Simulink® Parameter Estimation 1
User's Guide

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
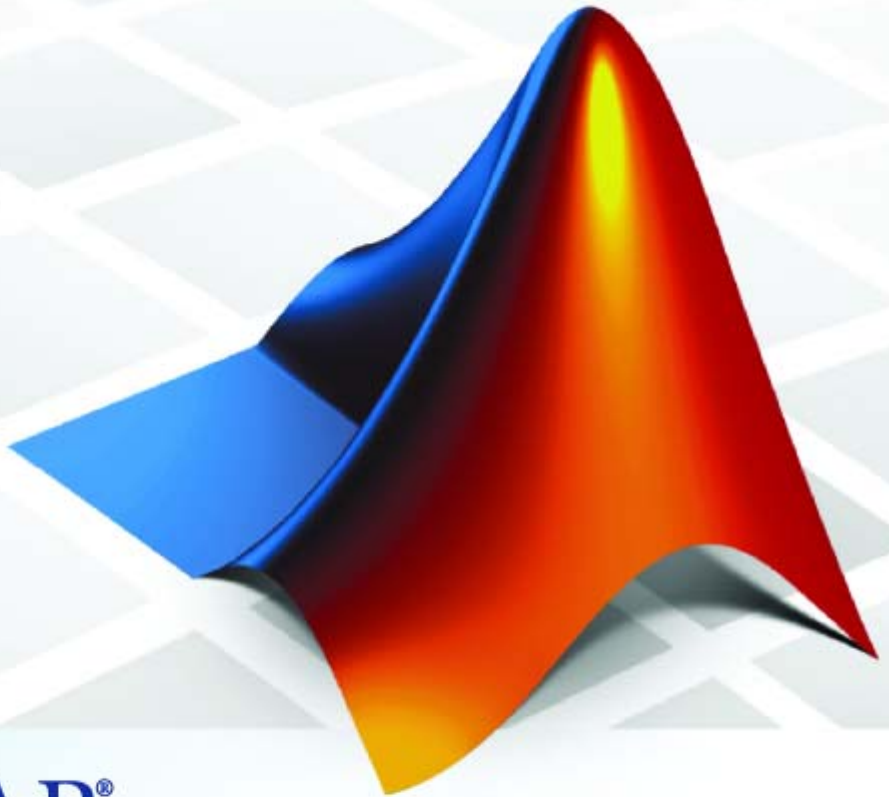
How to Contact The MathWorks

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	Technical Support
@	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink Parameter Estimation User's Guide

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

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

Patents

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

Revision History

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)
September 2005	Online only	Revised for Version 1.1.2 (Release 14SP3)
March 2006	Second printing	Revised for Version 1.1.3 (Release 2006a)
September 2006	Online only	Revised for Version 1.1.4 (Release 2006b)
March 2007	Online only	Revised for Version 1.2 (Release 2007a)
September 2007	Online only	Revised for Version 1.2.1 (Release 2007b)

Contents

Getting Started

1

Estimating Initial Conditions

2

Preprocessing Data

3

Managing Multiple Projects

4

Adaptive Lookup Tables

5

Estimating from the Command Line

6

Blocks — Alphabetical List

7

Functions — Alphabetical List

8

Index

1

Getting Started

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 to the 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 by using measurements from the physical system.

Simulink Parameter Estimation provides the tools used to

1 Set up the problem.

2 Specify which model parameters to estimate.

3 Import and prepare the experimental data for parameter estimation (or *preprocess*).

4 View the estimation progress.

5 Validate the estimation results based on plots of measured versus. simulated data and residuals.

What You Need to Get Started

In this section...
"Prerequisite Software and Optional Software" on page 1-4
"Required Knowledge" on page 1-4
"Demos" on page 1-4

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 the

- MathWorks Web site at
  http://www.mathworks.com/products/simparameter/related.jsp

- 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. As you gain familiarity with Simulink Parameter Estimation, you might 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 list the available

demos. Alternatively, if you have the Help browser open, you can select the Demos pane in the Help browser and then select **Simulink > Simulink Parameter Estimation**.

How Simulink Parameter Estimation Works

In this section...
"Basic Steps in the Estimation Process" on page 1-6
"Structure of an Estimation Project" on page 1-7
"Adding, Deleting and Renaming an Estimation Project" on page 1-7

Basic Steps in the Estimation Process

Simulink Parameter Estimation compares empirical data with data generated by a Simulink model. Using optimization techniques, Simulink Parameter Estimation estimates the parameter and (optionally) initial conditions of states such that a user-selected cost function is minimized. The cost function typically calculates a least-square error between the empirical and model data signals.

After you build a Simulink model, follow these steps to configure and run a parameter estimation:

1 Select **Tools > Parameter Estimation** in your Simulink model window.

   This opens the Control and Estimation Tools Manager, creates a new project, and adds an **Estimation** node to the workspace directory tree.

2 Import the input and output data set for estimating parameters of your Simulink model.

3 Select the parameters and initial conditions you want to estimate.

4 Configure the estimation itself, including cost functions and data views.

5 Run the estimation.

6 Check the results by examining either the cost-function values, plots, or parameter values.

Structure of an Estimation Project

The Control and Estimation Tools Manager, which is a graphical user interface (GUI) for performing parameter estimation, stores and organizes all data from a given Simulink model inside a *project*. To open the Control and Estimation Tools Manager GUI, select **Tools > Parameter Estimation** in the Simulink model window. See Control and Estimation Tools Manager GUI on page 1-10 for a picture showing this GUI.

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

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

Each estimation task can include

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

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

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

Adding, Deleting and Renaming an Estimation Project

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

1 Right-click the project or task node in the workspace directory tree.

2 Select the appropriate command from the shortcut menu.

Setting Up the Estimation Data

In this section...
"Creating an Estimation Project" on page 1-9
"Importing Transient Data" on page 1-11
"Specifying Initial Conditions" on page 1-14
"Selecting Parameters for Estimation" on page 1-16
"Selecting States for Estimation" on page 1-18
"Specifying Initial Guesses and Upper/Lower Bounds" on page 1-19

Creating an Estimation Project

Before beginning the estimation process, you must create an estimation project and set up the problem by configuring the appropriate parameters, solvers, and cost functions. 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 perform the setup:

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

```
engine_idle_speed
```

at the MATLAB prompt.

The model appears as shown next.
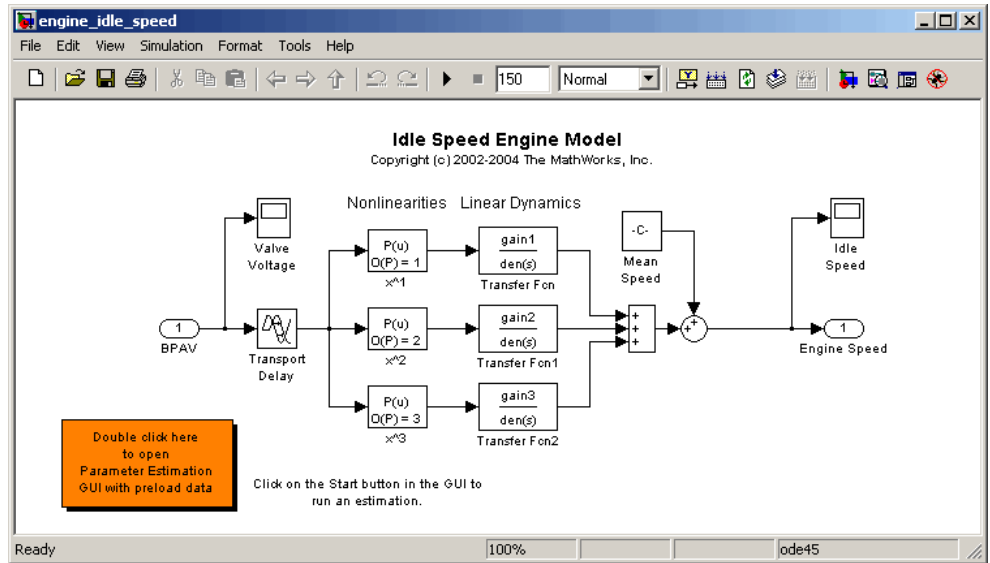
2 Open the Control and Estimation Tools Manager GUI by selecting **Tools > Parameter Estimation** in the Simulink model window.

The workspace directory tree displays the project name. Estimation tasks are organized inside the **Estimation Task** node.
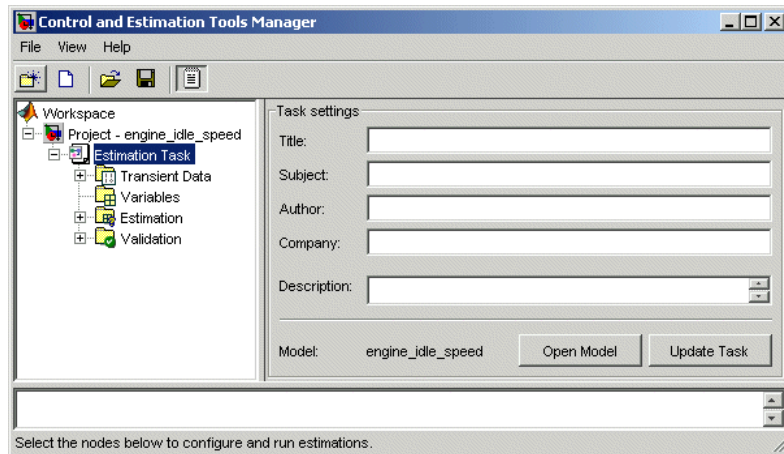
Control and Estimation Tools Manager GUI

Importing Transient Data

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

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

2 Right-click **Transient Data** and select **New** to create a **New Data** node. Alternatively, you can use the **New** button to create this node.
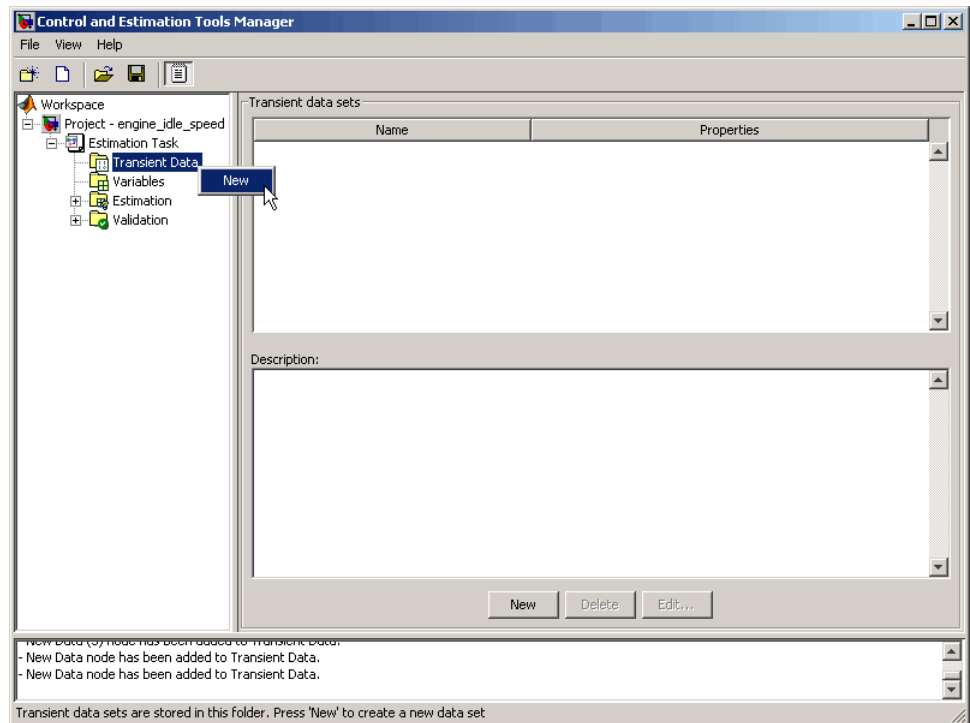
3 Select the **New Data** node in the workspace directory tree.
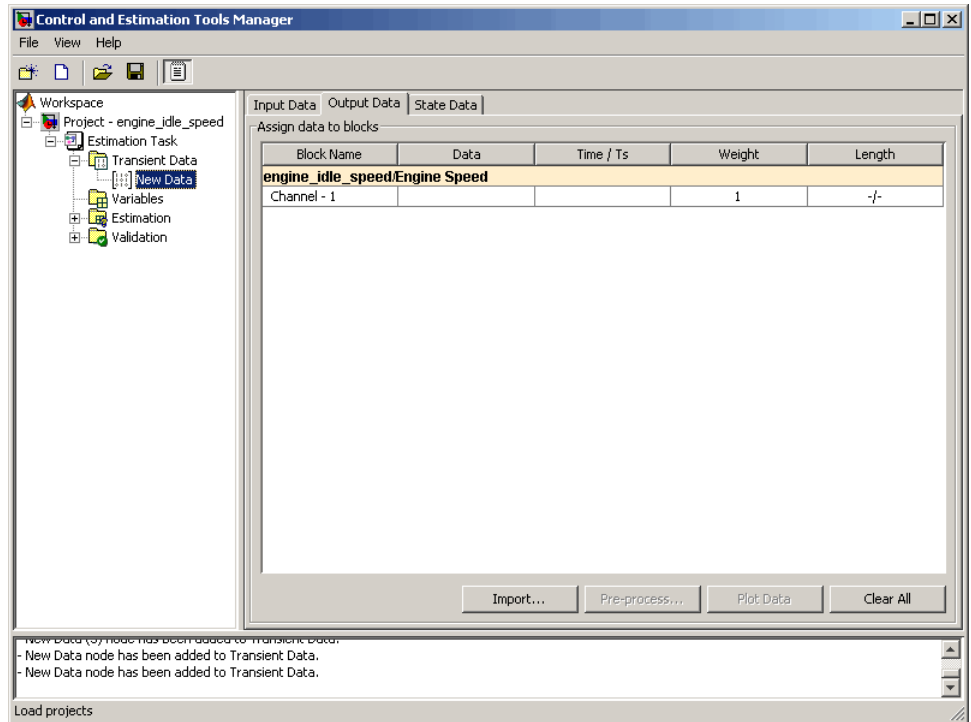
Import Data into the Control and Estimation Tools Manager

Importing Input Data

To import the model input data:

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

2 Right-click the **Data** cell and select **Import** to open the Data Import dialog box. Alternatively, you can use the **Import** button to open this dialog box.
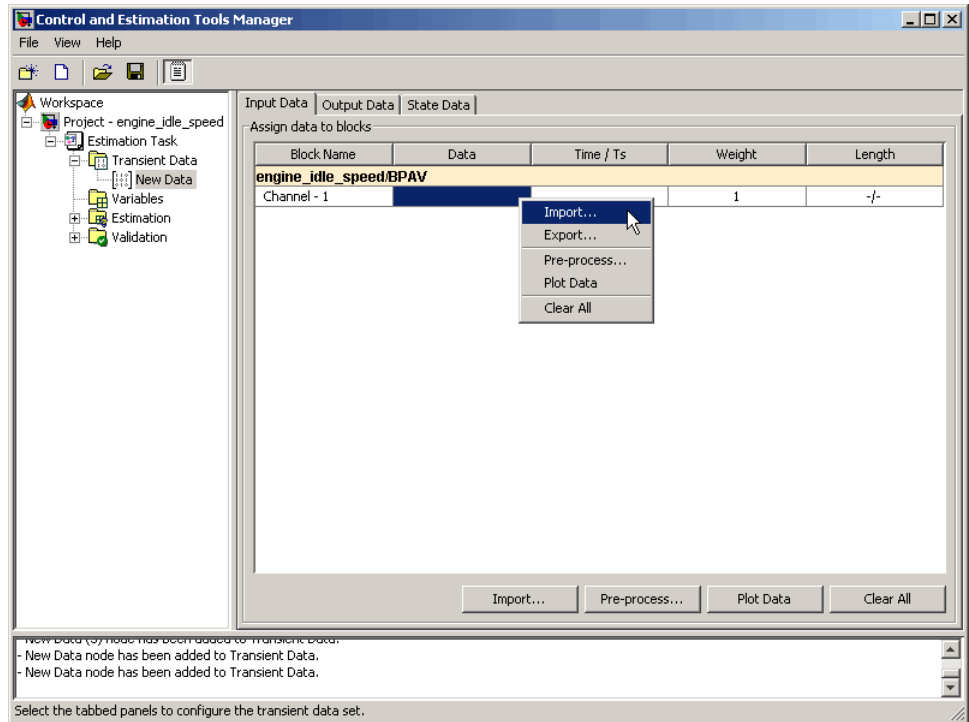
3 In the Data Import dialog box, select iodata from the list of variables.
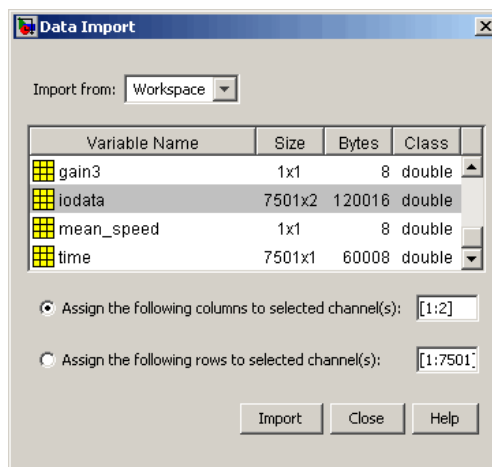
The idle-speed model of an automotive engine contains the measured data stored in the `iodata` array. The array contains two columns: the first for model input data, and the second for model output data.

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

5 Click the **Time/Ts** cell, and select `time` in the Data Import dialog box.

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

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

Importing Output Data

To import the model output data:

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

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

3 Select `iodata` from the list of variables.

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

5 Click the **Time/Ts** cell, and select `time` in the Data Import dialog box.

6 Click **Import** to import the `time` vector for the output data.

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

Specifying Initial Conditions

By default, the estimation uses initial conditions specified in the Simulink model. If you want to specify initial conditions other than the defaults, use the State Data tab. You can open it by selecting **Transient Data > New Data** in the workspace directory tree, and then clicking the **State Data** tab.
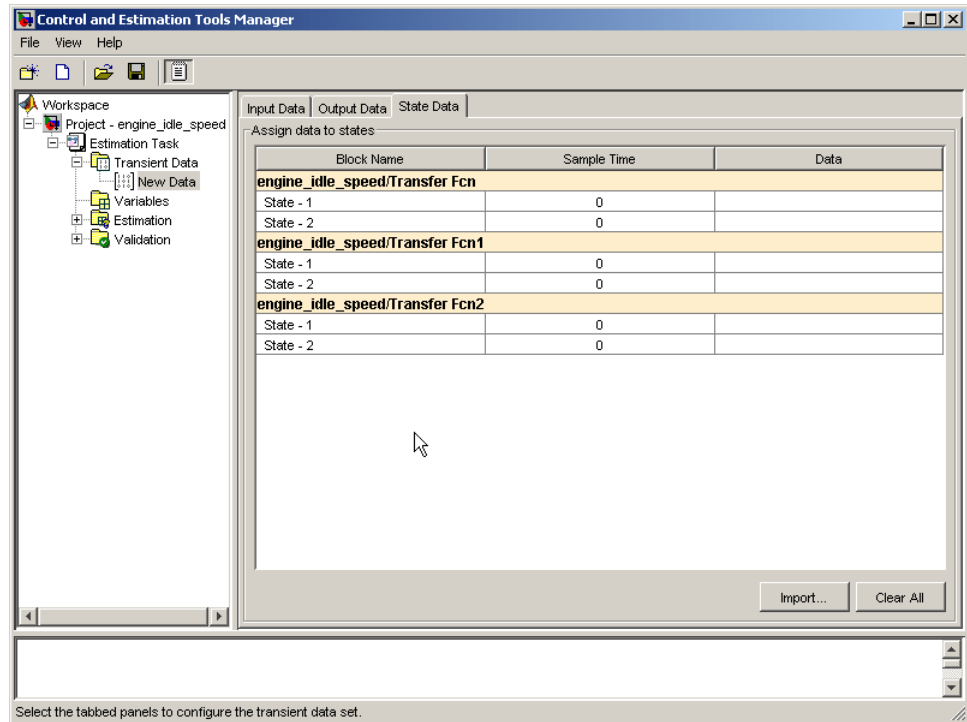
To specify an initial condition of a state:

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

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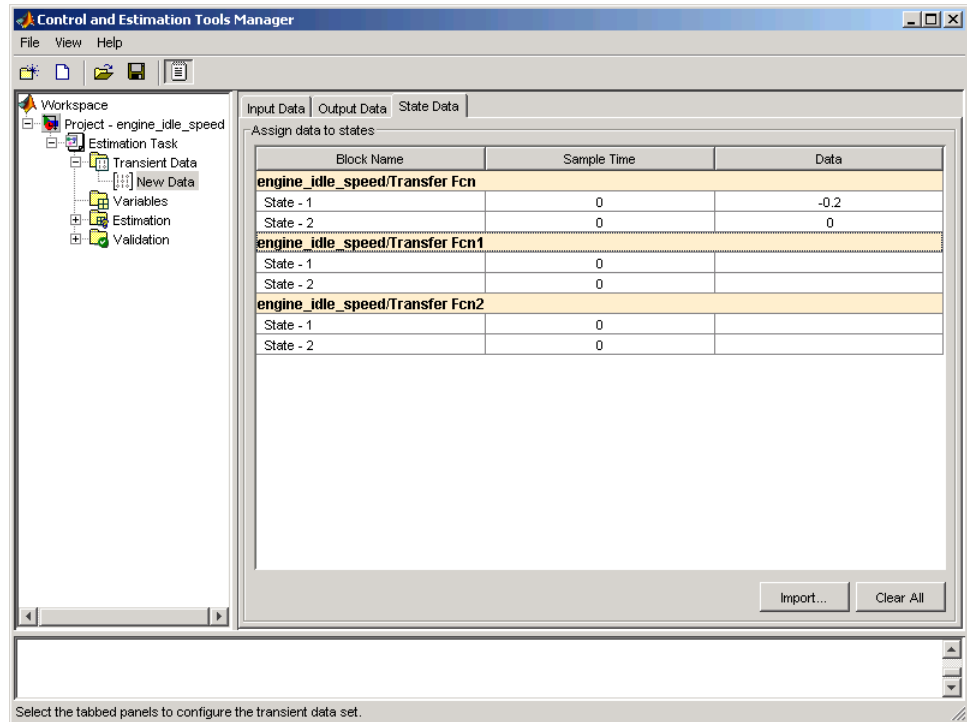
Selecting Parameters for Estimation

To select parameters for estimation:

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

2 In the **Estimated Parameters** pane, click **Add** to open the Select Parameters dialog box.
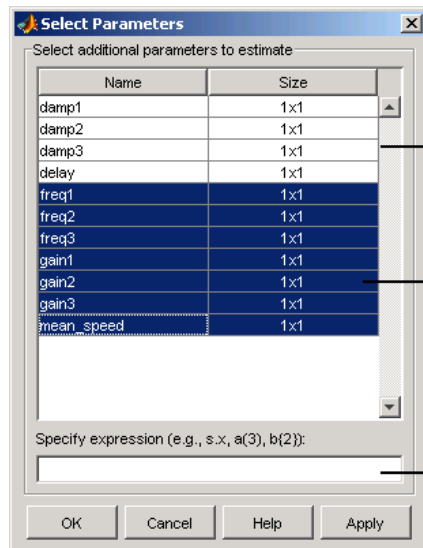
By default, the Select Parameters dialog box looks at all variables in the model workspace and the MATLAB workspace that are used by the model.

List of parameters

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

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

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

In general, you can enter parameters, (separated by commas, in the **Specify expression** field. The parameters can be stored in one of the following:

• Simulink parameter object

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

• Structure

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

• Cell array

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

• MATLAB array

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

1-17

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

Often, it is more practical to estimate a small group of parameters and use the final estimated values as a starting point for further estimation of parameters that are trickier. Making these sorts of choices involves experience, intuition, and a solid understanding of the strengths and limitations of your Simulink model.

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 1-57.

Selecting States for Estimation

In general, you choose to estimate only those states that are not already in the model. To estimate initial conditions (or initial states) if they are not known:

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

2 Click the **Estimated States** tab.

3 Click **Add** to open the Select States dialog box.
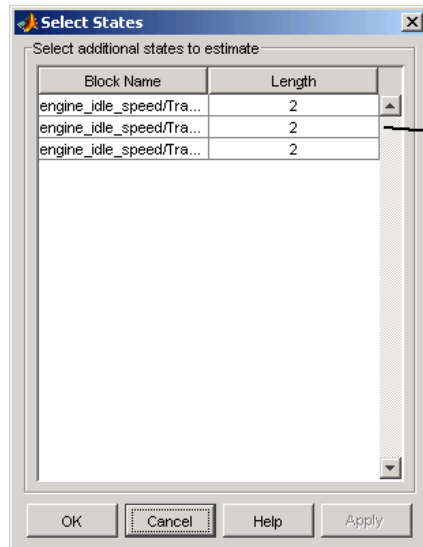
By default, the Select States dialog box looks at all states of all blocks in the model.

List of states

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

4 Examine the available states but do not select any for this example.

Specifying Initial Guesses and Upper/Lower Bounds

After you select parameters for estimation, the Control and Estimation Tools Manager looks like the following figure.
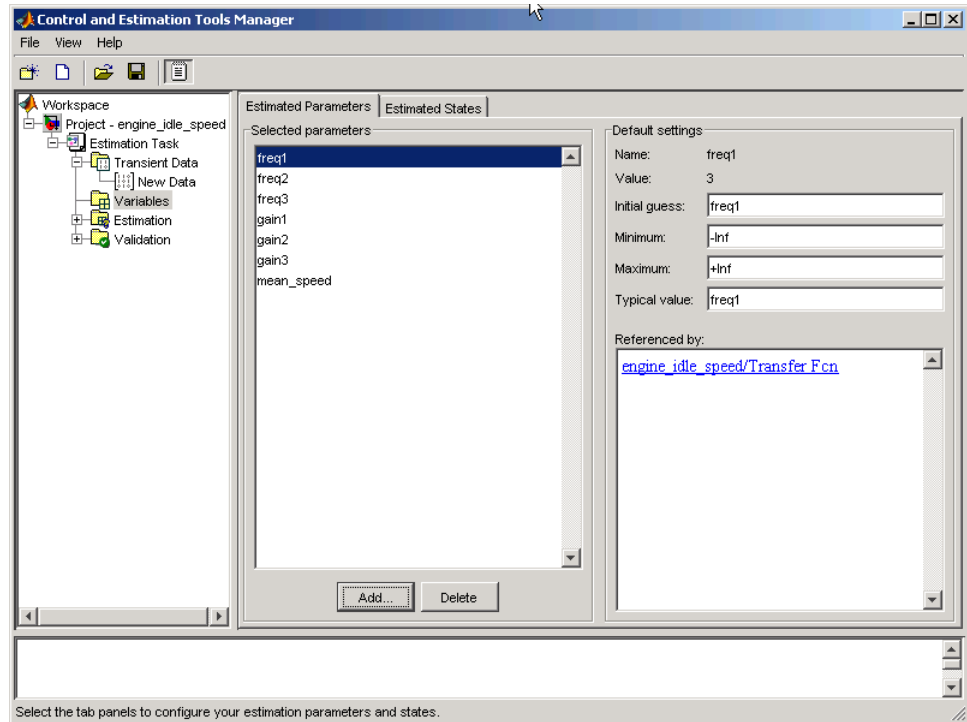
For each parameter, use the **Default settings** pane to specify the following:

- **Initial guess** — The value the estimation uses to start the process.

- **Minimum** — The smallest allowable parameter value. The default is -Inf.

- **Maximum** — The largest allowable parameter value. The default is +Inf.

- **Typical value** — The average order of magnitude. If you expect your parameter to vary over several orders of magnitude, enter the number that specified the average order of magnitude you expect. For example, if your initial guess is 10, but you expect the parameter to vary between 10 and 1000, enter 100 (the average of the order of magnitudes) for the typical value.

You use the typical value in two ways:

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

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

or

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

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

Setting Up an Estimation Project

In this section...

Creating an Estimation Task

After you import the transient data and select the parameters and any initial conditions (or states) to estimate, you must create an estimation task and configure the estimation settings. To create a container that stores the estimation settings:

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

2 Click the **New Estimation** node.

Adding Data Sets

After you select the **New Estimation** node, the **Data Sets** tab appears. Here you choose the output data from the Simulink model that you want to use in the estimation.

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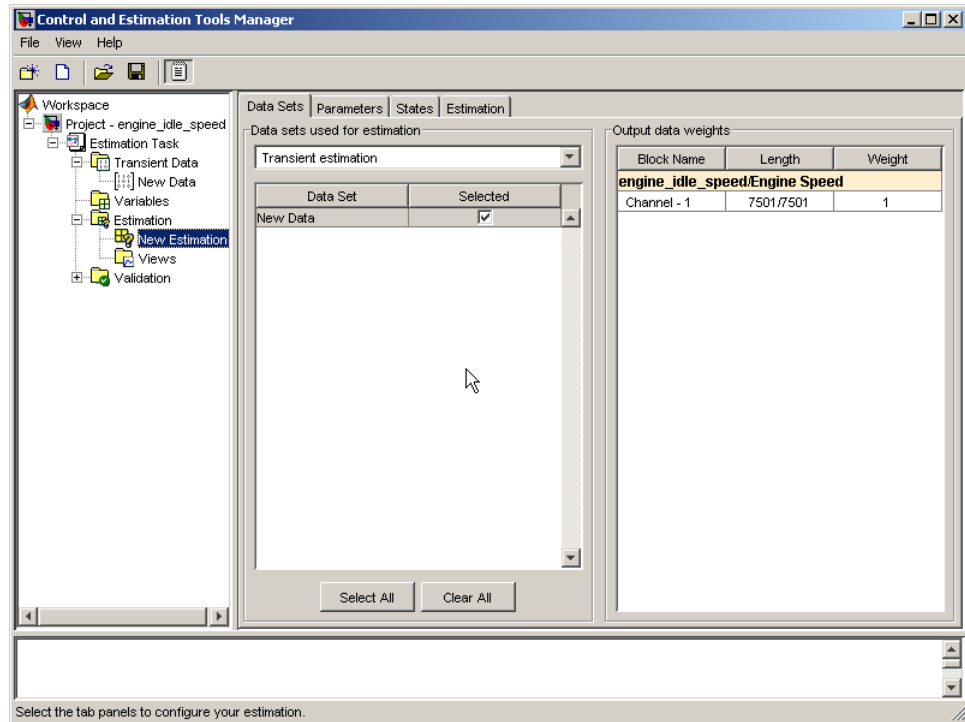
Note If you imported multiple data sets, you can select them for estimation by selecting the check box to the right of each desired data set. When using several data sets, you increase the estimation precision. However, you also increase the number of required simulations: for N parameters and M data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

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

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

- Use less weight when an output is noisy.

- Use more weight when an output strongly affects parameters.

• Use more weight when it is more important to accurately match this model output to the data.

Specifying and Setting Up Parameters

In the **New Estimation** node in the workspace directory tree, click the **Parameters** tab in the Control and Estimation Tools Manager. In this pane, you select which parameters to estimate and the range of values for the estimation.

Note When you specify the **Minimum** and **Maximum** values for the parameters here, it does not affect your settings in the **Variables** node. You make these choices on a per estimation basis. You can move data to and from the **Variables** node into the **Estimation** node.
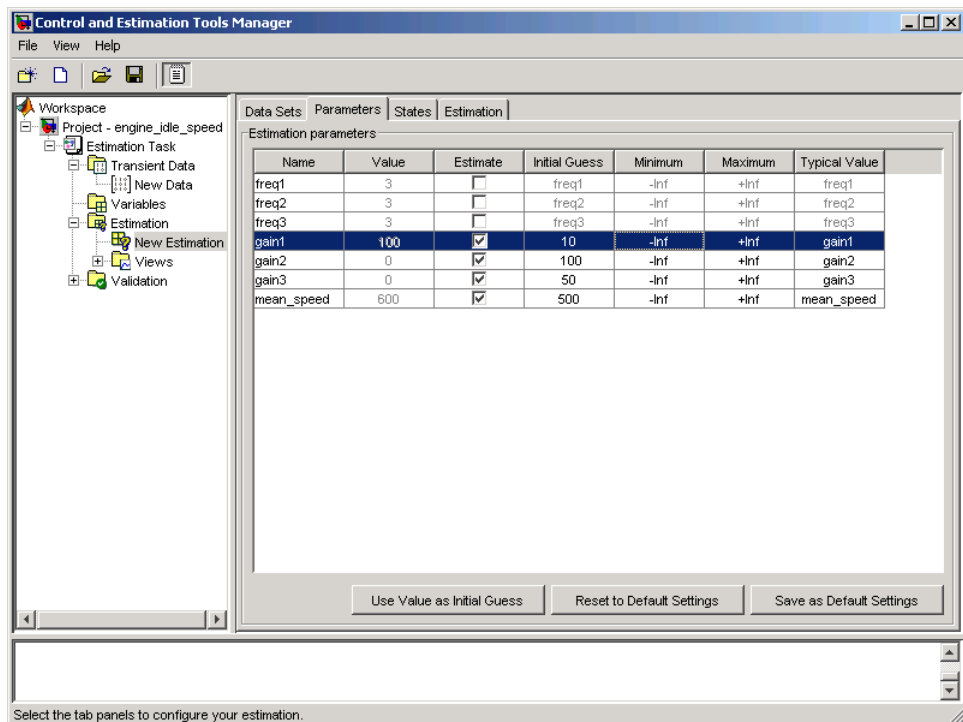
You can select the parameters you want to estimate by selecting the check–box in the **Estimate** column. Enter initial values for your parameters in the **Initial Guess** column. The default values in the **Minimum** and **Maximum** columns are `-Inf` and `+Inf`, respectively, but you can select any range you want.

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

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

Specifying Settings for Estimation

In the **New Estimation** node in the workspace directory tree, click the **Estimation** tab to specify the settings for estimation.
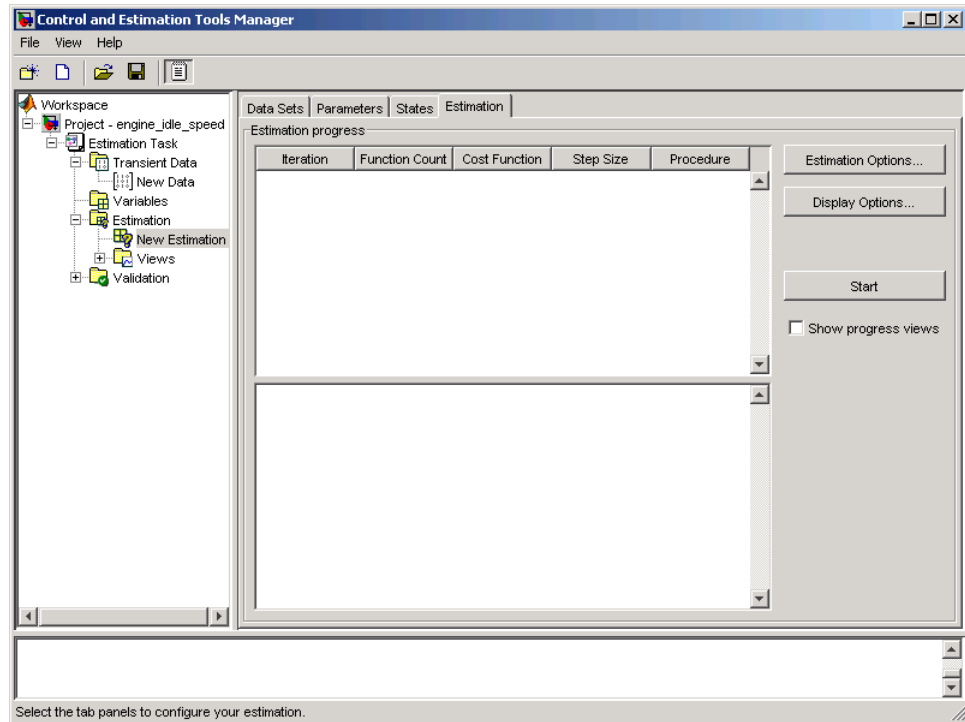
Before you start, you can click **Estimation Options** to specify various algorithms and simulation features. See "Setting Options for Optimization" on page 1-45for more information.

Display Options

You can specify the display options by clicking **Display Options**. This opens the following dialog box.
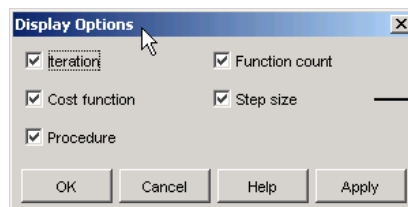
By default, all boxes are checked. Uncheck any feature that you don't want to view during the estimation process.

Clearing a check box implies that feature will not appear in the display table as the estimation progresses.

Selecting Views for Plotting

In this section...
"Basic Steps for Creating Plots" on page 1-28
"Types of Plots" on page 1-31

Basic Steps for Creating Plots

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

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

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

2 In the workspace directory tree, select **New View** to open the **View Setup** pane.
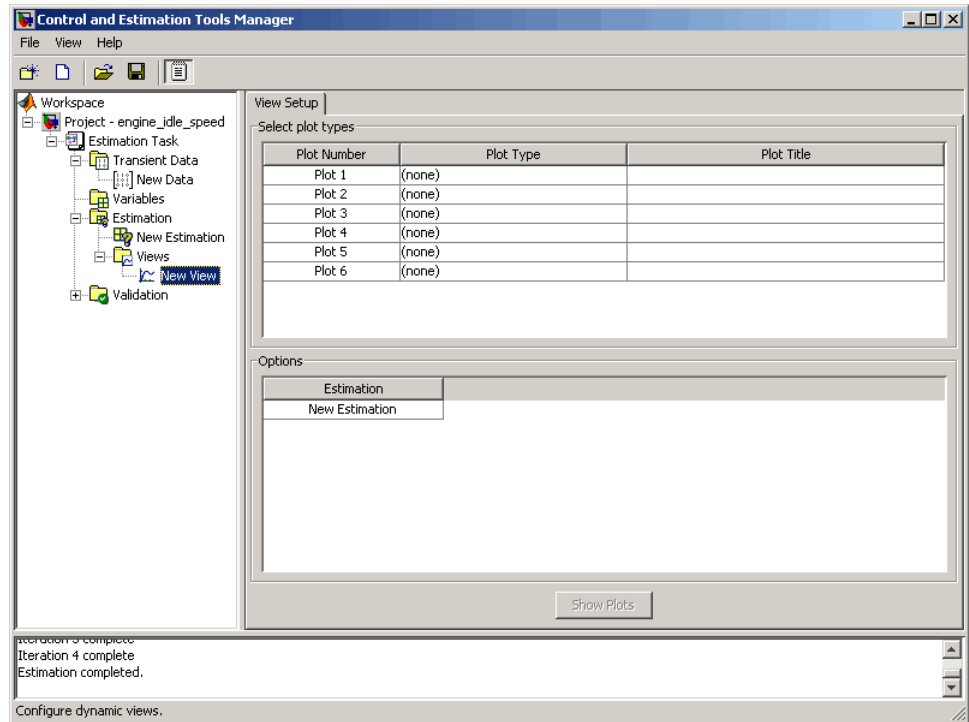
3 In the **Select plot types** table, select the **Plot Type** from the drop-down
list. In this example, select Cost function.
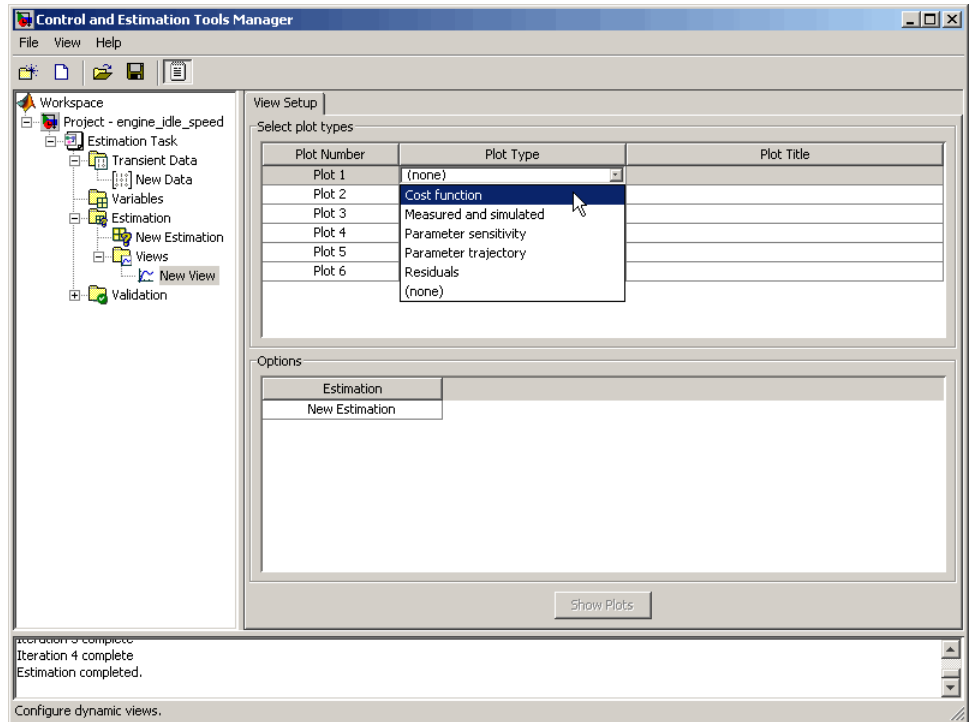
4 Select Measured and simulated as the **Plot Type** for Plot 2. This plot will be used in "Performing Validation" on page 1-38.

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

6 Click **Show Plots**. This displays an empty cost function plot and a plot of the measured data. When you run the estimation, the plot updates automatically.

This figure shows the plot generated by running the estimation, as described in "Running the Estimation" on page 1-32.
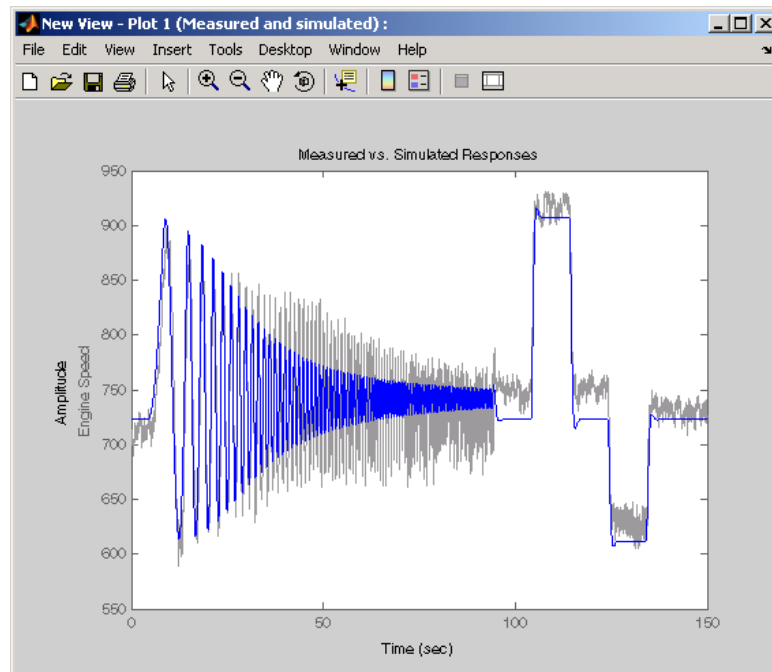
Types of Plots

You can choose the plot type from the drop-down list under **Plot Type**.
Various types of plots are available, including

- `Cost function` — Plot the cost function values.

- `Measured and simulated` — Plot empirical data against simulated data.

- `Parameter sensitivity` — Plot the rate of change of the cost function as a
  function of the change in the parameter. That is, plot the derivative of the
  cost function with respect to the parameter being varied.

- `Parameter trajectory` — Plot the parameter values as they change.

- `Residuals` — Plot the error between the experimental data and the
  simulated output.

Running the Estimation

In the Control and Estimation Tools Manager, select the **New Estimation** node and click the **Estimation** tab.

Click **Start** to begin the estimation process. At the end of the iterations, the window should resemble the following:
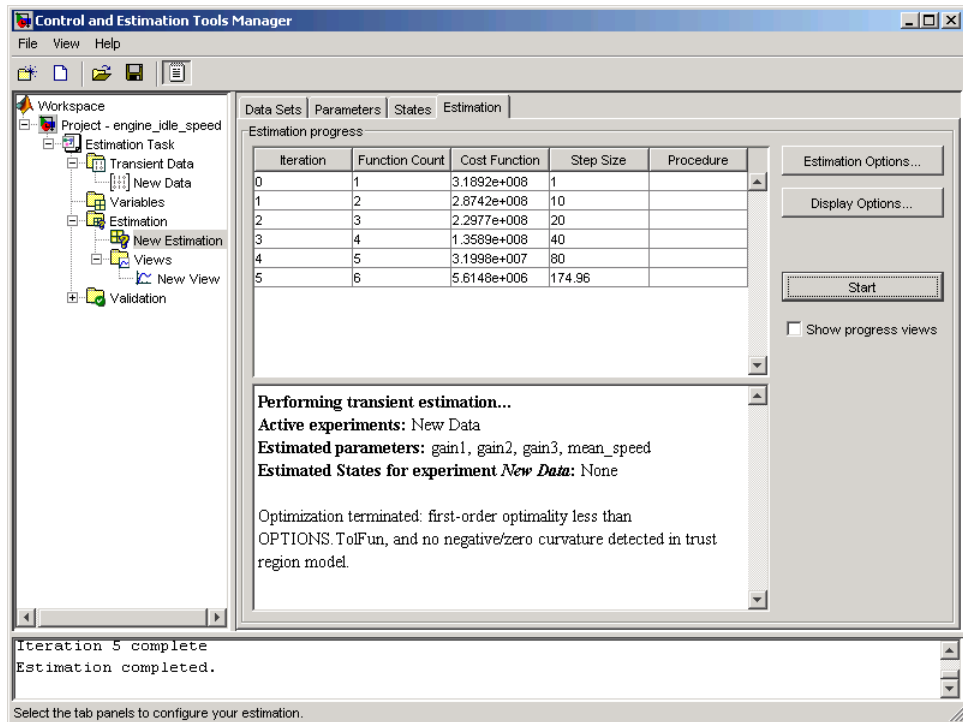
Usually, a lower cost function value indicates a successful estimation, meaning that the experimental data matches the model simulation with the estimated parameters.

The **Estimation** pane displays each iteration of the optimization algorithm. To see the final values for the parameters, click the **Parameters** tab.
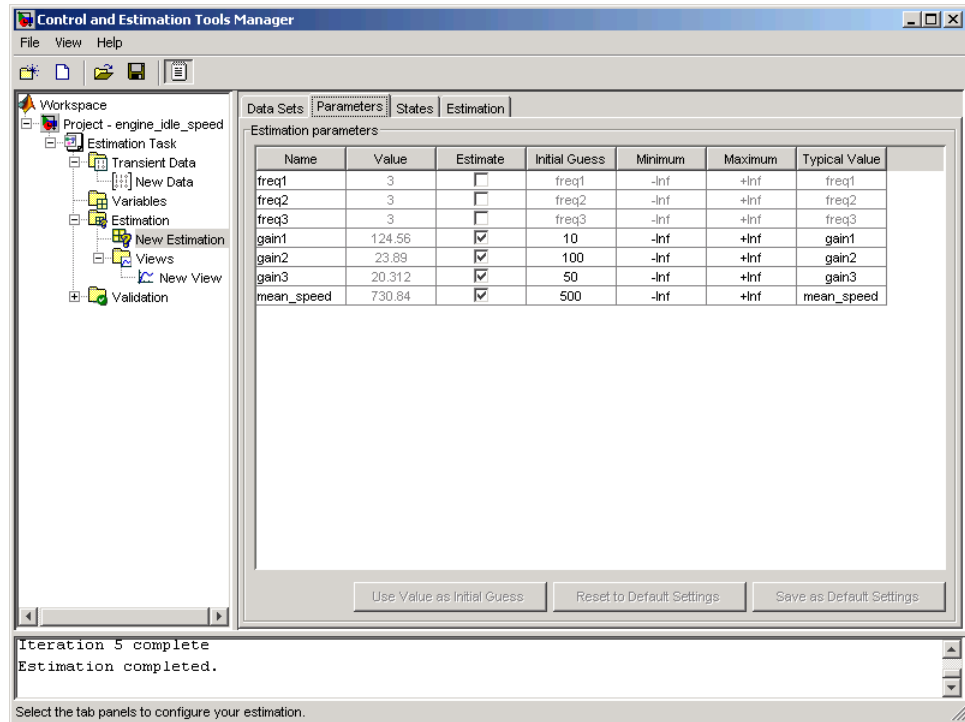
The values of these parameters are also updated in the MATLAB workspace. If you specify the variable name in the **Initial Guess** column, you can restart the estimation from where you left off at the end of a previous estimation.

After the estimation process completes, the cost function minimization plot appears as shown in the following figure.
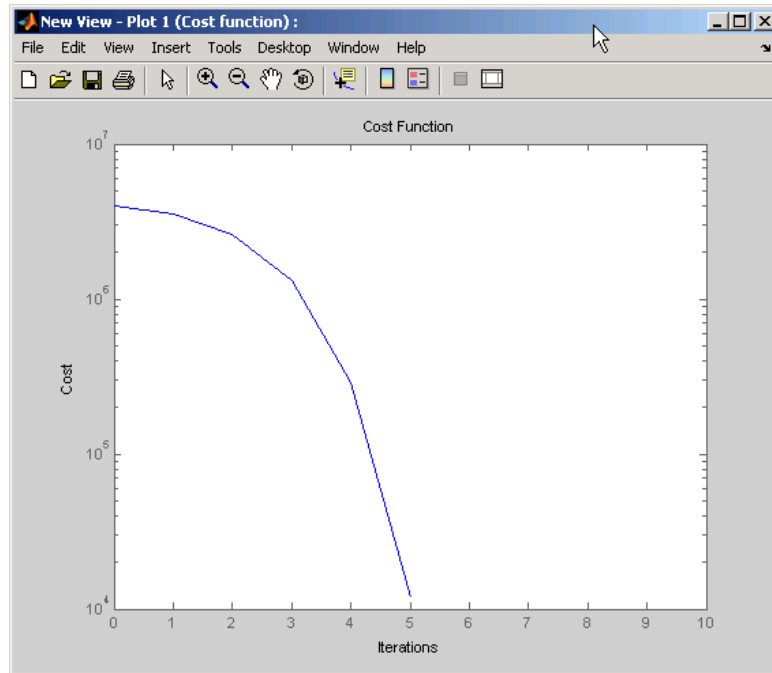
If the optimization went well, you should see your cost function converge on a minimum value. The lower the cost, the more successful is the estimation.

Model Validation

In this section...
"Basic Steps for Model Validation" on page 1-35
"Example: Validating the Engine Idle Speed Model" on page 1-36
"Loading and Importing the Validation Data" on page 1-37
"Performing Validation" on page 1-38
"Comparing Residuals" on page 1-42

Basic Steps for Model Validation

After you complete estimating the parameters, 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 set to the **Transient Data** node.

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

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

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

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

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

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

Example: Validating the Engine Idle Speed Model

If you have not run the engine idle speed demo, type:

```
engine_idle_speed
```

at the MATLAB prompt and run the estimation. For more information on how to run the estimation, see "How Simulink Parameter Estimation Works" on page 1-6. You can double-click 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.
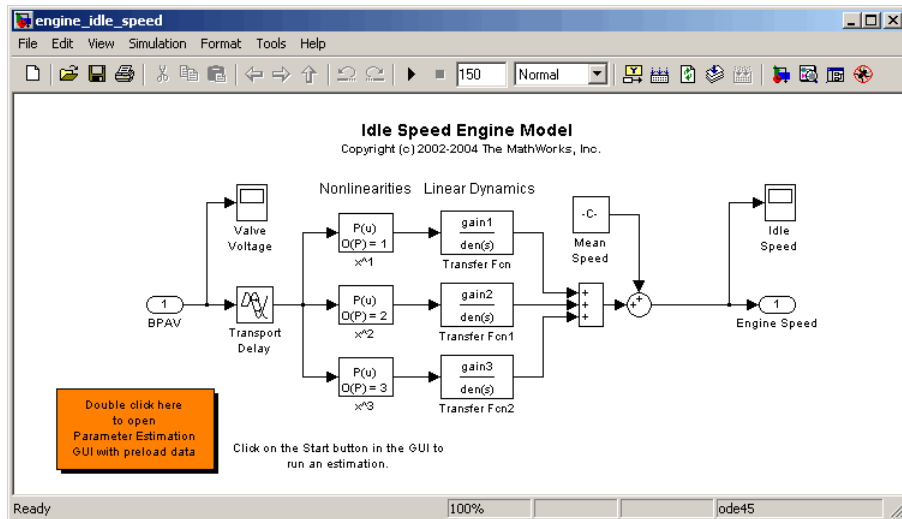
engine_idle_speed Simulink Model

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, type

```
load iodataval
```

at the MATLAB prompt. This loads the data into the MATLAB workspace. The next step is to import this data into the Control and Estimation Tools Manager. See "Importing Transient Data" on page 1-11 for information on importing data, but the quickest way is to follow these steps:

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

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

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

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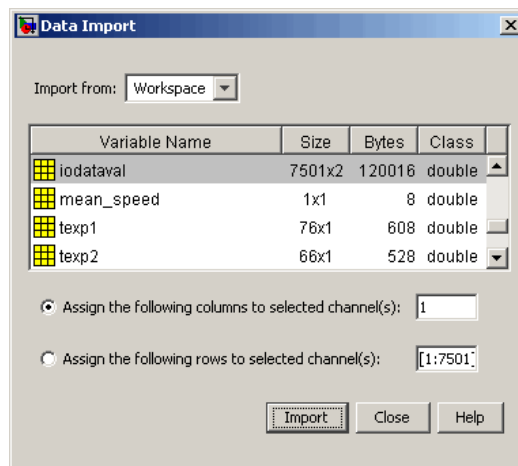
5 Select the **Time/Ts** cell and import time using the Data Import dialog box.

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

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

Your Control and Estimation Tools Manager should resemble this figure.
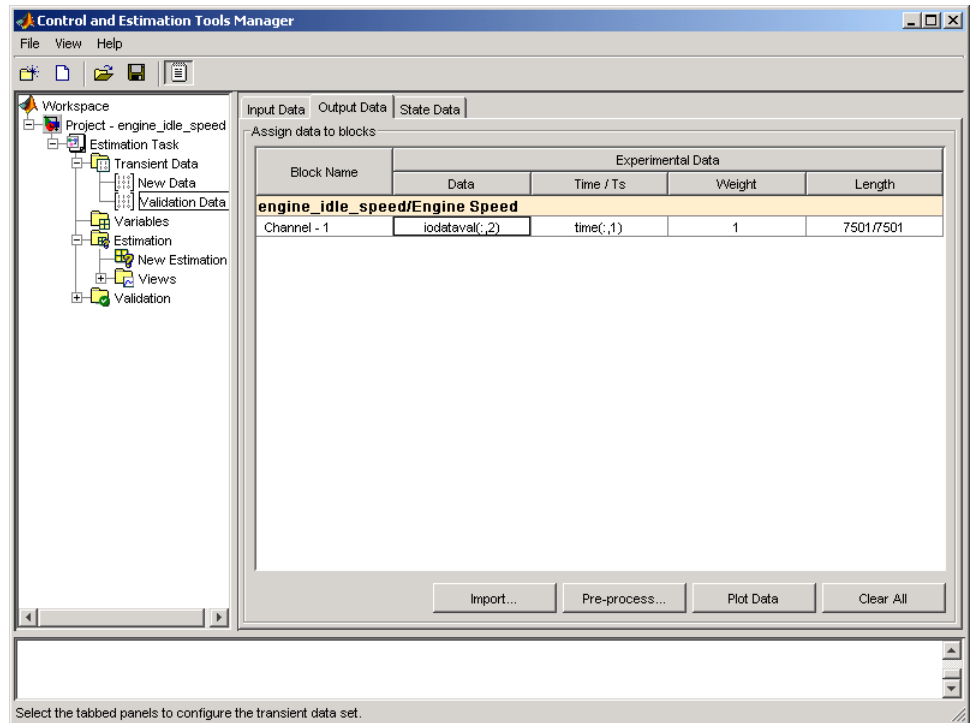
Performing Validation

After you import the data, right-click the **Validation** node and select **New**. This creates a **New Validation** node in the Control and Estimation Tools Manager.
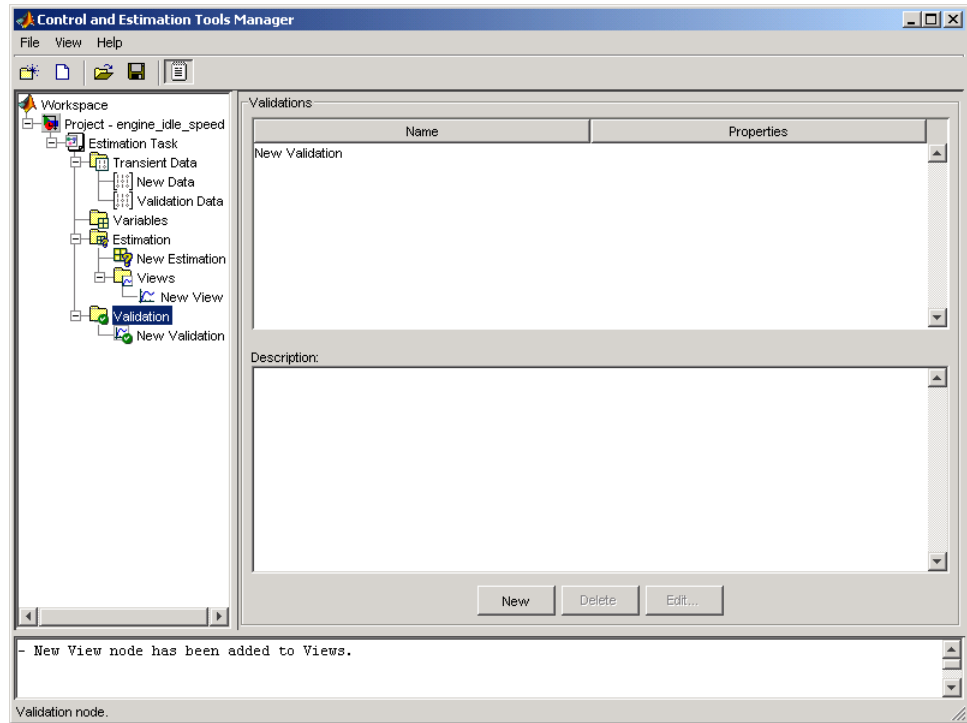
To perform the validation:

1 Select **New Validation** in the workspace directory tree to open the **Validation Setup** pane.
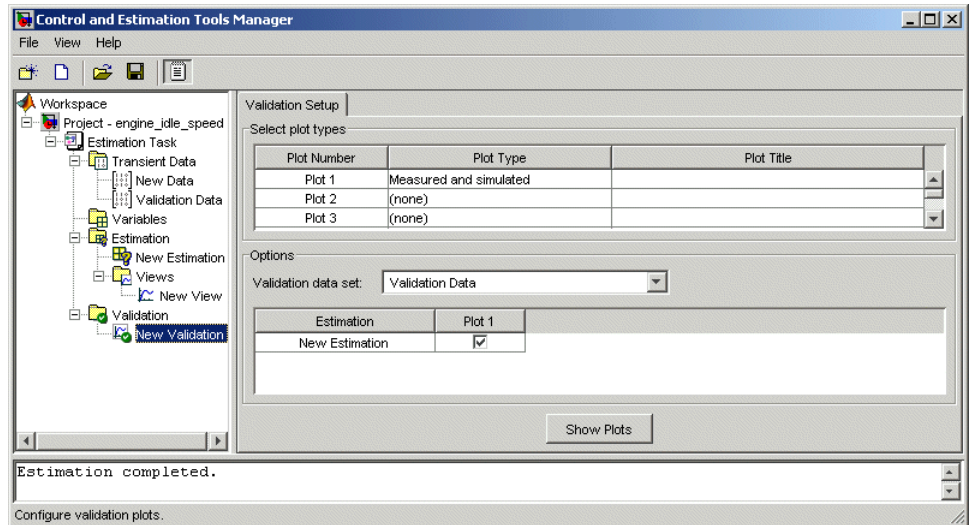
2 Click the **Plot Type** cell for Plot 1 and select Measured and simulated from the drop-down menu.

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

4 Click **Show Plots** to open a plot figure window as shown next.
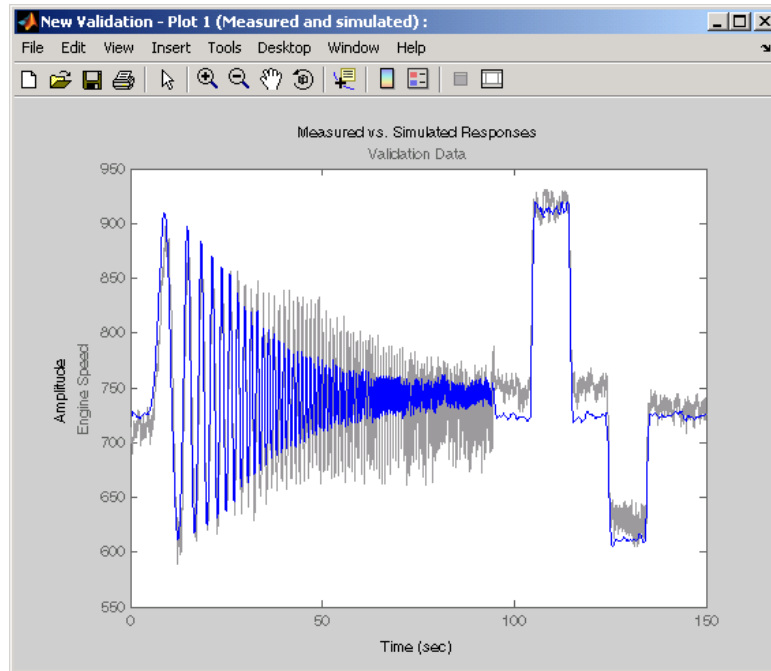
Measured Versus Simulated Data Plot for Validation Data

5 Compare this plot with the plot of `Measured and simulated` data for
the validation data. For more information on how to create this plot, see
"Selecting Views for Plotting" on page 1-28.
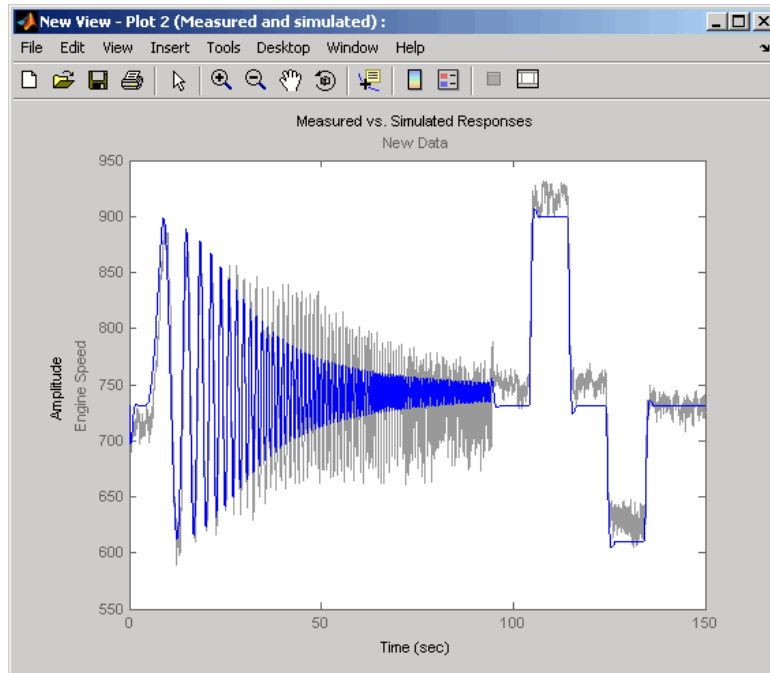
Measured and Simulated Data Views Plot

Transient Data node as a

Comparing Residuals

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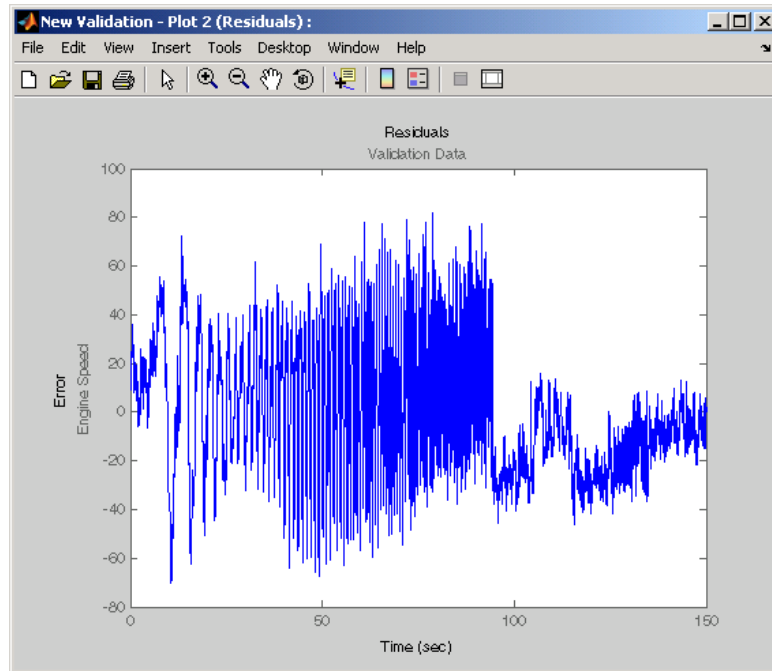
Plot of Residuals Using the Validation Data

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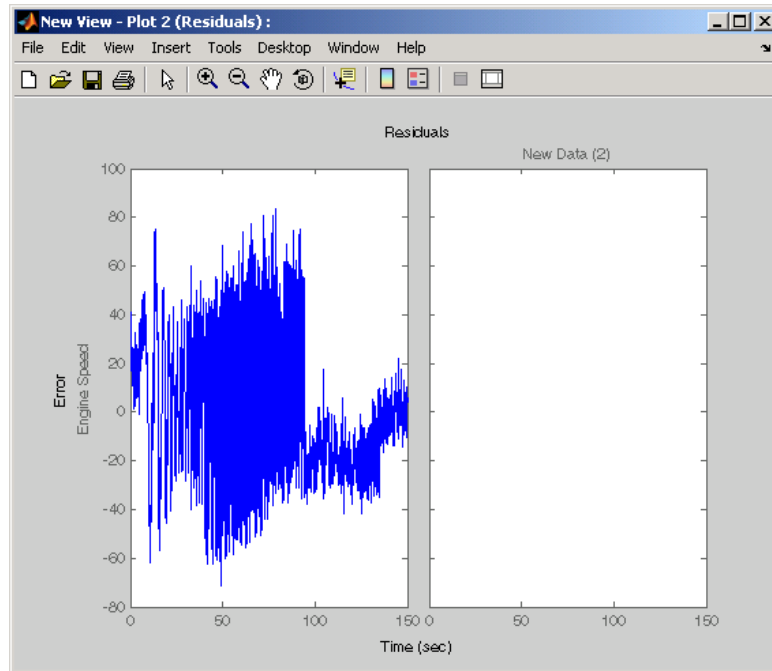
Plot of Residuals Using the Test Data

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

Setting Options for Optimization

In this section...
"Tuning the Results of Optimization" on page 1-45
"Selecting Optimization Methods" on page 1-46
"Selecting Optimization Termination Options" on page 1-47
"Selecting Additional Optimization Options" on page 1-48
"Specifying the Cost Function" on page 1-48

Tuning the Results of Optimization

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

To set options for optimization:

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

2 Click the **Estimation** tab.

3 Click **Estimation Options** to open the Options dialog box.
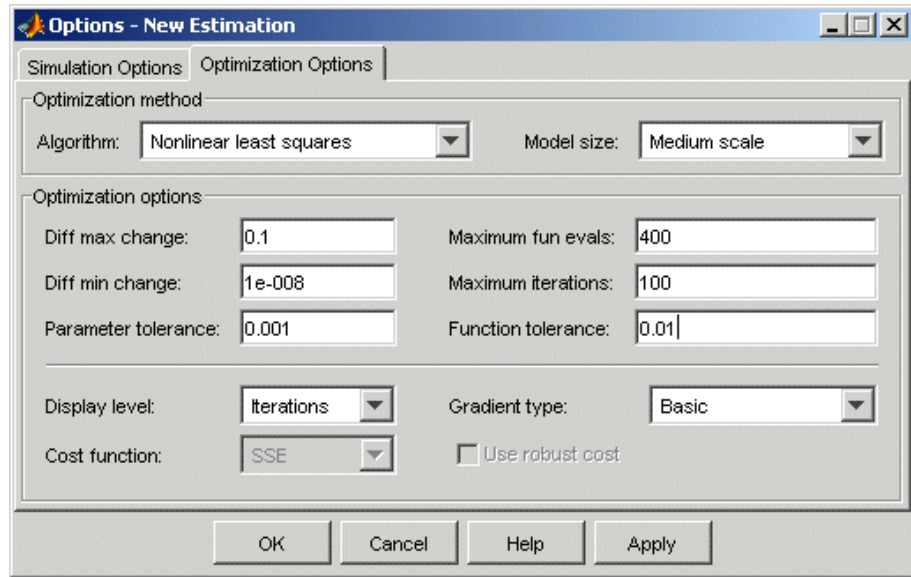
4 Click the **Optimization Options** tab and specify the options, as described in the following sections:

Selecting Optimization Methods

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

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

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

- Nonlinear least squares — Uses a nonlinear least squares 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`, which is a direct search method to optimize the response. `Simplex search` is most useful for simple problems and is sometimes faster than `Function minimization` for models that contain discontinuities.

For **Model size**, the two options are `Large scale` and `Medium scale`. By default, **Model size** is set to `Large scale`. The `Large scale` methods will always run faster, if there is some sparsity structure in the Jacobian (or Hessian) that can be taken advantage of.

For more information on these algorithms, refer to "Medium- and Large-Scale Algorithms", "Large-Scale Algorithms", and "Standard Algorithms" in Optimization Toolbox documentation.

Selecting Optimization Termination Options

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

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

Several options define when the optimization terminates:
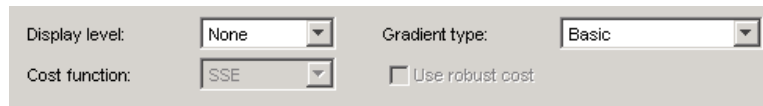
- **Diff max change** — The maximum allowable change in variables for finite-difference derivatives. See `fmincon` in the Optimization Toolbox documentation for details.

- **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** pane is a group of additional optimization options.

Additional options for optimization include

- **Display level** — Specifies the form of the output that appears in the MATLAB command window. The options are Iteration, which displays information after each iteration, None, which turns off all output, Notify, which displays output only if the function does not converge, and Final, which only displays the final output. Refer to the Optimization Toolbox documentation for more information on what type of iterative output each algorithm displays.

- **Gradient type** — When using Gradient Descent or Nonlinear least squares as the **Algorithm**, 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

In this section...

Specifying Simulation Options

To optimize the response signals of a model, Simulink Parameter Estimation runs simulations of the model.

To set options for simulation:

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

2 Click the **Estimation** tab.

3 Click **Estimation Options** to open the Options dialog box.
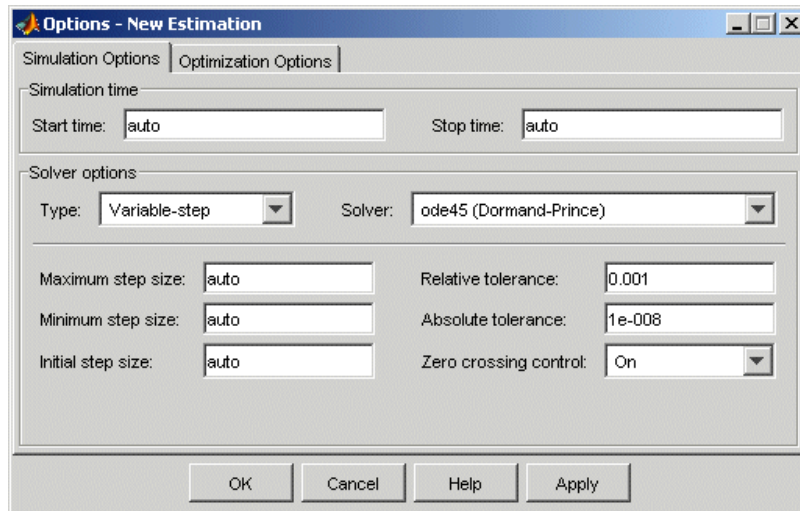
4 Click the **Simulation Options** tab and specify the options, as described in the following sections.

Selecting Simulation Time

Simulation time

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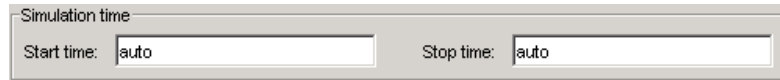
By default, **Start time** and **Stop time** are automatically computed based on the start and stop times specified in the Simulink model.

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

Simulation Time for Data Sets with Different Time Lengths

Simulink Parameter Estimation can handle simulation of models containing empirical data sets of different time lengths. You can use experimental data sets for estimation that contain I/O samples collected at different time points.

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

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

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

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

Simulink Parameter Estimation computes the simulation time *t* as:

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

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

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

- **Start time** — Typically, the start time in the Simulink model is set to 0. For a nonzero start time, the simulated data corresponding to time points

  before t_1^1 for *y1(t)* and t_1^2 for *y2(t)* are discarded.

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

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

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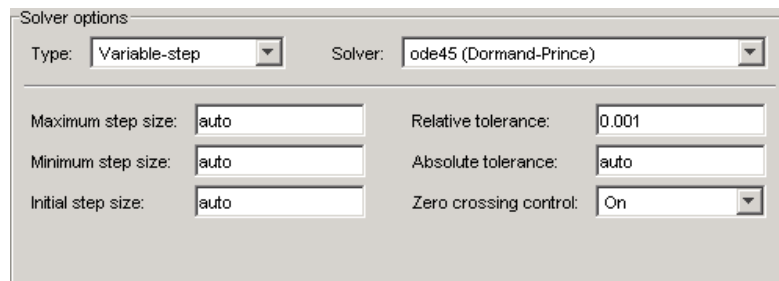
When running the simulation, Simulink solves the dynamic system using one of several solvers. You can specify several solver options using the **Solver options** area in the Options dialog box. The **Type** of solver can be variable-step or fixed-step. Variable-step solvers keep the error within specified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

rapidly, a variable-step solver is often faster. See the Simulink documentation for information about solvers.

Variable-Step Solvers

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

- `discrete (no continuous states)`
- `ode45 (Dormand-Prince)`
- `ode23 (Bogacki-Shampine)`
- `ode113 (Adams)`
- `ode15s (stiff/NDF)`
- `ode23s (stiff/Mod. Rosenbrock)`
- `ode23t (Mod. stiff/Trapezoidal)`
- `ode23tb (stiff/TR-BDF2)`

Variable-Step Solver Options

When you select `Variable-step` as the 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.

Fixed-Step Solvers

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

- `discrete (no continuous states)`
- `ode5 (Dormand-Prince)`
- `ode4 (Runge-Kutta)`
- `ode3 (Bogacki-Shanpine)`
- `ode2 (Heun)`
- `ode1 (Euler)`

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

Accelerating the Estimation

About Accelerating Estimation

You can accelerate the parameter estimation computations by changing the simulation mode of your Simulink model. Simulink Parameter Estimation supports `Normal` and `Accelerator` simulation modes. For more information about these modes, see in the Simulink documentation.

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

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

Limitations

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

Setting the Accelerator Mode for Parameter Estimation

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

- Select **Simulation > Accelerator**.

- Choose `Accelerator` from the drop-down list as shown in the next figure.
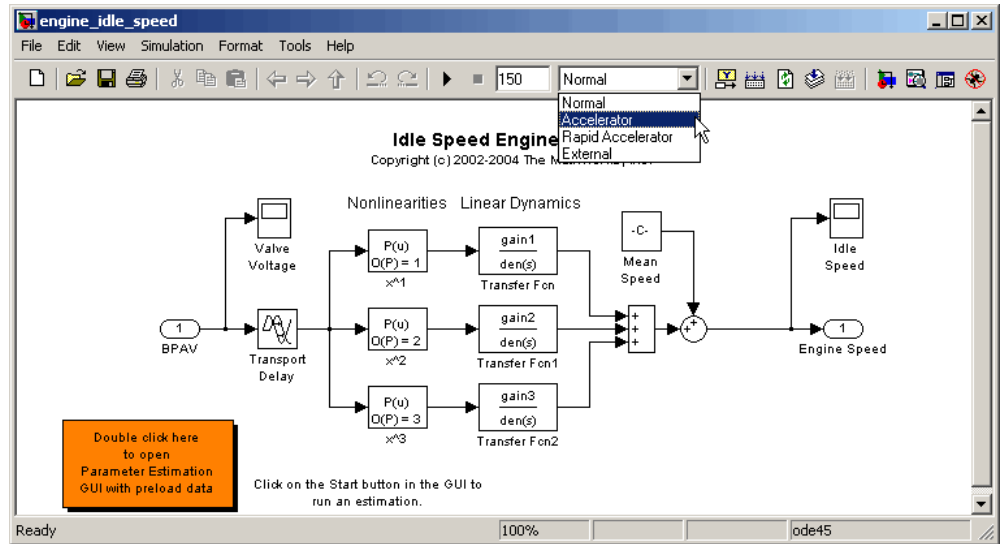
Tip To obtain the maximum performance from the Accelerator mode, close all Scope blocks in your model.

Estimating Independent Parameters

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

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

Example: Estimating Independent Parameters

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

1 At the MATLAB prompt, type

   srotut1

This opens the srotut1 model window.

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

3 In the Model Hierarchy tree, select the **srotut1 > Model Workspace** node.
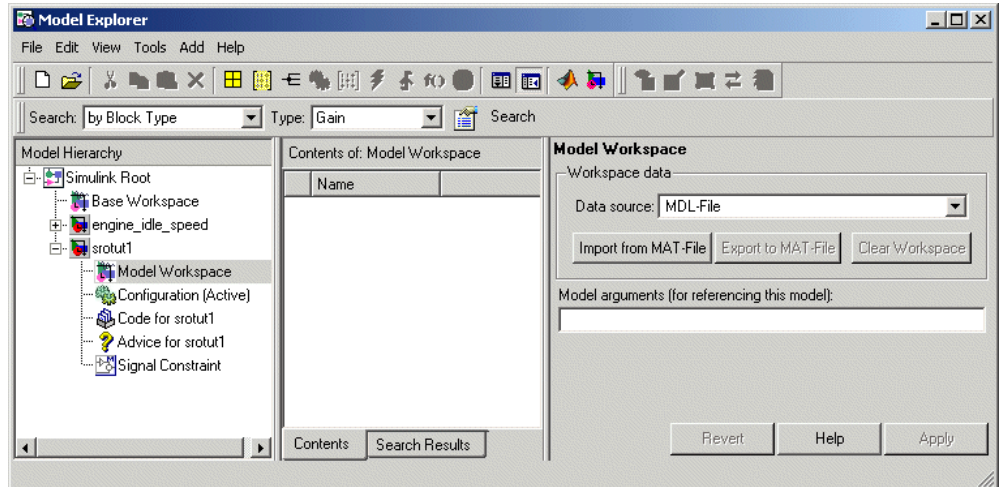
4 Select **Add > MATLAB Variable** to add a new variable to the model workspace. A new variable with a default name Var appears in the **Contents of: Model Workspace** pane.

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

6 Repeat steps 4 and 5 to add a variable y with an initial value of -0.7. The Model Explorer window should resemble the following figure.
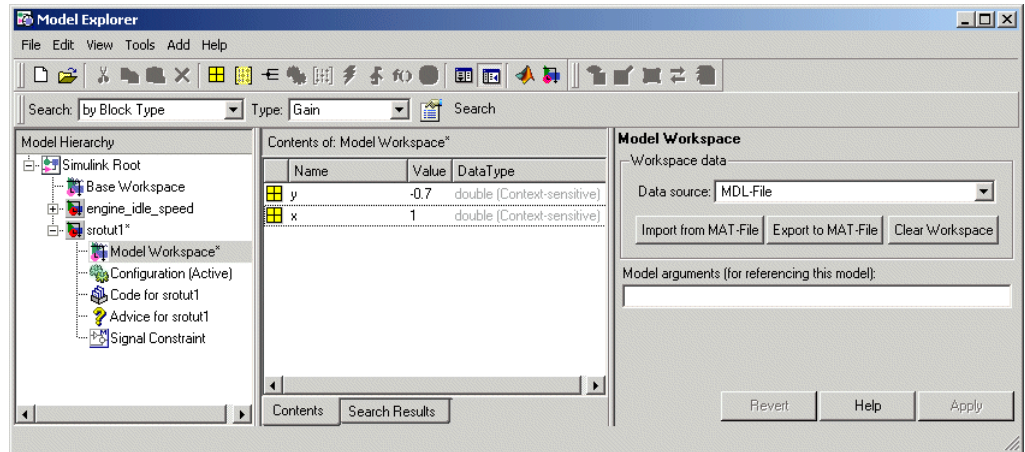
7 To add the Simulation Start function defining the relationship between Kint and the independent parameters x and y, select **File > Model Properties** in the Simulink model for srotut1.

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

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

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

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

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

11 When you select parameters for estimation in the **Variables** node, x and y appear in the Select Parameters dialog box.
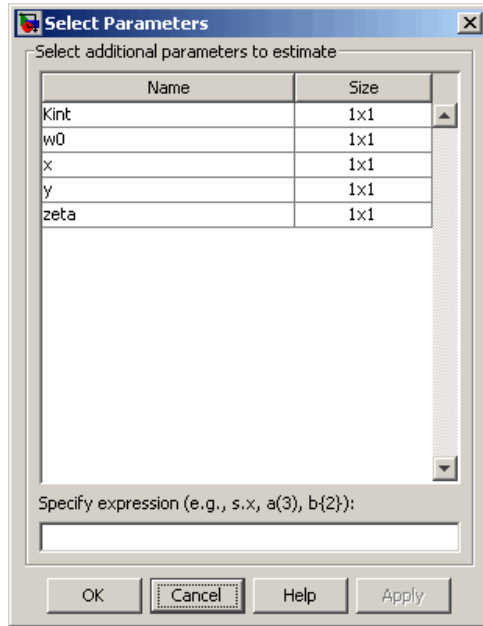
12 Follow the workflow described in "Setting Up an Estimation Project" on page 1-22 to estimate the parameters.

2

Estimating Initial Conditions

Why Estimate Initial Conditions?

Often, sets of measured data are collected at various times and under different initial conditions. When you estimate model parameters using one data set and subsequently run another estimation with a second data set, your parameter values may not match. Given that 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** pane 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.

This chapter focuses on the steps required to estimate initial conditions, and then estimate the parameters from these initial conditions.

Estimating Initial Conditions for Blocks with External Initial Conditions

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

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

Example: Mass-Spring-Damper System

Loading the Example

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

```
msd_system
```

at the MATLAB prompt.

The figure shown next is a model of a mass-spring-damper system.
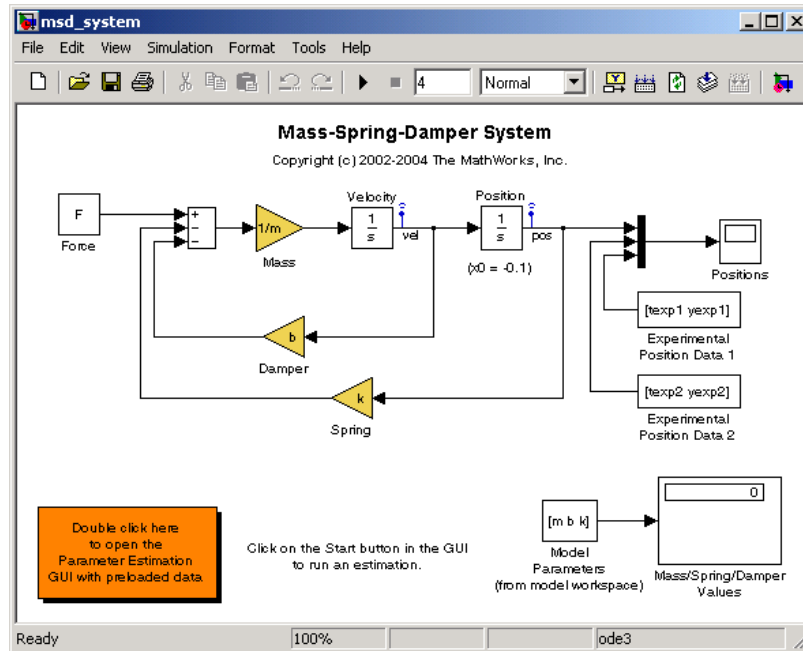
You can run the demo from **Simulink > Simulink Parameter Estimation** on the **Demo** pane of the Help browser.

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

Model Parameters

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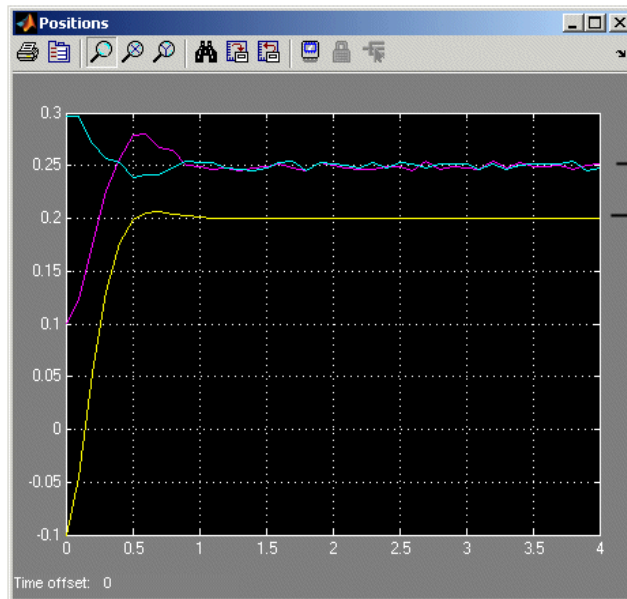
Magenta and cyan lines are emperical responses for data with different initial conditions.

Yellow line 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, x0=0.1 and x0=0.3, and also contain additive noise. A plot of these data sets is shown in the figure above (top curves), along with the simulated response (bottom curve) of the Simulink model msd_system for x0=-0.1 and a nominal set of parameter values, m=8, k=500, and b=100.

Setting Up the Estimation Project

To set up the estimation of initial conditions and then transient state space data, select **Tools > Parameter Estimation** in the msd_system model window.
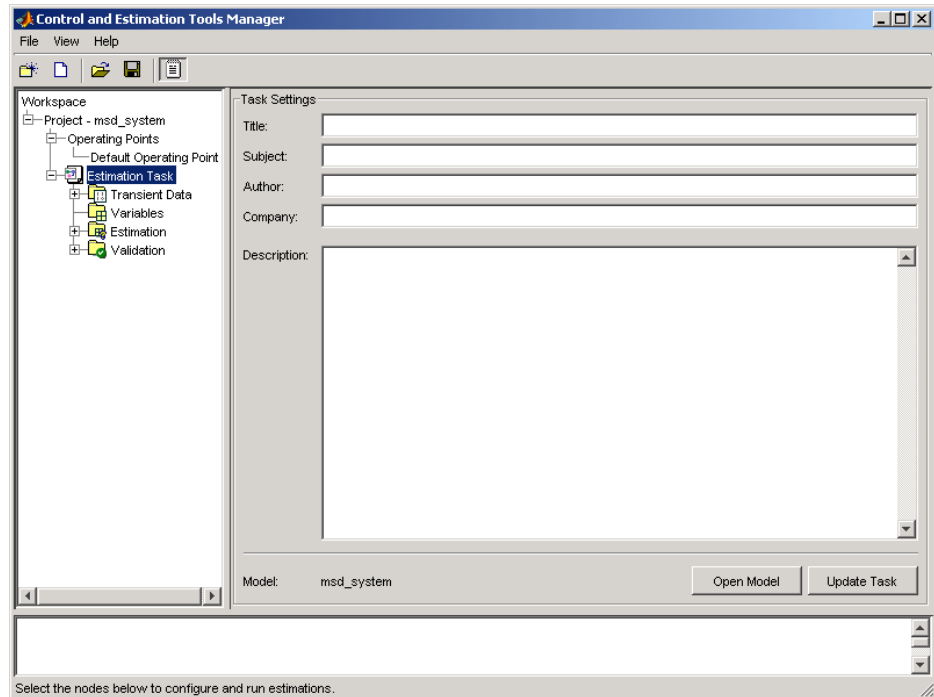
Importing Transient Data and Selecting Parameters for Estimation

The process for importing transient data and selecting parameters for estimation is discussed in "Importing Transient Data" on page 1-11 and "Selecting Parameters for Estimation" on page 1-16.

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

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

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

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

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

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

Your Control and Estimation Tools Manager should resemble this figure:
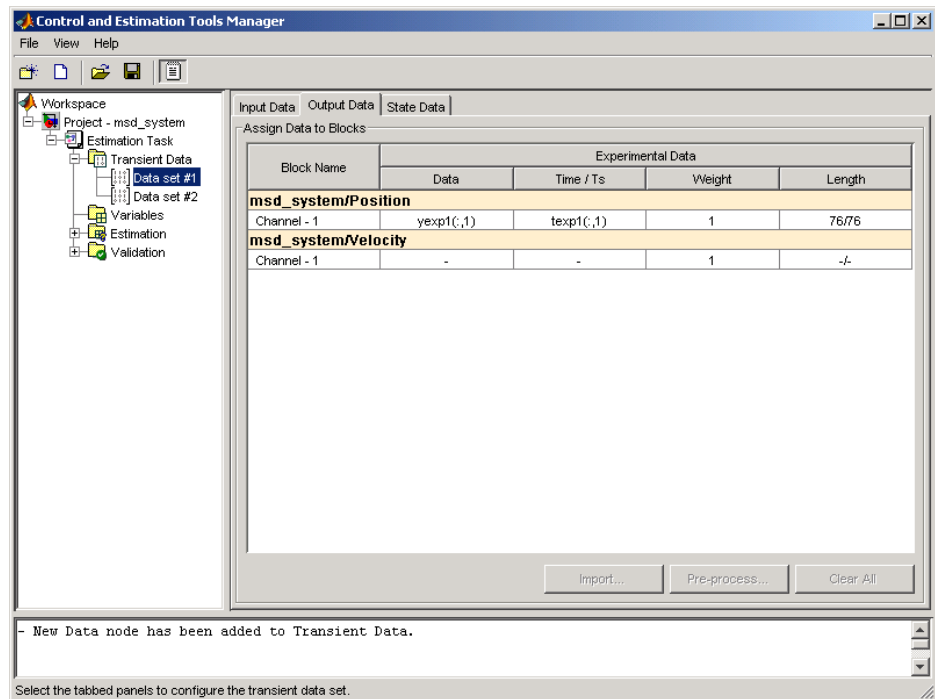
Selecting Parameters and Initial Conditions for Estimation

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

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

2 Click the **Estimation Parameters** tab.

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

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

5 Do the same with the **Estimation States** pane, and select msd_system/Position from the Select States dialog box.
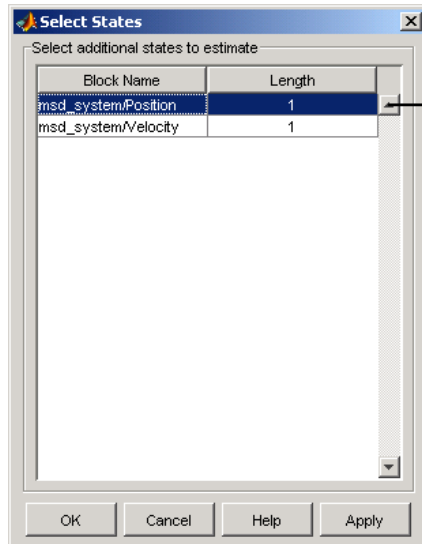
Select states with initial conditions that you want to estimate.

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.
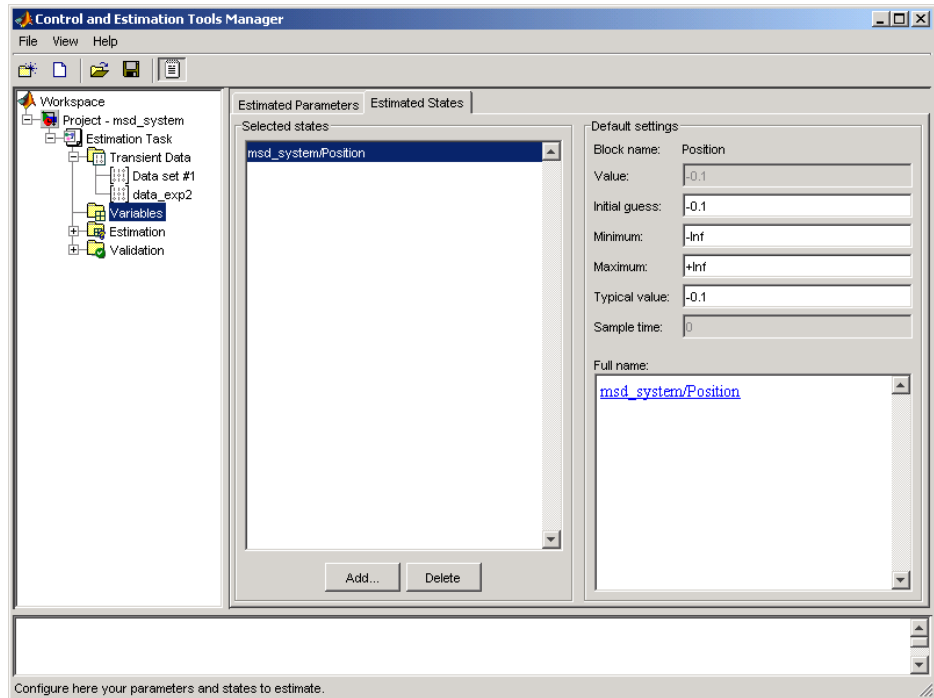
Creating the Estimation Task

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

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

The initial position estimates for the two data sets are known to differ, but set the initial state guesses for both data sets to -0.1.
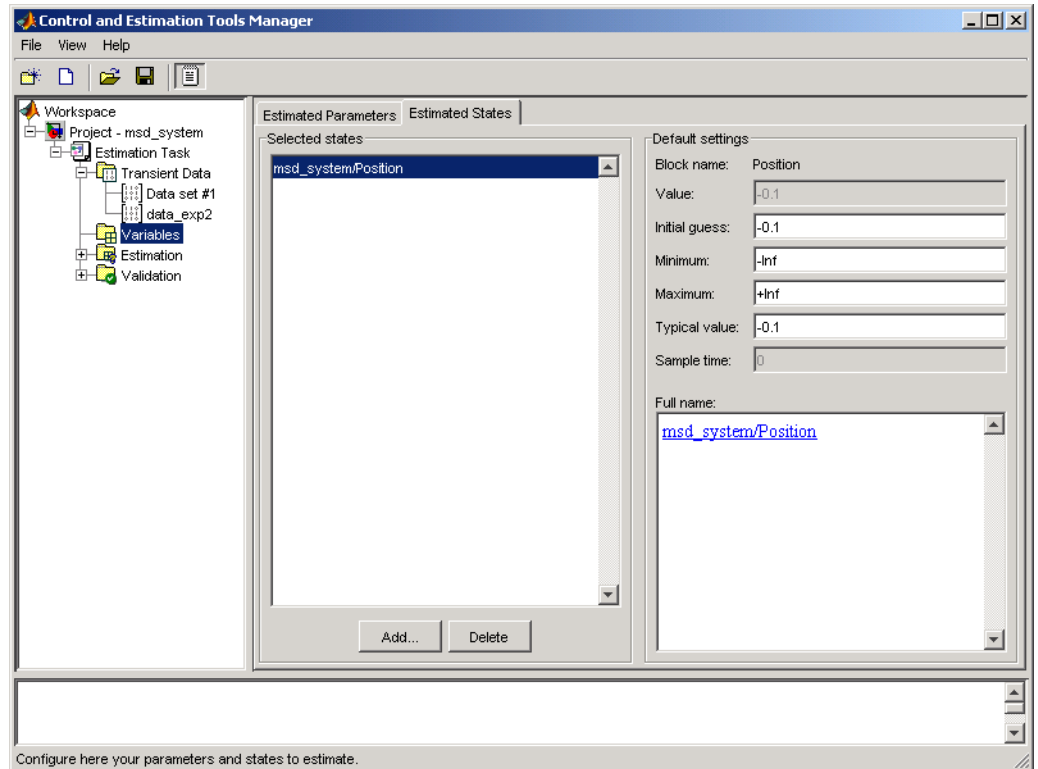
Running the Estimation and Viewing Results

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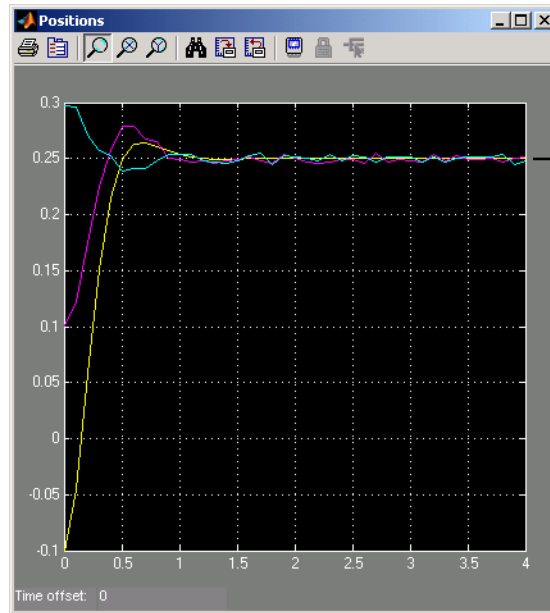
The simulated data is a good match to the measured (experimental) data.

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

3

Preprocessing Data

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.

You can overwrite the existing data with the preprocessed data, or store it in a new file.

Data Preprocessing Tool

Simulink Parameter Estimation provides a GUI for data preprocessing, which is the Data Preprocessing Tool. To open it:

1 Open the Control and Estimation Tools Manager.

2 Select the **Transient Data** node in the workspace directory tree, and then choose the data you want to modify in the **Input Data**, or **Output Data** pane. This enables the **Pre-process** button.
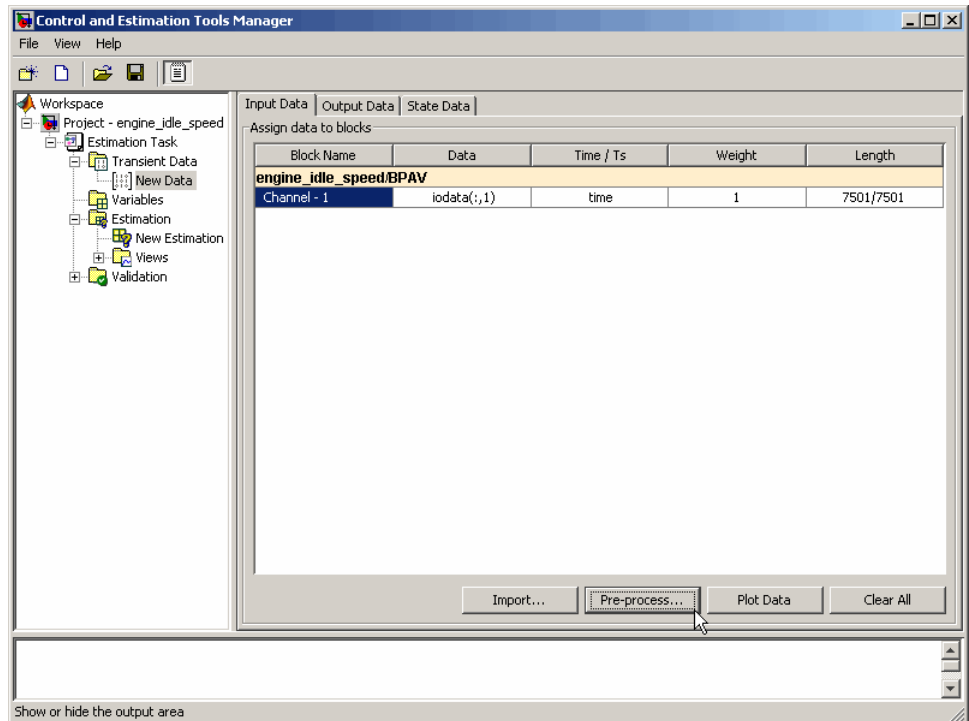
3 Click the **Pre-process** button to open the Data Preprocessing Tool.
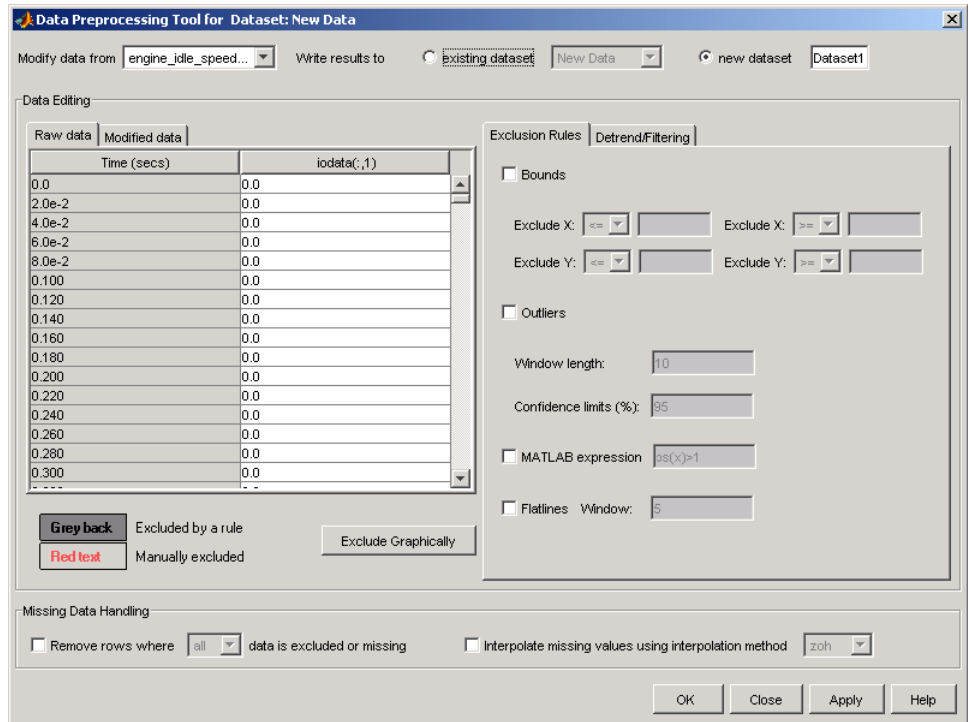
In this chapter, the sample data set is the same as used in the
engine_idle_speed Simulink model. See "How Simulink Parameter
Estimation Works" on page 1-6 for an overview of creating estimation projects
and adding data sets.

With the Data Preprocessing Tool, 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

Techniques for Excluding Data in the Data Preprocessing Tool

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

- Selecting data from the Data Editing Table.

- Selecting data from a plot of the Data.

- Specifying a Rule.

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

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

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.

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

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

In the **Modified data** pane, you can choose to remove the excluded data completely or interpolate it. See "Miscellaneous Data Handling" on page 3-16 for more information.

After you select data for exclusion, you can view it graphically by clicking **Exclude Graphically**.
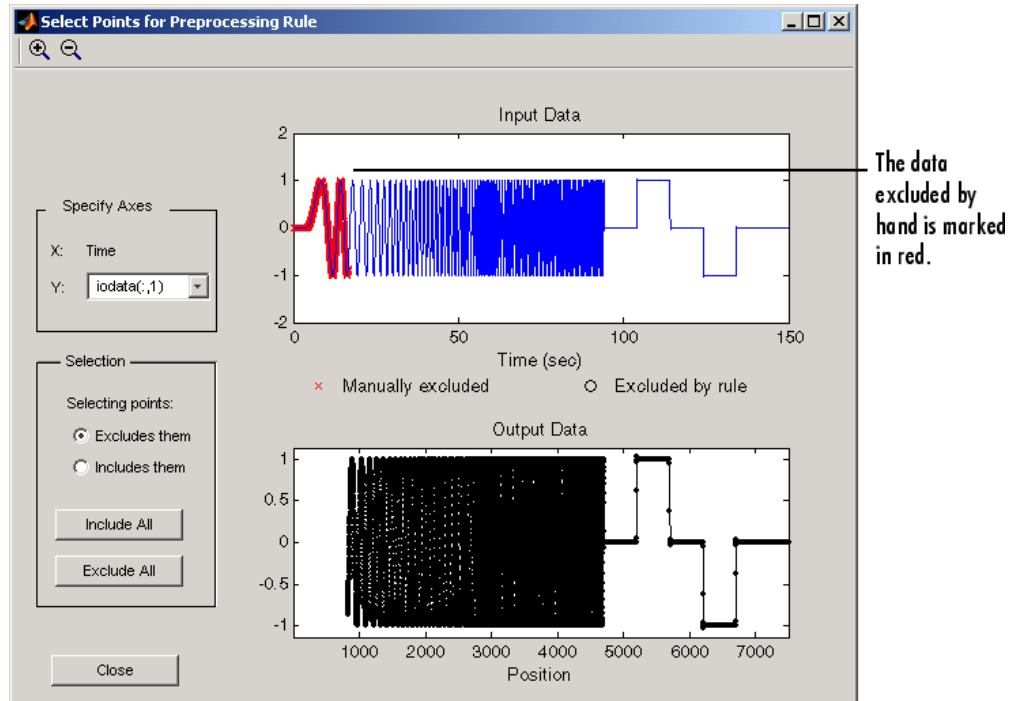
As you make changes in the **Data Editing** pane, they immediately appear in the Select Points for Preprocessing Rule window, and vice versa.

Selecting Data for Exclusion from a Plot of the Data

You can exclude data graphically. Click **Exclude Graphically** to open the Select Points for Preprocessing Rule window.
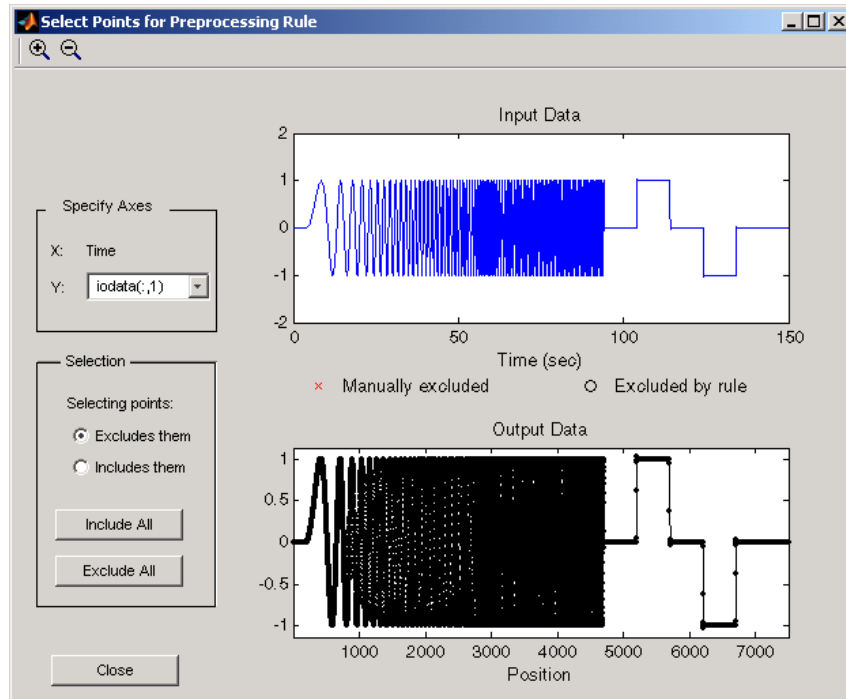
The way you exclude data is similar to the way you select a region for zooming: place your cursor in the **Input Data** plot and drag the mouse to draw a region of exclusion.

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

The excluded area is red in the Input Data plot.

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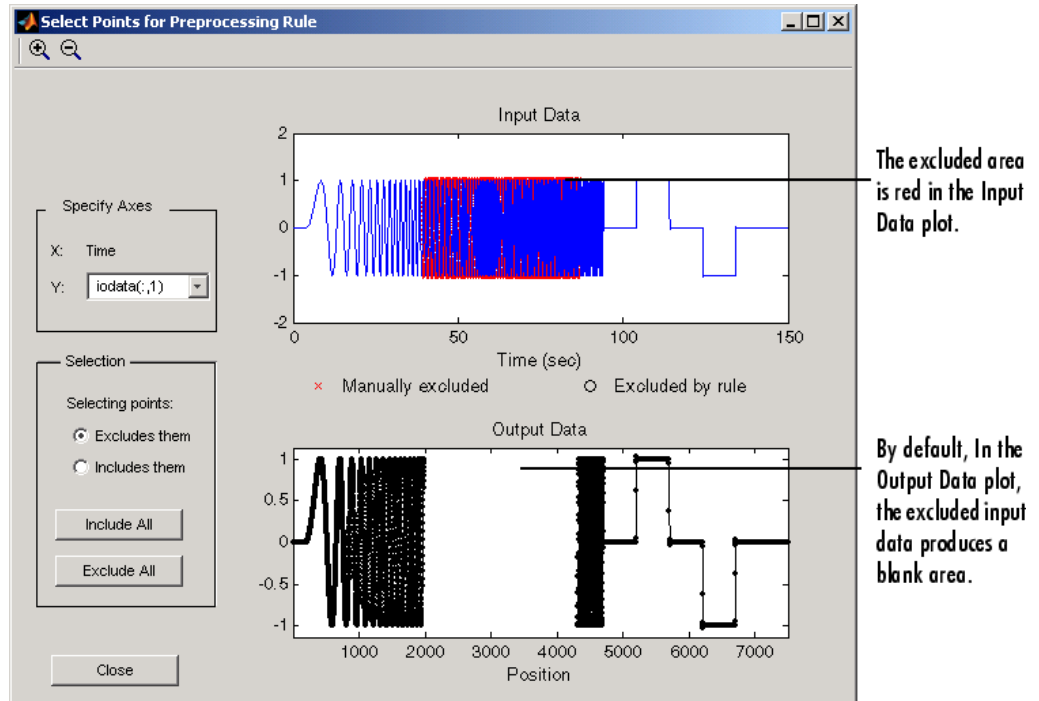
In the **Output Data** plot, the excluded input data produces a blank area by default. This corresponds to the NaNs that now represent excluded data. If you choose to interpolate or remove the excluded data, the output data shows the interpolated points.

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

Selection Pane

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

Use these radio buttons to toggle between including and excluding selected data.

Use these buttons to include or exclude all the data.
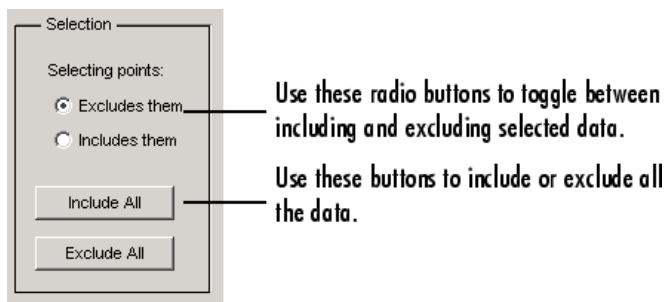
Selecting Data for Exclusion by a Rule

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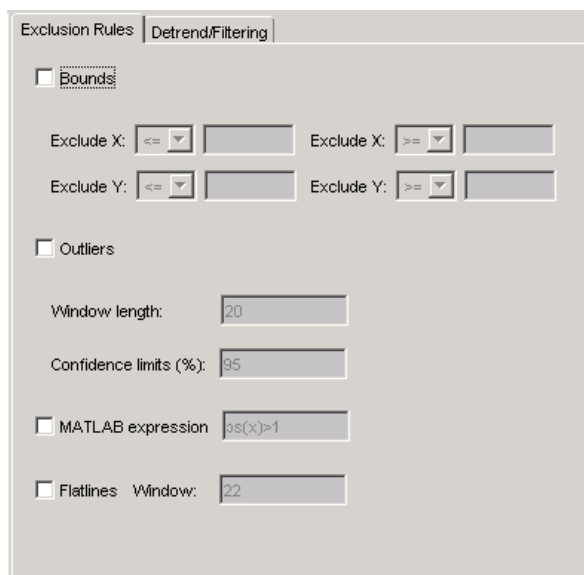
These are the rules you can use to exclude data:

- "Upper and Lower Bounds" on page 3-12
- "Outliers" on page 3-12

- "MATLAB Expressions" on page 3-12
- "Flatlines" on page 3-12

Upper and Lower Bounds

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

Outliers

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

MATLAB Expressions

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

Flatlines

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

Example of Rule Exclusion

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

The region of data excluded by rule is shaded gray.
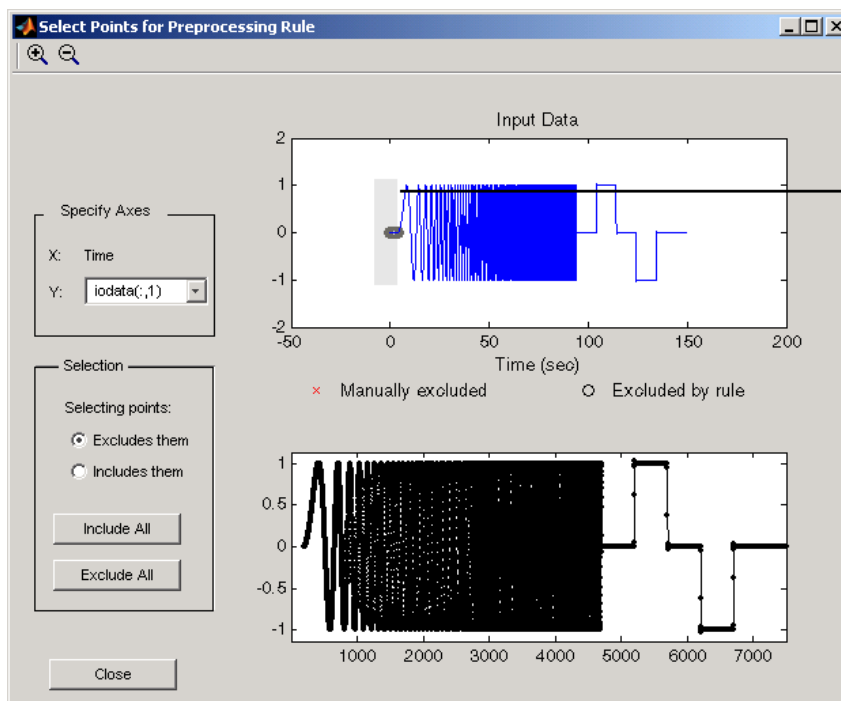
Detrending and Filtering

Detrending and Filtering Data in the Data Preprocessing Tool

You can both detrend and filter data using the **Detrend/Filtering** pane in the Data Preprocessing Tool.
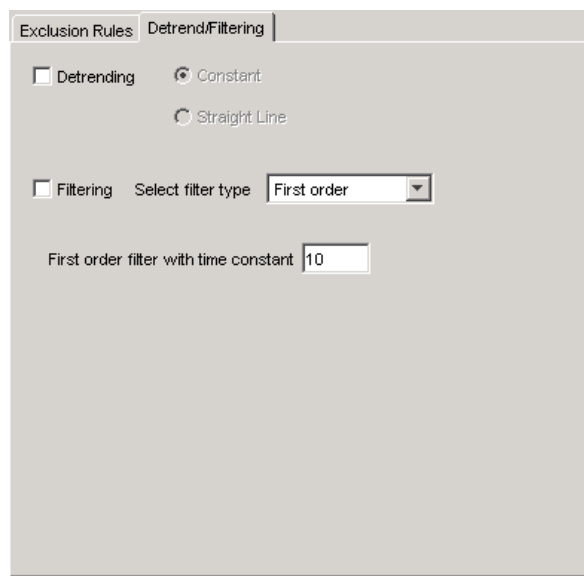
Detrending

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 (in the least-squares sense) and then removes them.

Filtering

You have these choices for filtering your data:

- First order — A filter of the type $\dfrac{1}{\tau s + 1}$

  where τ is the time constant that you specify in the associated field.

- Transfer function — A filter of the type

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

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

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

Miscellaneous Data Handling

In this section...
"Handling Missing Data" on page 3-16
"Loading Data and Saving Modified Data Sets" on page 3-16

Handling Missing Data

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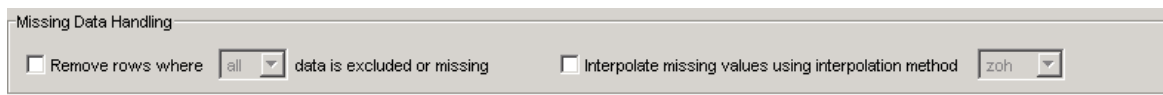
Removing Rows

If you select the **Remove rows where** check box, the affected rows are removed from the **Modified data** pane. 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). Select the **Interpolate missing values using interpolation method** check box and choose which method you want from the list. The results appear in the **Modified data** pane.

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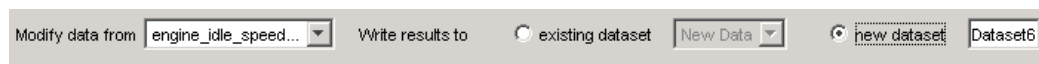
When you have multiple data sets, select the one you want to preprocess from the **Modify data from** list.

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

4

Managing Multiple 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.
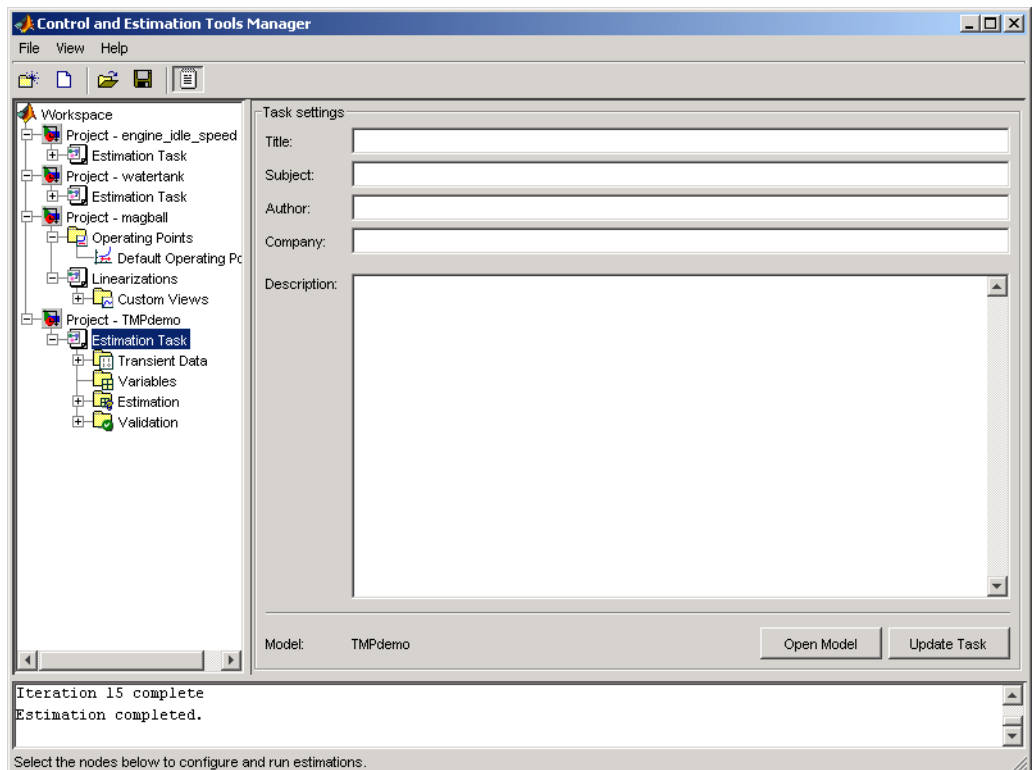
You can save projects individually, or group multiple projects together in one saved file. This chapter describes how to do this.

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 project as a MAT-file, select **File > Save** in the Control and Estimation Tools Manager window.
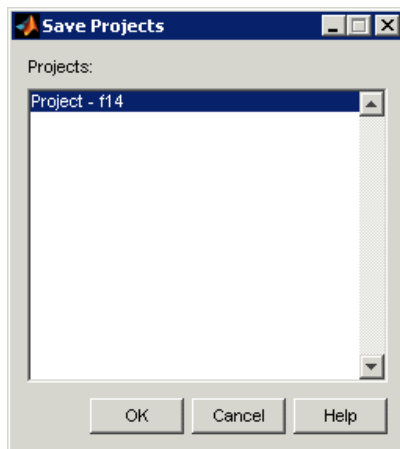
To save multiple projects within one file:

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

2 Click **OK**.

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

Opening Control and Estimation Tools Manager Projects

To open previously saved projects, select **File > Load** in the Control and
Estimation Tools Manager window.
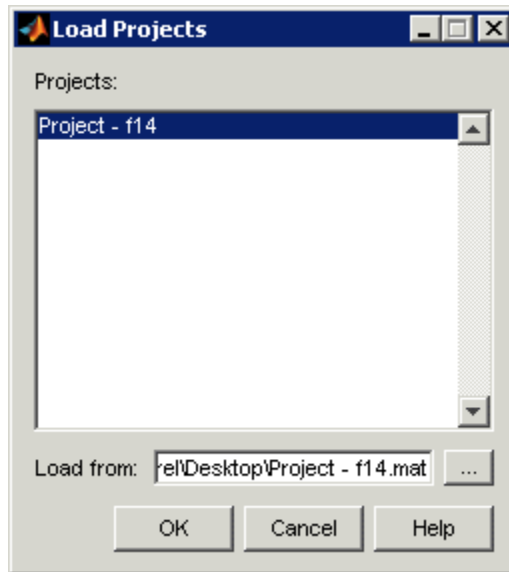
In the Load Projects dialog box, choose a project file by either browsing for
the directory and file, or by typing the full path and filename in the **Load
from** field. Project files are always MAT-files. The projects within this file
appear in the **Projects** list.

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

5

Adaptive Lookup Tables

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 previously, 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 previously, 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

Simulink Blocks for 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 next.
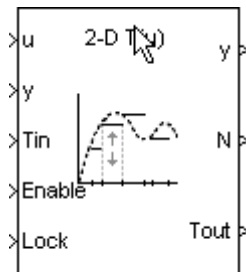
Adaptive Lookup Table Block Showing Inputs and Outputs

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 = [rpm, pressure] and y = [efficiency].

- The *initial table* data may be entered either as a dialog box parameter (by double-clicking the block) or as an input port (i.e., the input port Tin 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 box), and if required, the whole adapted table data (Tout).

Adaptive Lookup Table Library

Three adaptive lookup tables are available in Simulink Parameter Estimation. To access the Adaptive Lookup Tables library, type the following command at the MATLAB prompt:

    spelookup
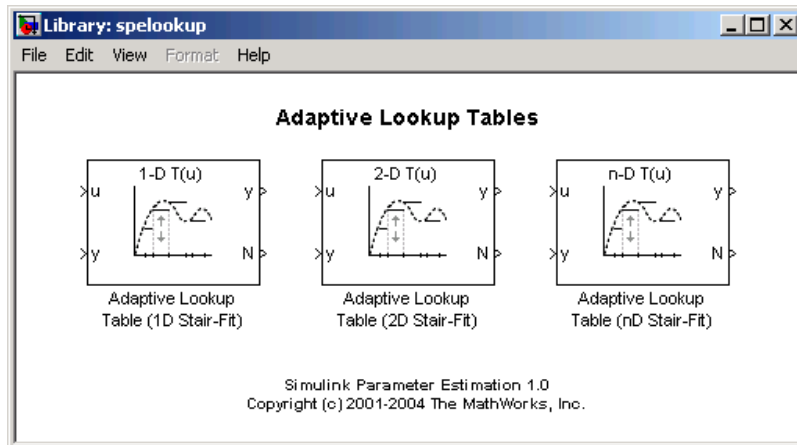
The three blocks are

- Adaptive Lookup Table (1D Stair-Fit) — One-dimensional adaptive lookup

- Adaptive Lookup Table (2D Stair-Fit) — Two-dimensional adaptive lookup

- Adaptive Lookup Table (nD Stair-Fit) — 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 next.
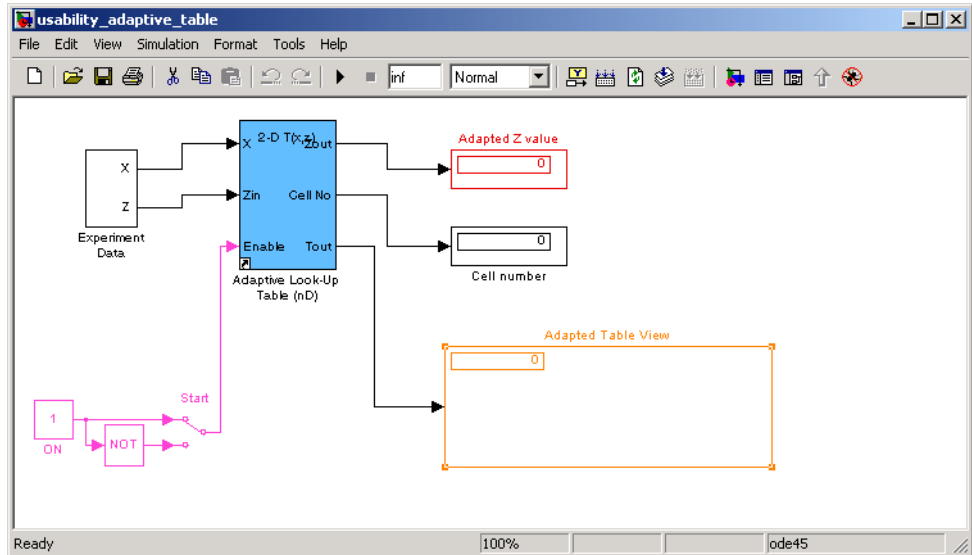
Simulink Diagram Using an Adaptive Lookup Table

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®, 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

The Adaptive Lookup Table parameters are configured in the Function Block Parameters dialog box. Double-click the Adaptive Lookup Tables Simulink block to open this dialog box.

Adaptive lookup table dimensions

Block input values for input variables

Initial table output values

Number values assigned to cells

Algorithm for adaptation

Assign weight to new data

Customize block I/O channels

Allow adaptation for out-of-range data
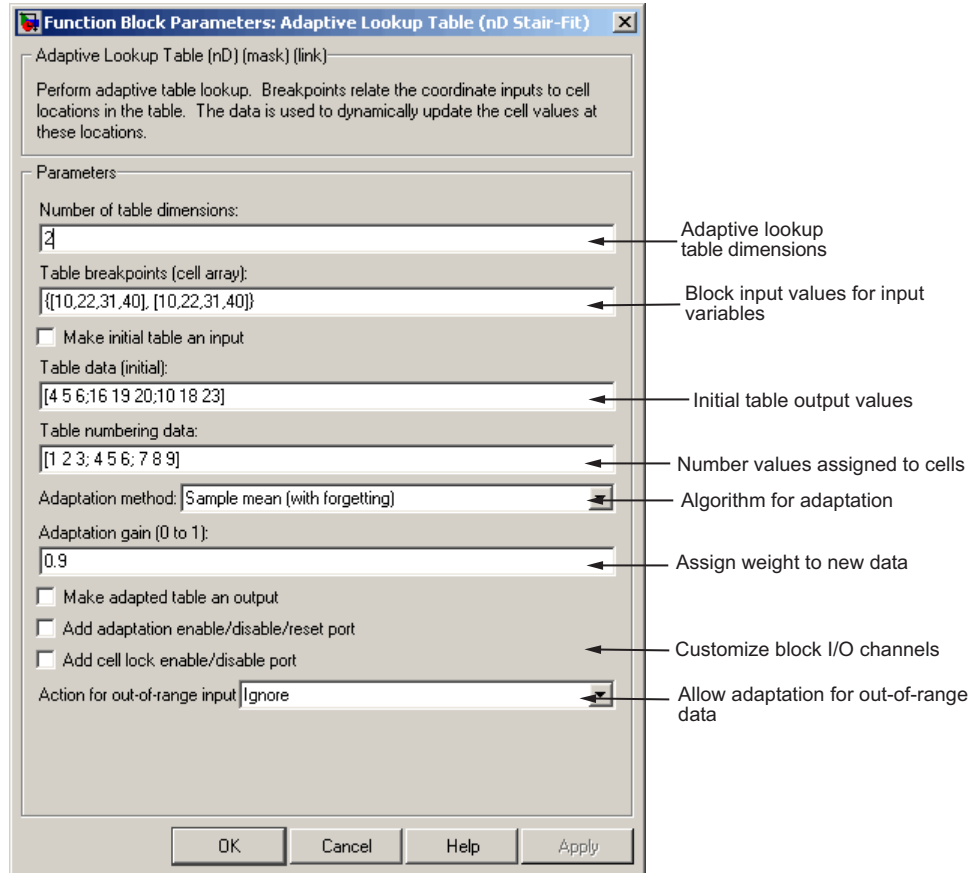
n-D Adaptive Lookup Table Dialog Box

For details on how to set these parameters, see the individual Chapter 7, "Blocks — Alphabetical List" reference pages.

Selecting an Adaptation Method

You can choose Sample mean or Sample mean (with forgetting) as the **Adaptation Method** in the Function Block Parameters dialog box. This section discusses the details of these algorithms.

Sample Mean

Sample mean provides the average value of n output data samples and is defined as:

$$y(n) = \frac{1}{n}\sum_{i=1}^{n} d(i)$$

where $d(\text{i})$ is the ith measurement collected within a particular *cell*. For each input data pair($x1$, $x2$), the sample mean at the corresponding cell is updated using the output data measurement, d. Instead of accumulating n samples of data for each cell, a recursive relation is used to calculate the sample mean. The recursive expression is obtained as follows:

$$y(n) = \frac{1}{n}\left[\sum_{i=1}^{n-1} d(i) + d(n)\right] = \frac{n-1}{n}\left[\frac{1}{n-1}\sum_{i=1}^{n-1} d(i)\right] + \frac{1}{n}d(n) = \frac{n-1}{n}y(n-1) + \frac{1}{n}d(n)$$

where $d(\text{n})$ is the nth data sample. Defining *a priori estimation error* as $e(n) = d(n) - y(n-1)$, the recursive relation can be written as:

$$y(n) = y(n-1) + \frac{1}{n}e(n)$$

where $n \geq 1$ and the initial estimate $y(0)$ is arbitrary.

In this expression, instead of n data samples, only the number of samples, n, for each cell is stored in memory.

Sample Mean with Forgetting

The adaptation method "Sample Mean" on page 5-9 has an *infinite memory*. The past data samples have the same weight as the final sample in calculating the sample mean. Sample mean (with forgetting) uses an algorithm with a *forgetting factor* or **Adaptation gain** that puts more weight on the more recent samples. This algorithm provides robustness against initial response transients of the plant and an adjustable speed of adaptation. Sample mean (with forgetting) is defined as:

$$y(n) = \frac{1}{\sum_{i=1}^{n} \lambda^{n-i}} \sum_{i=1}^{n} \lambda^{n-i} d(i) = \frac{1}{\sum_{i=1}^{n} \lambda^{n-i}} \left[\sum_{i=1}^{n-1} \lambda^{n-i} d(i) + d(n) \right] = \frac{s(n-1)}{s(n)} y(n-1) + \frac{1}{s(n)} d(n)$$

where $\lambda \in [0,1]$ is the **Adaptation gain** and $s(k) = \sum_{i=1}^{k} \lambda^{n-i}$. Defining *a priori estimation error* as $e(n) = d(n) - y(n-1)$, where $n \geq 1$ and the initial estimate $y(0)$ is arbitrary, the recursive relation can be written as:

$$y(n) = y(n-1) + \frac{1}{s(n)} e(n) = y(n-1) + \frac{1-\lambda}{1-\lambda^n} e(n)$$

A small value of λ results in faster adaptation. A value of 0 indicates short memory (last data becomes the table value), and a value of 1 indicates long memory (average all data received in a cell).

Example: n-D Adaptive Lookup Table

In this section...

"Loading the Example" on page 5-11

"Running the Example" on page 5-12

Loading the Example

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

```
enginetable
```

at the MATLAB prompt to open this model.
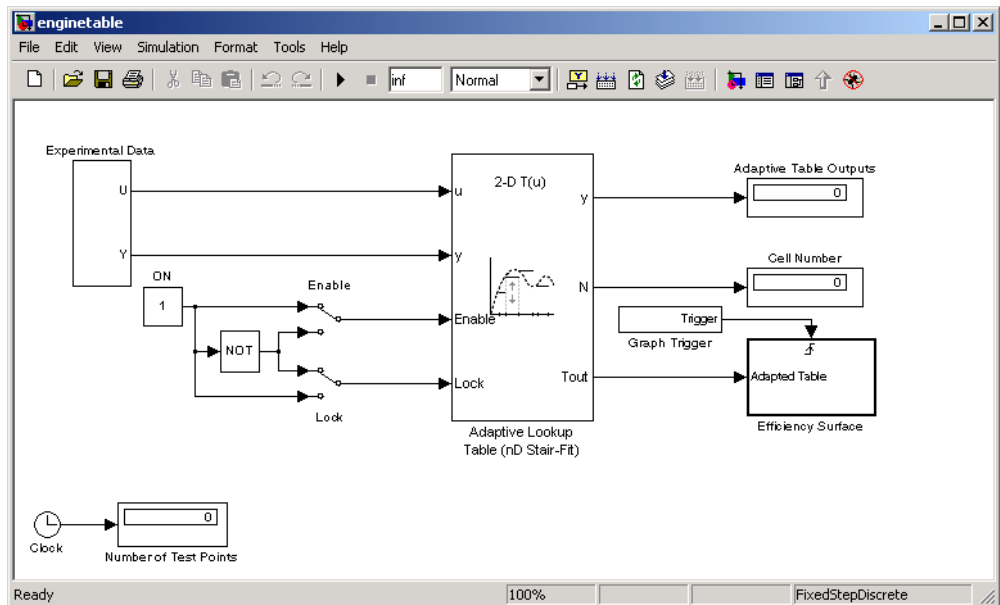
This model has several key features:

- 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, select **Simulation > Start**. Or, on Microsoft Windows, click the **Start Simulation** button ▶ in the Simulink toolbar.

The simulation begins by populating the adaptive lookup table with random data. This figure shows the input and adaptive data side by side.
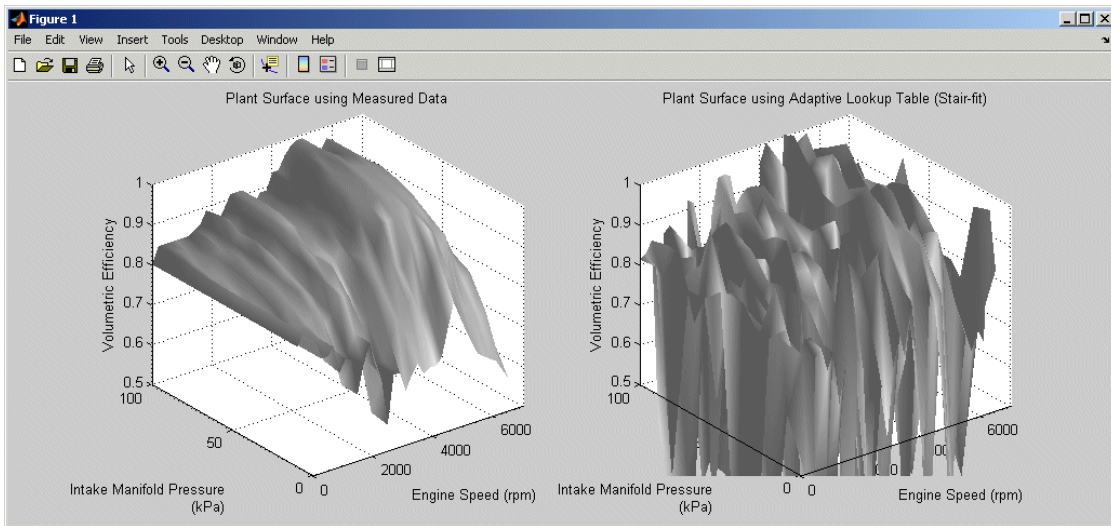
As the simulation progresses, the surface on the right adapts to match the measured input data. This figure shows the final adaptation.
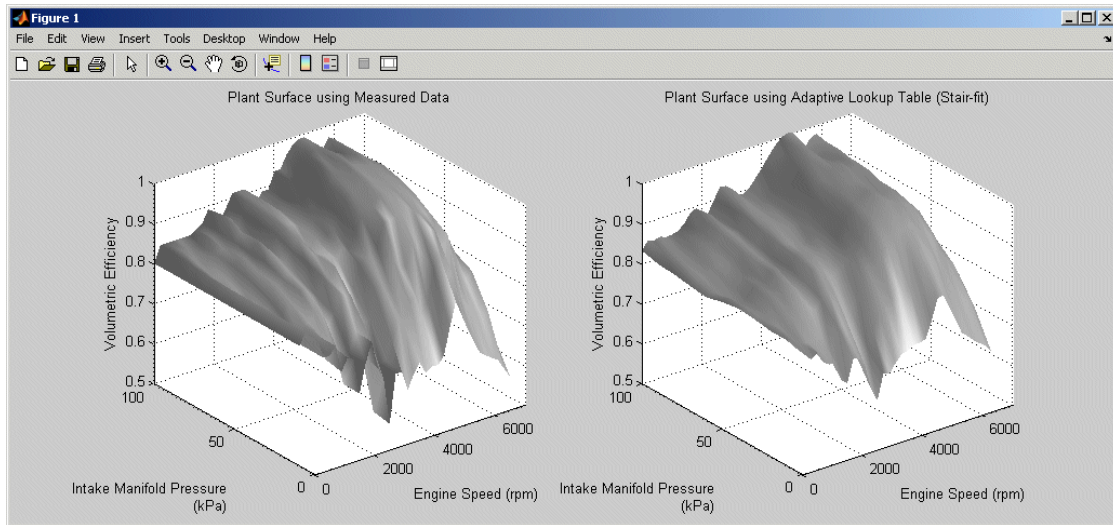
The fit is quite good. Try using the enable and lock features to see how they change the adaptation.

6

Estimating from the Command Line

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 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 the MATLAB Programming documentation for a description 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:

1 Defining experiments consisting of empirical data sets and the operating conditions and/or initial conditions of your model

2 Selecting the variables and states to be estimated

3 Performing the estimation

4 Reviewing the results and iterating as necessary

5 Validating estimation results

The following sections discuss these topics:

- "Example: Estimating Parameters and Initial Conditions of the F14 Model" on page 6-5 — How to perform the estimation process using command-line functions
- "Creating and Customizing Estimation Projects" on page 6-15 — How to use methods and properties to customize your estimation project's features

Example: Estimating Parameters and Initial Conditions of the F14 Model

In this section...
"Loading the F14 Jet Model" on page 6-5
"Baseline Simulation" on page 6-6
"Creating a Transient Experiment Object" on page 6-8
"Assigning Experimental Data to Inputs and Outputs of the Model" on page 6-9
"Creating Parameter Objects for Estimation" on page 6-10
"Creating an Estimation Object and Running the Estimation" on page 6-11

Loading the F14 Jet Model

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

```
f14
```

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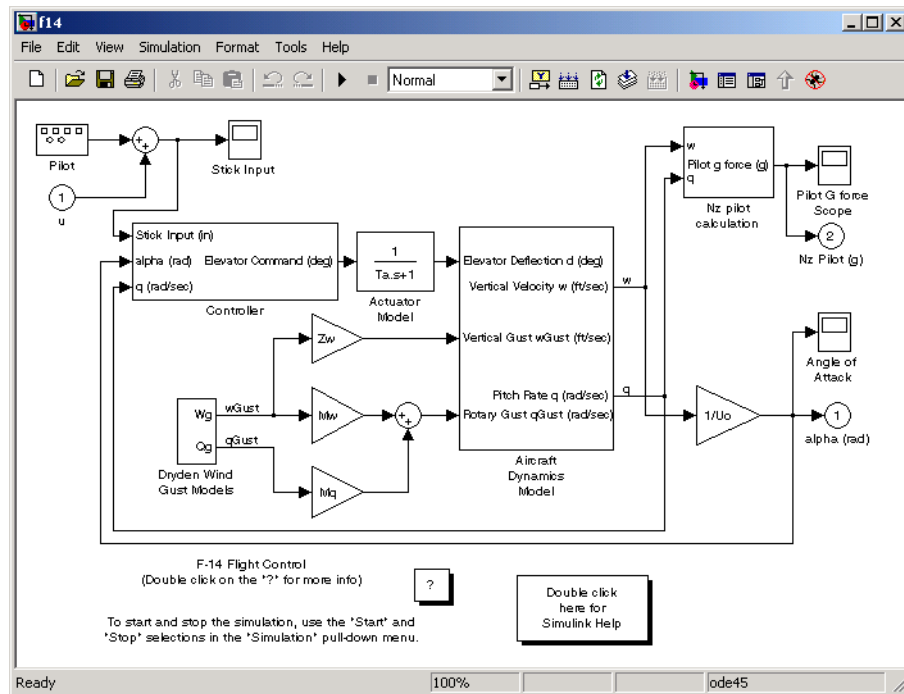
F14 Fighter Jet Model

This example outlines the basics of constructing an estimation project using object-oriented code. Only what you need to run the example is presented in this section. See "Creating and Customizing Estimation Projects" on page 6-15 for details on all the properties and methods associated with parameter estimation.

Baseline Simulation

Before running an estimation, you need a baseline for data comparison. First, you must choose parameters and states' initial conditions for estimation. This example uses Ta, the actuator time constant, and Zd and Md, the vertical velocity and pitch rate gains, respectively. Then use the following code to run the Simulink f14 model. Note that this is standard Simulink code and does not involve Simulink 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: Ta = 0.05)
Ta = 0.5;

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

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

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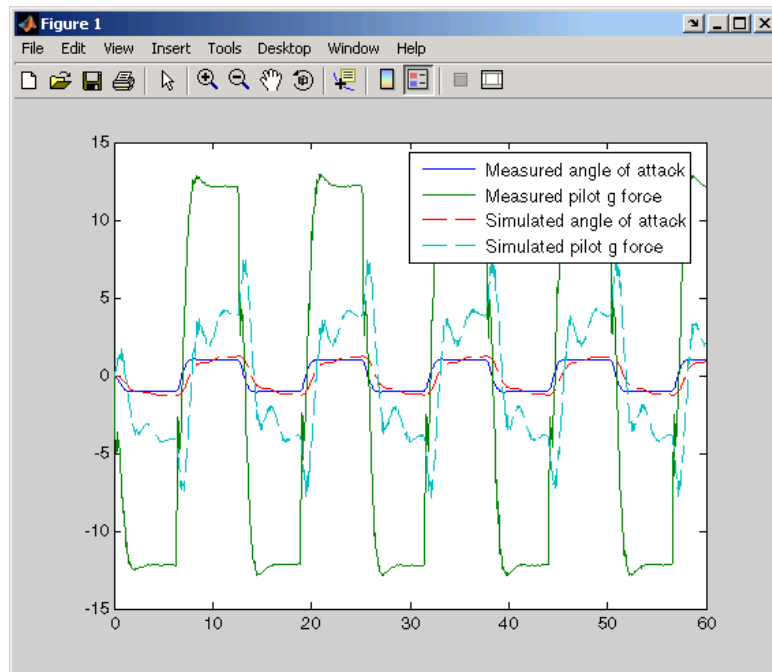
As you can see, the measured and simulated data are a poor match. The rest of this section describes how to estimate values for Ta, Zd, and Md that result in a better match of data sets.

Creating a Transient Experiment Object

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

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

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

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

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

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

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

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

Creating Parameter Objects for Estimation

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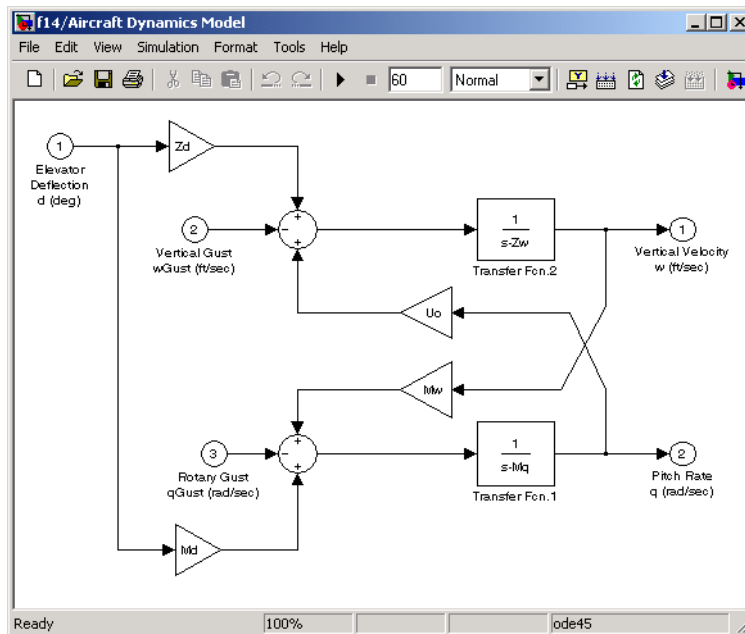
First, create ParameterEstimator objects for the parameters you want to estimate.

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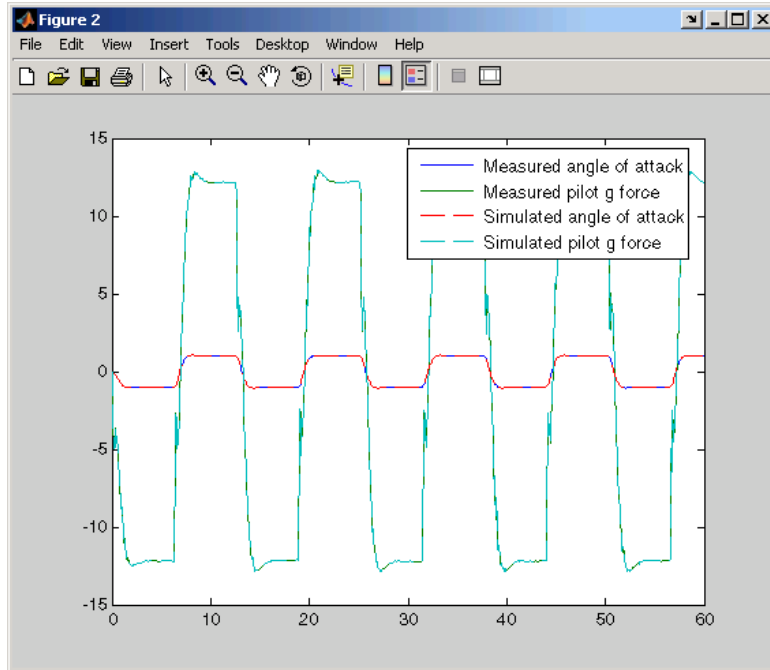
The measured and simulated outputs now appear to be a close match. Next, look at the estimated values to see how they compare with the default values of the f14 model.

```
%% 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:
```

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

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

finds only scalar estimated parameters. However,

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

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

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

```
clear all
f14
whos
```

Creating and Customizing Estimation Projects

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

First, a quick look at terminology:

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

Creating Transient Data Objects

What is a Transient Data Object

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

- Inport blocks

- Outport blocks

- Internal blocks used in conjunction with signal logging.

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

Constructor

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

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

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

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

Properties of Transient Data Objects

Descriptions of properties of the transient data object and the associated input parameters are as follows.

Transient Data Object Properties

Property	Description
Block	Name of the Simulink block with which the data is associated. Must be a string.
PortType	The type of signal that this object represents is determined in the constructor from the Block property, which may be Inport, Outport, or Signal.
PortNumber	For data associated with the outputs of regular blocks or subsystems, this property specifies the output port number of interest. The default value is 1.
Dimensions	Dimensions of the data required for this data set. It is computed from the CompiledPortDimensions property of the appropriate port of the block, and it defines the size of other properties. Currently, Simulink supports scalar, vector, or matrix signals, so Dimensions is either a scalar or a 1-by-2 array.
Data	Actual experimental data. Its size must be consistent with the Dimensions property. To conform with Simulink conventions, the data is stored in the following formats: • Scalar or vector-valued data. The data is of the form Ns m, where Ns is the number of data samples, and m is the number of channels in the signal. • Multidimensional data (matrix and higher dimensions). The data is of the form $m1 \ . \ . \ . \ mn \ Ns$, where Ns is the number of data samples, and mi is the number of channels in the ith dimension of the signal. • For missing or unspecified data, NaNs are used.

Transient Data Object Properties (Continued)

Property	Description
Ts, Tstart, Tstop	For uniformly sampled data, Ts is the sample time and Tstart is the start time of the signal. The stop time Tstop and the time vector Time are given by $Tstop = Tstart + Ts * (Ns\ \text{-}1)$ $Time = Tstart : Ts : Tstop$ For nonuniform time data, Ts is set to NaN, and the start and stop times are calculated from the time vector.

Time	The time data in column vector format. The length of Time must be consistent with the number of samples in Data.
	
	For a nonuniformly spaced Time vector, its length should match the length of Data.
	
	Otherwise, Time is automatically adjusted based on the length of Data.
	
	Modifying Ts resets Time internally. In this case, Time is a virtual property whose value is computed from Ts and Tstart when you request it. The rules for setting time related properties are
	
	• Modifying Time sets
	
	    `Ts = NaN`
	    `Tstart = Time(1)`
	
	• If the time vector is uniformly spaced, a sample time Ts is calculated.
	
	• Modifying Tstart translates time forward or backward.
	
	• Modifying Ts sets Time = [] internally and generates it when required by the simulation.

Transient Data Object Properties (Continued)

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

Modifying Properties of Transient Data Objects

After a transient data object is created, you can modify its properties using this syntax:

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

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

Example: Assigning Input Port Data

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

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

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

Descriptions of two important methods are given next:

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

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

  To extract data from a subset of available channels, use

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

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

Creating State Data Objects

What is a State Data Object

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

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

Constructor

The syntax for creating this object is

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

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

Properties of the State Data Object

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

State Data Object Properties

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

Example: Initial Condition Data

To create an empty initial condition object for the engine_idle_speed/TransferFcn2, use

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

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

Modifying Properties

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

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

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

Using Class Methods

Description of two important methods are given next:

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

Creating Transient Experiment Objects

What is a Transient Experiment Object

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

Constructor

The syntax to create a transient experiment object is

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

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

Properties of Transient Experiment Objects

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

Transient Experiment Object Properties

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

Example: Creating an F14 Experiment

To create an empty transient experiment for the f14 model, use

```
exp1 = ParameterEstimator.TransientExperiment('f14')
Experimental (Transient) data set for the model 'f14':
Outputs
(1) f14/alpha (rad)
(2) f14/Nz Pilot (g)
Inputs
(1) f14/u
Initial States
(1) f14/Actuator Model
(2) f14/Aircraft Dynamics Model/Transfer Fcn.1
(3) f14/Aircraft Dynamics Model/Transfer Fcn.2
(4) f14/Controller/Alpha-sensor Low-pass Filter
(5) f14/Controller/Pitch Rate Lead Filter
```

```
(6) f14/Controller/Proportional plus integral compensator
(7) f14/Controller/Stick Prefilter
(8) f14/Dryden Wind Gust Models/Q-gust model
(9) f14/Dryden Wind Gust Models/W-gust model
```

Example: Creating a Van der Pol Experiment from User Objects

To create a transient experiment from user objects for I/Os and states, use

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

Modifying Properties

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

Using Class Methods

The description of one important method is given next:

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

Creating Parameter Objects

In this section...

What is a Parameter Object

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

Constructor

The syntax to create a parameter object is

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

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

Properties of Parameter Objects

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

Parameter Object Properties

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

Parameter Object Properties (Continued)

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

Example: F14 Model

To create a parameter object for the parameter Ta in the f14 model, use

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

Example: Gain Matrix

To create a parameter object for a matrix parameter K of size 4-by-1, use

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

Modifying Properties

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

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

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

Using Class Methods

Descriptions of two important methods are given next:

- `hiliteBlock` — Highlights the referenced blocks associated with parameter objects in the Simulink diagram.

- `update` — Updates the parameter object after the Simulink model has been modified. If the size of the `Value` property changes, then the other properties are reset to their default values.

Creating State Objects

In this section...

What is a State Object

The @State object is similar to the @Parameter object. One @State object corresponds to each Simulink block with states in the model.

Constructor

The syntax to create a state object is

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

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

Properties of State Objects

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

State Object Properties

Property	Description
Block	Name of the Simulink block whose states are defined by this object.
Dimensions	Scalar value to store the number of states of the relevant block.
Value	Column vector to store the value of the state for the block specified by this object. The length of this vector should be consistent with the Dimensions property.
Estimated	A Boolean array of the same size as that of Value. Depending on the value of the elements of the Estimated property, the behavior of the corresponding elements of Value is as follows: • The elements of Value are estimated if the corresponding elements in Estimate are set to true. The result is stored in the Value property. • The elements of Value are not estimated if the corresponding elements in Estimated are set to false. However, these elements are used to reset the corresponding states during estimations. This property is set to false by default, meaning that the state value is not estimated.

State Object Properties (Continued)

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

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

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

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

Using Class Methods

Description of two important methods are given next:

- `hiliteBlock` — Highlights the referenced blocks associated with state objects in the Simulink diagram.

- `update` — Updates the state object after the Simulink model has been modified. If the size of `Value` property changes, then the other properties are reset to their default values.

Creating Estimation Objects

What is an Estimation Object

The @Estimation object is the coordinator between the model, experiments, and parameter objects.

Constructor

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

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

Properties of Estimation Objects

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

Estimation Object Properties

Property	Description
Model	Name of the Simulink model with which this estimation is associated.

Estimation Object Properties (Continued)

Property	Description
Experiments	Experiments to be used in estimations. For multiple experiments, the cost function uses a concatenation of the output error vectors obtained using each experimental data set.
Parameters	Parameter objects to be used in estimations.
States	State objects to be used in estimations. This is a handle matrix with as many columns as there are experiments, and as many rows as there are states in Model. The handle matrix is created automatically in the constructor. You can reorganize its rows to specify shared states between experiments, and set the Estimated flag of desired states. If state data is provided in an experiment, the state objects stored in the columns of this matrix are initialized from the experiments.
SimOptions	Same as simset structure. This property is initialized to simget(this.Model).
OptimOptions	Same as optimset structure.
EstimInfo	This property is used to store estimation-related information at each iteration of the optimizer, and is initialized as this.EstimInfo = struct('Cost', [],... 'Covariance', [],... 'FCount', [],... 'FirstOrd', [],... 'Gradient', [],... 'Iteration', [],... 'Procedure', [],... 'StepSize', [],... 'Values', []);

Example: F14 Model

To create an estimation object for the f14 model to estimate the parameters Ta and Kf and two states, use

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

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

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

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

Using Class Methods

Descriptions of some of the important methods are given next:

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

7

Blocks — Alphabetical List

Adaptive Lookup Table (1D Stair-Fit)

Purpose Perform one-dimensional adaptive table lookup
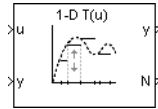
Description

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) block to model time-varying systems.

Data Type Support Doubles only
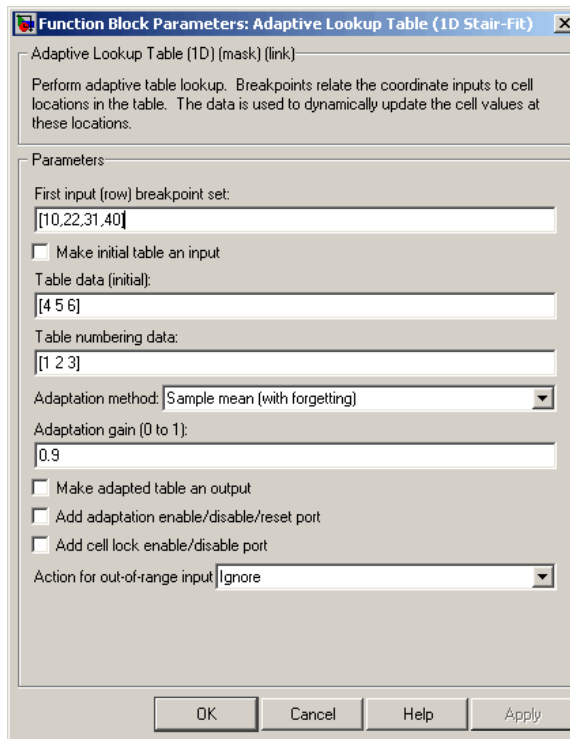
Dialog Box

First input (row) breakpoint set

    The vector of values containing possible block input values. The input vector must be monotonically increasing.

Make initial table an input

    Selecting this check box forces the Adaptive Lookup Table (1D Stair-Fit) block to ignore the **Table data (initial)** parameter. 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.

Adaptive Lookup Table (1D Stair-Fit)

Table numbering data

> Number values assigned to cells. This vector must be the same size as the table data vector, and each value must be unique.

Adaptation method

> Choose `Sample mean` or `Sample mean (with forgetting)`. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see "Selecting an Adaptation Method" on page 5-8.

Adaptation gain (0 to 1)

> A number between 0 and 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output

> Selecting this check box creates an additional output port 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.
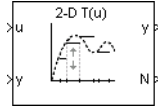
Action for out-of-range input

> `Ignore` or `Adapt` by extrapolating beyond the extreme breakpoints.

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) block to model
time-varying systems.

Adaptive Lookup Table (2D Stair-Fit)

Dialog Box

Function Block Parameters: Adaptive Lookup Table (2D Stair-Fit)

Adaptive Lookup Table (2D) (mask) (link)

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters

First input (row) breakpoint set:

[10,22,31,40]

Second input (column) breakpoint set:

[10,22,31,40]

☐ Make initial table an input

Table data (initial):

[4 5 6;16 19 20;10 18 23]

Table numbering data:

[1 2 3; 4 5 6; 7 8 9]

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1):

0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input Ignore

OK   Cancel   Help   Apply

First input (row) breakpoint set
>The vector of values containing possible block input values for the first input variable. The first input vector must be monotonically increasing.

Second input (column) breakpoint set
>The vector of values containing possible block input values for the second input variable. The second input vector must be monotonically increasing.

Make initial table an input

Selecting this check box forces the Adaptive Lookup Table (2D Stair-Fit) block to ignore the **Table data (initial)** parameter. 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 2-by-2 matrix must be of size (n-1)-by-(m-1), where n is the number of first input breakpoints and m is the number of second input breakpoints.

Table numbering data

Number values assigned to cells. This matrix must be the same size as the table data matrix, and each value must be unique.

Adaptation method

Choose Sample mean or Sample mean with forgetting. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see "Selecting an Adaptation Method" on page 5-8.

Adaptation gain (0 to 1)

A number from 0 to 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output

Selecting this check box creates an additional output port for the adapted table.

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

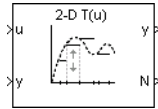Ignore or Adapt by extrapolating beyond the extreme breakpoints.

Purpose   Create adaptive lookup table of arbitrary dimension
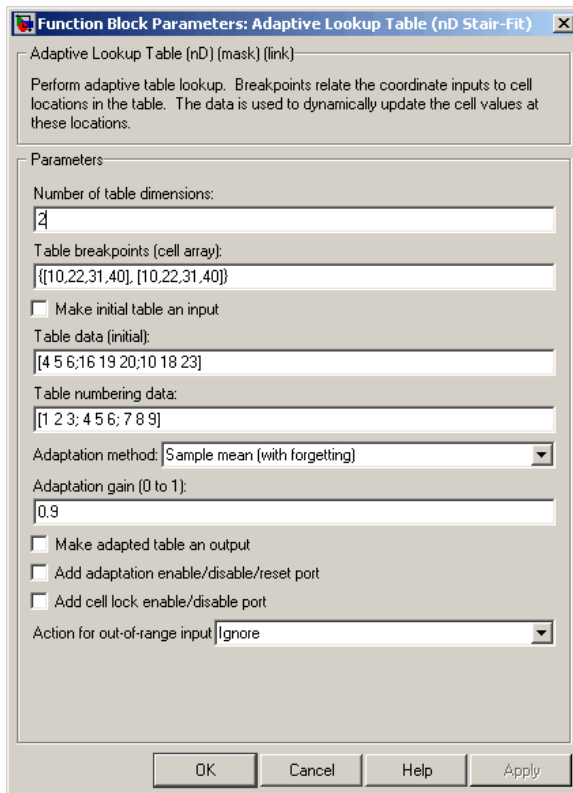
Description

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) block to model time-varying systems.

Adaptive Lookup Table (nD Stair-Fit)

Dialog Box

Perform adaptive table lookup. Breakpoints relate the coordinate inputs to cell locations in the table. The data is used to dynamically update the cell values at these locations.

Parameters:

Number of table dimensions:

`2`

Table breakpoints (cell array):

`{[10,22,31,40], [10,22,31,40]}`

☐ Make initial table an input

Table data (initial):

`[4 5 6;16 19 20;10 18 23]`

Table numbering data:

`[1 2 3; 4 5 6; 7 8 9]`

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1):

`0.9`

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

Action for out-of-range input: Ignore

OK   Cancel   Help   Apply

Number of table dimensions

The number of dimensions for the adaptive lookup table.

Table breakpoints (cell array)

A set of one-dimensional vectors that contains possible block input values for the input variables. Each input row must be monotonically increasing, but the rows do not have to be the same length. For example, if the **Number of table dimensions** is 3, you can set the table breakpoints as follows:

```
{[1 2 3], [5 7], [1 3 5 7]}
```

Make initial table an input

Selecting this check box forces the Adaptive Lookup Table (nD Stair-Fit) block to ignore the **Table data (initial)** parameter. 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 (n-D) array must be of size (n-1)-by-(n-1) ... -by- (n-1), (D times), where D is the number of dimensions and n is the number of input breakpoints.

Table numbering data

Number values assigned to cells. This vector must be the same size as the table data array, and each value must be unique.

Adaptation method

Choose Sample mean or Sample mean with forgetting. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter. For more information, see "Selecting an Adaptation Method" on page 5-8.

Adaptation gain (0 to 1)

A number from 0 to 1 that regulates the weight given to new data during the adaptation. A 0 means short memory (last data becomes the table value), and 1 means long memory (average all data received in a cell).

Make adapted table an output

Selecting this check box creates an additional output port for the adapted table.

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.

Functions — Alphabetical List

spetool

Purpose	Open Simulink Parameter Estimation GUI
Syntax	`spetool('modelname')`
Description	`spetool('modelname')` opens Simulink Parameter Estimation GUI for the Simulink model with the name `modelname`.
See Also	For more information on estimating parameters using Simulink Parameter Estimation GUI, see Chapter 1, "Getting Started".

Index