

# Simulink<sup>®</sup> Parameter Estimation

For Use with Simulink<sup>®</sup>

- Modeling
- Simulation
- Implementation

## How to Contact The MathWorks:



www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup



support@mathworks.com	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	Mail
--------------------------------------------------------------------	------

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

### *Simulink Parameter Estimation User's Guide*

© COPYRIGHT 2004-2005 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.

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Revision History:	June 2004	First printing	New for Version 1.0 (Release 14)
	October 2004	Online only	Revised for Version 1.1 (Release 14SP1)
	March 2005	Online only	Revised for Version 1.1.1 (Release 14SP2)

## Getting Started

1

<b>What Is Simulink Parameter Estimation?</b> .....	1-2
<b>What You Need to Get Started</b> .....	1-3
Installation .....	1-3
Prerequisite Software and Optional Software .....	1-3
Required Knowledge .....	1-3
Demos .....	1-3
<b>How Simulink Parameter Estimation Works</b> .....	1-5
Basic Steps in the Estimation Process .....	1-5
Structure of an Estimation Project .....	1-5
<b>Setting Up the Estimation Data</b> .....	1-7
Importing Transient Data .....	1-9
Specifying Initial Conditions .....	1-12
Selecting Parameters for Estimation .....	1-14
Selecting States for Estimation .....	1-15
Initial Guesses and Upper/Lower Bounds .....	1-16
<b>Setting Up an Estimation Project</b> .....	1-18
Adding Data Sets .....	1-18
Specifying and Setting Up Parameters .....	1-20
Opening the Estimation Window .....	1-22
<b>Selecting Views for Plotting</b> .....	1-24
<b>Running the Estimation</b> .....	1-26
<b>Model Validation</b> .....	1-29
Example: Validating the Engine Idle Speed Model .....	1-30
Loading and Importing the Validation Data .....	1-31
Adding the Validation Task .....	1-33
Running the Validation .....	1-35

Residuals .....	1-37
<b>Setting Options for Optimization .....</b>	<b>1-39</b>
Selecting Optimization Methods .....	1-39
Selecting Optimization Termination Options .....	1-40
Selecting Additional Optimization Options .....	1-41
Specifying the Cost Function .....	1-42
<b>Setting Options for the Simulation .....</b>	<b>1-43</b>
Selecting Simulation Time .....	1-43
Selecting Solvers .....	1-44
<b>Estimating Independent Parameters .....</b>	<b>1-46</b>

## Estimating Initial Conditions

### 2

<b>Why Estimate Initial Conditions? .....</b>	<b>2-2</b>
<b>Example: Mass-Spring-Damper System .....</b>	<b>2-3</b>
Model Parameters .....	2-4
Setting Up the Estimation Project .....	2-5
Importing Transient Data and Selecting Parameters for Estimation .....	2-5
Selecting Parameters and Initial Conditions for Estimation ..	2-7
Creating the Estimation Task .....	2-8
Running the Estimation and Viewing Results .....	2-9

## Preprocessing Data

### 3

<b>Why Preprocess Data? .....</b>	<b>3-2</b>
<b>The Data Preprocessing Tool .....</b>	<b>3-3</b>

<b>Excluding Data</b> .....	<b>3-5</b>
Selecting Data for Exclusion from the Data Editing Table . . . .	<b>3-5</b>
Selection Data for Exclusion from a Plot of the Data .....	<b>3-8</b>
Selecting Data for Exclusion by a Rule .....	<b>3-10</b>
<b>Detrending and Filtering</b> .....	<b>3-13</b>
<b>Miscellaneous Data Handling</b> .....	<b>3-15</b>
Handling Missing Data .....	<b>3-15</b>
Loading Data and Saving Modified Data Sets .....	<b>3-15</b>

## Managing Multiple Projects

### 4

<b>Multiple Projects and Tasks</b> .....	<b>4-2</b>
<b>Saving Control and Estimation Tools Manager Projects</b> . .	<b>4-3</b>
<b>Opening Control and Estimation Tools Manager Projects</b> .	<b>4-4</b>

## Adaptive Lookup Tables

### 5

<b>What Are Lookup Tables?</b> .....	<b>5-2</b>
<b>How Adaptive Lookup Tables Work</b> .....	<b>5-3</b>
<b>Implementation of Adaptive Lookup Tables</b> .....	<b>5-4</b>
Adaptive Lookup Table Library .....	<b>5-5</b>
Using Adaptive Lookup Tables in Simulink Models .....	<b>5-6</b>
Real-Time Lookup Tables .....	<b>5-6</b>
Setting Adaptive Lookup Table Parameters .....	<b>5-7</b>

<b>Example: n-D Adaptive Lookup Table</b> .....	<b>5-8</b>
Running the Example .....	<b>5-9</b>

## Estimating from the Command Line

# 6

<b>Introduction</b> .....	<b>6-2</b>
<b>Example: Estimating Parameters and Initial Conditions of the F14 Model</b> .....	<b>6-4</b>
Baseline Simulation .....	<b>6-5</b>
Creating a Transient Experiment Object .....	<b>6-6</b>
Assigning Experimental Data to Inputs and Outputs of the Model .....	<b>6-7</b>
Creating Parameter Objects for Estimation .....	<b>6-7</b>
Creating an Estimation Object and Running the Estimation .....	<b>6-9</b>
<b>Creating and Customizing Estimation Projects</b> .....	<b>6-12</b>
<b>Creating Transient Data Objects</b> .....	<b>6-13</b>
Properties of Transient Data Objects .....	<b>6-13</b>
Modifying Transient Data Object Properties .....	<b>6-16</b>
Using Class Methods .....	<b>6-16</b>
<b>Creating State Data Objects</b> .....	<b>6-17</b>
Properties of the State Data Object .....	<b>6-17</b>
Example: Initial Condition Data .....	<b>6-19</b>
Modifying Properties .....	<b>6-19</b>
Using Class Methods .....	<b>6-19</b>
<b>Creating Transient Experiment Objects</b> .....	<b>6-20</b>
Properties of Transient Experiment Objects .....	<b>6-20</b>
Example: Creating an F14 Experiment .....	<b>6-21</b>
Example: Creating a Van der Pol Experiment from User Objects .....	<b>6-21</b>
Modifying Properties .....	<b>6-22</b>

Using Class Methods .....	6-22
<b>Creating Parameter Objects .....</b>	<b>6-23</b>
Constructor .....	6-23
Properties of Parameter Objects .....	6-23
Example: F14 Model .....	6-25
Example: Gain Matrix .....	6-25
Modifying Properties .....	6-25
Using Class Methods .....	6-26
<b>Creating State Objects .....</b>	<b>6-27</b>
Constructor .....	6-27
Properties of State Objects .....	6-27
Example: F14 Model .....	6-29
Modifying Properties .....	6-29
Using Class Methods .....	6-29
<b>Creating Estimation Objects .....</b>	<b>6-31</b>
Constructor .....	6-31
Properties of Estimation Objects .....	6-31
Example: F14 Model .....	6-32
Modifying Properties .....	6-33
Using Class Methods .....	6-33

## Block Reference

# 7

Adaptive Lookup Table (1D Stair-Fit) .....	7-2
Adaptive Lookup Table (2D Stair-Fit) .....	7-4
Adaptive Lookup Table (nD Stair-Fit) .....	7-7

## Index





# Getting Started

---

What Is Simulink Parameter Estimation? (p. 1-2)	A brief description of the product
What You Need to Get Started (p. 1-3)	Requirements and options for getting started with Simulink Parameter Estimation
How Simulink Parameter Estimation Works (p. 1-5)	How Simulink Parameter Estimation handles the estimation problem
Setting Up the Estimation Data (p. 1-7)	How to set up basic estimation information, including importing empirical data, choosing parameters for estimation, and so on
Setting Up an Estimation Project (p. 1-18)	Steps involved in creating the estimation project, which includes the data and the tasks you want to perform on the data
Selecting Views for Plotting (p. 1-24)	Plotting estimation project data
Running the Estimation (p. 1-26)	How to run the estimation and see the resulting data
Model Validation (p. 1-29)	How to compare your model's output with validation data
Setting Options for Optimization (p. 1-39)	Fine tuning the optimization process for your estimation
Setting Options for the Simulation (p. 1-43)	How to select simulation time and solvers for your Simulink model to use while estimation occurs
Estimating Independent Parameters (p. 1-46)	How to estimate parameters that are not explicitly defined in your model

## What Is Simulink Parameter Estimation?

Simulink® Parameter Estimation is a Simulink based product for estimating and calibrating model parameters from experimental data. This product supports

- **Transient Estimation** — Estimate parameters by comparing model output history and experimental data for a given input.
- **Initial Condition Estimation** — Estimate the initial conditions of states using experimental data.
- **Adaptive Lookup Tables** — Estimate the table values at the prescribed breakpoints using measurements from the physical system.

Simulink Parameter Estimation provides the tools to

- 1** Set up the problem.
- 2** Specify which model parameters to estimate.
- 3** Import and preprocess the experimental data.
- 4** Follow the estimation progress.
- 5** Validate the estimation results through various plots.

# What You Need to Get Started

## Installation

Instructions for installing Simulink Parameter Estimation can be found in the MATLAB® installation documentation for your platform. We recommend that you store the files from this product in a subdirectory named `slestim` under the main `matlab` directory. To determine if Simulink Parameter Estimation is already installed on your system, check for a subdirectory named `slestim` within the main `blockset` directory or folder.

## Prerequisite Software and Optional Software

Simulink Parameter Estimation requires MATLAB, Simulink, and the Optimization Toolbox.

The MathWorks provides several products that are relevant to the kinds of task you can perform with Simulink Parameter Estimation. For more information about any of these products, see either

- The MathWorks Web site, at <http://www.mathworks.com/products/simparameter/related.jsp>
- The online documentation for related products if they are installed on your system

## Required Knowledge

It is not necessary that you have a strong background in optimization theory or practice, but as you gain insight into the use of Simulink Parameter Estimation, you may find it helpful to consult the Optimization Toolbox documentation for more details about optimization algorithms.

## Demos

Simulink Parameter Estimation provides demonstration files that show you how to use the blockset to perform control design tasks in various settings. To run these demos, type

```
demo
```

at the MATLAB prompt. This opens the **Demos** pane in the Help browser. Select **Simulink->Simulink Parameter Estimation** to see a list of available

demos. Alternatively, if you have the Help browser open, you can select the **Demos** pane directly and follow the same procedure.

## How Simulink Parameter Estimation Works

Simulink Parameter Estimation compares empirical data with data generated by a Simulink model. Using optimization techniques, it estimates states' initial conditions and/or parameters so that a user-selected cost function, typically involving the least-square error between the two data signals, is minimized.

### Basic Steps in the Estimation Process

Once you have a Simulink model built, the basic steps involved in creating and running a parameter estimation are as follows:

- 1 Select **Parameter Estimation** from the **Tools** menu of your Simulink model.
- 2 Import an input/output data set.
- 3 Select the parameters and initial conditions you want to estimate.
- 4 Set up the estimation itself, including cost functions, data for viewing, etc.
- 5 Run the estimation.
- 6 Check the results.

### Structure of an Estimation Project

All the data stored and created by Simulink Parameter Estimation is saved as a *project*. Each project consists of a number of tasks, and the tasks consist of

- One or more data sets
- One or more sets of estimation data

For Simulink Parameter Estimation, the estimation task is associated with your Simulink model. The default name for the project is your Simulink model name. The project name is shown in the *workspace directory tree* of the Control and Estimation Tools Manager, a graphical user interface (GUI) for performing parameter estimation. See “Control and Estimation Tools Manager” on page 1-8 for a picture showing the workspace directory tree.

You can create multiple estimation tasks within a project that focus on specific parts of your estimation problem.

Projects can be shared among several of the MathWorks products. If you own Simulink Control Design or the Model Predictive Control Toolbox, then you can associate their tasks with your project. See the related product documentation for more information.

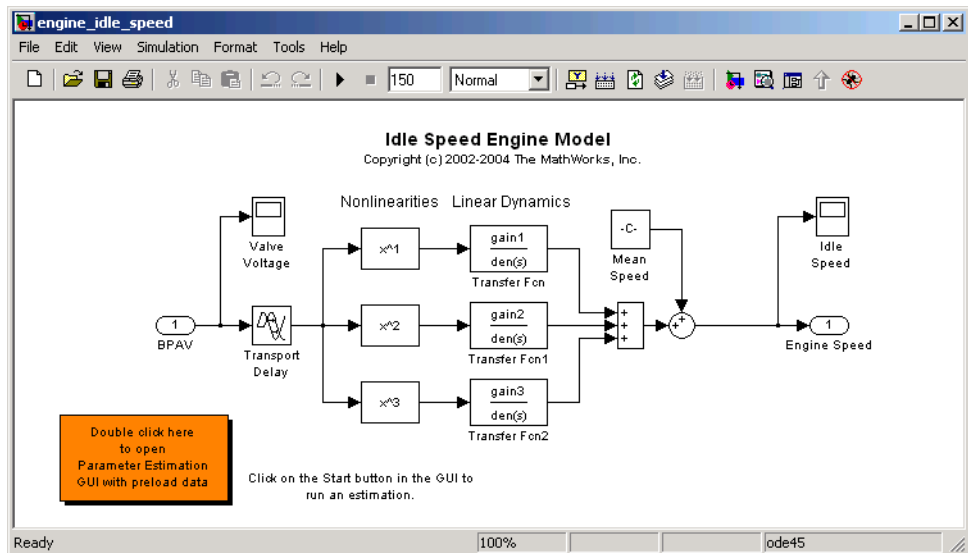
## Setting Up the Estimation Data

Before beginning the estimation process, you must set up the problem so that the appropriate parameters, solvers, cost functions, etc., are in place. Simulink Parameter Estimation provides a graphical user interface (GUI) that makes this setup process quick and easy. This section describes how to use this GUI to do a complete setup.

To show the steps of the setup, open a nonlinear model of an automotive engine's idle speed by typing

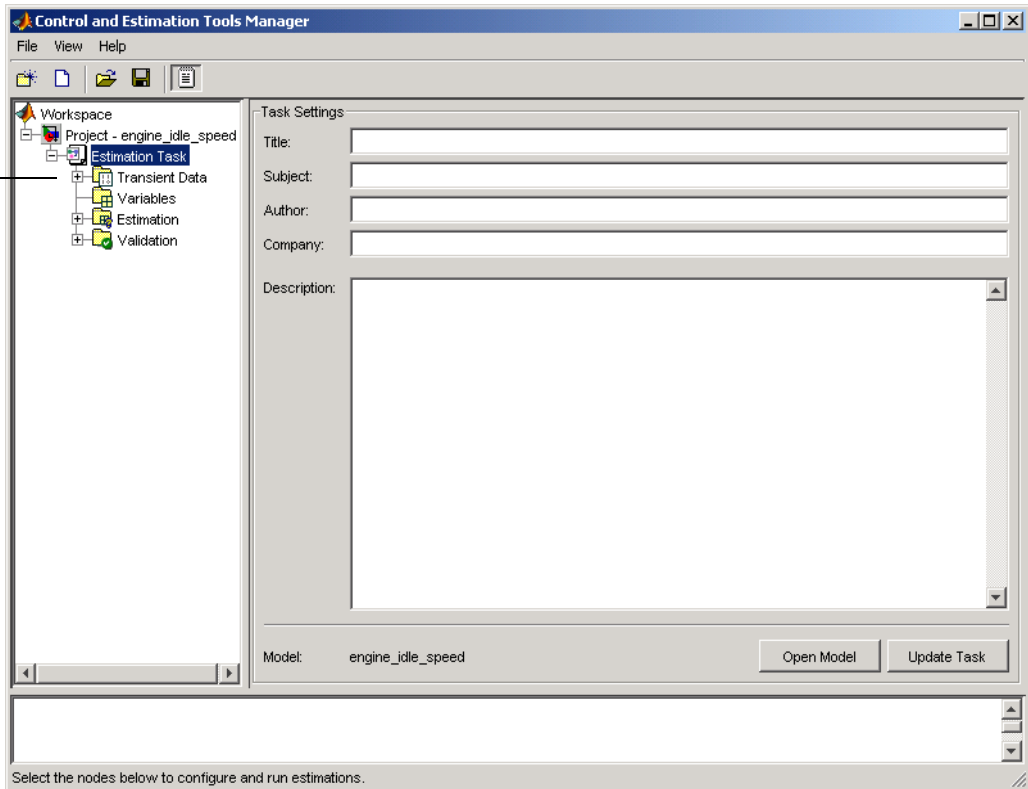
```
engine_idle_speed
```

at the MATLAB prompt. This model opens.



The **Control and Estimation Tools Manager** is a graphical user interface (GUI) for managing estimation projects. To open it, select **Parameter Estimation** from the engine idle speed model's **Tools** menu.

The workspace directory tree displays the name of your project and manages all the tasks of an estimation. Right-click on any node to add, delete, or in some cases rename the associated project or task.



## Control and Estimation Tools Manager

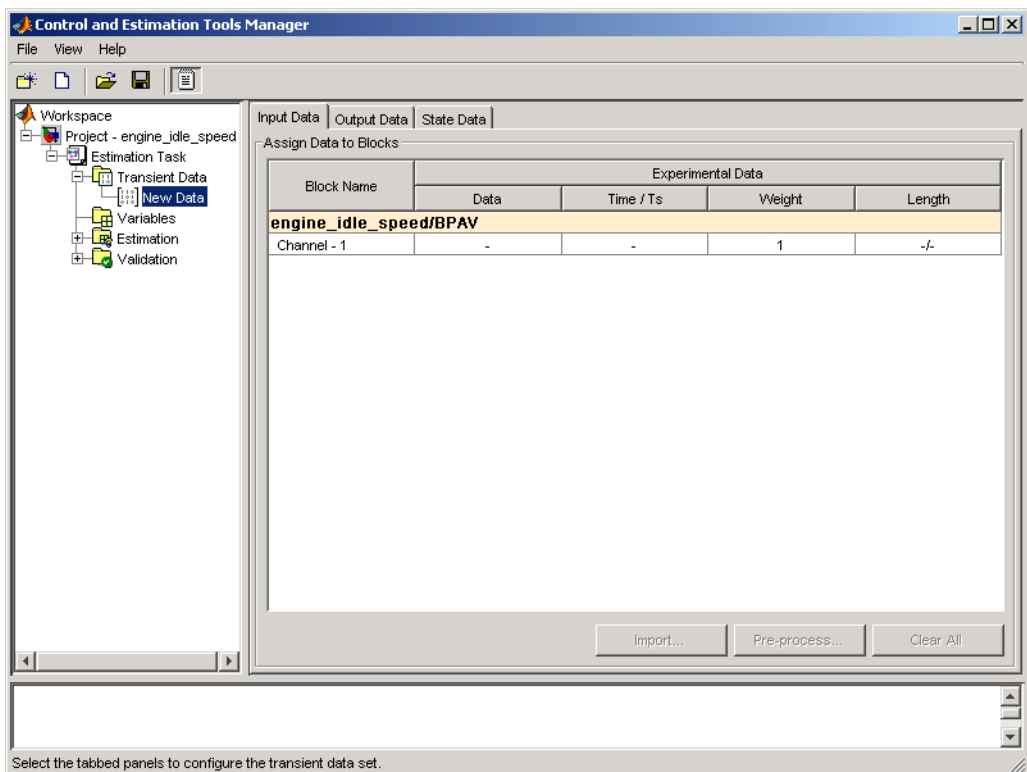
You can use the Simulink Control and Estimation Tools Manager to

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



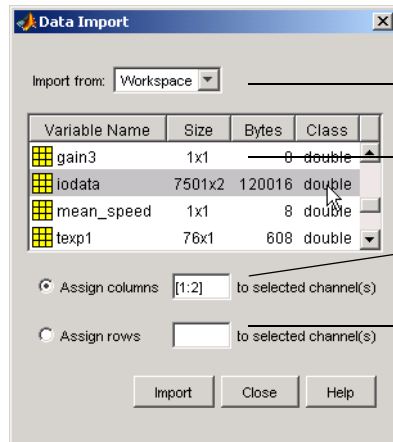
## Importing Transient Data

To import measured (empirical) data, select **Transient Data** under the **Estimation Task** folder in the Control and Estimation Tools Manager. Right-click on **Transient Data** and select **Add** to create a new data set. The engine speed model has measured data included with it in an array called `iodata`. Click on the **New Data** node in the tree to open a data panel in the tools manager.



### Data Importing in the Control and Estimation Tools Manager

The `iodata` array contains two columns, the first for input data and the second for output data. Time is stored in a separate array called `time`. Starting with the **Input Data** tab, click on the **Data** cell and click **Import**. This opens the **Data Import** dialog box.



By default, the Data Import dialog box looks at all variables in the MATLAB workspace. You can specify searches for MAT-files, Microsoft Excel (XLS) files, CSV files, or ASCII flat files.

List of available data.

In the case of multicolumn data, select the column(s) you want to import.

If your array is transposed, that is, if the data is organized in rows instead of columns, specify row numbers here.

### Use the Data Import Dialog Box to Select Your Data

To import input data,

- 1 Select the **Input Data** tab.
- 2 Select the **Data** cell in the **Input Data** table of the Control and Estimation Manager.
- 3 Click **Import** in the Control and Estimation Manager. This opens the **Data Import** window.
- 4 Select `iodata` from the list of variable names.
- 5 Enter 1 in the **Assign columns** field.
- 6 Click **Import** on the **Data Import** window.

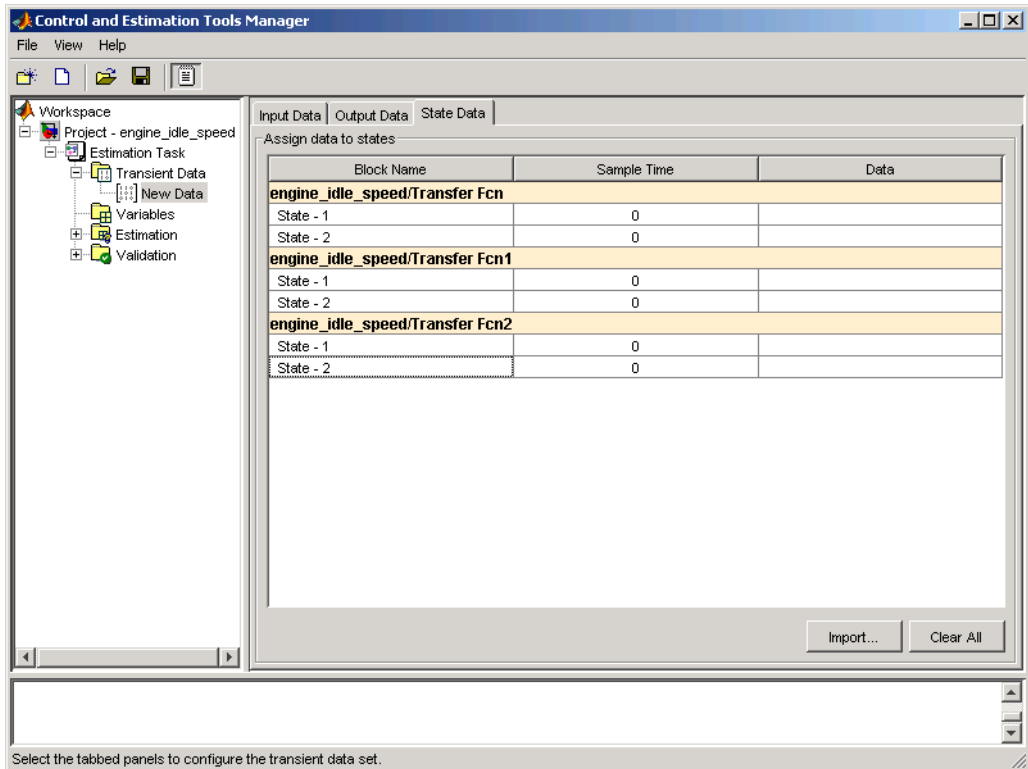
To import the time vector, follow the same procedure using the **Time/Ts** cell and the time variable.

To import the output data,

- 1** Select the **Output Data** tab.
- 2** Select the **Data** cell in the **Output Data** table of the Control and Estimation Manager.
- 3** Click **Import** in the Control and Estimation Tools Manager to open the **Data Import** Window.
- 4** Select `iodata` from the list of variables.
- 5** Specify 2 is the **Assign columns** field (i.e., use the second column of `iodata`).
- 6** Click **Import** on the **Data Import** window.

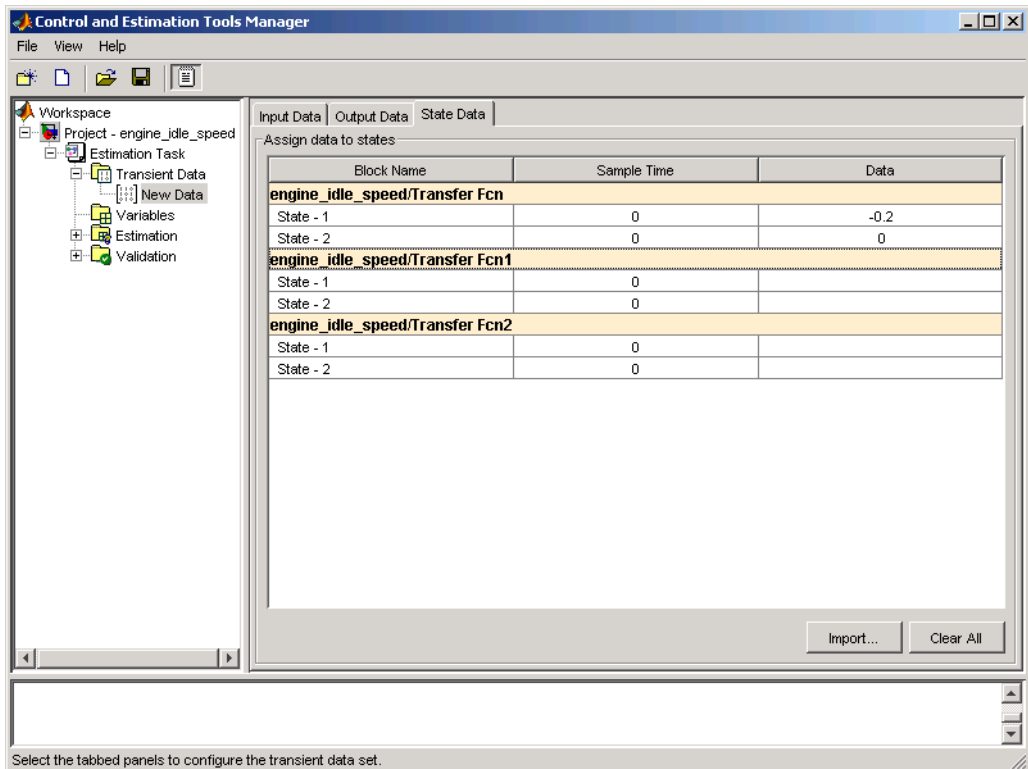
## Specifying Initial Conditions

If you want to specify initial conditions of your model's states that differ from the initial conditions specified in the Simulink model, use the **State Data** panel. You can open it by selecting **Transient Data->New Data** in the workspace directory tree, and then clicking on the **State Data** tab.



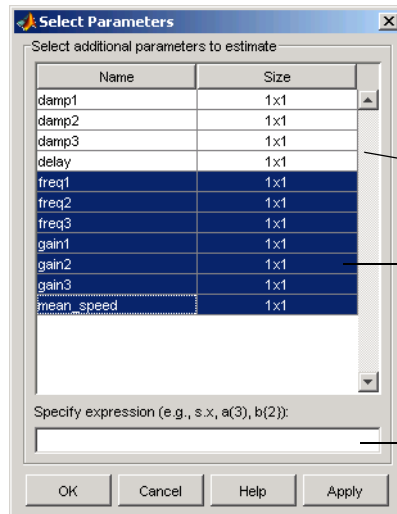
To specify a new initial condition for a state:

- 1 Select the **Data** cell associated with the state
- 2 Enter the initial condition you want. In this example, set **State - 1** of the **engine\_idle\_speed/Transfer Fcn** to **-0.2**, by selecting the associated data cell and type in the number. For **State - 2**, keep the default value of 0.



## Selecting Parameters for Estimation

To select parameters for estimation, select the **Variables** node from the workspace directory tree in the Control and Estimation Tools Manager. When the **Estimated Parameters** pane opens, click **Add** to open the **Select Parameters** dialog box.



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

List of available data.

Use your mouse to select data. Hold **Shift** down to select adjacent parameters. Hold **Ctrl** down to select nonadjacent parameters.

Use the text field to specify the parameters in MATLAB array, structure, or cell array format. Note that you cannot use mathematical expressions such as  $x + 5$ .

### Select Parameters Dialog Box

For this example, select the last seven parameters: `freq1`, `freq2`, `freq3`, `gain1`, `gain2`, `gain3`, and `mean_speed`. In general estimations, you can, of course, select fewer or more variables. 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 other, trickier parameters. Making these sorts of choices involves experience, intuition, and a solid understanding of your Simulink model's strengths and limitations.

---

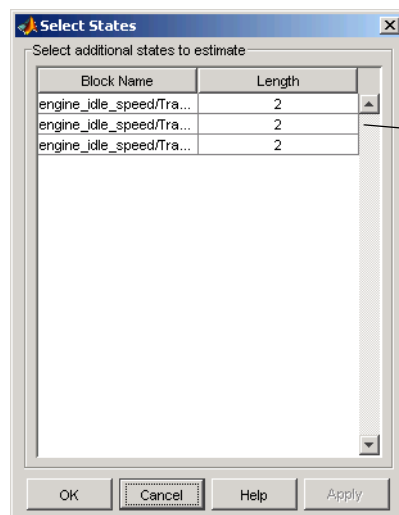
**Note** You do not have to estimate the parameters selected here all at once. You can first select all the parameters that you are interested in, then later decide which ones to estimate in a particular estimation.

---

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 “Estimating Independent Parameters” on page 1-46.

## Selecting States for Estimation

To select states for estimation, select the **Variables** node from the workspace directory tree in the Control and Estimation Tools Manager. Select the **Estimated States** pane and click **Add** to open the **Select States** dialog box.



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

List of available data.

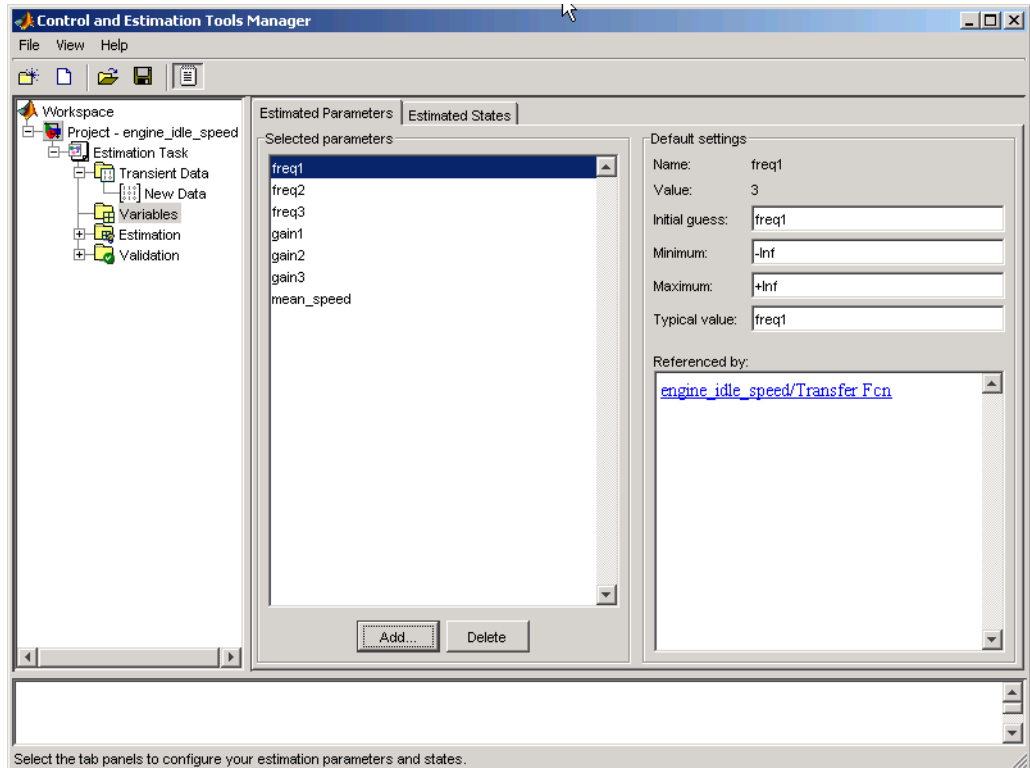
Use your mouse to select data. Hold **Shift** down to select adjacent states. Hold **Ctrl** down to select nonadjacent states.

### Select States Dialog Box

For the purposes of this example, do not select any states to estimate. In general, you can choose to estimate any, all, or none of the states in your model.

## Initial Guesses and Upper/Lower Bounds

Once you have selected parameters for estimation, the Control and Estimation Tools Manager looks like this figure.



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

- **Initial guess** — The value the estimation uses to start the process.
- **Minimum** — The smallest allowable parameter value. The default is  $-\text{Inf}$ .
- **Maximum** — The largest allowable parameter value. The default is  $+\text{Inf}$ .
- **Typical value** — The average order of magnitude. You can use the initial value here, but if you expect your parameter to range over a large number of values, use a number that averages the order of magnitude that you predict. For example, if your initial guess is 10, but you expect the parameter to vary



between 10 and 1000, use 100 (the average of the order of magnitudes) for the typical value.

The typical value has two functions:

- Use a larger value to put more emphasis on a parameter during estimation.
- Try to select the typical values so that

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

or

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

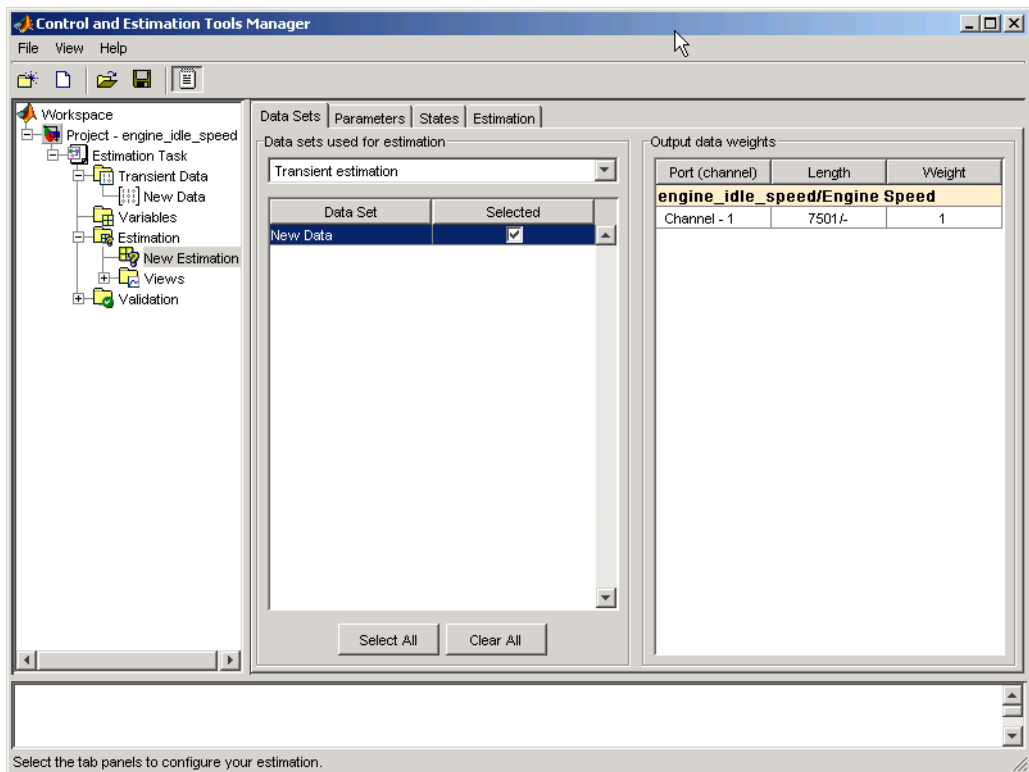
Either case places equal emphasis on all parameters.

## Setting Up an Estimation Project

With the transient data imported and the parameters for estimation selected, you are now ready to set up an estimation.

### Adding Data Sets

Right-click on **Estimation** in the workspace directory tree of the Control and Estimation Tools Manager and select **Add** to create a **New Estimation** node.



Select the **New Estimation** node in the directory tree and then select **New Data** in the table to add your engine data to the estimation.

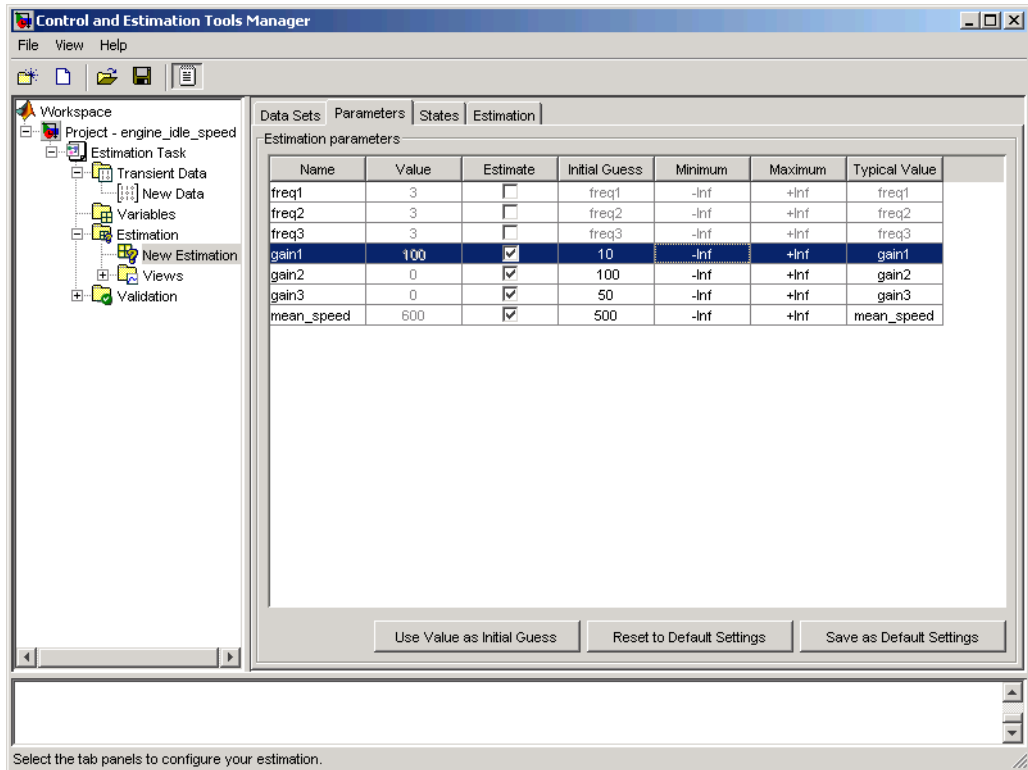
You can specify the weight of each output in this data set by setting the **Weight** column in the **Output data weights** table. Specifying weights is similar to setting typical values for parameters.

The weights are used to place more or less emphasis on parts of the output data. The following are a few guidelines for specifying weights:

- Use less weight if your output data is noisy.
- Use more weight if your output data has more effect on parameters.
- Use more weight if matching that data output accurately is more important.

## Specifying and Setting Up Parameters

You can use the **Parameters** pane for your new estimation in the Control and Estimation Tools Manager to select which parameters to estimate and the range of values for the estimation.



Select the parameters you want to estimate in the **Estimate** column. Enter initial values for your estimation 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. 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 is computational advantage in specifying finite bounds if you can.

---

**Note** Setting min, max, and so on values here does not affect the settings you had in the **Variables** node. You make these choices on a per estimation basis. Using the buttons in this panel, however, you can move data to and from the **Variables** node.

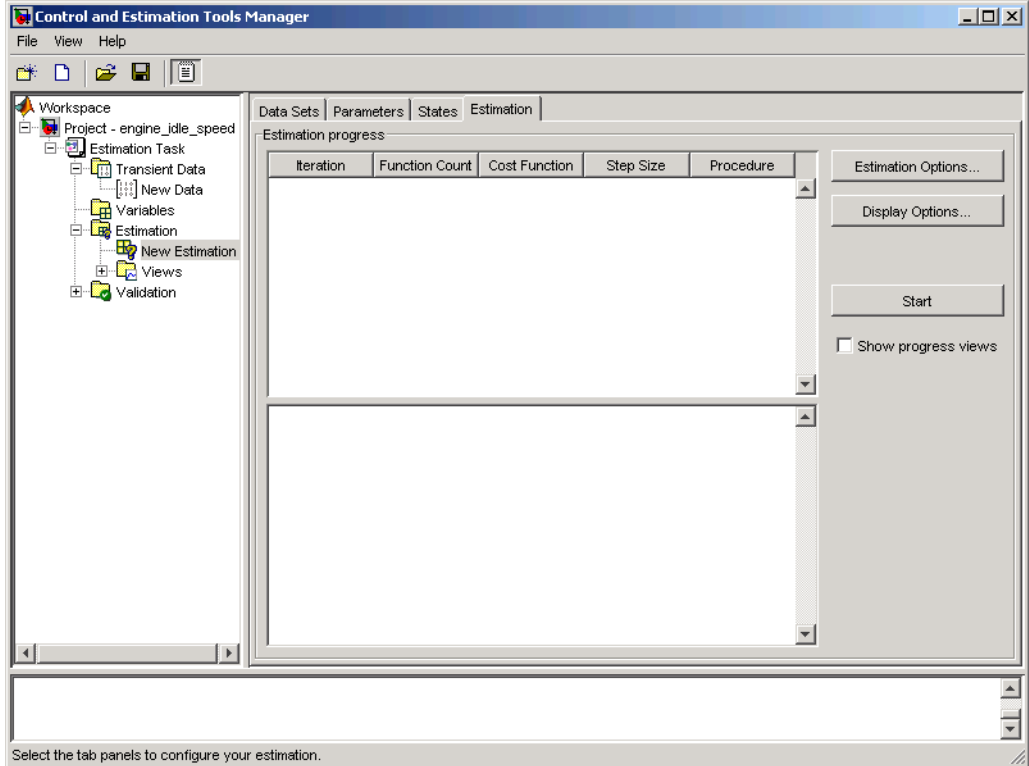
---

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.

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

## Opening the Estimation Window

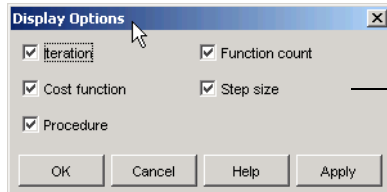
Select the **Estimation** tab to open the Estimation pane for your new estimation.



Before you start, you can click **Estimation Options** to specify various algorithm and simulation features. See “Selecting Optimization Methods” on page 1-39 for more information.

## Display Options

Clicking **Display Options** opens this dialog box.

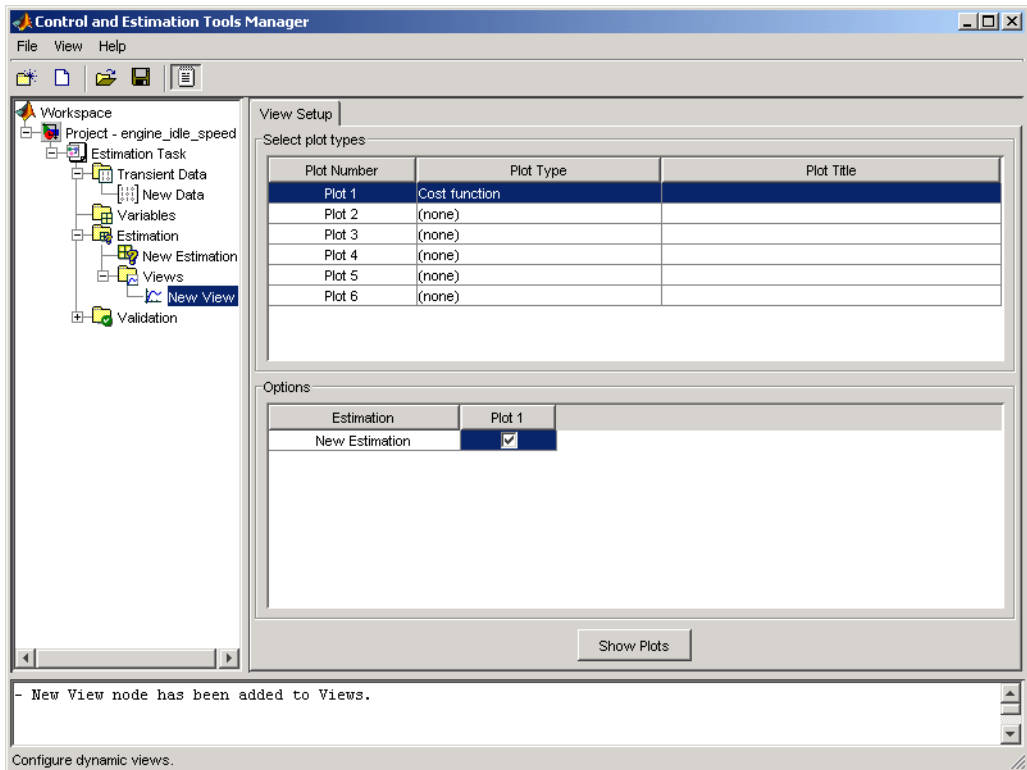


By default, all boxes are checked. Uncheck any feature that you don't want to view during the estimation process.

Unchecking a box means that data will not show up in the display table for the estimation.

## Selecting Views for Plotting

You can watch the minimization process occur by right-clicking on **Views** in the workspace directory tree of the Control and Estimation Tools Manager and then selecting **Add**. In the workspace directory tree, select **New View**. This opens the **View Setup** panel.



Select the desired plot type by clicking on the first cell under the **Plot Type** column in the **Select Plot Types** table. Various types of plots are available, including

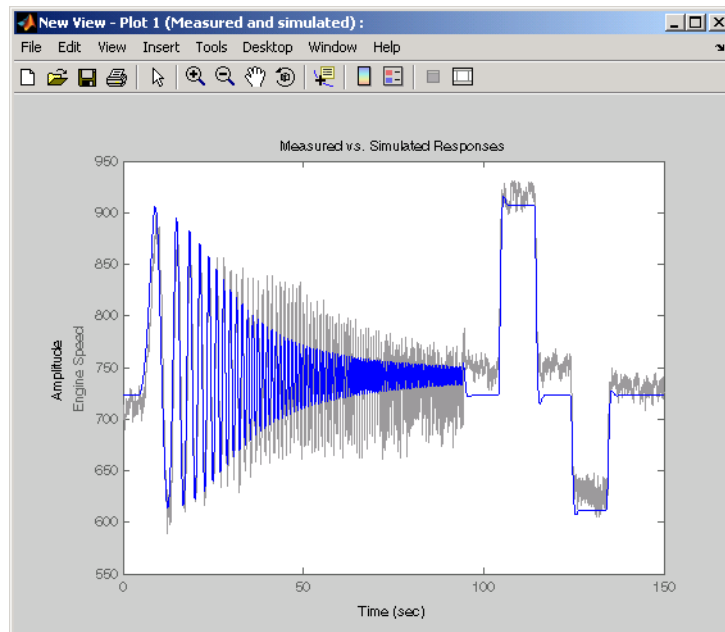
- **Cost function** — Plot the cost function values.
- **Measured and simulated** — Plot the empirical data against the simulated data.



- **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.

For this example, select **Cost function**. Check **Plot 1** in the **Options** table, and then click **Show Plots**. This opens a plot window for the cost function. When you run your estimation, the plot updates automatically.

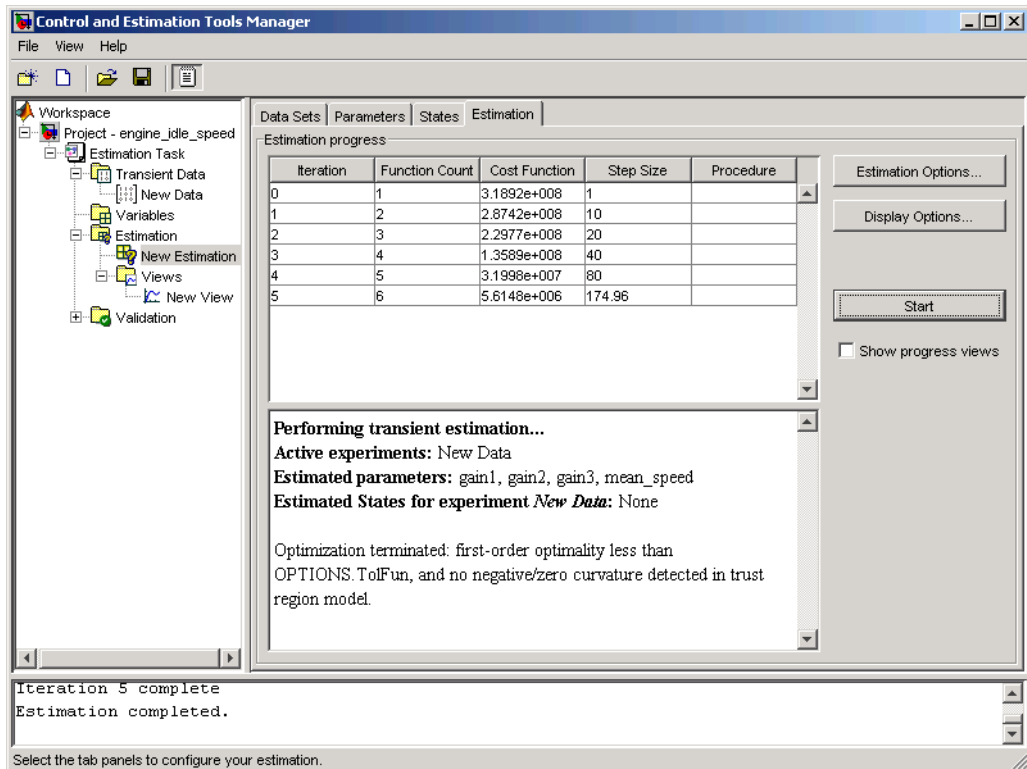
This figure shows the plot.



**Note** An estimation must be created prior to creating views. Otherwise, the Options table will be empty.

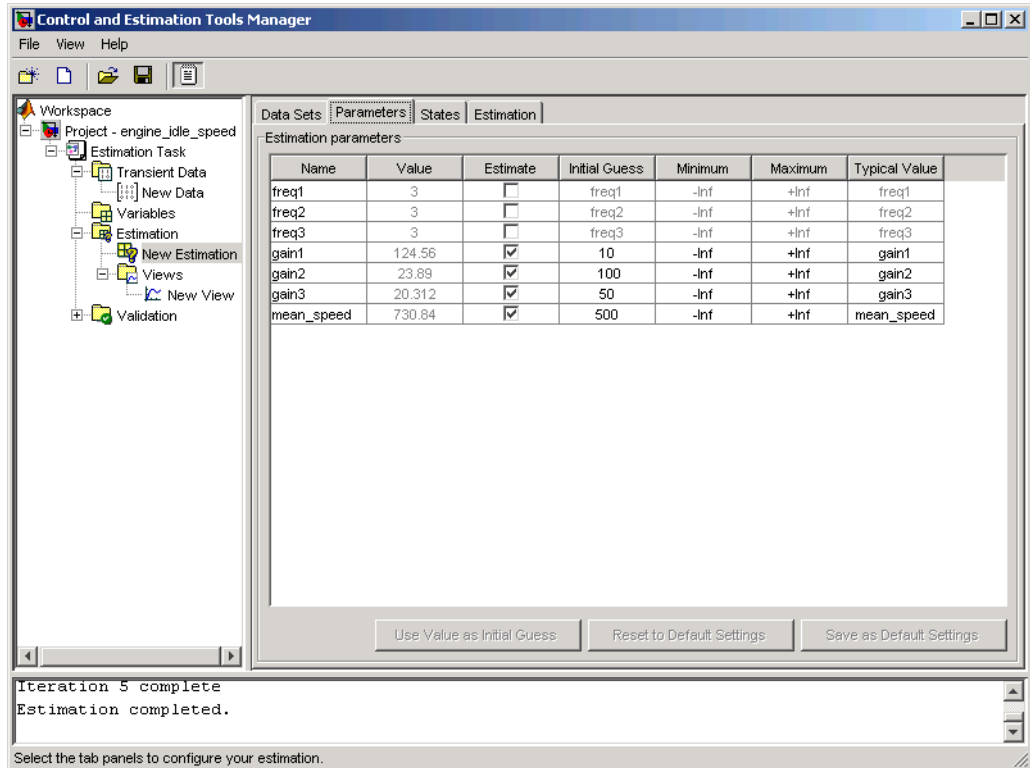
## Running the Estimation

In the Control and Estimation Tools Manager, select the **Estimation** panel for your **New Estimation** and click **Start** to begin the estimation process. At the end of the iterations, the window should look something like this.



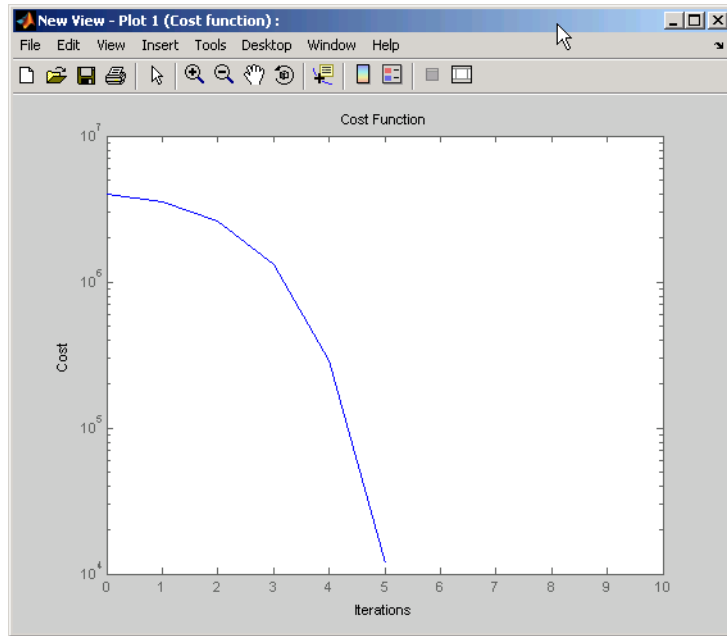
Usually, a lower cost function value indicates a successful estimation, meaning that the experimental data matches the model simulation with the estimated parameters.

The **Estimation** panel displays each iteration of the optimization algorithm. To see the final values for the parameters, go to the **Parameters** panel.



The values of these parameters are also updated in the MATLAB workspace. So, 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.

The cost function minimization is plotted below.



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

## Model Validation

Once you've completed an estimation, you can validate your results against another set of data.

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

- 1 Add the validation data to the **Transient Data** sets.
- 2 Add a new validation in the **Validation** node of the workspace directory tree.
- 3 Edit the validation—select plot types you want from the **Validation Setup** panel and select the validation data set you want to use.
- 4 Click **Show Plots** in the Validation Setup panel and view the results in the plot window.
- 5 Compare the validation plots to corresponding view plots to see if the match is close.

The basic difference between the validation and views features is that you can run validations after your estimation is complete. All views should be set up prior to 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.

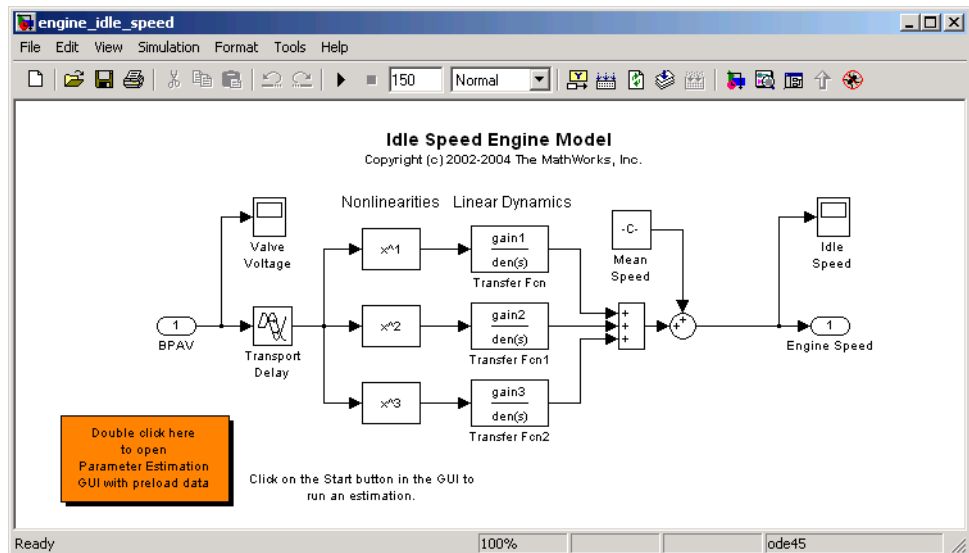
You can validate your data by comparing measured vs. simulated data for your estimation data and validation data sets. Also, it is often useful to compare residuals in the same way.

## Example: Validating the Engine Idle Speed Model

If you haven't run the engine idle speed demo, type

```
engine_idle_speed
```

at the MATLAB prompt and run the estimation. If you haven't run an estimation yet, see "How Simulink Parameter Estimation Works" on page 1-5. To save time, double-click on the box in the upper-left corner of the model to import data and populate the required fields in the Control and Estimation Tools Manager.



### The engine\_idle\_speed Simulink Model

Now that the estimation data is loaded, and the estimation task has been created, the next step is to import validation data into the Control and Estimation Tools Manager.

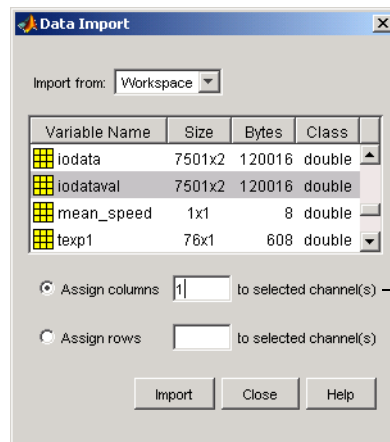
## Loading and Importing the Validation Data

To load the validation data, enter

```
load iodataval
```

at the MATLAB prompt. This loads the data into the MATLAB workspace. The next step is to import this data into the tools manager. See “Importing Transient Data” on page 1-9 for information on importing data, but the quickest way is to follow these steps:

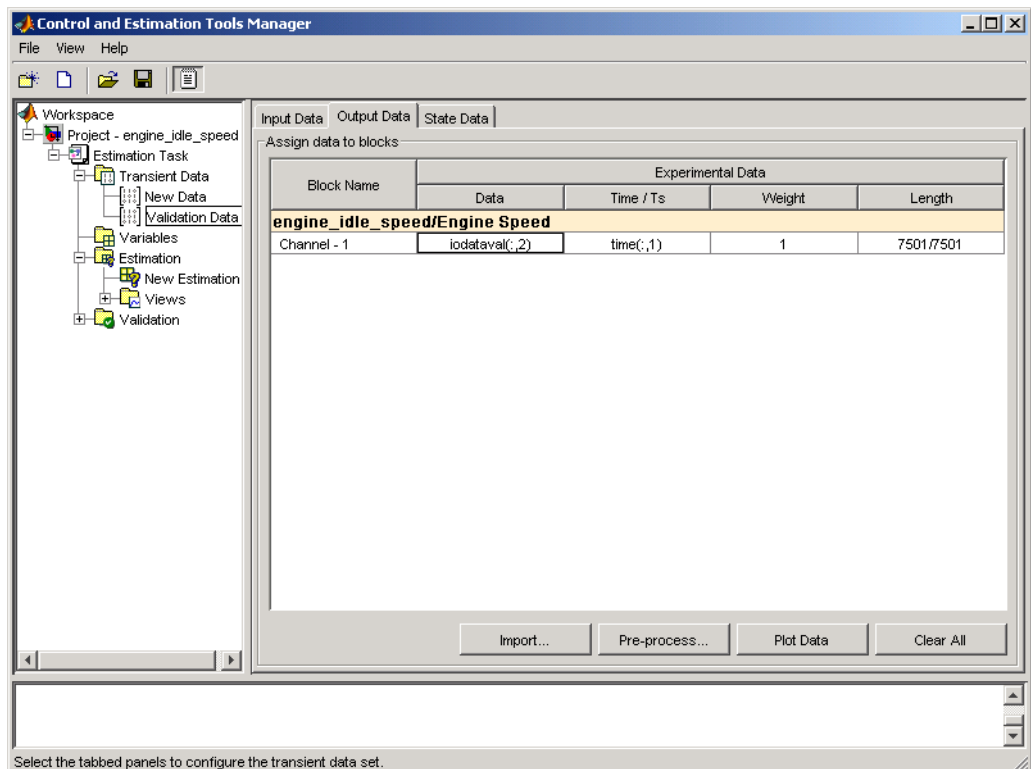
- 1 Right-click on the **Transient Data** node of the workspace directory tree of the Control and Estimation Tools Manager and select **Add**.
- 2 Select **New Data 2** from the **Transient data sets** panel and click **Edit**.
- 3 Select the **New Data (2)** node from the workspace directory tree of the Control and Estimation Tools Manager. Right-click and select **Rename**. Change the name of the data to **Validation Data**. (You can also change the name by double-clicking on **New Data (2)** in the **Transient data sets** panel and selecting **Rename**.)
- 4 On the **Input Data** panel, select the **Data** cell associated with **Channel 1 - 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 data.



Enter 1 in the **Assign columns** field to import the first (input) column of data in the **iodataval** array.

- 5 Select the **Time/Ts** cell and import time using the **Data Import** dialog box.
- 6 Similarly, on the **Output Data** panel, select **Time/Ts** and import time.
- 7 On the **Output Data** panel, 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 data.

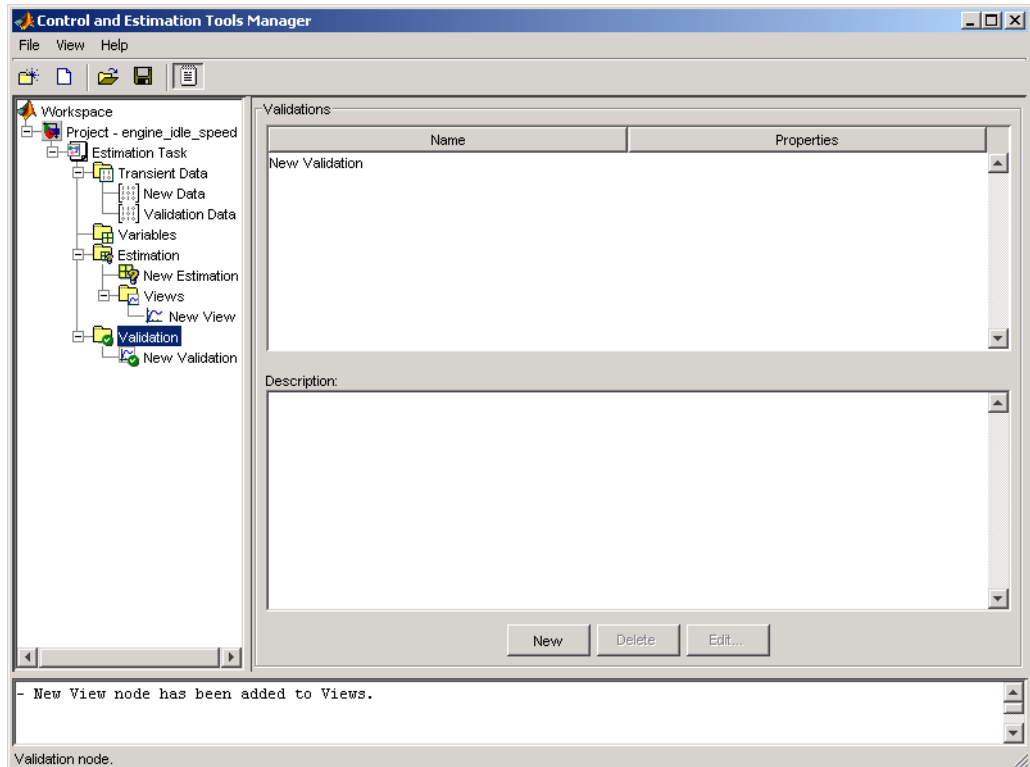
Your Control and Estimation Tools Manager should look like this figure.





## Adding the Validation Task

Once you've imported the data, right-click on the **Validation** node and select **Add**. This opens the **Validations** panel in the Control and Estimation Tools Manager.

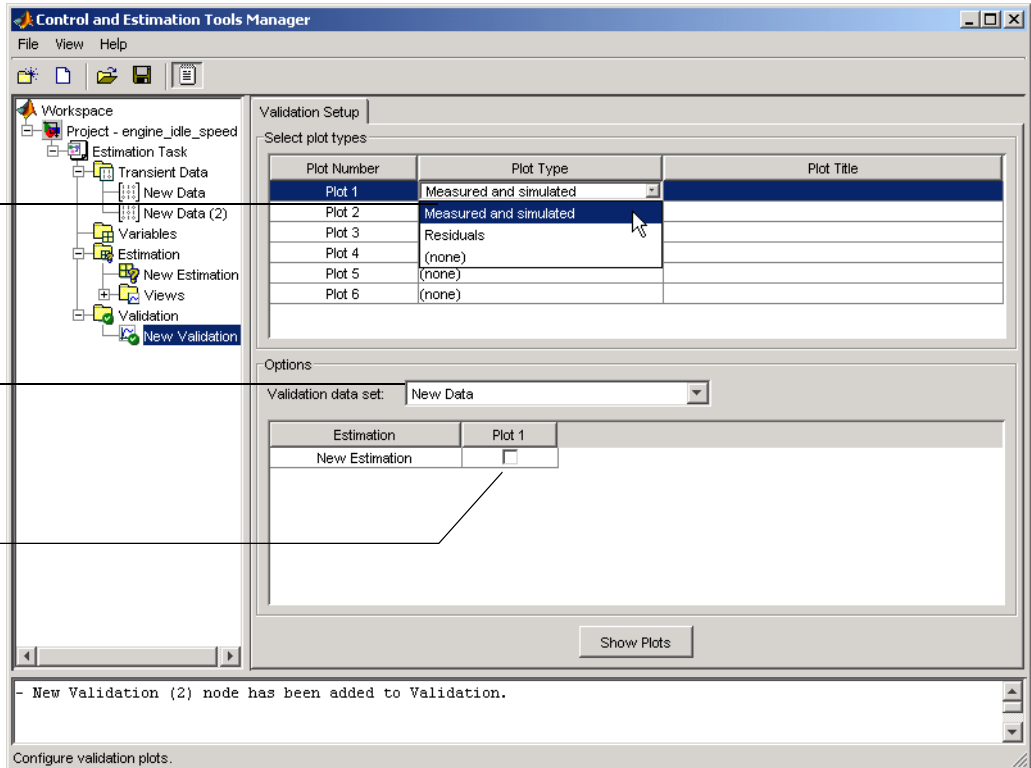


Click on **New Validation** in the workspace directory tree to open the **Validation Setup** panel.

Click on the **Plot Type** cell associated with **Plot 1** to open the **Plot Type** menu. Choose the plot type from this menu.

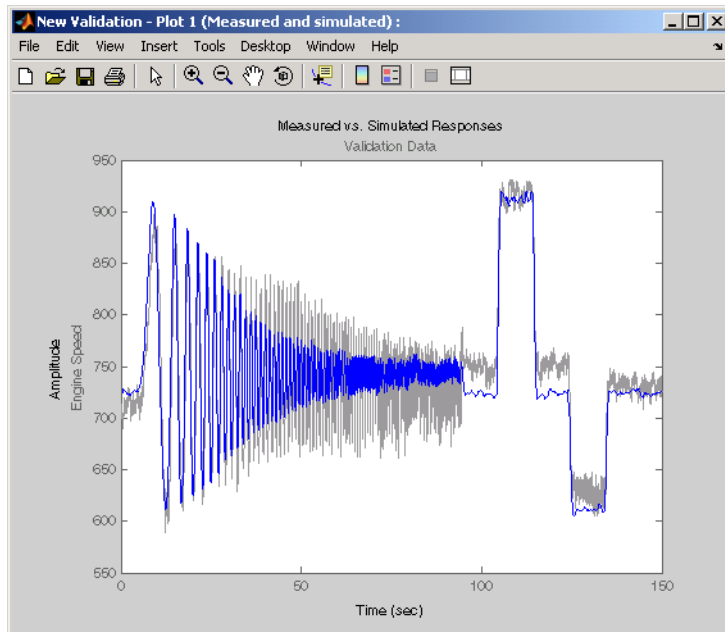
Choose the data you want to use for validation from the **Validation data set** pull down menu.

Check the box to add the data to the plot.



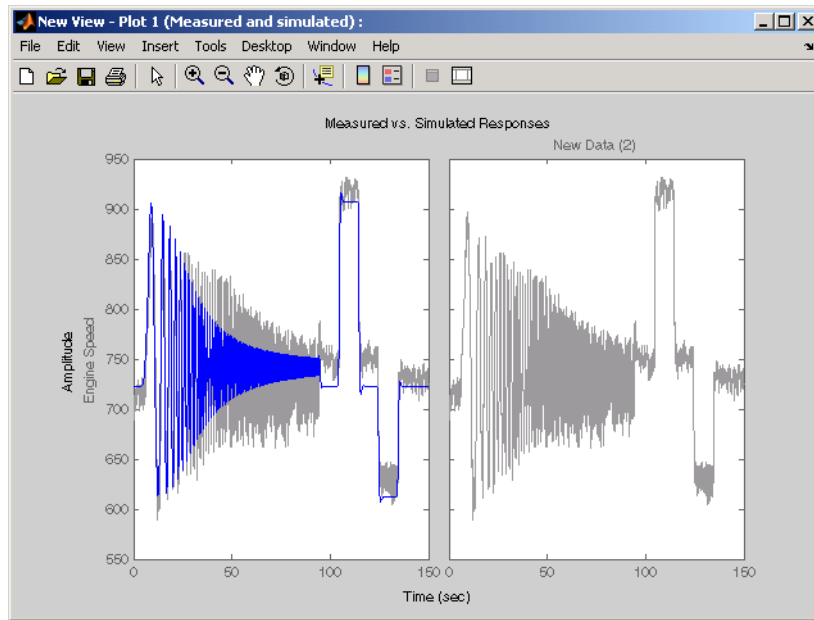
## Running the Validation

For plot 1, select Measured and simulated from the **Plot Type** menu, and choose Validation Data from the **Validation data set** menu. Click **Show Plots**. This opens a plot figure window as shown below.



**Measured (Validation) vs. Simulated Data Plot**

Compare this with the plot of measured and simulated data from the **Views** node of the workspace directory tree.

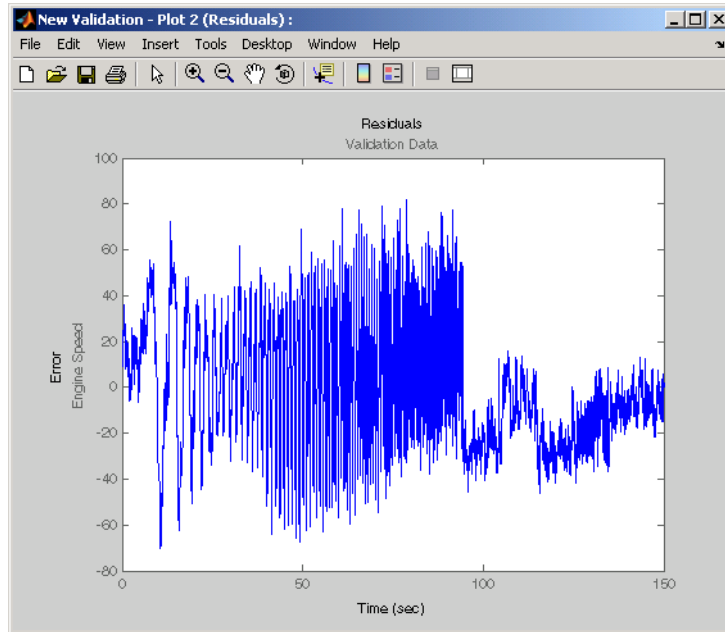


### Measured and Simulated Data Views Plot

Since the validation data is entered as a Transient Data set, it shows up in the right side plot as measured data. The left-hand plot, however, is the measured and simulated data that you should compare to the measured and simulated data plot that used the validation data.

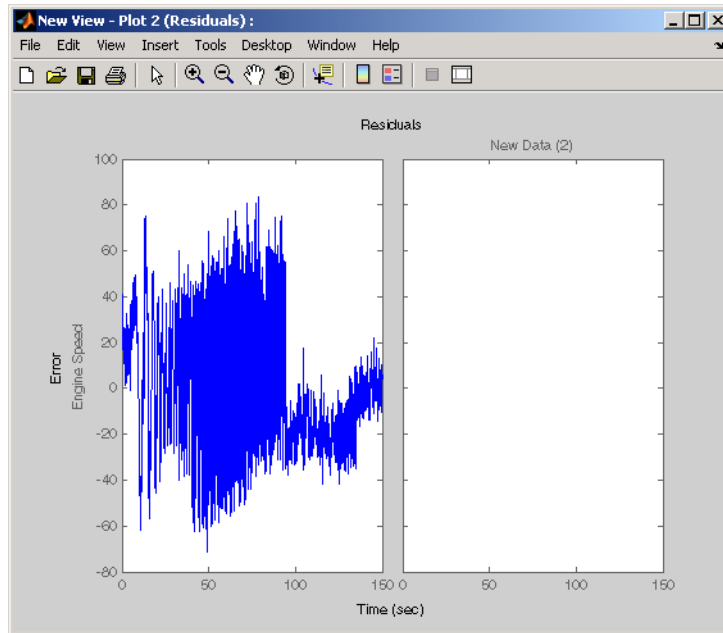
## Residuals

To look at the residuals in **Plot 2**, select Residual from the **Plot 2** menu. This figure shows the resulting plot.



**Plot of Residuals Using the Validation Data**

Compare the validation data residuals to the original data set residuals from the Views node of the workspace directory tree.

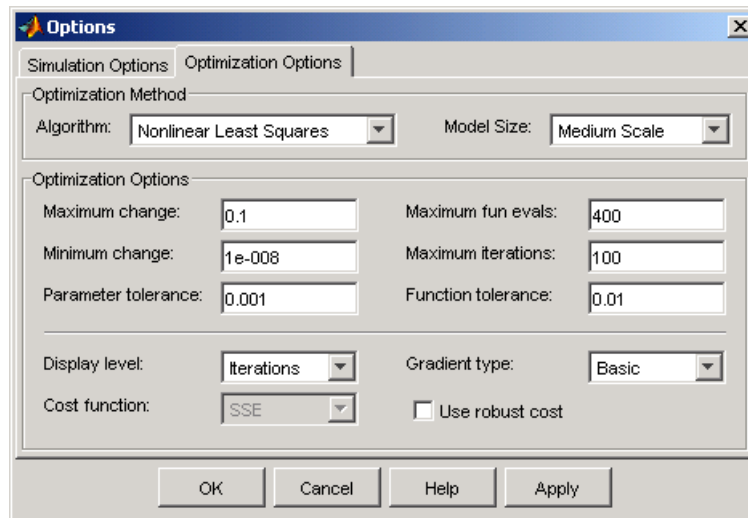


### Plot of Residuals Using the Test Data

The left side plot agrees closely with the residual data for the validation data. The right side plot is empty since no residuals were calculated for the validation data during the original estimation process.

## Setting Options for Optimization

Several options can be set to tune the results of the optimization. These options include the optimization algorithms and the tolerances the algorithms use. To set options for optimization, click **Estimation Options** in the **Estimation** panel of the Control and Estimation Tools Manager. This opens the **Options** dialog box.



### Selecting Optimization Methods

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



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

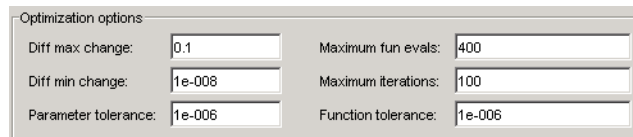
- **Gradient descent** — Uses the Optimization Toolbox function `fmincon` to optimize the response signal subject to the constraints

- **Nonlinear least squares** — Uses a nonlinear least squares optimization algorithm.
- **Pattern search** — Uses an advanced pattern search algorithm. 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 Function minimization for models that contain discontinuities.

By default, the **Model Size** parameter is set to `Large scale`. When the number of parameters you want to estimate is large, **Model Size** should use the default to increase computation speed. If your model is not very large, it may be more efficient to select `Medium scale`. See the Optimization Toolbox documentation for more information about the optimization methods.

## Selecting Optimization Termination Options

Use the **Optimization Options** panel to specify termination options.



The screenshot shows a panel titled "Optimization options" with six input fields arranged in two columns. The left column contains "Diff max change" (0.1), "Diff min change" (1e-008), and "Parameter tolerance" (1e-006). The right column contains "Maximum fun evals" (400), "Maximum iterations" (100), and "Function tolerance" (1e-006).

Option	Value
Diff max change:	0.1
Diff min change:	1e-008
Parameter tolerance:	1e-006
Maximum fun evals:	400
Maximum iterations:	100
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 Toolbox documentation 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.

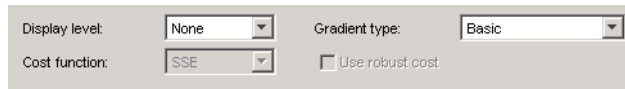


- **Maximum iterations** — The maximum number of iterations allowed. The optimization terminates when the number of iterations exceeds this value.
- **Function tolerance** — The optimization terminates when successive function values are less than this value.

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 panel is a group of additional optimization options.



The screenshot shows a portion of the Optimization Options panel. It contains four controls: 'Display level' is a dropdown menu set to 'None'; 'Gradient type' is a dropdown menu set to 'Basic'; 'Cost function' is a dropdown menu set to 'SSE'; and 'Use robust cost' is an unchecked checkbox.

Additional options for optimization include

- **Display level** — This option 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**, Simulink Parameter Estimation calculates gradients 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.

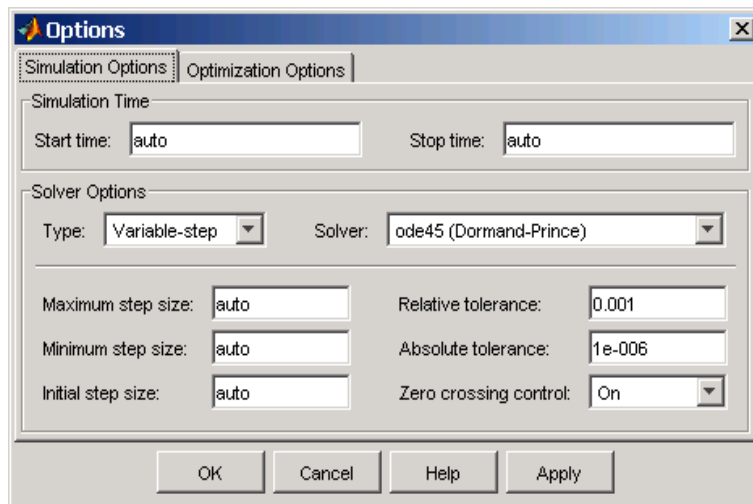
## Specifying the Cost Function

The *cost function* is a function that optimization algorithms attempt to minimize. You have the following options when selecting a cost function:

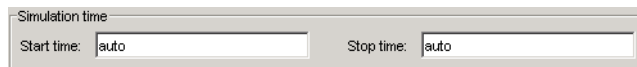
- **Cost function** — The default is SSE (sum of squared errors), which uses a least-squares approach. You can also use SAE, the sum of absolute errors.
- **Use robust cost** — Makes the optimizer use a robust cost function instead of the default least-squares cost. This is useful if the experimental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

## Setting Options for the Simulation

To optimize the response signals of a model, Simulink Parameter Estimation runs simulations of the model. You can set options for these simulations by clicking **Estimation Options** in the **Estimation** panel of the Control and Estimation Tools Manager. This opens the **Options** dialog box.

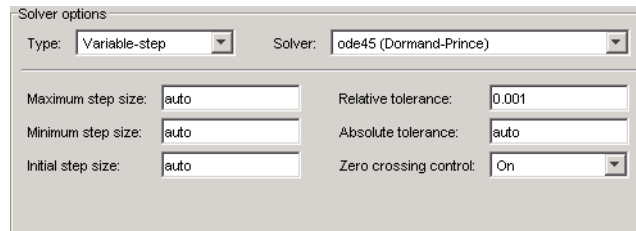


### Selecting Simulation Time



By default, the **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**.

## Selecting Solvers



When running the simulation, Simulink solves the dynamic system using one of several solvers. You can specify several solver options using the **Solver Options** panel 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.

### 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 solver **Type**, you can also set several other parameters that affect the step-size of the simulation:

- **Maximum step size** — The largest step-size Simulink can use during a simulation

- **Minimum step size** — The smallest step-size Simulink can use during a simulation
- **Initial step size** — The step-size Simulink 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, Simulink automatically chooses values for these options. 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-Shampine)
- 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.

## Estimating 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 use Simulink Parameter Estimation to estimate independent parameters:

- 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. Add these parameters to the list of parameters to be estimated.

---

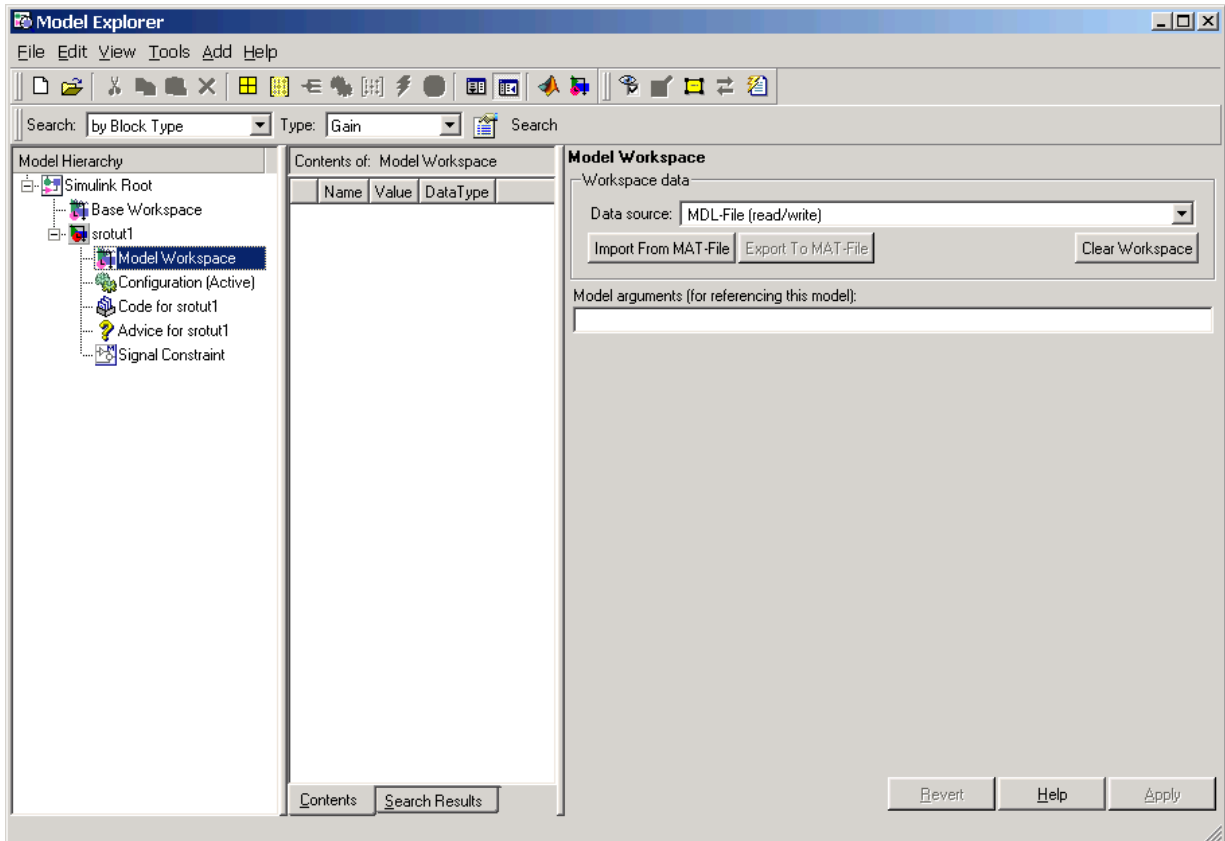
**Caution** Avoid adding independent parameters together with their corresponding dependent parameters to the lists of parameters to be estimated. Otherwise the estimation could give incorrect results. For example, when a parameter  $x$  depends on the parameters  $a$  and  $b$ , avoid adding all three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

---

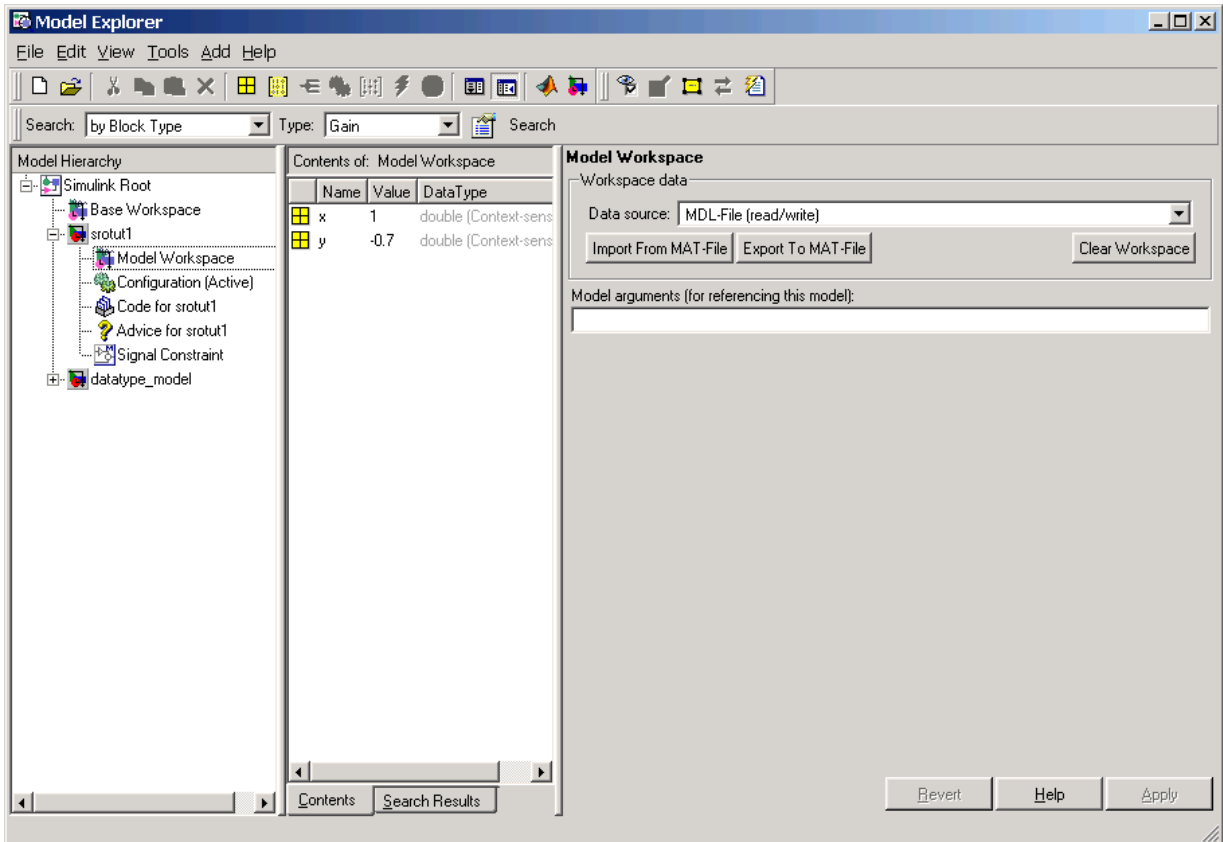
### Example:

Assume that the parameter  $K_{int}$  in the model `srotut1` is related to the parameters  $x$  and  $y$  according to the relationship  $K_{int}=x+y$ . Also assume that the initial values of  $x$  and  $y$  are 1 and -0.7 respectively. To estimate  $x$  and  $y$  instead of  $K_{int}$ , first define these parameters in the model workspace. To do this

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



- 3 Select **Add -> MATLAB Variable** within the **Model Explorer** to add a new variable to the model workspace. A new variable appears within the pane labeled **Contents of: Model Workspace**. Change the variable name to **x** and the initial value to **1**.
- 4 Repeat step 3 to add a variable **y** with an initial value of **-0.7**. The Model Explorer window should now look like the following figure.



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



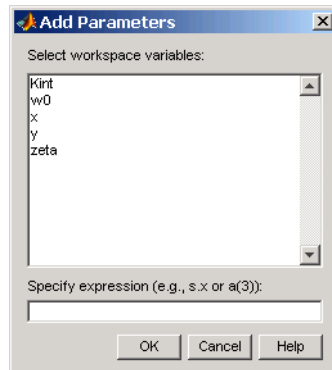
```
wks = get_param(gcs, 'ModelWorkspace')
x = wks.evalin('x')
y = wks.evalin('y')
Kint = x+y;
```

---

**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 parameter to be estimated, `x` and `y` should now appear in the **Add Parameters** dialog box.





# Estimating Initial Conditions

---

Why Estimate Initial Conditions?  
(p. 2-2)

Example: Mass-Spring-Damper  
System (p. 2-3)

Reasons for estimating initial conditions of states in your model

An example that takes you step-by-step through an estimation of the initial position of a mass attached to a spring

### Why Estimate Initial Conditions?

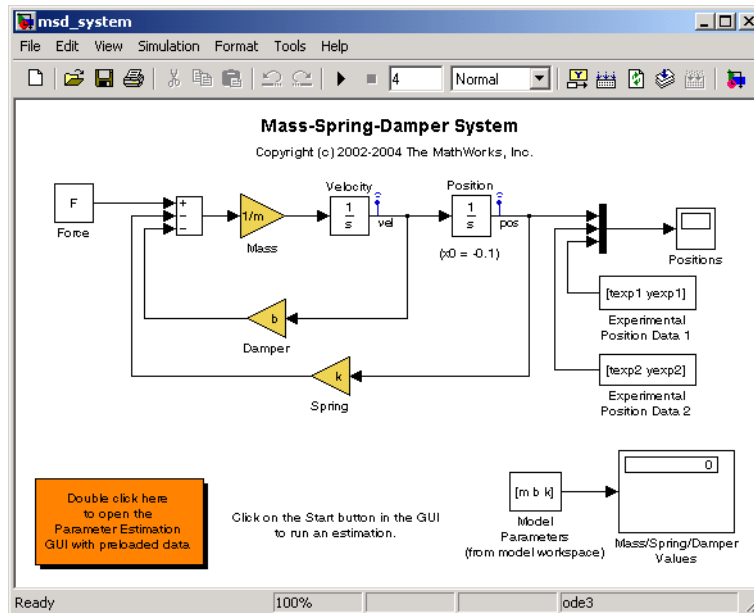
Often, sets of measured data are collected at various times and under different initial conditions. If you estimate parameters for your Simulink model using one set, then try again with another, your parameter values may not match. Given that Simulink Parameter Estimation attempts to find constant values for parameters, this is clearly a problem.

Fortunately, Simulink Parameter Estimation has features that make this task simpler. The Control and Estimation Tools Manager has an **Estimated States** panel that lists the states available for initial condition estimation. So, you can estimate 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 following sections discuss the steps required to estimate initial conditions, and then estimate parameters given the initial condition estimates.

## Example: Mass-Spring-Damper System

The figure below is a Simulink model of a mass-spring-damper system.




This model is available as a demo for Simulink Parameter Estimation, but this section discusses each step of the task in more depth. If you want to run the demo, see the listings under **Simulink** for **Simulink Parameter Estimation** on the **Demo** page of the Help browser.

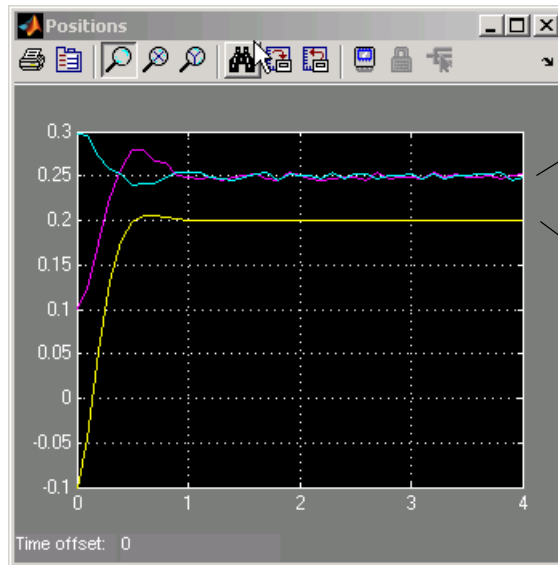
To open the model and two sets of model data with differing initial conditions, type

```
msd_system
```

at the MATLAB prompt.

### 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,  $x_0$ , for the mass displacement.  $x_0$  is indicated by the initial condition of the Position integrator block. Click the Start button  to run the simulation once and observe the response of the model to two sets of parameter values.



These magenta and cyan curves are empirical responses for data with different initial conditions.

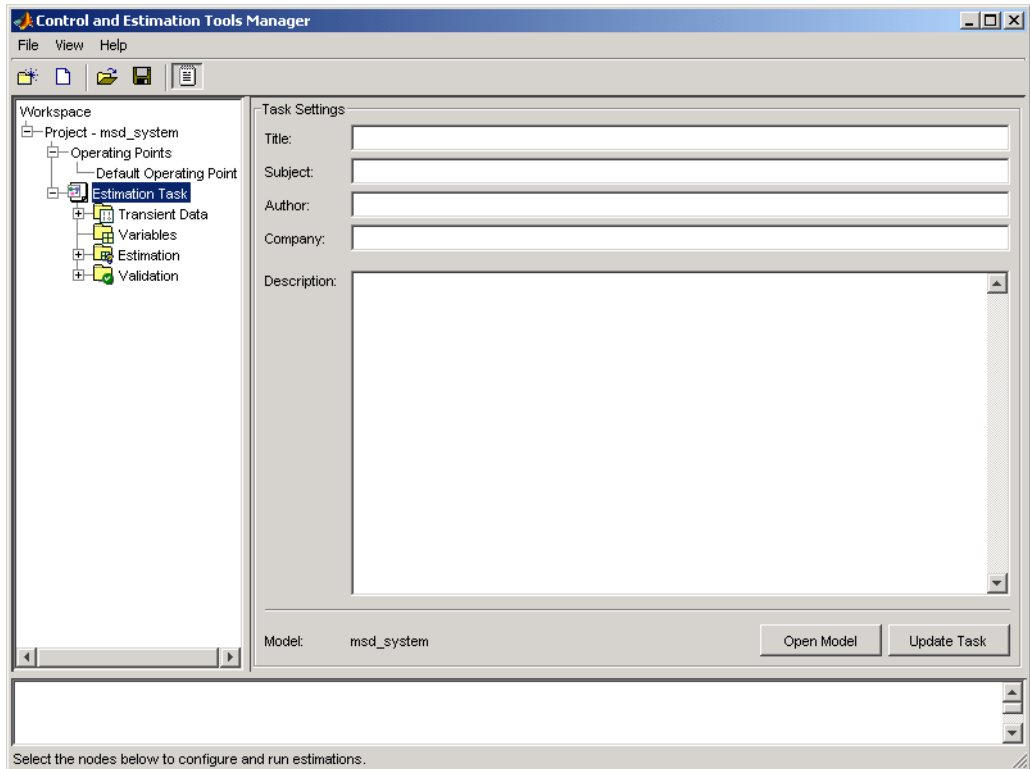
This yellow curve is the response of the model to a constant force.

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,  $x_0=0.1$  and  $x_0=0.3$ , 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  $x_0=-0.1$  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 **Parameter Estimation** from the **Tools** menu of the model `msd_system` window.



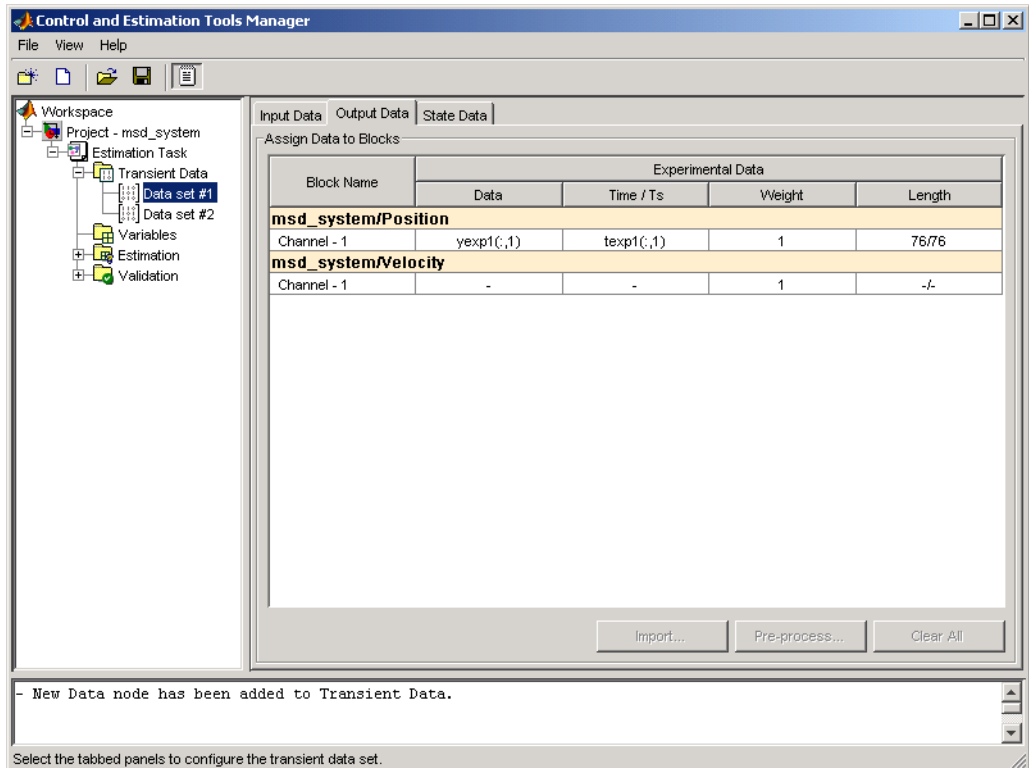
## Importing Transient Data and Selecting Parameters for Estimation

The process for importing transient data and selecting parameters for estimation is discussed in “Importing Transient Data” on page 1-9 and “Selecting Parameters for Estimation” on page 1-14. Briefly,

- 1 Select the **Transient Data** node from the workspace directory tree of the Control and Estimation Tools Manager.

- 2 Right-click **Add** to add a new data set.
- 3 Select the New Data entry in the **Transient Data** column and click **Edit** to open the **Input Data**, **Output Data**, and **State Data** panels.
- 4 In the **Output Data** panel, 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 on **New Data** in the workspace directory tree and rename it to `Data set #1`.
- 6 Follow the same procedure in steps 1 through 4 to add the second data set, `yexp2` and `texp2`, and rename it to `Data set #2`.

Your Control and Estimation Tools Manager should look like this.

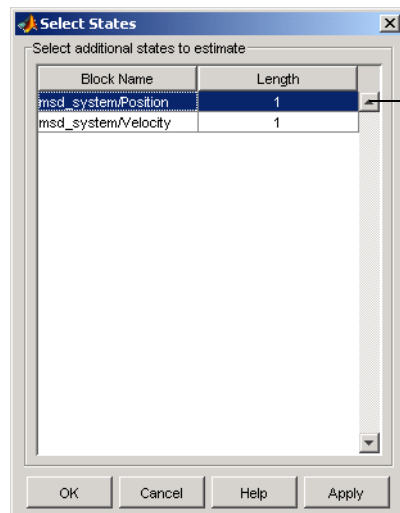




## Selecting Parameters and Initial Conditions for Estimation

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

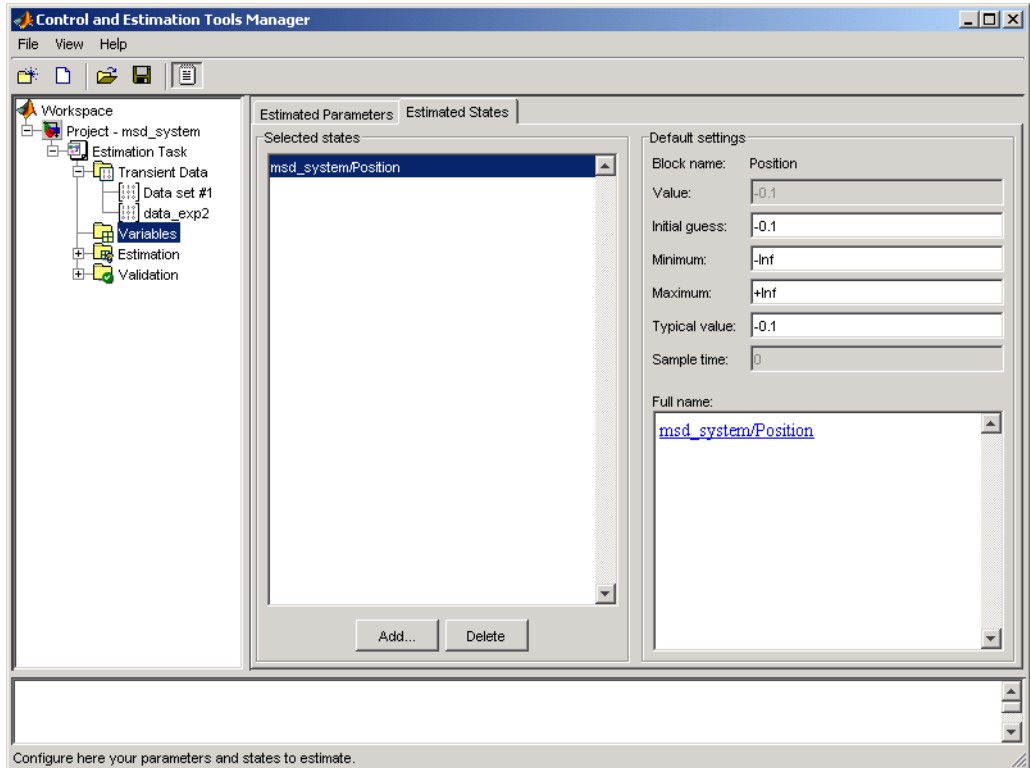
- 1 Select the **Variables** node in the workspace directory tree of the Control and Estimation Tools Manager.
- 2 Select the **Estimation Parameters** panel.
- 3 Click **Add** and select `b`, `k`, and `m` from the **Select Parameters** dialog box that opens.
- 4 Do the same with the **Estimation States** panel, and select `msd_system/Position` from the **Select States** dialog box.



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.

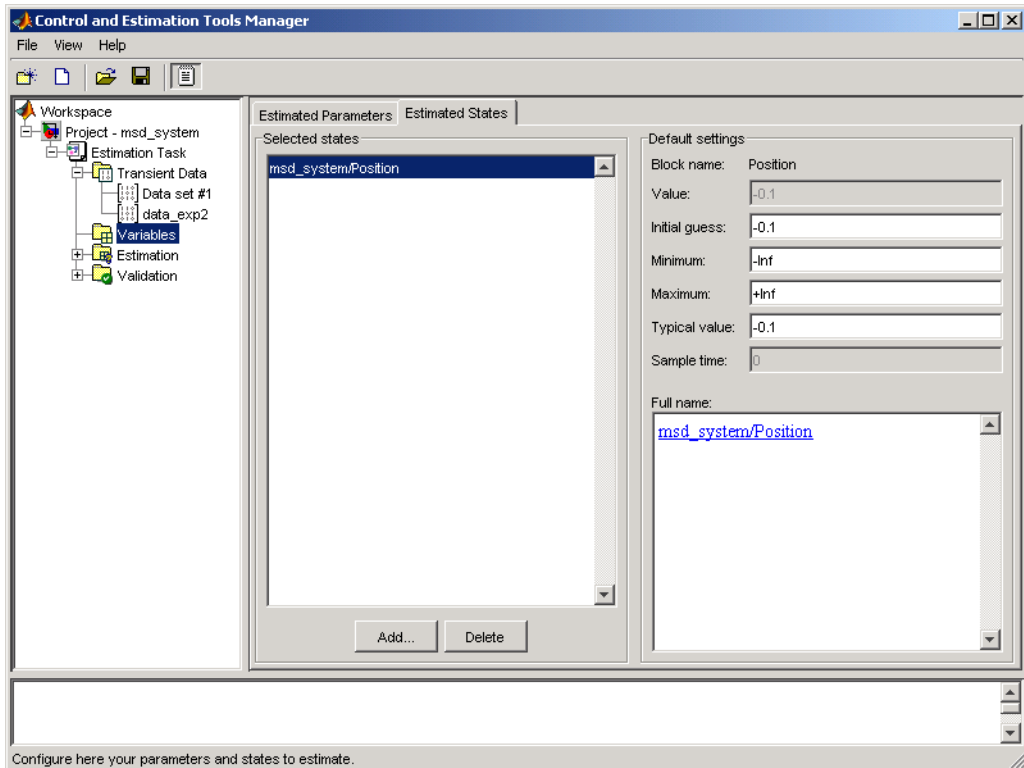


### Creating the Estimation Task

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

In the **Data Sets**, **Parameters**, and **States** panels for the **New Estimation** task, check all the boxes you see in each of the tables. Be sure to check **Position** for both data sets on the **States** panel. This directs the estimation project to estimate the initial condition for the spring's position.

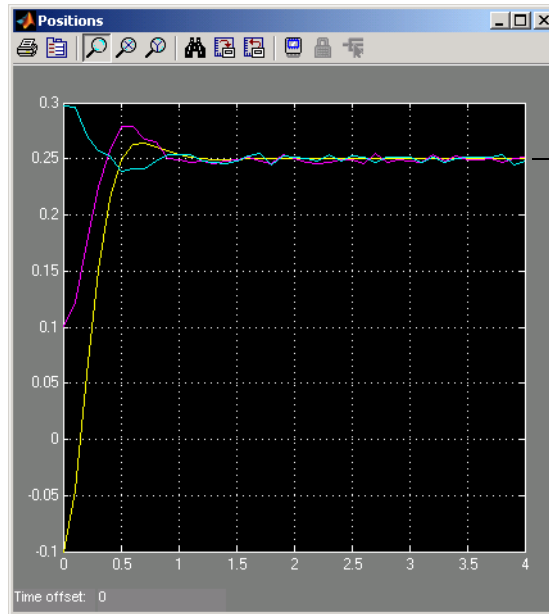
The initial position estimates for the two data sets are known to differ, but set the initial state guesses for both data sets to  $-0.1$ .



## Running the Estimation and Viewing Results

Click **Start** in the **Estimation** panel 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 ( $x_0$  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.

# Preprocessing Data

---

Simulink Parameter Estimation provides for detrending, exclusion, and filtering of data.

Why Preprocess Data? (p. 3-2)

An introduction to data preprocessing

The Data Preprocessing Tool (p. 3-3)

An introduction to a graphical user interface (GUI) for data preprocessing

Excluding Data (p. 3-5)

Various ways to exclude data from your data sets

Detrending and Filtering (p. 3-13)

Various ways to detrend and filter your data sets

Miscellaneous Data Handling (p. 3-15)

Additional features of the Data Preprocessing Tool

### Why Preprocess Data?

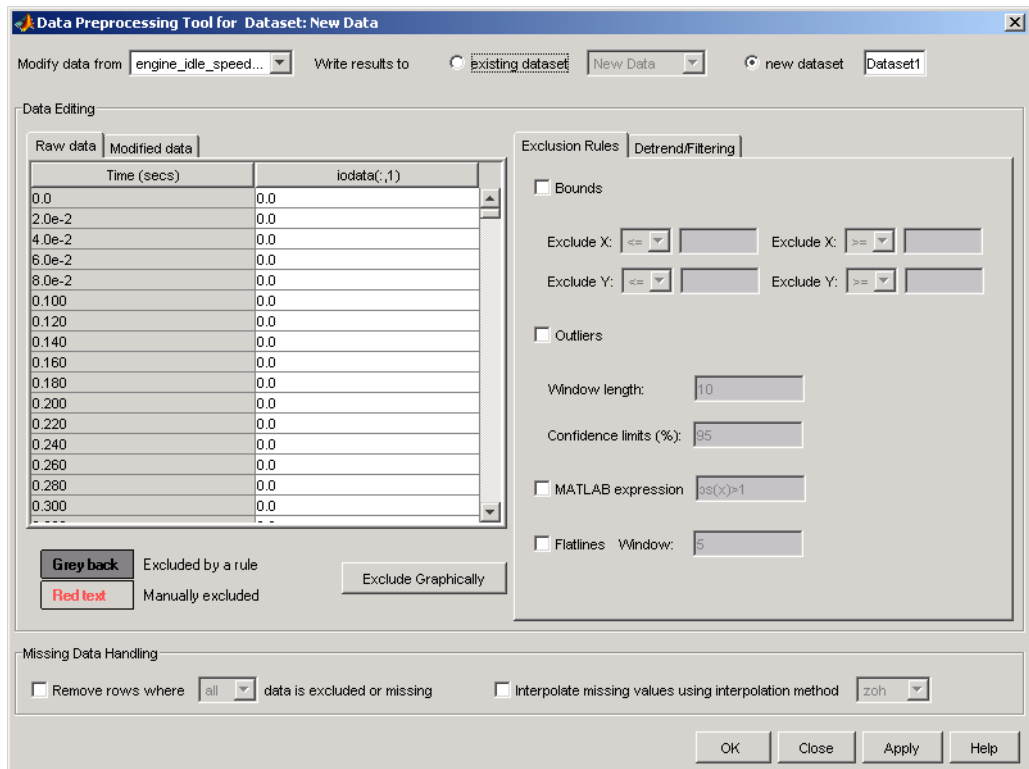
When dealing with empirical data, it is often useful to remove outliers, smooth, detrend, or otherwise treat the data to make it more tractable for analysis and estimation purposes. Simulink Parameter Estimation provides features that perform the following tasks:

- Exclusion — Eliminate outliers, represent them as NaNs, or use interpolation.
- Detrending — Remove mean values or a straight line trend.
- Filter — Smooth data using a first-order filter, an arbitrary transfer function, or an ideal filter.

Data can overwrite existing data, or be stored in a new file.

## The Data Preprocessing Tool

Simulink Parameter Estimation provides a GUI for data preprocessing, the Data Preprocessing Tool. To open it, open the Control and Estimation Tools Manager and select the data you want to modify from the **Input Data**, **Output Data**, or **State Data** panels from the Transient Data node of the workspace directory tree. Click **Pre-process** to open the Data Preprocessing Tool.



For this chapter, the data used is from the `engine_idle_speed` Simulink model. See “How Simulink Parameter Estimation Works” on page 1-5 for an overview of creating estimation projects and adding data sets.

The Data Preprocessing Tool has numerous features. With it you can

- Exclude data by selecting it with your mouse

- Exclude data graphically by selecting regions on a plot
- Exclude data by rules, such as upper or lower bounds
- Detrend data
- Filter data



## Excluding Data

There are three ways to exclude data, covered in the following sections:

- “Selecting Data for Exclusion from the Data Editing Table” on page 3-5
- “Selection Data for Exclusion from a Plot of the Data” on page 3-8
- “Selecting Data for Exclusion by a Rule” on page 3-10

The first two are by hand, the last by a rule. When you exclude data by hand (not by a rule), the excluded data is red. When you exclude data by a rule, the background color of the cell becomes gray. If a piece of data is excluded by both a rule and by hand, the data (numbers) is red, and the background is gray.

**Note** Changes in data are visible everywhere. If you use the **Data Editing** table, the results are viewable in the plot of the data, and vice versa.

### Selecting Data for Exclusion from the Data Editing Table

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

The screenshot shows the 'Data Editing' window with two tabs: 'Raw data' and 'Modified data'. The 'Raw data' tab is active, displaying a table with two columns: 'Time (secs)' and 'ioddata(:,1)'. The data rows are as follows:

Time (secs)	ioddata(:,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

Below the table, there are three buttons: 'Grey back' (Excluded by a rule), 'Red text' (Manually excluded), and 'Exclude Graphically'. The 'Red text' button is highlighted in red. A callout box points to the 'Exclude Graphically' button with the text: 'Click this button to view the data graphically.'

Another callout box points to the blue highlighted rows with the text: '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.'

There are two tables in the **Data Editing** table, **Raw Data** and **Modified Data**. The **Raw Data** panel is your working copy. For example, if you exclude rows of data in the **Raw Data** table, those rows of numbers become red in that table. By default the **Modified Data** table represents the removed rows by inserting NaNs.

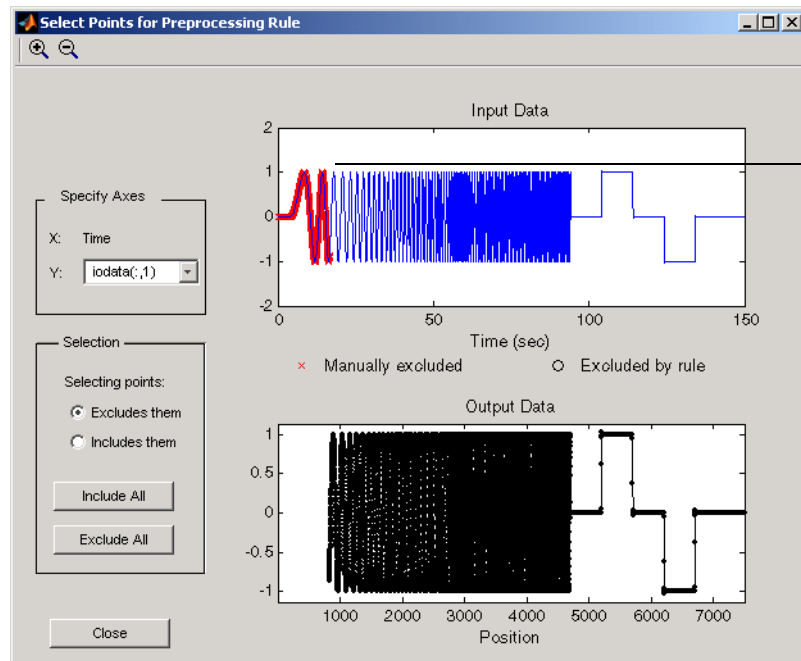
Time (secs)	iodata(:,1)*
8.760	0.979
8.780	0.976
8.800	0.972
8.820	0.969
8.840	NaN
8.860	NaN
8.880	NaN
8.900	NaN
8.920	NaN
8.940	NaN
8.960	NaN
8.980	NaN
9.0	NaN
9.20	NaN
9.40	NaN
9.60	NaN
...	...

Excluded by a rule  
 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** table, you can choose to remove the excluded data completely or interpolate it. See “Miscellaneous Data Handling” on page 3-15 for more information.

Once you've selected data for exclusion, you can view it graphically by clicking **Exclude Graphically**.

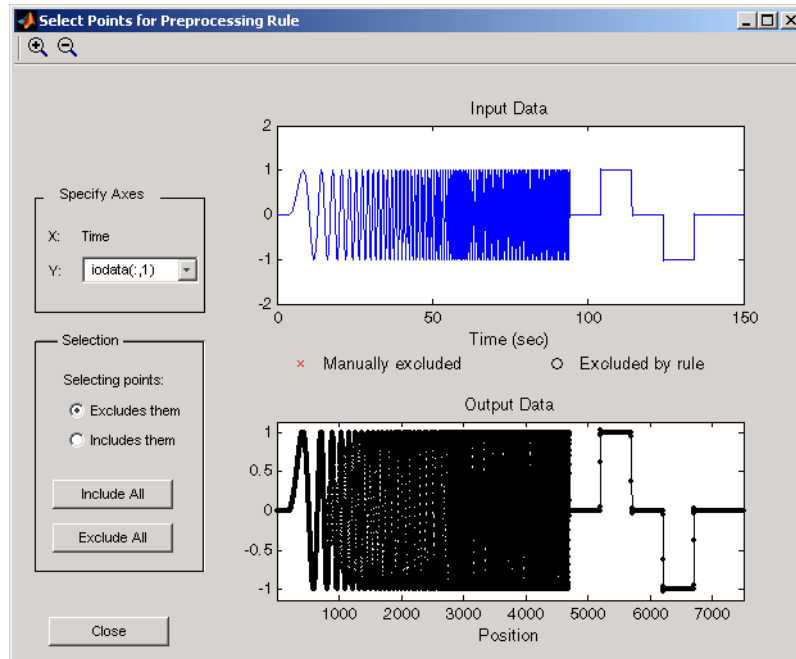


The data excluded by hand is marked in red.

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

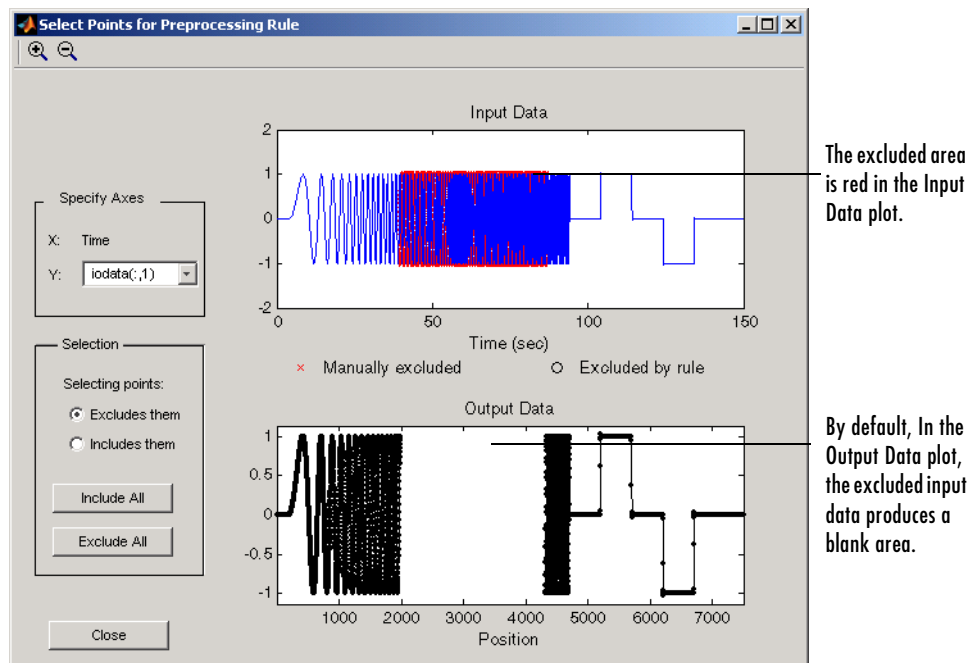
## Selection Data for Exclusion from a Plot of the Data

Another way to exclude data is to do so graphically. Click **Exclude Graphically** to open the **Select Points for Preprocessing Rule** window.



You can exclude data in much the same way as you would select a region for zooming. Place your cursor in the Input Data plot, click and drag the mouse to draw a region of exclusion.

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

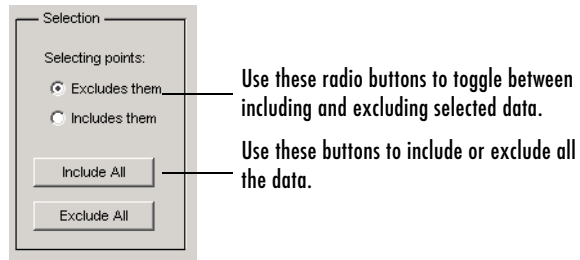


In the Output Data plot, the excluded input data produces a blank area by default. This corresponds to the NaN representation of excluded data. If you choose to interpolate or remove the excluded data, the output data shows the interpolated points.

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

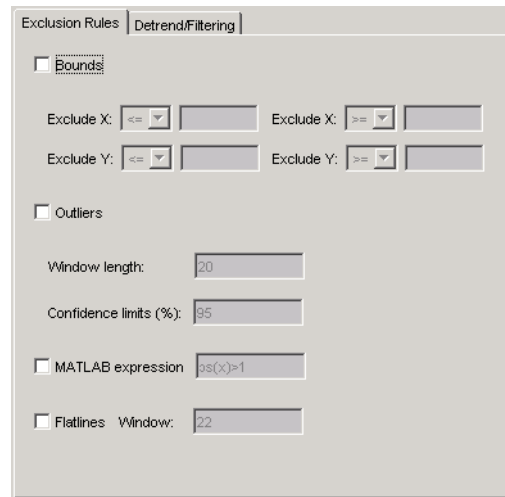
## The Selection Panel

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** panel on the left side of the **Select Points for Preprocessing Rule** window.



## Selecting Data for Exclusion by a Rule

A more precise way to exclude data is to use mathematical rules. The **Exclusion Rules** panel of Data Preprocessing Tool allows you to enter customized rules of exclusion.



These are the rules you can use to exclude data:

- Upper and lower bounds
- Outliers
- MATLAB expressions
- Flatlines

### Upper and Lower Bounds

Check the **Bounds** 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.

### Outliers

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

### MATLAB Expressions

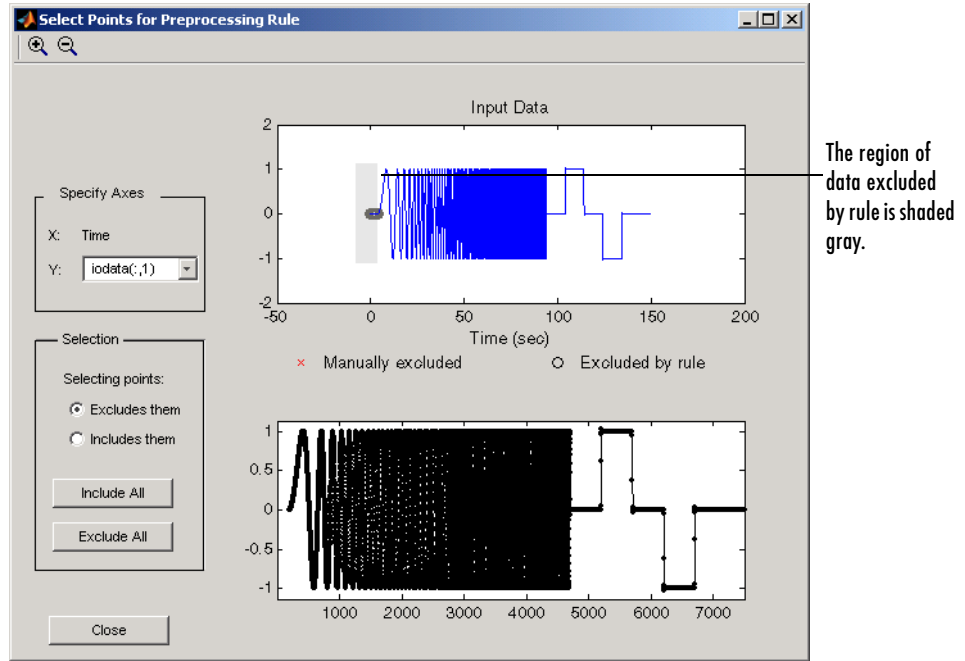
Use the **MATLAB expression** field to enter any mathematical expression needed in MATLAB code. Use *x* as the variable name in your expression for data being tested.

### 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

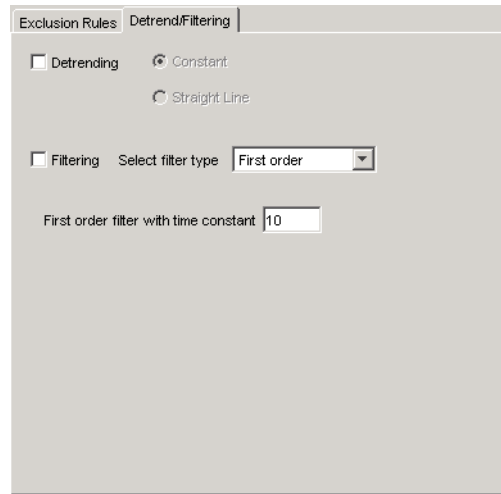
This figure shows data with a region of the x-axis excluded.





## Detrending and Filtering

You can both detrend and filter data using the **Detrend/Filtering** panel on the **Data Preprocessing Tool**.



### Detrending

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 and then removes them.

### Filtering

You have these choices for filtering your data:

- First order — A filter of the type

$$\frac{1}{\tau s + 1}$$

where  $\tau$  is the time constant that you specify in the associated field.

- **Transfer function** — A filter of the type

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

where you specify the coefficients as vectors in the associated **A** 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.

## Miscellaneous Data Handling

There are a few miscellaneous data handling features in the **Data Preprocessing Tool**.

### Handling Missing Data

You can use the Missing Data Handling panel at the bottom of the **Data Preprocessing Tool** to remove rows of data or interpolate between points to fill in missing data.



#### Removing Rows

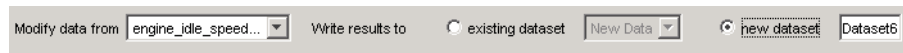
If you check **Remove rows where all/any data is excluded or missing**, the affected rows are removed from the **Modified Data** table. If you have multiple columns of data, select **all** to remove rows in which all the data is excluded. Select **any** to remove any excluded cell. In the case of one-column data, **any** and **all** are equivalent.

#### Interpolation

You have two choices if you want to interpolate data: zero-order hold (**zoh**) and linear interpolation (**Linear**). Check the **Interpolate missing values using interpolation method** box and choose which method you want from the menu. The results appear in the **Modified Data** table.

### Loading Data and Saving Modified Data Sets

At the top of the **Data Preprocessing Tool**, there is a region for selecting data sets for preprocessing, and for saving modified data sets.



When you have multiple data sets, select the one you want to preprocess from the **Modify data from** menu.

To overwrite an existing data set, select the **existing dataset** radio button and choose the data set you want to overwrite. If you want to save the data set under a new name, select the **new dataset** radio button and type the new name in the associated field.

# Managing Multiple Projects

---

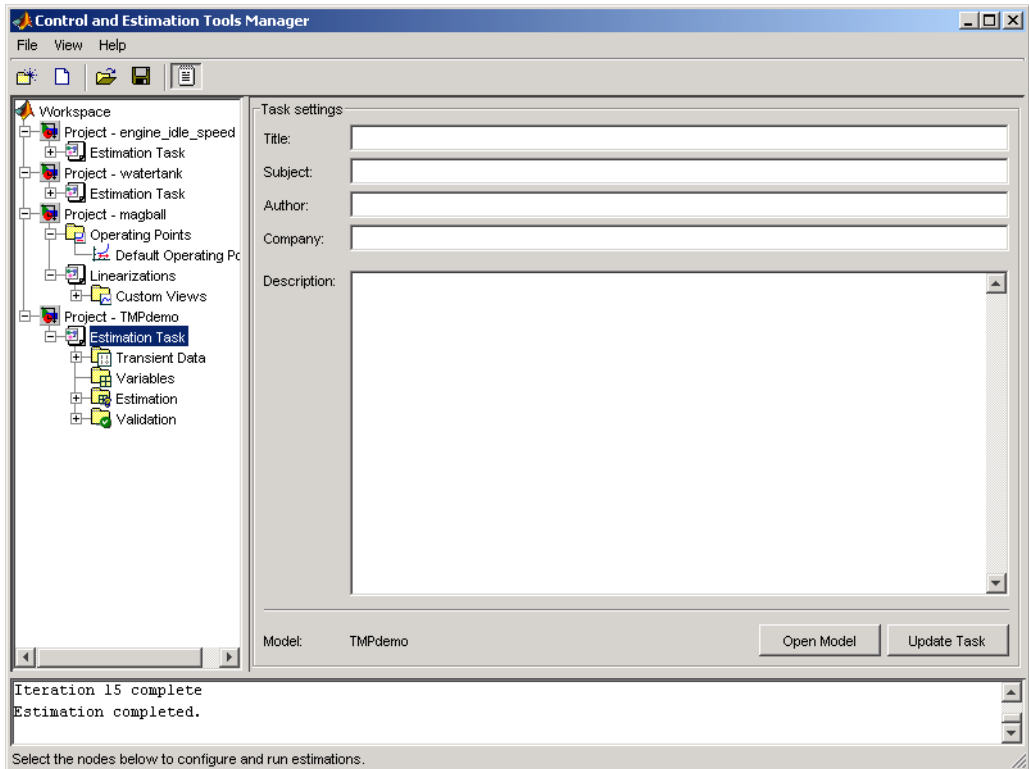
Simulink Parameter Estimation works seamlessly with other MathWorks products to perform multiple tasks on multiple projects.

Multiple Projects and Tasks (p. 4-2)	A brief discussion of handling multiple projects with multiple tasks
Saving Control and Estimation Tools Manager Projects (p. 4-3)	How to save projects for later
Opening Control and Estimation Tools Manager Projects (p. 4-4)	How to open existing projects

## 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, you can use these products to perform tasks on projects that you have created in Simulink Parameter Estimation, and vice versa.

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

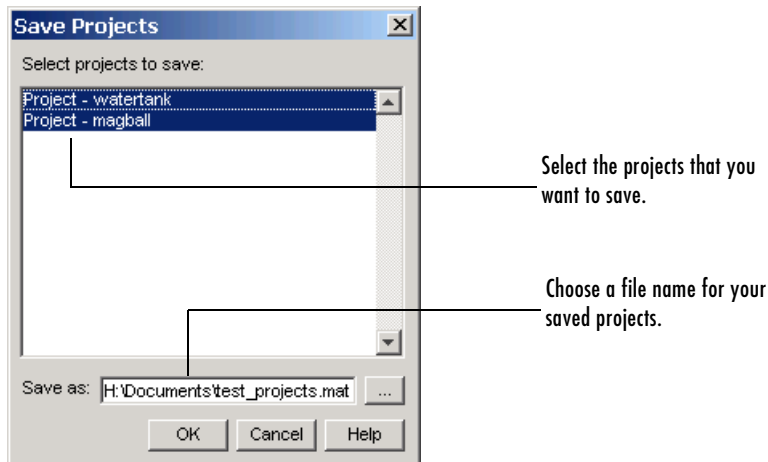


You can save projects individually, or group multiple projects together in one saved file. The next section describes how to do this.

## Saving Control and Estimation Tools Manager Projects

A Control and Estimation Tools Manager project can consist of multiple tasks including those from Simulink Control Design, Simulink Parameter Estimation, and the Model Predictive Control Toolbox. Each task contains data, objects, and results for the analysis of a particular model.

To save your projects, choose **File -> Save** in the Control and Estimation Tools Manager window.

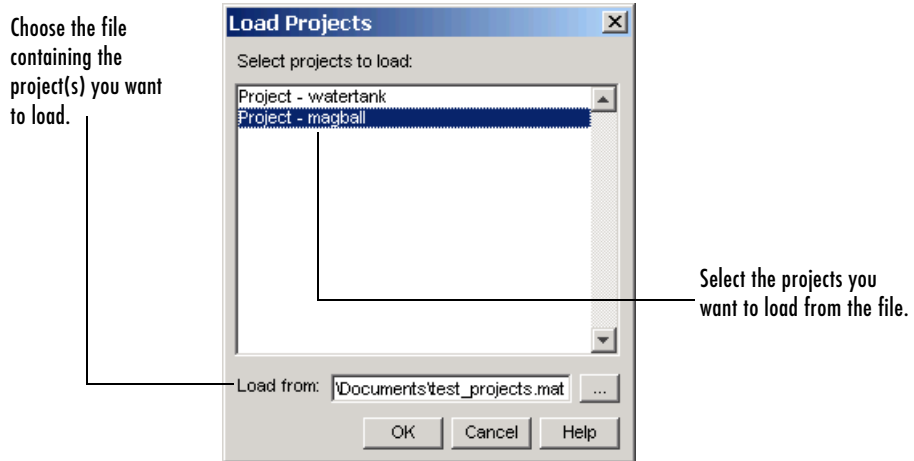


### Save Projects Dialog Box

In the **Save Projects** dialog box, select the projects that you want to save. You can save multiple projects within one file. Next, choose a directory and name for your project file by either browsing for a file or typing the full pathname and filename in the **Save as** box, and then click **OK**. The project file is saved as a MAT-file.

# Opening Control and Estimation Tools Manager Projects

To open previously saved projects, choose **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 typing the full pathname and filename in the **Load from** box. Project files are always MAT-files. The projects within this file appear in the list box. 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.



# Adaptive Lookup Tables

---

What Are Lookup Tables? (p. 5-2)

A brief description of the lookup table concept

How Adaptive Lookup Tables Work  
(p. 5-3)

More details on adaptive lookup tables

Implementation of Adaptive Lookup  
Tables (p. 5-4)

What adaptive lookup tables look like in Simulink

Example: n-D Adaptive Lookup Table  
(p. 5-8)

An example using an multidimensional adaptive lookup  
table

## What Are Lookup Tables?

Lookup tables are used to 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.

Lookup tables are usually generated by experimentally collecting or artificially creating the input and output data of a system. In general, as many indexing parameters are required 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*. So, a system with  $M$  inputs is girded by  $M$  sets of breakpoints. Given the input data, the breakpoints are then used to locate the array elements, where the output data of the system are stored. For a system with  $N$  outputs,  $N$  array elements are located and the corresponding data are stored at these locations.

Once a lookup table is created using the input and output measurements as described above, the corresponding multidimensional array of values can be used in applications without the need of remeasuring the system outputs. In fact, only the input data is required to locate the appropriate array elements in the lookup table and the approximate system output can be read 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.

## How Adaptive Lookup Tables Work

The generation of lookup tables as described above establishes a permanent and static mapping of input-output behavior of a physical system. Statically defined lookup tables cannot accommodate the *time-varying* behavior (characteristics) of a physical plant. On the other hand, it is well known that the behavior of actual physical systems often vary with time due to wear, environmental conditions, and manufacturing tolerances. Under 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*, on the other hand, 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.

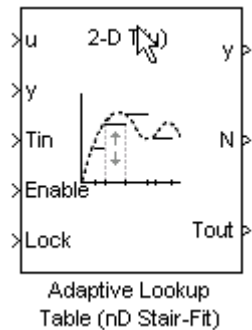
The adaptive lookup table receives the input and output measurements of a plant's behavior, which are then used to dynamically create and update the content of the underlying lookup table. In addition to requiring the input data to create the lookup table, the adaptive lookup table also uses the output data of the plant to recalculate the table values. As an example, the output data of the plant can be collected by placing sensors at appropriate locations in a physical system.

The input measurements are used to locate the array elements by comparing these input values with the breakpoints defined for each indexing variable. Next, the output measurements are used 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 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.

## Implementation of Adaptive Lookup Tables

The MathWorks implements adaptive lookup tables as Simulink blocks. These blocks create multidimensional lookup tables from measured or simulated data. The inputs and outputs of a n-D Adaptive Lookup Table block with two inputs are shown below.



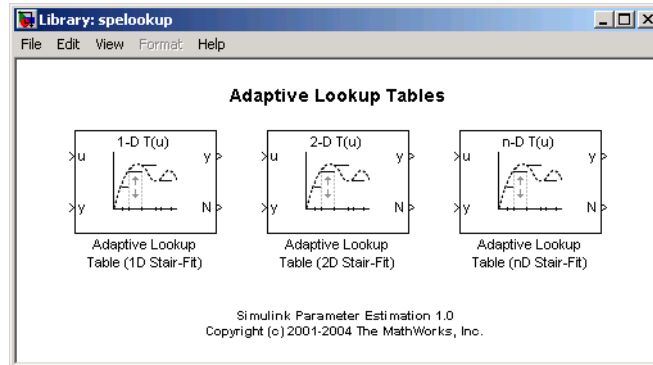
### Adaptive Lookup Table Block Showing Inputs and Outputs

The following are descriptions of the input and output parameters:

- The *inputs*  $u$  and  $y$  are the coordinate data and system output measurements, respectively. For example, if you want to create a lookup table to model the behavior of an engine's efficiency as a function of engine rpm and manifold pressure,  $u = [\text{rpm}, \text{pressure}]$  and  $y = [\text{efficiency}]$ .
- The *initial table* data may be entered either as a dialog parameter (by double-clicking on the block) or as an input port (i.e., the input port  $T_{in}$  in the figure). You can start, stop, and reset the adaptation through the *Enable* input port.
- The *outputs* of the block include the value of the currently adapted table cell ( $Y$ ), the number ( $N$ ) of that cell (which may be specified through the block dialog), and if required, the whole adapted table data ( $T_{out}$ ).

## Adaptive Lookup Table Library

Three adaptive lookup tables are available in Simulink Parameter Estimation.

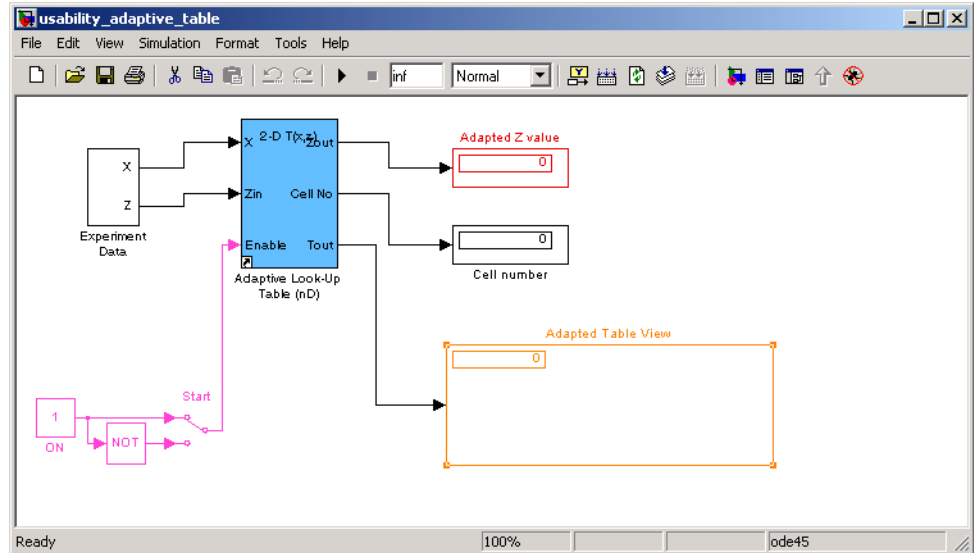


The three blocks are

- “Adaptive Lookup Table (1D Stair-Fit)” on page 7-2 — One-dimensional adaptive lookup
- “Adaptive Lookup Table (2D Stair-Fit)” on page 7-4 — Two-dimensional adaptive lookup
- “Adaptive Lookup Table (nD Stair-Fit)” on page 7-7 — Multidimensional adaptive lookup (use this for dimension 3 or higher)

## Using Adaptive Lookup Tables in Simulink Models

A typical Simulink diagram using an adaptive lookup table block is shown below.



### Simulink Diagram Using an Adaptive Lookup Table

In this figure, the Experiment Data block imports a set of experimental data into the Simulink environment through MATLAB workspace variables. The initial table is specified through a constant matrix block. 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.

## Real-Time Lookup Tables

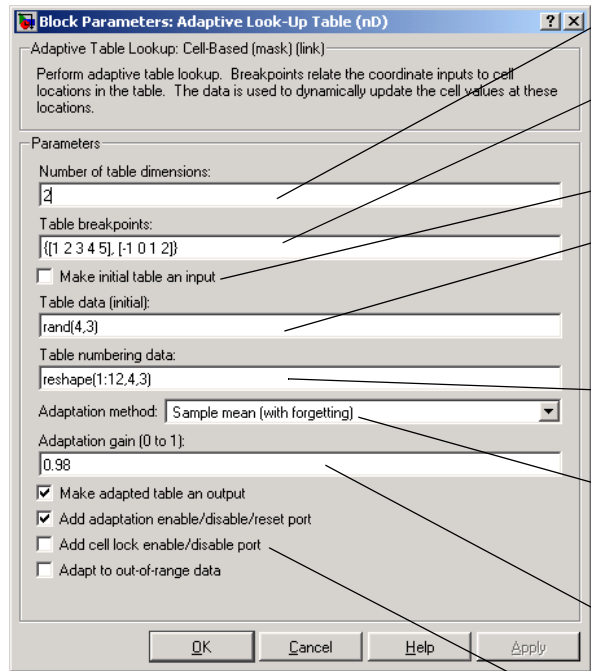
You can use experimental data from sensor measurements collected by running various tests on a system in real time. The measured data is then sent to the adaptive table block to generate a lookup table describing the relation between the system inputs and output.

The adaptive lookup table block may also be used in a real-time environment, where some time-varying properties of a system need to be captured. This can be done by generating C code using Real-Time Workshop<sup>®</sup>, which can then be

run in an xPC or dSpace environment. Since the adaptation may be started, stopped, or reset if desired, some logic may be used to adapt the table data only when it is desired. The Cell No output, and the Enable and Lock inputs facilitate this process. The Enable input is used to start and stop the adaptation, while the Lock input is used to update only one of the table cells. The Lock input combined with some logic using the Cell No output provide the means for updating only the desired table cells during a simulation run.

## Setting Adaptive Lookup Table Parameters

Adaptive lookup tables are highly configurable, as shown below.



The dialog box 'Block Parameters: Adaptive Look-Up Table (nD)' contains the following parameters and options:

- Adaptive Table Lookup: Cell-Based (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:** {[1 2 3 4 5], [-1 0 1 2]}
  - Make initial table an input
  - Table data (initial):** rand(4,3)
  - Table numbering data:** reshape(1:12,4,3)
  - Adaptation method:** Sample mean (with forgetting)
  - Adaptation gain (0 to 1):** 0.98
  - Make adapted table an output
  - Add adaptation enable/disable/reset port
  - Add cell lock enable/disable port
  - Adapt to out-of-range data

Callout descriptions:

- The number of dimensions for the adaptive lookup table.
- A set of one-dimensional vectors that contains possible block input values for the input variables.
- Use this port to input table data.
- 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.
- Number values assigned to cells. This vector must be the same size as the table data array, and each value must be unique.
- Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data.
- A number between 0 and 1 that regulates the weight given to new data during the adaptation.
- Check boxes for customizing the I/O channels of the block and allowing adaptation to out-of-range data.

### n-D Adaptive Lookup Table Dialog Box

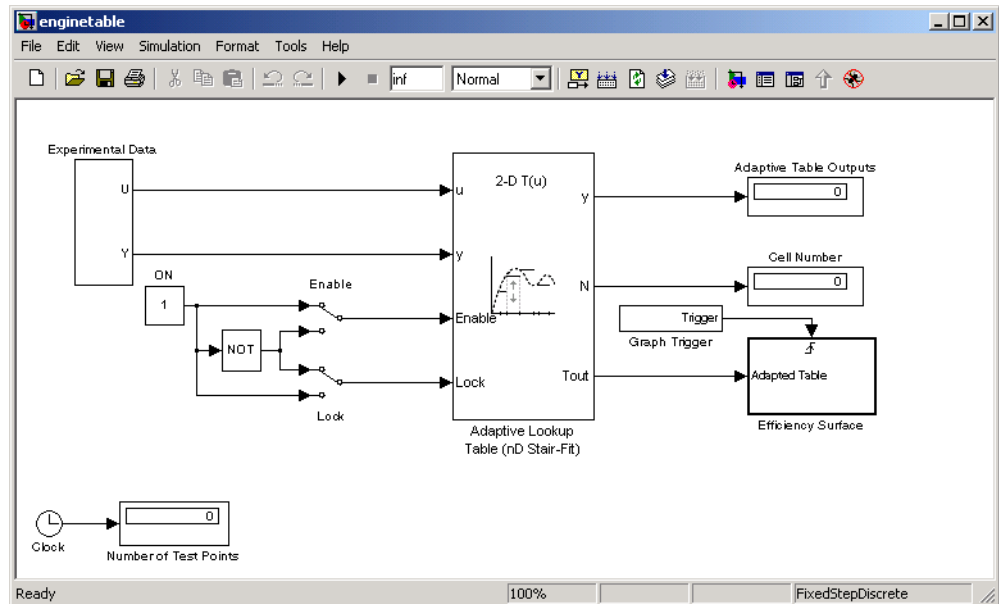
For details on how to set these parameters, see the individual reference pages.

## Example: n-D Adaptive Lookup Table

This example shows an n-D adaptive lookup table at work and includes many of the key features associated with adaptive lookup tables. Type

`enginetable`

at the MATLAB prompt to open this model.




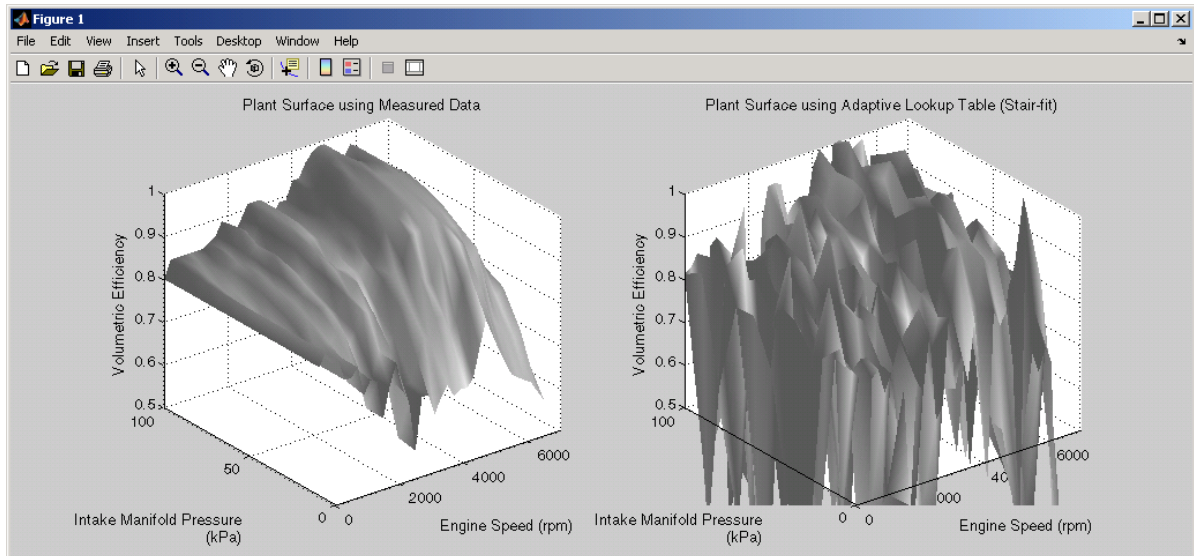
This model has several key features:

- **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. This is useful if you have one section in your data that is highly erratic or otherwise difficult for the algorithm to handle.
- **Output** — Adaptive lookup breakpoints are the output data.

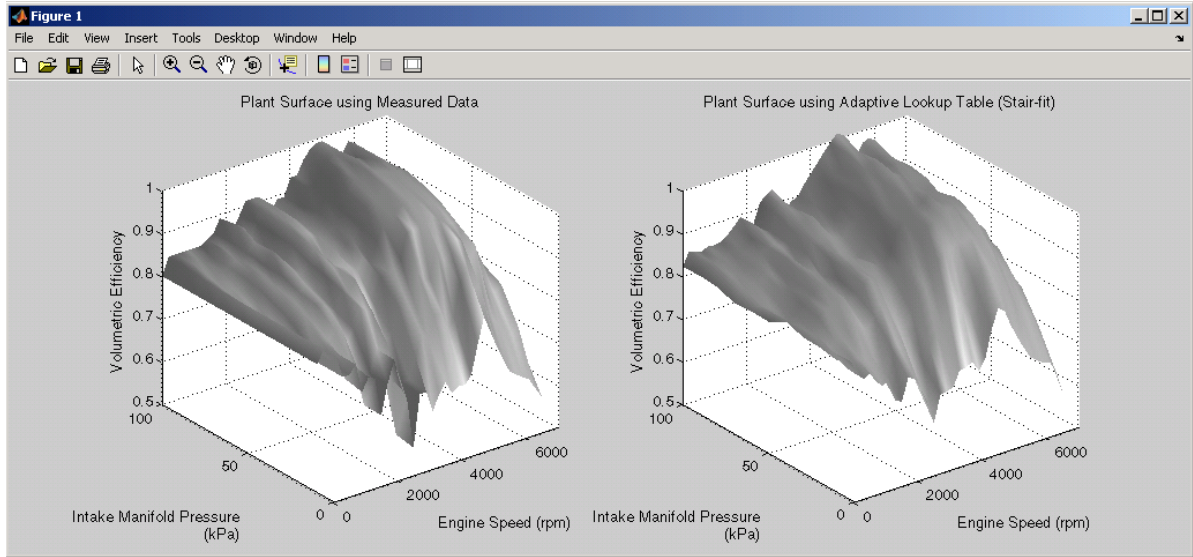


## Running the Example

To start the enginetable simulation, pull down the **Simulation** menu and choose the **Start** command or, on Microsoft Windows, click the Start button  on the Simulink toolbar. The simulation begins by populating the adaptive lookup table with random data. This figure shows the input and adaptive data side by side.



As the simulation progresses, the surface on the right adapts to match the measured input data. This figure shows the final adaptation.



The fit is quite good. Try using the enable and lock features to see how they change the adaptation.

# Estimating from the Command Line

---

Simulink Parameter Estimation provides an object-oriented command-line API for the estimation problem.

Introduction (p. 6-2)

A brief discussion of the estimation problem in an object-oriented context

Example: Estimating Parameters and Initial Conditions of the F14 Model (p. 6-4)

How to create and simulate an estimation project from the command line

Creating and Customizing Estimation Projects (p. 6-12)

Using properties and methods to specify features of the estimation project

Creating Transient Data Objects (p. 6-13)

How to instantiate and use transient data objects, which contain input and output data

Creating State Data Objects (p. 6-17)

How to instantiate and use state data objects, which contain information about known states in your Simulink model

Creating Transient Experiment Objects (p. 6-20)

How to instantiate and use parameter objects, which maintain data about parameters you want to estimate

Creating Parameter Objects (p. 6-23)

How to instantiate and use state objects, which maintain data about the block states you want to estimate

Creating State Objects (p. 6-27)

How to instantiate and use transient experiment objects

Creating Estimation Objects (p. 6-31)

How to instantiate and use estimation objects, which coordinate your model, experiment, parameter, and state objects

# Introduction

In addition to the Control and Estimation Tools Manager, Simulink Parameter Estimation provides a collection of functions for performing 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 Simulink Parameter Estimation GUI, Simulink Parameter Estimation creates 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, for example, you may want to test large numbers of models.

---

**Note** Simulink Parameter Estimation uses MATLAB objects to perform estimation tasks. This chapter discusses what you need to know about object-oriented programming for using Simulink Parameter Estimation, but see “MATLAB Classes and Objects” in the MATLAB Programming documentation for a detailed description of the rules of object-oriented programming in MATLAB.

---

The command-line interface for Simulink Parameter Estimation requires a Simulink model as a starting point for analysis and estimation. Once you have selected a candidate model, the estimation process consists of these steps:

- Defining experiments consisting of empirical data sets and the operating conditions and/or initial conditions of your model
- Selecting the variables and states to be estimated
- Performing the estimation
- Reviewing the results and iterating as necessary
- Validating estimation results

The following sections discuss these topics:

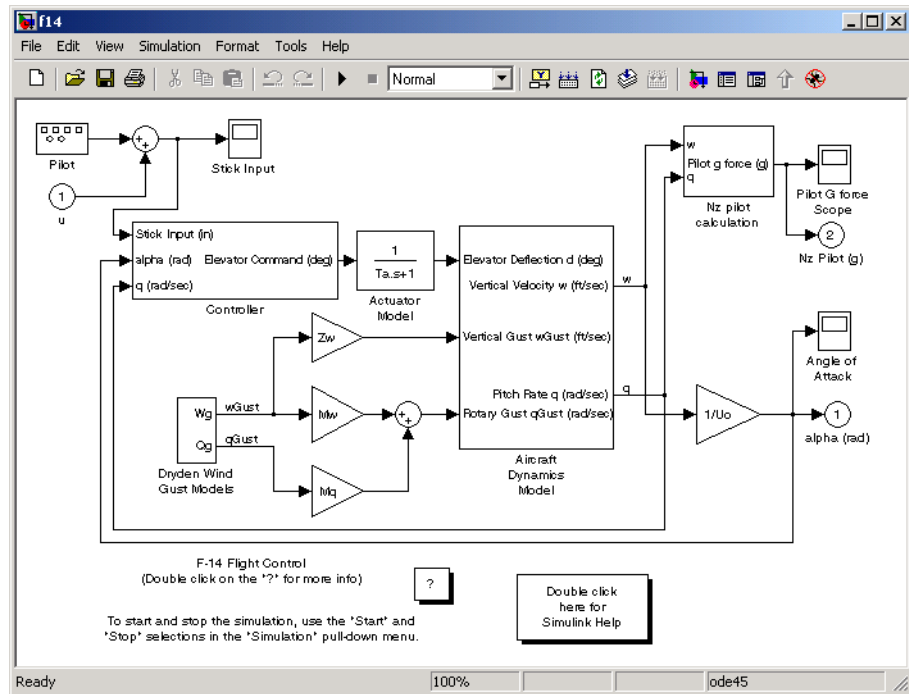
- “Example: Estimating Parameters and Initial Conditions of the F14 Model”  
— How to perform the estimation process using command-line functions
- “Creating and Customizing Estimation Projects” — How to use methods and properties to customize your estimation project’s features

## Example: Estimating Parameters and Initial Conditions of the F14 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 figure below shows the f14 model.



### F14 Fighter 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; see “Creating and Customizing Estimation Projects” on page 6-12 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  $T_a$ , the actuator time constant, and  $Z_d$  and  $M_d$ , the vertical velocity and pitch rate gains, respectively. Then use the code below to run the Simulink `f14` model. Note that this is standard Simulink code and does not involve Simulink Parameter Estimation in any way. See `sim` in the Simulink Reference documentation for information about running Simulink models from the MATLAB command line.

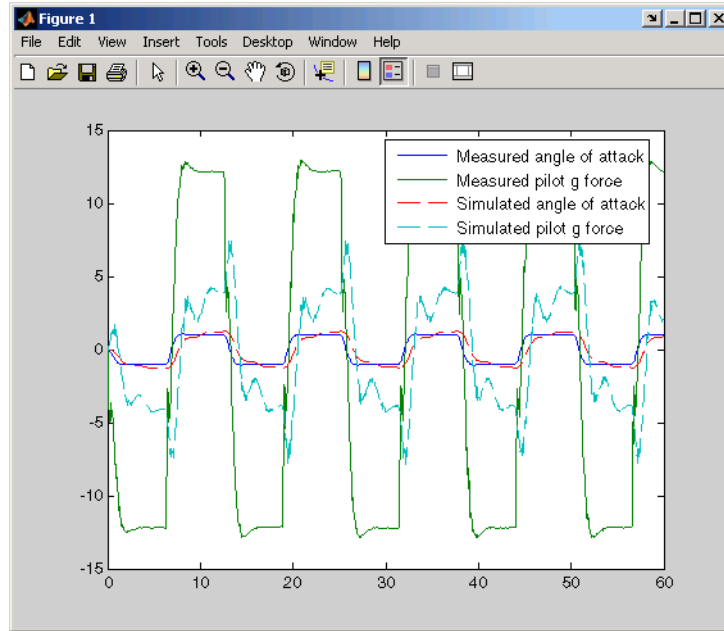
```
%% Open the model and load experimental data.
open_system('f14')
load f14_estim % Load empirical I/O data.

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

% Aircraft dynamic model parameters (ideal:  $M_d = -6.8847,$ 
%  $Z_d = -63.998$ )
 $M_d = -1; Z_d = -80;$ 

%% 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');
```

This figure appears.



### Baseline Comparison of Measured and Simulated F14 I/O Data

As you can see, the measured and simulated data are a poor match. The rest of this section describes how to estimate values for  $T_a$ ,  $Z_d$ , and  $M_d$  that result in a better match of data sets.

### Creating a Transient Experiment Object

Once you have a model, and have identified 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');
```

MATLAB responds with 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

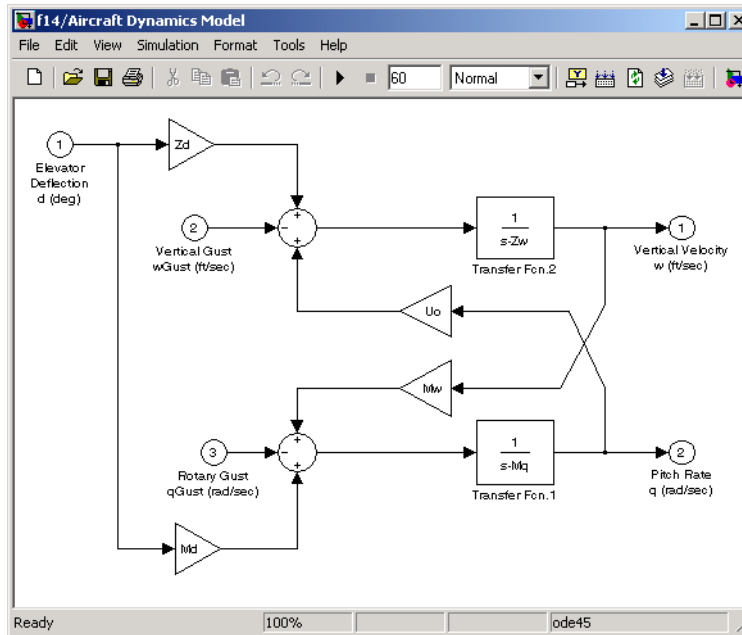
Once you've created a `ParameterEstimator` object, assign input and output experimental (i.e., empirical) data.

```
%% 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);
```

## 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 Subsystem

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;
```

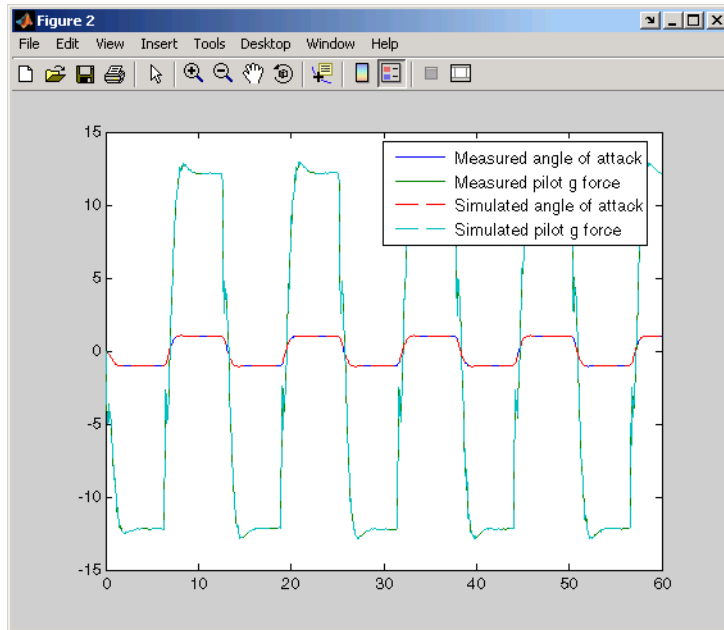
## Creating an Estimation Object and Running the Estimation

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)
```

MATLAB responds with

(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:

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
```

## Creating and Customizing Estimation Projects

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

- “Creating Transient Data Objects” on page 6-13
- “Creating State Data Objects” on page 6-17
- “Creating Transient Experiment Objects” on page 6-20
- “Creating Parameter Objects” on page 6-23
- “Creating State Objects” on page 6-27
- “Creating Estimation Objects” on page 6-31

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

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

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

This table lists the properties of the transient data object and the associated input parameters.

#### 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	<p>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:</p> <ul style="list-style-type: none"> <li>• Scalar or vector-valued data. The data is of the form <math>N_s m</math>, where <math>N_s</math> is the number of data samples, and <math>m</math> is the number of channels in the signal.</li> <li>• Multidimensional data (matrix and higher dimensions). The data is of the form <math>m_1 \dots m_n N_s</math>, where <math>N_s</math> is the number of data samples, and <math>m_i</math> is the number of channels in the <math>i</math>th dimension of the signal.</li> <li>• For missing or unspecified data, NaNs are used.</li> </ul>
Ts, Tstart, Tstop	<p>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</p> $Tstop = Tstart + Ts * (Ns - 1)$ $Time = Tstart : Ts : Tstop$ <p>For nonuniform time data, Ts is set to NaN, and the start and stop times are calculated from the time vector.</p>



**Transient Data Object Properties (Continued)**

<b>Property</b>	<b>Description</b>
Time	<p>The time data in column vector format. The length of Time must be consistent with the number of samples in Data.</p> <p>For a nonuniformly spaced Time vector, its length should match the length of Data.</p> <p>Otherwise, Time is automatically adjusted based on the length of Data.</p> <p>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</p> <ul style="list-style-type: none"> <li>• Modifying Time sets           <ul style="list-style-type: none"> <li>Ts = NaN</li> <li>Tstart = Time(1)</li> </ul> </li> <li>• If the time vector is uniformly spaced, a sample time Ts is calculated.</li> <li>• Modifying Tstart translates time forward or backward.</li> <li>• Modifying Ts sets Time = [] internally and generates it when required by the simulation.</li> </ul>
Weight	<p>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.</p>
InterSample	<p>Interpolation method between samples can be zero-order hold (zoh) or first-order hold (foh). This property is used for data preprocessing.</p>

## Modifying Transient Data Object Properties

Once 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 Dimensions property.

### Example: Assigning Input Port Data

To assign data to an input port with 2x3 port dimensions, use

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

MATLAB responds with

```
(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

The description of some of the important methods is given below.

- `select` — Used to extract 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

The `ParameterEstimator.StateData` object defines the 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.

This is the syntax for creating this object:

```
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

The description of some of the important properties is given in the table below.

#### 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)**

<b>Property</b>	<b>Description</b>
Data	<p>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 <code>Dimensions</code> 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:</p> <ul style="list-style-type: none"> <li>• If <code>Data</code> is not empty, use it when forming the state vector.</li> <li>• If <code>Data</code> 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.</li> <li>• 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.</li> </ul>
Ts	<p>Sampling time of discrete blocks. Set to 0 for continuous blocks. This property is read only and is currently used for information only.</p>
Domain	<p>String to hold the physical domain of the block. Used for <code>SimMechanics</code> or <code>SimPowerSystems</code> blocks with states.</p>

## 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

Once a state data object is created, its properties can be modified using the following 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

The description of some of the important methods is given below:

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

## Creating Transient Experiment Objects

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

The syntax to create a transient experiment object is

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

h = ...
ParameterEstimator.TransientExperiment('model', hIn, hOut, hIc);
```

The second constructor is used when data objects are available. An empty argument in these constructors (`[]`) means the default behavior, which is to use *no* I/O ports or states depending on the position of the empty argument.

## Properties of Transient Experiment Objects

The description of some of the important properties is given in the table below.

### Transient Experiment Object Properties

Property	Description
Model	Simulink model with which this experiment is associated
InputData, OutputData	<p>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:</p> <ul style="list-style-type: none"> <li>• For input ports, assign zeros to these ports/channels during simulation.</li> <li>• For output ports, don't use these ports/channels in the cost function.</li> </ul>

**Transient Experiment Object Properties**

Property	Description
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 below:

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



## Creating Parameter Objects

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

The description of some of the important properties of parameter objects is given in the table below.

#### 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.

**Parameter Object Properties (Continued)**

<b>Property</b>	<b>Description</b>
Estimated	<p>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:</p> <ul style="list-style-type: none"> <li>• The elements of Value is estimated if the corresponding elements in Estimate are set to true. The result is stored in the Value property.</li> <li>• 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.</li> </ul> <p>This property is set to false by default, meaning that the parameter value is not estimated.</p>
InitialGuess	<p>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.</p> <p>Depending on the value of the elements of the Estimated property, the behavior of the corresponding elements of InitialGuess is as follows:</p> <ul style="list-style-type: none"> <li>• If any element in Estimated is set to true, then the corresponding element of InitialGuess is used to initialize the workspace parameter during estimations.</li> <li>• If any element in Estimated is set to false, then the corresponding element of InitialGuess is not used in any way.</li> </ul>

**Parameter Object Properties (Continued)**

Property	Description
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**

Once a parameter object is created, its properties can be modified using the following 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

The description of some of the important methods is given below:

- `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 in some way. If the size of `Value` property changes, then the other properties are reset to their default values.

## Creating State Objects

One @State object corresponds to each Simulink block with states in your 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

The description of some of the important properties of state objects is given in the table below.

#### 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	<p>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:</p> <ul style="list-style-type: none"> <li>• The elements of Value are estimated if the corresponding elements in Estimate are set to true. The result is stored in the Value property.</li> <li>• 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.</li> </ul> <p>This property is set to false by default, meaning that the state value is not estimated.</p>
InitialGuess	<p>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.</p> <p>Depending on the value of the elements of the Estimated property, the behavior of the corresponding elements of InitialGuess is as follows:</p> <ul style="list-style-type: none"> <li>• If any element in Estimated is set to true, then the corresponding element of InitialGuess is used to initialize the state during estimations.</li> <li>• If any element in Estimated is set to false, then the corresponding element of InitialGuess is not used in any way.</li> </ul>
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)
```

MATLAB returns

(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**

Once a state object is created, its properties can be modified using the following 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**

The description of some of the important methods is given below:

- `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 in some way. If the size of `Value` property changes, then the other properties are reset to their default values.



## Creating Estimation Objects

The @Estimation object is the coordinator of the model, experiment, and parameter objects.

### Constructor

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

The description of some of the important properties of estimation objects is given in the table below.

#### 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.

**Estimation Object Properties (Continued)**

Property	Description
SimOptions	Same as simset structure. This property is initialized to <code>simget(this.Model)</code> .
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 <pre>this.EstimInfo = struct( 'Cost', [],...                         'Covariance', [],...                         'FCount', [],...                         'FirstOrd', [],...                         'Gradient', [],...                         'Iteration', [],...                         'Procedure', [],...                         'StepSize', [],...                         'Values', [] );</pre>

**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
```

MATLAB returns

```
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

Once an estimation object is created, its properties can be modified using the following 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

The description of some of the important methods is given below:

- `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 in some way.



# Block Reference

---

Simulink Parameter Estimation includes three blocks that instantiate adaptive lookup tables in Simulink models.

Adaptive Lookup Table (1D Stair-Fit)    One-dimensional adaptive lookup tables  
(p. 7-2)

Adaptive Lookup Table (2D Stair-Fit)    Two-dimensional adaptive lookup tables  
(p. 7-4)

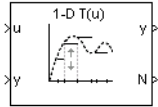
Adaptive Lookup Table (nD Stair-Fit)    n-dimensional adaptive lookup tables  
(p. 7-7)

# Adaptive Lookup Table (1D Stair-Fit)

## Purpose

Perform a one-dimensional adaptive table lookup

## Description



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, *ydata*, of your system to do the adaptations.

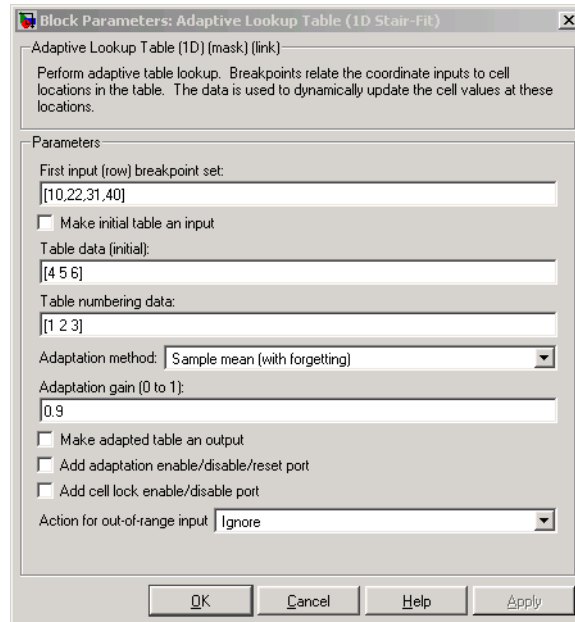
Each indexing parameter *U* may take a value within a set of adapting data points, which are called *breakpoints*. Two breakpoints in each dimension define a *cell*. The set of all breakpoints in one of the dimensions defines a *grid*. In the one-dimensional case, each cell has two breakpoints, and the cell is a line segment.

You can use the Adaptive Lookup Table (1D Stair Fit) to model time-varying systems.

## Data Type Support

Doubles only

## Dialog Box



# Adaptive Lookup Table (1D Stair-Fit)

---

## **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 box forces the Adaptive Lookup Table (1D Stair-Fit) block to ignore the **Table data (initial)** parameter. Instead, a new port appears with **Tin** next to it. Use this port to input 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.

## **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 box creates an additional output port for the adapted table.

## **Add adaptation enable/disable/reset port**

Add an input port that enables, disables, or resets the adaptive lookup table. 0 = disable; 1 = enable; 2 = reset to initial table data.

## **Add cell lock enable/disable port**

A port that provides the means for updating only specified cells during a simulation run. 0 = unlock; 1 = lock current cell.

## **Action for out-of-range input**

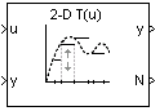
Ignore or Adapt by extrapolating beyond the extreme breakpoints.

# Adaptive Lookup Table (2D Stair-Fit)

## Purpose

Perform two-dimensional adaptive table lookup

## Description

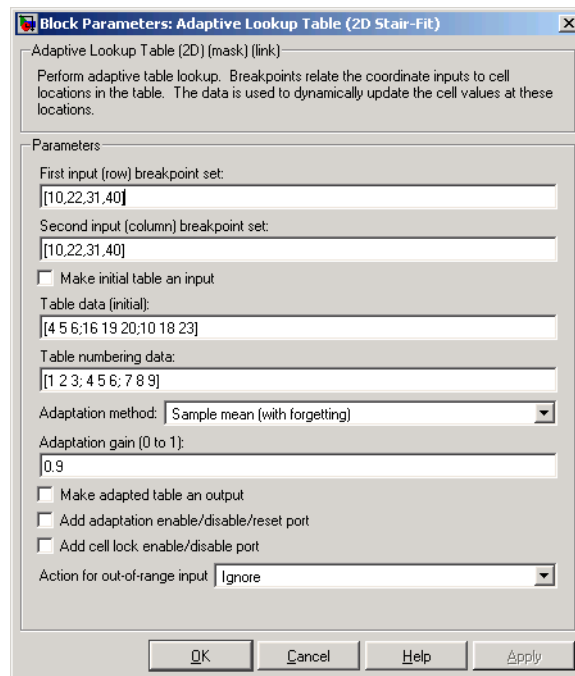


The Adaptive Lookup Table (2D Stair-Fit) block creates a two-dimensional adaptive lookup table by dynamically updating the underlying lookup table. The block uses the outputs (*ydata*) of your system to do the adaptations.

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) to model time-varying systems.

## Dialog Box





# Adaptive Lookup Table (2D Stair-Fit)

---

## **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 box forces the Adaptive Lookup Table (2D Stair-Fit) block to ignore the **Table data (initial)** parameter. Instead, a new port appears with  $T_{in}$  next to it. Use this port to input 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.

## **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 box creates an additional output port for the adapted table.

## **Add adaptation enable/disable/reset port**

Add an input port that enables, disables, or resets the adaptive lookup table.

# Adaptive Lookup Table (2D Stair-Fit)

---

## **Add cell lock enable/disable port**

A port that provides the means for updating only specified cells during a simulation run.

## **Action for out-of-range input**

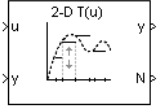
Ignore or Adapt by extrapolating beyond the extreme breakpoints.

# Adaptive Lookup Table (nD Stair-Fit)

## Purpose

Create an adaptive lookup table of arbitrary dimension

## 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.

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) to model time-varying systems.

## Dialog Box

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:

Table breakpoints (cell array):

Make initial table an input

Table data (initial):

Table numbering data:

Adaptation method:

Adaptation gain (0 to 1):

Make adapted table an output

Add adaptation enable/disable/reset port

Add cell lock enable/disable port

Action for out-of-range input:

OK Cancel Help Apply

# Adaptive Lookup Table (nD Stair-Fit)

---

## 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 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 box forces the Adaptive Lookup Table (nD Stair-Fit) block to ignore the **Table data (initial)** parameter. Instead, a new port appears with  $T_{in}$  next to it. Use this port to input 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.

## 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 box creates an additional output port for the adapted table.

# Adaptive Lookup Table (nD Stair-Fit)

---

## **Add adaptation enable/disable/reset port**

Add an input port that enables, disables, or resets the adaptive lookup table.

## **Add cell lock enable/disable port**

A port that provides the means for updating only specified cells during a simulation run.

## **Action for out-of-range input**

Ignore or Adapt by extrapolating beyond the extreme breakpoints.

# Adaptive Lookup Table (nD Stair-Fit)

---

## A

- adaptive lookup tables 5-2
- adding data sets 1-18

## C

- command-line estimation 6-2
- Controls and Estimation Tools Manager 1-8

## D

- data
  - detrending 3-13
  - exclusion 3-5
  - filtering 3-13
  - preprocessing 3-3
- Data Import dialog box 1-10
- data sets
  - adding 1-18
- detrending data 3-13
- display options for estimation 1-23

## E

- estimation
  - display options 1-23
  - example of command-line estimation 6-4
  - from the command line 6-2
  - running 1-26
  - selecting parameters 1-14
  - selecting states 1-15
  - setting up a project 1-18
  - specifying initial conditions 2-1
- excluding data 3-5

## F

- filtering data 3-13

## I

- importing
  - initial conditions 1-12
  - transient data 1-9
- initial conditions
  - example of estimating 2-3
  - importing 1-12
  - specifying for estimation 2-1
- initial guesses 1-16

## L

- lookup tables
  - adaptive 5-2

## M

- multiple projects and tasks 4-2

## O

- optimization
  - setting options for 1-39

## P

- parameters
  - selecting for estimation 1-14
  - specification of 1-20
- preprocessing data 3-3

- project
  - definition of 1-5
- projects
  - saving 4-3

## **R**

- running an estimation 1-26

## **S**

- saving projects 4-3
- selecting views 1-24
- setting optimization options 1-39
- setting options for simulation 1-39
- setting upper/lower bounds 1-16
- simulation
  - setting options for 1-39
- specifying parameters 1-20
- states
  - selecting for estimation 1-15

## **T**

- transient data
  - importing 1-9

## **U**

- upper/lower bounds
  - setting 1-16

## **V**

- views
  - selecting 1-24