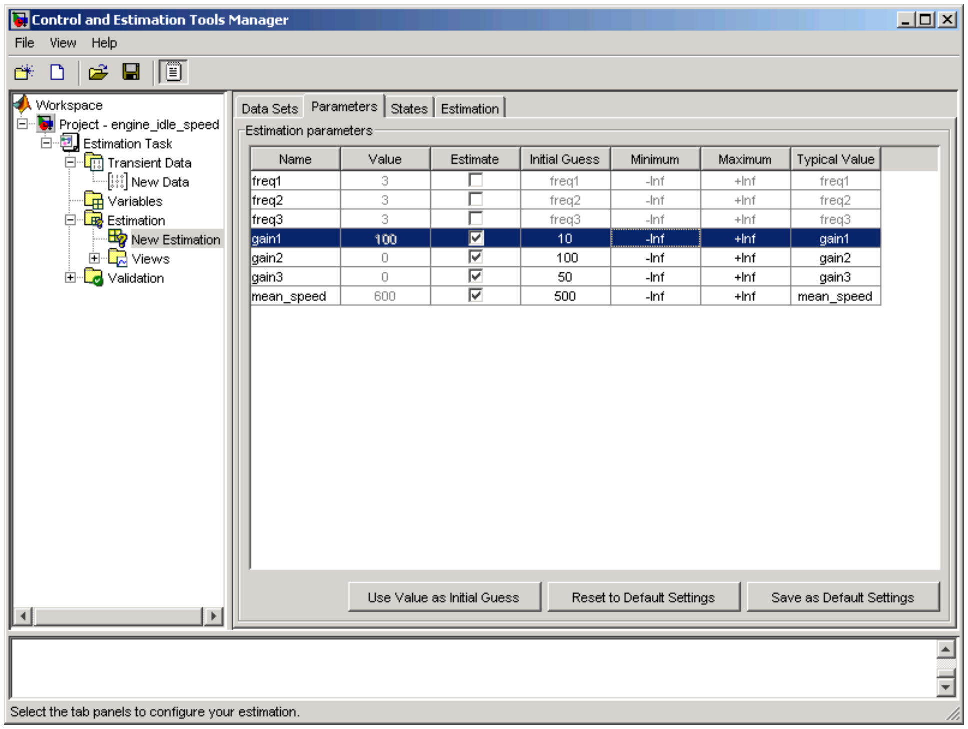
   The default values in the **Minimum** and **Maximum** columns are `-Inf` and `+Inf`, respectively, but you can select any range you want. For more information, see "Specifying Initial Guesses and Upper/Lower Bounds" on page 2-11.

   Note When you specify the **Minimum** and **Maximum** values for the parameters here, it does not affect your settings in the **Variables** node. You make these choices on a per estimation basis. You can move data to and from the **Variables** node into the **Estimation** node.

For this example, select `gain1`, `gain2`, `gain3` and `mean_speed` for estimation and set `gain1` to 10, `gain2` to 100, `gain3` to 50, and `mean_speed` to 500. Alternatively, use any initial values you like.

If you have good reason to believe a parameter lies within a finite range, it is usually best not to use the default minimum and maximum values. Often, there are computational advantages in specifying finite bounds if you can. It can be very important to specify lower and upper bounds. For example, if a parameter specifies the weight of a part, be sure to specify `0` as the absolute lower bound if better knowledge is unavailable.

The Control and Estimation Tools Manager now resembles the next figure.

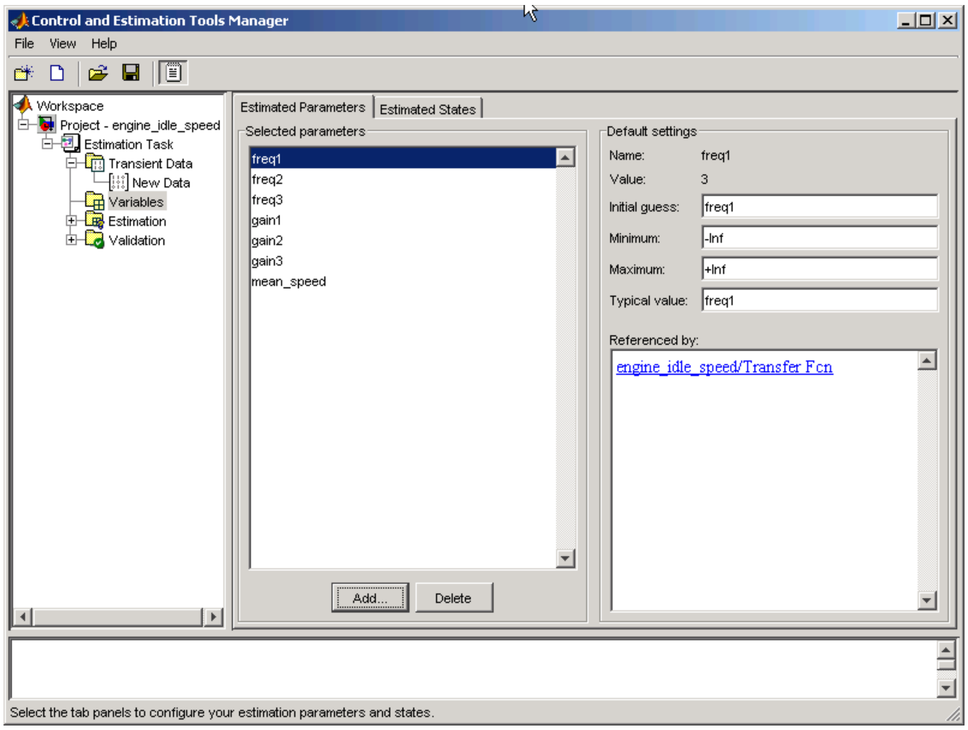


Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
freq1	3	☐	freq1	-Inf	+Inf	freq1
freq2	3	☐	freq2	-Inf	+Inf	freq2
freq3	3	☐	freq3	-Inf	+Inf	freq3
gain1	100	☑	10	-Inf	+Inf	gain1
gain2	0	☑	100	-Inf	+Inf	gain2
gain3	0	☑	50	-Inf	+Inf	gain3
mean_speed	600	☑	500	-Inf	+Inf	mean_speed

Select the tab panels to configure your estimation.

Specifying Initial Guesses and Upper/Lower Bounds

After you select parameters for estimation in the **Variables** node of the Control and Estimation Tools Manager GUI, the **Estimated Parameters** tab in the Control and Estimation Tools Manager looks like the following figure.



Selected parameters: freq1, freq2, freq3, gain1, gain2, gain3, mean_speed

Default settings

Name:	freq1
Value:	3
Initial guess:	freq1
Minimum:	-Inf
Maximum:	+Inf
Typical value:	freq1

Referenced by: engine_idle_speed/Transfer Fcn

Select the tab panels to configure your estimation parameters and states.

For each parameter, use the **Default settings** pane to specify the following:

- **Initial guess** — The value the estimation uses to start the process.
- **Minimum** — The smallest allowable parameter value. The default is `-Inf`.
- **Maximum** — The largest allowable parameter value. The default is `+Inf`.
- **Typical value** — The average order of magnitude. If you expect your parameter to vary over several orders of magnitude, enter the number that specified the average order of magnitude you expect. For example, if your initial guess is 10, but you expect the parameter to vary between 10 and 1000, enter 100 (the average of the order of magnitudes) for the typical value.

You use the typical value in two ways:

- To scale parameters with radically different orders of magnitude for equal emphasis during the estimation. For example, try to select the typical values so that

  $$\frac{\text{anticipated value}}{\text{typical value}} \cong 1$$

  or

  $$\frac{\text{initial value}}{\text{typical value}} \cong 1$$

- To put more of less emphasis on specific parameters. Use a larger typical value to put more emphasis on a parameter during estimation.

Specifying Parameter Dependency

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 specify independent parameters for estimation:

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 Select Parameters dialog box. Add these parameters to the list of parameters to be estimated.

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

For an example of how to specify independent parameters, see "Example: Specifying Independent Parameters for Estimation" on page 2-14.

Example: Specifying Independent Parameters for Estimation

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

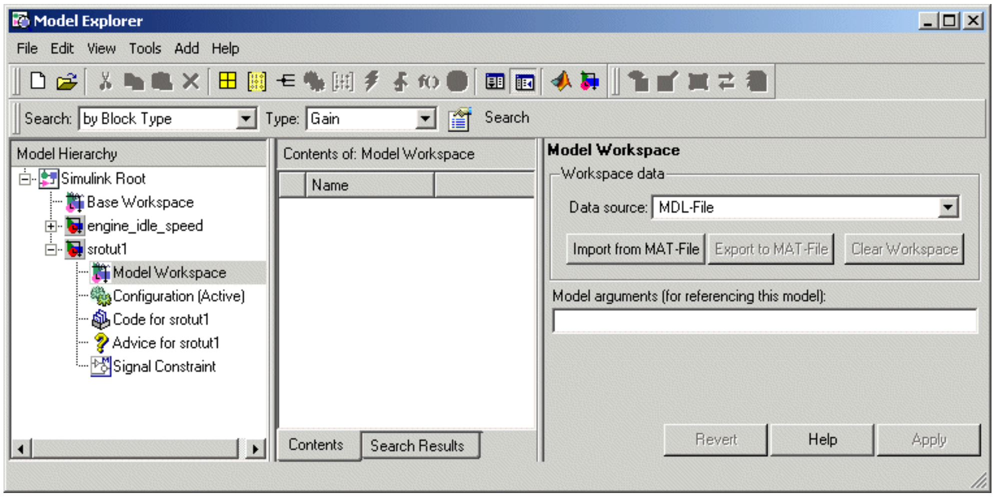
1 At the MATLAB prompt, type

```
srotut1
```

This opens the `srotut1` model window.

2 Select **View > Model Explorer** from the `srotut1` window to open the Model Explorer window.

3 In the Model Hierarchy tree, select the **srotut1 > Model Workspace** node.

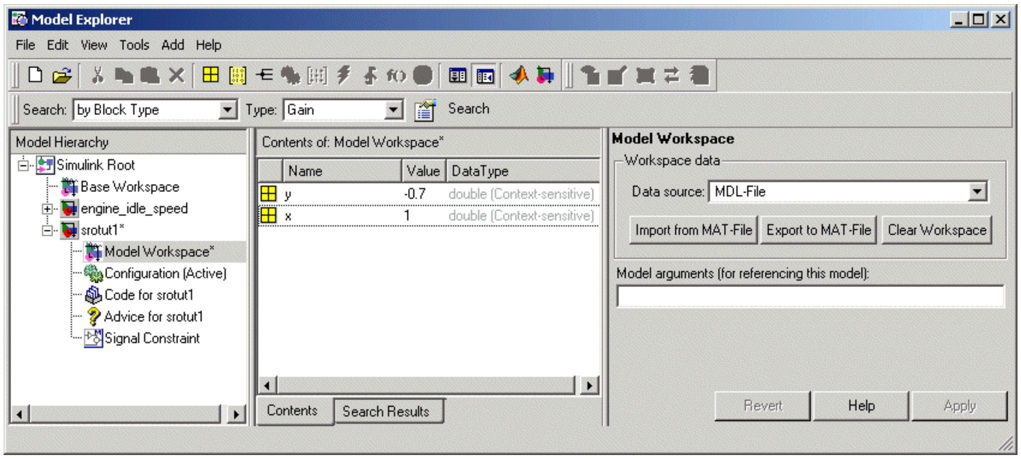


Search: by Block Type
Type: Gain
Data source: MDL-File
Model arguments (for referencing this model):

4 Select **Add > MATLAB Variable** to add a new variable to the model workspace. A new variable with a default name `Var` appears in the **Contents of: Model Workspace** pane.

5 Double-click `Var` to make it editable and change the variable name to `x`. Edit the initial `Value` to `1`.

6 Repeat steps 4 and 5 to add a variable `y` with an initial value of `-0.7`. The Model Explorer window should resemble the following figure.



Search: by Block Type Type: Gain

Contents of: Model Workspace*

Name	Value	DataType
y	-0.7	double (Context-sensitive)
x	1	double (Context-sensitive)

Model Workspace
Workspace data
Data source: MDL-File
Model arguments (for referencing this model):

7 To add the Simulation Start function defining the relationship between `Kint` and the independent parameters `x` and `y`, select **File > Model Properties** in the Simulink model for `srotut1`.

8 In the Model Properties window, click the **Callbacks** tab.

9 To enter a Simulation start function in **StartFcn***, type the name of a new M-file, for example, `srotut1_start`.

10 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 resemble the following:

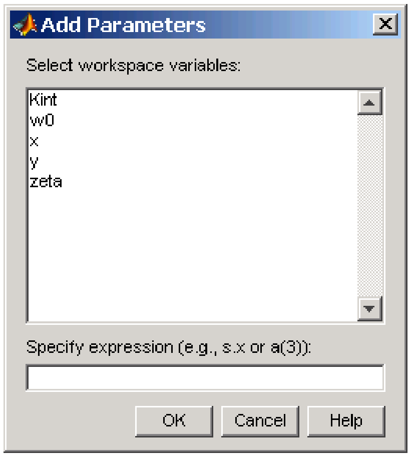
```
wks = get_param(gcs, 'ModelWorkspace')
x = wks.evalin('x')
```

```
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`.

---

11 When you select parameters for estimation in the **Variables** node, `x` and `y` appear in the Select Parameters dialog box.



Select workspace variables:

Kint
w0
x
y
zeta

Specify expression (e.g., s.x or a(3)):

Specifying Initial States

- "When to Specify Initial States Versus Estimate Initial States" on page 2-16
- "How to Specify Initial States in the GUI" on page 2-17

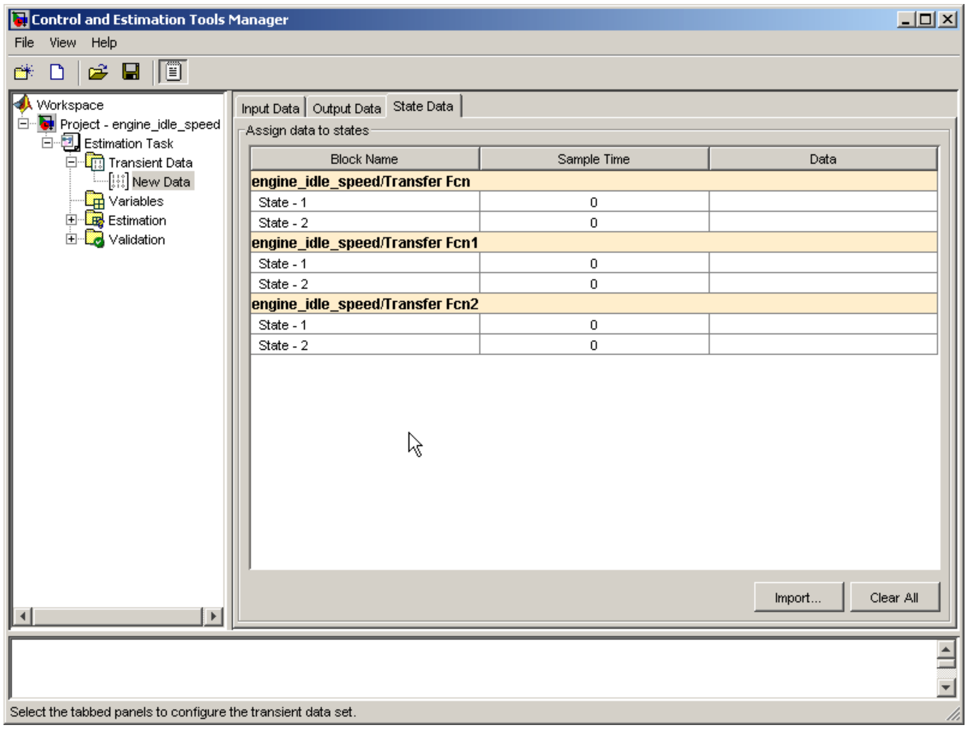
When to Specify Initial States Versus Estimate Initial States

Often, sets of measured data are collected at various times and under different initial conditions. When you estimate model parameters using one data set and subsequently run another estimation with a second data set, your parameter values may not match. Given that the Simulink Design Optimization software attempts to find constant values for parameters, this is clearly a problem.

You can estimate the initial conditions using procedures that are similar to those you use to estimate parameters. You can then use these initial condition estimates as a basis for estimating parameters for your Simulink model. The Control and Estimation Tools Manager has an **Estimated States** pane that lists the states available for initial condition estimation. To learn how to estimate initial states, see "Estimating Initial States" on page 2-62.

How to Specify Initial States in the GUI

After you select parameters for estimation, as described in "Specifying Parameters to Estimate" on page 2-6, you can specify initial conditions of states in your model. By default, the estimation uses initial conditions specified in the Simulink model. If you want to specify initial conditions other than the defaults, use the **State Data** tab. You can select the **State Data** tab in the **New Data** node under the **Transient Data** node in the workspace directory tree.



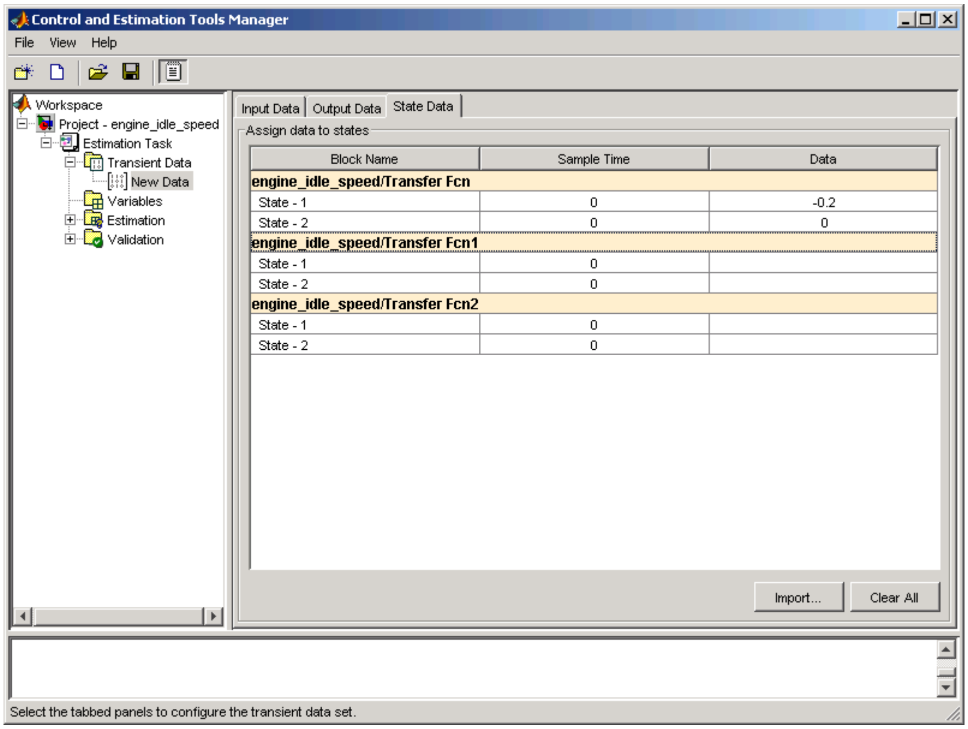
Block Name	Sample Time	Data
engine_idle_speed/Transfer Fcn		
State - 1	0	
State - 2	0	
engine_idle_speed/Transfer Fcn1		
State - 1	0	
State - 2	0	
engine_idle_speed/Transfer Fcn2		
State - 1	0	
State - 2	0	

Select the tabbed panels to configure the transient data set.

To specify the initial condition of a state for the `engine_idle_speed` model:

1 Select the **Data** cell associated with the state.

2 Enter the initial conditions. In this example, enter `-0.2 for` **State - 1** of the **engine_idle_speed/Transfer Fcn**. For **State - 2**, enter `0`.



Block Name	Sample Time	Data
engine_idle_speed/Transfer Fcn		
State - 1	0	-0.2
State - 2	0	0
engine_idle_speed/Transfer Fcn1		
State - 1	0	
State - 2	0	
engine_idle_speed/Transfer Fcn2		
State - 1	0	
State - 2	0	

Select the tabbed panels to configure the transient data set.

Selecting Views for Plotting

- "Types of Plots" on page 2-18
- "Basic Steps for Creating Plots" on page 2-19

Types of Plots

You can choose the plot type from the **Plot Type** drop-down list. The following types of plots are available for viewing and evaluating the estimation:

- `Cost function` — Plot the cost function values.
- `Measured and simulated` — Plot empirical data against simulated data.
- `Parameter sensitivity` — Plot the rate of change of the cost function as a function of the change in the parameter. That is, plot the derivative of the cost function with respect to the parameter being varied.
- `Parameter trajectory` — Plot the parameter values as they change.
- `Residuals` — Plot the error between the experimental data and the simulated output.

Basic Steps for Creating Plots

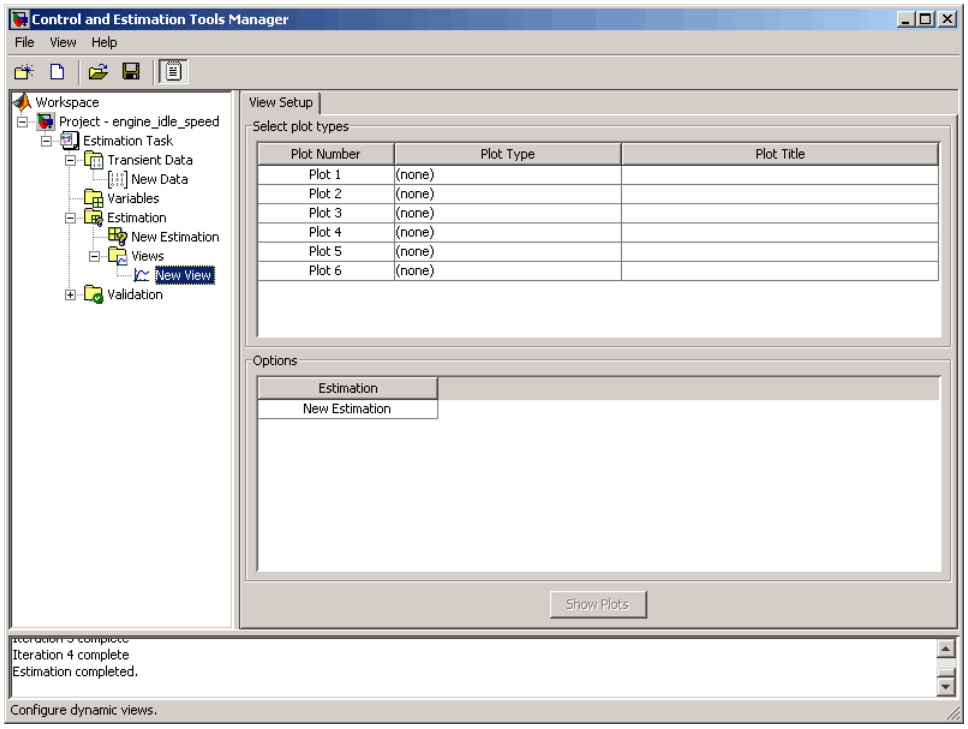
Before you begin estimating the parameters, you must create the plots for viewing the progress of the estimation.

Note An estimation must be created before creating views. Otherwise, the **Options** table will be empty. To learn more, see "Creating an Estimation Task" on page 2-3.

To create plots for viewing the estimation progress, follow the steps below:

1 Right-click the **Views** node in the Control and Estimation Tools Manager and select **New**.

2 In the workspace directory tree, select **New View** to open the **View Setup** pane.



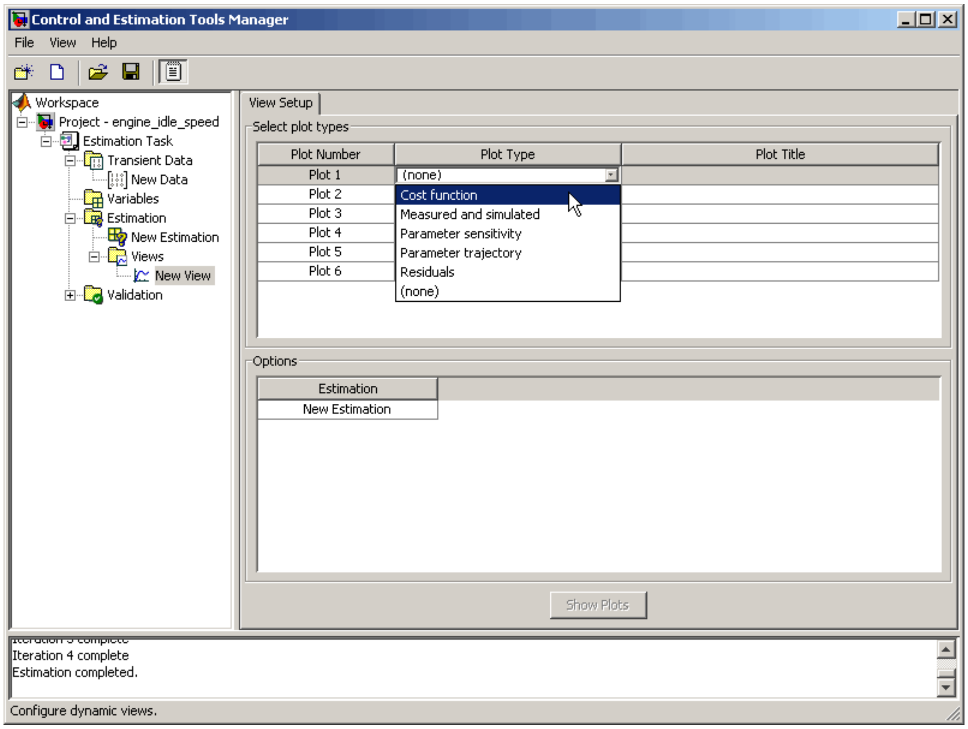
Plot Number	Plot Type	Plot Title
Plot 1	(none)	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

Estimation
New Estimation

```
Iteration 3 complete
Iteration 4 complete
Estimation completed.
```

Configure dynamic views.

3 In the **Select plot types** table, select the **Plot Type** from the drop-down list. In this example, select `Cost function`.



Plot Number	Plot Type	Plot Title
Plot 1	(none)	
Plot 2		
Plot 3		
Plot 4		
Plot 5		
Plot 6		

Cost function
Measured and simulated
Parameter sensitivity
Parameter trajectory
Residuals
(none)

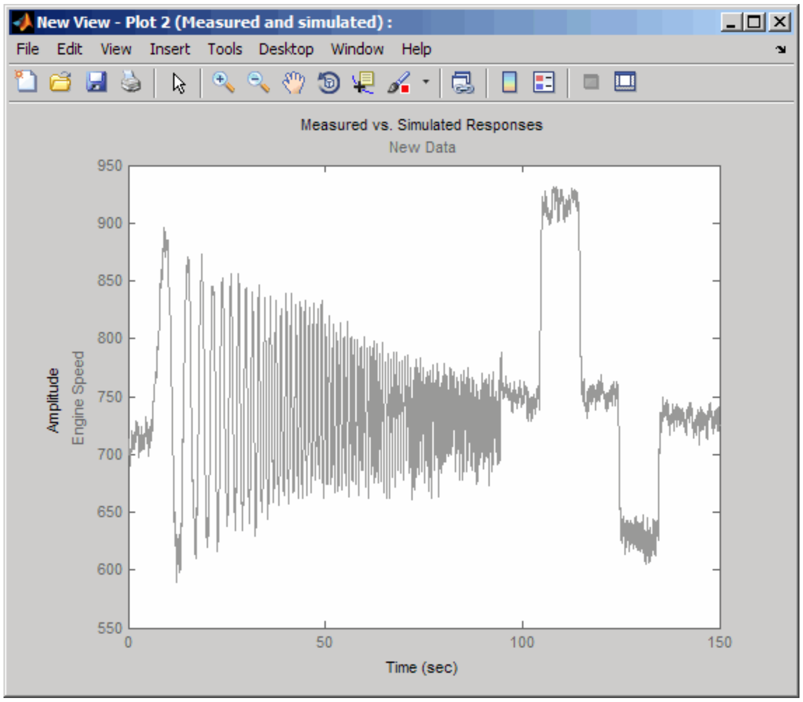
Estimation
New Estimation

```
Iteration 3 complete
Iteration 4 complete
Estimation completed.
```

4 Select `Measured and simulated` as the **Plot Type** for `Plot 2`. This plot will be used in "Performing Validation" on page 2-40.

5 In the **Options** area, select the check-box for both `Plot 1` and `Plot 2`.

6 Click **Show Plots**. This displays an empty cost function plot and a plot of the measured data.


New View - Plot 2 (Measured and simulated) :
File Edit View Insert Tools Desktop Window Help
Measured vs. Simulated Responses
New Data
Amplitude
Engine Speed
950
900
850
800
750
700
650
600
550
0
50
100
150
Time (sec)

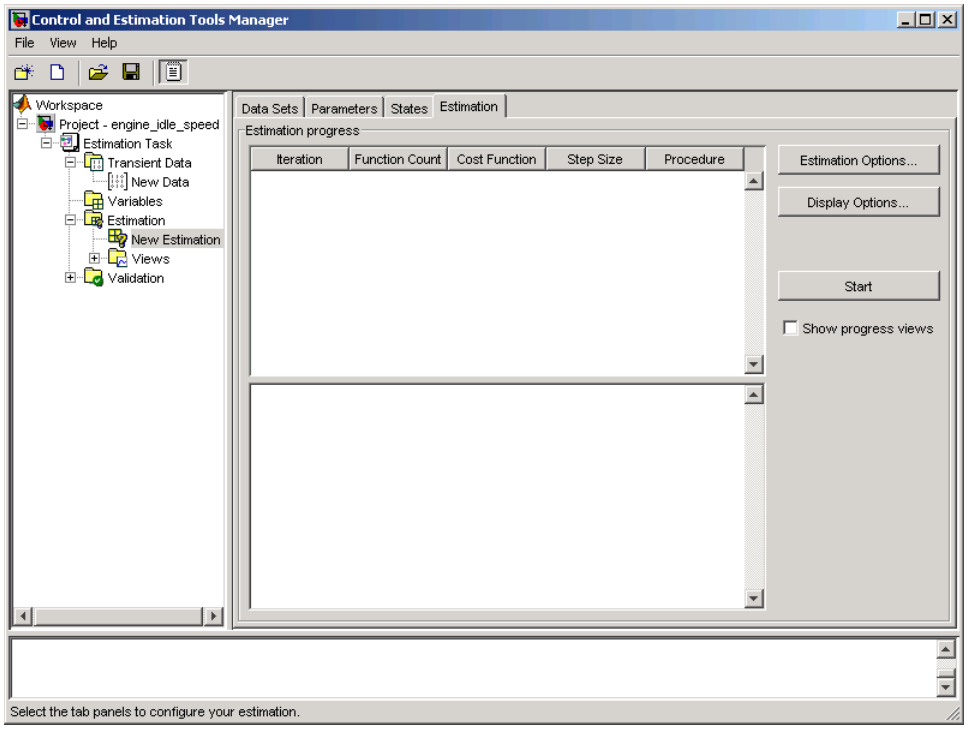
When you perform the estimation, the plot updates automatically.

Specifying Estimation Options

- "Accessing Estimation Options" on page 2-23
- "Specifying Goodness of Fit Criteria (Cost Function)" on page 2-24
- "Supported Estimation Algorithms" on page 2-24
- "Selecting Optimization Termination Options" on page 2-25
- "Selecting Additional Optimization Options" on page 2-26
- "How to Specify Estimation Options in the GUI" on page 2-26

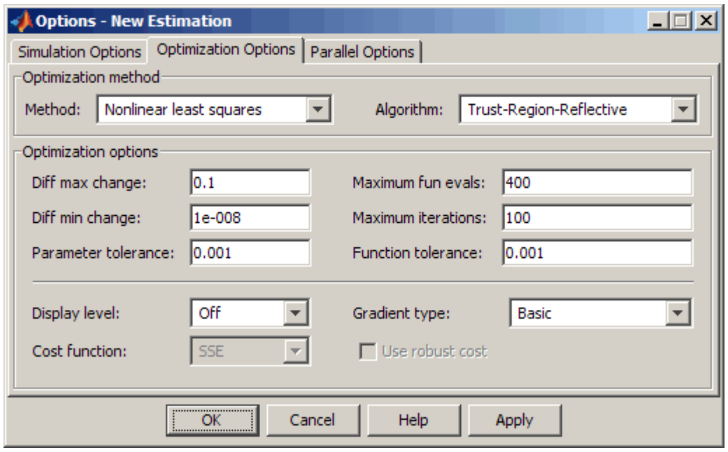
Accessing Estimation Options

In the **New Estimation** node in the workspace directory tree, click the **Estimation** tab.



Select the tab panels to configure your estimation.

Click **Estimation Options**. This action opens the Options- New Estimation dialog box where you can specify the algorithm, algorithm options and cost function for the estimation.

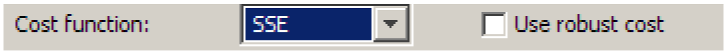


Method: Nonlinear least squares
Algorithm: Trust-Region-Reflective
Diff max change: 0.1
Maximum fun evals: 400
Diff min change: 1e-008
Maximum iterations: 100
Parameter tolerance: 0.001
Function tolerance: 0.001
Display level: Off
Gradient type: Basic
Cost function: SSE

The following sections describe the algorithm settings and cost function:

Specifying Goodness of Fit Criteria (Cost Function)

The *cost function* is a function that estimation algorithms attempt to minimize. You can specify the cost function at the bottom of the **Optimization options** area.

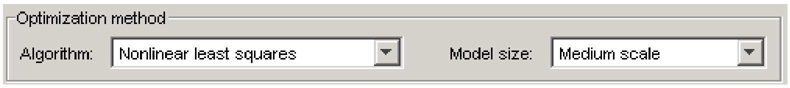


You have the following options when selecting a cost function:

- **Cost function** — The default is `SSE` (sum of squared errors), which uses a least-squares approach. You can also use `SAE`, the sum of absolute errors.
- **Use robust cost** — Makes the optimizer use a robust cost function instead of the default least-squares cost. This is useful if the experimental data has many outliers, or if your data is noisy.

Supported Estimation Algorithms

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



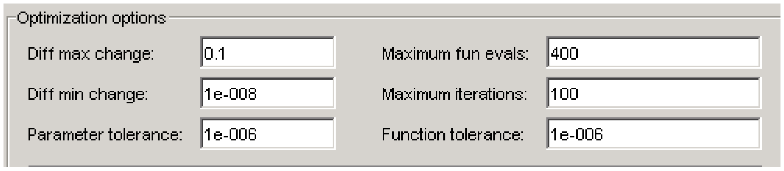
For the **Algorithm** parameter, the four options are

- `Gradient descent` — Uses the Optimization Toolbox™ function `fmincon` to optimize the response signal subject to the constraints
- `Nonlinear least squares` — Uses a nonlinear least squares, lsqnonlin, optimization algorithm.
- `Pattern search` — Uses an advanced pattern search algorithm. This option requires Genetic Algorithm and Direct Search Toolbox™ software.
- `Simplex search` — Uses the Optimization Toolbox function `fminsearch`, which is a direct search method to optimize the response. `Simplex search` is most useful for simple problems and is sometimes faster than `Function minimization` for models that contain discontinuities.

For **Model size**, the two options are `Large scale` and `Medium scale`. By default, **Model size** is set to `Large scale`. The `Large scale` methods will always run faster, if there is some sparsity structure in the Jacobian (or Hessian) that can be taken advantage of. To learn more, see "Large-Scale vs. Medium-Scale Algorithms" in the Optimization Toolbox documentation.

Selecting Optimization Termination Options

Specify termination options in the **Optimization options** area.



Optimization options

Diff max change:	0.1	Maximum fun evals:	400
Diff min change:	1e-008	Maximum iterations:	100
Parameter tolerance:	1e-006	Function tolerance:	1e-006

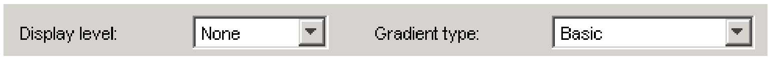
Several options define when the optimization terminates:

- **Diff max change** — The maximum allowable change in variables for finite-difference derivatives. See `fmincon` in the Optimization Toolboxdocumentation for details.
- **Diff min change** — The minimum allowable change in variables for finite-difference derivatives. See `fmincon` in the Optimization Toolbox documentation for details.
- **Parameter tolerance** — Optimization terminates when successive parameter values change by less than this number.
- **Maximum fun evals** — The maximum number of cost function evaluations allowed. The optimization terminates when the number of function evaluations exceeds this value.
- **Maximum iterations** — The maximum number of iterations allowed. The optimization terminates when the number of iterations exceeds this value.
- **Function tolerance** — The optimization terminates when successive function values are less than this value.

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** pane is a group of additional optimization options.



Display level: None
Gradient type: Basic

Additional options for optimization include:

- **Display level** — Specifies the form of the output that appears in the MATLAB command window. The options are `Iteration`, which displays information after each iteration, `None`, which turns off all output, `Notify`, which displays output only if the function does not converge, and `Final`, which only displays the final output. Refer to the Optimization Toolbox documentation for more information on what type of iterative output each algorithm displays.
- **Gradient type** — When using `Gradient Descent` or `Nonlinear least squares` as the **Algorithm**, the gradients are calculated based on finite difference methods. The `Refined` method offers a more robust and less noisy gradient calculation method than `Basic`, although it does take longer to run optimizations using the `Refined` method.

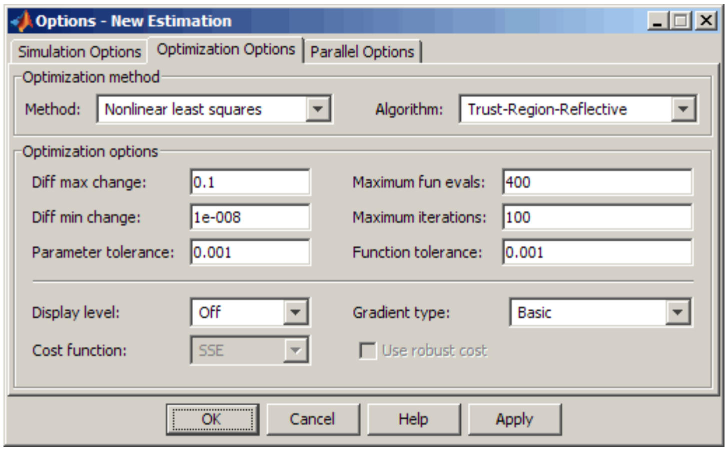
How to Specify Estimation Options in the GUI

You can set several options to tune the results of the estimation. These options include the optimization algorithms and their tolerances.

To set options for estimation:

1. Select the **New Estimation** node in the workspace directory tree.
2. Click the **Estimation** tab.
3. Click **Estimation Options** to open the Options dialog box.

4 Click the **Optimization Options** tab and specify the options.



Method: Nonlinear least squares
Algorithm: Trust-Region-Reflective

Diff max change: 0.1
Diff min change: 1e-008
Parameter tolerance: 0.001
Maximum fun evals: 400
Maximum iterations: 100
Function tolerance: 0.001

Display level: Off
Gradient type: Basic
Cost function: SSE
Use robust cost

Specifying Simulation Options

- "Accessing Simulation Options" on page 2-27
- "Selecting Simulation Time" on page 2-28
- "Selecting Solvers" on page 2-30

Accessing Simulation Options

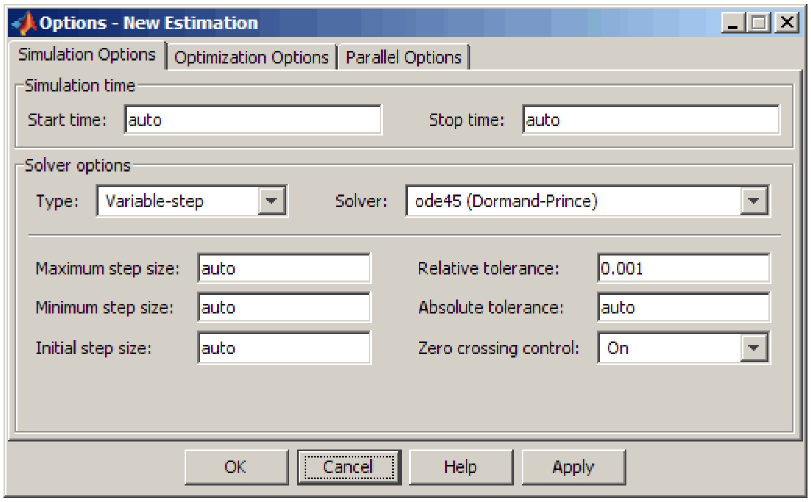
To optimize the response signals of a model, Simulink Design Optimization software runs simulations of the model.

To set options for simulation:

1 Select the **New Estimation** node in the workspace directory tree.

2 Click the **Estimation** tab.

3 Click **Estimation Options** to open the Options dialog box.



Simulation time

Start time: auto
Stop time: auto

Solver options

Type: Variable-step
Solver: ode45 (Dormand-Prince)

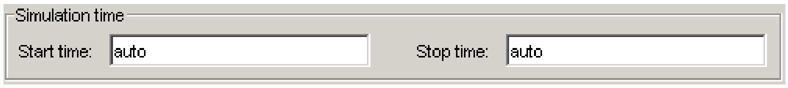
Maximum step size: auto
Relative tolerance: 0.001
Minimum step size: auto
Absolute tolerance: auto
Initial step size: auto
Zero crossing control: On

4 Click the **Simulation Options** tab and specify the options, as described in the following sections:

- "Selecting Simulation Time" on page 2-28
- "Selecting Solvers" on page 2-30

Selecting Simulation Time

You can specify the simulation start and stop times in the **Simulation time** area of the **Simulation Options** tab.



Simulation time

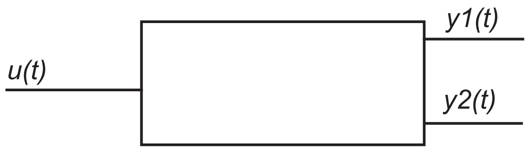
Start time: auto
Stop time: auto

By default, **Start time** and **Stop time** are automatically computed based on the start and stop times specified in the Simulink model.

To set alternative start and stop times for the optimization, enter them under **Simulation time**. This action overwrites the simulation start and stop times specified in the Simulink model.

Simulation Time for Data Sets with Different Time Lengths. Simulink Design Optimization software can simulate models containing empirical data sets of different time lengths. You can use experimental data sets for estimation that contain I/O samples collected at different time points.

The following example shows a single-input, two-output model for which you want to estimate the parameters.



The model uses two output data sets containing transient data samples for parameter estimation:

- Output *y1(t)* at time points $t1 = \left\{t_1^1, t_2^1, \ldots. t_n^1\right\}$.
- Output *y2(t)* at time points $t2 = \left\{t_1^2, t_2^2, \ldots.. t_m^2\right\}$.

The simulation time *t* is computed as:

$$t = t1 \cup t2 = \left\{t_1^1, t_1^2, t_2^1, t_2^2, \ldots.. t_n^1, t_m^2\right\}$$

This new set ranges from *tmin* to *tmax*. The values *tmin* and *tmax* represent the minimum and maximum time points in *t* respectively.

When you run the estimation, the model is simulated over the time range *t*. Simulink extracts the simulated data for each output based on the following criteria:

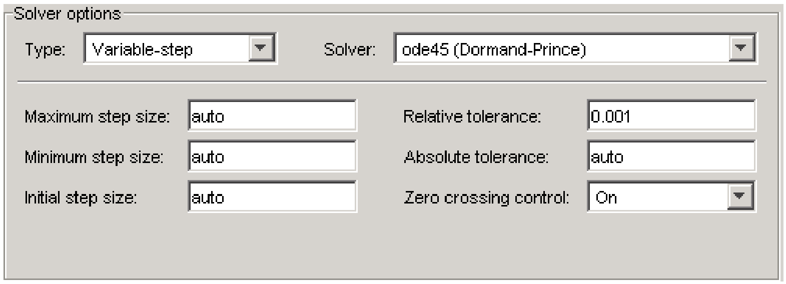
- **Start time** — Typically, the start time in the Simulink model is set to `0`. For a nonzero start time, the simulated data corresponding to time points before t_1^1 for *y1(t)* and t_1^2 for *y2(t)* are discarded.

- **Stop time** — If the stop time $t_{stop} \geq t_{\max}$, the simulated data corresponding to time points in *t1* are extracted for *y1(t)*. Similarly, the simulated data for time points in *t2* are extracted for *y2(t)*.

  If the stop time $t_{stop} < t_{\max}$, the data spanning time points $> t_{stop}$ are discarded for both *y1(t)* and *y2(t)*.

Selecting Solvers

You can specify the solver in the **Solver options** area of the **Simulation Options** tab.



Solver options

Type: Variable-step

Solver: ode45 (Dormand-Prince)

Maximum step size: auto

Minimum step size: auto

Initial step size: auto

Relative tolerance: 0.001

Absolute tolerance: auto

Zero crossing control: On

When running the simulation, the dynamic system is solved using one of several Simulink solvers. You can specify several solver options using the **Solver options** area 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 states are likely to vary rapidly, a variable-step solver is often faster. See the Simulink documentation for information about solvers.

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)`

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 the solver can use during a simulation.
- **Minimum step size** — The smallest step-size the solver can use during a simulation.
- **Initial step size** — The step-size the 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.

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

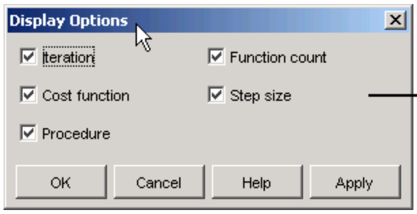
- `Discrete (no continuous states)`
- `ode5 (Dormand-Prince)`
- `ode4 (Runge-Kutta)`
- `ode3 (Bogacki-Shanpine)`
- `ode2 (Heun)`

- `ode1 (Euler)`

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, a value for this option is automatically chosen.

Specifying Display Options

You can specify the display options by clicking **Display Options** in the **Estimation** tab in the Control and Estimation tools Manager. This opens the following dialog box.



By default, all boxes are checked. Uncheck any feature that you don't want to view during the estimation process.

Clearing a check box implies that feature will not appear in the display table as the estimation progresses. To learn more about the display table, see "Displaying Iterative Output" in the Optimization Toolbox documentation.

Estimating and Validating Parameters in the GUI

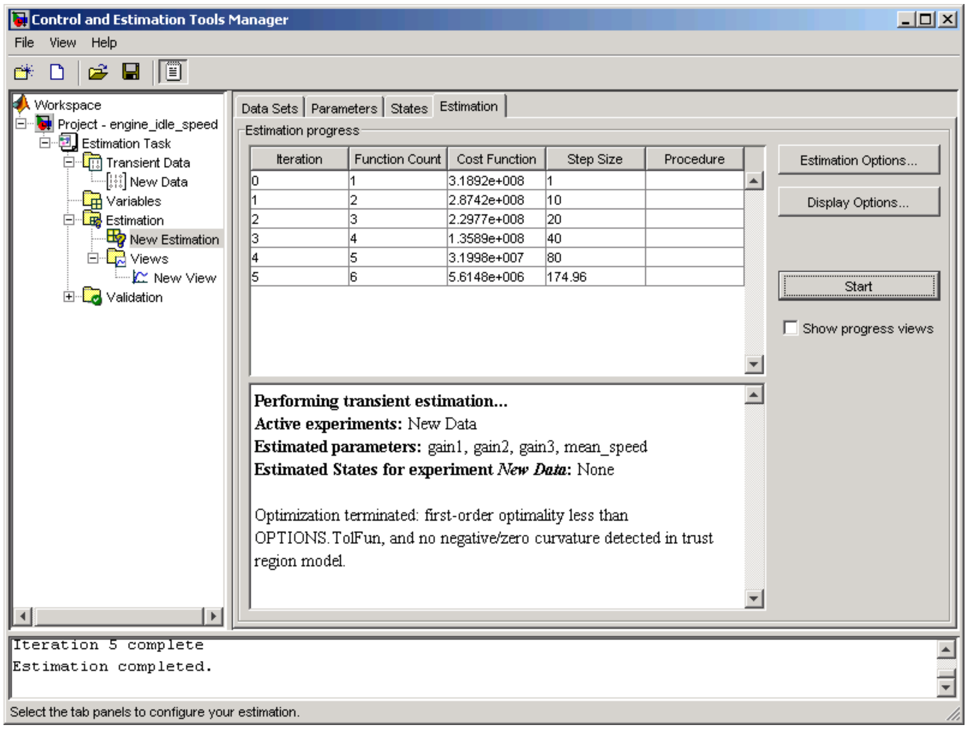
In this section...

Performing Estimation

Before you begin estimating the parameters, you must have configured the estimation data and parameters, and specified estimation and simulation options, as described in "Configuring Parameter Estimation in the GUI" on page 2-3.

To start the estimation, select the **New Estimation** node in the Control and Estimation Tools Manager and select the **Estimation** tab.

Click **Start** to begin the estimation process. At the end of the iterations, the window should resemble the following:



Iteration	Function Count	Cost Function	Step Size	Procedure
0	1	3.1892e+008	1	
1	2	2.8742e+008	10	
2	3	2.2977e+008	20	
3	4	1.3589e+008	40	
4	5	3.1998e+007	80	
5	6	5.6148e+006	174.96	

Performing transient estimation...
Active experiments: New Data
Estimated parameters: gain1, gain2, gain3, mean_speed
Estimated States for experiment *New Data*: None

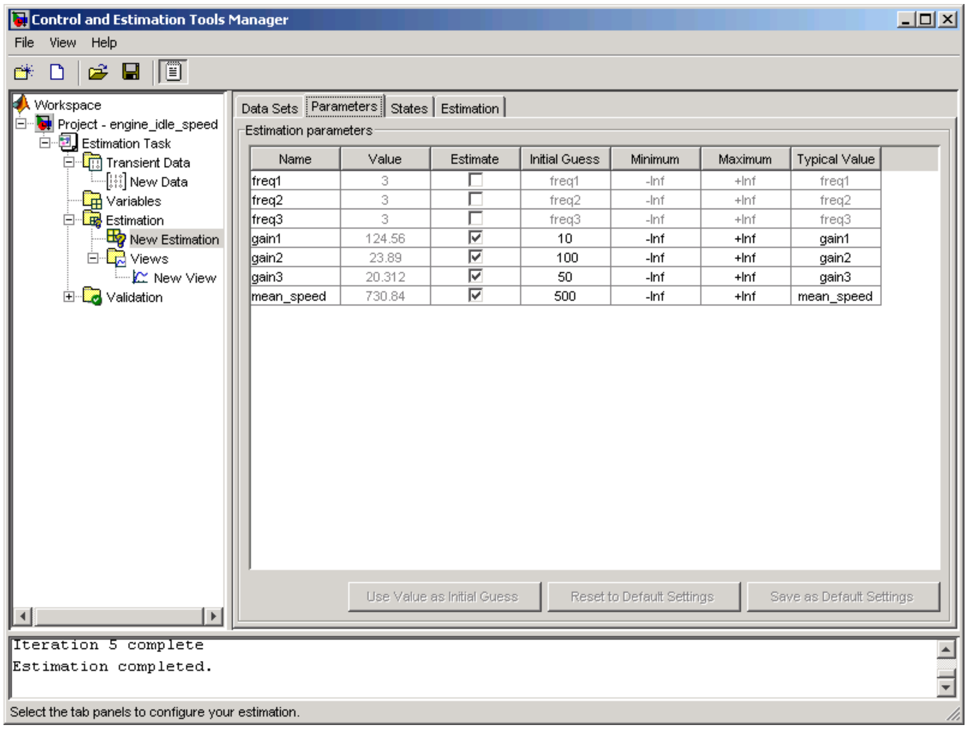
Optimization terminated: first-order optimality less than OPTIONS.TolFun, and no negative/zero curvature detected in trust region model.

```
Iteration 5 complete
Estimation completed.
```

Select the tab panels to configure your estimation.

Usually, a lower cost function value indicates a successful estimation, meaning that the experimental data matches the model simulation with the estimated parameters.

The **Estimation** pane displays each iteration of the optimization algorithm. To see the final values for the parameters, click the **Parameters** tab.



Estimation parameters

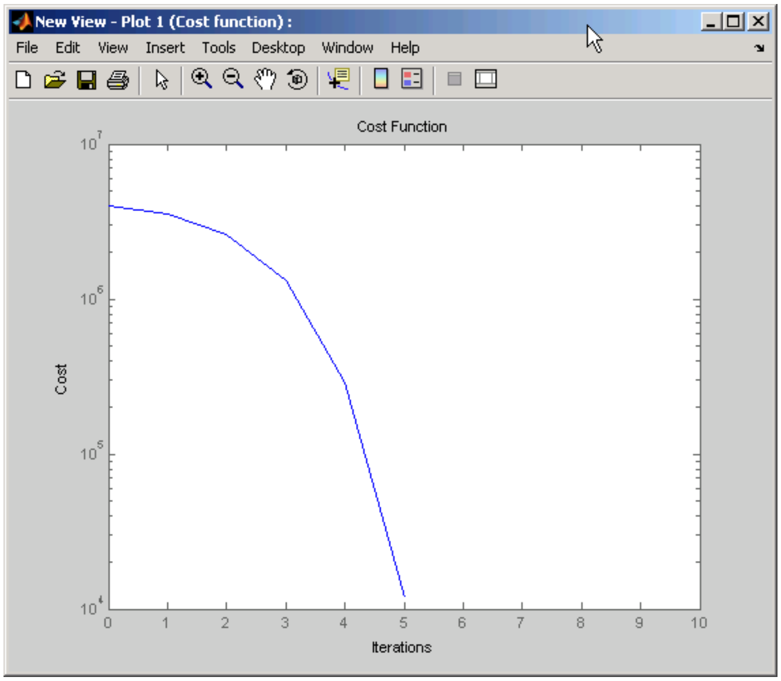
Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
freq1	3	☐	freq1	-Inf	+Inf	freq1
freq2	3	☐	freq2	-Inf	+Inf	freq2
freq3	3	☐	freq3	-Inf	+Inf	freq3
gain1	124.56	☑	10	-Inf	+Inf	gain1
gain2	23.89	☑	100	-Inf	+Inf	gain2
gain3	20.312	☑	50	-Inf	+Inf	gain3
mean_speed	730.84	☑	500	-Inf	+Inf	mean_speed

```
Iteration 5 complete
Estimation completed.
```

Select the tab panels to configure your estimation.

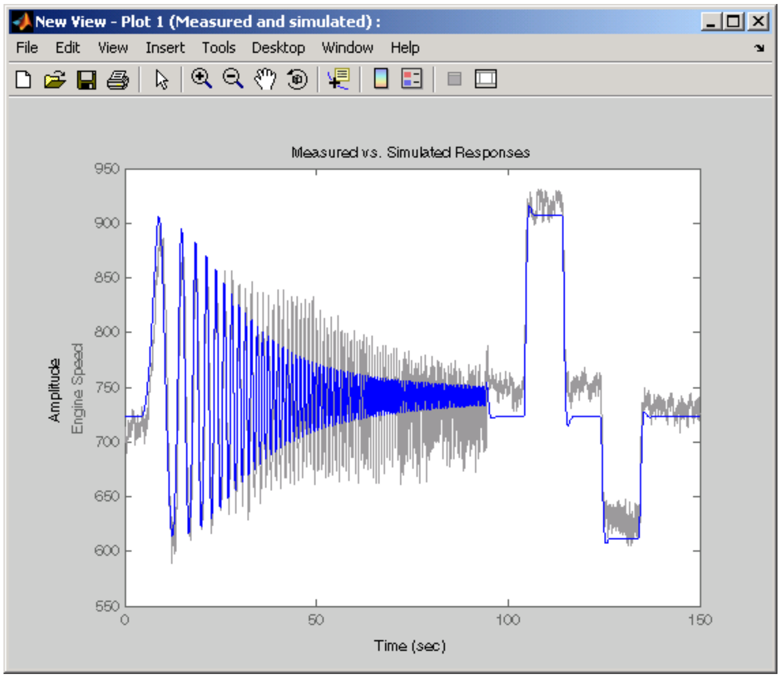
The values of these parameters are also updated in the MATLAB workspace. If you specify the variable name in the **Initial Guess** column, you can restart the estimation from where you left off at the end of a previous estimation.

After the estimation process completes, the cost function minimization plot appears as shown in the following figure.



If the optimization went well, you should see your cost function converge on a minimum value. The lower the cost, the more successful is the estimation.

You can also examine the measured versus simulated data plot to see how closely the simulated data matches the measured estimation data. The next figure shows the measured versus simulated data plot generated by running the estimation of the `engine_idle_speed` model.



Basic Steps for Model Validation

After you complete estimating the parameters, as described in “Performing Estimation” on page 2-33, you must validate the results against another set of data.

These are the basic steps to validate a model using the Control and Estimation Tools Manager:

1 Import the validation data set to the **Transient Data** node.

2 Add a new validation task in the **Validation** node in the workspace directory tree.

3 Configure the validation settings by selecting the plot types and the validation data set from the **Validation Setup** pane.

4 Click **Show Plots** in the **Validation Setup** pane and view the results in the plot window.

5 Compare the validation plots to the corresponding view plots to see if they match.

The basic difference between the validation and views features is that you can run validation after the estimation is complete. All views should be set up before an estimation, and you can watch the views update in real time. Validations can use other validation data sets for comparison with the model response. Also, validations appear after you have completed an estimation and do not update.

You can validate your data by comparing measured vs. simulated data for your estimation data and validation data sets. Also, it is often useful to compare residuals in the same way.

Loading and Importing the Validation Data

To validate the estimated parameters computed in "Performing Estimation" on page 2-33, you must first import the data into the Control and Estimation Tools Manager GUI.

To load the validation data, type

```
load iodataval
```

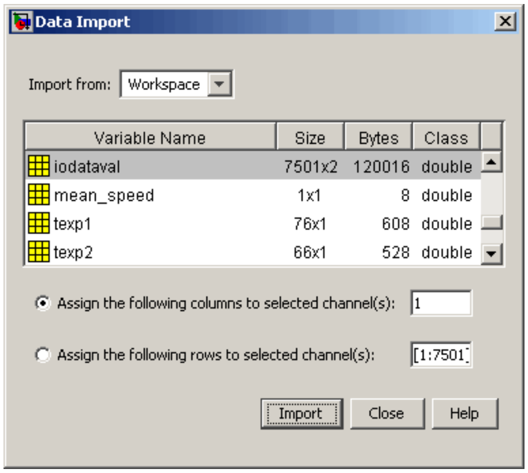
at the MATLAB prompt. This loads the data into the MATLAB workspace. The next step is to import this data into the Control and Estimation Tools Manager. See "Importing Data into the GUI" on page 1-5 for information on importing data, but the quickest way is to follow these steps:

1 Right-click the **Transient Data** node in the workspace directory tree in the Control and Estimation Tools Manager and select **New**.

2 Select `New Data (2)` from the **Transient Data** pane.

3 Right-click the `New Data (2)` node in the workspace directory tree and select **Rename**. Change the name of the data to **Validation Data**.

4 In the **Input Data** pane, select the **Data** cell associated with `Channel - 1` and click **Import**. In the Data Import dialog box, select `iodataval` and assign column 1 to the selected channel by entering `1` in the **Assign columns** field. Click **Import** to import the input data.



Data Import

Import from: Workspace

Variable Name	Size	Bytes	Class
iodataval	7501x2	120016	double
mean_speed	1x1	8	double
texp1	76x1	608	double
texp2	66x1	528	double

[✓] Assign the following columns to selected channel(s): 1

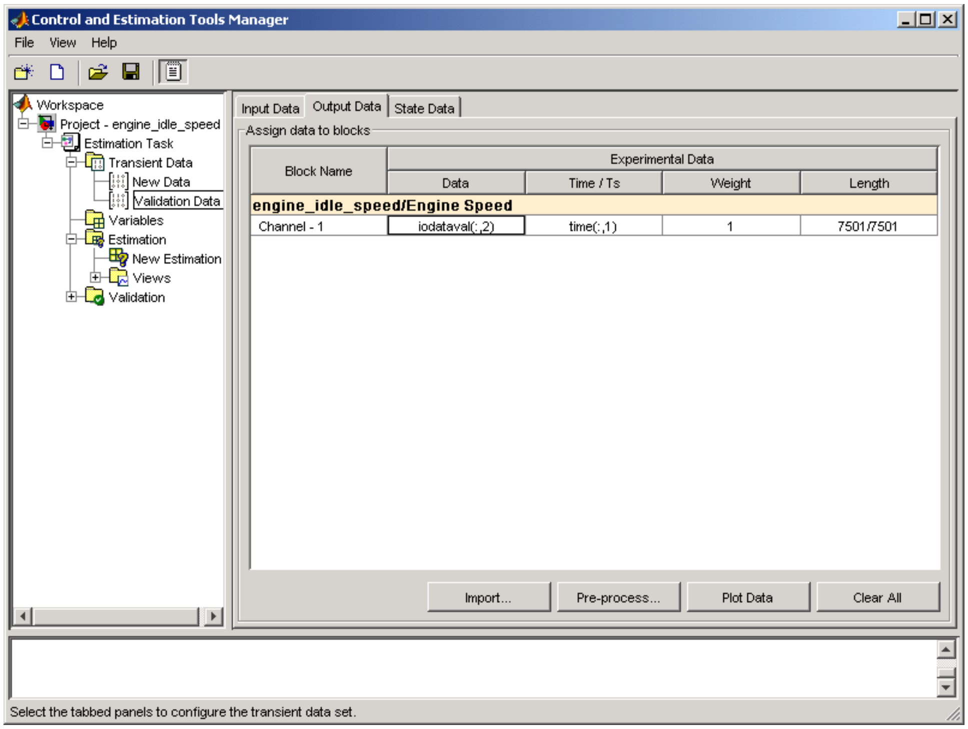
Assign the following rows to selected channel(s): [1:7501]

5 Select the **Time/Ts** cell and import `time` using the Data Import dialog box.

6 Similarly, in the **Output Data** pane, select **Time/Ts** and import `time`.

7 In the **Output Data** pane, select the **Data** cell associated with `Channel - 1` and click **Import**. Import the second column of data in `iodataval` by selecting it from the list in the Import Data dialog box and entering 2 in the **Assign columns** field. Click **Import** to import the output data.

The Control and Estimation Tools Manager should resemble the next figure.

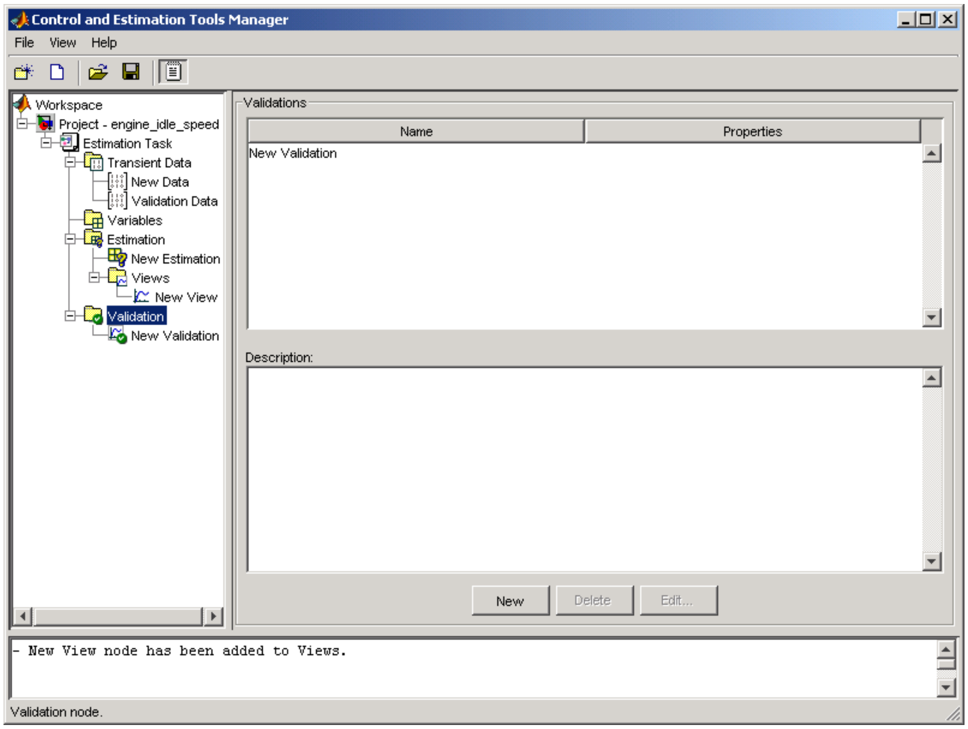


Block Name	Experimental Data			
	Data	Time / Ts	Weight	Length
engine_idle_speed/Engine Speed				
Channel - 1	iodataval(:,2)	time(:,1)	1	7501/7501

Select the tabbed panels to configure the transient data set.

Performing Validation

After you import the validation data, as described in “Loading and Importing the Validation Data” on page 2-38, right-click the **Validation** node and select **New**. This creates a **New Validation** node in the Control and Estimation Tools Manager.

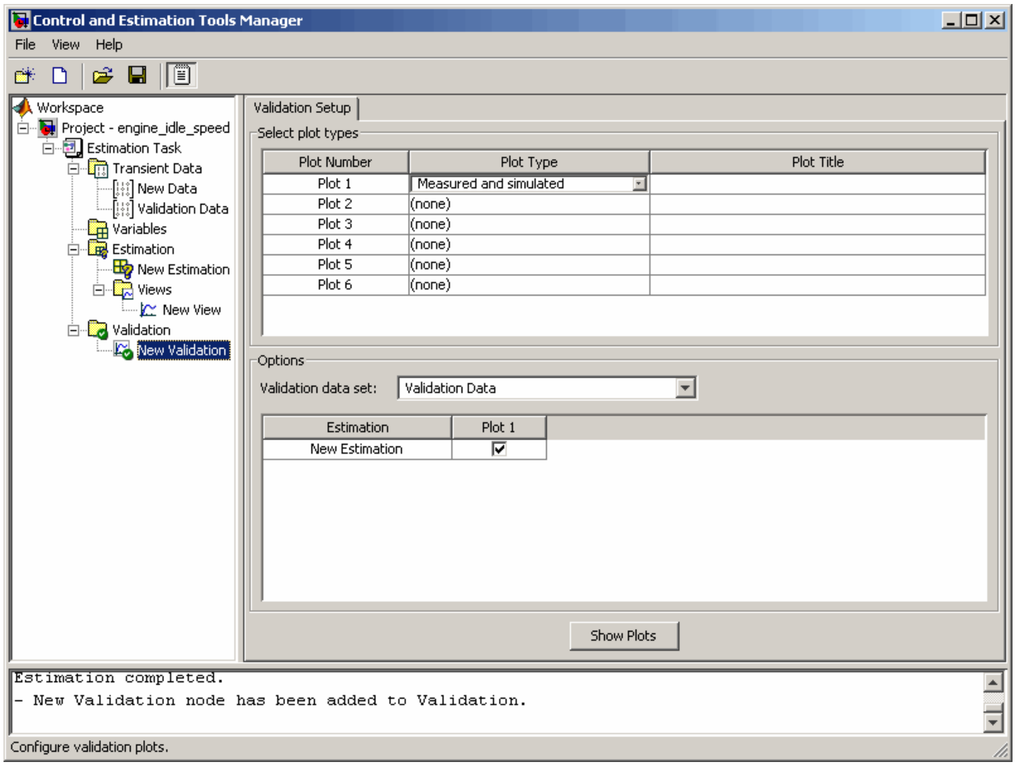


```
- New View node has been added to Views.
```

Validation node.

To perform the validation:

1 Select the **New Validation** node in the workspace directory tree to open the **Validation Setup** pane.



Plot Number	Plot Type	Plot Title
Plot 1	Measured and simulated	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

Validation data set: Validation Data

Estimation	Plot 1
New Estimation	✓

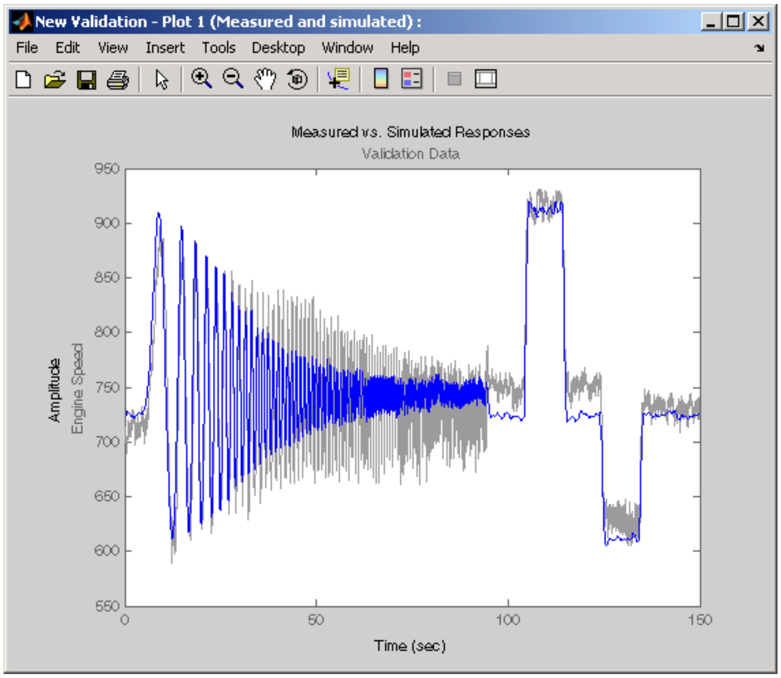
```
Estimation completed.
- New Validation node has been added to Validation.
```

Configure validation plots.

2 Click the **Plot Type** cell for `Plot 1` and select `Measured and simulated` from the drop-down menu.

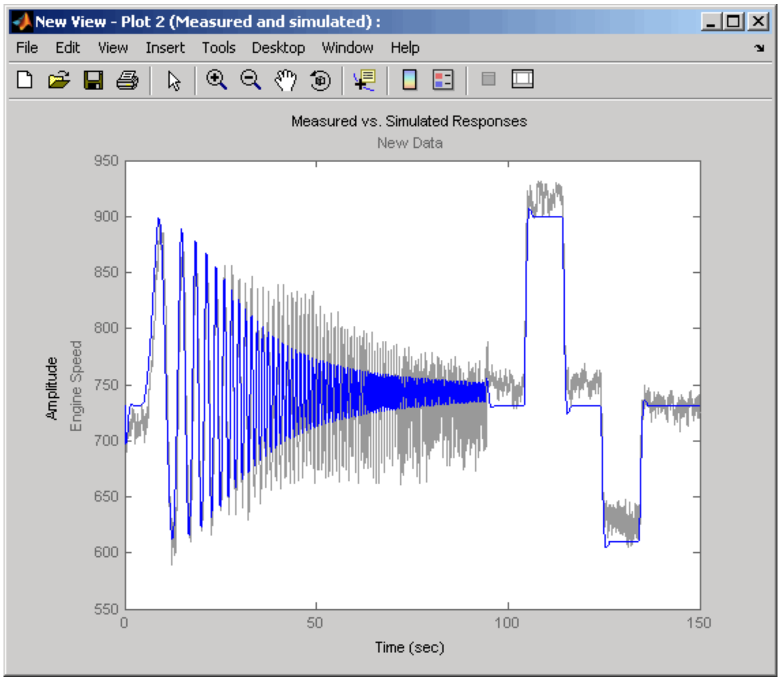
3 In the **Options** area, select `Validation Data` in the **Validation data set** drop-down list.

4 Click **Show Plots** to open a plot figure window as shown next.



Measured Versus Simulated Data Plot for Validation Data

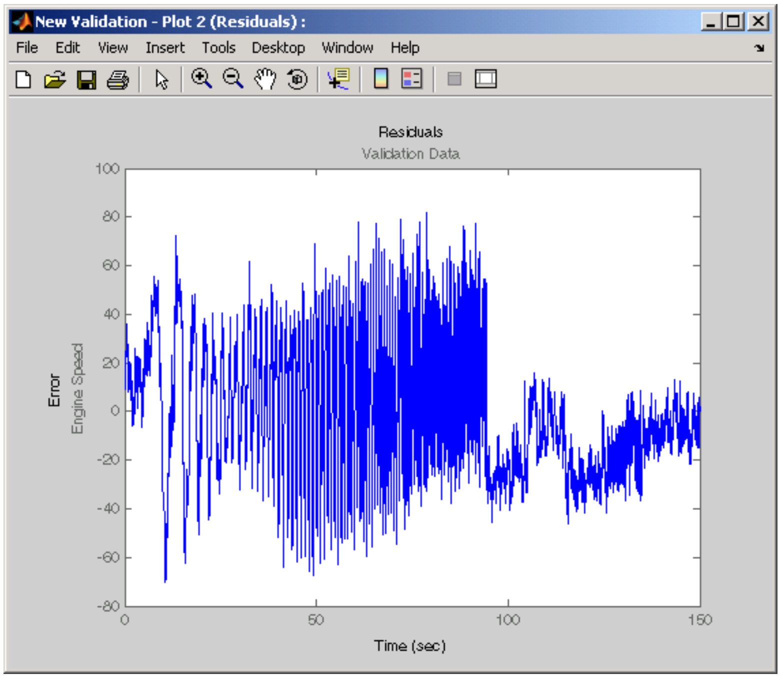
5 Compare this plot with the plot of `Measured and simulated` data for the validation data. For more information on how to create this plot, see "Selecting Views for Plotting" on page 2-18.



Measured and Simulated Data Views Plot

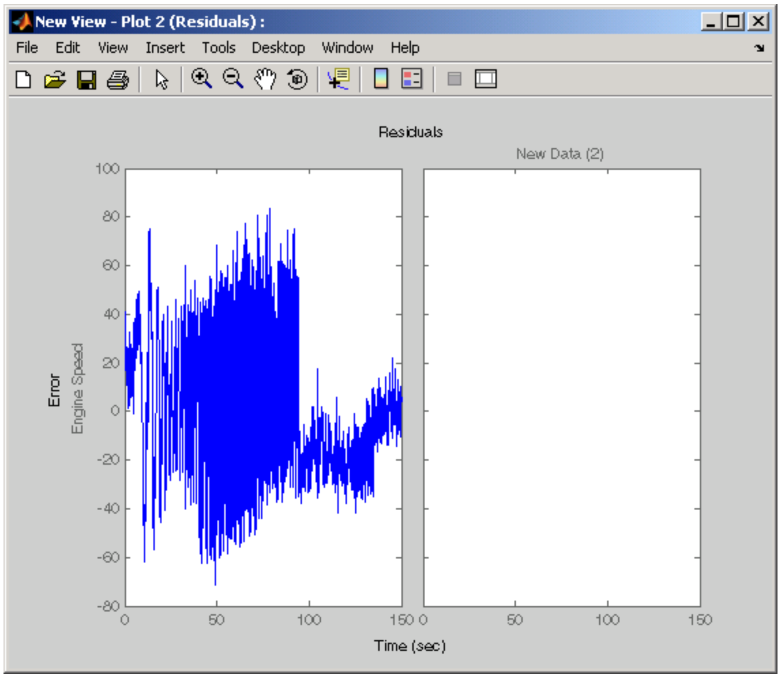
Comparing Residuals

To look at the residuals, select `Residuals` as the **Plot Type** for `Plot 2` in the **New Validation** pane. In the **Options** area, select the **Plot 2** check box and click **Show Plots**. The following figure shows the resulting residuals plot.


New Validation - Plot 2 (Residuals) :
File Edit View Insert Tools Desktop Window Help
Residuals
Validation Data
Error
Engine Speed
Time (sec)
100
80
60
40
20
0
-20
-40
-60
-80
0
50
100
150

Plot of Residuals Using the Validation Data

Compare the validation data residuals with the original data set residuals from the **Views** node in the workspace directory tree. To create the plot of residuals for the original data set, select the **New View** node and choose `Residuals` as the **Plot Type**.



Plot of Residuals Using the Test Data

The plot on the left agrees with the plot of the residuals for the validation data. The right side has no plot because residuals were not calculated for the validation data during the original estimation process.

Accelerating Model Simulations During Estimation

In this section...

About Accelerating Model Simulations During Estimation

You can accelerate the parameter estimation computations by changing the simulation mode of your Simulink model. Simulink Design 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 software uses interpreted code, rather than compiled C code during simulations.

In the `Accelerator` mode, Simulink Design Optimization software runs simulations during estimation with compiled C code. Using compiled C code speeds up the simulations and reduces the time to estimate parameters.

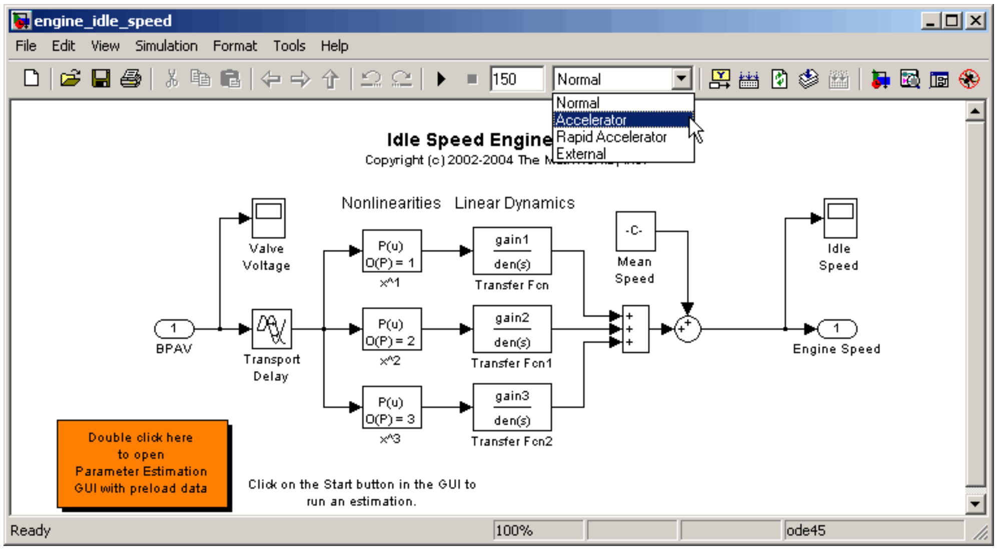
Limitations

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.

Setting the Accelerator Mode for Parameter Estimation

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.

Speeding Up Parameter Estimation Using Parallel Computing

In this section...

When to Use Parallel Computing for Estimating Model Parameters

You can use Simulink Design Optimization software with Parallel Computing Toolbox™ software to speed up parameter estimation of Simulink models. Using parallel computing may reduce the estimation time in the following cases:

- The model contains a large number parameters to estimate, and the `Nonlinear least squares` or `Gradient descent` is selected as the estimation algorithm.
- The `Pattern search` algorithm is selected as the estimation algorithm.
- The model is complex and takes a long time to simulate.

When you use parallel computing, Simulink Design Optimization software distributes independent simulations to run them in parallel on multiple MATLAB sessions, also known as *workers*. The time required to simulate the model dominates the total estimation time. Therefore, distributing the simulations significantly reduces the estimation time. For more information on the expected speedup, see "How Parallel Computing Speeds Up Parameter Estimation" on page 2-50.

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

- "Configuring Your System for Parallel Computing" on page 2-53
- "How to Use Parallel Computing in the GUI" on page 2-55
- "How to Use Parallel Computing at the Command Line" on page 2-110

How Parallel Computing Speeds Up Parameter Estimation

You can enable parallel computing with the `Nonlinear least squares`, `Gradient descent` and `Pattern search` estimation algorithms in the Simulink Design Optimization software. The following sections describe how parallel computing speeds up the estimation:

- "Parallel Computing with the `Nonlinear least squares` and `Gradient descent` Algorithms" on page 2-50
- "Parallel Computing with the `Pattern search` Algorithm" on page 2-51

Parallel Computing with the `Nonlinear least squares` and `Gradient descent` Algorithms

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

- Objective value computation — One simulation per iteration
- 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 parameters to estimate, and N_{ls} is the number of line searches.

When you use parallel computing, Simulink Design Optimization software distributes the simulations required for 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 reduction of the total estimation 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_{1s}=3, the expected reduction of the total estimation time equals $\frac{1 + 2 \times ceil\left(\frac{3}{4}\right) + 3}{1 + (2 \times 3) + 3} = 0.6$.

Parallel Computing with the `Pattern search` Algorithm

The `Pattern search` algorithm uses search and poll sets to create and compute a set of candidate solutions at each estimation 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 parameters to estimate, 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 Design 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 estimation 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 $\frac{ceil(3 \times \frac{15}{4}) + ceil(3 \times \frac{2}{4})}{3 \times (15 + 2)} = 0.27$.

Using the `Pattern search` algorithm with parallel computing may not speed up the estimation time. When you do not use parallel computing, the algorithm stops searching for a candidate solution at each iteration as soon as it finds a solution better than the current solution. When you use parallel computing, the candidate solution search is more comprehensive. Although the number of iterations may be larger, the estimation without using parallel computing may be faster.

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 Design 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 "Scope of 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 estimation using parallel computing:

1. Add the path dependencies, as described "How to Use Parallel Computing in the GUI" on page 2-55 and "How to Use Parallel Computing at the Command Line" on page 2-110.
2. Add the file dependencies, as described in "Configuring Parallel Computing on Multiprocessor Networks" on page 2-54.

Note When you use parallel computing, verify that the remote MATLAB workers can access all the model dependencies. The optimization errors out if all the remote workers cannot access all the model dependencies.

Configuring Your System for Parallel Computing

You can use parallel computing on multi-core processors or multi-processor networks. To configure your system for parallel computing, see the following sections:

- “Configuring Parallel Computing on Multicore Processors” on page 2-54
- “Configuring Parallel Computing on Multiprocessor Networks” on page 2-54

After you configure your system for parallel computing, you can use the GUI or the command-line functions to estimate the model parameters.

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.

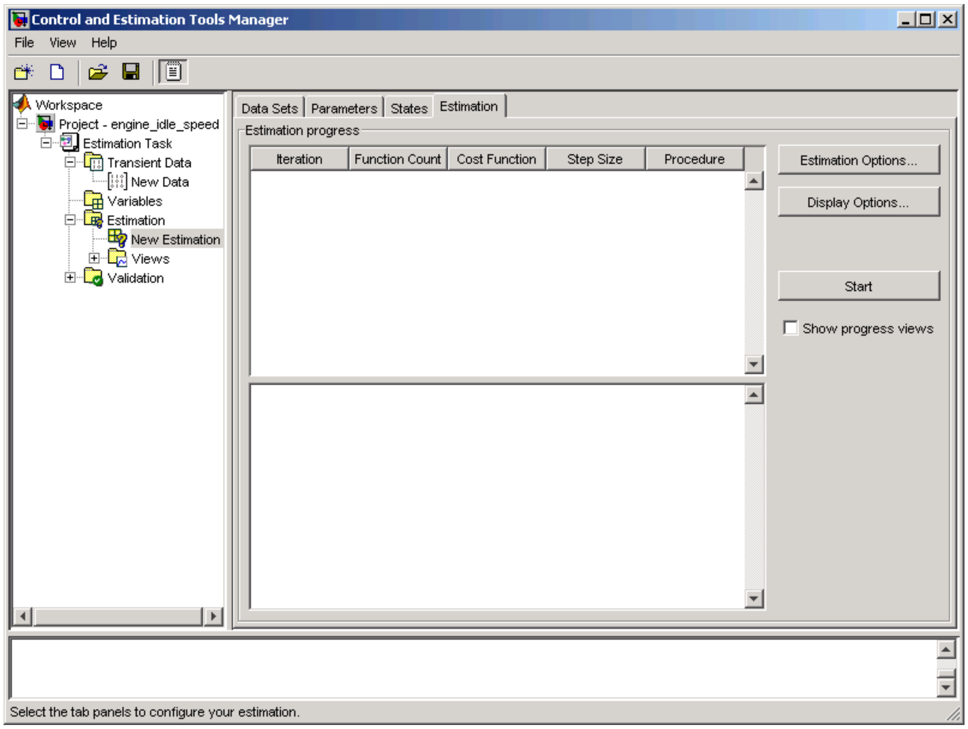
How to Use Parallel Computing in the GUI

After you configure your system for parallel computing, as described in "Configuring Your System for Parallel Computing" on page 2-53, you can use the GUI to estimate the model parameters.

Tip If you want to use functions to estimate parameters using parallel computing, see "How to Use Parallel Computing at the Command Line" on page 2-110.

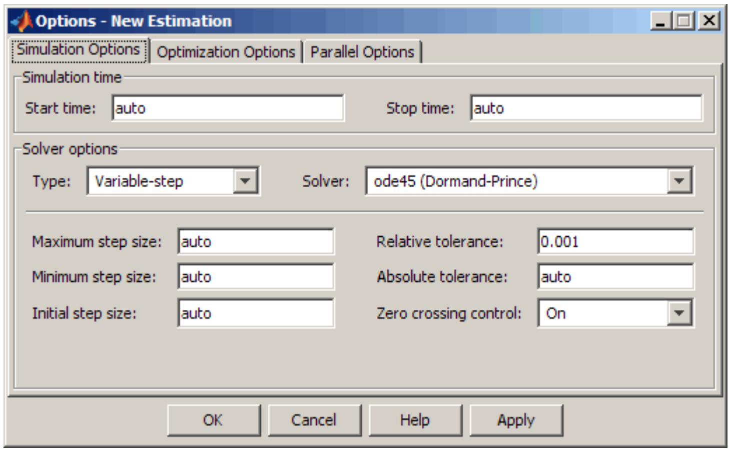
---

1. Open the Simulink model by typing the model name at the MATLAB prompt.
2. Configure the model for parameter estimation, as described in "Configuring Parameter Estimation in the GUI" on page 2-3.
3. In the **Estimation** tab of the **New Estimation** node, click **Estimation Options**.



Select the tab panels to configure your estimation.

This action opens the Options - New Estimation dialog box.

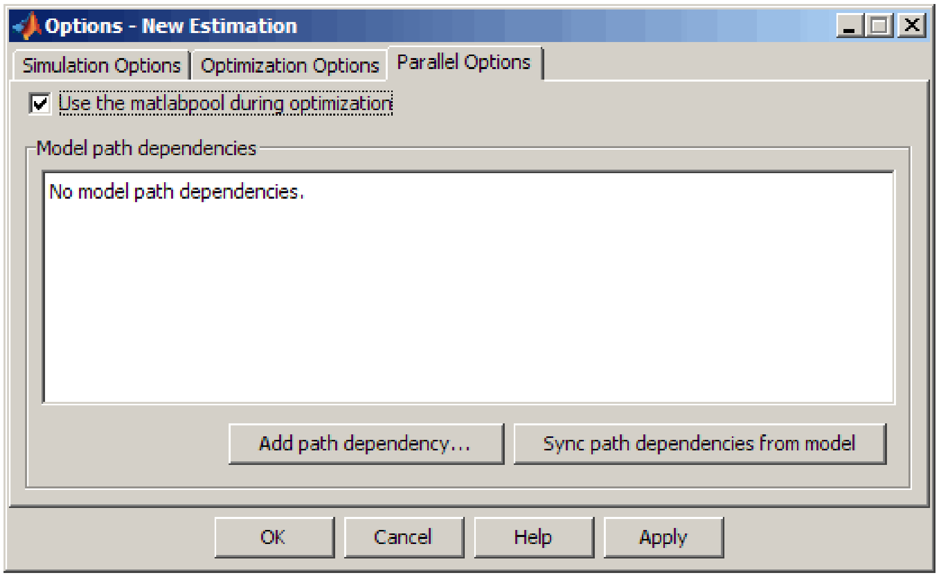


Start time: auto
Stop time: auto
Type: Variable-step
Solver: ode45 (Dormand-Prince)
Maximum step size: auto
Relative tolerance: 0.001
Minimum step size: auto
Absolute tolerance: auto
Initial step size: auto
Zero crossing control: On

4 In the **Parallel Options** tab, 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 "Model Dependencies" on page 2-53, the automatic path dependencies check may not detect all the path dependencies in your model.



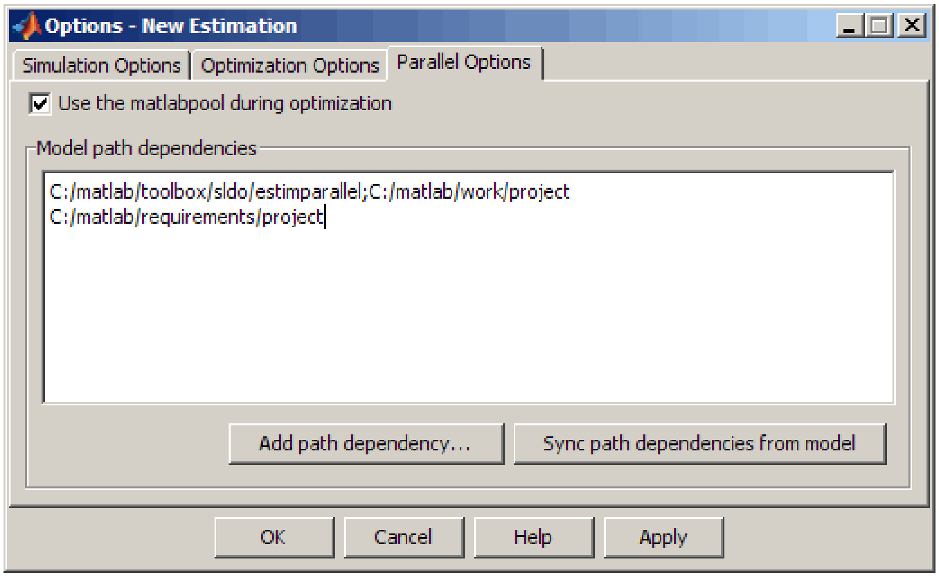
☑ Use the matlabpool during optimization

No model path dependencies.

5 (Optional) Add the path dependencies that the automatic check does not detect.

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.



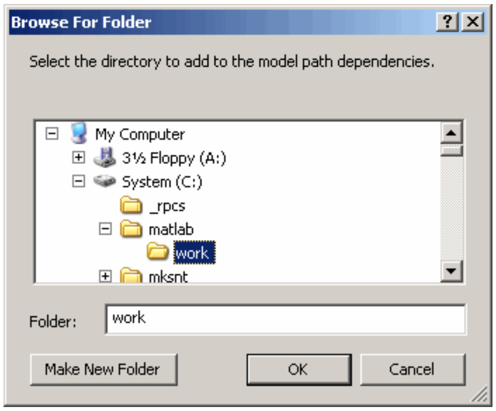
☑ Use the matlabpool during optimization

Model path dependencies

```
C:/matlab/toolbox/sldo/estimparallel;C:/matlab/work/project
C:/matlab/requirements/project
```

b Click **Apply** to include the new paths.

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



Select the directory to add to the model path dependencies.

Folder: work

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 **Estimation** tab, click **Start** to estimate the model parameters using parallel computing.

9. Examine the values of the estimated parameters in the **Value** column of the **Parameters** tab.

   For more information on how to troubleshoot estimation results you obtained using parallel computing, see “Troubleshooting” on page 2-59.

Troubleshooting

- “Why are the estimation results with and without using parallel computing different?” on page 2-59
- “Why do I not see the estimation speedup I expected using parallel computing?” on page 2-60
- “Why does the estimation using parallel computing not make any progress?” on page 2-61
- “Why do I receive an error "Cannot save model tpe5468c55_910c_4275_94ef_305e2eeeeef4"?” on page 2-61
- “Why does the estimation using parallel computing not stop when I click **Stop**?” on page 2-61

Why are the estimation results with and without using parallel computing different?

The values of the estimated parameters obtained using parallel computing may differ from the values obtained without using parallel computing. The results can be different under the following conditions:

- Different numerical precision on the client and worker machines can produce marginally different simulation results. Thus, the estimation algorithm may take a completely different solution path and produce a different result.

Note Numerical precision can differ because of different operating systems or hardware on the client and worker machines.

- The state of the model on the client and the worker machines can differ, and thus lead to a different result. For example, the state can become different if you change a parameter value initialized by a callback function on the client machine after the workers have loaded the model. The model parameter values on the workers and the client are now out of sync, which can lead to a different result.

  After you change the model parameter values initialized by a callback function, verify that the parameters exist in the model workspace or update the callback function so that the remote workers have access to the changed parameter values.
- When you use parallel computing with the `Pattern search` algorithm, the algorithm searches for a candidate solution more comprehensively than when you do not use parallel computing. This more comprehensive search can result in a different solution. To learn more, see "Parallel Computing with the `Pattern search` Algorithm" on page 2-51.

Why do I not see the estimation speedup I expected using parallel computing?

- The resulting estimation time may not be faster when you estimate a small number of model parameters or when the model does not take long to simulate. In such cases, the overheads associated with creating and distributing the parallel tasks outweighs the benefits of running the simulations during estimation in parallel.
- Using `Pattern search` algorithm with parallel computing may not speed up the estimation time. When you do not use parallel computing, the algorithm stops searching for a candidate solution at each iteration as soon as it finds a solution better than the current solution. The candidate solution search is more comprehensive when you use parallel computing.

Although the number of iterations may be larger, the optimization without using parallel computing is faster.

To learn more about the expected speedup, see "Parallel Computing with the `Pattern search` Algorithm" on page 2-51.

Why does the estimation using parallel computing not make any progress?

In some cases, the gradient computations on the remote worker machines may silently error out when you use parallel computing. In such cases, the **Estimation progress** table shows that the f(x) values do not change, and the optimization terminates after two iterations.

To troubleshoot the problem:

1. Run the optimization for a few iterations without parallel computing to see if the optimization progresses.

2. Check if the remote workers have access to all model dependencies. To learn more, see "Model Dependencies" on page 2-53.

Why do I receive an error "Cannot save model tpe5468c55_910c_4275_94ef_305e2eeeeef4"?

When you select `Refined` as the **Gradient type**, the software may error out when it saves a temporary model to a nonwriteable directory, and then displays this error message. Change the **Gradient type** to `Basic` to clear this error. To learn more, see "Selecting Additional Optimization Options" on page 2-26.

Why does the estimation using parallel computing not stop when I click Stop?

When you use parallel computing, the software has to wait till the current iteration completes before it notifies the workers to stop the estimation. The estimation does not terminate immediately when you click **Stop**, and appears to continue to run.

Estimating Initial States

In this section...

How to Estimate Initial States in the GUI

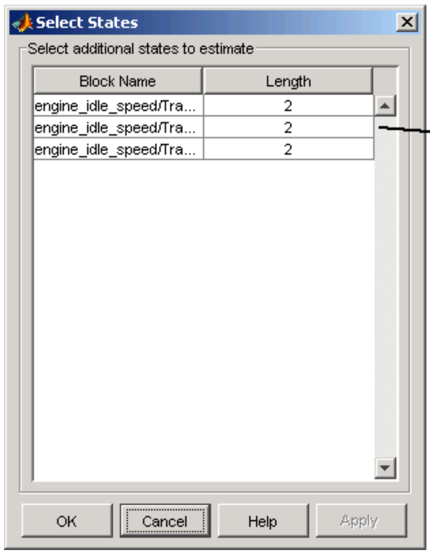
In general, you choose to estimate only those states that are not already in the model. Before you estimate initial conditions, you must have already imported the estimation data and specified the parameters to estimate, as described in "Specifying Data for Parameter Estimation" on page 2-4 and "Specifying Parameters to Estimate" on page 2-6, respectively.

To estimate initial conditions (or initial states) if they are not known:

1 In the Control and Estimation Tools Manager, select the **Variables** node in the workspace directory tree.

2 Click the **Estimated States** tab.

3 Click **Add** to open the Select States dialog box.



Block Name	Length
engine_idle_speed/Tra...	2
engine_idle_speed/Tra...	2
engine_idle_speed/Tra...	2

By default, the Select States dialog box looks at all states of all blocks in the model.

List of states

Use your mouse to select data. To select adjacent states, hold down the **Shift** key while clicking the first and last state in the selection. To select nonadjacent states, hold down the **Ctrl** key while clicking each state.

4 Select the states to estimate and click **OK**.

The states selected for estimation are added to the **Estimated States** tab. For an example of estimating initial states, see "Example — Estimating Initial States of a Mass-Spring-Damper System" on page 2-64.

Estimating Initial Conditions for Blocks with External Initial Conditions

When an integrator block uses an initial-condition port, which you specify by an IC block feeding into the integrator block, you cannot estimate the initial conditions (ICs) of the integrator using Simulink Design Optimization software. This is because external ICs have priority over the ICs of a specific block to maintain the integrity of the model.

To tune the ICs of an integrator block with external ICs, you must modify the model to make the external signal into a tunable parameter. For example, you can set the IC block that feeds into the integrator to be a tunable variable and estimate it.

Example — Estimating Initial States of a Mass-Spring-Damper System

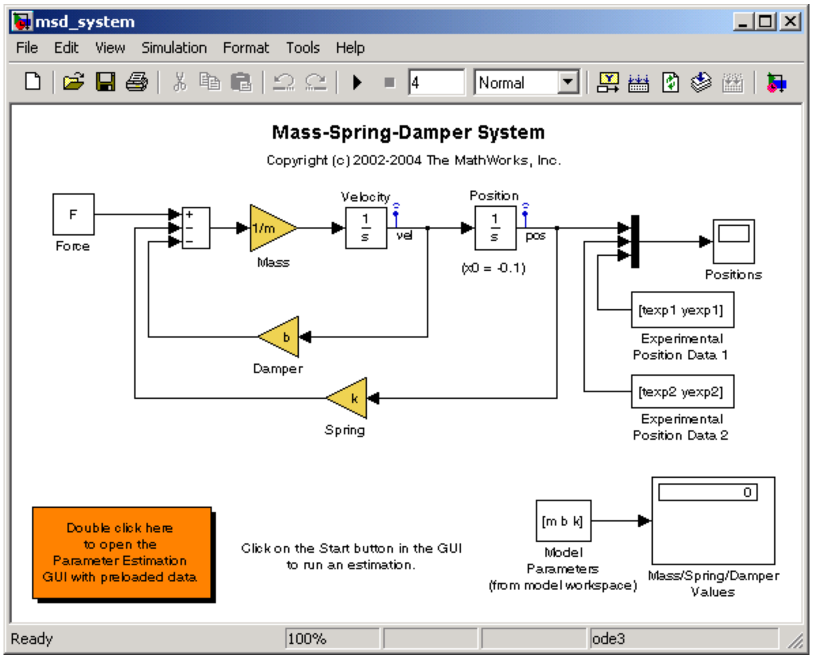
Loading the Example

To open the Simulink model of a mass-spring-damper system and two sets of model data with differing initial conditions, type:

```
msd_system
```

at the MATLAB prompt.

The figure shown next is a model of a mass-spring-damper system.

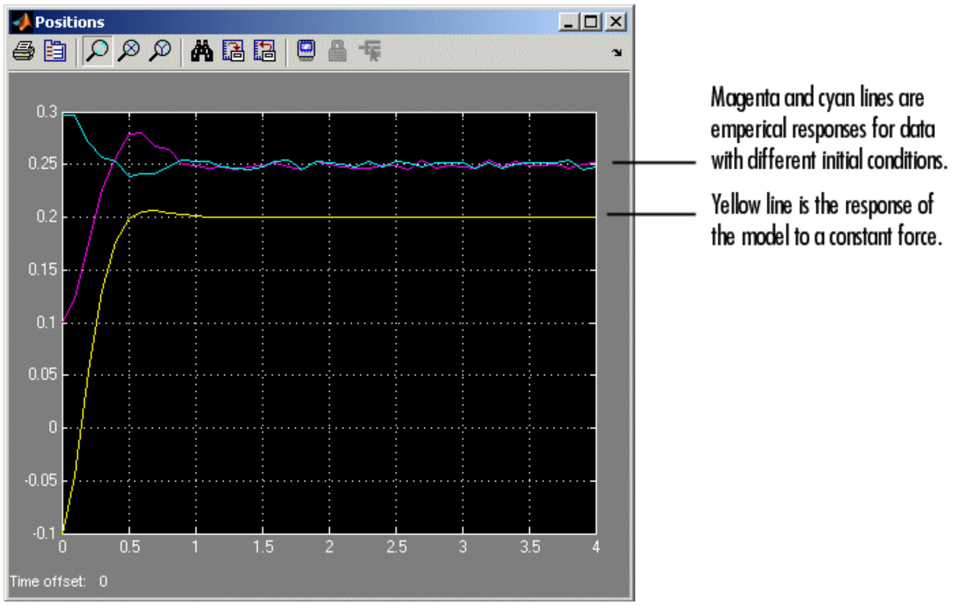

msd_system
File Edit View Simulation Format Tools Help
4
Normal
Mass-Spring-Damper System
Copyright (c) 2002-2004 The MathWorks, Inc.
F
Force
1/m
Mass
Velocity
1/s
vel
Position
1/s
pos
(x0 = -0.1)
Positions
[texp1 yexp1]
Experimental Position Data 1
[texp2 yexp2]
Experimental Position Data 2
b
Damper
k
Spring
Double click here to open the Parameter Estimation GUI with preloaded data
Click on the Start button in the GUI to run an estimation.
[m b k]
Model Parameters (from model workspace)
0
Mass/Spring/Damper Values
Ready
100%
ode3

You can run the demo from **Simulink > Simulink Design Optimization** on the **Demo** pane of the Help browser.

This example goes beyond what is included in the Simulink Design Optimization demo that uses this model by providing in-depth discussion of each task.

Model Parameters

The Simulink `msd_system` model's output is the displacement (or position) of the mass in a mass-spring-damper system, subject to a constant force `F`, and an initial condition, `x0`, for the mass displacement. `x0` is indicated by the initial condition of the Position integrator block. Click the **Start Simulation** button ▶ to run the simulation once and observe the response of the model to two sets of parameter values.

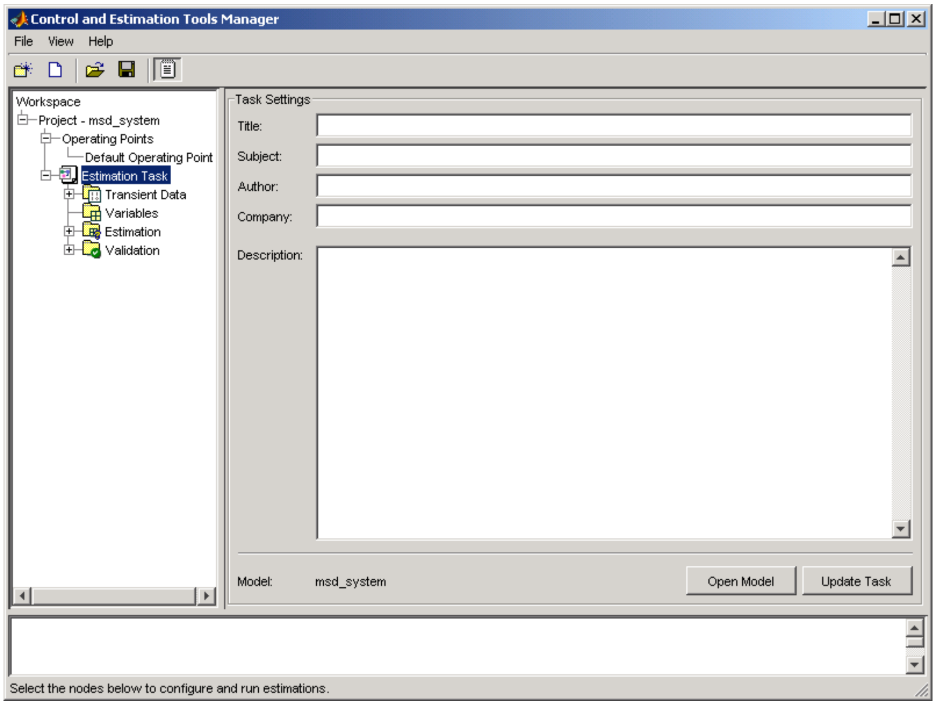


The model parameters of interest are the mass, `m`, the viscous damping, `b`, and the spring constant, `k`. For more information about physical modeling of mass-spring-damper systems, see any elementary book on mathematical modeling or on automatic control systems.

For the estimation of the model parameters `m`, `b`, and `k`, this model uses two sets of experimental data. These data sets were obtained using two different initial positions, `x0=0.1` and `x0=0.3`, and also contain additive noise. A plot of these data sets is shown in the figure above (top curves), along with the simulated response (bottom curve) of the Simulink model `msd_system` for `x0=-0.1` and a nominal set of parameter values, `m=8`, `k=500`, and `b=100`.

Setting Up the Estimation Project

To set up the estimation of initial conditions and then transient state space data, select **Tools > Parameter Estimation** in the `msd_system` model window.



Model: msd_system

Select the nodes below to configure and run estimations.

Importing Transient Data and Selecting Parameters for Estimation

The process for importing transient data and selecting parameters for estimation is discussed in "Importing Data into the GUI" on page 1-5, and "Specifying Parameters to Estimate" on page 2-6.

1 In the Control and Estimation Tools Manager, select **Estimation Task > Transient Data** in the workspace directory tree.

2 Right-click **Transient Data** and select **New** to add a new data set.

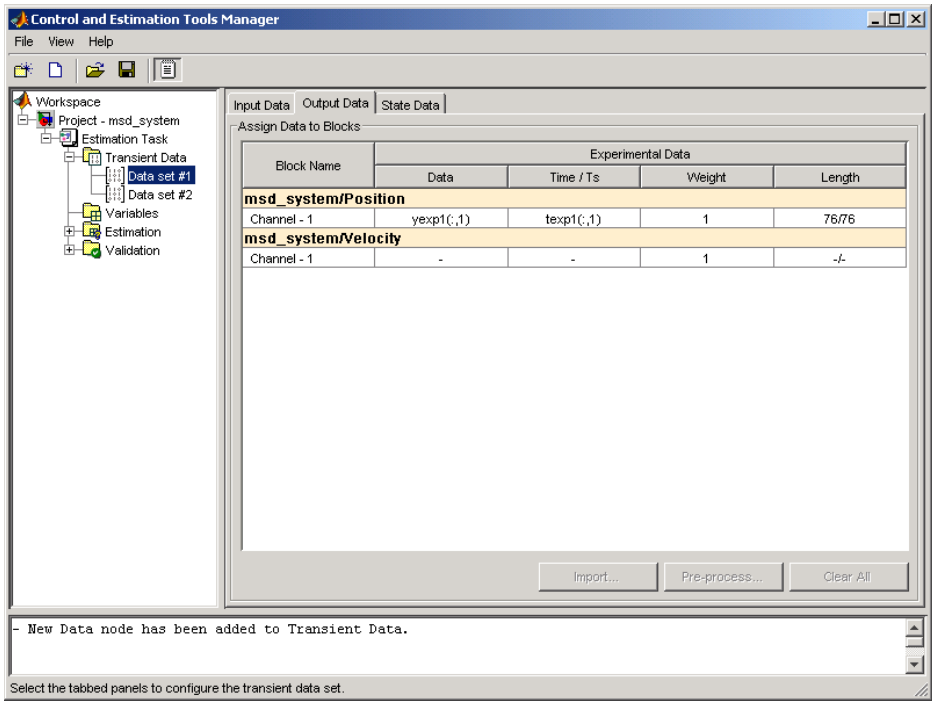
3 Right-click the **New Data** node in the workspace directory tree and select **Edit** to open the **Input Data**, **Output Data**, and **State Data** panes.

4 In the **Output Data** pane, click **Import** and add `yexp1` to the **Data** column and `texp1` to the **Time/Ts** column of the `msd_system/Position` state.

5 If you like, right-click **New Data** in the workspace directory tree and rename it to `Data set #1`.

6 Repeat steps 1 to 5 to add a second data set, `yexp2` and `texp2`, and rename it to `Data set #2`.

The Control and Estimation Tools Manager should resemble the next figure:



Block Name	Experimental Data			
	Data	Time / Ts	Weight	Length
msd_system/Position				
Channel - 1	yexp1(:,1)	texp1(:,1)	1	76/76
msd_system/Velocity				
Channel - 1	-	-	1	-/-

```
- New Data node has been added to Transient Data.
```

Select the tabbed panels to configure the transient data set.

Selecting Parameters and Initial Conditions for Estimation

First, select the parameters you want to estimate for the Simulink `msd_system` model. In this case, select `b`, `k`, and `m`. To do this:

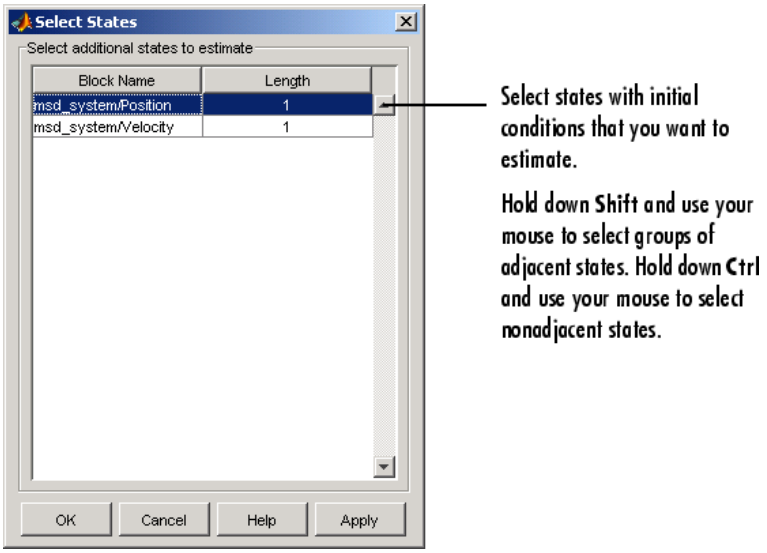
1 Select the **Variables** node in the workspace directory tree of the Control and Estimation Tools Manager.

2 Click the **Estimation Parameters** tab.

3 Click **Add** to open the Select Parameters dialog box.

4 Select the parameters `b`, `k`, and `m`, and then click **OK**.

5 Do the same with the **Estimation States** pane, and select `msd_system/Position` from the Select States dialog box.



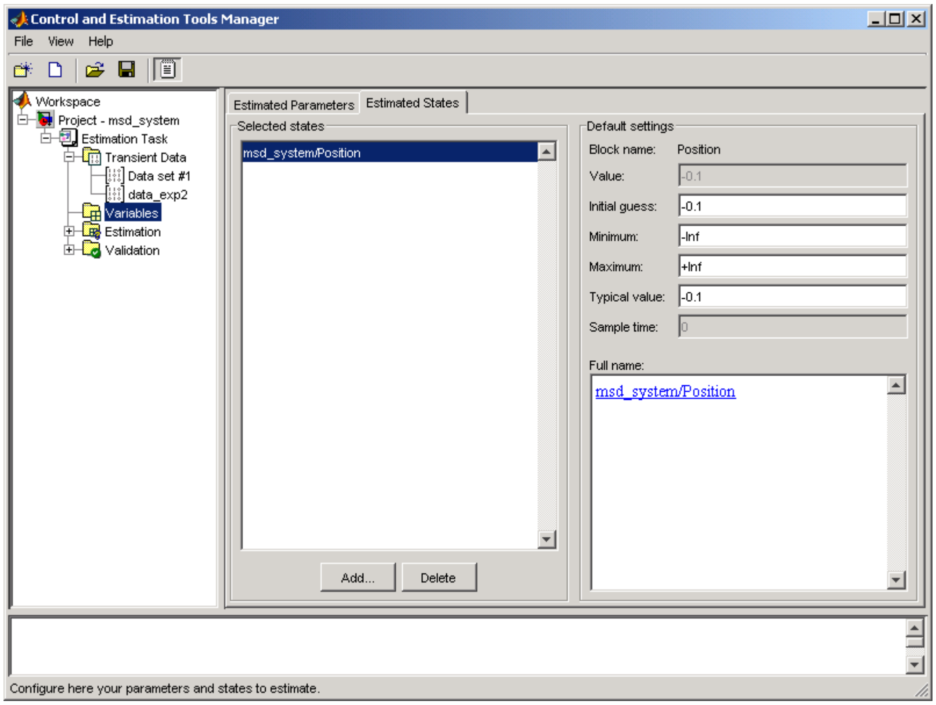
Select States

Select additional states to estimate

Block Name	Length
msd_system/Position	1
msd_system/Velocity	1

OK
Cancel
Help
Apply
Select states with initial conditions that you want to estimate.
Hold down Shift and use your mouse to select groups of adjacent states. Hold down Ctrl and use your mouse to select nonadjacent states.

Your Control and Estimation Tools Manager should look like this.



Selected states

msd_system/Position

Default settings

Block name: Position
Value: -0.1
Initial guess: -0.1
Minimum: -Inf
Maximum: +Inf
Typical value: -0.1
Sample time: 0

Full name:

msd_system/Position

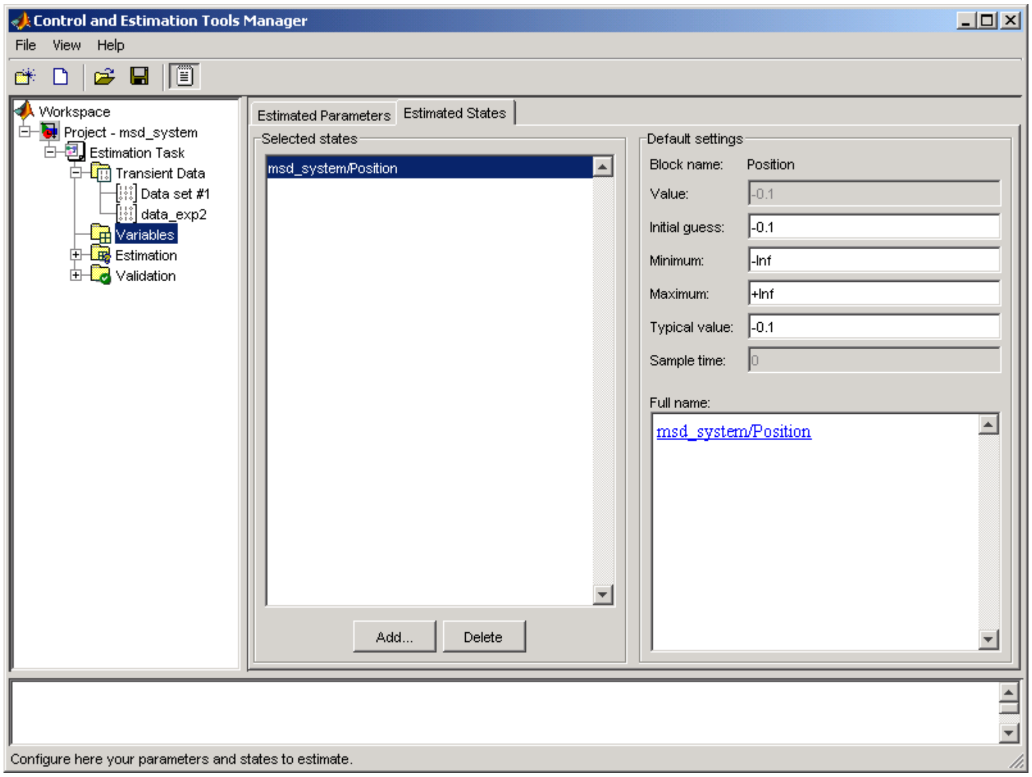
Configure here your parameters and states to estimate.

Creating the Estimation Task

To create the **New Estimation** task in the Control and Estimation Tools Manager, right-click the **Estimation** node in the workspace directory tree and select **Add**. While the initial velocity is also a state of the model, assume (for simplicity) that it is known to be 0. The estimation task for this case is `Estim (with IC)`.

In the **Data Sets**, **Parameters**, and **States** panes for the **New Estimation** task, select all the check boxes in each table. Be sure to select **Position** for both data sets in the **States** pane to estimate the initial condition for the spring's position.

The initial position estimates for the two data sets are known to differ, but set the initial state guesses for both data sets to `-0.1`.



Selected states

msd_system/Position

Default settings

Block name:	Position
Value:	-0.1
Initial guess:	-0.1
Minimum:	-Inf
Maximum:	+Inf
Typical value:	-0.1
Sample time:	0

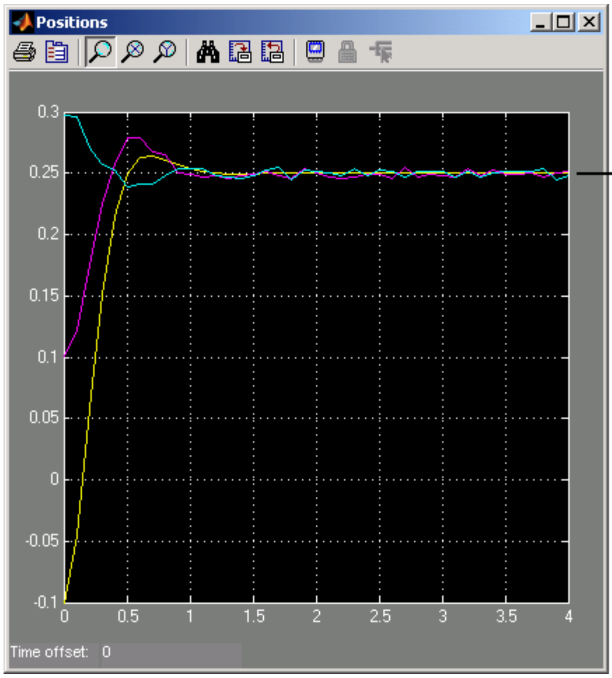
Full name:

msd_system/Position

Configure here your parameters and states to estimate.

Running the Estimation and Viewing Results

Click **Start** in the **Estimation** pane to run the estimation. As the estimation proceeds, the most current estimation of position response (yellow curve) updates itself in the Scope. The curve appears to toggle between the two experimental data sets, since the estimator uses the two sets successively to update the estimates of the parameter values. The estimator converges to the correct parameter values, within the scope of experimental noise and optimization options settings, as indicated by the closeness of the estimated response (yellow) to the experimental data (magenta). Good state estimates for the initial position are also obtained, as can be observed from the **States** tab of `Estim(with IC)` estimation task.



The estimation of initial states is important for obtaining the correct estimates of the model parameters. Why not set the initial states (x0 in this case) as parameters as well? The reason is that the initial states are not fixed physical properties of the system. For different experimental data or operating conditions, these states need not be unique. In this example, two data sets, with distinct initial positions, were used together for a single estimation of model parameters. While the estimates of the model parameters are unique, the initial state (position) is different, and is estimated individually for each data set.

Working with Estimation Projects

In this section...

Structure of an Estimation Project

The Control and Estimation Tools Manager, which is a graphical user interface (GUI) for performing parameter estimation, stores and organizes all data from a given Simulink model inside a *project*. To open the Control and Estimation Tools Manager GUI, select **Tools > Parameter Estimation** in the Simulink model window.

When using the Control and Estimation Tools Manager for parameter estimation, you can

- Manage estimation projects.
- Select parameters and initial conditions to configure the estimation.
- Specify cost functions.
- Import experimental data (to be matched by the output of your Simulink model).
- Specify the initial conditions of your model.

Each estimation task can include

- One or more data sets
- Parameter information
- One or more sets of estimation settings, or configurations

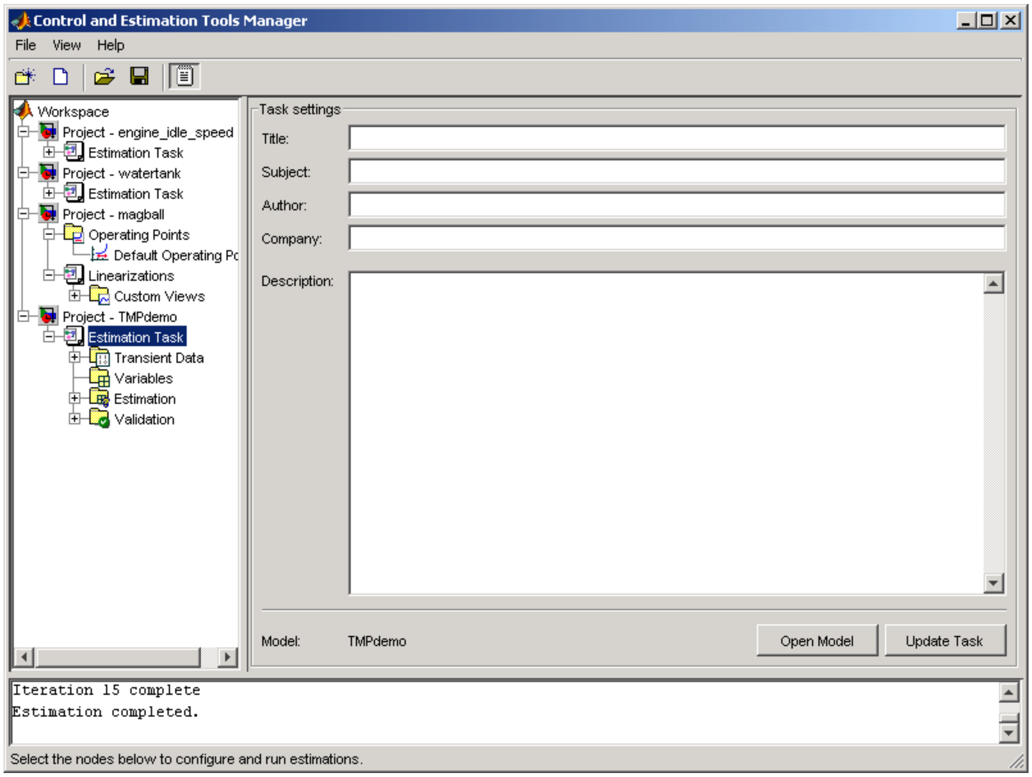
The default project name is the same as the Simulink model name. The project name is shown in the *workspace directory tree* of the Control and Estimation Tools Manager.

You can also add tasks from Simulink® Control Design™ and Model Predictive Control Toolbox™ software to the current project, if these products are installed on your system.

Managing Multiple Projects and Tasks

The Control and Estimation Tools Manager works seamlessly with products in the Controls and Estimation family. In particular, if you have licenses for Simulink Control Design or Model Predictive Control Toolbox software, you can use these products to perform tasks on projects that you have created in Simulink Design Optimization software, and vice versa.

This figure shows a tools manager with multiple projects and multiple tasks.



```
Iteration 15 complete
Estimation completed.
```

You can save projects individually, or group multiple projects together in one saved file, as described in:

- "Saving Control and Estimation Tools Manager Projects" on page 2-76
- "Loading Control and Estimation Tools Manager Projects" on page 2-77

Adding, Deleting and Renaming an Estimation Project

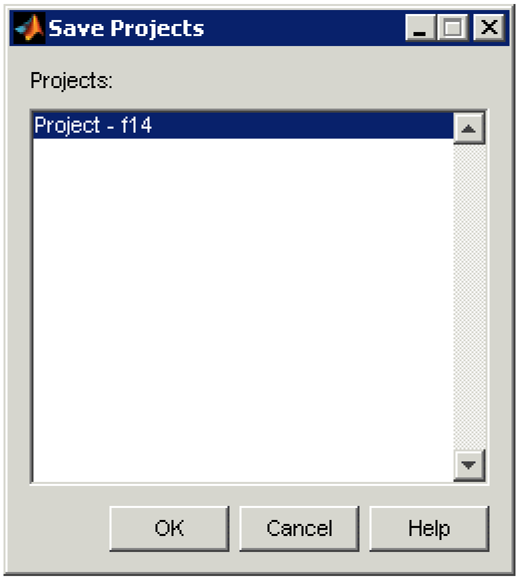
To add, delete, or rename the project or task:

1. Right-click the project or task node in the workspace directory tree.
2. Select the appropriate command from the shortcut menu.

Saving Control and Estimation Tools Manager Projects

A Control and Estimation Tools Manager project can consist of tasks from products such as Simulink Control Design, Simulink Design Optimization, and Model Predictive Control Toolbox software. Each task contains data, objects, and results for the analysis of a particular model.

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



To save multiple projects within one file:

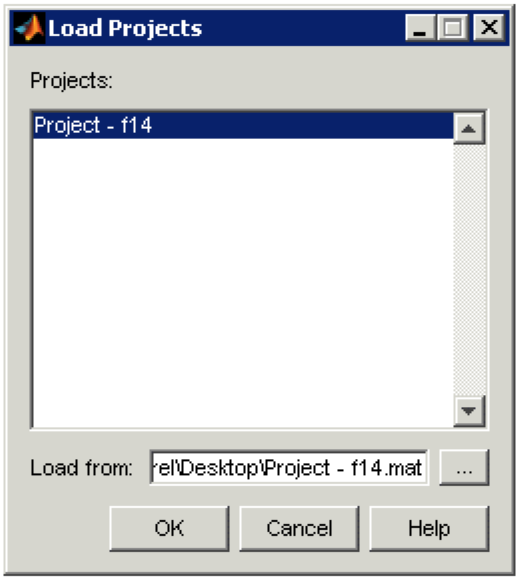
1 In the Save Projects dialog box, select the projects that you want to save.

2 Click **OK**.

3 Choose a directory and name for your project file by either browsing for a file or typing the full path and filename in the **Save as** field. Click **Save**.

Loading Control and Estimation Tools Manager Projects

To open previously saved projects, select **File > Load** in the Control and Estimation Tools Manager window.



In the Load Projects dialog box, choose a project file by either browsing for the directory and file, or by typing the full path and filename in the **Load from** field. Project files are always MAT-files. The projects within this file appear in the **Projects** list.

Select the projects that you want to load, then click **OK**. When a file contains multiple projects, you can choose to load them all or just a few.

Estimating Parameters at the Command Line

In this section...

Workflow for Estimating Parameters at the Command Line

In addition to the Control and Estimation Tools Manager GUI, you can also use Simulink Design Optimization functions to perform parameter and state estimation. These functions perform the same tasks as the tools manager, but have the advantages of command-line execution. When you perform a state or parameter estimation using the GUI, you create MATLAB objects for all the states and parameters of your model. If you have a large number of states or parameters, this can use up large amounts of memory and cause computational delays. With the command-line approach, only those states and parameters that you select are assigned MATLAB objects, which is more efficient.

In addition, the command-line approach is useful for batch jobs where you can estimate parameters for a large numbers of models.

Simulink Design Optimization software uses MATLAB objects to perform estimation tasks. To learn more about object-oriented programming, see the *Object-Oriented Programming* documentation for a description of object-oriented programming in MATLAB.

Simulink Design Optimization command-line interface requires a Simulink model as a starting point for analysis and estimation.

Note The Simulink model must contain an Inport or Outport block or logged signals to enable assigning data to the signals. For more information on how to configure a Simulink model for parameter estimation, see "Configuring a Model for Importing Data " on page 1-2.

After you configure your model for parameter estimation, as described in "Configuring a Model for Importing Data " on page 1-2, the estimation process at the command line consists of the following steps:

1. Defining experiments consisting of empirical data sets, and the operating conditions and/or initial conditions of your model.
2. Selecting the variables and states to be estimated.
3. Performing the estimation.
4. Reviewing the results and iterating as necessary.
5. Validating estimation results.

The following sections discuss these topics:

- "Example — Estimating Parameters and Initial States at the Command Line" on page 2-100 — How to perform the estimation using command-line functions
- "Objects for Parameter Estimation" on page 2-79 — How to use methods and properties to perform parameter estimation

Objects for Parameter Estimation

The following sections describe in more detail how to create and modify transient data and estimation objects:

- "Creating Transient Data Objects" on page 2-80
- "Creating State Data Objects" on page 2-84
- "Creating Transient Experiment Objects" on page 2-87
- "Creating Parameter Objects" on page 2-90

- “Creating State Objects” on page 2-93
- “Creating Estimation Objects” on page 2-97

First, a quick look at terminology:

- *Objects* are instantiations of *classes*.
- *Classes* contain, or rather, define, *properties* and *methods*.
- You use a *constructor* to create an instance of an object, and use the `set` method or dot notation to modify the properties of your objects.

Creating Transient Data Objects

- “What is a Transient Data Object” on page 2-80
- “Constructor” on page 2-80
- “Properties of Transient Data Objects” on page 2-81
- “Modifying Properties of Transient Data Objects” on page 2-83
- “Using Class Methods” on page 2-84

What is a Transient Data Object. The `@TransientData` object encapsulates the data measured at a single input or output of a physical system during an experiment. Transient data objects are associated with three types of Simulink blocks:

- Inport blocks
- Outport blocks
- Internal blocks used in conjunction with signal logging.

Each `@TransientData` object describes the time history of a signal at a Simulink port. A data set is identified by the `Block` property of this object corresponding to a block name in the Simulink model. A `PortNumber` value is also necessary for internal blocks to uniquely identify signals within the block diagram.

Constructor. Estimating parameters requires a transient data object, which you create using a constructor. The syntax to create a transient data object is

```
% I/O port block
h = ParameterEstimator.TransientData('block');
% Internal block
h = ParameterEstimator.TransientData('block',portnumber);

h = ParameterEstimator.TransientData('block',data,time);
h = ParameterEstimator.TransientData('block',data,Ts);
h = ParameterEstimator.TransientData('block',portnumber,data,time);
h = ParameterEstimator.TransientData('block',portnumber,data,Ts);
```

Properties of Transient Data Objects. Descriptions of properties of the transient data object and the associated input parameters are as follows.

Transient Data Object Properties

Property	Description
`Block`	Name of the Simulink block with which the data is associated. Must be a string.
`PortType`	The type of signal that this object represents is determined in the constructor from the `Block` property, which may be `Inport`, `Outport`, or `Signal`.
`PortNumber`	For data associated with the outputs of regular blocks or subsystems, this property specifies the output port number of interest. The default value is `1`.
`Dimensions`	Dimensions of the data required for this data set. It is computed from the `CompiledPortDimensions` property of the appropriate port of the block, and it defines the size of other properties. Currently, Simulink supports scalar, vector, or matrix signals, so `Dimensions` is either a scalar or a 1-by-2 array.

Transient Data Object Properties (Continued)

Property	Description
`Data`	Actual experimental data. Its size must be consistent with the `Dimensions` property. To conform with Simulink conventions, the data is stored in the following formats: • Scalar or vector-valued data. The data is of the form *Ns* *m*, where *Ns* is the number of data samples, and *m* is the number of channels in the signal. • Multidimensional data (matrix and higher dimensions). The data is of the form *m1 . . . mn Ns*, where *Ns* is the number of data samples, and *mi* is the number of channels in the *i*th dimension of the signal. • For missing or unspecified data, `NaN`s are used.
`Ts`, `Tstart`, `Tstop`	For uniformly sampled data, `Ts` is the sample time and `Tstart` is the start time of the signal. The stop time `Tstop` and the time vector `Time` are given by *Tstop* = *Tstart* + *Ts* * (*Ns* -1) *Time* = *Tstart* : *Ts* : *Tstop* For nonuniform time data, `Ts` is set to NaN, and the start and stop times are calculated from the time vector.
`Time`	The time data in column vector format. The length of `Time` must be consistent with the number of samples in `Data`. For a nonuniformly spaced `Time` vector, its length should match the length of `Data`. Otherwise, `Time` is automatically adjusted based on the length of `Data`. Modifying `Ts` resets `Time` internally. In this case, `Time` is a virtual property whose value is computed from `Ts` and `Tstart` when you request it. The rules for setting time related properties are • Modifying `Time` sets

Transient Data Object Properties (Continued)

Property	Description

	`Ts = NaN`
	`Tstart = Time(1)`
	
	• If the time vector is uniformly spaced, a sample time `Ts` is calculated.
	• Modifying `Tstart` translates time forward or backward.
	• Modifying `Ts` sets `Time = []` internally and generates it when required by the simulation.

`Weight`	The weight associated with each channel of this data set. It is used to specify the relative importance of signals. The default value is `1`.
`InterSample`	Interpolation method between samples can be zero-order hold (`zoh`) or first-order hold (`foh`). This property is used for data preprocessing.

Modifying Properties of Transient Data Objects. After a transient data object is created, you can modify its properties using this syntax:

```
in1.Data = rand(2,1,10); % 10 data values each of size [2 1]
in1.Time = 1:10; % Automatically converted to column vector
```

Some properties (e.g., `Weight`) support scalar expansion with respect to the value of the `Dimensions` property.

Example: Assigning Input Port Data

To assign data to an input port with 2-by-3 port dimensions, use

```
in1 = ParameterEstimator.TransientData(gcb, rand(2,3,100), 0.05)
```

This command returns the following result:

```
(1) Transient data for Inport block 'portdata_test_noSim/By//Pass
Air Valve Voltage':
Sampling interval: 0.05 sec.
Data set has 100 samples and 6 channels.
```

Using Class Methods. Descriptions of two important methods are given next:

- `select` — Extracts a portion of data. The result is returned in a new transient data object.

  ```
  in2 = select(in1, 'Sample', 10:100); % 91 samples
  in3 = select(in1, 'Range', [1 4]); % Samples for 1<t<4
  % ... or an alternative
  in3 = select(in1, 'Sample', find(in1.Time > 1 & in1.Time < 4));
  ```

  To extract data from a subset of available channels, use

  ```
  in4 = select(in1, 'Channel', [1 3 2]);
  % channels 1,3,and 2 in this order
  ```

- `hiliteBlock` — Highlights the block associated with this object in the Simulink diagram.

Creating State Data Objects

What is a State Data Object. The `ParameterEstimator.StateData` object defines the known states of a dynamic Simulink block. It is used in a transient estimation context to define known initial conditions of a block diagram model, and in a steady-state estimation context to define the known states of the model.

For example, the Simulink model of a simple mass-spring-damper system has two integrator blocks to generate velocity and position signals from acceleration and velocity values, respectively, during simulation. If the corresponding physical system is known to be at rest at the beginning of an experiment, the initial states (velocity and position) of these integrators are zero. So, two `@StateData` objects can be created to describe these known initial conditions.

Constructor. The syntax for creating this object is

```
h = ParameterEstimator.StateData('block');
h = ParameterEstimator.StateData('block', data);
```

In the first constructor, the state vector is initialized from the model containing the block.

Properties of the State Data Object. Descriptions of some important properties are given in the following table .

State Data Object Properties

Property	Description
`Block`	Name of the Simulink block whose states are defined by this object.
`Dimensions`	Scalar value to store the number of states of the relevant block.

State Data Object Properties (Continued)

Property	Description
`Data`	Column vector to store the initial value of the state for the block specified by this object. The length of this vector should be consistent with the `Dimensions` property. Since the underlying Simulink model also stores an initial state vector for all dynamic blocks, the following conventions are used to resolve the initial state values during estimations: • If `Data` is not empty, use it when forming the state vector. • If `Data` is empty, get the state vector for this block from the model. This behavior is useful when using helper methods to create an experiment object that instantiates empty state data objects for all dynamic blocks in the Simulink model. • If there is no state data object for a dynamic block in the model, get the state vector of that block from the model. This behavior is useful for command-line users, when there are too many states in the model and only a few of them have to be set to different initial values.
`Ts`	Sampling time of discrete blocks. Set to `0` for continuous blocks. This property is read only and is currently used for information only.
`Domain`	String to hold the physical domain of the block. Used for `SimMechanics` or `SimPowerSystems` blocks with states.

Example: Initial Condition Data. To create an empty initial condition object for the `engine_idle_speed/ TransferFcn2`, use

```
st1 = ParameterEstimator.StateData ...
('engine_idle_speed/Transfer Fcn2', [1 2])

(1) State data for 'f14/Dryden Wind Gust Models/W-gust model'
block:
The block has 2 continuous state(s).
State value : [1;2]
```

Modifying Properties. After a state data object is created, you can modify its properties using this syntax:

```
st1.Data = [2 3]; % State vector of size 2
```

Some properties (e.g., `Data`) support scalar expansion with respect to the value of the `Dimensions` property.

Using Class Methods. Description of two important methods are given next:

- `hiliteBlock` — Highlights the block associated with this object in the Simulink diagram.
- `update` — Updates the object after the Simulink model has been modified. If the `Dimensions` property value changes, the other properties are reset to their default values.

Creating Transient Experiment Objects

- “Using Class Methods” on page 2-90

What is a Transient Experiment Object. The `@TransientExperiment` object encapsulates the data measured at the input and output ports of a system during a single experiment, as well as the system’s known initial states.

Constructor. The syntax to create a transient experiment object is

```
h = ParameterEstimator.TransientExperiment('model');
```

where `model` specifies the name of the Simulink model.

Properties of Transient Experiment Objects. Descriptions of some important properties are given in the following table.

Transient Experiment Object Properties

Property	Description
`Model`	Simulink model with which this experiment is associated.
`InputData`, `OutputData`	Transient data objects associated with appropriate I/O blocks in the model. Blocks with unassigned objects or objects with no data are not used in estimations, meaning: • For input ports, assign zeros to these ports/channels during simulation. • For output ports, don’t use these ports/channels in the cost function.
`InitialStates`	State data objects associated with appropriate dynamic blocks in the model.
`InitFcn`	Function to be executed to configure the model for this particular experiment.

Example: Creating an F14 Experiment. To create an empty transient experiment for the `f14` model, use

```
exp1 = ParameterEstimator.TransientExperiment('f14')
Experimental (Transient) data set for the model 'f14':
Outputs
(1) f14/alpha (rad)
(2) f14/Nz Pilot (g)
Inputs
(1) f14/u
Initial States
(1) f14/Actuator Model
(2) f14/Aircraft Dynamics Model/Transfer Fcn.1
(3) f14/Aircraft Dynamics Model/Transfer Fcn.2
(4) f14/Controller/Alpha-sensor Low-pass Filter
(5) f14/Controller/Pitch Rate Lead Filter
(6) f14/Controller/Proportional plus integral compensator
(7) f14/Controller/Stick Prefilter
(8) f14/Dryden Wind Gust Models/Q-gust model
(9) f14/Dryden Wind Gust Models/W-gust model
```

Example: Creating a Van der Pol Experiment from User Objects. To create a transient experiment from user objects for I/Os and states, use

```
out1 = ParameterEstimator.TransientData('vdp/Out1');
ic1 = ParameterEstimator.StateData('vdp/x1');
exp1 = ParameterEstimator.TransientExperiment...
(gcs, [], out1, ic1);
Experimental (Transient) data set for the model 'vdp':
Outputs
(1) vdp/Out1
Inputs
(none)
Initial States
(1) vdp/x1
```

Modifying Properties. The objects referred in `InputData`, `OutputData`, and `InitialStates` properties can be modified or removed as necessary.

Using Class Methods. The description of one important method is given next:

`update` — Updates the object after the Simulink model has been modified. The object listed in the `InputData`, `OutputData`, and `InitialStates` properties are updated in turn.

Creating Parameter Objects

What is a Parameter Object. The `@Parameter` object refers to the parameters of the Simulink model marked for estimation. Some of the Simulink model parameters are to be estimated and storage is required for the initial values, current values, ranges, etc. One `@Parameter` object corresponds to each parameter in the Simulink model to be potentially estimated. These objects represent estimation parameters of any type such as scalars, vectors, and multidimensional arrays.

Constructor. The syntax to create a parameter object is

```
h = ParameterEstimator.Parameter('Name');
h = ParameterEstimator.Parameter('Name', Value);
h = ParameterEstimator.Parameter('Name', Value, Minimum,
    Maximum);
```

In the first case, `Name` is a workspace variable. In the other cases, `Name` does not need to exist in the workspace at the time of object creation. However, it is required at estimation time.

Properties of Parameter Objects. Descriptions of some important properties are given in the following table.

Parameter Object Properties

Property	Description
`Name`	Parameter name. The parameter can be a multidimensional array of any size.
`Dimensions`	Dimensions of the value of the parameter. This is the defining property for the size of other properties.
`Value`	The current or estimated value of the parameter. This is the defining property for size checking and scalar expansions.
`Estimated`	A Boolean array of the same size as that of `Value`. Depending on the value of the elements of the `Estimated` property, the behavior of the corresponding elements of `Value` is as follows: • The elements of `Value` is estimated if the corresponding elements in `Estimate` are set to true. The result is stored in the `Value` property. • The elements of `Value` are not estimated if the corresponding elements in `Estimated` are set to false. However, these elements are used to reset the corresponding workspace parameter during estimations. This property is set to false by default, meaning that the parameter value is not estimated.

Parameter Object Properties (Continued)

Property	Description
`InitialGuess`	Separate properties are required to hold the initial and current values of the parameters. So, when the `InitialGuess` property is initialized with a value, both it and the `Value` property are assigned the same value. Depending on the value of the elements of the `Estimated` property, the behavior of the corresponding elements of `InitialGuess` is as follows: • If any element in `Estimated` is set to true, then the corresponding element of `InitialGuess` is used to initialize the workspace parameter during estimations. • If any element in `Estimated` is set to false, then the corresponding element of `InitialGuess` is not used in any way.
`Minimum`, `Maximum`	Parameter range.
`TypicalValue`	The typical values of the parameters. This property is used in estimations for scaling purposes. The default value is `1`.

Example: F14 Model. To create a parameter object for the parameter `Ta` in the `f14` model, use

```
par1 = ParameterEstimator.Parameter('Ta')
(1) Parameter data for 'Ta':
Parameter value : 0.05
Initial value : 0.05
Estimated : false
Referenced by the blocks:
f14/Actuator Model
```

Example: Gain Matrix. To create a parameter object for a matrix parameter `K` of size 4-by-1, use

```
par1 = ParameterEstimator.Parameter('K', [1 2 3 4]')
(1) Parameter data for 'K':
Parameter value : [1;2;3;4]
Initial value : [1;2;3;4]
Estimated elements : [false;false;false;false]
Referenced by the blocks:
```

Modifying Properties. After a parameter object is created, you can modify its properties using this syntax:

```
par1.Estimated = true; % Estimate this parameter
```

Most of the properties, for example, `Estimated` and `TypicalValue` support scalar expansion with respect to the size of `Value`.

Using Class Methods. Descriptions of two important methods are given next:

- `hiliteBlock` — Highlights the referenced blocks associated with parameter objects in the Simulink diagram.
- `update` — Updates the parameter object after the Simulink model has been modified. If the size of the `Value` property changes, then the other properties are reset to their default values.

Creating State Objects

- "What is a State Object" on page 2-94
- "Constructor" on page 2-94
- "Properties of State Objects" on page 2-94
- "Example: F14 Model" on page 2-96
- "Modifying Properties" on page 2-96
- "Using Class Methods" on page 2-97

What is a State Object. The `@State` object is similar to the `@Parameter` object, and refers to the states of the Simulink model marked for estimation. One `@State` object corresponds to each Simulink block with states in the model.

Constructor. The syntax to create a state object is

```
h = ParameterEstimator.State('block');
h = ParameterEstimator.State('block', Value);
h = ParameterEstimator.State('block', Value, Minimum,
    Maximum);
```

In the first case, the state vector is initialized from the model containing the `block`. In the other cases, `block` does not need to exist in the workspace at the time of object creation. However, it is required at estimation time.

Properties of State Objects. Descriptions of some important properties of state objects are given in the following table.

State Object Properties

Property	Description
`Block`	Name of the Simulink block whose states are defined by this object.
`Dimensions`	Scalar value to store the number of states of the relevant block.
`Value`	Column vector to store the value of the state for the block specified by this object. The length of this vector should be consistent with the `Dimensions` property.

State Object Properties (Continued)

Property	Description
`Estimated`	A Boolean array of the same size as that of `Value`. Depending on the value of the elements of the `Estimated` property, the behavior of the corresponding elements of `Value` is as follows: • The elements of `Value` are estimated if the corresponding elements in `Estimate` are set to true. The result is stored in the `Value` property. • The elements of `Value` are not estimated if the corresponding elements in `Estimated` are set to false. However, these elements are used to reset the corresponding states during estimations. This property is set to false by default, meaning that the state value is not estimated.
`InitialGuess`	Separate properties are required to hold the initial and current values of the states. So, when the `InitialGuess` property is initialized with a value, both it and the `Value` property are assigned the same value. Depending on the value of the elements of the `Estimated` property, the behavior of the corresponding elements of `InitialGuess` is as follows: • If any element in `Estimated` is set to true, then the corresponding element of `InitialGuess` is used to initialize the state during estimations. • If any element in `Estimated` is set to false, then the corresponding element of `InitialGuess` is not used in any way.
`Minimum`, `Maximum`	State vector range.
`TypicalValue`	The typical values of the states. This property is used in estimations for scaling purposes. The default value is `1`.

State Object Properties (Continued)

Property	Description
`Ts`	Sampling time of discrete blocks. Set to zero for continuous blocks. This property is read-only and is currently used for information only.
`Domain`	String to hold the physical domain of the block. Used for SimMechanics™ or SimPowerSystems™ blocks with states.

Example: F14 Model. To create a state object for the `f14/Actuator Model` block in the `f14` model, use

```
st1 = ParameterEstimator.State(gcb)
```

This command returns the following result:

```
(1) State data for f14/Actuator Model block:

    The block has 1 continuous state(s).

           State value : 0
         Initial guess : 0
             Estimated : false
```

Modifying Properties. After a state object is created, you can modify its properties using this syntax:

```
ic1.Estimated = true; % Estimate this state
```

Most of the properties, for example, `Estimated` and `TypicalValue`, support scalar expansion with respect to the size of `Value`.

Using Class Methods. Description of two important methods are given next:

- `hiliteBlock` — Highlights the referenced blocks associated with state objects in the Simulink diagram.
- `update` — Updates the state object after the Simulink model has been modified. If the size of `Value` property changes, then the other properties are reset to their default values.

Creating Estimation Objects

- "What is an Estimation Object" on page 2-97
- "Constructor" on page 2-97
- "Properties of Estimation Objects" on page 2-97
- "Example: F14 Model" on page 2-99
- "Modifying Properties" on page 2-99
- "Using Class Methods" on page 2-99

What is an Estimation Object. The `@Estimation` object defines the estimation problem, and is the coordinator between the model, experiments, parameter objects, and state objects.

Constructor. The `@Estimation` object is the coordinator of the model, experiment, and parameter objects. The syntax to create an estimation object is

```
h = ParameterEstimator.Estimation('model');
h = ParameterEstimator.Estimation('model', hParam);
h = ParameterEstimator.Estimation('model', hParam, hExps);
```

Properties of Estimation Objects. Descriptions of some important properties of estimation objects are given in the following table.

Estimation Object Properties

Property	Description
`Model`	Name of the Simulink model with which this estimation is associated.
`Experiments`	Experiments to be used in estimations. For multiple experiments, the cost function uses a concatenation of the output error vectors obtained using each experimental data set.
`Parameters`	Parameter objects to be used in estimations.
`States`	State objects to be used in estimations. This is a handle matrix with as many columns as there are experiments, and as many rows as there are states in `Model`. The handle matrix is created automatically in the constructor. You can reorganize its rows to specify shared states between experiments, and set the `Estimated` flag of desired states. If state data is provided in an experiment, the state objects stored in the columns of this matrix are initialized from the experiments.
`SimOptions`	Same as `simset` structure. This property is initialized to `simget(this.Model)`.
`OptimOptions`	Same as `optimset` structure.

`EstimInfo`	This property is used to store estimation-related information at each iteration of the optimizer, and is initialized as
	
	`this.EstimInfo = struct( 'Cost', [],...`
	`'Covariance', [],...`
	`'FCount', [],...`
	`'FirstOrd', [],...`
	`'Gradient', [],...`
	`'Iteration', [],...`
	`'Procedure', [],...`
	`'StepSize', [],...`
	`'Values', [] );`

Example: F14 Model. To create an estimation object for the `f14` model to estimate the parameters `Ta` and `Kf` and two states, use

```
exp1 = ParameterEstimator.TransientExperiment(gcs);
par1 = ParameterEstimator.Parameter('Ta', 'Estimated', true);
par2 = ParameterEstimator.Parameter('Kf', 'Estimated', true);
est1 = ParameterEstimator.Estimation(gcs, [par1, par2], exp1);
est1.States(1,1).Estimated = true;
est1.States(6,1).Estimated = true;
est1
```

This command returns the following result:

```
Estimated variables for the model 'f14':

Estimated Parameters

Using Experiments
   (1) f14 experiment

Estimated States for Experiment 'f14 experiment'
   (1) f14/Actuator Model
   (6) f14/Controller/Proportional plus integral compensator
```

Modifying Properties. After an estimation object is created, you can modify its properties using this syntax:

```
est.OptimOptions.Algorithm = 'fmincon'; % Estimation method
est.OptimOptions.Display = 'iter'; % Show estimation information
...in workspace
est.Parameters(1).Estimated = false; % Do not estimate first
...parameter
est.States(2,3).Estimated = false; % Do not estimate second state
...of third expression
```

Using Class Methods. Descriptions of some of the important methods are given next:

- `compare` — Compares an experiment and a simulation.
- `simulate` — Simulates the model with current parameters and states.
- `estimate` — Runs an estimation.
- `restart` — Restarts an estimation after it has finished running.
- `update` — Updates the estimation object after the Simulink model has been modified.

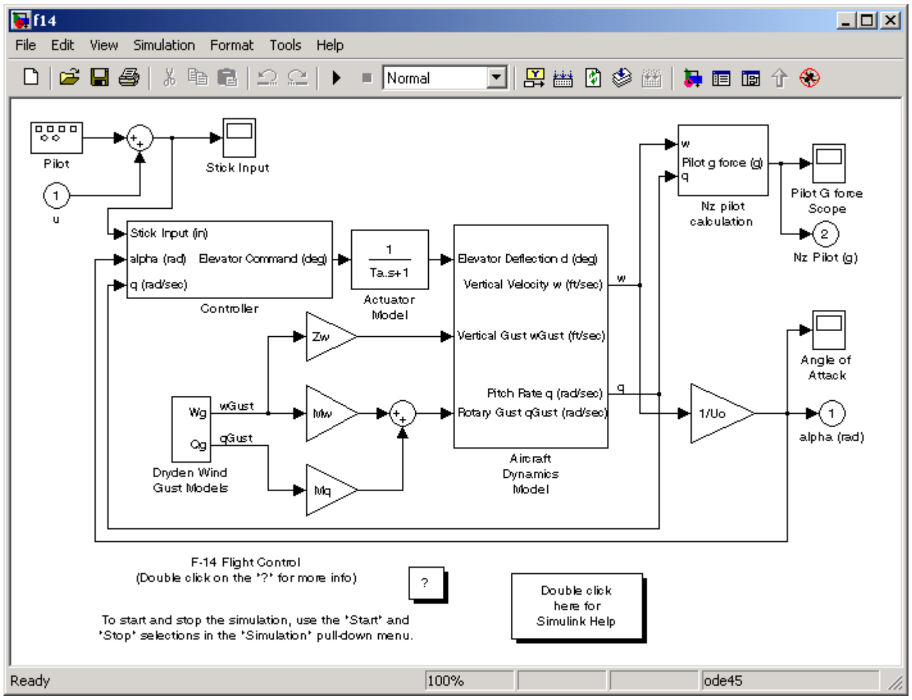
Example — Estimating Parameters and Initial States at the Command Line

Loading the F14 Jet Model

To define an experiment, you must start with a Simulink model. For this example, type

```
f14
```

to load the F14 fighter jet model into the MATLAB workspace. The following figure shows the `f14` model.



F14 Fighter Jet Model

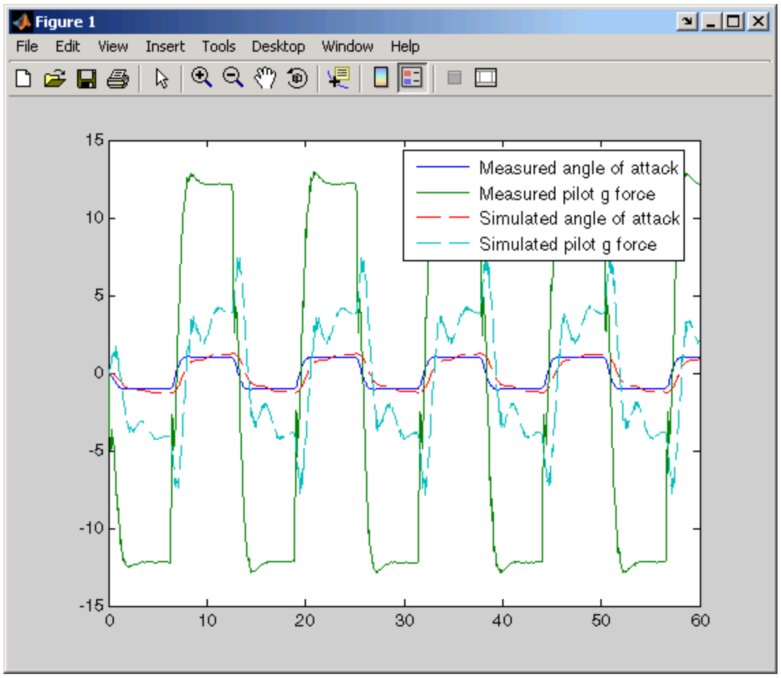
This example outlines the basics of constructing an estimation project using object-oriented code. Only what you need to run the example is presented in this section. See "Objects for Parameter Estimation" on page 2-79 for details on all the properties and methods associated with parameter estimation.

Baseline Simulation

Before running an estimation, you need a baseline for data comparison. First, you must choose parameters and states' initial conditions for estimation. This example uses `Ta`, the actuator time constant, and `Zd` and `Md`, the vertical velocity and pitch rate gains, respectively. Then use the following code to run the Simulink `f14` model. Note that this is standard Simulink code and does not involve Simulink Design Optimization command-line interface in any way. See `sim` in the Simulink Reference documentation for information about running Simulink models from the MATLAB command line.

```
%% Open the model and load experimental data.
open_system('f14')
load f14_estim % Load empirical I/O data.

%% Set initialize unknown parameters
% Actuator time constant (ideal: Ta = 0.05)
Ta = 0.5;

% Aircraft dynamic model parameters (ideal: Md = -6.8847,
% Zd = -63.998)
Md = -1; Zd = -80;

%% Plot measured data and simulation results
[T,X,Y] = sim('f14', time, [], [time iodata(:,1)]);
plot(time, iodata(:,2:3), T, Y, '-');
legend( 'Measured angle of attack',  'Measured pilot g force', ...
        'Simulated angle of attack', 'Simulated pilot g force');
```

The following figure appears.



As you can see, the measured and simulated data are a poor match. The rest of this section describes how to estimate values for `Ta`, `Zd`, and `Md` that result in a better match of data sets.

Creating a Transient Experiment Object

After you have a model and identify the parameters you want to estimate, the next step is to create the objects required for an estimation. `ParameterEstimator` is both the name of the *class* and the *object* instantiated by that class. Classes are created by a *constructor*; objects are created by invoking the class name with parameters.

First, create an estimation project object. This is the constructor syntax:

```
hExp = ParameterEstimator.TransientExperiment('f14')
```

This command returns the following information about the `f14` model.

```
Experimental transient data set for the model 'f14':

Output Data
   (1) f14/alpha (rad)
   (2) f14/Nz Pilot (g)

Input Data
   (1) f14/u

Initial States
   (1) f14/Actuator Model
   (2) f14/Aircraft Dynamics Model/Transfer Fcn.1
   (3) f14/Aircraft Dynamics Model/Transfer Fcn.2
   (4) f14/Controller/Alpha-sensor Low-pass Filter
   (5) f14/Controller/Pitch Rate Lead Filter
   (6) f14/Controller/Proportional plus integral compensator
   (7) f14/Controller/Stick Prefilter
   (8) f14/Dryden Wind Gust Models/Q-gust model
   (9) f14/Dryden Wind Gust Models/W-gust model
```

Assigning Experimental Data to Inputs and Outputs of the Model

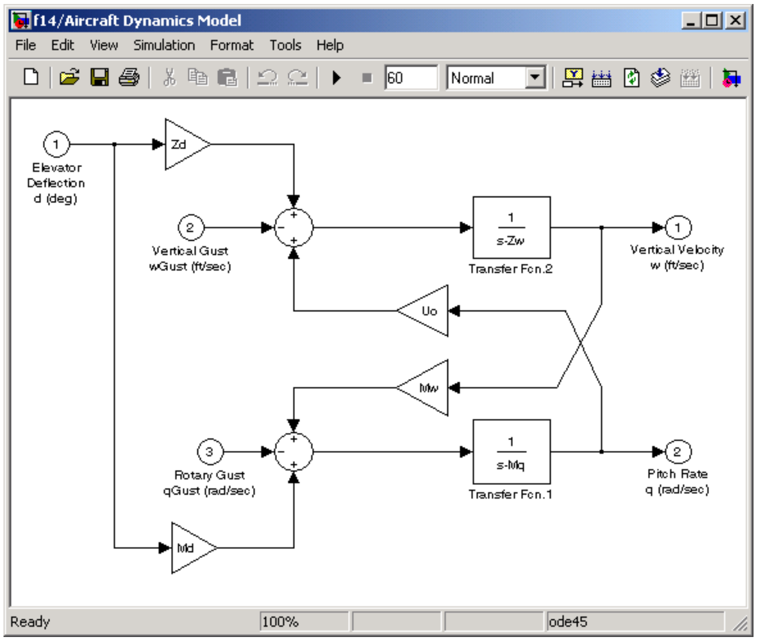
After you create a `ParameterEstimator` object, assign input and output experimental (i.e., empirical) data.

```
%% Create objects to represent the experimental data sets.
set(hExp.InputData(1), 'Data', iodata(:,1), 'Time', time);

set(hExp.OutputData(1), 'Data', iodata(:,2), 'Time', ...
    time, 'Weight', 5);
set(hExp.OutputData(2), 'Data', iodata(:,3), 'Time', time);
```

Note In general, for models with multiple inputs and outputs, you must independently assign one data object to each input and output port. The data object you assign to a specific port can be a vector or a matrix that corresponds to that channel. You cannot use a single I/O port to represent multiple channels.

Creating Parameter Objects for Estimation

To activate parameters for estimation, you must create parameter objects for the parameters you want to estimate. For this example, use `Ta`, the actuator time constant, and `Zd` and `Md`, the vertical velocity and pitch rate gains, respectively. The `Zd` and `Md` gains are located in the F14 aircraft dynamics subsystem.


f14/Aircraft Dynamics Model
File Edit View Simulation Format Tools Help
60
Normal
1
Elevator Deflection d (deg)
Zd
2
Vertical Gust wGust (ft/sec)
1
s-Zw
Transfer Fcn.2
1
Vertical Velocity w (ft/sec)
Uo
Mw
3
Rotary Gust qGust (rad/sec)
1
s-Mq
Transfer Fcn.1
2
Pitch Rate q (rad/sec)
Md
Ready
100%
ode45

First, create `ParameterEstimator` objects for the parameters you want to estimate.

```
%% Create objects to represent parameters.
hPar(1) = ParameterEstimator.Parameter('Ta');
set(hPar(1), 'Minimum', 0.01, 'Maximum', 1, 'Estimated', true)

hPar(2) = ParameterEstimator.Parameter('Md');
set(hPar(2), 'Minimum', -10, 'Maximum', 0, 'Estimated', true)

hPar(3) = ParameterEstimator.Parameter('Zd');
set(hPar(3), 'Minimum', -100, 'Maximum', 0, 'Estimated', true)

%% Create objects to represent initial states.
hIc(1) = ParameterEstimator.State('f14/Actuator Model');
set(hIc(1), 'Minimum', 0, 'Estimated', false);
```

You can also use dot notation here. For example, instead of

```
set(hPar(2), 'Minimum', -10, 'Maximum', 0, 'Estimated', true)
```

you can write

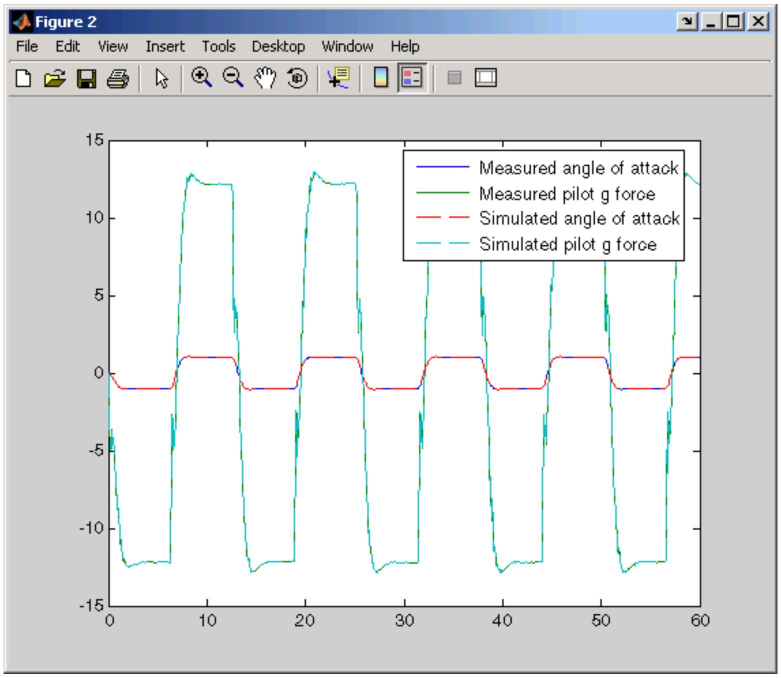
```
hPar(2).Estimated=true;
hPar(2).Minimum=-10;
hPar(2).Maximum=0;
```

Creating an Estimation Object and Running the Estimation

Finally, create an estimation object and run the estimation, using `gcs` to get the full pathname to the Simulink model.

```
hEst = ParameterEstimator.Estimation(gcs, hPar, hExp);
hEst.States = hIc;

%% Setup estimation options
hEst.OptimOptions.Algorithm    = 'lsqnonlin';
hEst.OptimOptions.GradientType = 'refined';
hEst.OptimOptions.Display      = 'iter';

%% Run the estimation
estimate(hEst);

%% Plot measured data and final simulation results
[T,X,Y] = sim('f14', time, [], [time iodata(:,1)]);
figure
plot(time, iodata(:,2:3), T, Y, '-');
legend( 'Measured angle of attack',  'Measured pilot g force', ...
        'Simulated angle of attack', 'Simulated pilot g force');
```

This figure shows the results of the estimation.



The measured and simulated outputs now appear to be a close match. Next, look at the estimated values to see how they compare with the default values of the `f14` model.

```
%% Look at the estimated values
find(hEst.Parameters, 'Estimated', true)
```

This command returns the following result:

```
(1) Parameter data for 'Ta':

       Parameter value : 0.05
         Initial guess : 0.5

             Estimated : true

          Referenced by:

(2) Parameter data for 'Md':

       Parameter value : -6.884
         Initial guess : -1

             Estimated : true

          Referenced by:

(3) Parameter data for 'Zd':

       Parameter value : -63.99
         Initial guess : -80

             Estimated : true

          Referenced by:
```

Note You can use the `find` command to identify scalar, vector, or matrix parameters. The dimensions of the `Estimated` value you specify as the `find` argument must match the dimensions of the parameters you are trying to find. For example,

```
find(hEst.Parameters, 'Estimated', true)
```

finds only scalar estimated parameters. However,

```
find(hEst.Parameters, 'Estimated', [true;true])
```

finds only vector estimated parameters with dimensions 1-by-2 and excludes all scalar parameters.

You can verify that these values match the default values of the `f14` model by clearing your workspace, loading the model, and checking the values.

```
clear all
f14
whos
```

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 2-53, you can estimate the model parameters using the command-line functions. To learn more about parameter estimation using parallel computing, see "When to Use Parallel Computing for Estimating Model Parameters" on page 2-49, and "How Parallel Computing Speeds Up Parameter Estimation" on page 2-50.

To use parallel computing for parameter estimation at the command line:

1. Open the Simulink model by typing the model name at the MATLAB prompt.

2. Configure an estimation project, as described in "Workflow for Estimating Parameters at the Command Line" on page 2-78.

3 Enable the parallel computing option in the estimation project by typing the following command:

```
hEst.OptimOptions.UseParallel='always';
```

To view that the `UseParallel` property has been set, type the following command:

```
hEst.OptimOptions
```

4 Find the model path dependencies by typing the following command:

```
dirs=hEst.finddepend;
```

This command returns the model path dependencies in your Simulink model in the `dirs` cell array.

Note As described in "Model Dependencies" on page 2-53, 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 by typing the following command.

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

6 Assign the path dependencies to the estimation project by typing the following command:

```
hEst.OptimOptions.ParallelPathDependencies=dirs;
```

7 Run the estimation by typing the following command:

```
estimate(hEst);
```

For more information on how to troubleshoot estimation results you obtained using parallel computing, see "Troubleshooting" on page 2-59.

Optimizing Model Parameters

Overview of Optimizing Model Parameters

When you optimize parameters of a Simulink model to meet time-domain design requirements, Simulink Design Optimization software automatically converts time-domain requirements into a constrained optimization problem, and then solves the problem using optimization techniques. The constrained optimization problem iteratively simulates the Simulink model, compares the results of the simulations with the constraint objectives, and uses gradient methods to adjust tuned parameters to better meet the objectives.

The process for optimizing model parameters to meet time-domain design requirements consists of the following task:

1. "Choosing Signals to Constrain" on page 3-3
2. "Specifying Design Requirements" on page 3-5
3. "Specifying Parameters to Optimize" on page 3-18
4. "Specifying Optimization Options" on page 3-24
5. "Running the Optimization" on page 3-36

Configuring Parameter Optimization

In this section...

Choosing Signals to Constrain

Simulink Design 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.

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 in the Simulink Design Optimization library in the Simulink Library Browser. Alternatively, you can open Simulink Design Optimization library by typing `sdolib` 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).

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. For more information, see "Specifying Design Requirements" on page 3-5. You can also specify parameters to optimize and optimization settings in this block.

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 Design Requirements" on page 3-5
- "Specifying Parameters to Optimize" on page 3-18

To save the project for use in a later session, see “Saving and Loading Response Optimization Projects” on page 3-79.

Specifying Design Requirements

Enforcing Signal Bounds

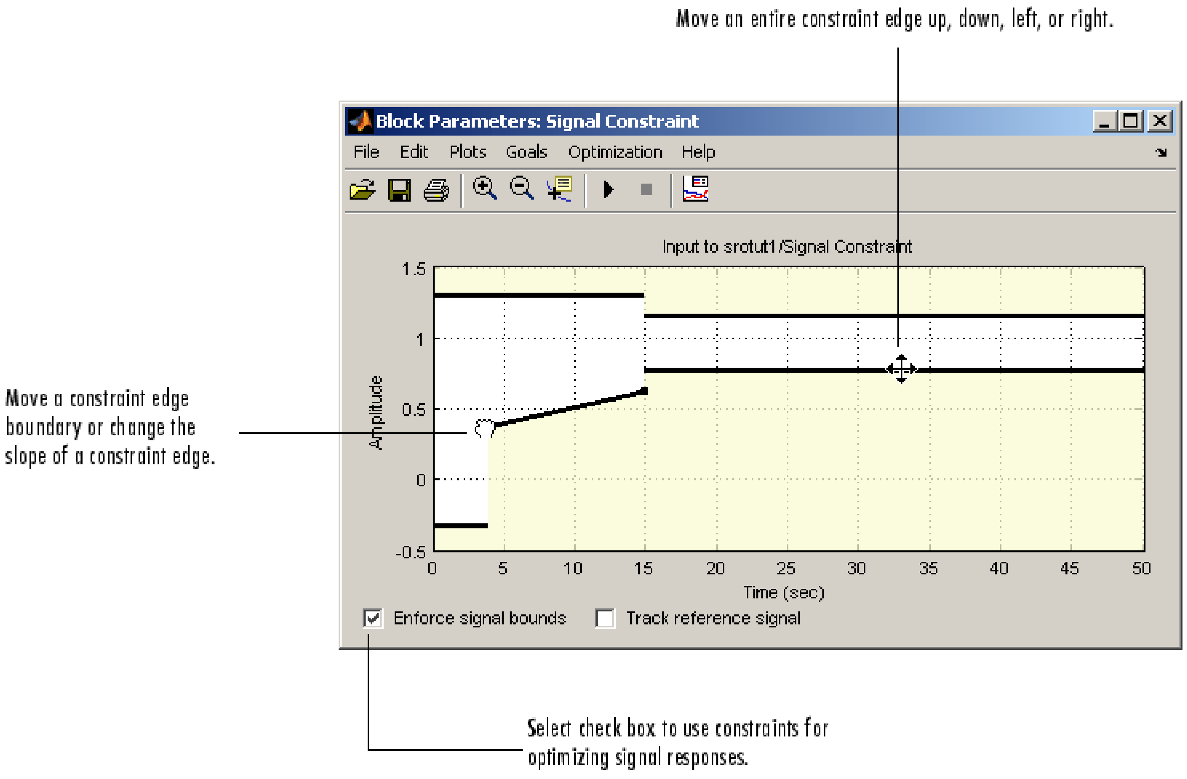
You can specify the desired response of a signal by enforcing signal bounds or by tracking a reference signal. To enforce signal bounds, select this option at the bottom of the Signal Constraint window, and then position time-domain-based constraint bound segments in the Signal Constraint window. To 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.

When using a Signal Constraint block to directly optimize a Simulink model, by default, the start and stop time are inherited from the Simulink model. However, you can change them with the Simulation Options dialog box. Choose a stop time that captures enough of the desired response's characteristics. When you want the response to settle to a final value, use at least 10 to 20% of the simulation time for constraining the steady-state response. This ensures the proper weighting of requirements on the final value and overall stability.

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.


Move an entire constraint edge up, down, left, or right.
Move a constraint edge boundary or change the slope of a constraint edge.
Select check box to use constraints for optimizing signal responses.
Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Input to srotut1/Signal Constraint
Amplitude
Time (sec)
1.5
1
0.5
0
-0.5
0 5 10 15 20 25 30 35 40 45 50
Enforce signal bounds
Track reference signal

- 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.

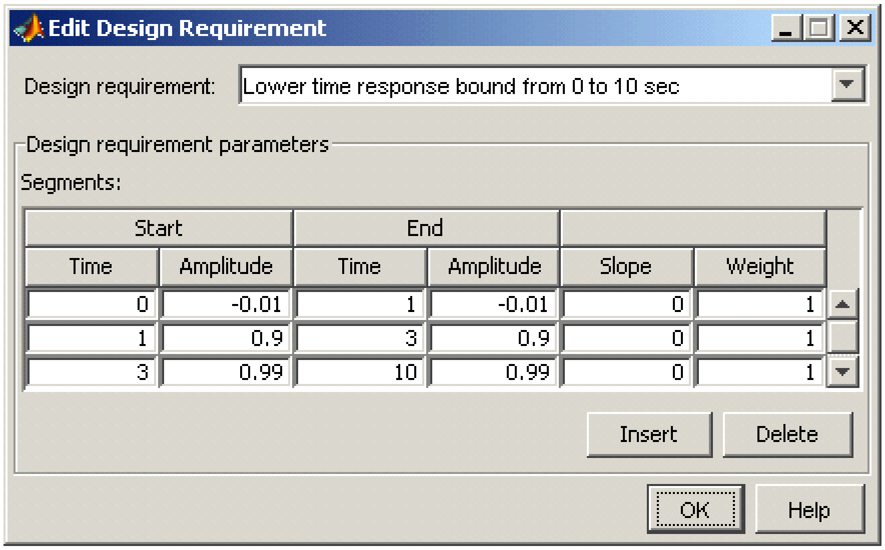
Note It is possible to move a lower bound constraint segment above an upper bound constraint segment, or vice versa, but this produces an error when you attempt to run the optimization.

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 3-9.



Design requirement: Lower time response bound from 0 to 10 sec

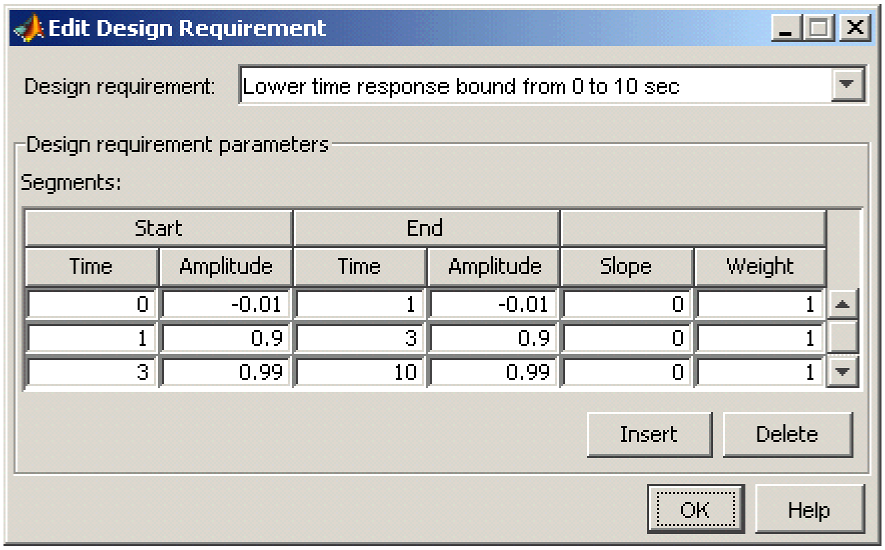
Design requirement parameters

Segments:

Start		End			
Time	Amplitude	Time	Amplitude	Slope	Weight
0	-0.01	1	-0.01	0	1
1	0.9	3	0.9	0	1
3	0.99	10	0.99	0	1

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 3-9.



Design requirement: Lower time response bound from 0 to 10 sec

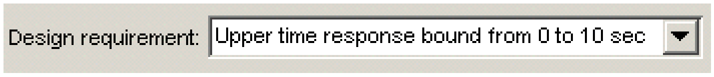
Start		End			
Time	Amplitude	Time	Amplitude	Slope	Weight
0	-0.01	1	-0.01	0	1
1	0.9	3	0.9	0	1
3	0.99	10	0.99	0	1

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.



Design requirement: Upper time response bound from 0 to 10 sec

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)

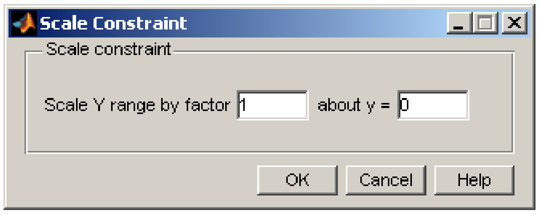
Parameter	Found in	Description
Natural frequency	SISO Tool Root Locus Editor	Defines a constraint segment for a particular natural frequency. To specify the constraint, choose `at least` or `at most` from the menu, and then specify the natural frequency of interest.
Real	SISO Tool Root Locus Editor	Defines 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.

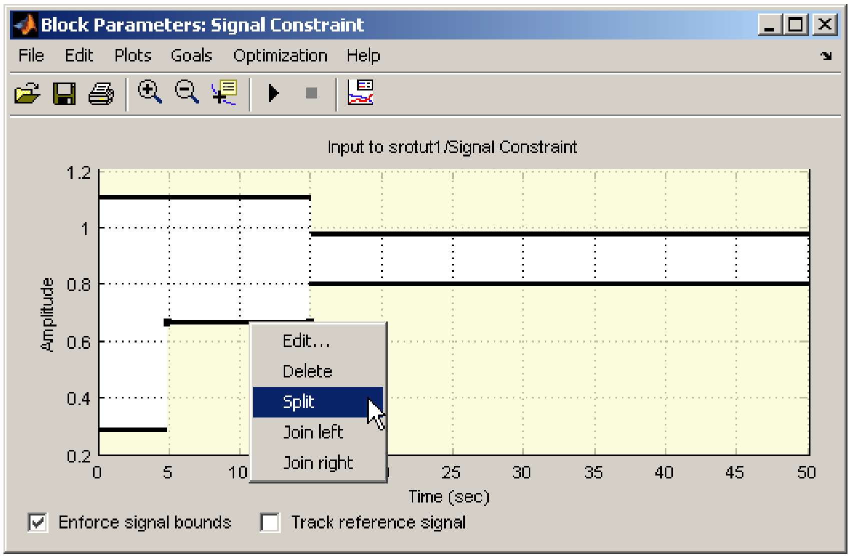


Scale Y range by factor 1 about y = 0

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.

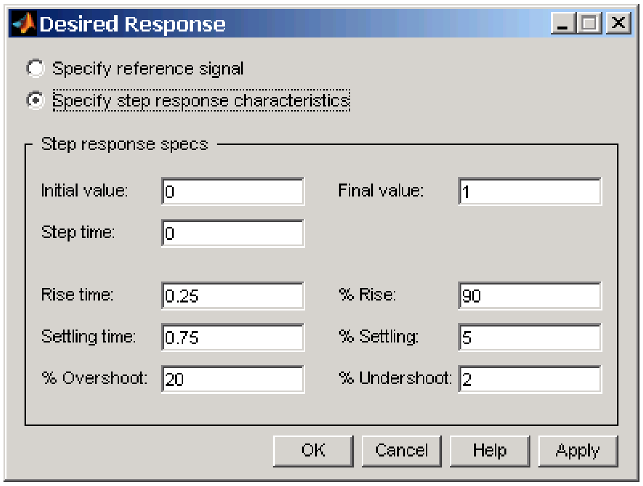

Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Input to srotut1/Signal Constraint
Amplitude
Time (sec)
Edit...
Delete
Split
Join left
Join right
Enforce signal bounds
Track reference signal

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.



( ) Specify reference signal
(●) Specify step response characteristics

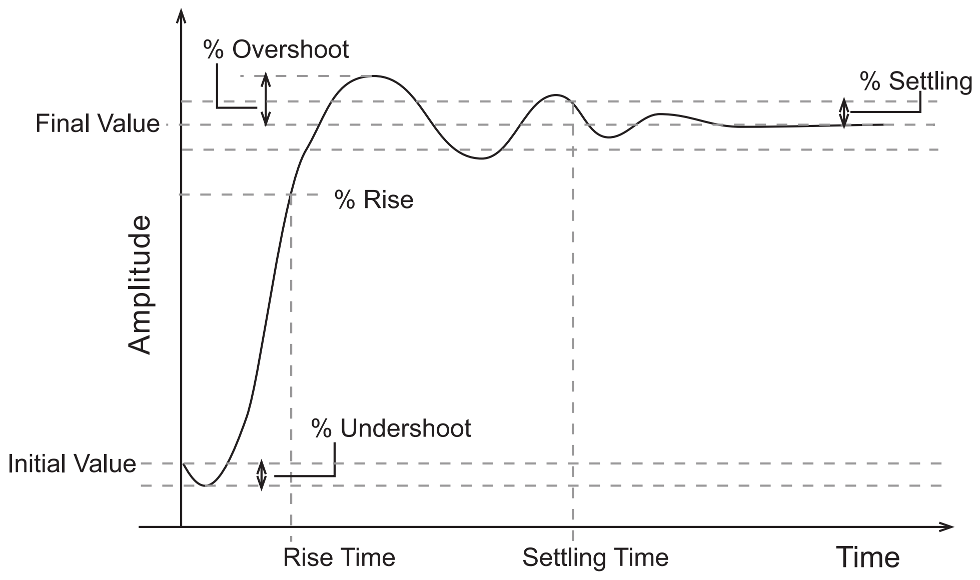
Step response specs

Initial value:	0	Final value:	1
Step time:	0		
Rise time:	0.25	% Rise:	90
Settling time:	0.75	% Settling:	5
% Overshoot:	20	% Undershoot:	2

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.

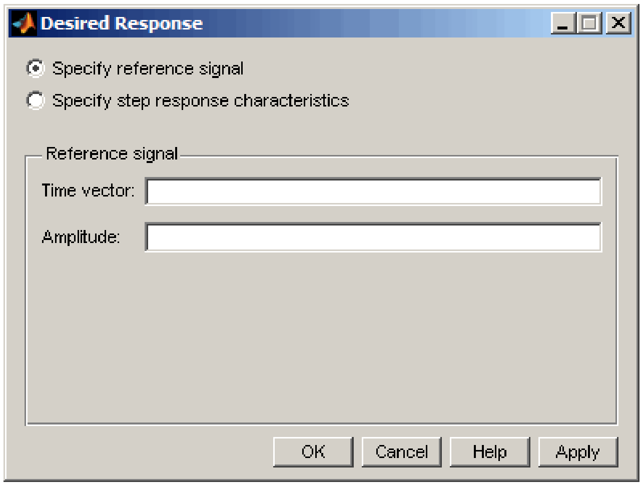
Enter values for the response specifications in the Response Specifications dialog box, based on the requirements of your model, and then click **OK**. The constraint segments now reflect the constraints specified.

Tracking Reference Signals

- "Specifying a Reference Signal" on page 3-17
- "Plotting the Reference Signal" on page 3-17

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.


Desired Response

[✓ Specify reference signal]
Specify step response characteristics

Reference signal
Time vector:
Amplitude:

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**.

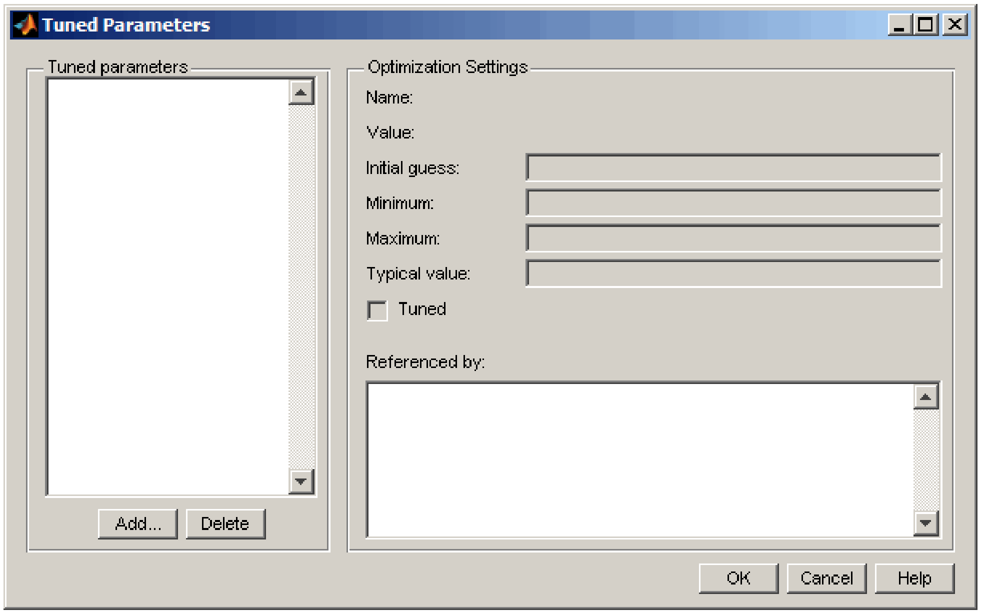
Specifying Parameters to Optimize

Defining Tunable Parameters

Before running the optimization, you need to define which system parameters are tunable. By tuning these parameters, Simulink Design 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 Design 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.

Specifying 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:

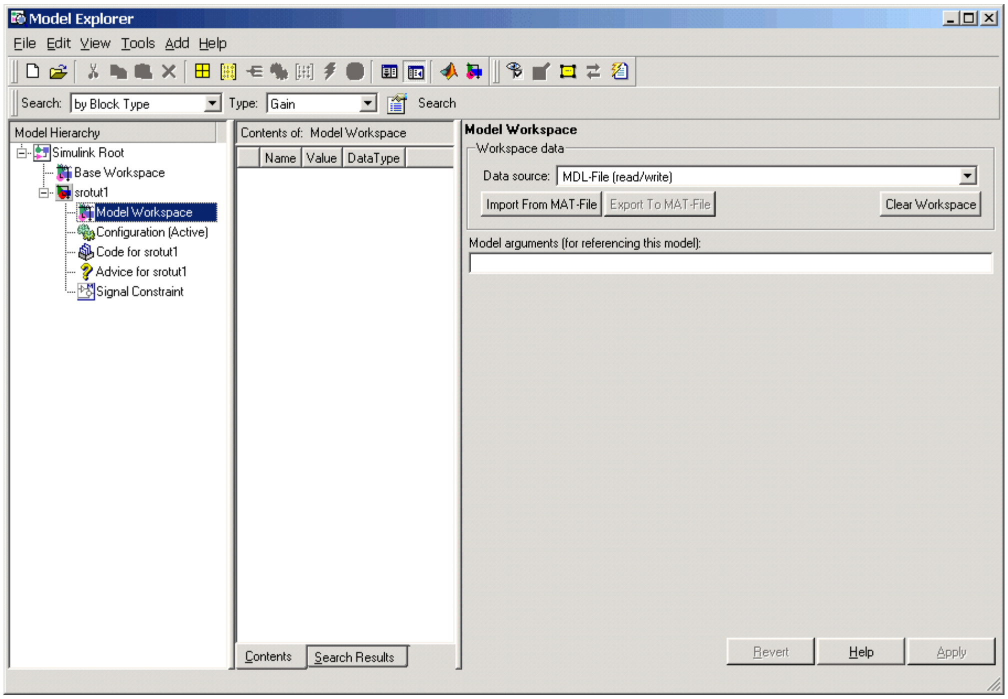
1. Add the independent parameters to the model workspace (along with initial values).
2. Define a Simulation Start function that runs before each simulation of the model. This Simulation Start function defines the relationship between the dependent parameters in the model and the independent parameters 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 — Specifying Independent Parameters

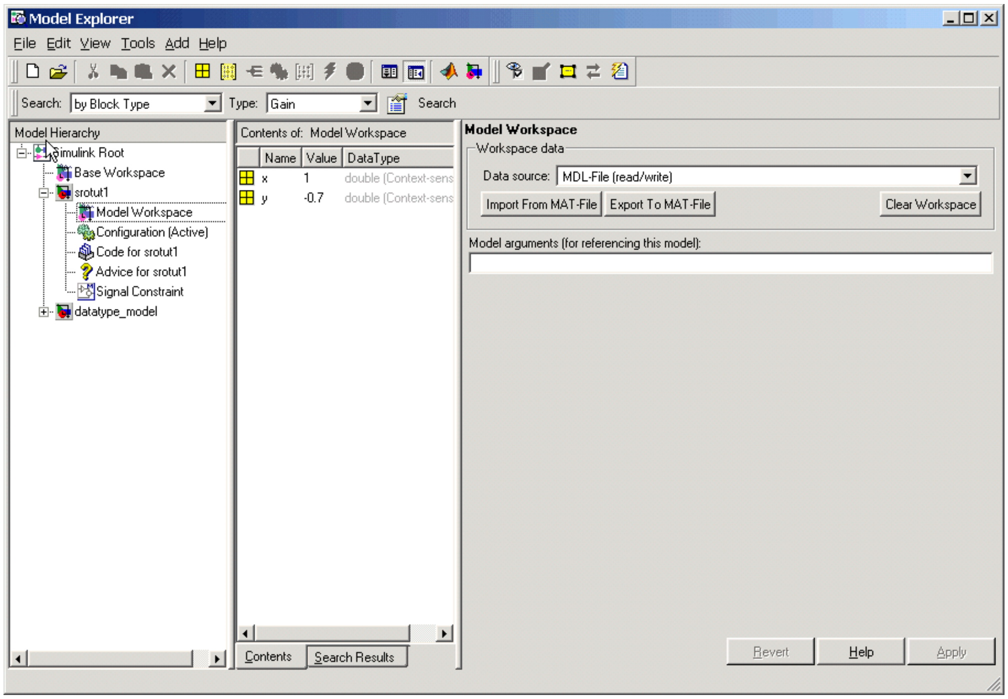
Assume that the parameter `Kint` in the model `srotut1` is related to the parameters `x` and `y` according to the relationship `Kint=x+y`. Also assume that the initial values of `x` and `y` are 1 and -0.7, respectively. To tune `x` and `y` instead of `Kint`, first define these parameters in the model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



Search: by Block Type
Type: Gain

Name	Value	DataType
x	1	double (Context-sens
y	-0.7	double (Context-sens

Data source: MDL-File (read/write)

Model arguments (for referencing this model):

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`.

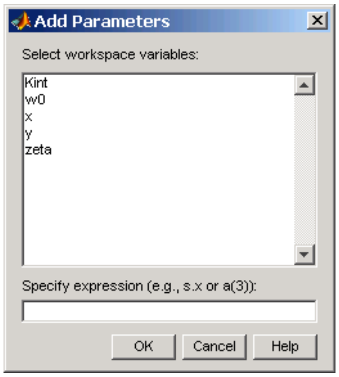
7 Create a new M-file with this name. The contents of the M-file should define the relationship between the parameters in the model and the parameters in the workspace. For this example, the M-file should look something like the following:

```
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.

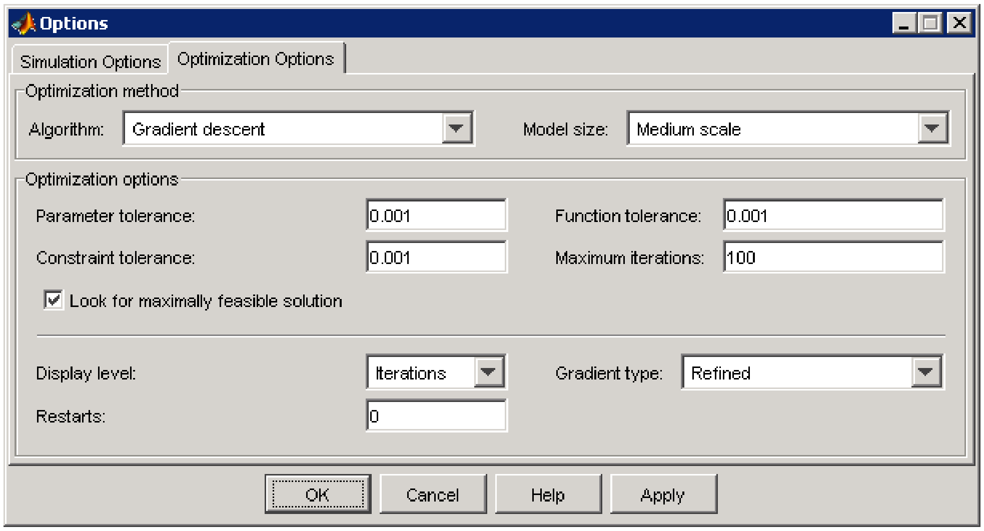

Add Parameters
Select workspace variables:
Kint
w0
x
y
zeta
Specify expression (e.g., s.x or a(3)):
OK
Cancel
Help

Specifying Optimization Options

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.



Optimization method

Algorithm: Gradient descent

Model size: Medium scale

Optimization options

Parameter tolerance: 0.001

Function tolerance: 0.001

Constraint tolerance: 0.001

Maximum iterations: 100

☑ Look for maximally feasible solution

Display level: Iterations

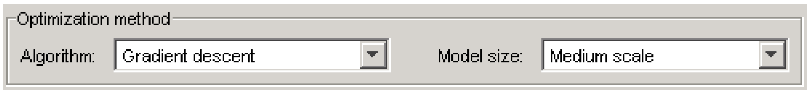
Gradient type: Refined

Restarts: 0

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 3-27.

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.


Optimization method

Algorithm: Gradient descent

Model size: Medium scale

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

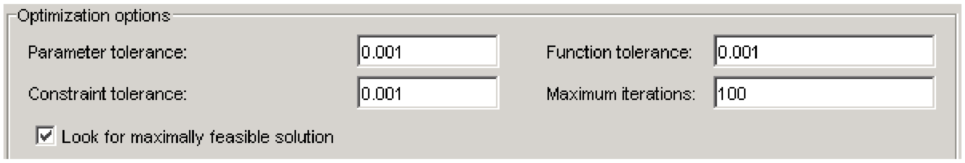
the response. This option requires the Genetic Algorithm and Direct Search Toolbox.

- `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.



Optimization options

Parameter tolerance: 0.001

Function tolerance: 0.001

Constraint tolerance: 0.001

Maximum iterations: 100

[✓] Look for maximally feasible solution

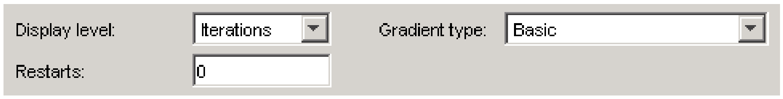
- **Parameter tolerance**: When using the `Simplex search` algorithm, the optimization terminates when successive parameter values change by less than this number. For more details, refer to the discussion of the parameter `TolX` in the reference page for the Optimization Toolbox function `fmincon`.
- **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: Iterations
Gradient type: Basic
Restarts: 0

- "Display Level" on page 3-27
- "Restarts" on page 3-28
- "Gradient Type" on page 3-28

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 Design 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`.

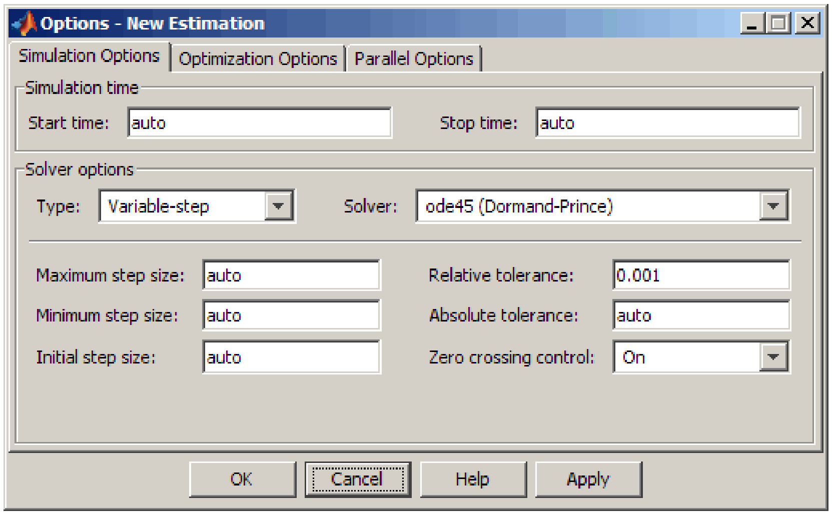
Specifying the Simulation Options

- "Accessing Simulation Options" on page 3-29
- "Selecting Simulation Time" on page 3-29
- "Selecting Solvers" on page 3-30

Accessing Simulation Options

To optimize the response signals of a model, Simulink Design 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.



Options - New Estimation

Simulation Options | Optimization Options | Parallel Options

Simulation time

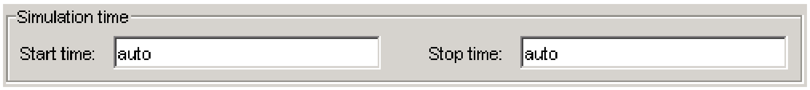
Start time: auto
Stop time: auto

Solver options

Type: Variable-step
Solver: ode45 (Dormand-Prince)

Maximum step size: auto
Minimum step size: auto
Initial step size: auto
Relative tolerance: 0.001
Absolute tolerance: auto
Zero crossing control: On

Selecting Simulation Time



Simulation time

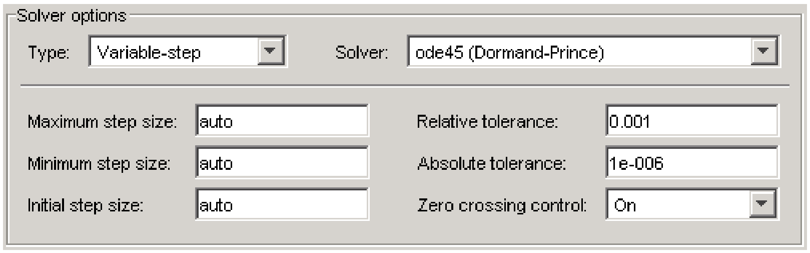
Start time: auto
Stop time: auto

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 Design 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.



Solver options

Type: Variable-step　Solver: ode45 (Dormand-Prince)

Maximum step size: auto　Relative tolerance: 0.001

Minimum step size: auto　Absolute tolerance: 1e-006

Initial step size: auto　Zero crossing control: On

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.

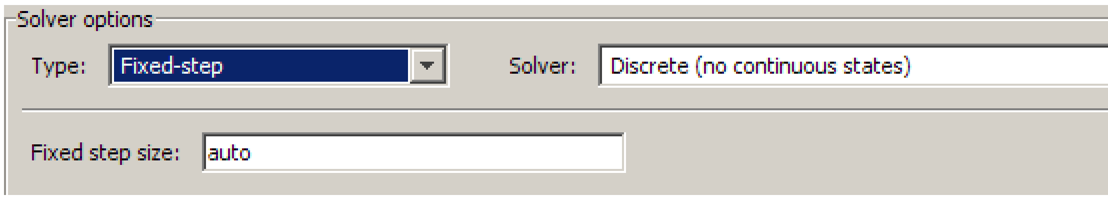
By default, the values for these options are automatically chosen. To choose your own values, enter them in the appropriate fields. For more information on these options, and the circumstances in which to use them, see the Simulink documentation.

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.


Solver options
Type: Fixed-step
Solver: Discrete (no continuous states)
Fixed step size: auto

Plotting Responses in the Signal Constraint Window

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 3-17.

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.

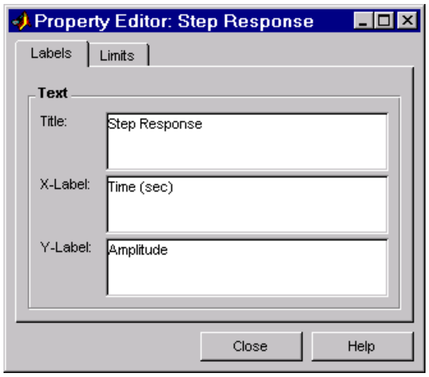
Response Plots Property Editor

- "Modifying Properties of Response Plots" on page 3-33
- "Labels Pane" on page 3-34
- "Limits Pane" on page 3-35

Modifying Properties of Response Plots. This section discusses how you can change the properties of response plots. Select **Edit > Axes Properties** in the Block Parameters: Signal Constraint window and select **Labels** to open the Property Editor dialog box.

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.



Text
Title: Step Response
X-Label: Time (sec)
Y-Label: Amplitude

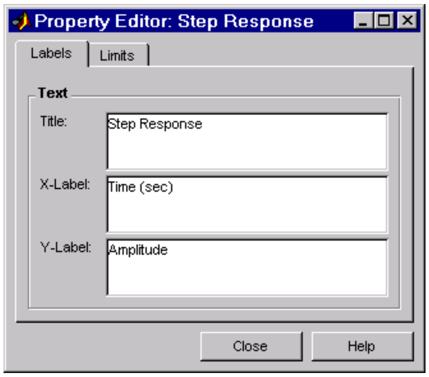
In general, you can change the following properties of response plots.

- **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.

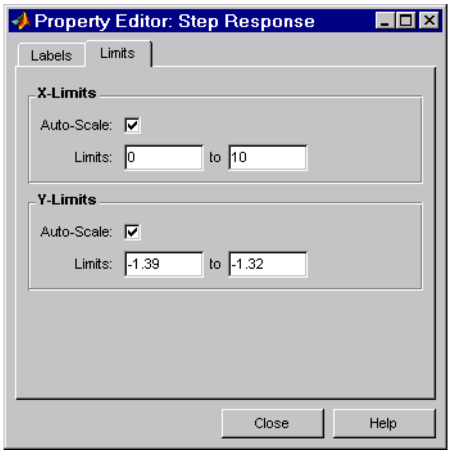


Title: Step Response
X-Label: Time (sec)
Y-Label: Amplitude

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.



X-Limits
Auto-Scale: ✓
Limits: 0 to 10

Y-Limits
Auto-Scale: ✓
Limits: -1.39 to -1.32

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.

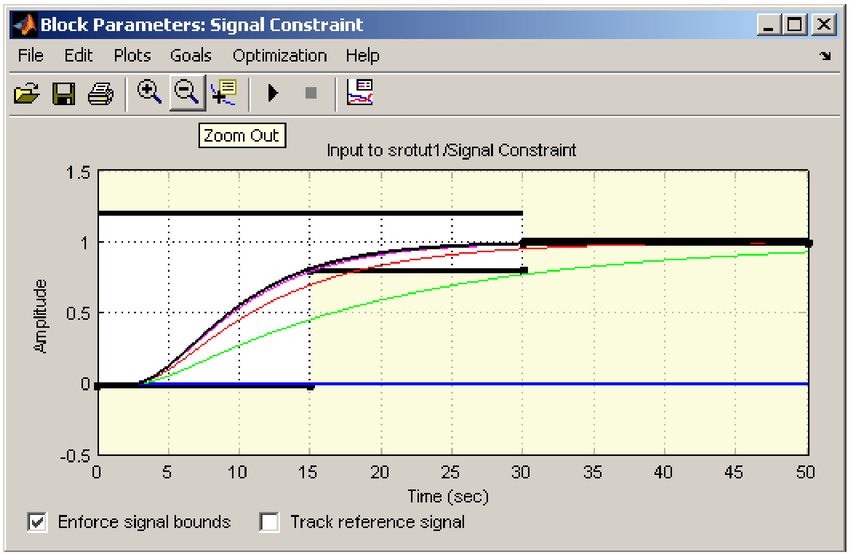
Running the Optimization

After you have specified constraints and the parameters to optimize, as described in "Specifying Design Requirements" on page 3-5 and "Specifying Parameters to Optimize" on page 3-18 respectively, you can 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 Design 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.

Simulink Design 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.

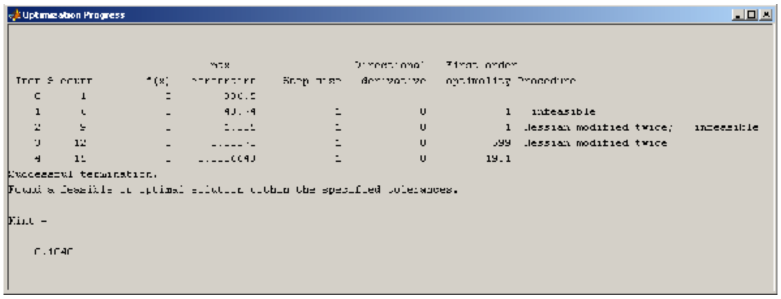

Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Zoom Out
Input to srotut1/Signal Constraint
Amplitude
Time (sec)
1.5
1
0.5
0
-0.5
0 5 10 15 20 25 30 35 40 45 50

☑ Enforce signal bounds ☐ Track reference signal

Simulink Design 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 3-25 and the discussion of **Display level** in “Selecting Additional Optimization Options” on page 3-27.



Optimization Progress

```
                               Max                    Directional   First-order
 Iter F-count        f(x)   constraint    Step-size   derivative   optimality Procedure
    0      1            [illegible]    [illegible]
    1      [illegible]  [illegible]    [illegible]      1            0            1  infeasible
    2      [illegible]  [illegible]    [illegible]      1            0            1 Hessian modified twice;   infeasible
    3     12            [illegible]    [illegible]      1            0          [illegible] Hessian modified twice
    4     15            [illegible]    [illegible]      1            0         19.1
Successful termination.
Found a feasible or optimal solution within the specified tolerances.

Kint =

   [illegible]
```

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 “Troubleshooting Optimization Results” on page 3-69.

If the optimization does not converge the first time, it often converges after adjusting the constraints or tuned parameter characteristics, or choosing different options. For more information, see “Troubleshooting Optimization Results” on page 3-69.

Optimizing Parameters for Model Robustness

In this section...

What Is Model Robustness?

A model is *robust* when it's response does not violate design requirements under parameter variations. When you optimize model parameters, your model may contain additional parameters whose values are not precisely known. Such parameters vary over a given range of values and are defined as *uncertain parameters*. You may know the nominal value and the range of values in which these uncertain parameters vary.

You can then use the Simulink Design Optimization software to incorporate the parameter uncertainty to test the robustness of your design. You can test and optimize parameters for model robustness in the following ways:

- **Before Optimization**. Specify the parameter uncertainty *before* you optimize the parameters to meet the design requirements. In this case, the optimization algorithm optimizes the signals based on both nominal parameter values as well as the uncertain values. This mode requires more time.
- **After Optimization**. Specify the parameter uncertainty *after* you have optimized the model parameters to meet design requirements. You can then test the effect of the uncertain parameters by plotting the model's response. If the response violates the design requirements, you can optimize the parameters again by including the parameter uncertainty during the optimization.

To learn more, see “Example — Optimizing Parameters for Model Robustness Using the GUI” on page 3-47.

Note You cannot add uncertainty to controller or plant parameters when designing controllers using optimization-based methods in the SISO Design Tool.

Sampling Methods for Computing Uncertain Parameter Values

There are two sampling methods for computing uncertain parameter values. Both methods create several sample parameter values within the range of uncertainty, as described in the following sections:

- “**Random (Monte Carlo)** Method” on page 3-40
- “**Grid** Method” on page 3-41

To learn how to specify the sampling method, see the following sections:

- “How to Optimize Parameters for Model Robustness Using the GUI” on page 3-43
- `gridunc` and `randunc` function reference pages

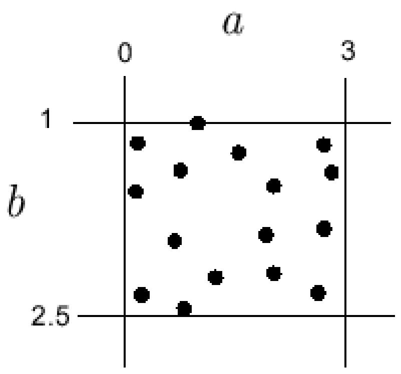
Random (Monte Carlo) Method

The **Random (Monte Carlo)** sampling method computes random values of uncertain parameters within a specified range. When you select the **Random (Monte Carlo)** method, you must also specify the following settings:

- Number of sample values
- Nominal parameter value
- Range of parameter values

When you specify more than one uncertain parameter, the sampling method creates random parameter values within a hypercube. This hypercube is defined by the minimum and maximum values of all uncertain parameters.

For example, the following figure shows two uncertain parameters, *a* and *b*, which range in value from 0 to 3 and 1 to 2.5 respectively. In the figure, the sample values appear as black dots and are scattered randomly within the rectangle.



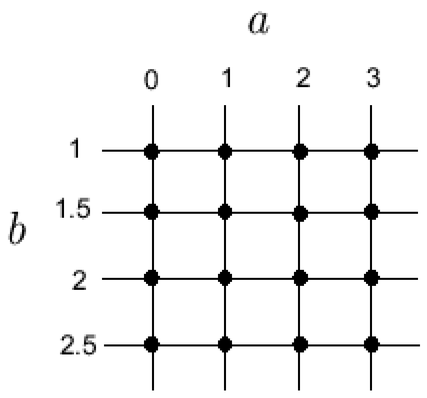
Grid Method

The **Grid** sampling method computes specified values of uncertain parameters within the range of uncertainty. When you select the **Grid** method, you must specify the following settings:

- Nominal parameter value
- Vector of sample parameter values

  The sampling method uses the sample parameter values to compute the number of samples for the uncertain parameter.

When you specify more than one uncertain parameter, the sample values form a grid of combinations. For example, the following figure shows two uncertain parameters, *a* and *b*, with sample values `[0 1 2 3]` and `[1 1.5 2 2.5]`. In the figure, the sample values appear as black dots to form the grid.

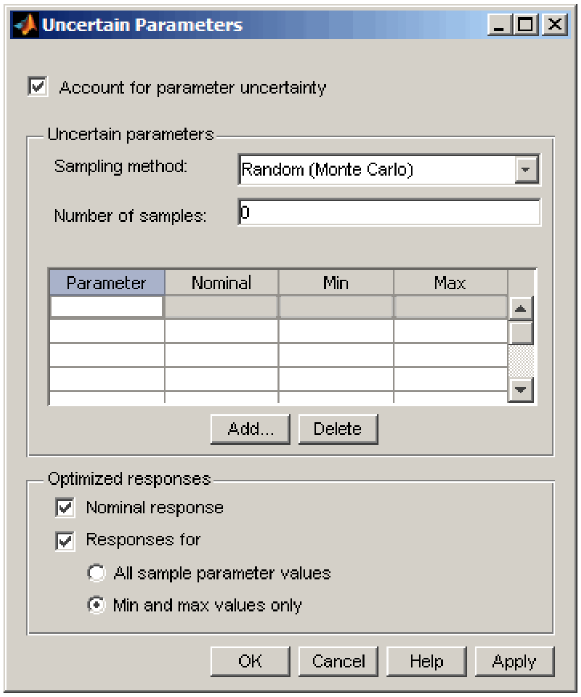

a
0
1
2
3
b
1
1.5
2
2.5

How to Optimize Parameters for Model Robustness Using the GUI

To optimize parameters for model robustness using the GUI:

1 In the Block Parameters: Signal Constraint window, select **Optimization > Uncertain Parameters**.

This action opens the Uncertain Parameters dialog box.



[✓] Account for parameter uncertainty

Uncertain parameters

Sampling method: Random (Monte Carlo)

Number of samples: 0

Parameter	Nominal	Min	Max

Optimized responses

[✓] Nominal response

[✓] Responses for

All sample parameter values [✓ Min and max values only]

By default, the **Account for parameter uncertainty** check box is selected. This implies that the optimization algorithm takes into account the parameter uncertainty during optimization. You can exclude the parameter uncertainty during optimization by clearing this option.

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

---

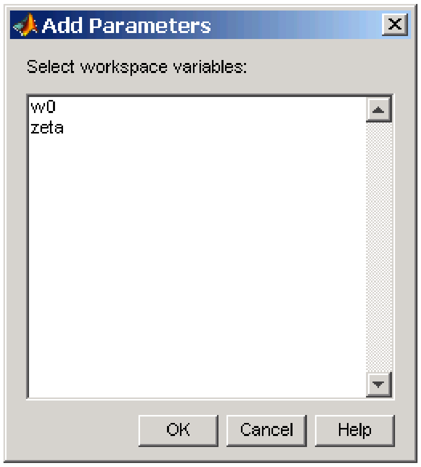
2 Select the sampling method from the **Sampling method** drop-down list.

To learn more about the sampling methods, see "Sampling Methods for Computing Uncertain Parameter Values" on page 3-40.

3 To add an uncertain parameter:

a Click **Add** to open the Add Parameters dialog box.

The dialog box lists all model parameters currently available in the MATLAB workspace.

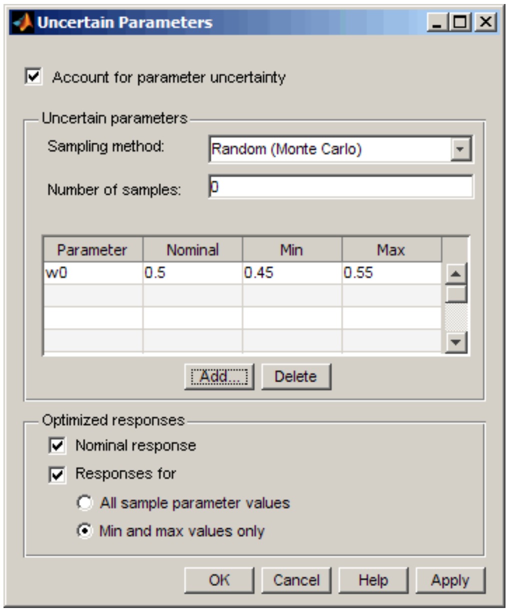


Note Parameters that are already specified for optimization or as uncertain parameters do not appear in the Add Parameters dialog box.

---

b Select the parameters in the Add Parameters dialog box, and then click **OK**.

This action adds the parameters to the Uncertain Parameters dialog box.


Uncertain Parameters

☑ Account for parameter uncertainty

Uncertain parameters

Sampling method: Random (Monte Carlo)

Number of samples: 0

Parameter	Nominal	Min	Max
w0	0.5	0.45	0.55

Optimized responses

☑ Nominal response

☑ Responses for

( ) All sample parameter values

(●) Min and max values only

For each parameter in the Uncertain Parameters dialog box, you can change the nominal, minimum and maximum values.

4 In the **Optimized responses** area of the GUI, configure the sample parameter values to use during optimization by selecting:

- **Nominal response** check box to include the nominal values of the uncertain parameters
- **All sample parameter values** check box to include all sample values of the uncertain parameters
- **Min and max values only** check box to include only the minimum and maximum values of the uncertain parameters

Tip Using only the minimum and maximum values during optimization increases the computation speed.

5 Click **OK** to add the uncertain parameters to the response optimization project.

When you optimize the parameters for robustness, the optimization algorithm uses the responses computed using all the uncertain parameter values to adjust the model parameters. For an example of testing and optimizing parameters for model robustness using the GUI, see "Example — Optimizing Parameters for Model Robustness Using the GUI" on page 3-47.

Commands for Optimizing Parameters for Model Robustness

You can also optimize parameters for model robustness by including parameter uncertainty at the command line. The following table summarizes the commands for model robustness. For detailed information about using each command, see the corresponding reference page.

Command	Purpose
`setunc`	Specify parameter uncertainty in response optimization project
`gridunc`	Sampling method for computing a grid of uncertain parameter values
`randunc`	Sampling method for computing random samples of uncertain parameter values

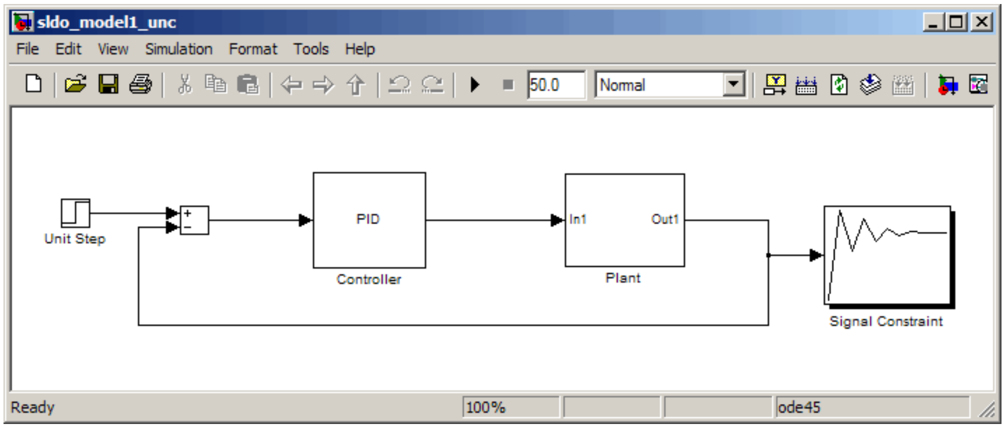
Example — Optimizing Parameters for Model Robustness Using the GUI

The following example shows how to optimize parameters for model robustness.

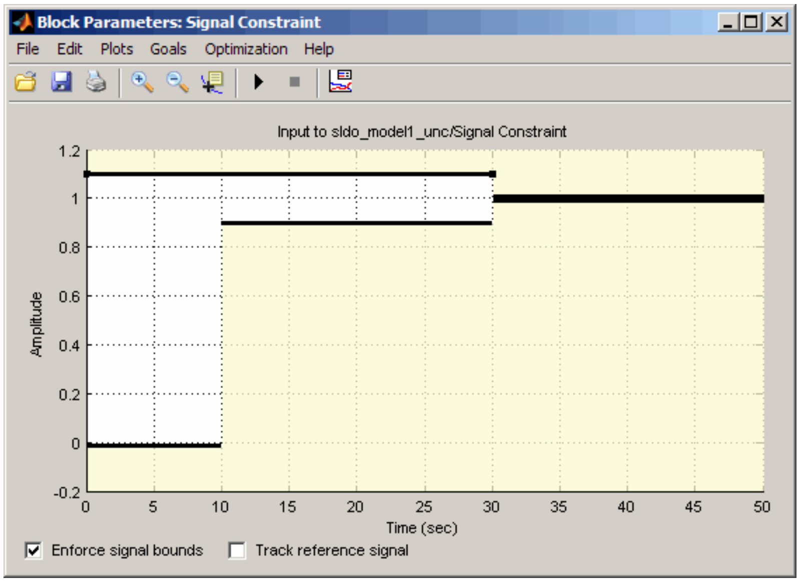
1 Open the Simulink model by typing the model name at the MATLAB prompt:

```
sldo_model1_desreq_optim
```

The following Simulink model opens.



The command also opens the Block Parameters: Signal Constraint window.


Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Input to sldo_model1_unc/Signal Constraint
Amplitude
Time (sec)
Enforce signal bounds
Track reference signal

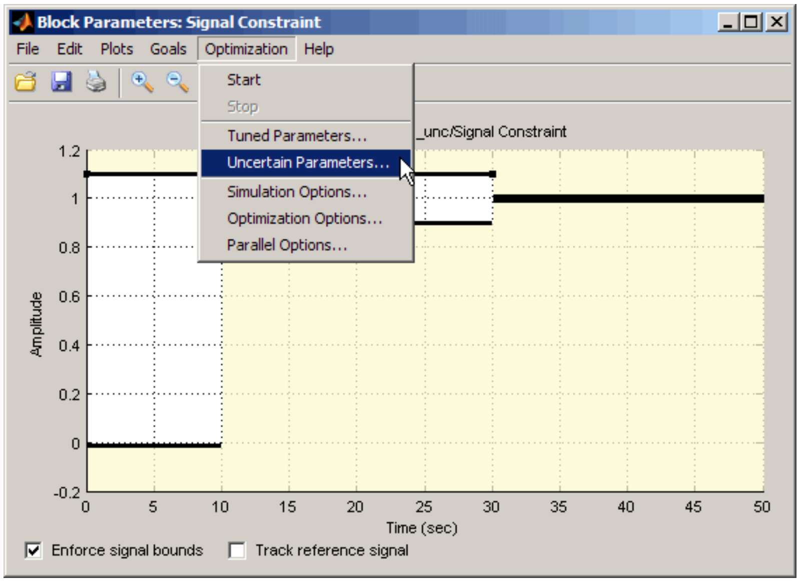
The Simulink model parameters have already been optimized to meet the following step response requirements:

- Maximum overshoot of 10%
- Maximum rise time of 10 seconds
- Maximum settling time of 30 seconds

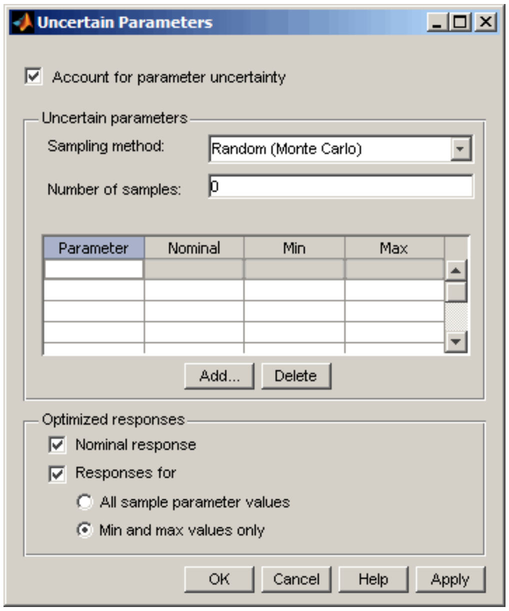
To learn how to optimize model parameters to meet design requirements, see “Tutorial — Optimizing Parameters to Meet Time-Domain Requirements Using the GUI” in the *Simulink Design Optimization*

2 To specify parameter uncertainty:

  a In the Block Parameters window, select **Optimization > Uncertain Parameters**.



This action opens the Uncertain Parameters dialog box.



[✓] Account for parameter uncertainty

Uncertain parameters

Sampling method: Random (Monte Carlo)

Number of samples: 0

Parameter	Nominal	Min	Max

Optimized responses

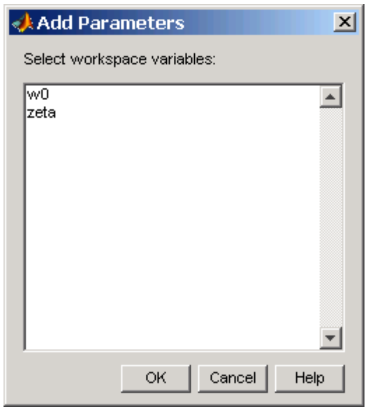
[✓] Nominal response

[✓] Responses for

All sample parameter values [✓ Min and max values only]

To learn more about the options in this dialog box, see "How to Optimize Parameters for Model Robustness Using the GUI" on page 3-43.

b Click **Add** to open the Add Parameters dialog box.

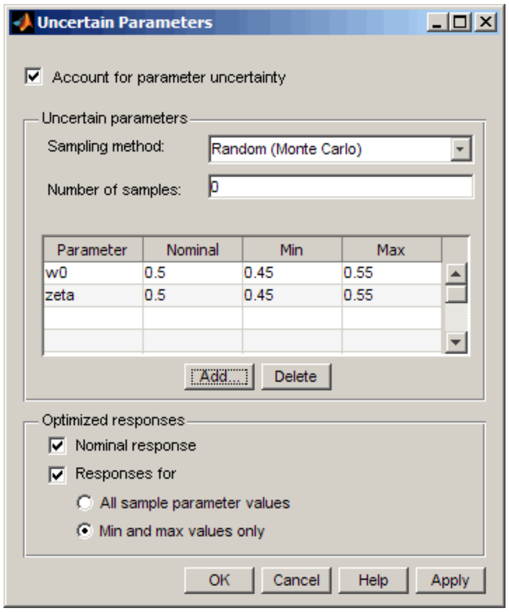


Select workspace variables:

w0
zeta

c Select `w0` and `zeta`, and click **OK**.

This action adds the parameters to the Uncertain Parameters dialog box.



[✓] Account for parameter uncertainty

Uncertain parameters

Sampling method: Random (Monte Carlo)

Number of samples: 0

Parameter	Nominal	Min	Max
w0	0.5	0.45	0.55
zeta	0.5	0.45	0.55

Optimized responses

[✓] Nominal response

[✓] Responses for

○ All sample parameter values

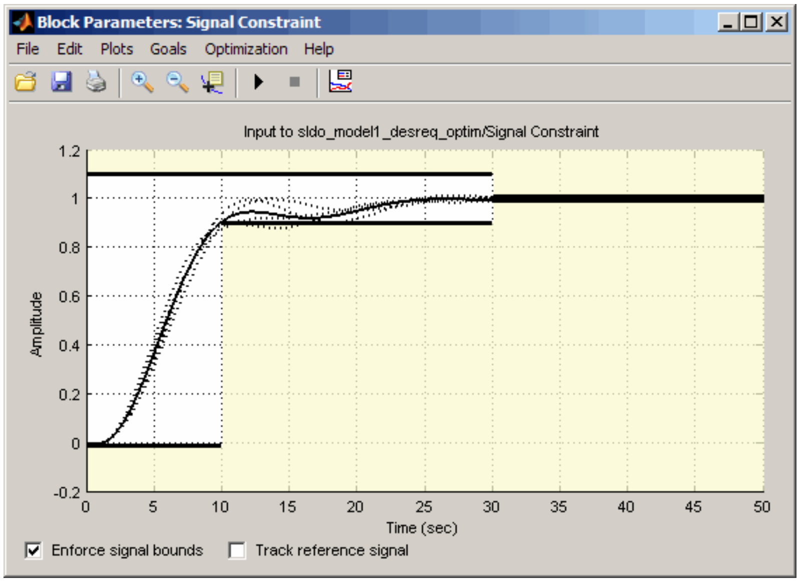
● Min and max values only

The **Nominal** column displays the nominal value of the parameters as specified in the Simulink model. The **Min** and **Max** columns specify the range in which the parameter can vary with respect to its nominal value. By default, the minimum and maximum parameter values vary by 10% of the nominal value.

d Click **OK** to close the Uncertain Parameters dialog box.

3 To test the model robustness to the uncertain parameters, select **Plots > Plot Current Response** in the Signal Constraint block window.

The Block Parameters window updates, as shown in the following figure.


Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Input to sldo_model1_desreq_optim/Signal Constraint
Amplitude
Time (sec)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0 5 10 15 20 25 30 35 40 45 50
☑ Enforce signal bounds
☐ Track reference signal

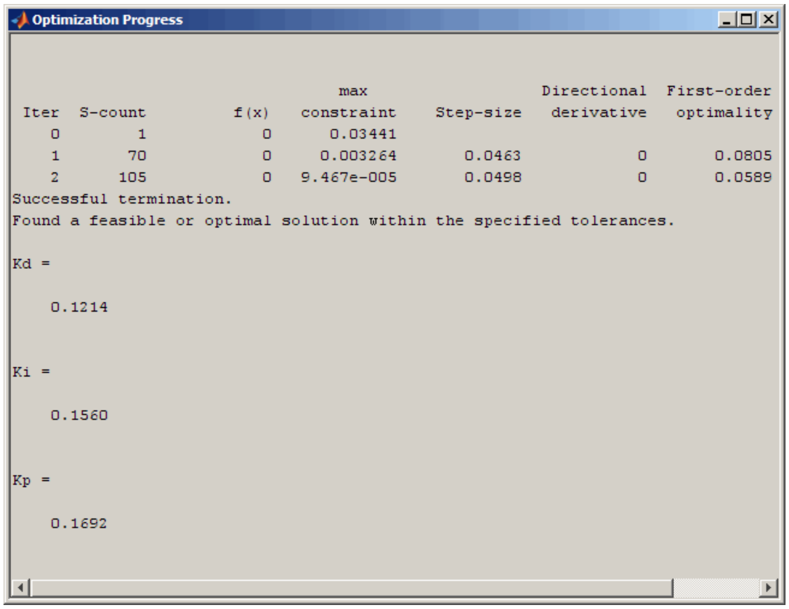
The window shows the following plot lines:

- The plot line shown as the solid black curve corresponds to the model's response computed using the optimized parameters and the nominal values of the uncertain parameter.
- The four plot lines shown as the dashed black curves correspond to the model's response with the minimum and maximum values of the uncertain parameters.

  The dashed plot lines show that the model's response during the period of 10 to 15 seconds violates the design requirements.

4 To optimize the parameters for model robustness, select **Optimization > Start**.

This action opens the Optimization Progress window, which displays the optimization iterations.

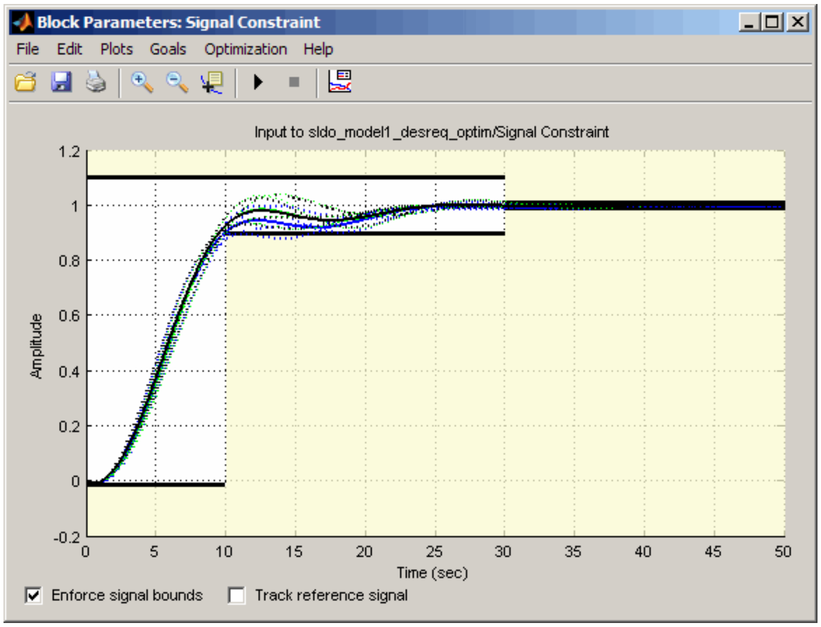


```
                                     max                    Directional   First-order
 Iter   S-count          f(x)     constraint    Step-size    derivative    optimality
    0         1             0        0.03441
    1        70             0       0.003264       0.0463             0        0.0805
    2       105             0      9.467e-005      0.0498             0        0.0589
Successful termination.
Found a feasible or optimal solution within the specified tolerances.

Kd =

    0.1214


Ki =

    0.1560


Kp =

    0.1692
```

After the optimization completes, the message `Successful termination` indicates that the model's response meets all the specified design requirements. The Optimization Progress window also displays the optimized parameter values.

5 Examine the final response in the updated Block Parameters window.

The final response of the model appears as the solid black curve. The model's response with the uncertain parameter values now meets the design requirements.


Block Parameters: Signal Constraint
File Edit Plots Goals Optimization Help
Input to sldo_model1_desreq_optim/Signal Constraint
Amplitude
Time (sec)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25
30
35
40
45
50
☑ Enforce signal bounds
☐ Track reference signal

Tip To view only the final response of the model, select **Plots > Clear Plots**. Then, select **Plots > Plot Current Response**.

Accelerating Model Simulations During Optimization

About Accelerating Optimization

You can accelerate the response optimization computations by changing the simulation mode of your Simulink model. Simulink Design 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 Design 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

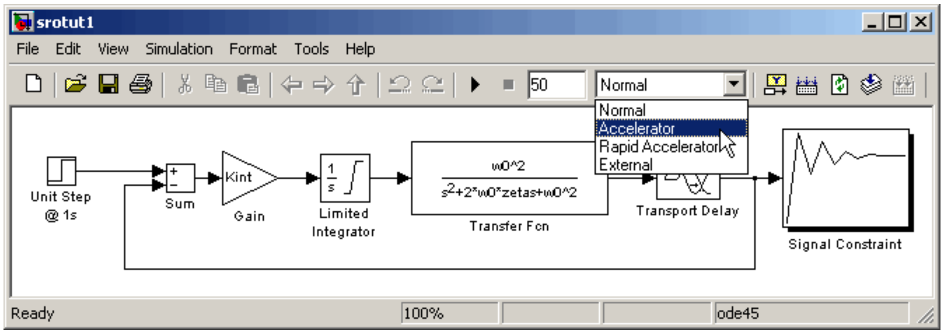
You cannot use the `Accelerator` mode if your model contains algebraic loops. If the model contains MATLAB function blocks, you must either remove them or replace them with Fcn blocks.

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.

Speeding Up Response Optimization Using Parallel Computing

In this section...

When to Use Parallel Computing for Response Optimization

You can use Simulink Design 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 Design 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 "How Parallel Computing Speeds Up Optimization" on page 3-59.

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

- "Configuring Your System for Parallel Computing" on page 3-62
- "How to Use Parallel Computing in the GUI" on page 3-64
- "How to Use Parallel Computing at the Command Line" on page 3-67

How Parallel Computing Speeds Up Optimization

You can enable parallel computing with the `Gradient descent` and `Pattern search` optimization algorithms in the Simulink Design Optimization software. When you enable parallel computing, Simulink Design 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" on page 3-59
- "Parallel Computing with the `Pattern search` Algorithm" on page 3-60

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 Design 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:

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

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 Design 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?" on page 3-75 in the "Troubleshooting Optimization Results" on page 3-69 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 3-62
- "Configuring Parallel Computing on Multiprocessor Networks" on page 3-62

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 Design 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 “Scope of 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 3-64, and “How to Use Parallel Computing at the Command Line” on page 3-67.

2. Add the file dependencies, as described in “Configuring Parallel Computing on Multiprocessor Networks” on page 3-62.

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.

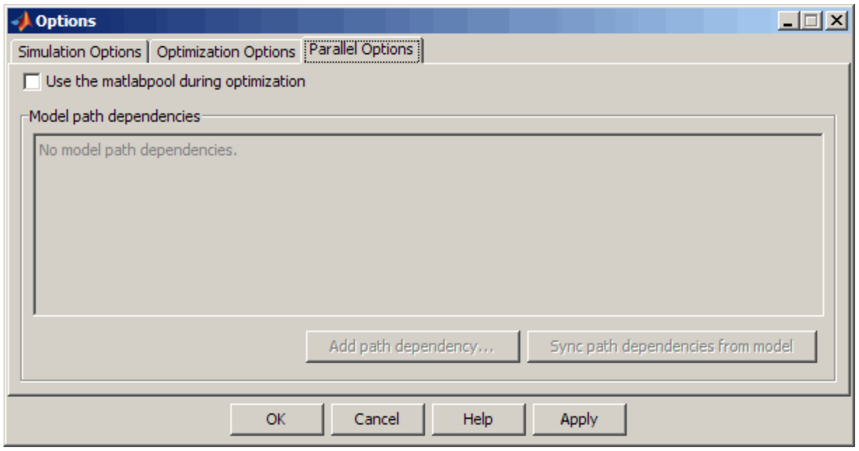
How to Use Parallel Computing in the GUI

After you configure your system for parallel computing, as described in "Configuring Your System for Parallel Computing" on page 3-62, 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.

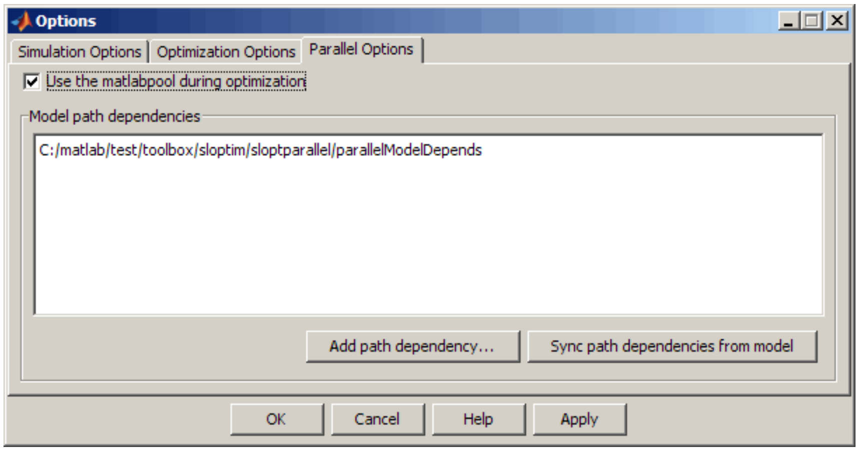
3. In the Signal Constraint block, select **Optimization > Parallel Options** to open the **Parallel Options** tab.



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 3-63, the automatic path dependencies check may not detect all the path dependencies in your model.



☑ Use the matlabpool during optimization

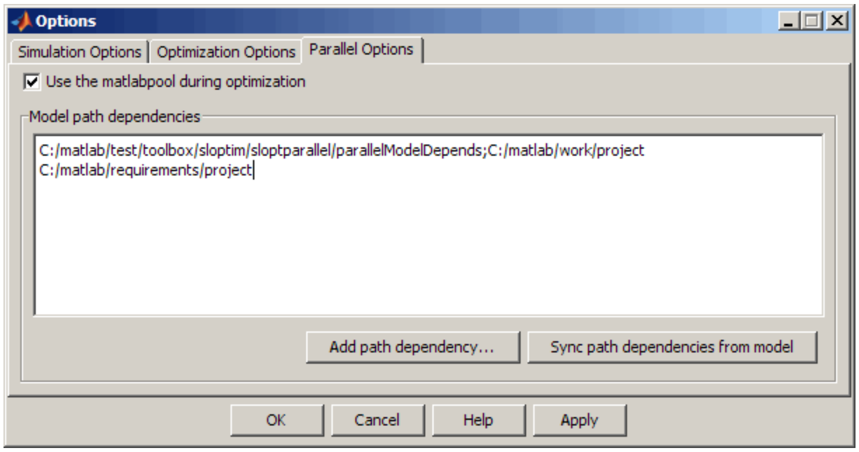
Model path dependencies

C:/matlab/test/toolbox/sloptim/sloptparallel/parallelModelDepends

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.

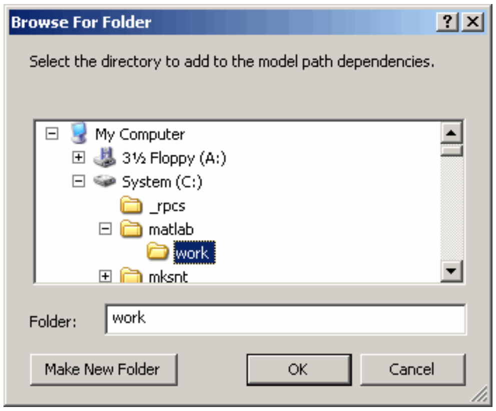


Model path dependencies

```
C:/matlab/test/toolbox/sloptim/sloptparallel/parallelModelDepends;C:/matlab/work/project
C:/matlab/requirements/project
```

   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.



Select the directory to add to the model path dependencies.

Folder: work

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 “Optimization Speed and Parallel Computing” on page 3-73.

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 3-62, 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`.

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 3-63, 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 “Optimization Speed and Parallel Computing” on page 3-73 in “Troubleshooting Optimization Results” on page 3-69.

Refining and Troubleshooting Optimization Results

Troubleshooting Optimization Results

When optimizing the model parameters, the optimization algorithm may run into issues. Simulink Design Optimization software provides visual cues to inform you about issues during the progress of an optimization. The following list represents the commonly encountered problems, and recommends solutions, advice, and tips to help you troubleshoot the issue.

- "Optimization Does Not Make Progress" on page 3-69
- "Optimization Convergence" on page 3-70
- "Optimization Speed and Parallel Computing" on page 3-73
- "Undesirable Parameter Values" on page 3-76
- "Reverting to Initial Parameter Values" on page 3-78

Optimization Does Not Make Progress

- "Should I worry about the scale of my responses and how constraints and design requirements are discretized?" on page 3-69
- "Why don't the responses and parameter values change at all?" on page 3-69
- "Why does the optimization stall?" on page 3-70

Should I worry about the scale of my responses and how constraints and design requirements are discretized?. No. Simulink Design Optimization software automatically normalizes constraints, design requirement and response data. Unlike its predecessor, the Nonlinear Control Design Blockset software, it does not discretize the constraints or design requirements.

Why don't the responses and parameter values change at all?.

- The optimization problem you formulated might be nonsmooth. This means that small parameter changes have no effect on the amount by which response signals satisfy or violate the constraints and only large changes will make a difference. Try switching to a search-based algorithm such as simplex search or pattern search. Alternatively, look for initial guesses outside of the dead zone where parameter changes have no effect. If you are directly optimizing the response of a Simulink model using a Signal Constraint block, you could also try removing nonlinear blocks such as the Quantizer or Dead Zone block.

- If you are using the `Refined` option for **Gradient type** with the gradient descent algorithm, try the `Basic` option for **Gradient type** instead. The gradient model that the `Refined` option uses might be invalid for your problem.

Why does the optimization stall?. When using a Signal Constraint block to directly optimize a Simulink model, certain parameter combinations can make the simulation stall for models with strong nonlinearities or frequent mode switching. In these cases, the ODE solvers take smaller and smaller step sizes. Stalling can also occur when the model's ODEs become too stiff for some parameter combinations. A symptom of this behavior is when the Simulink model status is `Running` and clicking the **Stop** button fails to interrupt the optimization. When this happens, you can try one of the following solutions:

- Switch to a different ODE solver, especially one of the stiff solvers.
- Specify a minimum step size.
- Disable zero crossing detection if chattering is occurring.
- Tighten the lower and upper bounds on parameters that cause simulation difficulties. In particular, eliminate regions of the parameter space where some model assumptions are invalid and the model behavior can become erratic.

Optimization Convergence

- "What to do if the optimization does not get close to an acceptable solution?" on page 3-71
- "Why does the optimization terminate before exceeding the maximum number of iterations, with a solution that does not satisfy all the constraints or design requirements?" on page 3-71
- "What to do if the optimization takes a long time to converge even though it is close to a solution?" on page 3-72

- "What to do if the response becomes unstable and does not recover?" on page 3-72

What to do if the optimization does not get close to an acceptable solution?.

- If you're using gradient descent, the default algorithm, try the `Refined` option for **Gradient type**. This option yields more accurate gradient estimates when using variable-step solvers and can facilitate convergence.
- If you are using pattern search, check that you have specified appropriate maximum and minimum values for all your tuned parameters or compensator elements. The pattern search algorithm looks inside these bounds for a solution. When they are set to their default values of `Inf` and `-Inf`, the algorithm searches within ±100% of the initial values of the parameters. In some cases this region is not large enough and changing the maximum and minimum values can expand the search region.
- Your optimization problem might have local minima. Consider running one of the search-based algorithms first to get closer to an acceptable solution.
- Reduce the number of tuned parameters and compensator elements by removing from the **Tuned parameters** list (when using a Signal Constraint block) or from the **Compensators** pane (when using a SISO Design Task) those parameters that you know only mildly influence the optimized responses. After you identify reasonable values for the key parameters, add the fixed parameters back to the tunable list and restart the optimization using these reasonable values as initial guesses.

Why does the optimization terminate before exceeding the maximum number of iterations, with a solution that does not satisfy all the constraints or design requirements?.

- It might not be possible to achieve your specifications. Try relaxing the constraints or design requirements that the response signals violate the most. After you find an acceptable solution to the relaxed problem, tighten some constraints again and restart the optimization.
- The optimization might have converged to a local minimum that is not a feasible solution. Restart the optimization from a different initial guess and/or use one of the search-based methods to identify another local minimum that satisfies the constraints.

What to do if the optimization takes a long time to converge even though it is close to a solution?.

- In a Signal Constraint window, use the Stop button, or select **Optimization > Stop**, to interrupt the optimization when you think the current optimized response signals are acceptable.

  When you use a SISO Design Task, click **Stop Optimization** in the **Optimization** panel of the **Response Optimization** node in the Control and Estimation Tools Manager, when you think the current optimized response signals are acceptable.
- If you use the gradient descent algorithm, try restarting the optimization. Restarting resets the Hessian estimate and might speed up convergence.
- Increase the convergence tolerances in the Optimization Options dialog to force earlier termination.
- Relax some of the constraints or design requirements to increase the size of the feasibility region.

What to do if the response becomes unstable and does not recover?. While the optimization formulation has explicit safeguards against unstable or divergent response signals, the optimization can sometimes venture into an unstable region where simulation results become erratic and gradient methods fail to find a way back to the stable region. In these cases, you can try one of the following solutions:

- Add or tighten the lower and upper bounds on compensator element and parameter values. Instability often occurs when you allow some parameter values to become too large.
- Use a search-based algorithm to find parameter values that stabilize the response signals and then start the gradient-based algorithm using these initial values.
- When optimizing responses in a SISO Design Task, you can try adding additional design requirements that achieve the same or similar goal. For example, in addition to a settling time design requirement on a step response plot, you could add a settling time design requirement on a root-locus plot that restricts the location of the real parts of the poles. By adding overlapping design requirements in this way, you can force the optimization to meet the requirements.

Optimization Speed and Parallel Computing

How can I speed up the optimization?.

- The optimization time is dominated by the time it takes to simulate the model. When using a Signal Constraint block to directly optimize a Simulink model, you can enable the Accelerator mode using **Simulation > Accelerator** in the Simulink model window, to dramatically reduce the optimization time.

  Note The Rapid Accelerator mode in Simulink software is not supported for speeding up the optimization. For more information, see the Accelerating the Optimization.

- The choice of ODE solver can also significantly affect the overall optimization time. Use a stiff solver when the simulation takes many small steps, and use a fixed-step solver when such solvers yield accurate enough simulations for your model. (These solvers must be accurate in the entire parameter search space.)
- Reduce the number of tuned compensator elements or parameters and constrain their range to narrow the search space.

- When specifying parameter uncertainty (not available when optimizing responses in a SISO Design Task), keep the number of sample values small since the number of simulations grows exponentially with the number of samples. For example, a grid of 3 parameters with 10 sample values for each parameter requires 10^3=1000 simulations per iteration.

Why are the optimization results with and without using parallel computing different?.

- When you use parallel computing, different numerical precision on the client and worker machines can produce marginally different simulation results. Thus, the optimization algorithm takes a completely different solution path and produces a different result.

  Note Numerical precision can differ because of different operating systems or hardware on the client and worker machines.

- When you use parallel computing, the state of the model on the client and the worker machines can differ, and thus lead to a different result. For example, the state can become different if you change a parameter value initialized by a callback function on the client machine *after* the workers have loaded the model. The model parameter values on the workers and the client are now out of sync, which can lead to a different result.

  After you change the model parameter values initialized by a callback function, verify that the parameters exist in the model workspace or update the callback function so that the remote workers have access to the changed parameter values.

- When you use parallel computing with the `Pattern search` algorithm, the `Pattern search` algorithm searches for a candidate solution more comprehensively than when you do not use parallel computing. This more comprehensive search can result in a different solution. To learn more, see “Parallel Computing with the `Pattern search` Algorithm”.

Why do I not see the optimization speedup I expected using parallel computing?.

- When you optimize a model that does not have a large number of parameters or does not take long to simulate, the resulting optimization time may not be any faster. In such cases, the overheads associated with creating and distributing the parallel tasks outweighs the benefits of running the simulations during optimization in parallel.
- Using `Pattern search` algorithm with parallel computing may not speed up the optimization time. When you do not use parallel computing, the algorithm stops searching for a candidate solution at each iteration as soon as it finds a solution better than the current solution. The candidate solution search is more comprehensive when you use parallel computing. Although the number of iterations may be larger, the optimization without using parallel computing may be faster.

  To learn more about the expected speedup, see "Parallel Computing with the `Pattern search` Algorithm".

Why does the optimization using parallel computing not make any progress?. In some cases, the gradient computations on the remote worker machines may silently error out when you use parallel computing. In such cases, the Optimization Progress window shows that the `f(x)` and `max constraint` values do not change, and the optimization terminates after two iterations with the message `Unable to satisfy constraints`. To troubleshoot the problem:

1. Run the optimization for a few iterations without parallel computing to see if the optimization progresses.

2. Check if the remote workers have access to all model dependencies. To learn more, see "Checking Model Dependencies".

Why do I receive an error "Cannot save model tpe5468c55_910c_4275_94ef_305e2eeeeef4"?. When you select `Refined` as the **Gradient type**, the software may error out when it saves a temporary model to a nonwriteable directory, and then displays this error message. Change the **Gradient type** to `Basic` to clear this error. To learn more, see "Gradient Type".

Why does the optimization using parallel computing not stop when I click the Stop optimization button?. When you use parallel computing, the software has to wait till the current iteration completes before it notifies the workers to stop the optimization. The optimization does not terminate immediately when you click the **Stop optimization** button ■, and appears to continue to run.

Undesirable Parameter Values

- "What to do if the optimization drives the tuned compensator elements and parameters to undesirable values?" on page 3-76
- "What to do if the optimization violates bounds on parameter values?" on page 3-76

What to do if the optimization drives the tuned compensator elements and parameters to undesirable values?.

- When a tuned compensator element or parameter is positive, or when its value is physically constrained to a given range, enter the lower and upper bounds (**Minimum** and **Maximum**) in one of the following:
  - Tuned Parameters dialog box (from a Signal Constraint block)
  - **Compensators** pane (in a SISO Design Task)

  This information helps guide the optimization algorithm towards a reasonable solution.
- In the Tuned Parameters dialog box (from a Signal Constraint block) or the **Compensators** pane (in a SISO Design Task), specify initial guesses that are within the range of desirable values.
- In the **Compensators** pane in a SISO Design Task, verify that no integrators/differentiators are selected for optimization. Optimizing the pole/zero location of integrators/differentiators can result in drastic changes in the system gain and lead to undesirable values.

What to do if the optimization violates bounds on parameter values?. The `Gradient descent` optimization algorithm fmincon violates the parameter bounds when it cannot simultaneously satisfy the signal constraints and the bounds. When this happens, try one of the following:

- Specify a different value for the parameter bound and restart the optimization. A guideline is to adjust the bound by 1% of the typical value.

  For example, for a parameter with a typical value of `1` and lower bound of `0`, change the lower bound to `0.01`.

- Relax the signal constraints and restart the optimization. This approach results in a different solution path for the `Gradient descent` algorithm.
- Restart the optimization immediately after it terminates by using the **Start optimization** button in the Signal Constraint window. This approach uses the previous optimization results as the starting point for the next optimization cycle to refine the results.
- Use the following two-step approach to perform the optimization:

  1. Run an initial optimization to satisfy the signal constraints.

     For example, run the optimization using the `Simplex search` algorithm. This algorithm satisfies the signal constraints but does not support the bounds on parameter values. The results obtained using this algorithm provide the starting point for the optimization performed in the next step. To learn more about this algorithm, see the fminsearch function reference page in the Optimization Toolbox documentation.

  2. Reconfigure the optimization by selecting a different optimization algorithm to satisfy both the signal constraints and the parameter bounds.

     For example, change the optimization algorithm to `Gradient descent` and run the optimization again.

Tip If Genetic Algorithm and Direct Search Toolbox software is installed, you can select the `Pattern search` optimization algorithm to optimize the model response.

Reverting to Initial Parameter Values

How do I quit an optimization and revert to my initial parameter values?.

- When using a Signal Constraint block in a Simulink model, click the **Stop** button or select **Optimization > Stop** in the Signal Constraint window to stop the optimization. To revert to your initial parameter values, select **Edit > Undo Optimize Parameters**.
- When using a SISO Design Task, the **Start Optimization** button becomes a **Stop Optimization** button after the optimization has begun. To quit the optimization, click the **Stop Optimization** button. To revert to the initial parameter values, select **Edit > Undo Optimize compensators** from the menu in the SISO Design Tool window.

Saving and Loading 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

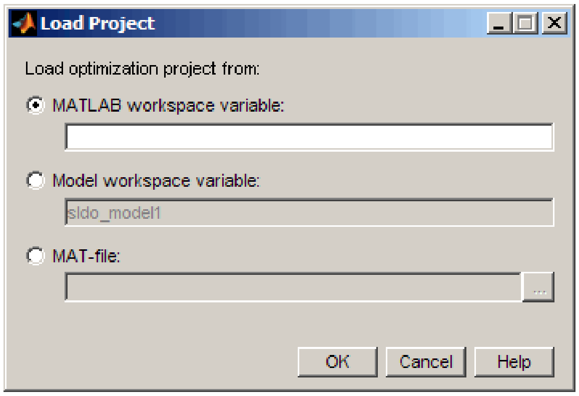
In addition to settings that you save with the response optimization project, there are several other settings that you save with the *model*. These settings include the following:

- 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** in 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.


Load Project
Load optimization project from:
MATLAB workspace variable:
Model workspace variable:
sldo_model1
MAT-file:
...
OK
Cancel
Help

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.

4

Optimization-Based Linear Compensator Design

Overview of Optimization-Based Compensator Design

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

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 optimization methods in 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 optimization techniques in the SISO Design Task to tune the response of the linearized Simulink model.

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

When using a SISO Design Task, Simulink Design Optimization software automatically sets the model's simulation start and stop time and you cannot directly change them. By default, the simulation starts at 0 and continues until the SISO Design Task determines that the dynamics of the model have settled out. In addition, when the design requirements extend beyond this point, the simulation continues to the extent of the design requirements. Although you cannot directly adjust the start or stop time of the simulation, you can adjust the design requirements to extend further in time and thus force the simulation to continue to a certain point.

Supported Time- and Frequency-Domain Requirements

In this section...

This section lists the time- and frequency-domain requirements that you can specify for optimization-based control design in Simulink Design 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 ζ is the damping ratio.

Damping Ratio

Specifying a damping ratio in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the damping ratio. In the discrete-time case, the 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

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

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 *abs(pole)* for continuous systems and on

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

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.

Designing Optimization-Based Controllers for LTI Systems

In this section...

How to Design Optimization-Based Controllers for LTI Systems

To design optimization-based linear controller for an LTI model:

1 Create and import a linear model into a SISO Design Task. You can create an LTI model at the MATLAB command line, as described in "Creating an LTI Plant Model" on page 4-10.

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" on page 4-14 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" on page 4-16 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" on page 4-17 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" on page 4-25 for more information.

Example — Frequency-Domain Optimization for LTI System

Introduction

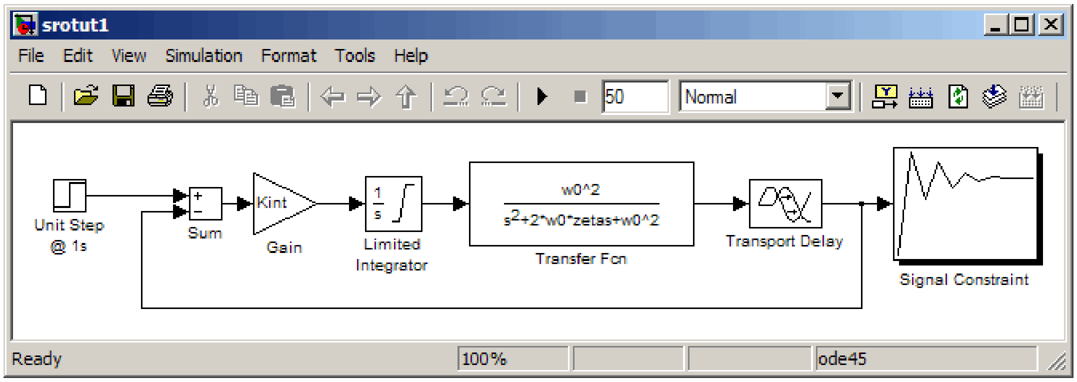
When you have Control System Toolbox software, you can place Simulink Design Optimization design requirements or constraints on plots in the SISO Design Tool graphical tuning editor and analysis plots that are part of a SISO Design Task. This allows you to include design requirements for response optimization in the frequency-domain in addition to the time-domain. This section guides you through an example using frequency-domain design requirements to optimize the response of a system in the SISO Design Task.

You can specify frequency-domain design requirements to optimize response signals for any model that you can design within a SISO Design Task:

- Command-line LTI models created with the Control System Toolbox commands
- Simulink models that have been linearized using Simulink Control Design software

Design Requirements

In this example, you use a linearized version of the following Simulink model.



You use optimization methods to design a compensator so that the closed loop system meets the following design specifications when you excite the system with a unit step input:

- A maximum 30-second settling time
- A maximum 10% overshoot
- A maximum 10-second rise time
- A limit of ±0.7 on the actuator signal

Creating an LTI Plant Model

In the `srotut1` model, the plant model is composed of a gain, a limited integrator, a transfer function, and a transport delay.

You want to design the compensator for the open loop transfer function of the linearized `srotut1` model. The linearized `srotut1` plant model is composed of the gain, an unlimited integrator, the transfer function, and a Padé approximation to the transport delay.

To create an open loop transfer function based on the linearized `srotut1` model, enter the following commands:

```
w0     = 1;
zeta   = 1;
Kint   = 0.5;
Tdelay = 1;
[delayNum,delayDen] = pade(Tdelay,1);
```

```
integrator   = tf(Kint,[1 0]);
transfer_fcn = tf(w0^2,[1 2*w0*zeta w0^2]);
delay_block  = tf(delayNum,delayDen);
open_loopTF  = integrator*transfer_fcn*delay_block;
```

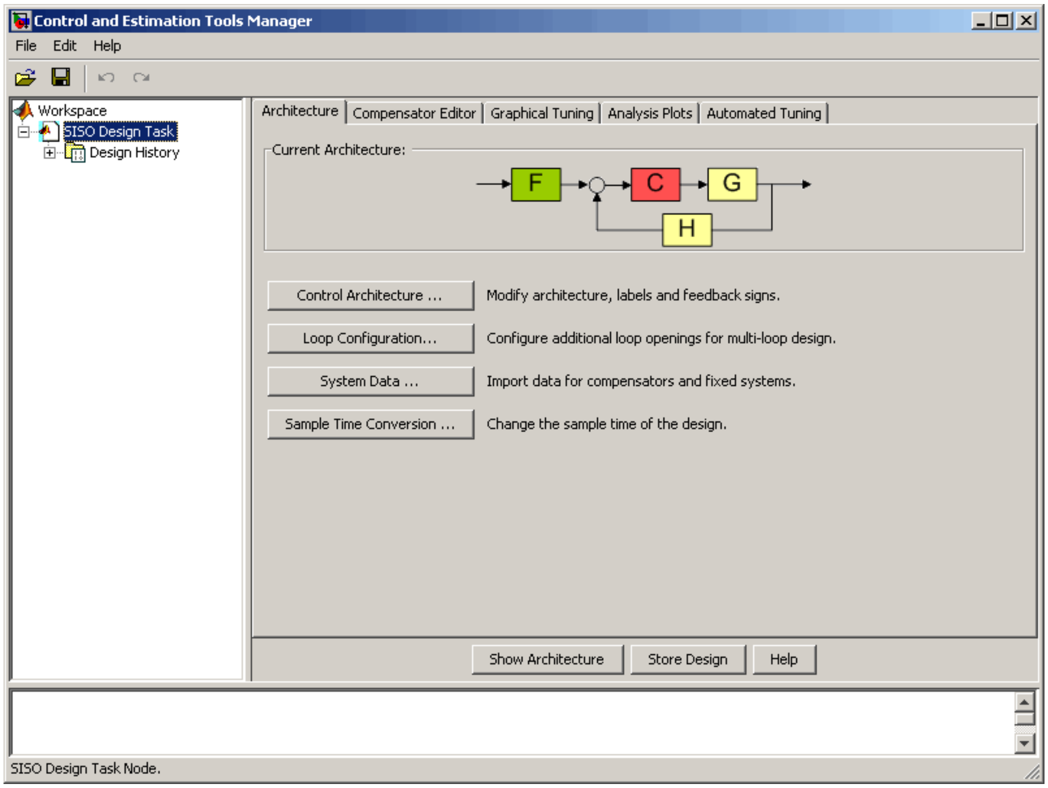
Note It is also possible to directly linearize the Simulink model `srotut1` using Simulink Control Design software.

Creating Design and Analysis Plots

This example uses a root locus diagram to design the response of the open loop transfer function, `open_loopTF`. To create a SISO Design Task, containing a root-locus plot for the open loop transfer function, use the following command:

```
sisotool('rlocus',open_loopTF)
```

A **SISO Design Task** is created within the Control and Estimation Tools Manager, as shown in the following figure.



Control Architecture ... Modify architecture, labels and feedback signs.

Loop Configuration... Configure additional loop openings for multi-loop design.

System Data ... Import data for compensators and fixed systems.

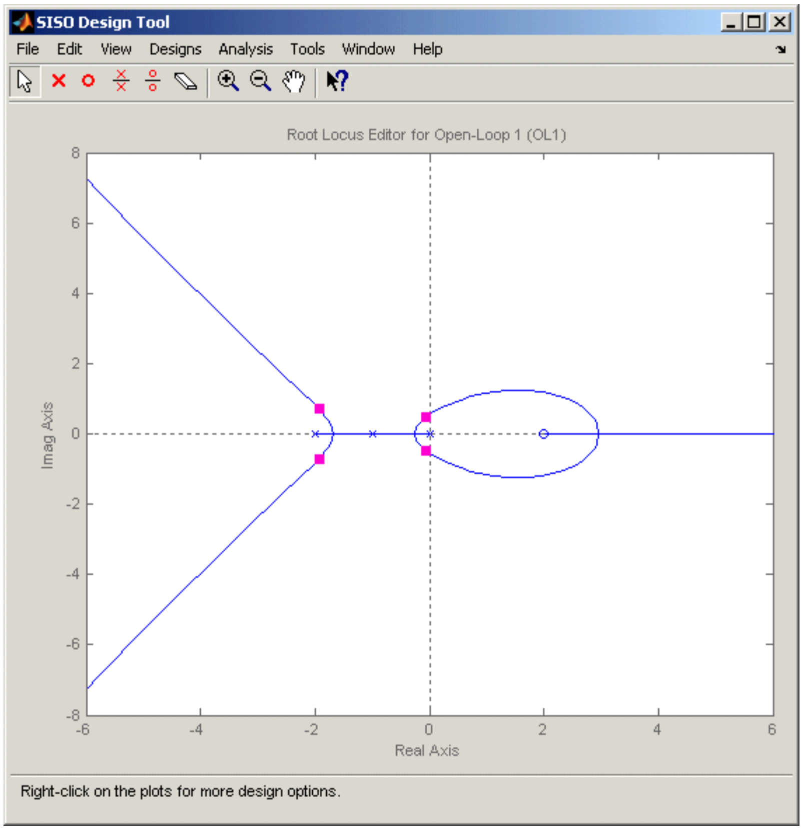
Sample Time Conversion ... Change the sample time of the design.

SISO Design Task Node.

The Control and Estimation Tools Manager is a graphical environment for managing and performing tasks such as designing SISO systems. The SISO Design Task node contains five panels that perform actions related to designing SISO control systems. For more information, see “Using the SISO Design Task in the Controls & Estimation Tools Manager” in “Control System Toolbox” documentation.

The **Architecture** pane, within the **SISO Design Task** node, lets you choose the architecture for the control system you are designing. This example uses the default architecture. In this system, the plant model, G, is the open loop transfer function `open_loopTF`, the prefilter, F, and the sensor, H, are set to `1`, and the compensator, C, is the compensator that will be designed using response optimization methods.

The SISO Design Task also contains a root locus diagram in the SISO Design Tool graphical tuning editor.


SISO Design Tool
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open-Loop 1 (OL1)
Imag Axis
Real Axis
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6

Right-click on the plots for more design options.

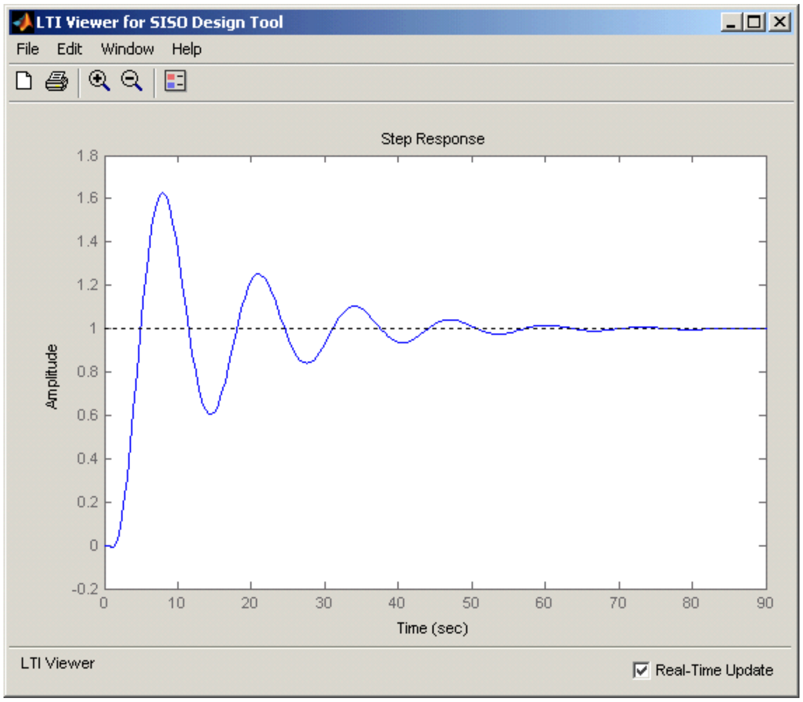
In addition to the root-locus diagram, it is helpful to visualize the response of the system with a step response plot. To add a step response:

1 Select the **Analysis Plots** pane with the **SISO Design Task** node of the Control and Estimation Tool Manager.

2 Select `Step` for the **Plot Type** of **Plot 1**.

3 Under **Contents of Plots**, select the check box in column **1** for the response **Closed Loop r to y**.

A step response plot appears in an LTI Viewer. The plot shows the response of the closed loop system from r (input to the prefilter, *F*) to y (output of the plant model, *G*):



Creating a Response Optimization Task

There are several possible methods for designing a SISO system; this example uses an automated approach involving response optimization methods. After creating the design and analysis plots as discussed in “Creating Design and Analysis Plots” on page 4-11, you are ready to start a response optimization task to design the compensator.

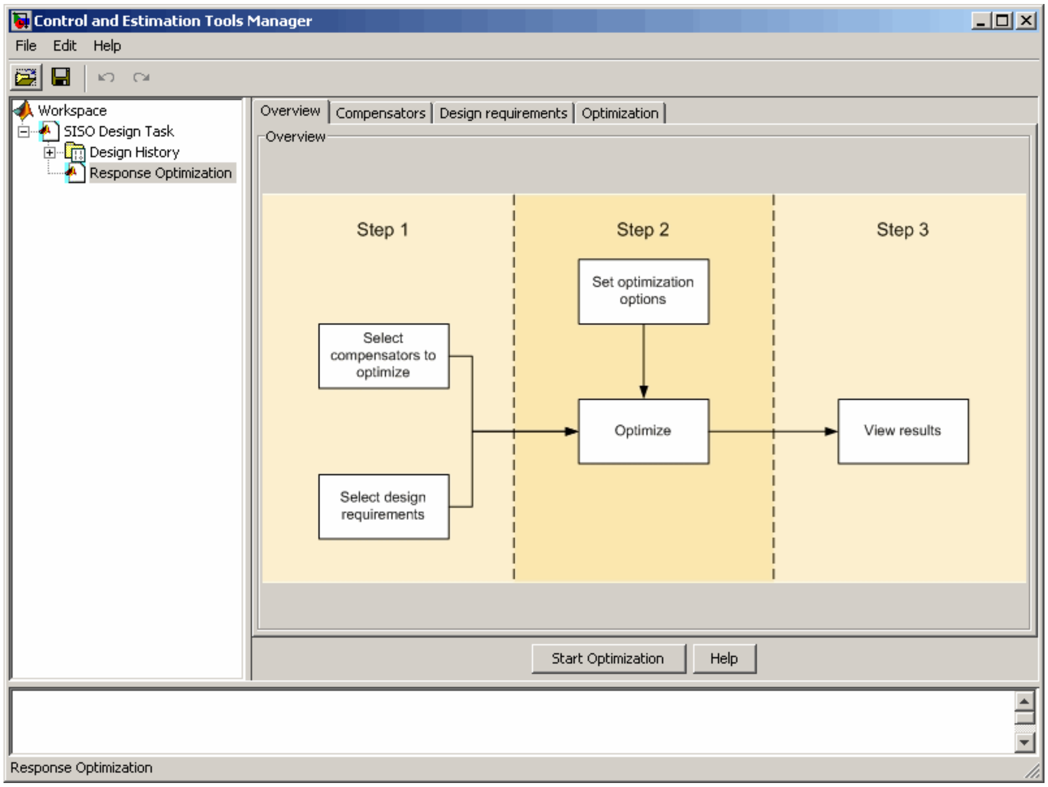
To create a response optimization task:

1 Select the **Automated Tuning** pane within the **SISO Design Task** node in the Control and Estimation Tools Manager.

2 In the **Automated Tuning** pane, select `Optimization based tuning` as the **Design Method**.

3 Click the **Optimize Compensators** button to create the **Response Optimization** node under the **SISO Design Task** node in the tree browser in the left pane of the Control and Estimation Tools Manager.

The **Response Optimization** node contains four panes as shown in the next figure.



With the exception of the first pane, each corresponds to a step in the response optimization process:

- **Overview**: A schematic diagram of the response optimization process.

- **Compensators**: Select and configure the compensator elements that you want to tune. See “Selecting Tunable Compensator Elements” on page 4-16.
- **Design requirements**: Select the design requirements that you want the system to meet after tuning the compensator elements. See “Adding Design Requirements” on page 4-17.
- **Optimization**: Configure optimization options and view the progress of the response optimization. See “Optimizing the System’s Response” on page 4-25.

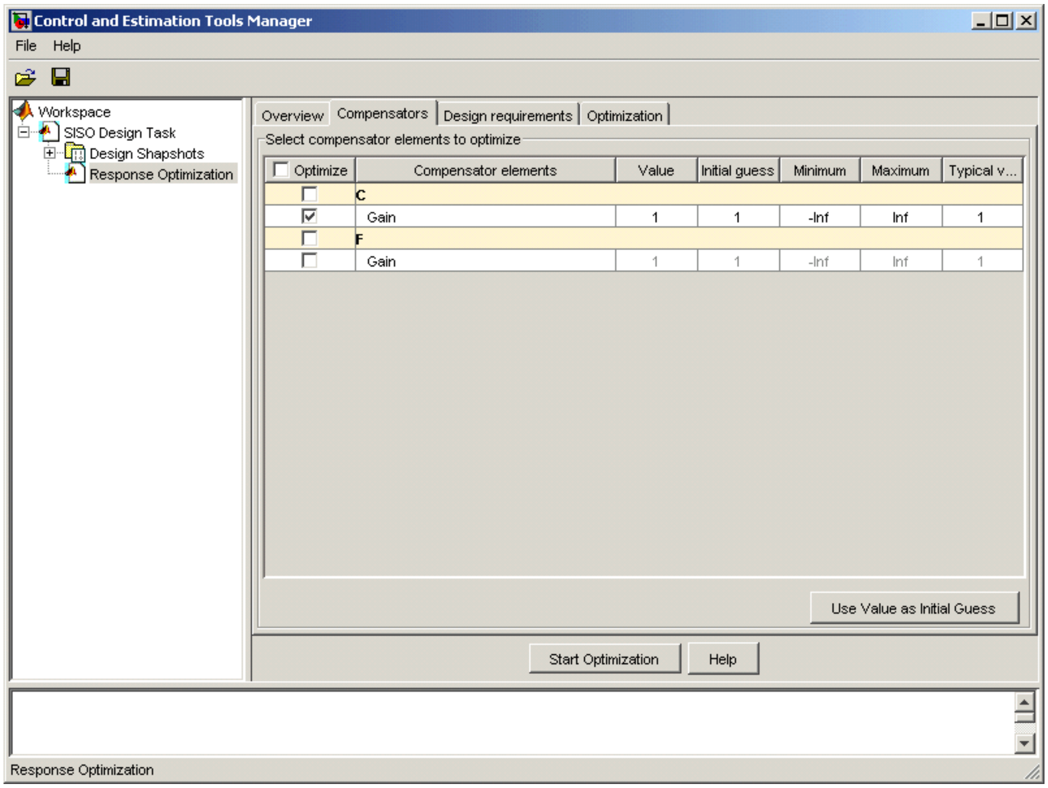
Note When optimizing responses in a SISO Design Task, you cannot add uncertainty to parameters or compensator elements.

Selecting Tunable Compensator Elements

You can tune elements or parameters within compensators in your system so that the response of the system meets the design requirements you specify. To specify the compensator elements to tune:

1. Select the **Compensators** pane within the **Response Optimization** node.
2. Within the **Compensators** pane, select the check boxes in the **Optimize** column that correspond to the compensator elements you want to tune.

   In this example, to tune the **Gain** in the compensator **C**, select the check box next to this element, as shown in the following figure.



Select compensator elements to optimize

Optimize	Compensator elements	Value	Initial guess	Minimum	Maximum	Typical v...
☐	C					
☑	Gain	1	1	-Inf	Inf	1
☐	F					
☐	Gain	1	1	-Inf	Inf	1

Note Compensator elements or parameters cannot have uncertainty when used with frequency-domain based response optimization.

Adding Design Requirements

You can use both frequency-domain and time-domain design requirements to tune parameters in a control system. The **Design requirements** pane within the **Response Optimization** node of the Control and Estimation Tools Manager provides an interface to create new design requirements and select those you want to use for a response optimization.

This example uses the design specifications described in “Design Requirements” on page 4-9. The following sections each create a new design requirement to meet these specifications:

- “Settling Time Design Requirement” on page 4-18
- “Overshoot Design Requirement” on page 4-19
- “Rise Time Design Requirement” on page 4-21
- “Actuator Limit Design Requirement” on page 4-22

After you add the design requirements, you can select a subset of requirements for controller design, as described in “Selecting the Design Requirements to Use During Response Optimization” on page 4-25.

Settling Time Design Requirement. The first design specification for this example is to have a settling time of 30 seconds or less. This specification can be represented on a root-locus diagram as a constraint on the real parts of the poles of the open loop system.

To add this design requirement:

1. Select the **Design requirements** pane within the **Response Optimization** node of the Control and Estimation Tools Manager.

2. Click the **Add new design requirement** button. This opens the New Design Requirement dialog box.

   Within this dialog box you can specify new design requirements and add them to a new or existing design or analysis plot.

3. Add a design requirement to the existing root-locus diagram:

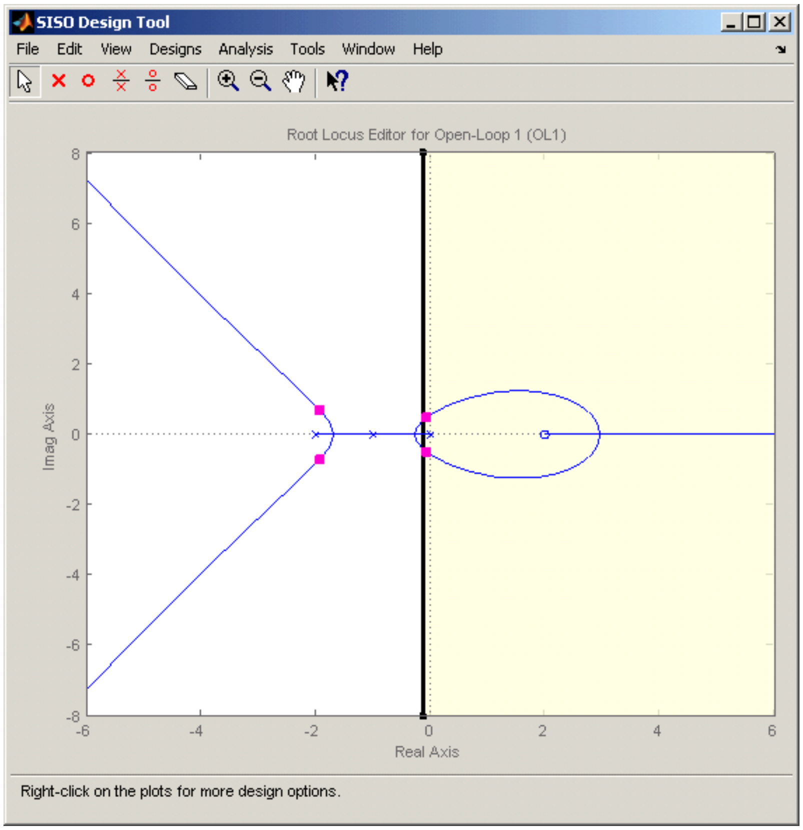
   a. Select `Pole/zero settling time` from the **Design requirement type** menu.

   b. Select `Open-Loop L` from the **Requirement for response** menu.

   c. Enter `30` seconds for the **Settling time**.

   d. Click **OK**.

A vertical line should appear on the root-locus diagram, as shown in the following figure.


SISO Design Tool
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open-Loop 1 (OL1)
Imag Axis
Real Axis
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6

Right-click on the plots for more design options.

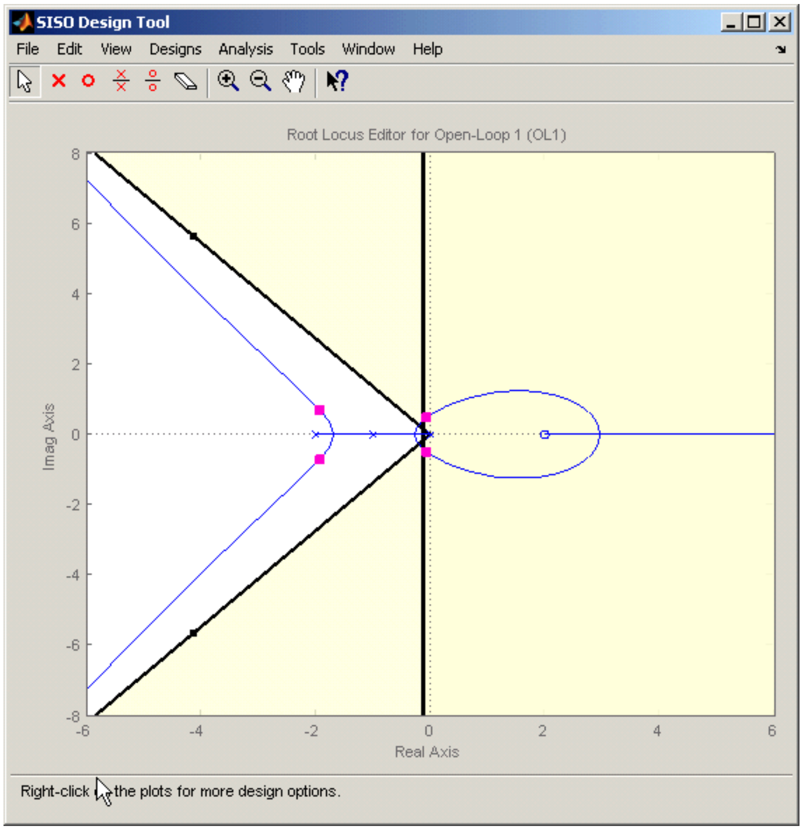
Overshoot Design Requirement. The second design specification for this example is to have a percentage overshoot of 10% or less. This specification is related to the damping ratio on a root-locus diagram. In addition to adding a design requirement with the **Add new design requirement** button, you can also right-click directly on the design or analysis plots to add the requirement, as shown next.

To add this design requirement:

1 Right-click anywhere within the white space of the root-locus diagram in the SISO Design Tool window. Select **Design Requirements > New** to open the New Design Requirement dialog box.

2 Select `Percent overshoot` as the **Design requirement type** and enter `10` as the **Percent overshoot**.

3 Click **OK** to add the design requirement to the root-locus diagram. The design requirement appears as two lines radiating at an angle from the origin, as shown in the following figure.



Right-click on the plots for more design options.

Rise Time Design Requirement. The third design specification for this example is to have a rise time of 10 seconds or less. This specification is related to a lower limit on a Bode Magnitude diagram.

To add this design requirement:

1. Select the **Graphical Tuning** pane in the **SISO Design Task** node of the Control and Estimation Tools Manager.

2. For Plot 2, set **Plot Type** to `Open-Loop Bode`.

3. Right-click anywhere within the white space of the open-loop bode diagram in the SISO Design Tool window. Select **Design Requirements > New** to open the New Design Requirement dialog box.

4. Create a design requirement to represent the rise time and add it to the new Bode plot:

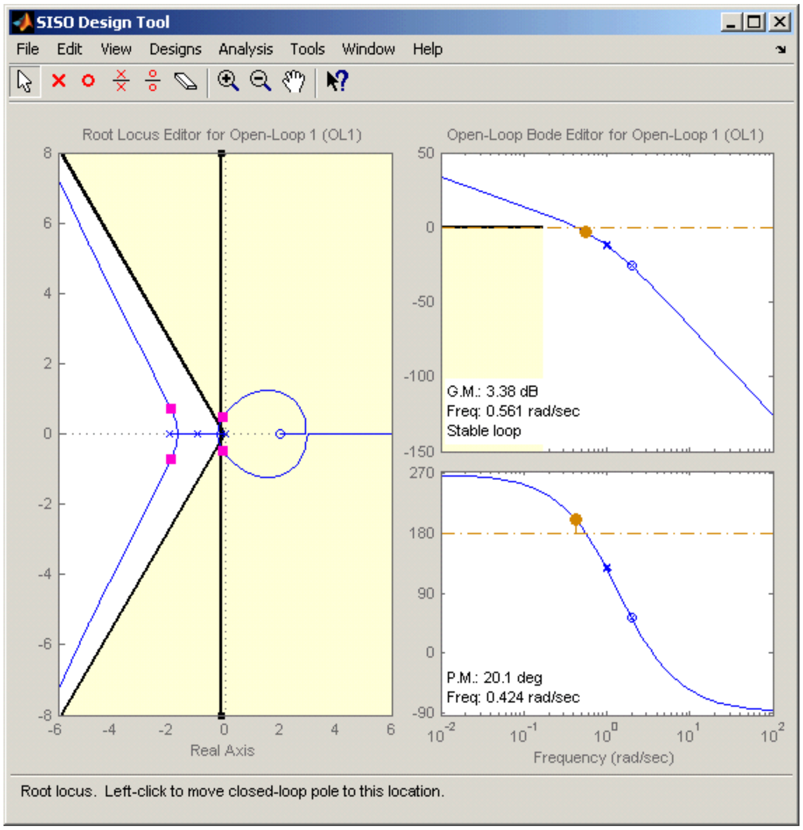
   a. Select `Lower gain limit` from the **Design requirement type** menu.

   b. Enter `1e-2` to `0.17` for the **Frequency** range.

   c. Enter `0` to `0` for the **Magnitude** range.

   d. Click **OK**.

   A Bode diagram appears within the SISO Design Tool window. The magnitude plot of the Bode diagram includes a horizontal line representing the design requirement, as shown in the following figure.


SISO Design Tool
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open-Loop 1 (OL1)
Open-Loop Bode Editor for Open-Loop 1 (OL1)
Real Axis
Frequency (rad/sec)
G.M.: 3.38 dB
Freq: 0.561 rad/sec
Stable loop
P.M.: 20.1 deg
Freq: 0.424 rad/sec
Root locus. Left-click to move closed-loop pole to this location.

Actuator Limit Design Requirement. The fourth design specification for this example is to limit the actuator signal to within ±0.7. To add this design requirement:

1 Select the **Design requirements** pane in the **Response Optimization** node of the Control and Estimation Tools Manager.

2 Click the **Add new design requirement** button to open the New Design Requirement dialog box.

3 Create a time-domain design requirement to represent the upper limit on the actuator signal, and add it to a new step response plot in the LTI Viewer:

a Select `Step response upper amplitude limit` from the **Design requirement type** menu.

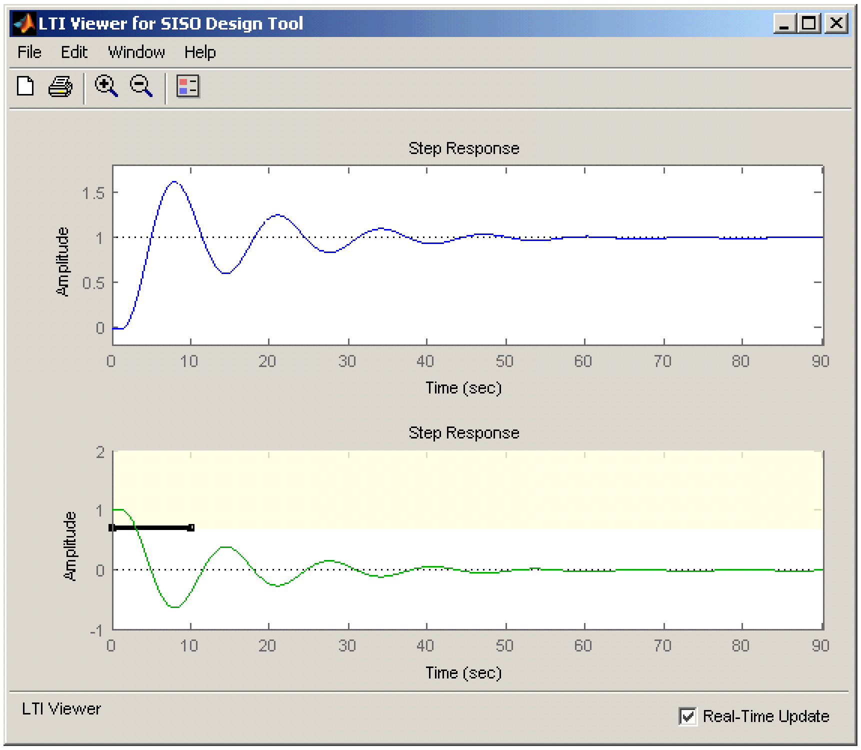
b Select `Closed Loop r to u` from the **Requirement for response** menu.

c Enter `0` to `10` for the **Time** range.

d Enter `0.7` to `0.7` for the **Amplitude** range.

e Click **OK**. A second step response plot for the closed loop response from `r` to `u` appears in the LTI Viewer. The plot contains a horizontal line representing the upper limit on the actuator signal.

f To extend this limit for all times (to t=∞), right click on the black edge of the design requirement, somewhere toward the right edge, and select **Extend to inf**. The diagram should now appear as shown next.



To add the corresponding design requirement for the lower limit on the actuator signal:

1. Select the **Design requirements** pane in the **Response Optimization** node of the Control and Estimation Tools Manager.

2. Click the **Add new design requirement** button to open the New Design Requirement dialog box.

3. Create a time-domain design requirement to represent the lower limit on the actuator signal, and add it to the step response plot in the LTI Viewer:

   a. Select `Step response lower amplitude limit` from the **Design requirement type** menu.

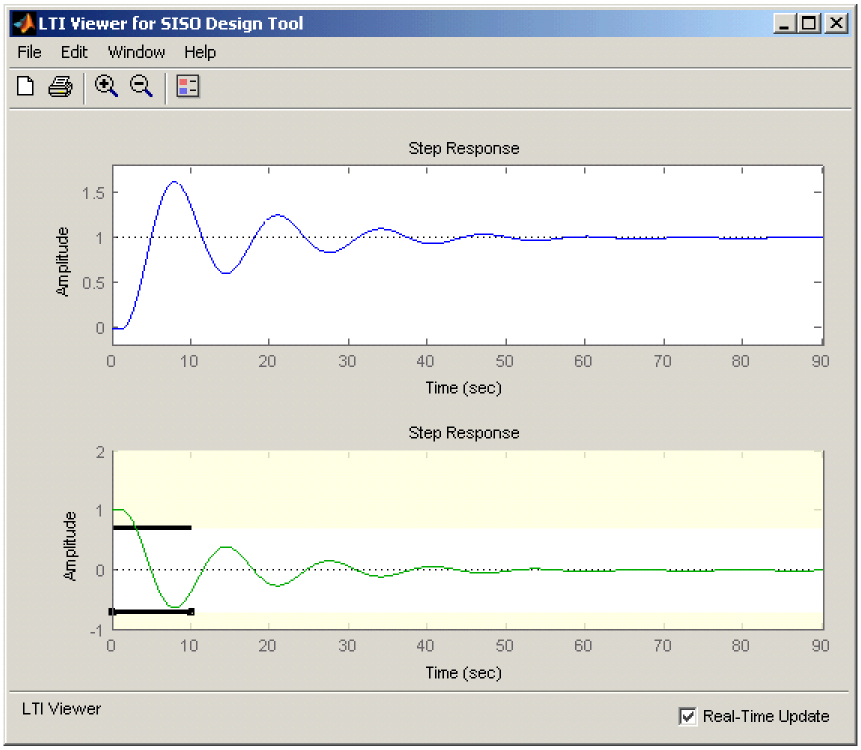
   b. Select `Closed Loop r to u` from the **Requirement for response** menu.

   c. Enter `0` to `10` for the **Time** range.

   d. Enter `-0.7` to `-0.7` for the **Amplitude** range.

   e. Click **OK**. The step response plot now contains a second horizontal line representing the lower limit on the actuator signal.

   f. To extend this limit for all times (to $t=\infty$), right-click in the yellow shaded area and select **Extend to inf**. The diagram should now appear as shown in the following figure.



Selecting the Design Requirements to Use During Response Optimization. The design requirements give constraints on the dynamics of the system and the values of response signals. The table in the **Design requirements** tab lists all design requirements in the design and analysis plots. Select the check boxes next to the design requirements you want to use in the response optimization. This example uses all the current design requirements.

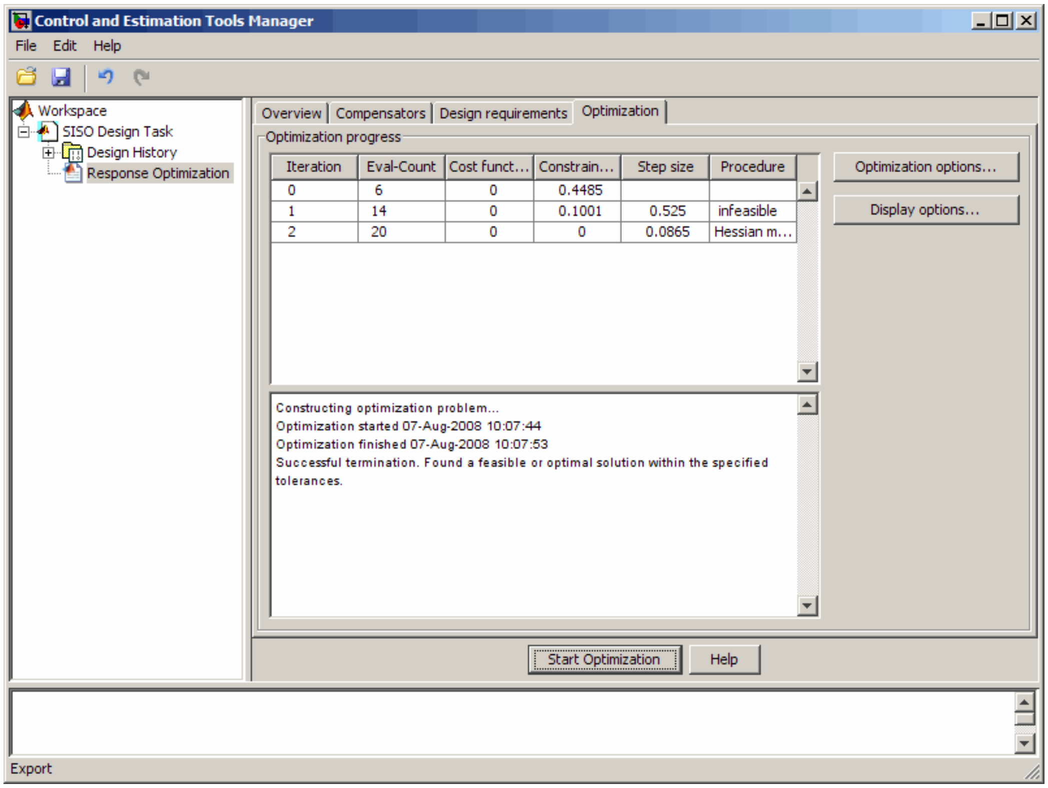
Optimizing the System's Response

After selecting the compensator elements to tune and adding design requirements for the response signals to satisfy, you are ready to being the response optimization.

The **Optimization** pane within the **Response Optimization** node of the Control and Estimation Tools Manager displays the progress of the response optimization. The pane also contains options to configure the types of progress information displayed during the optimization and options to configure the optimization methods and algorithms.

To optimize the response of the system in this example, click the **Start Optimization** button.

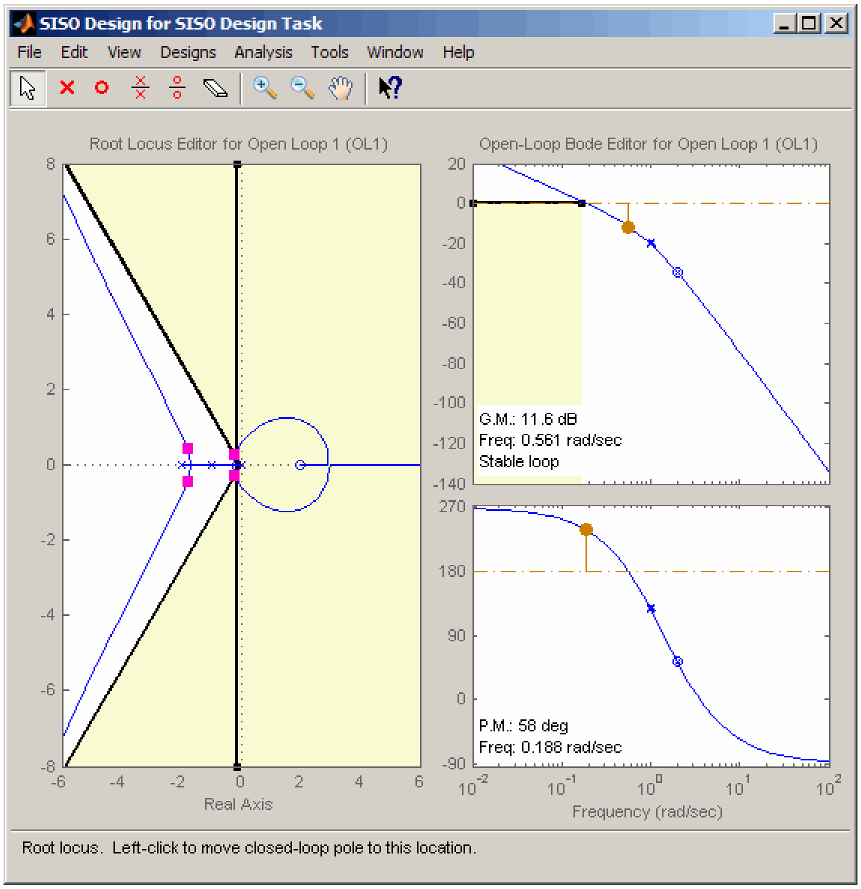
The **Optimization** pane displays the progress of the optimization, iteration by iteration, as shown next. Termination messages from the optimization algorithm and suggestions for improving convergence also appear here.



Iteration	Eval-Count	Cost funct...	Constrain...	Step size	Procedure
0	6	0	0.4485		
1	14	0	0.1001	0.525	infeasible
2	20	0	0	0.0865	Hessian m...

```
Constructing optimization problem...
Optimization started 07-Aug-2008 10:07:44
Optimization finished 07-Aug-2008 10:07:53
Successful termination. Found a feasible or optimal solution within the specified
tolerances.
```

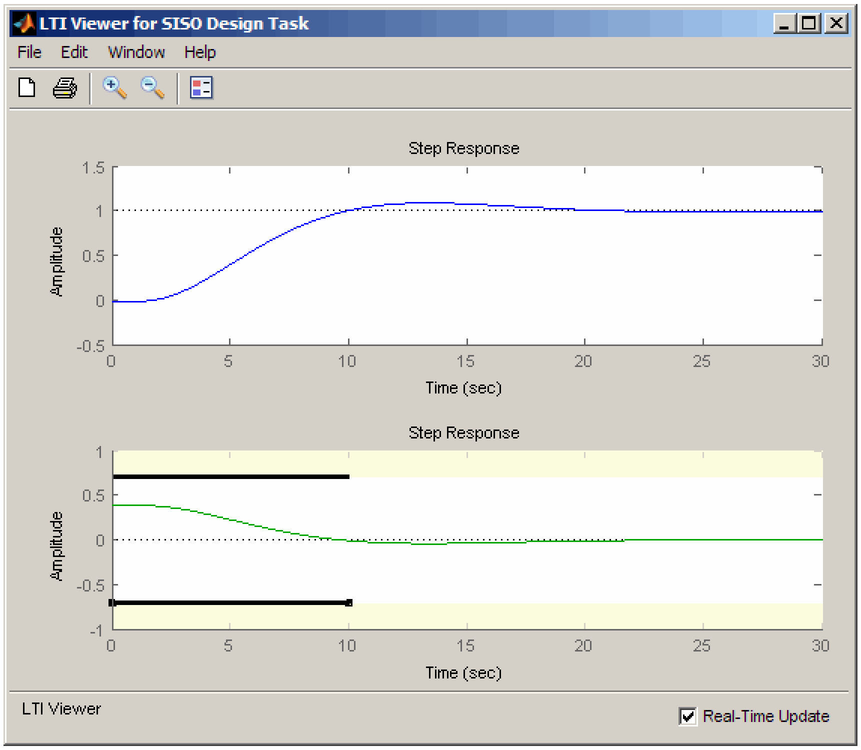
The optimized signals in the design and analysis plots appear as follows:


SISO Design for SISO Design Task
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open Loop 1 (OL1)
Real Axis
Open-Loop Bode Editor for Open Loop 1 (OL1)
Frequency (rad/sec)

G.M.: 11.6 dB
Freq: 0.561 rad/sec
Stable loop

P.M.: 58 deg
Freq: 0.188 rad/sec

Root locus. Left-click to move closed-loop pole to this location.



Creating and Displaying the Closed-Loop System

After designing a compensator by optimizing the response of the system, you can export the compensator to the MATLAB workspace, and create a model of the full closed-loop system.

1 Within the SISO Design Tool window, select **File > Export** to open the SISO Tool Export dialog box.

2 Select the compensator you designed, **Compensator C**, and then click the **Export to Workspace** button.

At the command line, enter the following command to create the closed-loop system, `CL`, from the open-loop transfer function, `open_loopTF`, and the compensator, `C`:

```
CL=feedback(C*open_loopTF,1)
```

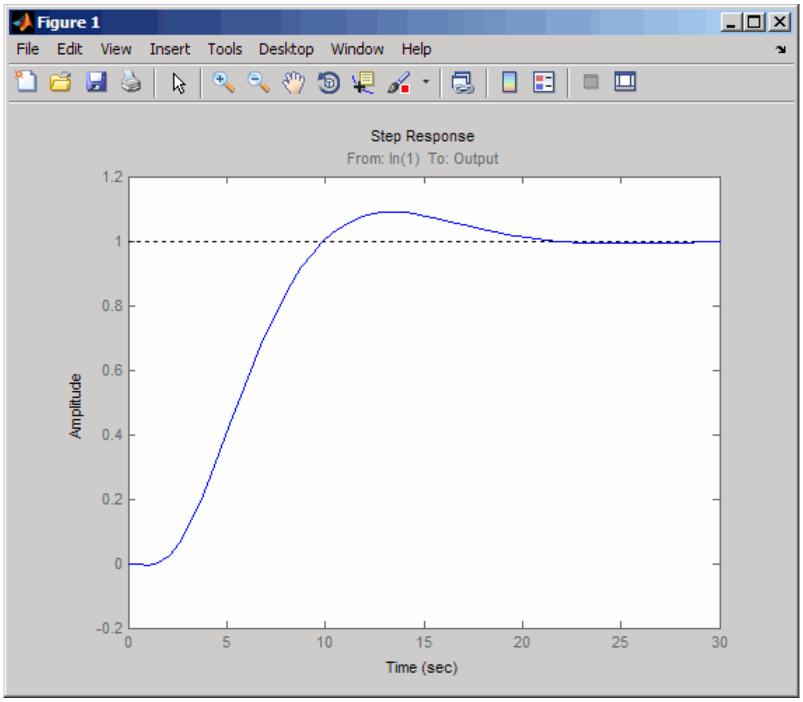
This returns the following model:

```
Zero/pole/gain from input to output "Output":
                -0.19414 (s-2)
----------------------------------------------
(s^2 + 0.409s + 0.1136) (s^2 + 3.591s + 3.418)
```

To create a step response plot of the closed loop system, enter the following command:

```
step(CL);
```

This produces the following figure:

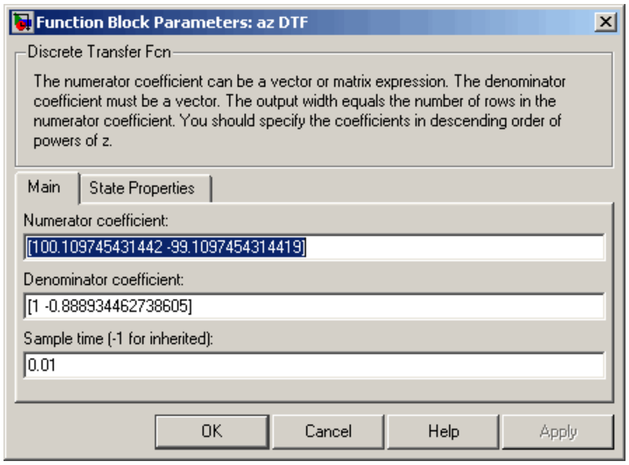


Designing Linear Controllers for Simulink Models

When you have Control System Toolbox and Simulink Control Design software, you can perform frequency-domain optimization of Simulink models.

You can use Simulink Control Design software to configure SISO Design Tool with compensators, inputs, outputs, and loops computed from a Simulink model. For more information, see “Designing Compensators” in Simulink Control Design documentation. After you configure the SISO Design Tool, you can use Simulink Design Optimization software to optimize the controller parameters of the linearized Simulink model. For an example of optimization-based control design for a model linearized using Simulink Control Design software, see “Tutorial — Designing a PID Controller Using Optimization-Based Tuning” in the *Simulink Design Optimization getting Started Guide.*.

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).



Function Block Parameters: az DTF

Discrete Transfer Fcn

The numerator coefficient can be a vector or matrix expression. The denominator coefficient must be a vector. The output width equals the number of rows in the numerator coefficient. You should specify the coefficients in descending order of powers of z.

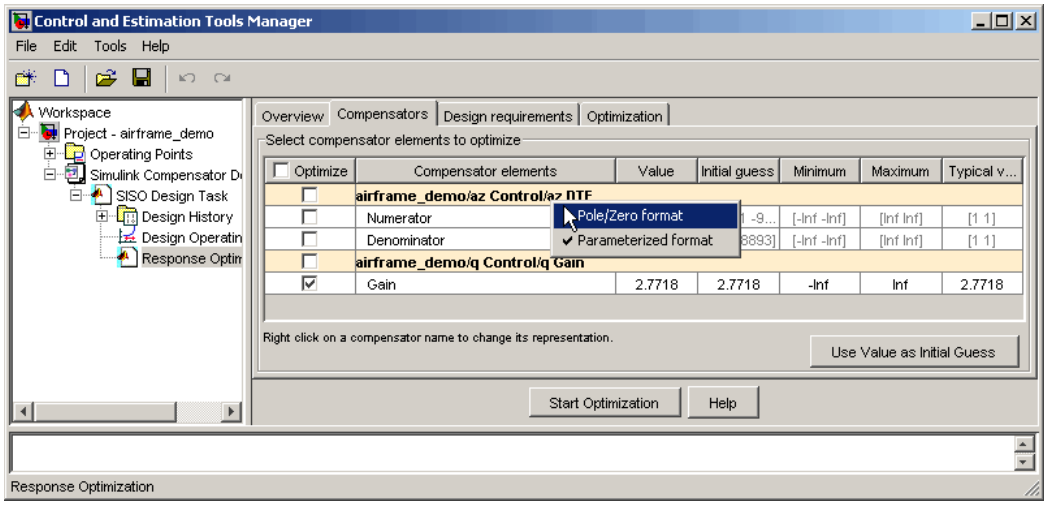
Main | State Properties

Numerator coefficient: [100.109745431442 -99.1097454314419]

Denominator coefficient: [1 -0.888934462738605]

Sample time (-1 for inherited): 0.01

Mask of a Simulink® compensator block



Select compensator elements to optimize

Optimize	Compensator elements	Value	Initial guess	Minimum	Maximum	Typical v...
☐	**airframe_demo/az Control/az DTF**					
☐	Numerator		1 -9...	[-Inf -Inf]	[Inf Inf]	[1 1]
☐	Denominator		8893]	[-Inf -Inf]	[Inf Inf]	[1 1]
☐	**airframe_demo/q Control/q Gain**					
☑	Gain	2.7718	2.7718	-Inf	Inf	2.7718

Right click on a compensator name to change its representation.

Response optimization compensators pane

Modeling Systems Using Lookup Tables

What Are Lookup Tables?

In this section...

Static Lookup Tables

Lookup tables are tables that store numeric data in a multidimensional array format. In the simpler two-dimensional case, lookup tables can be represented by matrices. Each element of a matrix is a numerical quantity, which can be precisely located in terms of two indexing variables. At higher dimensions, lookup tables can be represented by multidimensional matrices, whose elements are described in terms of a corresponding number of *indexing variables*.

Lookup tables provide a means to capture the dynamic behavior of a physical (mechanical, electronic, software) system. The behavior of a system with M inputs and N outputs can be approximately described by using N lookup tables, each consisting of an array with M dimensions.

You usually generate lookup tables by experimentally collecting or artificially creating the input and output data of a system. In general, you need as many indexing parameters as the number of input variables. Each indexing parameter may take a value within a predetermined set of data points, which are called the *breakpoints*. The set of all breakpoints corresponding to an indexing variable is called a *grid*. Thus, a system with M inputs is gridded by M sets of breakpoints. The software uses the breakpoints to locate the array elements, where the output data of the system are stored. For a system with N outputs, the software locates the N array elements and then stores the corresponding data at these locations.

After you create a lookup table using the input and output measurements as described previously, you can use the corresponding multidimensional array of values in applications without having to remeasure the system outputs. In fact, you need only the input data to locate the appropriate array elements in the lookup table because the software reads the approximate system output from the data stored at these locations. Therefore, a lookup table provides a suitable means of capturing the input-output mapping of a *static* system in the form of numeric data stored at predetermined array locations. For more information, see “About Lookup Table Blocks” in the Simulink documentation.

You can use Simulink Design Optimization software to estimate lookup table values, as described in “Estimating Values of Lookup Tables” on page 5-5.

Adaptive Lookup Tables

Statically defined lookup tables, as described in “Static Lookup Tables” on page 5-2, cannot accommodate the *time-varying* behavior (characteristics) of a physical plant. Static lookup tables establish a permanent and static mapping of input-output behavior of a physical system. Conversely, the behavior of actual physical systems often varies with time due to wear, environmental conditions, and manufacturing tolerances. With such variations, the static mapping of input-output behavior of a plant described by the lookup table may no longer provide a valid representation of the plant characteristics.

Adaptive lookup tables incorporate the time-varying behavior of physical plants into the lookup table generation and maintenance process while providing all of the functionality of a regular lookup table.

The adaptive lookup table receives the input and output measurements of a plant’s behavior, which are then used to dynamically create and update the content of the underlying lookup table. In addition to requiring the input data to create the lookup table, the adaptive lookup table also uses the output data of the plant to recalculate the table values. For example, you can collect the output data of the plant by placing sensors at appropriate locations in a physical system.

The software uses the input measurements to locate the array elements by comparing these input values with the breakpoints defined for each indexing variable. Next, it uses the output measurements to recalculate the numeric value stored at these array locations. However, unlike a regular table, which only stores the array data before the actual use of the lookup table, the adaptive table continuously improves the content of the lookup table. This continuous improvement of the table data is referred to as the *adaptation process* or *learning process*.

The adaptation process involves statistical and signal processing algorithms to recapture the input-output behavior of the plant. The adaptive lookup table always tries to provide a valid representation of the plant dynamics even though the plant behavior may be time varying. The underlying signal processing algorithms are also robust against reasonable measurement noise and they provide appropriate filtering of noisy output measurements. To learn more about how to model systems using adaptive lookup tables, see "Capturing Time-Varying System Behavior Using Adaptive Lookup Tables" on page 5-37.

Estimating Values of Lookup Tables

In this section...

How to Estimate Values of a Lookup Table

You can use lookup table Simulink blocks to approximate a system's behavior, as described in "Working with Lookup Tables" in the Simulink documentation. After you build your system using lookup tables, you can use Simulink Design Optimization software to estimate the table values from measured I/O data.

Estimating lookup table values is an example of estimating parameters which are matrices or multi-dimensional arrays. The workflow for estimating parameters of a lookup table consist of the following tasks:

1. Creating a Simulink model using lookup table blocks.
2. Importing the measured input and output (I/O) data from which you want to estimate the table values.
3. Analyzing and preparing the I/O data for estimation.
4. Estimating the lookup table values.
5. Validating the estimated table values using a validation data set.

The following examples illustrate how to estimate the lookup table values:

- "Example — Estimating Lookup Table Values from Data" on page 5-6
- "Example — Estimating Constrained Values of a Lookup Table" on page 5-20

Example — Estimating Lookup Table Values from Data

Objectives

This example shows how to estimate lookup table values from time-domain input-output (I/O) data.

About the Data

In this example, you use the I/O data in `lookup_regular.mat` to estimate the values of a lookup table. The MAT-file includes the following variables:

- `xdata1` — Consists of 63 uniformly-sampled input data points in the range [0,6.5].
- `ydata1` — Consists of output data corresponding to the input data samples.
- `time1` — Time vector.

You use the I/O data to estimate the lookup table values in the `lookup_regular` Simulink model. The lookup table in the model contains ten values, which are stored in the MATLAB variable `table`. The initial table values comprise a vector of 0s. To learn more about how to model a system using lookup tables, see “Working with Lookup Tables” in the Simulink documentation.

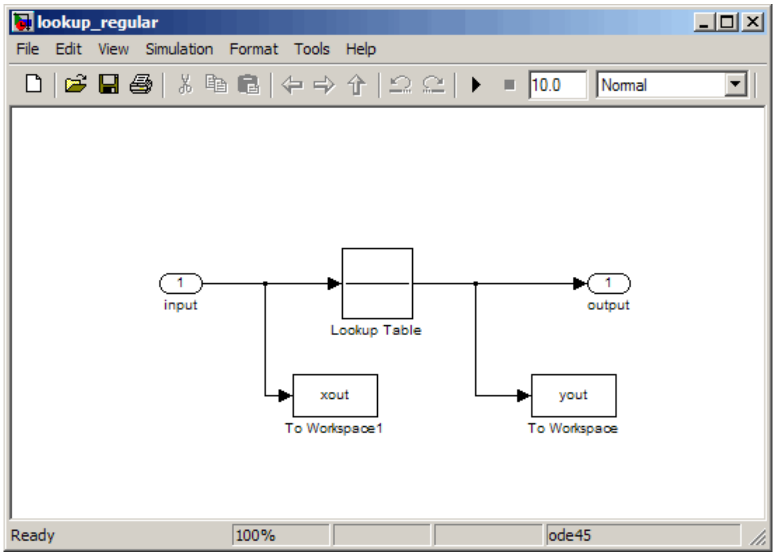
Configuring a Project for Parameter Estimation

To estimate the lookup table values, you must first configure a Control and Estimation Tools Manager project.

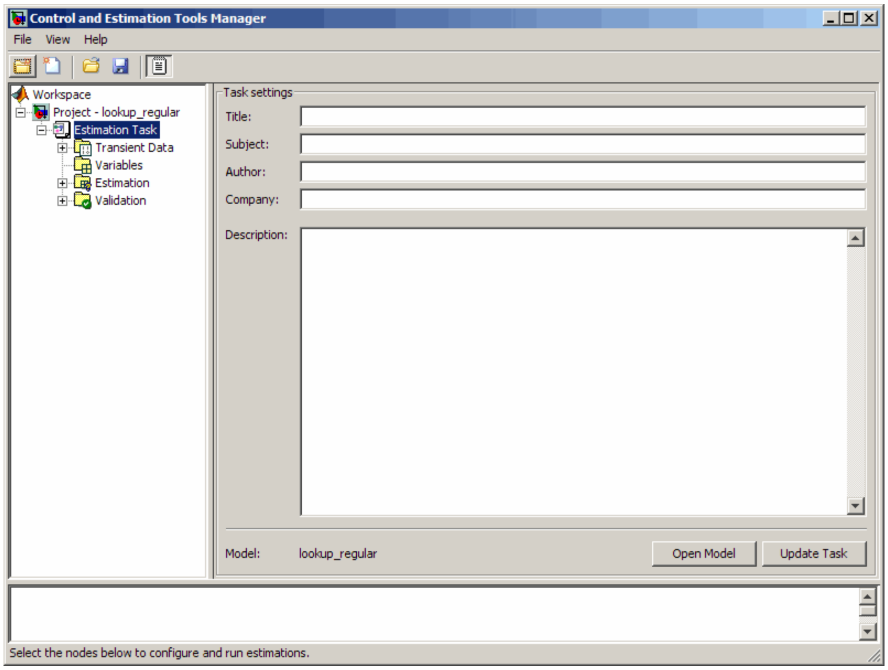
1 Open the lookup table model by typing the following command at the MATLAB prompt:

```
lookup_regular
```

This command opens the Simulink model, and loads the estimation data into the MATLAB workspace.



2 In the Simulink model, select **Tools > Parameter Estimation** to open a new project named **lookup_regular** in the Control and Estimation Tools Manager GUI.



Model: lookup_regular

Select the nodes below to configure and run estimations.

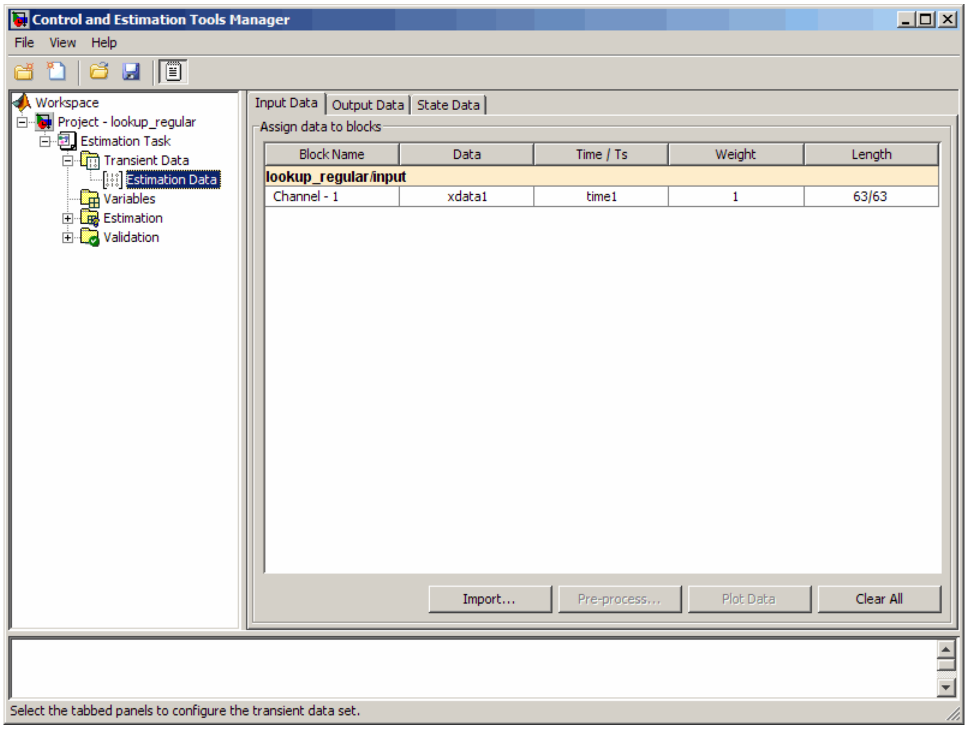
Estimating the Table Values Using Default Settings

After you configure a project for parameter estimation, as described in "Configuring a Project for Parameter Estimation" on page 5-6, use the following steps to estimate the lookup table values.

1 Import the estimation data, as described in the "Importing Data into the GUI" section of "Tutorial — Preparing Data for Parameter Estimation Using the GUI".

You can also load a preconfigured project that already contains the imported data. To do so, type the following command at the MATLAB prompt:

```
lookup_regular;
speload('lookup_regular_import', 'Project - lookup_regular',...
'Estimation Data');
```

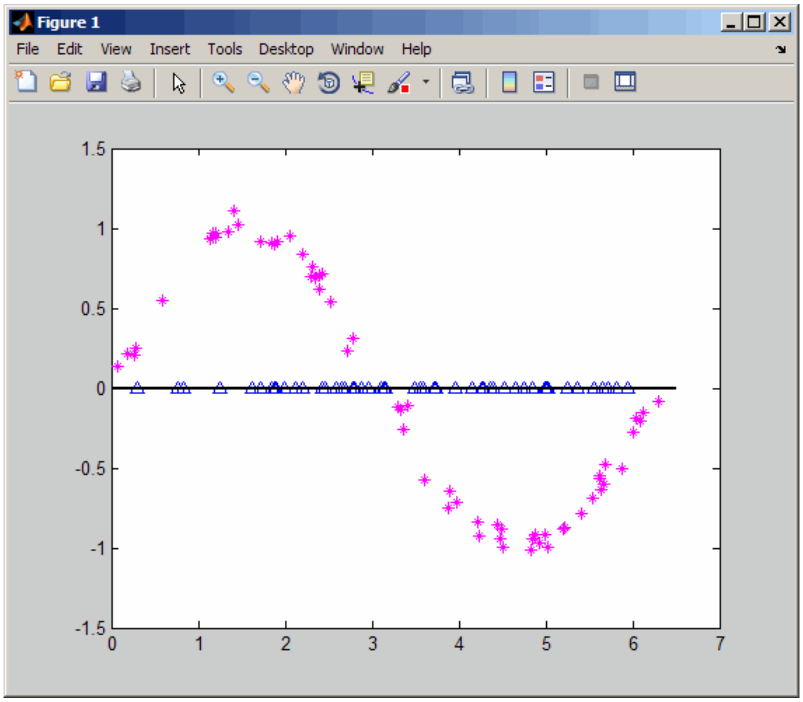


Block Name	Data	Time / Ts	Weight	Length
lookup_regular/input				
Channel - 1	xdata1	time1	1	63/63

Select the tabbed panels to configure the transient data set.

2 Run an initial simulation to view the I/O data, simulated output, and the initial table values. To do so, type the following commands at the MATLAB prompt:

```
sim('lookup_regular')
figure(1); plot(xdata1,ydata1, 'm*', xout, yout,'b^')
hold on; plot(linspace(0,6.5,10), table, 'k', 'LineWidth', 2)
```



The x- and y-axes of the figure represent the input and output data, respectively. The figure shows the following plots:

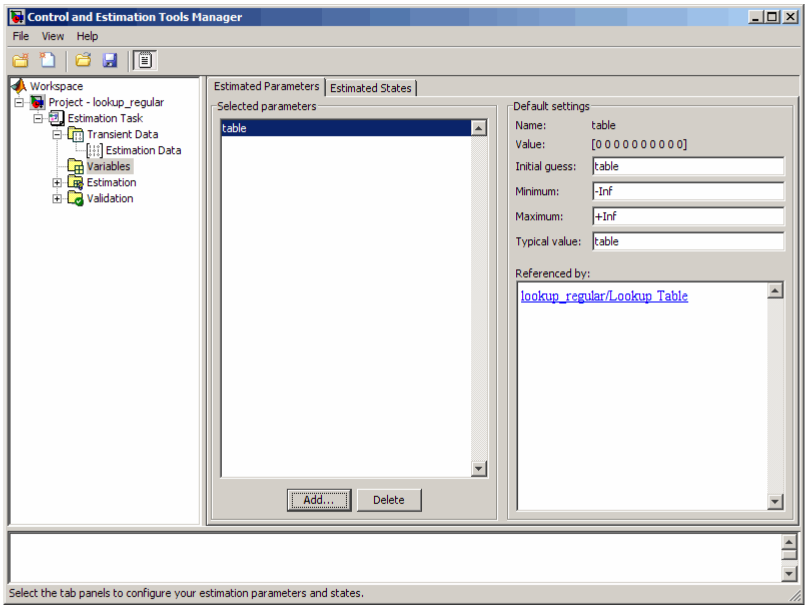
- Measured data — Represented by the magenta stars (*).
- Initial table values — Represented by the black line.
- Initial simulation data — Represented by the blue deltas (Δ).

3 Select the table values to estimate.

a In the Control and Estimation Tools Manager GUI, select the **Variables** node under the **Estimation Task** node.

b Click **Add** to open the Select Parameters dialog box, which shows the Simulink model parameters.

c Select `table`, and click **OK** to add the table values to the **Estimated Parameters** tab.



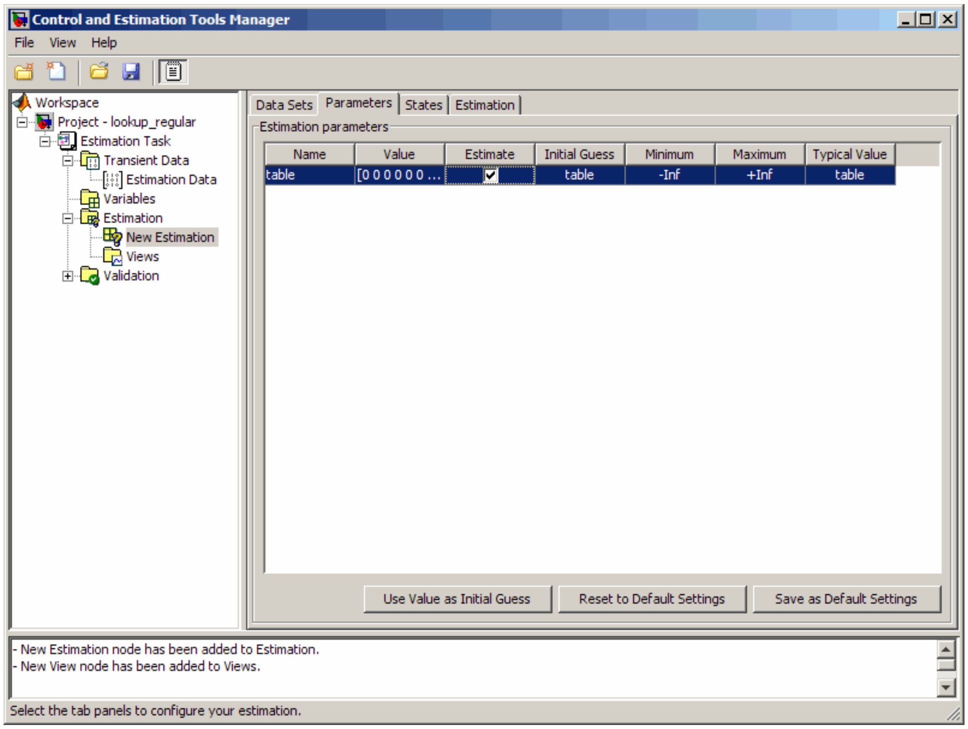
Name: table
Value: [0 0 0 0 0 0 0 0 0 0]
Initial guess: table
Minimum: -Inf
Maximum: +Inf
Typical value: table

Select the tab panels to configure your estimation parameters and states.

The **Default settings** area of the GUI displays the default settings for the table values. The **Value** field displays the initial table values, which comprise a vector of ten 0s.

d Select the **Estimation** node, and click **New** to add a **New Estimation** node.

e In the **Parameters** tab of the **New Estimation** node, select the **Estimate** check box to specify the lookup table values, `table`, for estimation.

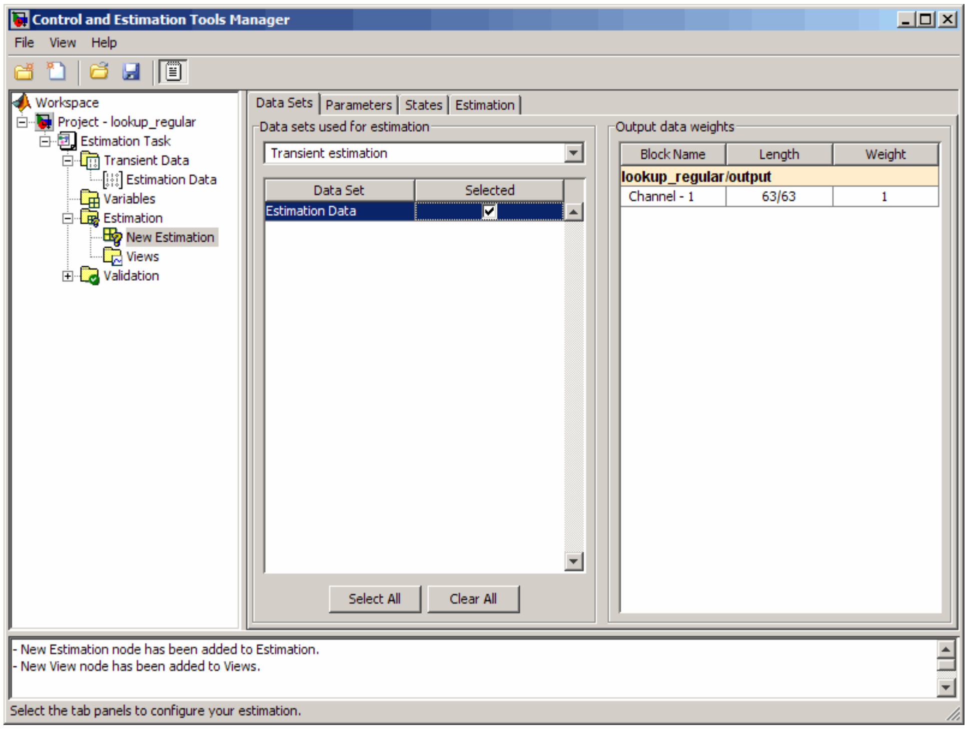


Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
table	[0 0 0 0 0 0 ...	✓	table	-Inf	+Inf	table

- New Estimation node has been added to Estimation.
- New View node has been added to Views.

Select the tab panels to configure your estimation.

4 In the **Data Sets** tab of the **New Estimation** node, select the **Selected** check box to specify the estimation data set.



Data Set	Selected
Estimation Data	✓

Block Name	Length	Weight
lookup_regular/output		
Channel - 1	63/63	1

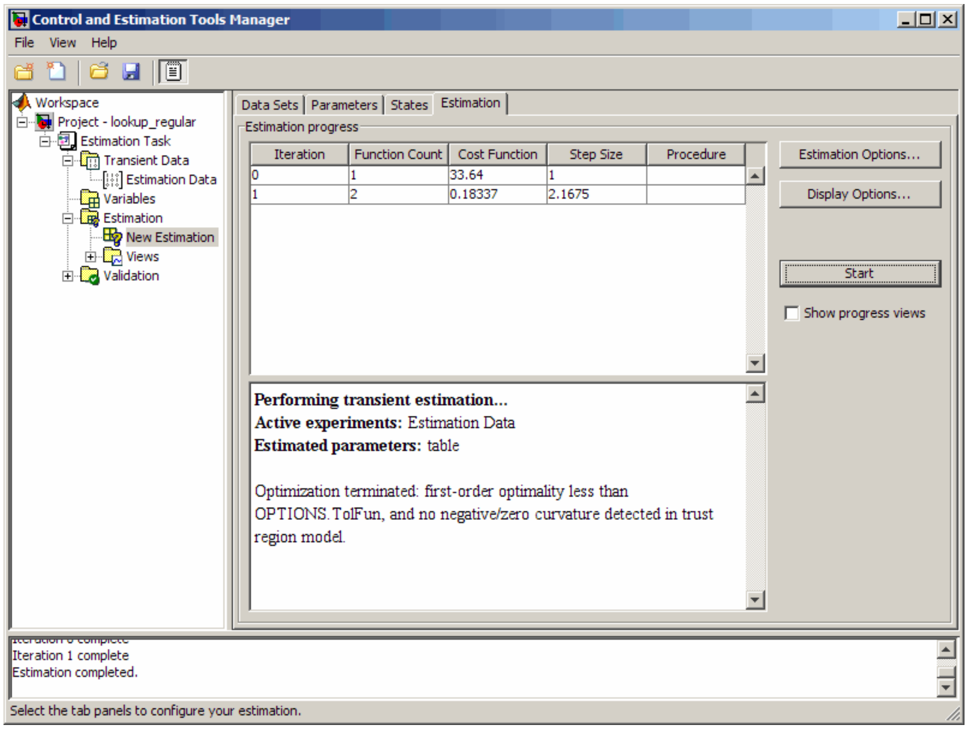
- New Estimation node has been added to Estimation.
- New View node has been added to Views.

Select the tab panels to configure your estimation.

5 Estimate the table values using the default settings.

a In the **Estimation** tab of the **New Estimation** node, click **Start** to start the estimation.

The Control and Tools Manager GUI updates at each iteration, and provides information about the estimation progress. After the estimation completes, the Control and Estimation Tools Manager GUI looks similar to the following figure.



Iteration	Function Count	Cost Function	Step Size	Procedure
0	1	33.64	1	
1	2	0.18337	2.1675	

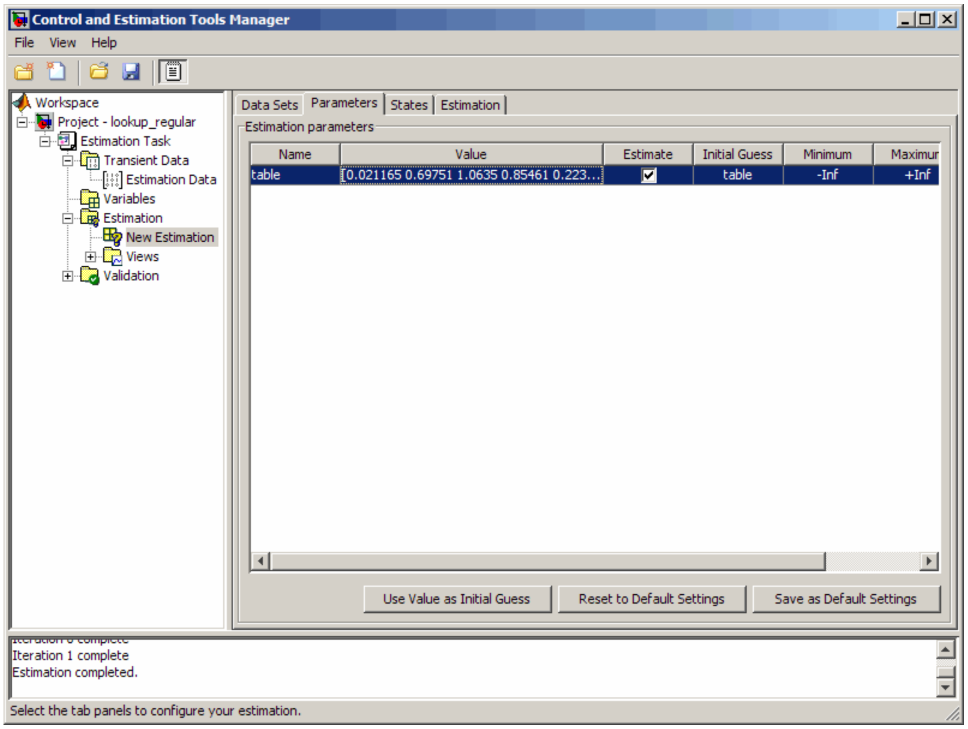
Performing transient estimation...
Active experiments: Estimation Data
Estimated parameters: table

Optimization terminated: first-order optimality less than OPTIONS.TolFun, and no negative/zero curvature detected in trust region model.

```
Iteration 0 complete
Iteration 1 complete
Estimation completed.
```

Select the tab panels to configure your estimation.

b Select the **Parameters** tab in the **New Estimation** node to view the estimated table values, which appear in the **Value** field.



Name	Value	Estimate	Initial Guess	Minimum	Maximur
table	[0.021165 0.69751 1.0635 0.85461 0.223...	✓	table	-Inf	+Inf

```
Iteration 1 complete
Estimation completed.
```

Select the tab panels to configure your estimation.

Validating the Estimation Results

After you estimate the table values, as described in "Estimating the Table Values Using Default Settings" on page 5-8, you must use another data set to validate that you have not overfitted the model. You plot and examine the following plots to validate the estimation results:

- Residuals plot
- Measured and simulated data plots

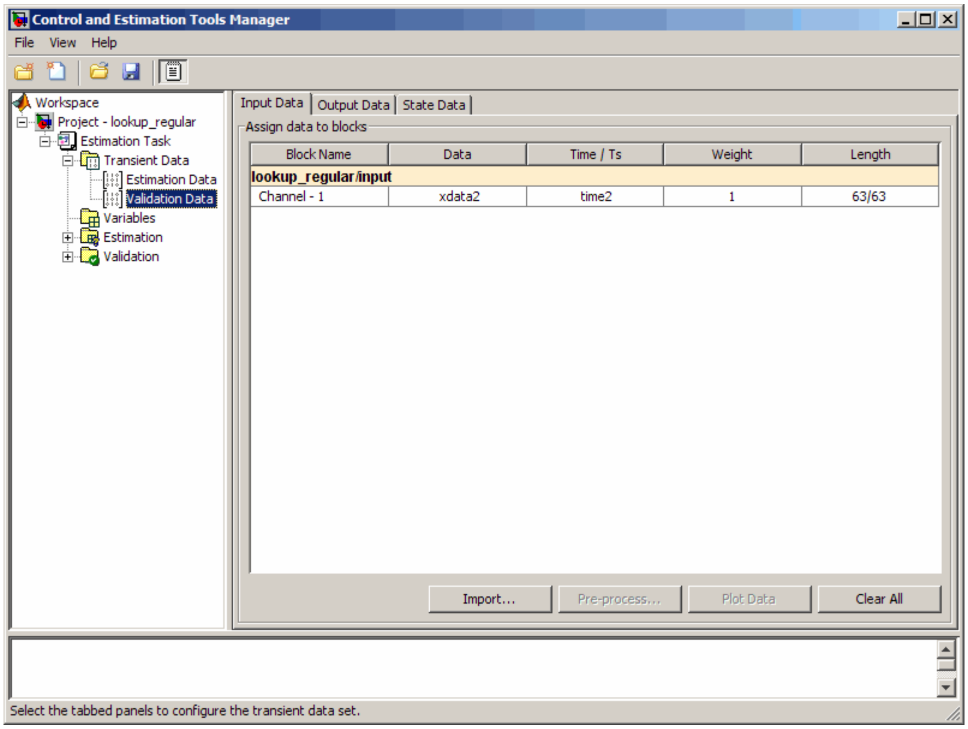
To validate the estimation results:

1 Import the validation data set in the Control and Estimation Tools Manager GUI, as described in “Importing Data into the GUI” section of “Tutorial — Preparing Data for Parameter Estimation Using the GUI”.

The validation data contains the input data, output data and time vector in the MATLAB variables `xdata2`, `ydata2` and `time2` respectively.

You can also load a project that already contains the estimated parameters, and the validation data set. To do so, type the following commands at the MATLAB prompt:

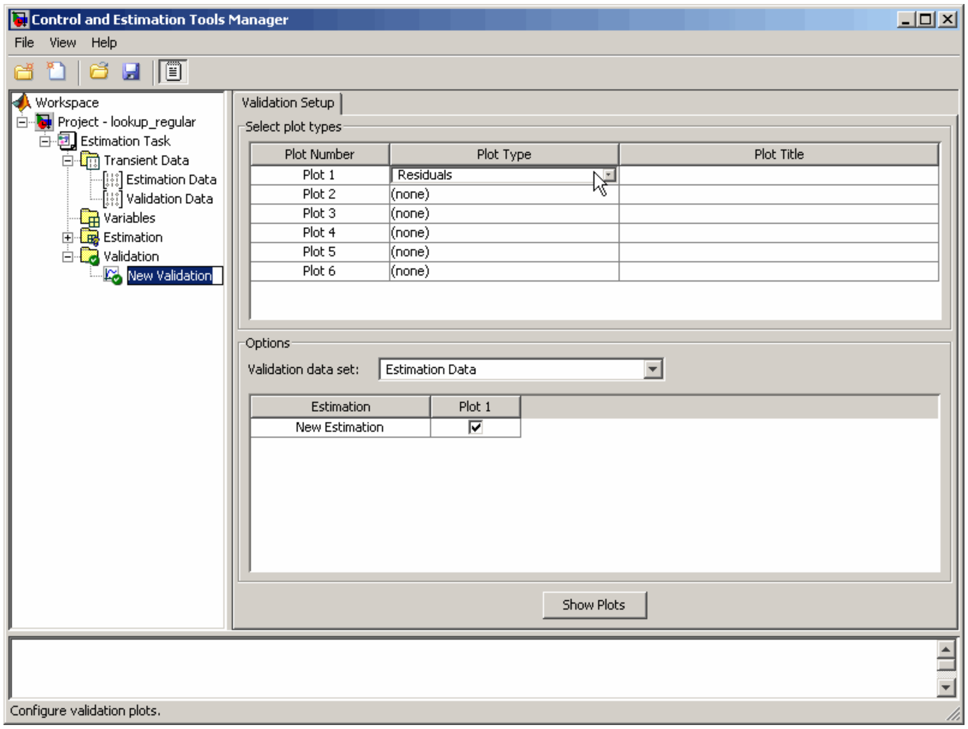
```
lookup_regular;
speload('lookup_regular_val', 'Project - lookup_regular',...
'Validation Data')
```



Block Name	Data	Time / Ts	Weight	Length
lookup_regular/input				
Channel - 1	xdata2	time2	1	63/63

Select the tabbed panels to configure the transient data set.

This project also contains the `Residuals` plot already configured in the **Select plot types** area of the GUI, as shown in the next figure.



Select plot types

Plot Number	Plot Type	Plot Title
Plot 1	Residuals	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

Options

Validation data set: Estimation Data

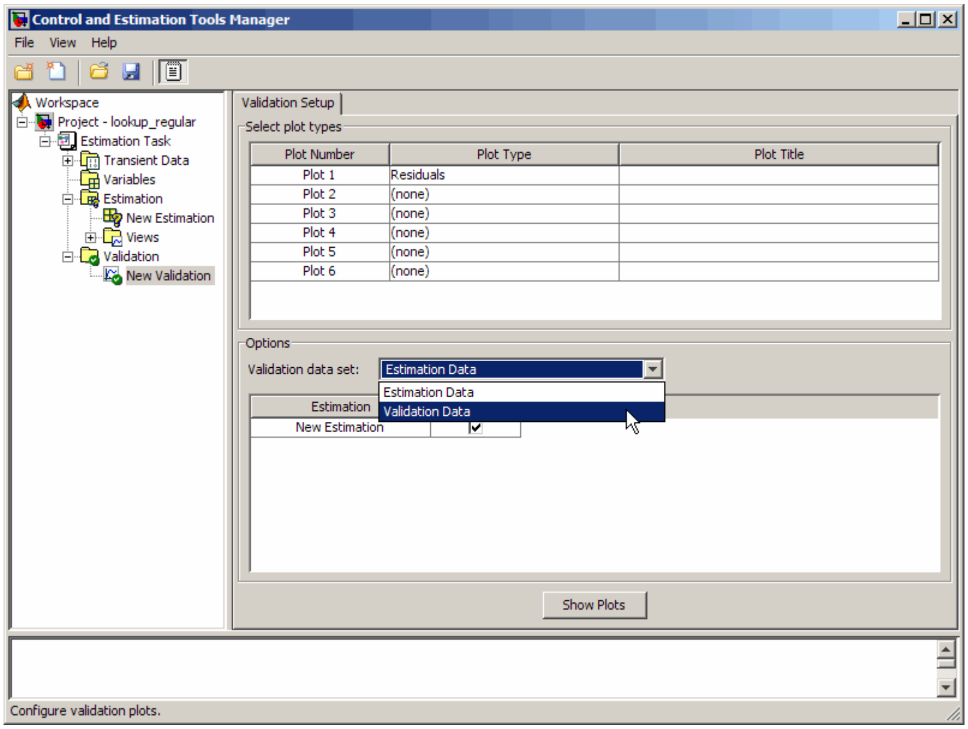
Estimation	Plot 1
New Estimation	✓

Configure validation plots.

2 Plot and examine the residuals:

a Select the **New Validation** node under the **Validation** node.

b In the **Options** area, select `Validation Data` from the **Validation data set** drop-down list.



Plot Number	Plot Type	Plot Title
Plot 1	Residuals	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

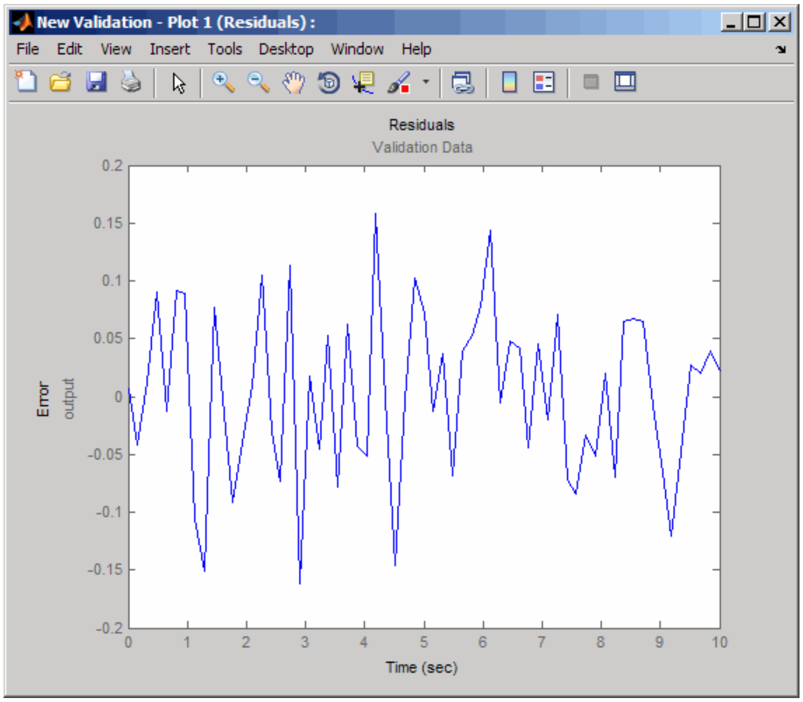
Validation data set: Estimation Data

Estimation Data
Validation Data

Estimation	
New Estimation	✓

Configure validation plots.

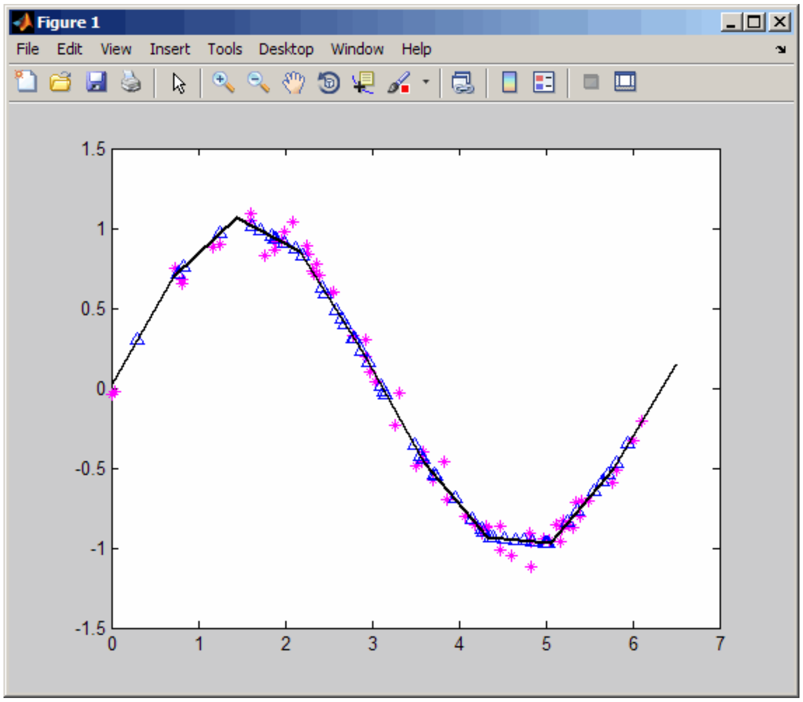
c Click **Show Plots** to open the residuals plot.



The residuals, which show the difference between the simulated and measured data, lie in the range [-0.15,0.15]— within 15% of the maximum output variation. This indicates a good match between the measured and the simulated table data values.

d Plot and examine the estimated table values against the validation data set and the simulated table values by typing the following commands at the MATLAB prompt.

```
sim('lookup_regular')
figure(2); plot(xdata2,ydata2, 'm*', xout, yout,'b^')
hold on; plot(linspace(0,6.5,10), table, 'k', 'LineWidth', 2)
```



The plot shows that the table values, displayed as the black line, match both the validation data and the simulated table values. The table data values cover the entire range of input values, which indicates that all the lookup table values have been estimated.

Example — Estimating Constrained Values of a Lookup Table

- “Objectives” on page 5-21

Objectives

This example shows how to estimate constrained values of a lookup table. You apply monotonically increasing constraints to the lookup table values, and use the GUI to estimate the table values.

About the Data

In this example, you use `lookup_increasing.mat`, which contains the measured I/O data for estimating the lookup table values. The MAT-file includes the following variables:

- `xdata1` — Consists of 602 uniformly-sampled input data points in the range [-5,5].
- `ydata1` — Output data corresponding to the input data samples.

Note The output data is a monotonically increasing function of the input data.

- `time1` — Time vector.

You use the I/O data to estimate the values of the lookup table in the `lookup_increasing` Simulink model. The table contains eleven values, which are stored in the MATLAB variable `table`. To learn more about how to specify the table’s values, see “Entering Breakpoints and Table Data” in the Simulink documentation.

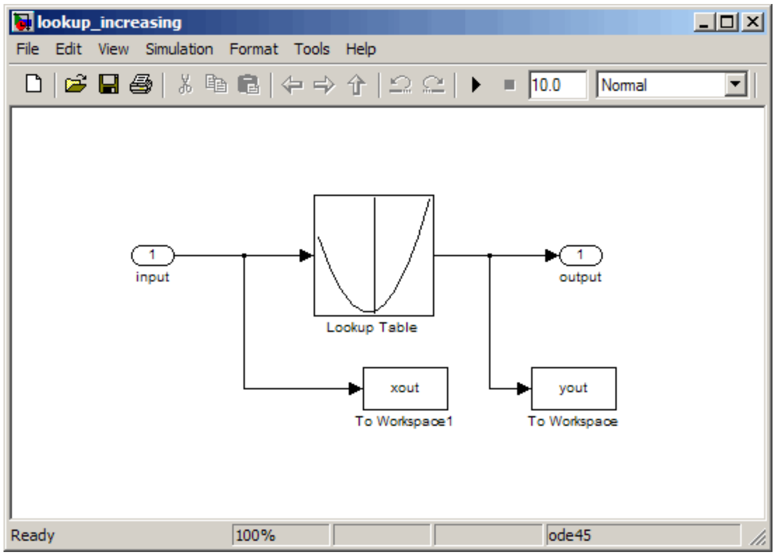
Configuring a Project for Parameter Estimation

To estimate the monotonically increasing lookup table values, you must first configure a Control and Estimation Tools Manager project.

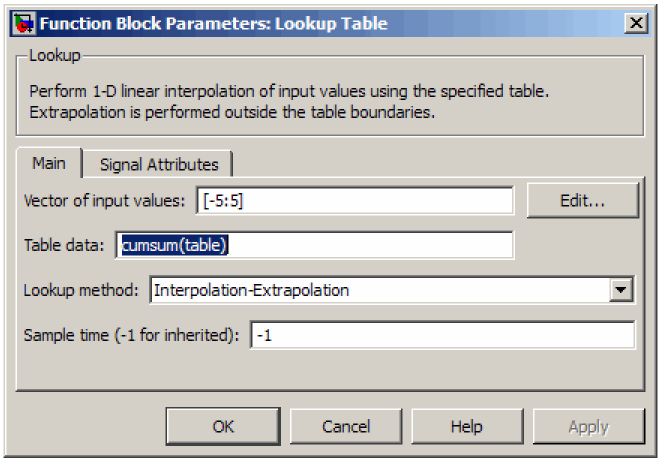
1 Open the lookup table model by typing the following command at the MATLAB prompt:

```
lookup_increasing
```

This command opens the Simulink model, and loads the estimation data in the MATLAB workspace.



2 Double-click the Lookup Table block to view the monotonically increasing constraint applied to the table output values.



Lookup

Perform 1-D linear interpolation of input values using the specified table. Extrapolation is performed outside the table boundaries.

Vector of input values: [-5:5]

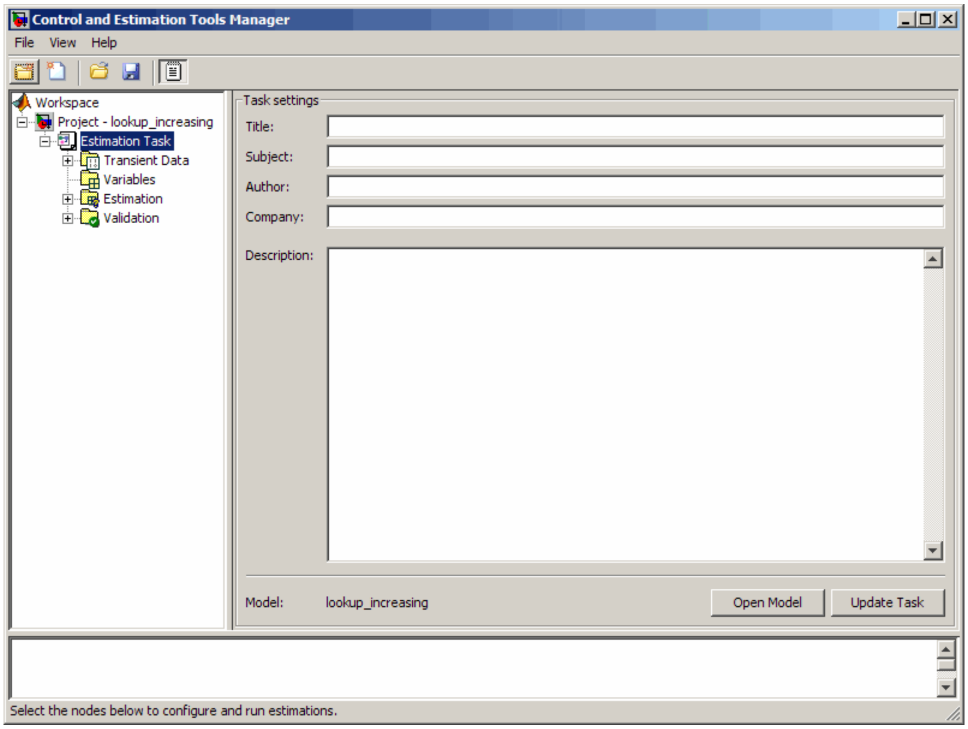
Table data: cumsum(table)

Lookup method: Interpolation-Extrapolation

Sample time (-1 for inherited): -1

The **Table data** field of the Function Block Parameters dialog box shows the constraint. The cumulative sum function, `cumsum`, applies a monotonically increasing constraint on the table output values. This function computes the cumulative sum of the table values based on estimation of the individual table elements from the I/O data.

3 In the Simulink model, select **Tools > Parameter Estimation** to open a new project named **lookup_increasing** in the Control and Estimation Tools Manager GUI.



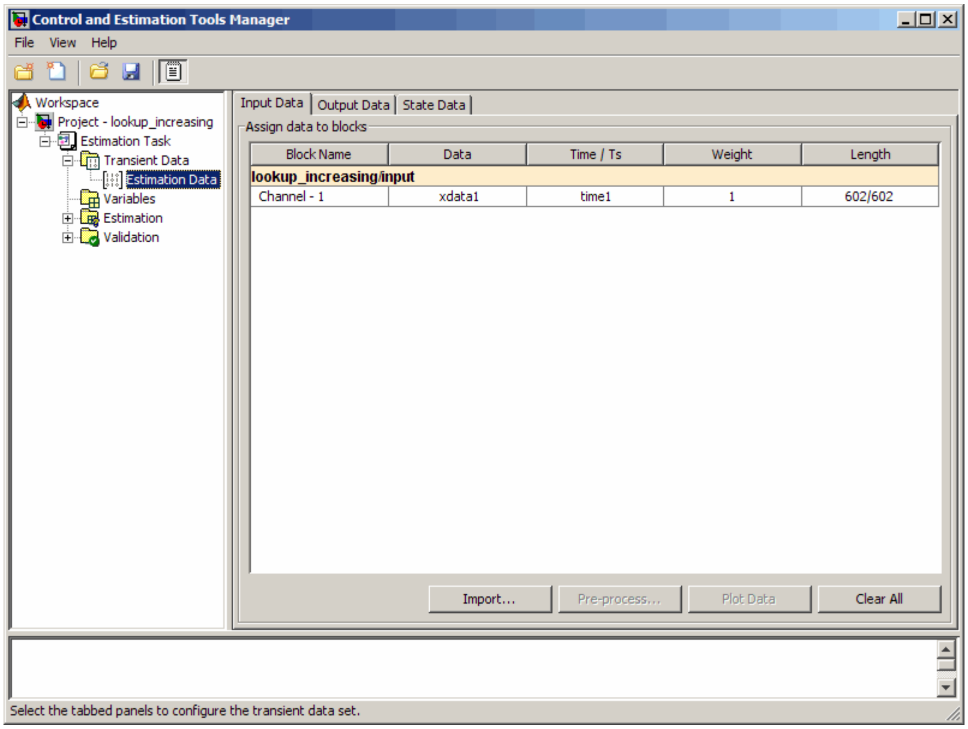
Estimating the Monotonically Increasing Table Values Using Default Settings

After you configure a project for parameter estimation, as described in "Configuring a Project for Parameter Estimation" on page 5-21, use the following steps to estimate the constrained lookup table values:

1 Import the estimation data, as described in the "Importing Data into the GUI" section of "Tutorial — Preparing Data for Parameter Estimation Using the GUI".

You can also load a preconfigured project that already contains the imported data. To do so, type the following commands at the MATLAB prompt:

```
lookup_increasing;
speload('lookup_increasing_import', 'Project - lookup_increasing',...
'Estimation Data')
```

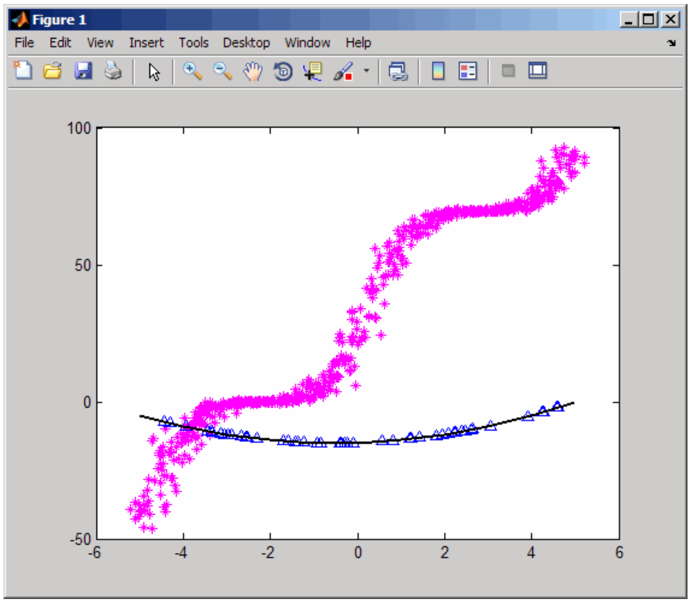


Block Name	Data	Time / Ts	Weight	Length
lookup_increasing/input				
Channel - 1	xdata1	time1	1	602/602

Select the tabbed panels to configure the transient data set.

2 Run an initial simulation to view the measured data, simulated table values and the initial table values by typing the following commands at the MATLAB prompt:

```
sim('lookup_increasing')
figure(1); plot(xdata1,ydata1, 'm*', xout, yout,'b^')
hold on; plot(-5:5, cumsum(table), 'k', 'LineWidth', 2)
```



The x- and y-axes represents the input and output data, respectively. The figure shows the following plots:

- Measured data — Represented by the magenta stars (*).

Note As described in “About the Data” on page 5-21, the output data is a monotonically increasing function of the input data.

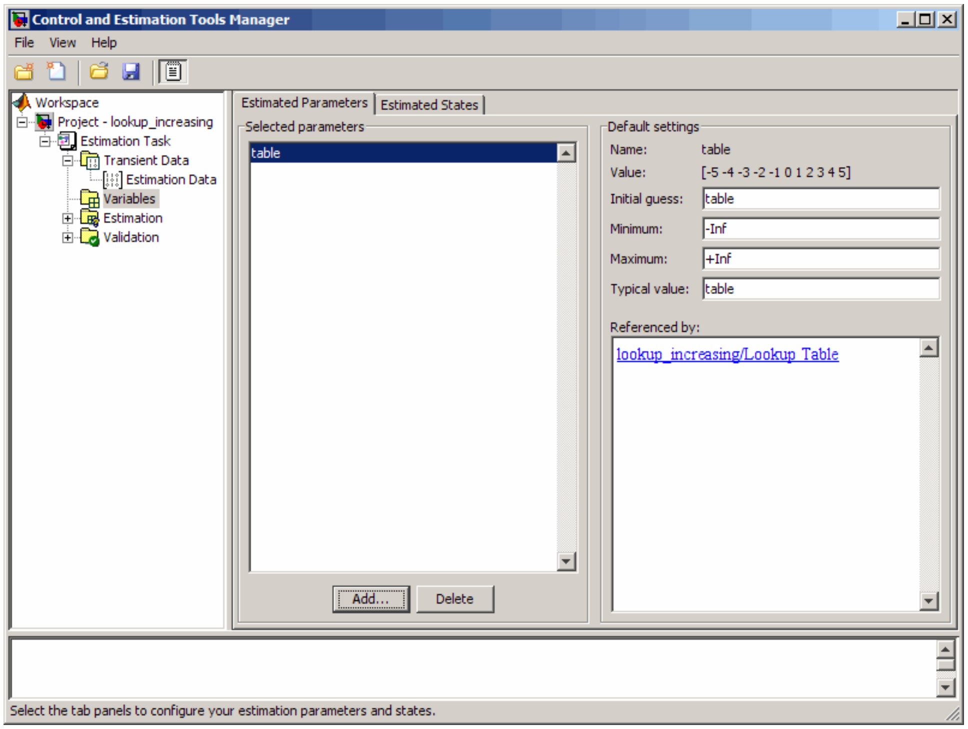
- Initial table values — Represented by the black line.
- Initial simulation data — Represented by the blue deltas (Δ).

3 Select the table output values to estimate.

a In the Control and Estimation Tools Manager GUI, select the **Variables** node under the **Estimation Task** node.

b Click **Add** to open the Select Parameters dialog box, where you see the Simulink model parameters.

c Select `table`, and click **OK** to add the table values to the **Estimated Parameters** tab.



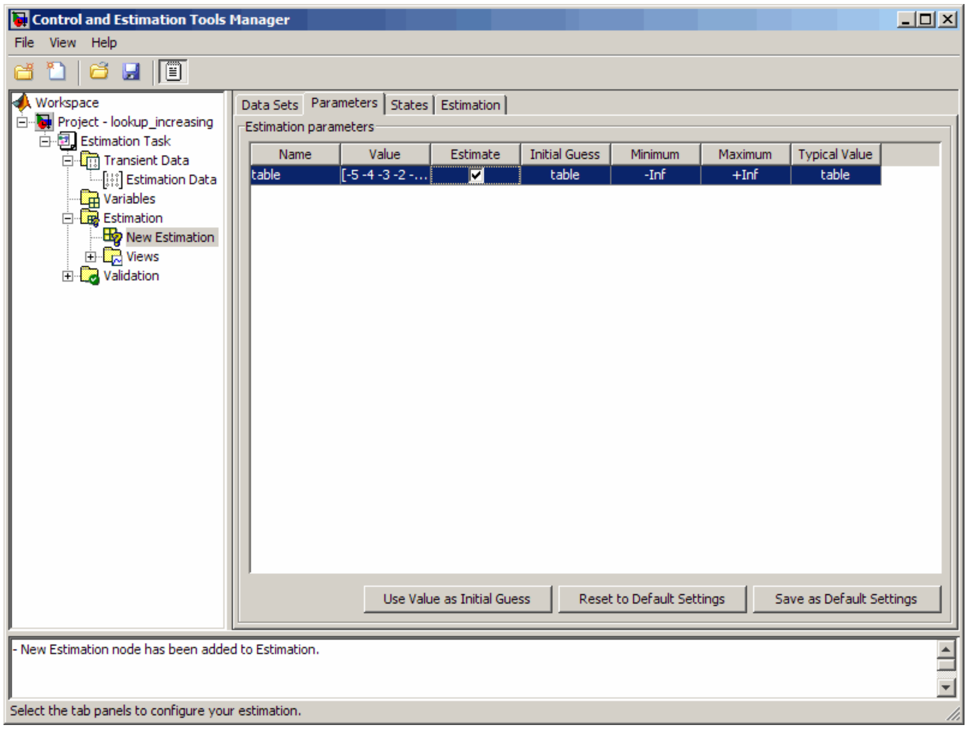
Name: table
Value: [-5 -4 -3 -2 -1 0 1 2 3 4 5]
Initial guess: table
Minimum: -Inf
Maximum: +Inf
Typical value: table

Select the tab panels to configure your estimation parameters and states.

The **Default settings** area of the GUI displays the default settings for the table values. The **Value** field displays the initial table values.

d Select the **Estimation** node, and click **New** to add a **New Estimation** node.

e In the **Parameters** tab of the **New Estimation** node, select the lookup table values `table` for estimation.

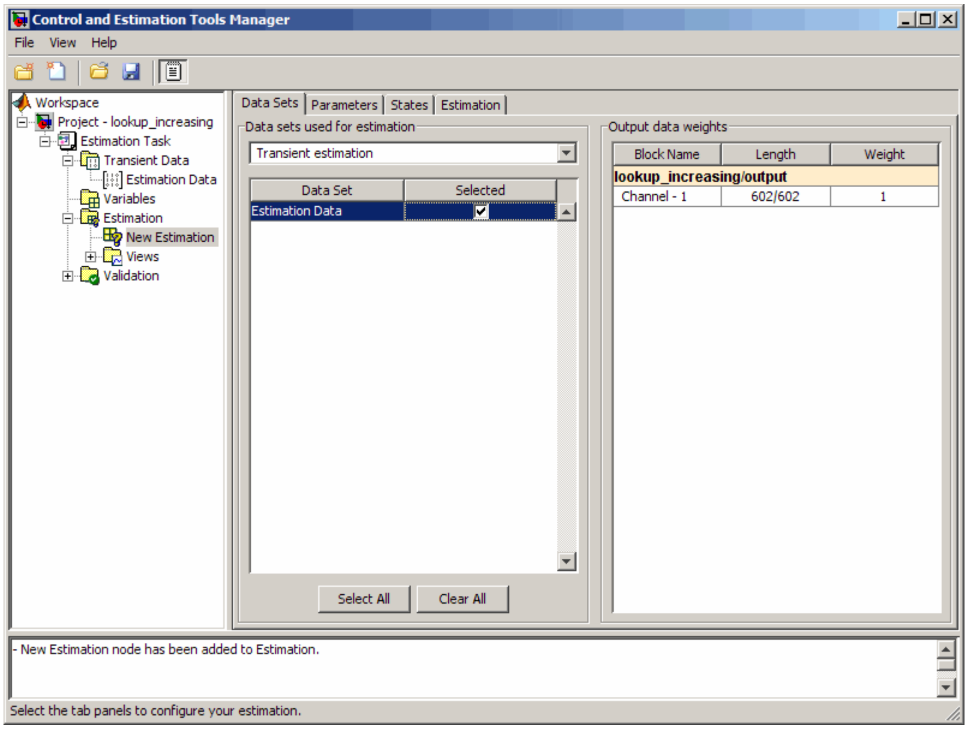


Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
table	[-5 -4 -3 -2 -...	✓	table	-Inf	+Inf	table

- New Estimation node has been added to Estimation.

Select the tab panels to configure your estimation.

4 In the **Data Sets** tab of the **New Estimation** node, select the **Selected** check-box to specify the estimation data set.



Data Set	Selected
Estimation Data	✓

Block Name	Length	Weight
lookup_increasing/output		
Channel - 1	602/602	1

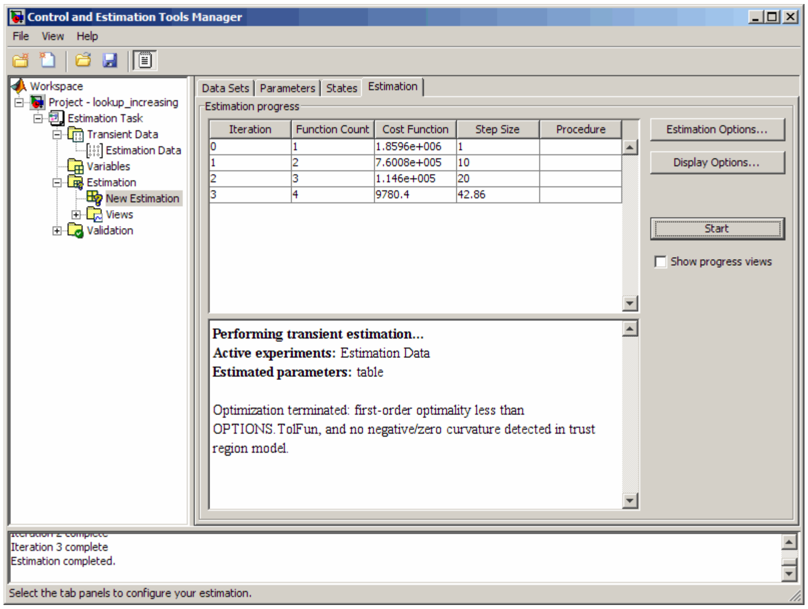
- New Estimation node has been added to Estimation.

Select the tab panels to configure your estimation.

5 Estimate the parameters using the default settings.

a In the **Estimation** tab of the **New Estimation** node, click **Start** to start the estimation.

The Control and Tools Manager GUI updates at each iteration, and provides information about the estimation progress. After the estimation completes, the Control and Estimation Tools Manager GUI looks similar to the following figure.



Iteration	Function Count	Cost Function	Step Size	Procedure
0	1	1.8596e+006	1	
1	2	7.6008e+005	10	
2	3	1.146e+005	20	
3	4	9780.4	42.86	

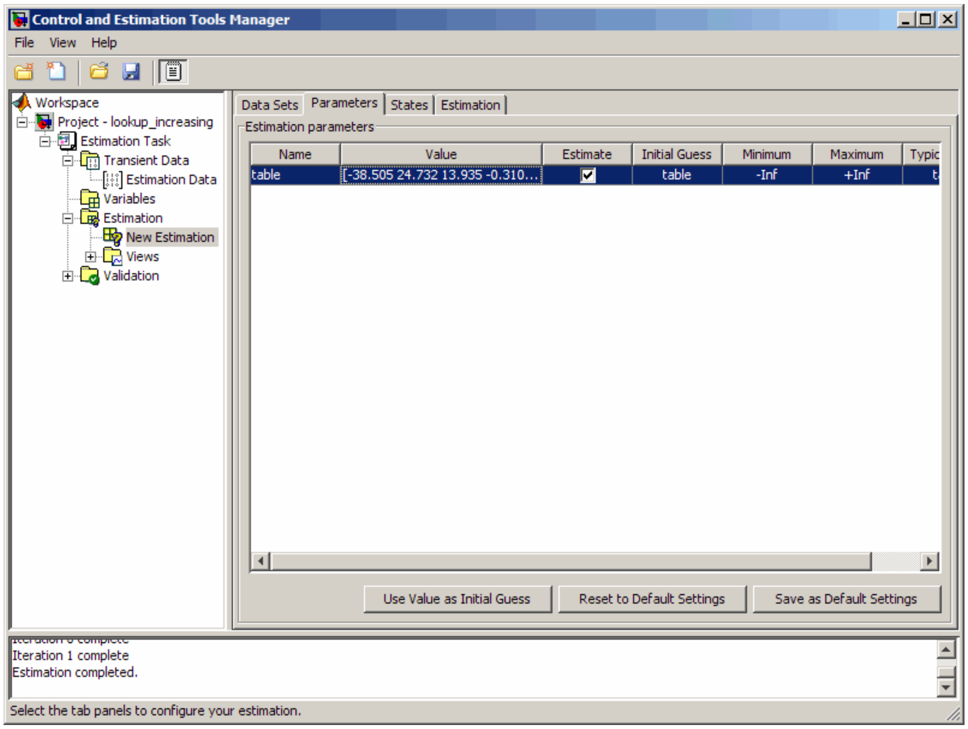
Performing transient estimation...
Active experiments: Estimation Data
Estimated parameters: table

Optimization terminated: first-order optimality less than OPTIONS.TolFun, and no negative/zero curvature detected in trust region model.

Iteration 3 complete
Estimation completed.

Select the tab panels to configure your estimation.

b Select the **Parameters** tab of the **New Estimation** node to view the estimated table values. The **Value** field displays the estimated table values.



Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typic
table	[-38.505 24.732 13.935 -0.310...	✓	table	-Inf	+Inf	t

```
Iteration 1 complete
Estimation completed.
```

Select the tab panels to configure your estimation.

Validating the Estimation Results

After you estimate the table values, as described in "Estimating the Monotonically Increasing Table Values Using Default Settings" on page 5-24, you must use another data set to validate that you have not overfitted the model. You plot and examine the following plots to validate the estimation results:

- Residuals plot
- Measured and simulated data plots

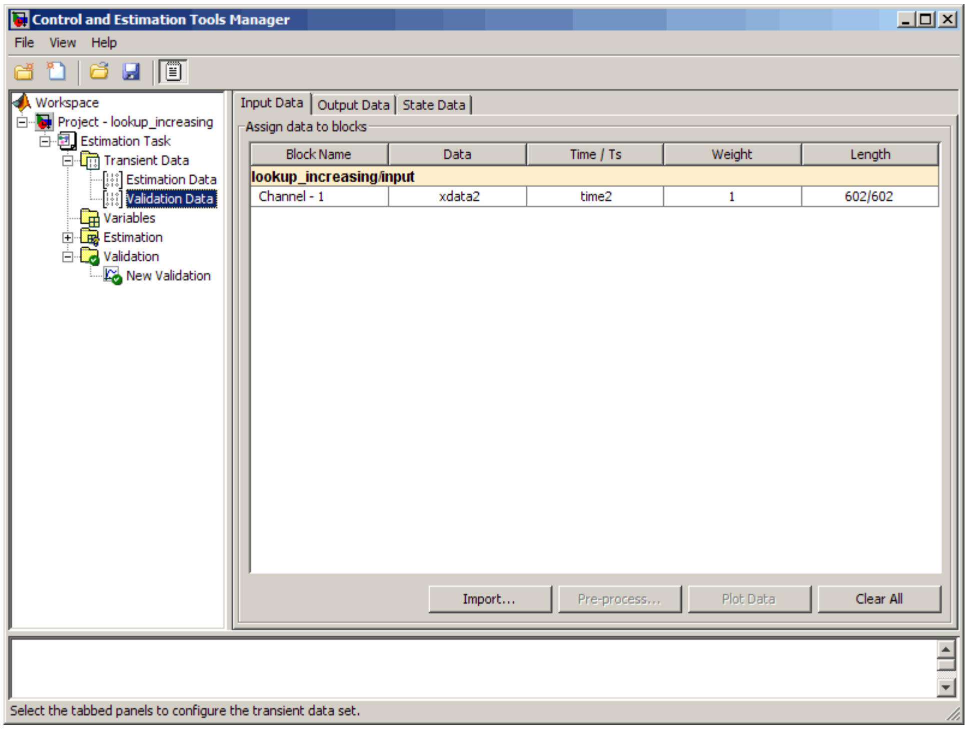
To validate the estimation results:

1 Import the validation data set in the Control and Estimation Tools Manager GUI, as described in “Importing Data into the GUI” section of “Tutorial — Preparing Data for Parameter Estimation Using the GUI”.

The validation data contains the input data, output data and time vector in the MATLAB variables `xdata2`, `ydata2` and `time2` respectively.

You can load a project that already contains the estimated parameters, validation data set, and residuals plot. To do so, type the following commands at the MATLAB prompt:

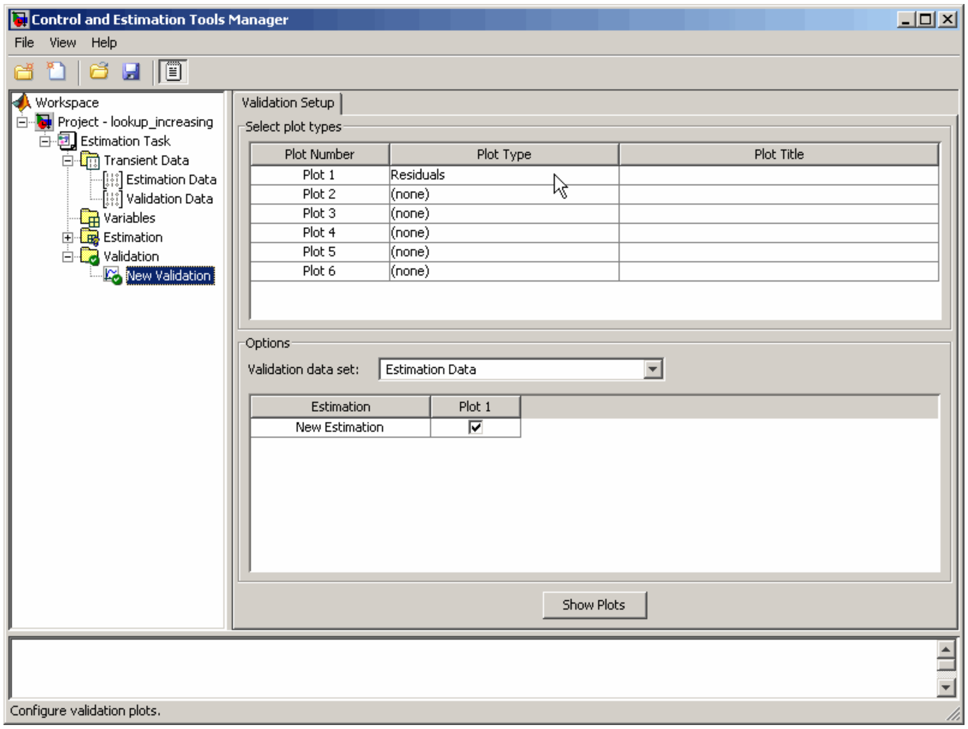
```
lookup_increasing;
speload('lookup_increasing_val', 'Project - lookup_increasing', ...
'Validation Data')
```



Block Name	Data	Time / Ts	Weight	Length
lookup_increasing/input				
Channel - 1	xdata2	time2	1	602/602

Select the tabbed panels to configure the transient data set.

This project also contains the `Residuals` plot already configured in the **Select plot types** area of the GUI, as shown in the next figure.



Plot Number	Plot Type	Plot Title
Plot 1	Residuals	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

Validation data set: Estimation Data

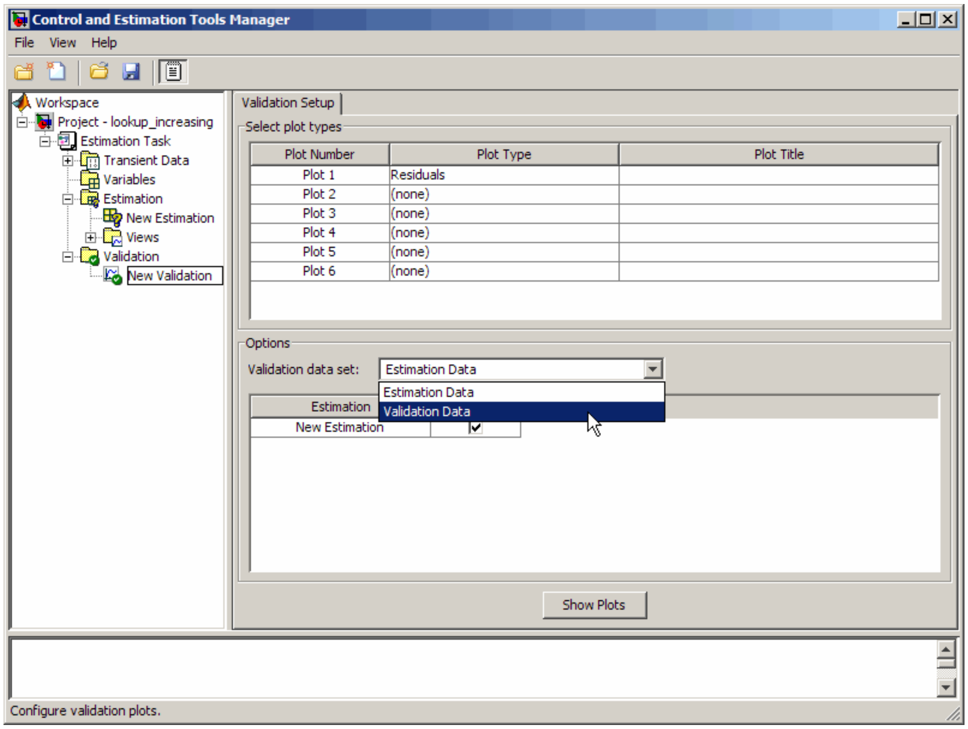
Estimation	Plot 1
New Estimation	✓

Configure validation plots.

2 Plot and examine the residuals.

a Select the **New Validation** node under the **Validation** node.

b In the **Options** area, select `Validation Data` from the **Validation data set** drop-down list.



Plot Number	Plot Type	Plot Title
Plot 1	Residuals	
Plot 2	(none)	
Plot 3	(none)	
Plot 4	(none)	
Plot 5	(none)	
Plot 6	(none)	

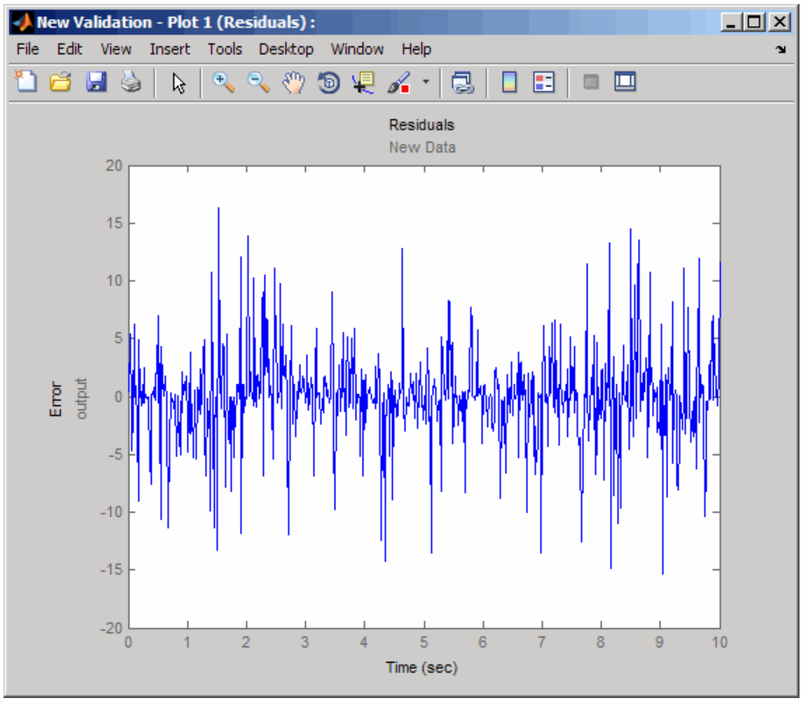
Validation data set: Estimation Data

Estimation Data
Validation Data

Estimation	
New Estimation	✓

Configure validation plots.

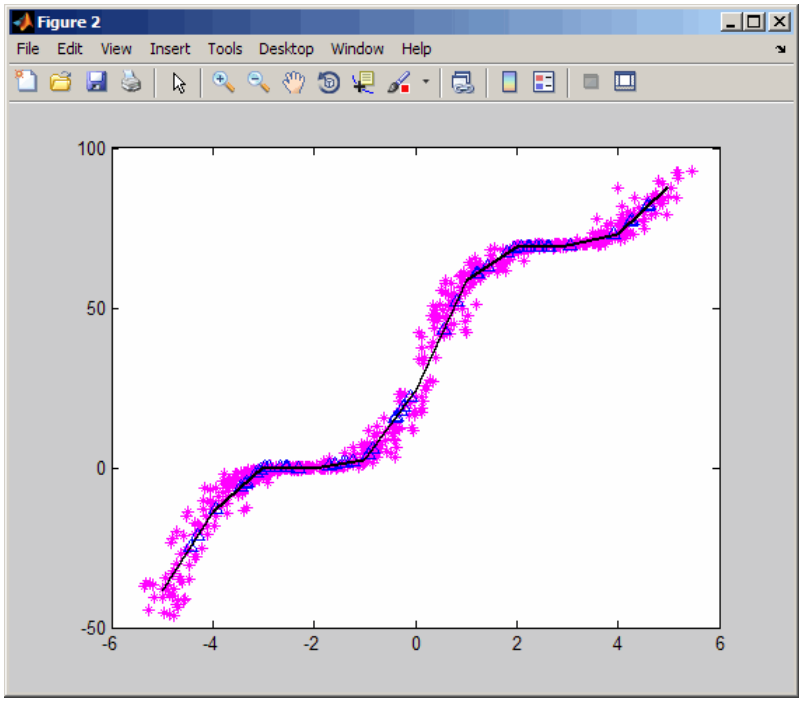
c Click **Show Plots** to open the residuals plot.


New Validation - Plot 1 (Residuals) :
File Edit View Insert Tools Desktop Window Help
Residuals
New Data
Error
output
Time (sec)
20
15
10
5
0
-5
-10
-15
-20
0
1
2
3
4
5
6
7
8
9
10

The residuals, which show the difference between the simulated and measured data, lie the range [-15,15]— within 20% of the maximum output variation. This indicates a good match between the measured and the simulated table data values.

3 Plot and examine the validation data, simulated data and estimated table values.

```
sim('lookup_increasing')
figure(2); plot(xdata2,ydata2, 'm*', xout, yout,'b^')
hold on; plot(-5:5, cumsum(table), 'k', 'LineWidth', 2)
```



The plot shows that the table values, shown as the black line, match both the measured data and the simulated table values. The table data values cover the entire range of input values, which indicates that all the lookup table values have been estimated.

Capturing Time-Varying System Behavior Using Adaptive Lookup Tables

In this section...

Building Models Using Adaptive Lookup Table Blocks

You can use the adaptive lookup table blocks in Simulink Design Optimization software to create lookup tables from measured or simulated data. For more information, see "Adaptive Lookup Tables" on page 5-3.

The Adaptive Lookup Table library has the following three blocks:

- Adaptive Lookup Table (1D Stair-Fit) — One-dimensional adaptive lookup table
- Adaptive Lookup Table (2D Stair-Fit) — Two-dimensional adaptive lookup table
- Adaptive Lookup Table (nD Stair-Fit) — Multidimensional adaptive lookup table

---

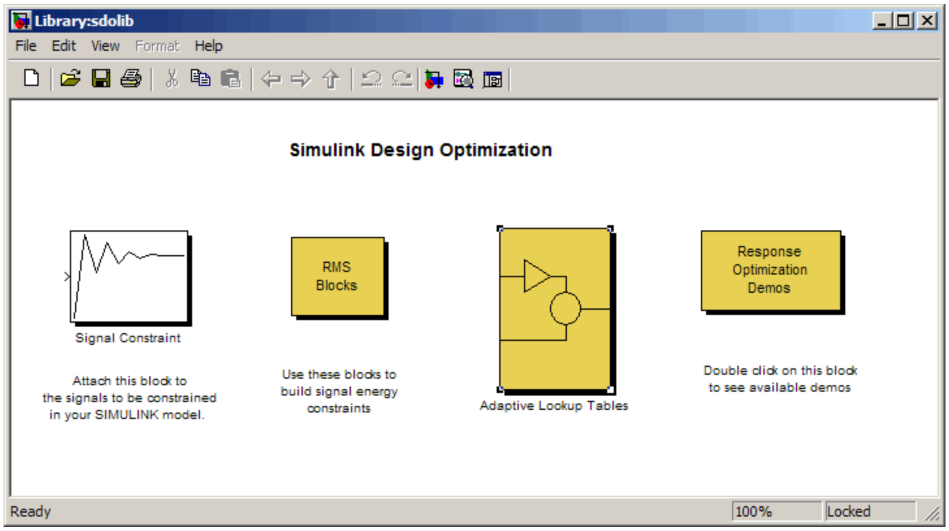
Note Use the n-D Adaptive Lookup Table block to create lookup tables of three or more dimensions.

To access the Adaptive Lookup Tables library:

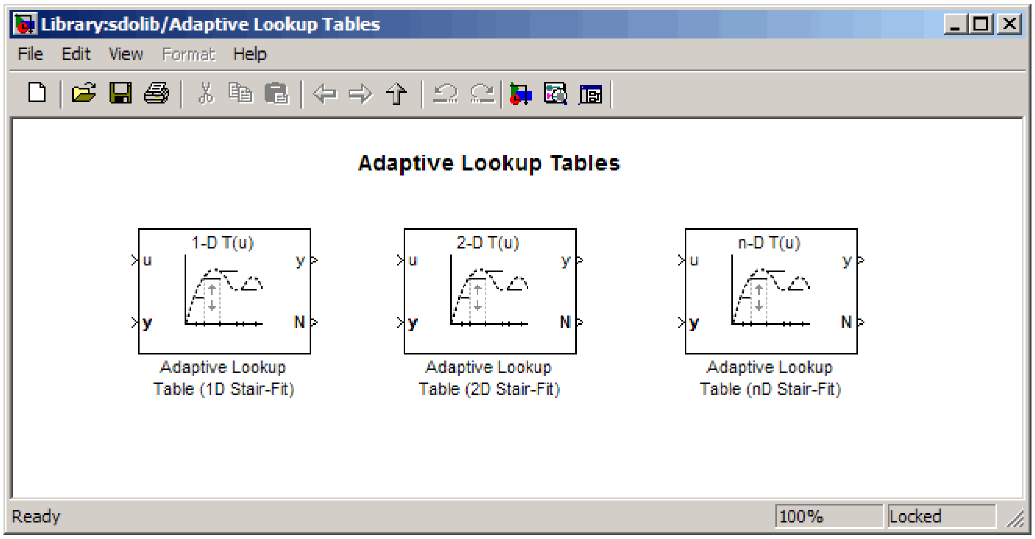
1 Type the following command at the MATLAB prompt:

```
sdolib
```

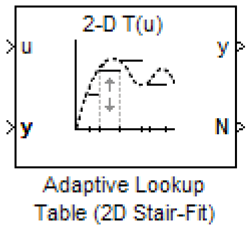
The Simulink Design Optimization library opens as shown in the next figure.


Library:sdolib
File Edit View Format Help
Simulink Design Optimization
Signal Constraint
Attach this block to the signals to be constrained in your SIMULINK model.
RMS Blocks
Use these blocks to build signal energy constraints
Adaptive Lookup Tables
Response Optimization Demos
Double click on this block to see available demos
Ready
100%
Locked

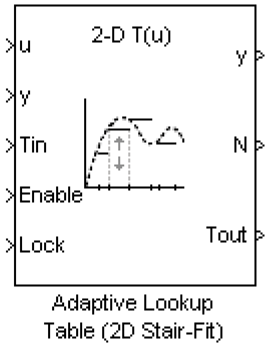
2 Double-click the Adaptive Lookup Tables block to open the Adaptive Lookup Tables library, as shown in the next figure.


Library:sdolib/Adaptive Lookup Tables
File Edit View Format Help
Adaptive Lookup Tables
1-D T(u)
u
y
y
N
Adaptive Lookup Table (1D Stair-Fit)
2-D T(u)
Adaptive Lookup Table (2D Stair-Fit)
n-D T(u)
Adaptive Lookup Table (nD Stair-Fit)
Ready
100%
Locked

By default, the Adaptive Lookup Table blocks have two inputs and outputs as shown in the next figure.



You can display additional inputs and outputs in a block by selecting the corresponding options in the Function Block Parameters dialog box. To learn more about the options, see Chapter 8, “Block Reference”.



Adaptive Lookup Table Block Showing Inputs and Outputs

The 2-D Adaptive Lookup Table block has the following inputs and outputs:

- `u` and `y` — Input and output data of the system being modeled, respectively

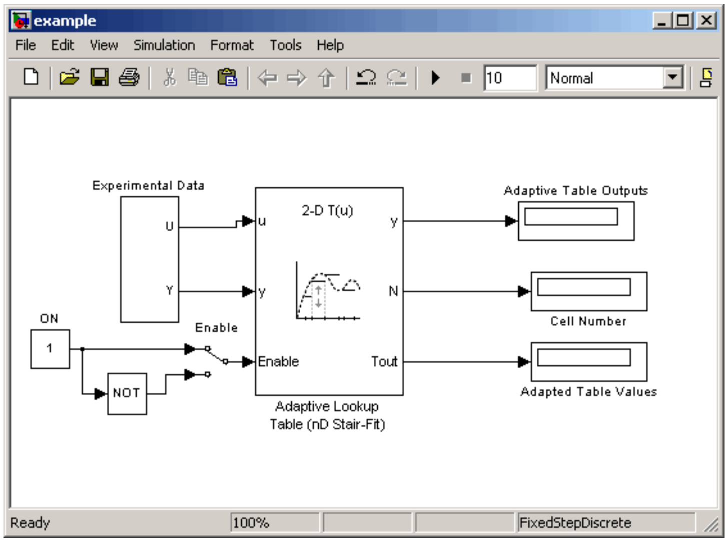
  For example, to model an engine’s efficiency as a function of engine rpm and manifold pressure, specify `u` as the rpm, `y` as the pressure, and `y` as the efficiency signals.

- `Tin` — The initial table data
- `Enable` — Signal to enable, disable, or reset the adaptation process

- `Lock` — Signal to update only specified cells in the table
- `y` — Value of the cell currently being adapted
- `N` — Number of the cell currently being adapted
- `Tout` — Values of the adapted table data

For more information on how to use adaptive lookup tables, see "Tutorial — Modeling a System Using Adaptive Lookup Table".

A typical Simulink diagram using an adaptive lookup table block is shown in the next figure.



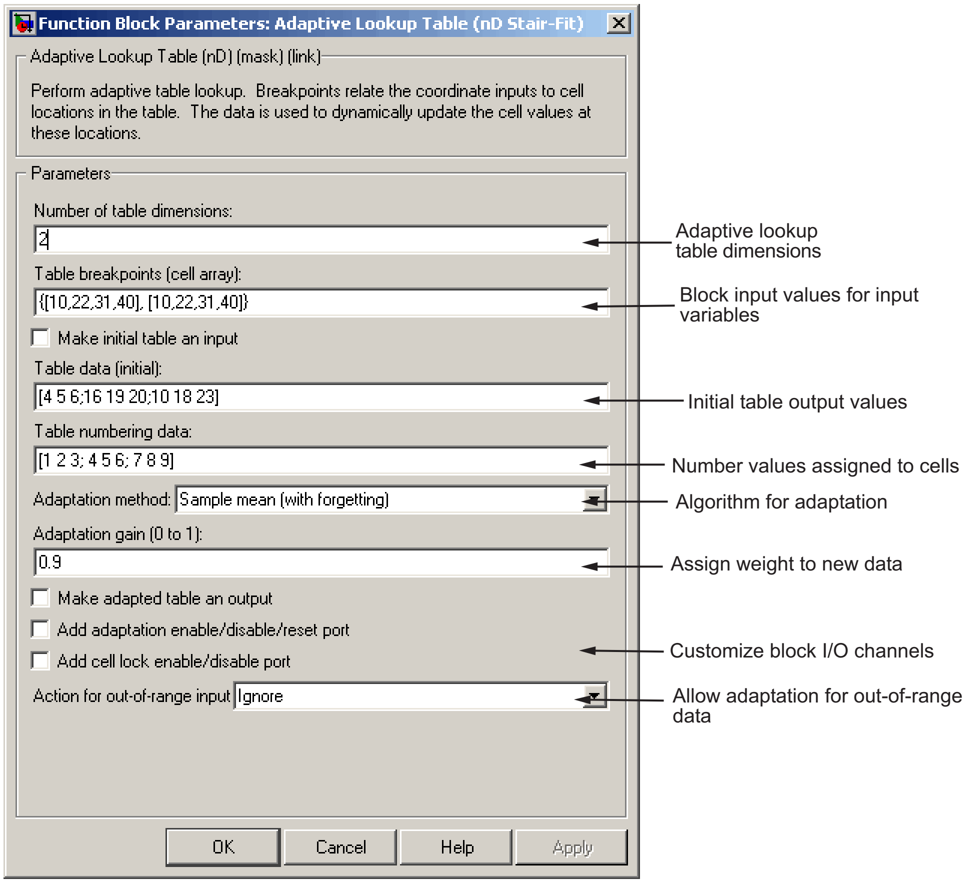
Simulink® Diagram Using an Adaptive Lookup Table

In this figure, the Experiment Data block imports a set of experimental data into Simulink through MATLAB workspace variables. The initial table is specified in the block mask parameters. When the simulation runs, the initial table begins to adapt to new data inputs and the resulting table is copied to the block's output.

Configuring Adaptive Lookup Table Blocks

Setting Adaptive Lookup Table Parameters

You can configure the Adaptive Lookup Table parameters in the Function Block Parameters dialog box. Double-click the block to open the dialog box shown in the next figure.



Function Block Parameters: Adaptive Lookup Table (nD Stair-Fit)

Adaptive Lookup Table (nD) (mask) (link)

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters

Number of table dimensions: 2

Table breakpoints (cell array): {[10,22,31,40], [10,22,31,40]}

☐ Make initial table an input

Table data (initial): [4 5 6;16 19 20;10 18 23]

Table numbering data: [1 2 3; 4 5 6; 7 8 9]

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input: Ignore

OK
Cancel
Help
Apply
Adaptive lookup table dimensions
Block input values for input variables
Initial table output values
Number values assigned to cells
Algorithm for adaptation
Assign weight to new data
Customize block I/O channels
Allow adaptation for out-of-range data

n-D Adaptive Lookup Table Dialog Box

For details on how to set these parameters, see the individual Chapter 8, “Block Reference” pages.

Selecting an Adaptation Method

You can select an adaptation algorithm from the **Adaptation Method** drop-down list in the Function Block Parameters dialog box. This section discusses the details of these algorithms.

Sample Mean. `Sample mean` provides the average value of n output data samples and is defined as:

$$\hat{y}(n) = \frac{1}{n}\sum_{i=1}^{n} y(i)$$

where *y(i)* is the i^{th} measurement collected within a particular *cell*. For each input data u, the sample mean at the corresponding cell is updated using the output data measurement, y. Instead of accumulating n samples of data for each cell, a recursive relation is used to calculate the sample mean. The recursive expression is obtained by the following equation:

$$\hat{y}(n) = \frac{1}{n}\left[\sum_{i=1}^{n-1} y(i) + y(n)\right] = \frac{n-1}{n}\left[\frac{1}{n-1}\sum_{i=1}^{n-1} y(i)\right] + \frac{1}{n}y(n) = \frac{n-1}{n}\hat{y}(n-1) + \frac{1}{n}y(n)$$

where *y(n)* is the n^{th} data sample.

Defining *a priori estimation error* as $e(n) = y(n) - \hat{y}(n-1)$, the recursive relation can be written as:

$$\hat{y}(n) = \hat{y}(n-1) + \frac{1}{n}e(n)$$

where $n \geq 1$ and the initial estimate $\hat{y}(0)$ is arbitrary.

In this expression, only the number of samples, n, for each cell— rather than n data samples—is stored in memory.

Sample Mean with Forgetting. The adaptation method "Sample Mean" on page 5-42 has an *infinite memory*. The past data samples have the same weight as the final sample in calculating the sample mean. `Sample mean (with forgetting)` uses an algorithm with a *forgetting factor* or **Adaptation gain** that puts more weight on the more recent samples. This algorithm provides robustness against initial response transients of the plant and an adjustable speed of adaptation. `Sample mean (with forgetting)` is defined as:

$$\begin{aligned}\hat{y}(n) &= \frac{1}{\sum_{i=1}^{n}\lambda^{n-i}}\sum_{i=1}^{n}\lambda^{n-i}y(i)\\ &= \frac{1}{\sum_{i=1}^{n}\lambda^{n-i}}\left[\sum_{i=1}^{n-1}\lambda^{n-i}y(i)+y(n)\right] = \frac{s(n-1)}{s(n)}\hat{y}(n-1)+\frac{1}{s(n)}y(n)\end{aligned}$$

where $\lambda \in [0,1]$ is the **Adaptation gain** and $s(k) = \sum_{i=1}^{k}\lambda^{n-i}$.

Defining *a priori estimation error* as $e(n) = y(n) - \hat{y}(n-1)$, where $n \geq 1$ and the initial estimate $\hat{y}(0)$ is arbitrary, the recursive relation can be written as:

$$\hat{y}(n) = \hat{y}(n-1) + \frac{1}{s(n)}e(n) = \hat{y}(n-1) + \frac{1-\lambda}{1-\lambda^{n}}e(n)$$

A small value of λ results in faster adaptation. A value of `0` indicates short memory (last data becomes the table value), and a value of `1` indicates long memory (average all data received in a cell).

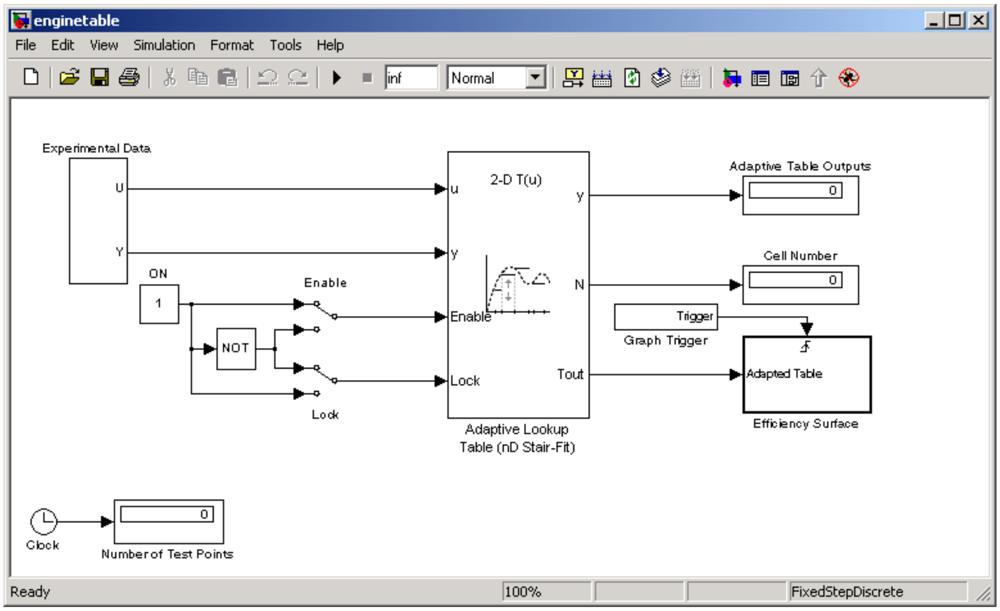
Example: Modeling an Engine Using n-D Adaptive Lookup Table

- "Loading the Example" on page 5-44
- "Running the Example" on page 5-45

Loading the Example

This example shows an n-D adaptive lookup table at work and includes many of the key features associated with adaptive lookup tables. Type the following command at the MATLAB prompt to open this model:

```
enginetable
```



This model has several key features:

- Input — The adaptive lookup table input is the experimental data. It is also possible to make the original table itself an input.
- An enable feature — You can turn the adaptation on and off during the estimation to see how the basic features work.
- A lock feature — You can lock the table so that only one cell is adapting. You may find this feature useful if you have one section in your data that is highly erratic or otherwise difficult for the algorithm to handle.
- Output — Adaptive lookup table values.

Running the Example

To start the `enginetable` simulation, select **Simulation > Start**. Alternatively, you can click the **Start Simulation** button ▶ in the Simulink toolbar.

The simulation begins by populating the adaptive lookup table with random data. This figure shows the input and adapted data side by side.

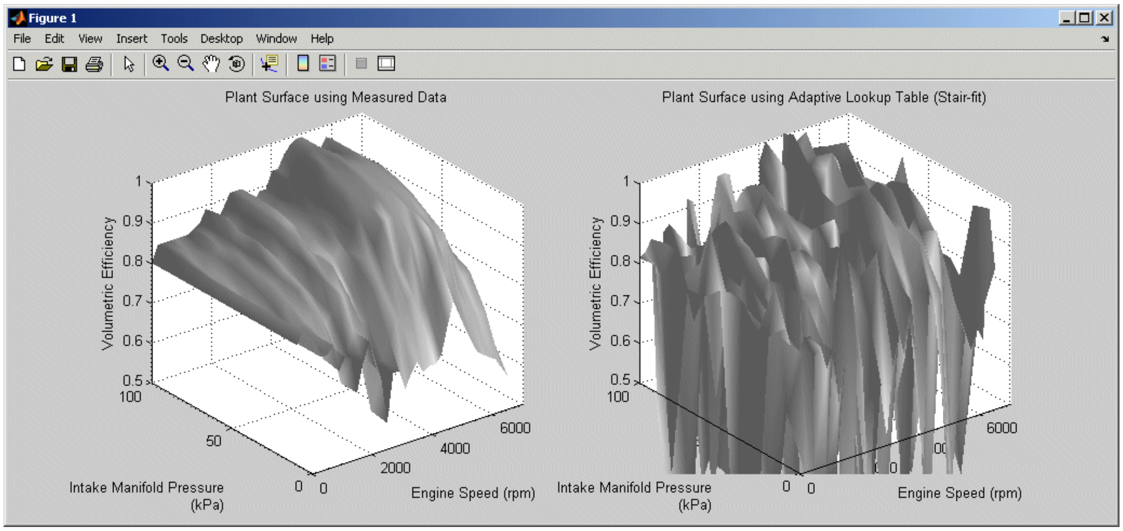
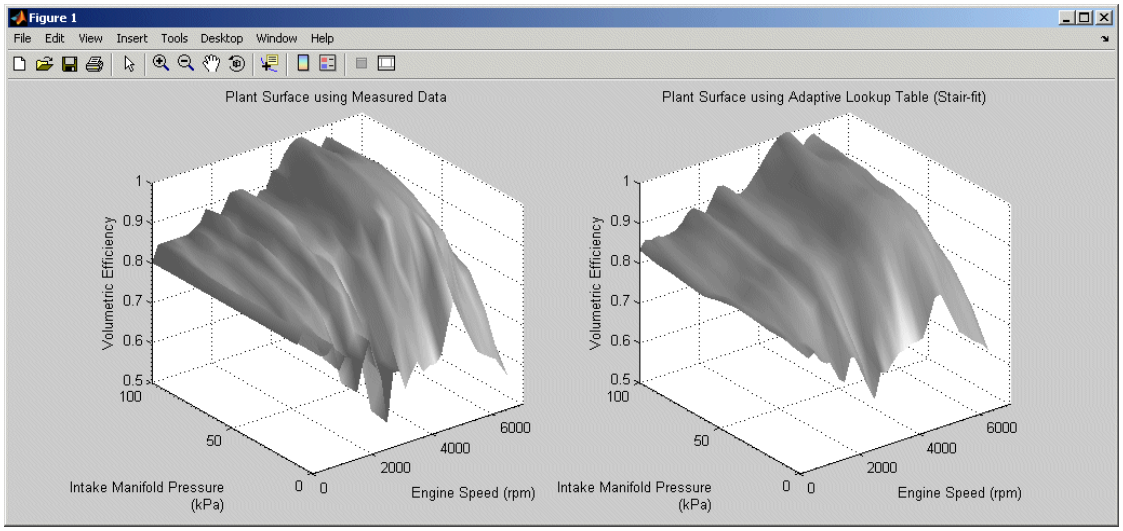


Figure 1
File Edit View Insert Tools Desktop Window Help
Plant Surface using Measured Data
Volumetric Efficiency
1
0.9
0.8
0.7
0.6
0.5
100
50
0
Intake Manifold Pressure (kPa)
0
2000
4000
6000
Engine Speed (rpm)
Plant Surface using Adaptive Lookup Table (Stair-fit)
Volumetric Efficiency
Intake Manifold Pressure (kPa)
Engine Speed (rpm)

As the simulation progresses, the surface on the right adapts to match the measured input data. The next figure shows the final adaptation.



The simulation indicates a very good fit. Next, try using the enable and lock features to see how they change the adaptation.

Using Adaptive Lookup Tables in Real-Time Environment

You can use experimental data from sensor measurements collected by running various tests on a system in real time. The measured data is then sent to the adaptive table block to generate a lookup table describing the relation between the system inputs and output.

You can also use the Adaptive Lookup Table block in a real-time environment, where some time-varying properties of a system need to be captured. To do so, generate C code using Real-Time Workshop® code generation software that can then be run in an xPC Target™ or dSPACE® software. Because you can start, stop, or reset the adaptation if you want, use logic to enable the adaptation of the table data only when it is desired. The cell number output `N`, and the `Enable` and `Lock` inputs facilitate this process. Use the `Enable` input to start and stop the adaptation and the `Lock` input to update only one of the table cells. The `Lock` input combined with some logic using the cell number output `N` provide the means for updating only the desired table cells during a simulation run.

Function Reference

Parameter Estimation

`spetool`	Create Estimation Task in Control and Estimation Tools Manager GUI

Parameter Optimization

Response Optimization Projects (p. 6-3)	Work with response optimization projects
Design Requirements (p. 6-3)	Manage design requirements
Parameters (p. 6-4)	Manage parameters to optimize
Model Robustness (p. 6-4)	Specify uncertain parameter values
Optimization Options (p. 6-4)	View and modify optimization settings
Simulation Options (p. 6-4)	View and modify simulation settings

Response Optimization Projects

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

Design Requirements

`findconstr`	Find constraints defined in Signal Constraint block

Parameters

`findpar`	Find specifications for given tuned parameter
`initpar`	Initialize tuned parameters in response optimization project

Model Robustness

`gridunc`	Construct N-D grid of uncertain parameter values
`randunc`	Randomly sample uncertain parameters
`setunc`	Specify parameter uncertainty in response optimization project

Optimization Options

`optimget`	Current optimizer settings
`optimset`	Modify optimization settings

Simulation Options

`simget`	Current simulation settings
`simset`	Modify simulation settings

Functions — Alphabetical List

findconstr

Purpose

Find constraints defined in Signal Constraint block

Syntax

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

Description

`constraints=findconstr(proj,'blockname')` returns a constraint object for the Signal Constraint block, `blockname`, used in the response optimization project, `proj`. This object contains the data defining the design requirements. Design requirements include the positions of the constraint bound segments and reference signals specified in the Signal Constraint block. 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 a signal. These properties define the amplitude and time for the beginning and ending points of each constraint segment.

- Use the `LowerBoundY` and `UpperBoundY` properties to specify the amplitude of the lower- and the upper-constraint bound segments, respectively.
- Use the `LowerBoundX`, and `UpperBoundX` properties to specify the time segments corresponding to the `LowerBoundY` and `UpperBoundY` amplitude properties, respectively.

  When you specify the time values for the constraint bound segments, make sure that two consecutive time values do not overlap or have gaps between them.

Modify the constraint object properties `ReferenceX` and `ReferenceY` to specify a new reference signal to track. These properties contain the time and amplitude vectors of the reference signal, respectively.

Note The Signal Constraint block does not update to display the modified constraints and reference signal. However, the updated constraint bounds and reference signal specified in the constraint object are used when you optimize the parameters from the command line.

Example

Open the model `srotut1` by typing the model name at the MATLAB prompt:

```
srotut1
```

Create a response optimization project:

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

Find the constraint object for this project:

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

This command returns the following result:

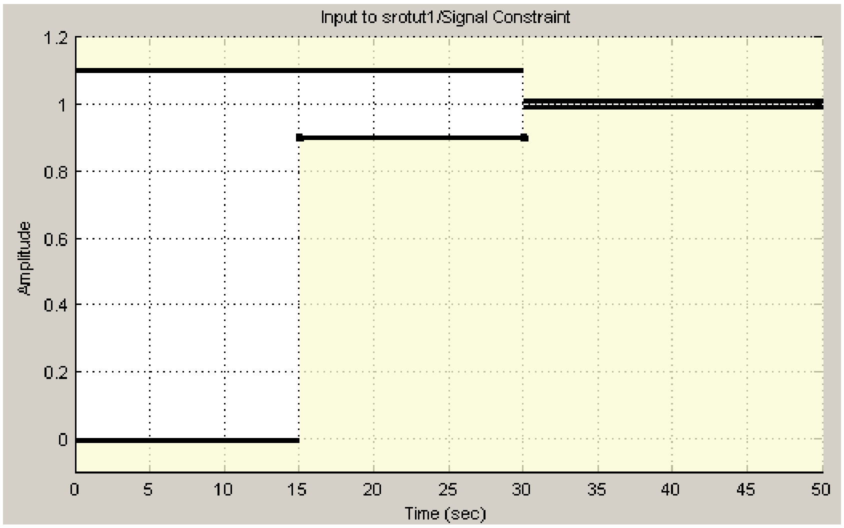
```
         ConstrEnable: 'on'
           isFeasible: 1
           CostEnable: 'off'
               Enable: 'on'
                 Name: 'Signal Constraint'
           SignalSize: [1 1]
          LowerBoundX: [3x2 double]
          LowerBoundY: [3x2 double]
     LowerBoundWeight: [3x1 double]
          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.UpperBoundX=[0 30;30 50];
constraint.LowerBoundY=[0 0;0.9 0.9;0.99 0.99];
constraint.LowerBoundX=[0 15;15 30;30 50];
```

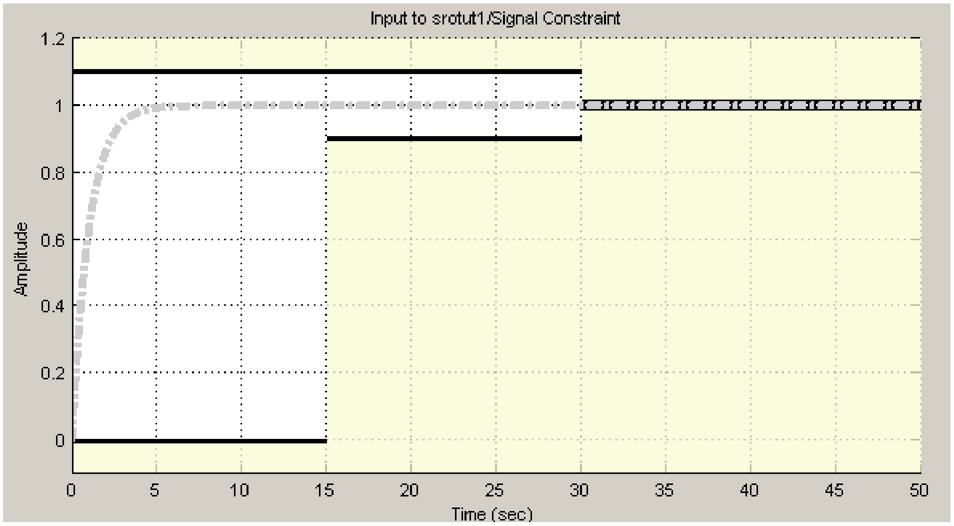
If you add these constraints graphically in the Signal Constraint block, the constraints appear as shown in the following figure. To learn more about how to add constraints graphically, see “Specifying Design Requirements” on page 3-5.



Include a reference signal using the following commands:

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

If you include the reference signal graphically in the Signal Constraint block, the reference signal appears as shown in the following figure. To learn more about how to specify a reference signal, see “Tracking Reference Signals” on page 3-17.



See Also

`getsro`, `newsro`, `optimize`

finddepend

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 3-67 in the *Simulink Design 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 command returns the following result:

```
              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

Extract 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 Design 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 the model name at the MATLAB prompt:

```
pidtune_demo
```

Extract the response optimization project from this model by typing the following command at the MATLAB prompt:

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

This command returns the following result:

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

Response Optimization Project.
```

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

See Also

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

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`

Purpose Initialize tuned parameters in response optimization project

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 currently in the model or base workspace.

Example Open the model `srotut1` by typing the model name at the MATLAB prompt:

```
srotut1
```

Create a response optimization project:

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

The `InitialGuess` property of the tuned parameter `Kint` defaults to its current value in the MATLAB workspace. View the `InitialGuess` property value by typing the following command:

```
proj.parameter.InitialGuess
```

This command returns the following result:

```
ans =

      0
```

Set the value of `Kint` to `1`, and then set the `InitialGuess` property of `Kint` using the `initpar` command:

```
Kint=2;
initpar(proj);
```

View the value of the `InitialGuess` property of the tuned parameter `Kint`:

```
proj.parameter.InitialGuess
```

This command returns the following result:

```
ans =

     2
```

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 Design Optimization library. The model must be open prior to calling `ncdupdate`. Nonlinear Control Design Blockset software is the version of Simulink Design 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 Design 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 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, `params`. The specified model should contain at least one block from Simulink Design Optimization library. Type `sdolib` 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 sroengine.OptimOptions]
           Tests: [1x1 ResponseOptimizer.SimTest]
           Model: 'pidtune_demo'

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
```

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: • `fmincon` — Optimization Toolbox function `fmincon` • `patternsearch` — Genetic Algorithm and Direct Search Toolbox function `patternsearch` • `fminsearch` — Optimization Toolbox function `fminsearch`	`{'fmincon'} | 'patternsearch' | 'fminsearch'`
`Display`	The level of information that the optimization displays: • `off` — No output • `iter` — Output at each iteration • `final` — Final output only • `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: • `basic` — Default method for computing the gradients • `refined` — Offers a more robust and less noisy gradient calculation method than `'basic'` The `refined` method is sometimes more expensive, and does not work with certain models such as SimPowerSystems models.	`{'basic'} | 'refined'`
`MaximallyFeasible`	Option to specify that the optimization algorithm continue after an initial solution has been found: • `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. • `1` — Continue the optimization after an initial solution is found. The optimization can continue	`{0} | 1`

Property	Description	Possible Settings
	to search for a maximally feasible solution that is typically located further inside the constraint region.	
`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
`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: • `fmincon` • `patternsearch` **Note** Parallel Computing Toolbox software must be installed to enable parallel computing for the optimization algorithms. When set to `'always'`, the algorithms compute the following in parallel: • `fmincon` — Computes finite difference gradients • `patternsearch` — Performs population evaluation 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: []
```

Purpose

Randomly sample uncertain parameters

Syntax

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

See Also

`optimget`, `simget`, `simset`

Description

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

`uset` is the uncertain parameters object. `N` is the number of samples for each uncertain parameter. `P1, P2, ...` are the uncertain parameters in a response optimization project. `Range1, Range2, ...` specify the lower and upper bounds for the corresponding uncertain parameter value.

For a scalar-valued parameter, `p`, specify the range as `[Min,Max]` or `{Min,Max}`. The software interprets the range of the uncertain parameter as:

```
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 software interprets the range of the uncertain parameters as:

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

The set of uncertain parameter values consists of:

- All vertices of the hypercube specified by the `Min` and `Max` values of the uncertain parameters. The total number of vertices of the hypercube is 2^S, where S is the number of uncertain parameters.
- `N` random samples inside the hypercube.

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
    3.4442
    3.7174
    1.3810
```

```
        3.7401
```

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

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

See Also

`gridunc`, `setunc`

Purpose

Specify parameter uncertainty in response 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 `simset` function.

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

Purpose

Create Estimation Task in Control and Estimation Tools Manager GUI

Syntax

```
spetool('modelname')
```

Description

`spetool('modelname')` opens the Simulink model with the name `modelname` and creates an estimation task in the Control and Estimation Tools Manager GUI.

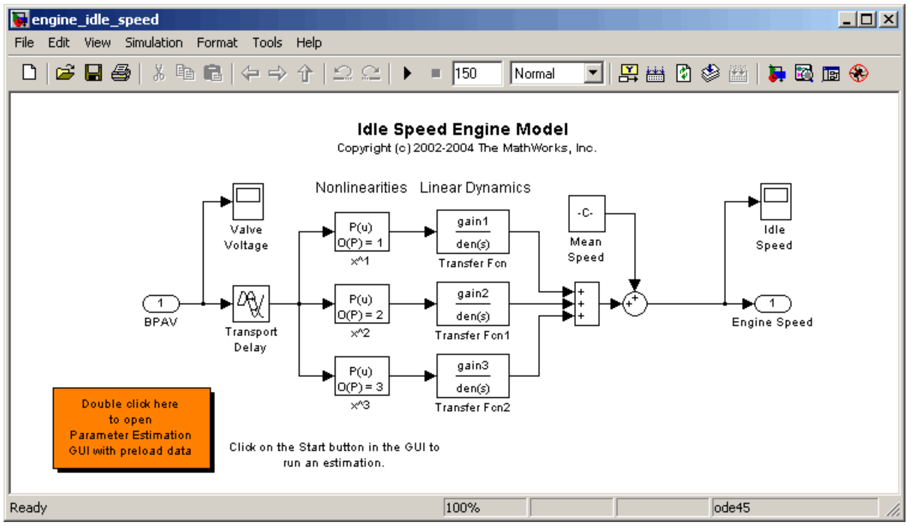
Example

Create an estimation task by typing the following command at the MATLAB prompt:

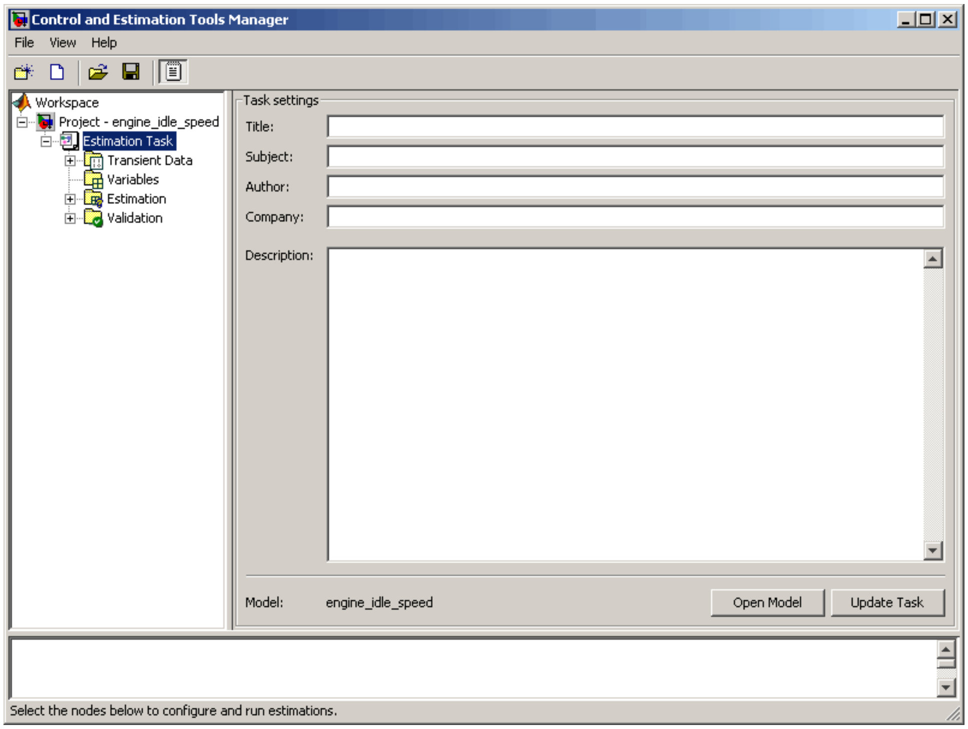
```
spetool('engine_idle_speed')
```

This command opens the following:

- Simulink model



- Control and Estimation Tools Manager containing a project with an estimation task



Task settings

Title:

Subject:

Author:

Company:

Description:

Model: engine_idle_speed

Select the nodes below to configure and run estimations.

See Also

"Importing Data into the GUI" on page 1-5, "Configuring Parameter Estimation in the GUI" on page 2-3

8

Block Reference

Adaptive Lookup Table (1D Stair-Fit)

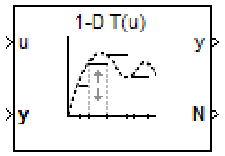
Purpose

Perform one-dimensional adaptive table lookup

Library

Simulink Design Optimization

Description



Adaptive Lookup Table (1D Stair-Fit)

The Adaptive Lookup Table (1D Stair-Fit) block creates a one-dimensional adaptive lookup table by dynamically updating the underlying lookup table. The block uses the outputs, *y*, of your system to do the adaptations.

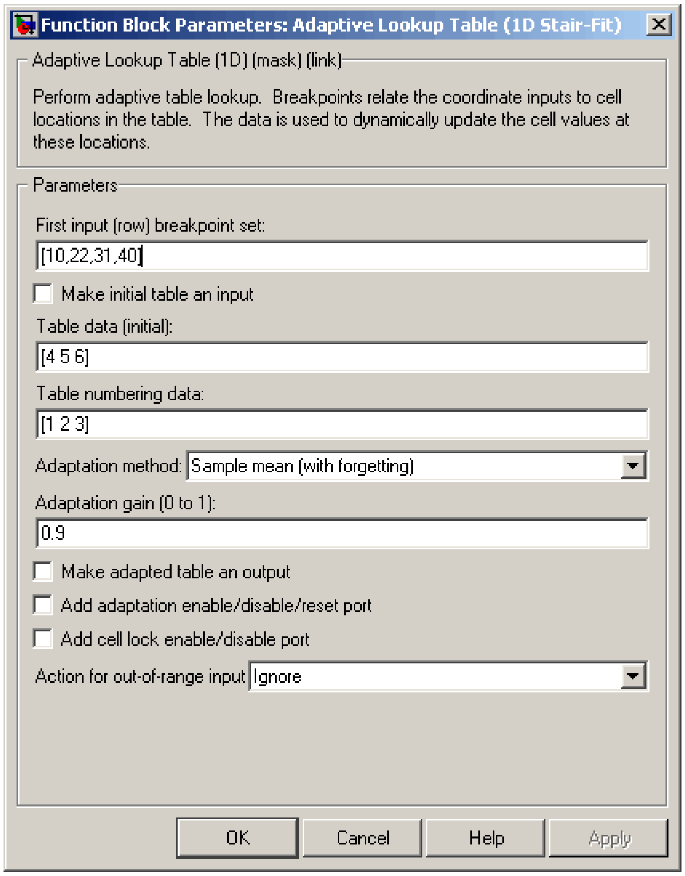
Each indexing parameter *u* may take a value within a set of adapting data points, which are called *breakpoints*. Two breakpoints in each dimension define a *cell*. The set of all breakpoints in one of the dimensions defines a *grid*. In the one-dimensional case, each cell has two breakpoints, and the cell is a line segment.

You can use the Adaptive Lookup Table (1D Stair Fit) block to model time-varying systems with one input.

Data Type Support

Doubles only

Dialog Box


Function Block Parameters: Adaptive Lookup Table (1D Stair-Fit)

Adaptive Lookup Table (1D) (mask) (link)

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters

First input (row) breakpoint set: [10,22,31,40]

☐ Make initial table an input

Table data (initial): [4 5 6]

Table numbering data: [1 2 3]

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input: Ignore

First input (row) breakpoint set
The vector of values containing possible block input values. The input vector must be monotonically increasing.

Make initial table an input
Selecting this check box forces the Adaptive Lookup Table (1D Stair-Fit) block to ignore the **Table data (initial)** parameter, and creates a new input port `Tin`. Use this port to input the table data.

Table data (initial)
: The initial table output values. This vector must be of size N-1, where N is the number of breakpoints.

Table numbering data
: Number values assigned to cells. This vector must be the same size as the table data vector, and each value must be unique.

Adaptation method
: Choose`Sample mean` or `Sample mean (with forgetting)`. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see .

Adaptation gain (0 to 1)
: A number between 0 and 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output
: Selecting this check box creates an additional output port `Tout` for the adapted table.

Add adaptation enable/disable/reset port
: Selecting this check box creates an additional input port `Enable` that enables, disables, or resets the adaptive lookup table. A signal value of `0` applied to the port disables the adaptation, and signal value of `1` enables the adaptation. Setting the signal value to `2` resets the table values to the initial table data.

Add cell lock enable/disable port
: Selecting this check box creates an additional input port `Lock` that provides the means for updating only specified cells during a simulation run. A signal value of `0` unlocks the specified cells and signal value of `1` locks the specified cells.

Action for out-of-range input
: `Ignore` or `Adapt` by extrapolating beyond the extreme breakpoints.

See Also

Adaptive Lookup Table (2D Stair-Fit), Adaptive Lookup Table (nD Stair-Fit), “Capturing Time-Varying System Behavior Using Adaptive Lookup Tables” on page 5-37

Adaptive Lookup Table (2D Stair-Fit)

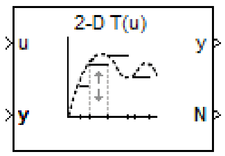
Purpose

Perform two-dimensional adaptive table lookup

Library

Simulink Design Optimization

Description



Adaptive Lookup Table (2D Stair-Fit)

The Adaptive Lookup Table (2D Stair-Fit) block creates a two-dimensional adaptive lookup table by dynamically updating the underlying lookup table. The block uses the outputs, *y*, of your system to do the adaptations.

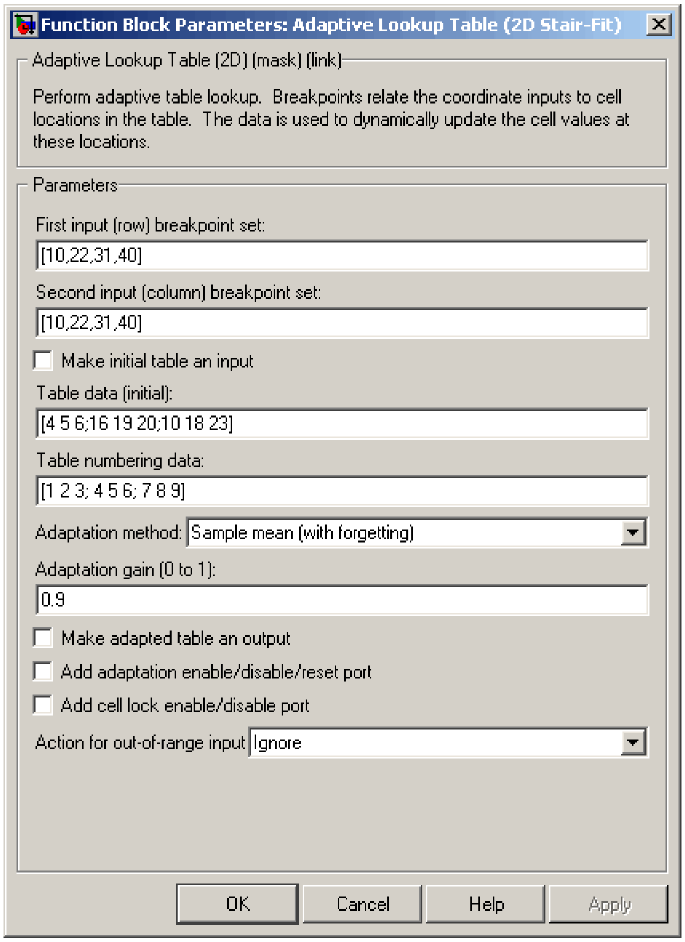
Each indexing parameter *u* may take a value within a set of adapting data points, which are called *breakpoints*. Two breakpoints in each dimension define a *cell*. The set of all breakpoints in one of the dimensions defines a *grid*. In the two-dimensional case, each cell has four breakpoints and is a flat surface.

You can use the Adaptive Lookup Table (2D Stair-Fit) block to model time-varying systems with two inputs.

Data Type Support

Doubles only

Dialog Box


Function Block Parameters: Adaptive Lookup Table (2D Stair-Fit)

Adaptive Lookup Table (2D) (mask) (link)

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters

First input (row) breakpoint set:
[10,22,31,40]

Second input (column) breakpoint set:
[10,22,31,40]

☐ Make initial table an input

Table data (initial):
[4 5 6;16 19 20;10 18 23]

Table numbering data:
[1 2 3; 4 5 6; 7 8 9]

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1):
0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input: Ignore

First input (row) breakpoint set

The vector of values containing possible block input values for the first input variable. The first input vector must be monotonically increasing.

Second input (column) breakpoint set
The vector of values containing possible block input values for the second input variable. The second input vector must be monotonically increasing.

Make initial table an input
Selecting this check box forces the Adaptive Lookup Table (2D Stair-Fit) block to ignore the **Table data (initial)** parameter, and creates a new input port `Tin`. Use this port to input the table data.

Table data (initial)
The initial table output values. This 2-by-2 matrix must be of size (n-1)-by-(m-1), where n is the number of first input breakpoints and m is the number of second input breakpoints.

Table numbering data
Number values assigned to cells. This matrix must be the same size as the table data matrix, and each value must be unique.

Adaptation method
Choose `Sample mean` or `Sample mean with forgetting`. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see .

Adaptation gain (0 to 1)
A number from 0 to 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output
Selecting this check box creates an additional output port `Tout` for the adapted table.

Add adaptation enable/disable/reset port
Selecting this check box creates an additional input port `Enable` that enables, disables, or resets the adaptive lookup table. A signal value of `0` applied to the port disables the adaptation, and

signal value of `1` enables the adaptation. Setting the signal value to `2` resets the table values to the initial table data.

Add cell lock enable/disable port
Selecting this check box creates an additional input port `Lock` that provides the means for updating only specified cells during a simulation run. A signal value of `0` unlocks the specified cells and signal value of `1` locks the specified cells.

Action for out-of-range input
`Ignore` or `Adapt` by extrapolating beyond the extreme breakpoints.

See Also

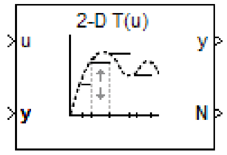
Adaptive Lookup Table (1D Stair-Fit), Adaptive Lookup Table (nD Stair-Fit), "Capturing Time-Varying System Behavior Using Adaptive Lookup Tables" on page 5-37

Adaptive Lookup Table (nD Stair-Fit)

Purpose Create adaptive lookup table of arbitrary dimension

Library Simulink Design Optimization

Description The Adaptive Lookup Table (nD Stair-Fit) block creates an adaptive lookup table of arbitrary dimension by dynamically updating the underlying lookup table. The block uses the outputs of your system to do the adaptations.



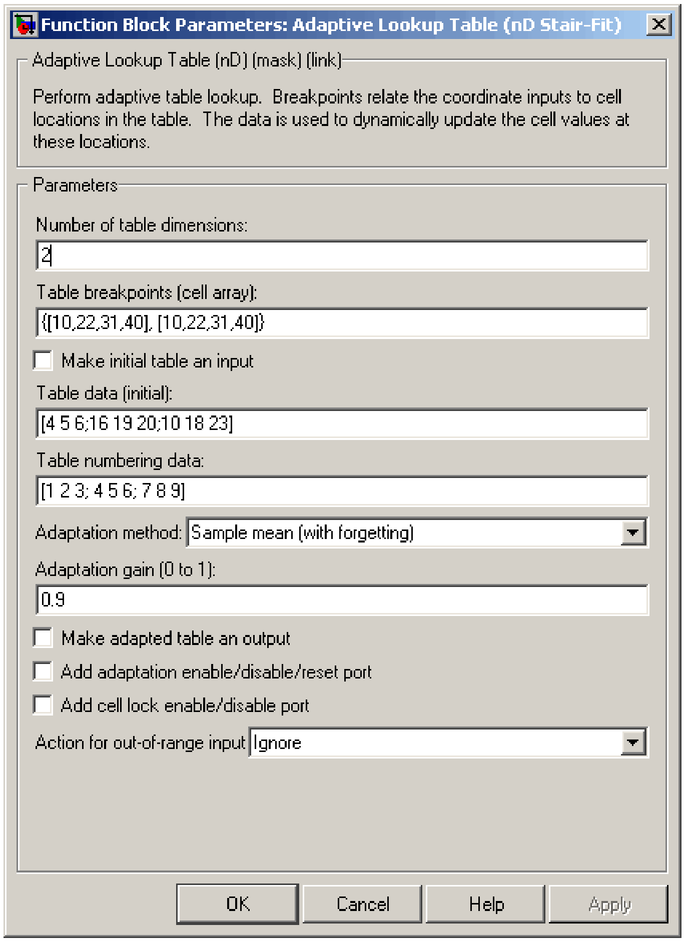
Adaptive Lookup Table (nD Stair-Fit)

Each indexing parameter may take a value within a set of adapting data points, which are called *breakpoints*. Breakpoints in each dimension define a *cell*. The set of all breakpoints in one of the dimensions defines a *grid*. In the n-dimensional case, each cell has two n breakpoints and is an (n-1) hypersurface.

You can use the Adaptive Lookup Table (nD Stair-Fit) block to model time-varying systems with 2 or more inputs.

Data Type Support Doubles only

Dialog Box



Function Block Parameters: Adaptive Lookup Table (nD Stair-Fit)

Adaptive Lookup Table (nD) (mask) (link)

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters

Number of table dimensions: 2

Table breakpoints (cell array): {[10,22,31,40], [10,22,31,40]}

☐ Make initial table an input

Table data (initial): [4 5 6;16 19 20;10 18 23]

Table numbering data: [1 2 3; 4 5 6; 7 8 9]

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input: Ignore

Number of table dimensions

The number of dimensions for the adaptive lookup table.

Table breakpoints (cell array)

A set of one-dimensional vectors that contains possible block input values for the input variables. Each input row must be monotonically increasing, but the rows do not have to be the same length. For example, if the **Number of table dimensions** is `3`, you can set the table breakpoints as follows:

```
{[1 2 3], [5 7], [1 3 5 7]}
```

Make initial table an input
Selecting this check box forces the Adaptive Lookup Table (nD Stair-Fit) block to ignore the **Table data (initial)** parameter, and creates a new input port `Tin`. Use this port to input the table data.

Table data (initial)
The initial table output values. This (n-D) array must be of size (n-1)-by-(n-1) ... -by- (n-1), (D times), where D is the number of dimensions and n is the number of input breakpoints.

Table numbering data
Number values assigned to cells. This vector must be the same size as the table data array, and each value must be unique.

Adaptation method
Choose `Sample mean` or `Sample mean with forgetting`. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see .

Adaptation gain (0 to 1)
A number from 0 to 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output
Selecting this check box creates an additional output port `Tout` for the adapted table.

Note The Adaptive Lookup Table (n-D Stair Fit) block cannot output a table of 3 or more dimensions.

Add adaptation enable/disable/reset port

Selecting this check box creates an additional input port `Enable` that enables, disables, or resets the adaptive lookup table. A signal value of `0` applied to the port disables the adaptation, and signal value of `1` enables the adaptation. Setting the signal value to `2` resets the table values to the initial table data.

Add cell lock enable/disable port

Selecting this check box creates an additional input port `Lock` that provides the means for updating only specified cells during a simulation run. A signal value of `0` unlocks the specified cells and signal value of `1` locks the specified cells.

Action for out-of-range input

`Ignore` or `Adapt` by extrapolating beyond the extreme breakpoints.

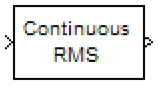
See Also

Adaptive Lookup Table (1D Stair-Fit), Adaptive Lookup Table (2D Stair-Fit), "Capturing Time-Varying System Behavior Using Adaptive Lookup Tables" on page 5-37

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

Library Simulink Design 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.



CRMS

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

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

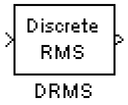
The R.M.S value gives a measure of the average energy in the signal.

See Also DRMS, Signal Constraint

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

Library Simulink Design 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

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

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

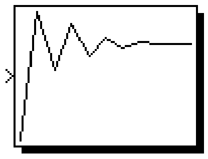
See Also CRMS, Signal Constraint

Signal Constraint

Purpose Specify desired signal response

Library Simulink Design Optimization

Description



Attach a Signal Constraint block to the signal in a Simulink model to optimize the model response to known inputs. Simulink Design 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 more information on how to use this block, see "Configuring Parameter Optimization" on page 3-3.

See Also CRMS, DRMS

Examples

Use this list to find examples in the documentation.

Estimating Parameters and Initial States

Optimizing Model Parameters

Optimization-Based Control Design

Estimating Lookup Table Values

Index