Simulink Parameter Estimation

For Use with Simulink®

- Modeling
- Simulation
- Implementation

User's Guide

Version 1

The MathWorks

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information
508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

Simulink Parameter Estimation User's Guide

Printing History: June 2004 First printing New for Version 1 (Release 14)

Contents

1 Introduction

What Is the Parameter Estimation Blockset 1-2

Installation 1-3

Demos 1-4

2 Getting Started

Introduction 2-2

Setting Up the Estimation Problem 2-3
- Importing Transient Data 2-5
- Specifying Operating Conditions 2-6
- Selecting Parameters for Estimation 2-8

Selecting Views for Plotting 2-9

Running an Estimation 2-10
- Adding Data Sets 2-10
- Specifying and Setting Up Parameters 2-11
- Opening the Estimation Window 2-12
- Running the Estimation 2-14

Setting Options for Optimization 2-17
- Selecting Optimization Methods 2-17
- Selecting Optimization Termination Options 2-18
- Selecting Additional Optimization Options 2-18
- Specifying the Cost Function 2-19

Setting Options for the Simulation 2-20

Selecting Solvers **2-20**

3 Adaptive Look-Up Tables

Lookup Tables **3-2**

Adaptive Lookup Tables **3-3**

Implementation of Adaptive Lookup Tables **3-4**
Adaptive Lookup Table Library **3-5**
Using Adaptive Lookup Tables in Simulink Models **3-6**
Real-Time Lookup Tables **3-6**
Setting Adaptive Lookup Table Parameters **3-7**

Example: N-D Adaptive Lookup Table **3-9**
Running the Example **3-10**

4 Estimating From the Command Line

Introduction **4-2**

Example: Estimating Parameters and Initial Conditions of the F14 Model **4-4**
Baseline Simulation **4-5**
Creating a Transient Experiment Object **4-6**
Creating Parameter Objects for Estimation **4-7**
Assigning Experimental Data to Inputs and Outputs of the Model **4-9**
Creating an Estimation Object and Running the Estimation **4-9**

Creating and Customizing Estimation Projects **4-12**

Creating a Transient Data Object **4-13**
Properties of Transient Data Objects **4-13**
Modifying Transient Data Object Properties **4-16**
Using Class Methods **4-16**
Helper Functions: Seven Ways to Represent
Sampling Instants **4-17**
Method for Logging Internal Block Signals. **4-18**

Creating Parameter Objects **4-19**
Constructor **4-19**
Properties of Parameter Objects **4-19**
Example: F14 Model **4-21**
Example: Gain Matrix **4-21**
Modifying Properties **4-21**
Using Class Methods **4-22**

Creating State Data Objects **4-23**
Properties of the State Data Object **4-23**
Example: Initial Condition Data **4-24**
Modifying Properties **4-25**
Using Class Methods **4-25**

Creating Transient Experiment Objects **4-26**
Properties of Transient Experiment Objects **4-27**
Example: Creating an f14 Experiment **4-27**
Example: Creating f14 Experiment Using Block Names **4-28**
Example: Creating Van der Pohl Experiment From
User Objects **4-28**
Modifying Properties **4-29**
Using Class Methods **4-29**

5 Block Reference

Index

Introduction

What Is the Parameter Estimation Blockset (p. 1-2)	A brief description of the product
Installation (p. 1-3)	How to install the Parameter Estimation Blockset
Demos (p. 1-4)	How to run Simulink Parameter Estimation demos

What Is the Parameter Estimation Blockset

The Parameter Estimation Blockset (PEB) is a Simulink® based product for estimating/calibrating model parameters from experimental data. This product supports

- DC Estimation—Tune DC parameters (e.g., resistance in RLC circuit) to best match the steady-state values observed at different operating points. Transients are ignored and trimming is used to obtain steady-state values from the Simulink model.
- Transient Estimation—Estimate parameters by comparing model output history and experimental data for a given input.
- Adaptive Look-Up Tables—Estimate the table values at the prescribed breakpoints using measurements from the physical system.

The Parameter Estimation Blockset provides graphical user interfaces (GUIs) to do the following:

- Set up the problem
- Specify which model parameters to estimate
- Import and preprocess the experimental data
- Follow the estimation progress
- Validate the estimation results through various plots

Installation

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

Demos

The Parameter Estimation Blockset 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 **Blocksets** and then **Parameter Estimation** to see a list of available demos. Alternatively, if you have the Help browser open, you can select the **Demos** pane directly and follow the same procedure.

Getting Started

Introduction (p. 2-2)	The estimation problem
Setting Up the Estimation Problem (p. 2-3)	What needs to be initialized to run an estimation
Running an Estimation (p. 2-10)	Steps involved in the actual estimation of parameters and initial conditions
Selecting Views for Plotting (p. 2-9)	Plotting cost functions
Setting Options for the Simulation (p. 2-20)	How to select simulation time and solvers for your Simulink model to use while estimation occurs
Setting Options for Optimization (p. 2-17)	Fine tuning the optimization process for your estimation

Introduction

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

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

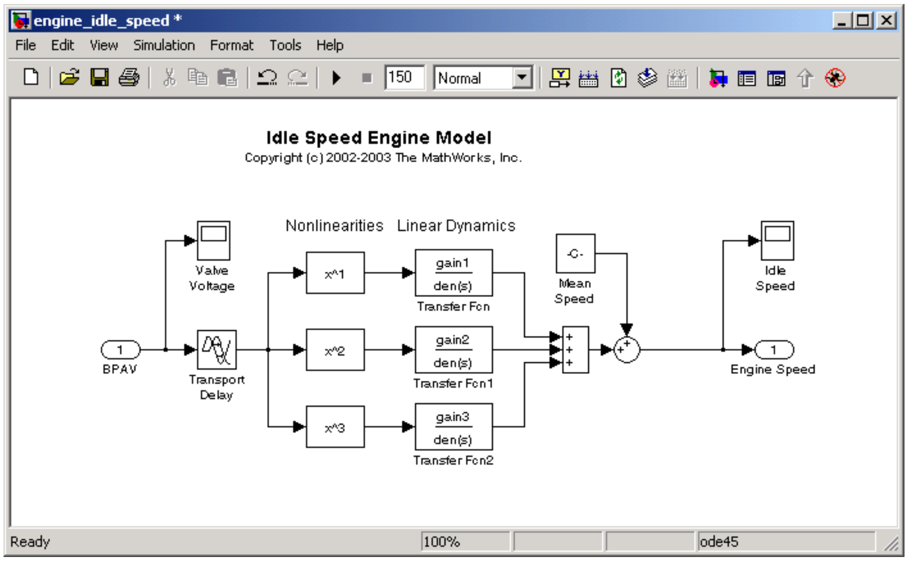
Setting Up the Estimation Problem

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

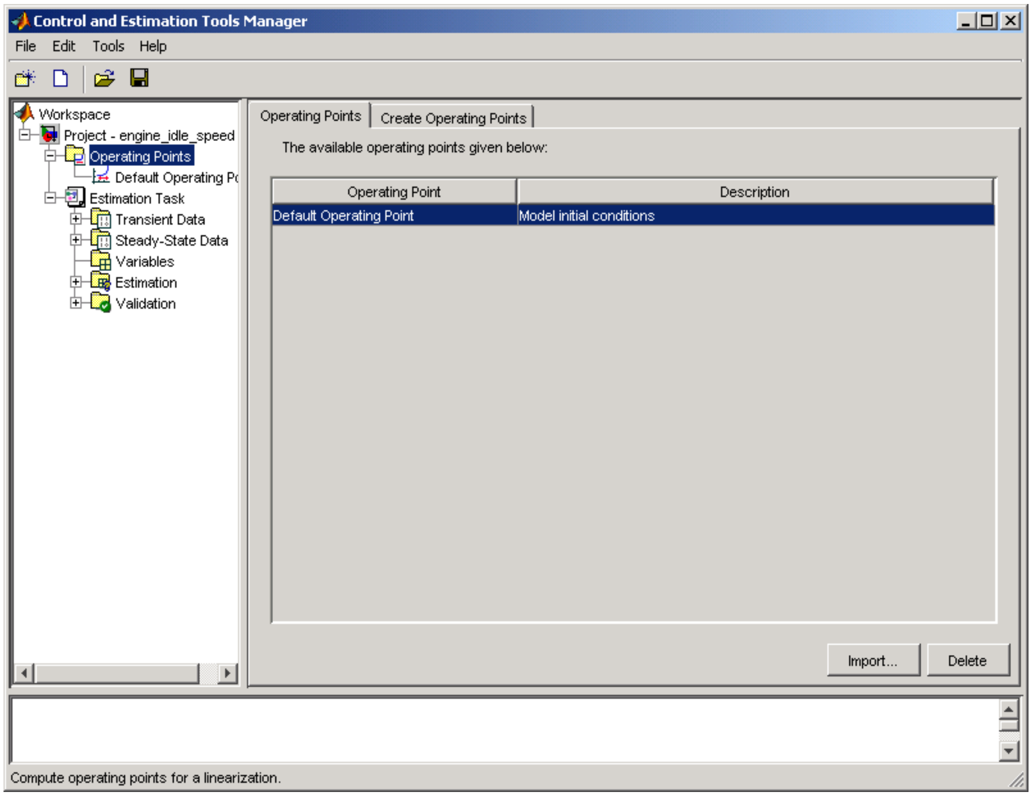
To show the steps of the setup, open a nonlinear model of an automotive engine's idle speed by typing

```
engine_idle_speed
```

at the MATLAB prompt. This model opens.



To open the **Control and Estimation Tools Manager,** select **Parameter Estimation** from the engine idle speed model's **Tools** menu.



The available operating points given below:

Operating Point	Description
Default Operating Point	Model initial conditions

Compute operating points for a linearization.

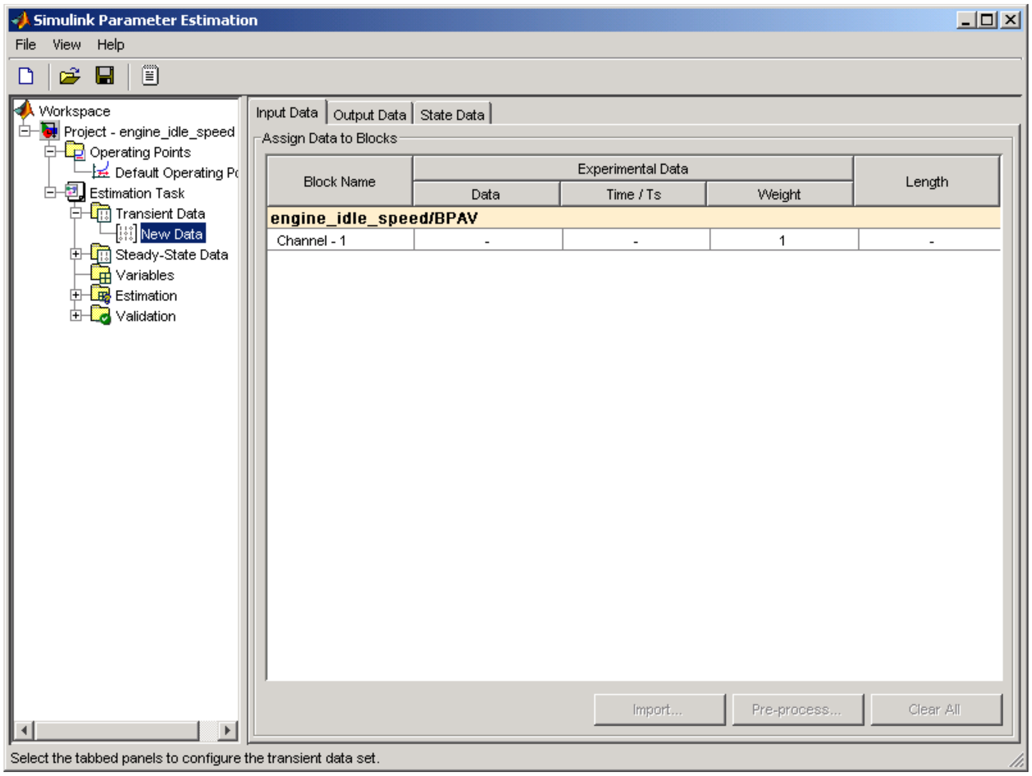
Control and Estimation Tools Manager

You can use the Simulink Control and Estimation Tools Manager to specify:

- Parameters to be estimated
- Cost functions
- Experimental data to be matched by your Simulink model
- Initial operating conditions (initial conditions) of your model

Importing Transient Data

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

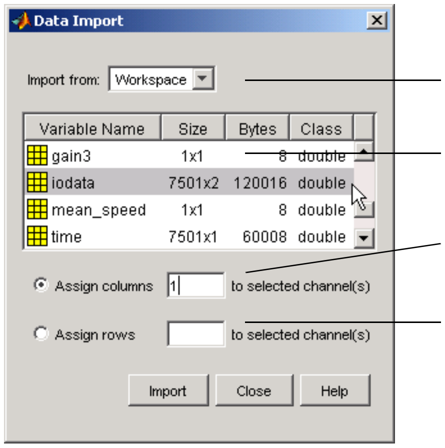


Block Name	Experimental Data			Length
	Data	Time / Ts	Weight	
engine_idle_speed/BPAV				
Channel - 1	-	-	1	-

Select the tabbed panels to configure the transient data set.

Data Importing in the Simulink Control and Estimation Tools Manager

The `iodata` array contains 2 columns, the first for input data and the second for output data. Time is stored in a separate array called `time`. Starting with the **Input Data** tab, double-click on the **Data** cell and click **Import**. This opens the **Data Import** window.



Import from: Workspace

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

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

Assign rows to selected channel(s)

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

List of available data.

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

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

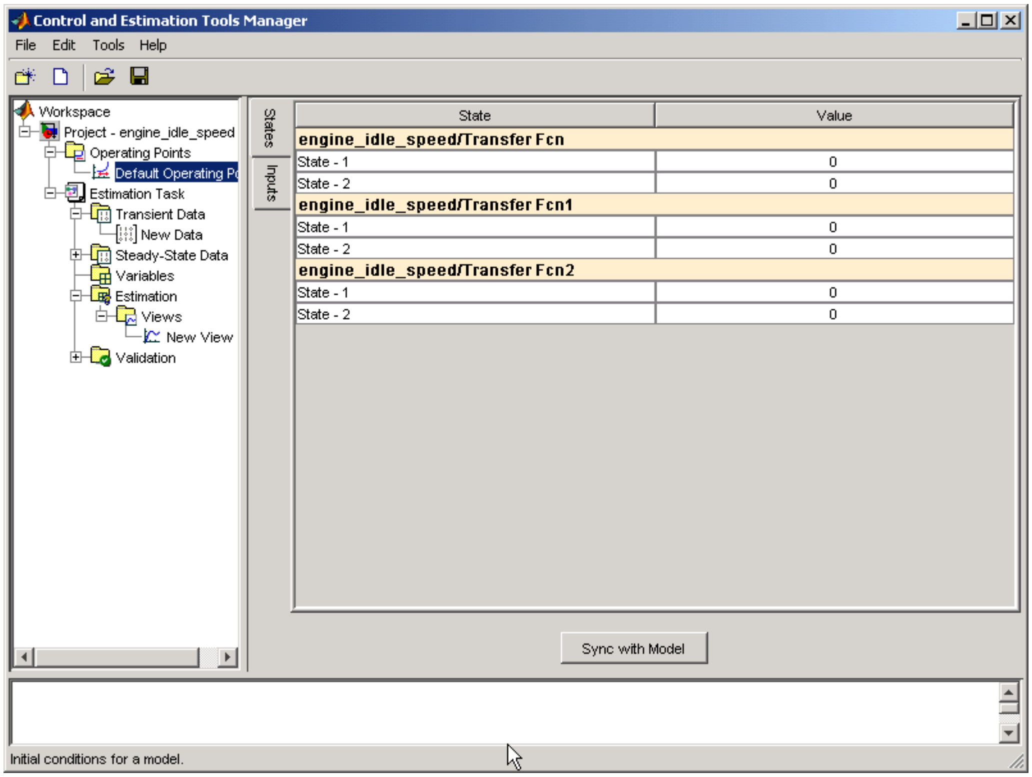
Use the Data Import Window to Select Your Data

To import input data, select `iodata` from the list of variable names, then enter `1` in the **Assign columns** field. Then click **Import**.

To import the `time` vector, follow the same procedure using the **Time/Ts cell**. To import the output data, select the **Output** Data tab and enter the data in the **Output Data** cell. Use the second column of the `iodata` array; that is, specify `2` is the **Assign columns** field.

Specifying Operating Conditions

To see the default operating points, both initial states and inputs, select **Default Operating Points** from the Operating Points folder in the left panel of the Simulink **Control and Estimation Tools Manager**. All default values are changeable, but for this example, use the default values.



State	Value
engine_idle_speed/Transfer Fcn	
State - 1	0
State - 2	0
engine_idle_speed/Transfer Fcn1	
State - 1	0
State - 2	0
engine_idle_speed/Transfer Fcn2	
State - 1	0
State - 2	0

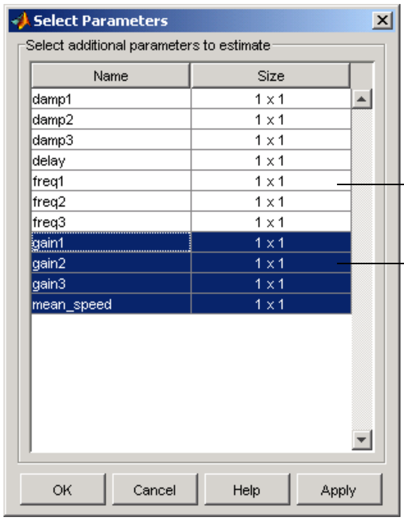
Initial conditions for a model.

Specify New Initial Conditions on the Operating Points Page

If you want to import new initial conditions from your Simulink model, click **Sync with model**.

Selecting Parameters for Estimation

To select parameters for estimation, select the Variables node. When the Global Parameters page opens, click **Add** to open the Select Parameters window.



Select additional parameters to estimate

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

By default, Select Parameters window looks at all variables in the MATLAB workspace.

List of available data.

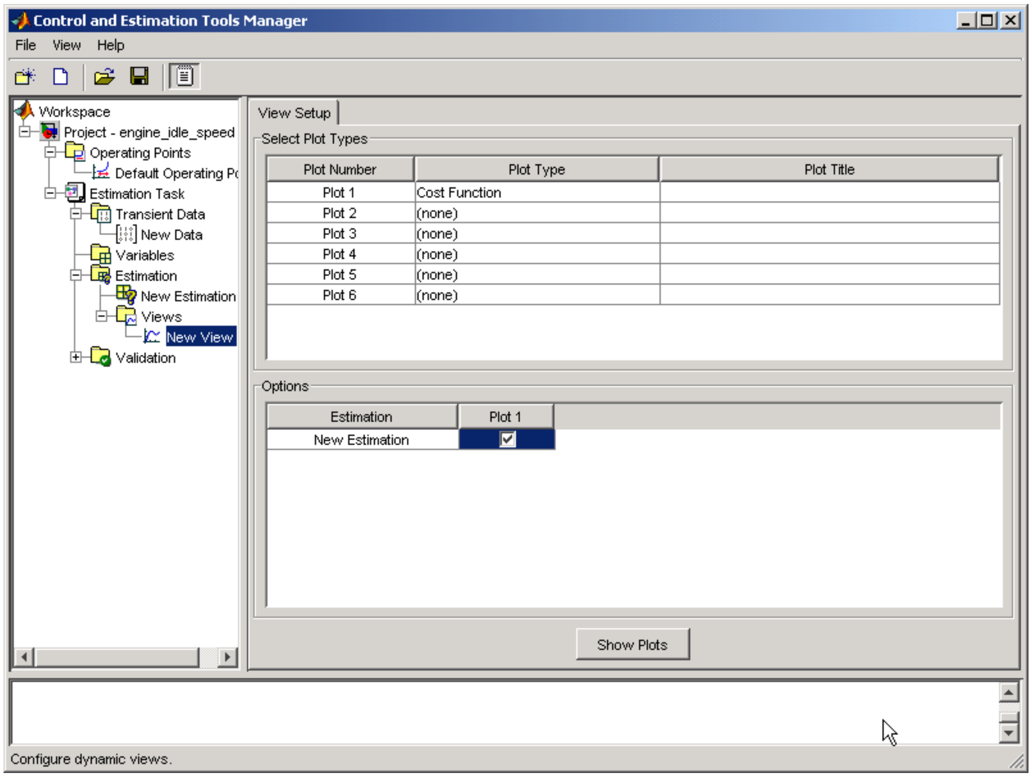
Use your mouse to select data. Hold **Shift** down to select adjacent parameters. Hold **Ctrl** down to select non-adjacent parameters.

Select Parameters Window

For this example, select the last four: `gain1`, `gain2`, `gain3`, and `mean_speed`. In general estimations you can, of course, select fewer or more variables. Often, it is more practical to estimate a small group of parameters and use the final estimated values as a starting point for further estimation of other, trickier parameters. Making these sorts of choices involves experience, intuition, and a solid understanding of your Simulink model's strengths and limitations.

Selecting Views for Plotting

You can watch the minimization process occur by right-clicking on **Views** and selecting **Add**. In the views window that opens, click **Edit** to open the **View Setup** tab.



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

Estimation	Plot 1
New Estimation	✓

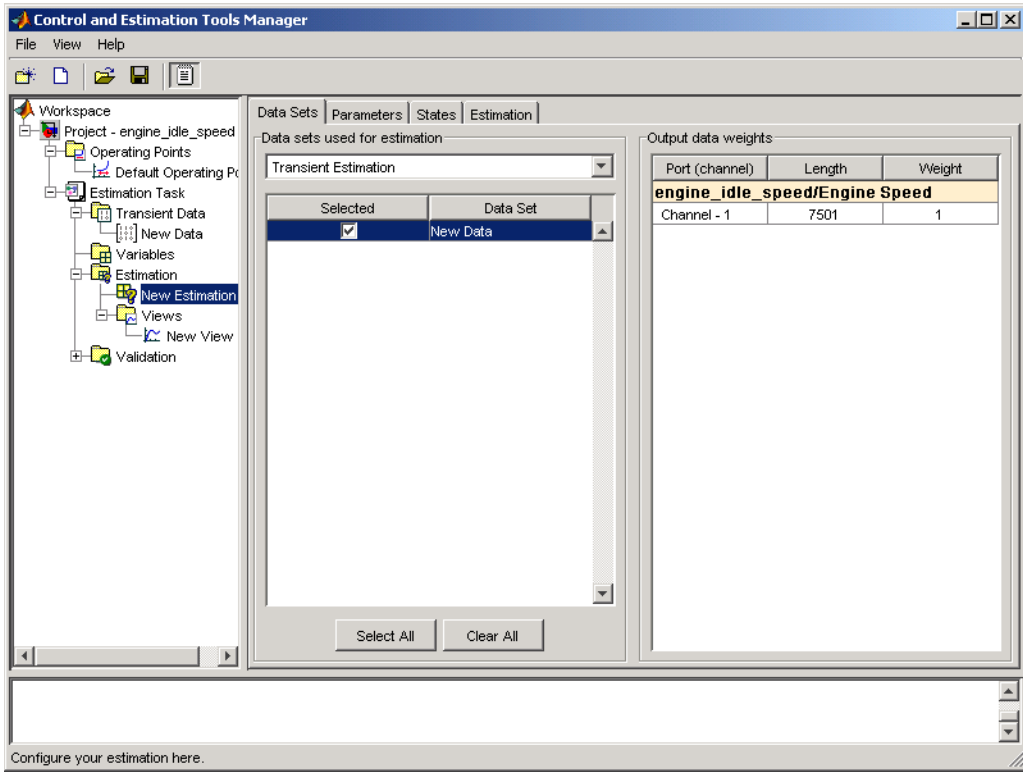
Check **Plot** in the **Options** panel, and then click **Show Plots**. This opens a plot window for the cost function. When you run your estimation, the plot updates automatically.

Running an Estimation

You are now ready to set up and run an estimation.

Adding Data Sets

Select **Estimation** from the directory tree and then right-click **Add**.



Selected	Data Set
✓	New Data

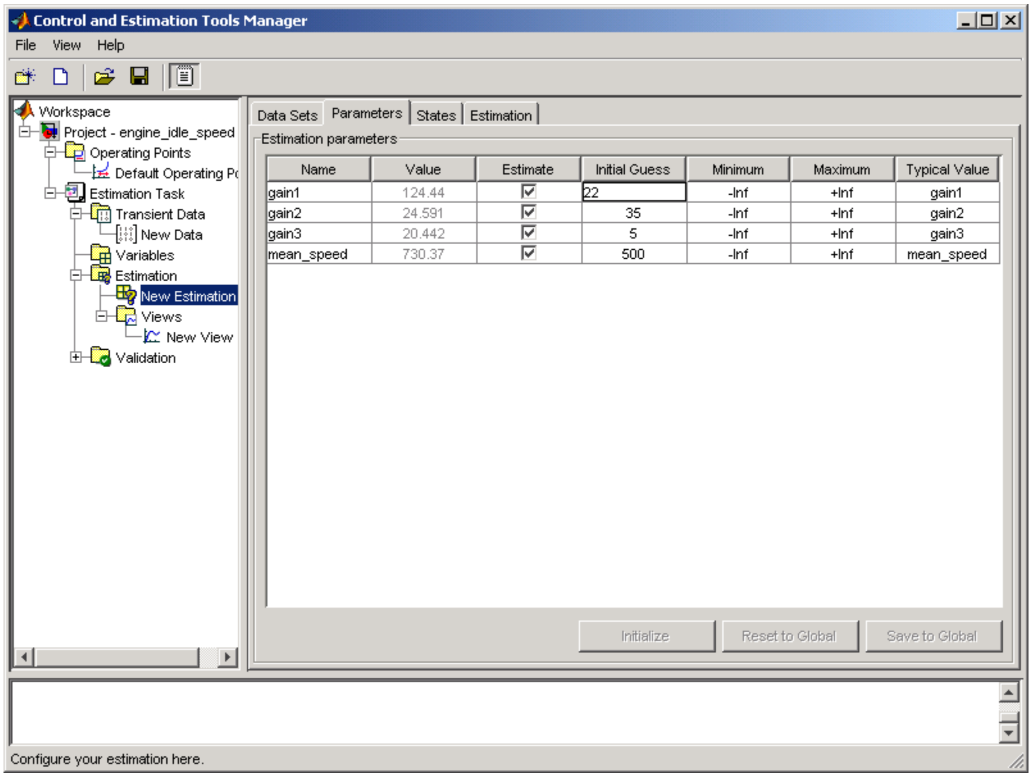
Port (channel)	Length	Weight
engine_idle_speed/Engine Speed		
Channel - 1	7501	1

Configure your estimation here.

Select **New Data** to add your engine data to the estimation.

Specifying and Setting Up Parameters

You can use the **Parameters** window to select which parameters to estimate and the range of values for the estimation.



Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
gain1	124.44	✓	22	-Inf	+Inf	gain1
gain2	24.591	✓	35	-Inf	+Inf	gain2
gain3	20.442	✓	5	-Inf	+Inf	gain3
mean_speed	730.37	✓	500	-Inf	+Inf	mean_speed

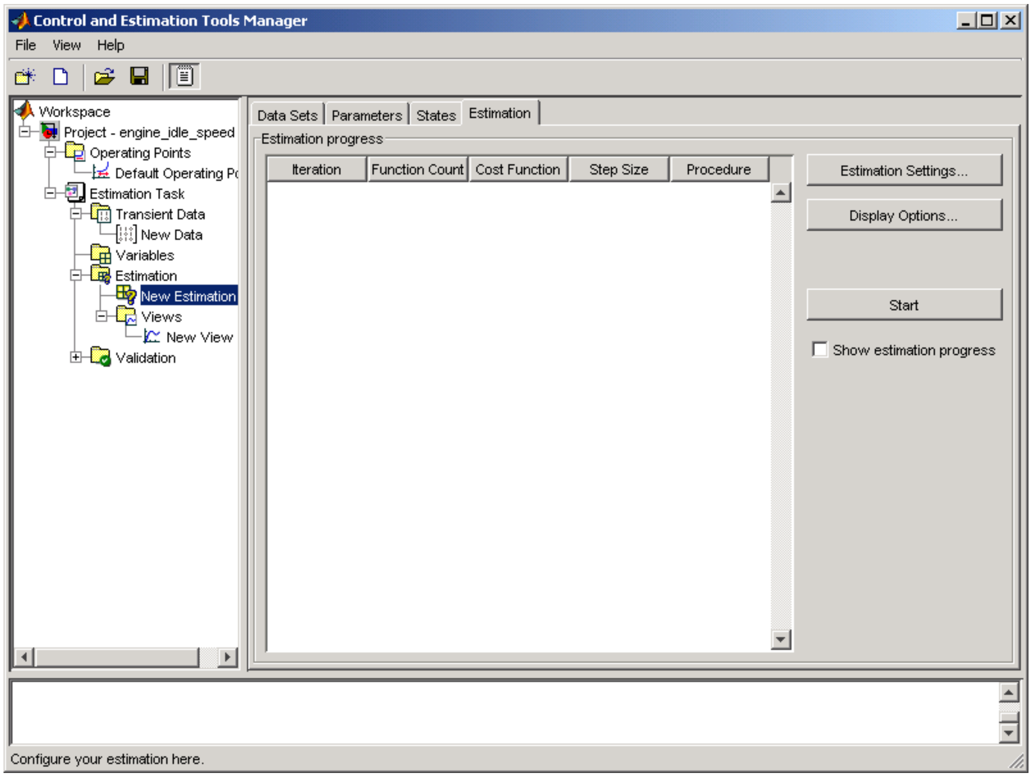
Configure your estimation here.

Select the parameters you want to estimate in the **Estimate** column. Enter initial values for your estimation parameters in the **Initial Guess** column. The default values in the **Minimum** and **Maximum** columns are `-Inf` and `Inf`, respectively, but you can select any range you want. If you have good reason to believe a parameter lies within a finite range, it is usually best not to use the default minimum and maximum values. Often there is computational advantage in specifying finite bounds if you can.

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

Opening the Estimation Window

Select the **Estimation** tab to open the Estimation window.

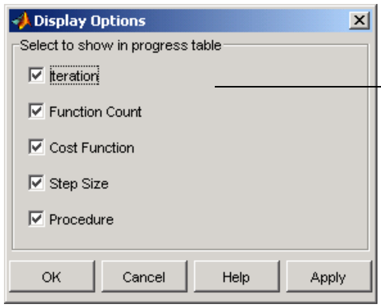


Configure your estimation here.

Before you start, you can select optimization settings to specify various algorithm features. See "Selecting Optimization Methods" on page 2-17 for more information.

Display Options

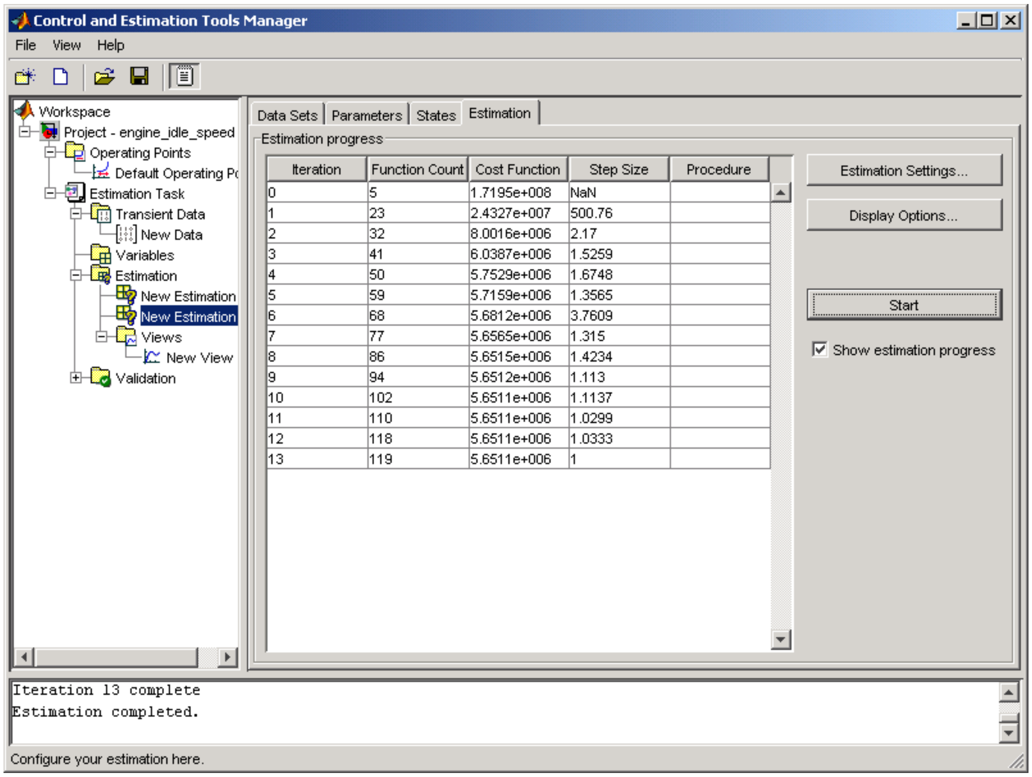
Clicking **Display Options** opens this window.



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

Running the Estimation

Click **Start** to begin the estimation. At the end of the iterations, the window should look something like this.

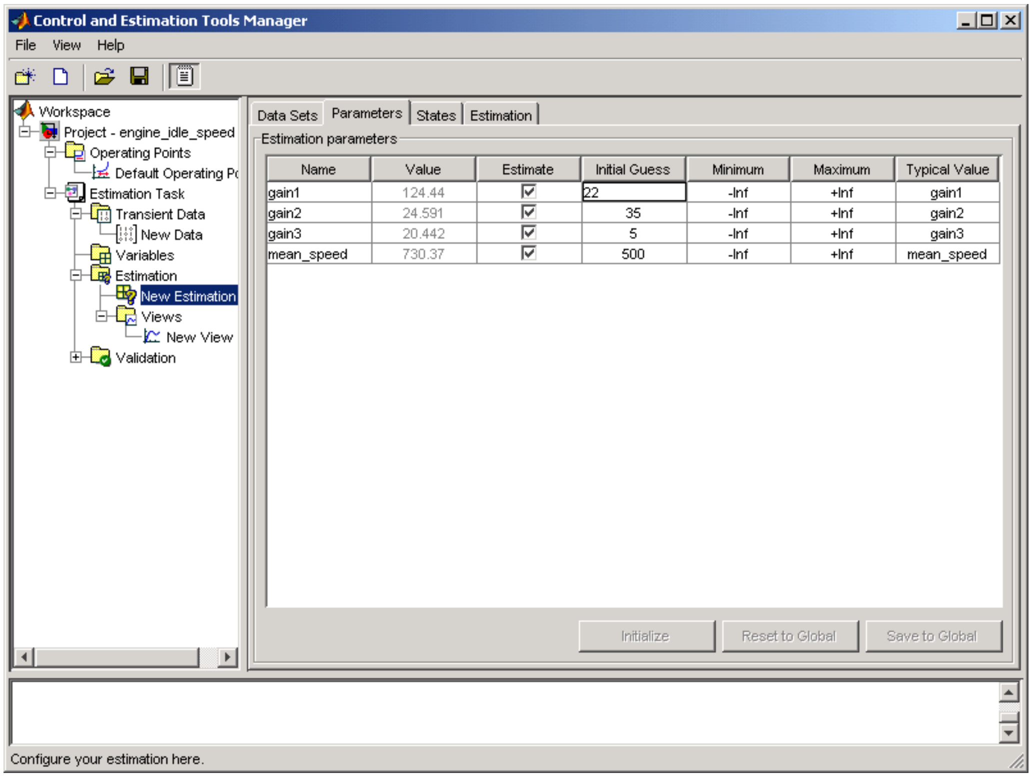


Iteration	Function Count	Cost Function	Step Size	Procedure
0	5	1.7195e+008	NaN	
1	23	2.4327e+007	500.76	
2	32	8.0016e+006	2.17	
3	41	6.0387e+006	1.5259	
4	50	5.7529e+006	1.6748	
5	59	5.7159e+006	1.3565	
6	68	5.6812e+006	3.7609	
7	77	5.6565e+006	1.315	
8	86	5.6515e+006	1.4234	
9	94	5.6512e+006	1.113	
10	102	5.6511e+006	1.1137	
11	110	5.6511e+006	1.0299	
12	118	5.6511e+006	1.0333	
13	119	5.6511e+006	1	

```
Iteration 13 complete
Estimation completed.
```

Configure your estimation here.

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

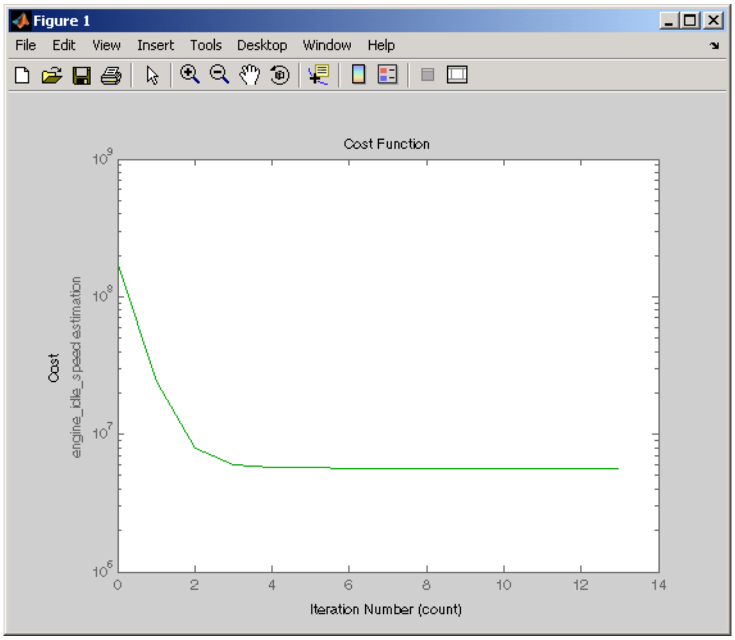


Name	Value	Estimate	Initial Guess	Minimum	Maximum	Typical Value
gain1	124.44	✓	22	-Inf	+Inf	gain1
gain2	24.591	✓	35	-Inf	+Inf	gain2
gain3	20.442	✓	5	-Inf	+Inf	gain3
mean_speed	730.37	✓	500	-Inf	+Inf	mean_speed

The final values are

- `gain1 = 124.44`
- `gain2 = 24.591`
- `gain3 = 20.442`
- `mean_speed = 730.37`

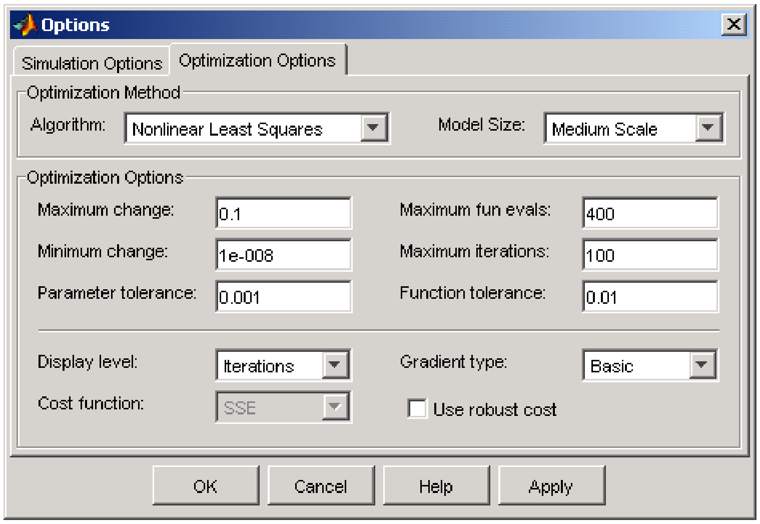
The cost function minimization is plotted below.



If the optimization went well, you should see your cost function converge on a minimum value.

Setting Options for Optimization

There are several options that can be set to tune the results of the optimization, including the optimization algorithm and the tolerances the algorithms use. To set options for optimization select **Optimization -> Optimization Options** in the **Signal Constraint** window.


Options

Simulation Options | Optimization Options

Optimization Method

Algorithm: Nonlinear Least Squares

Model Size: Medium Scale

Optimization Options

Maximum change: 0.1

Minimum change: 1e-008

Parameter tolerance: 0.001

Maximum fun evals: 400

Maximum iterations: 100

Function tolerance: 0.01

Display level: Iterations

Cost function: SSE

Gradient type: Basic

☐ Use robust cost

Selecting Optimization Methods

Both the algorithm and model size define the optimization method. For the **Algorithm** parameter, the two options are **Function Minimization** and **Simplex Search**. Function Minimization uses the Optimization Toolbox function `fmincon` to optimize the response signal subject to the constraints. Simplex Search uses the Optimization Toolbox function `fminsearch`, a direct search method, to optimize the response. Simplex Search is most useful for simple problems and is sometimes faster than `Function Minimization` for models that contain discontinuities. By default, the **Model Size** parameter is set to **Medium Scale**. When the model is very large and Function Minimization is selected as the optimization algorithm, **Model Size** can be changed to **Large Scale** to increase computation speed. See the Optimization Toolbox documentation for more information about the optimization methods.

Selecting Optimization Termination Options

There are also several options that define when the optimization will terminate:

- **Parameter tolerance**—Optimization will terminate when successive parameter values change by less than this number.
- **Constraint tolerance**—This number represents the maximum amount by which the constraints can be violated and still allow a successful convergence.
- **Function tolerance**—The optimization will terminate when successive function values are less than this value. Changing the default **Function tolerance** value is only useful when you are tracking a reference signal or using the `Simplex Search` algorithm.
- **Maximum iterations**—The maximum number of iterations allowed. The optimization will terminate when the number of iterations exceeds this number.

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

Additional options for optimization include

- **Display level**—This option specifies the form of the output that appears in the Optimization Progress window. The options are `Iter` which displays information after each iteration, `Off` 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.
- **Restarts**—In some optimizations the Hessian may become ill-conditioned and the optimization does not converge. In these cases it is sometimes useful to restart the optimization after it stops, using the end-point of the previous optimization as the starting point for the next one. To automatically restart the optimization, indicate the number of times you want to restart in this field.

- **Gradient Type**—When using `Function Minimization` as the **Algorithm**, Simulink Response Optimization 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 and it is usually only useful when a fixed-step solver is being used.

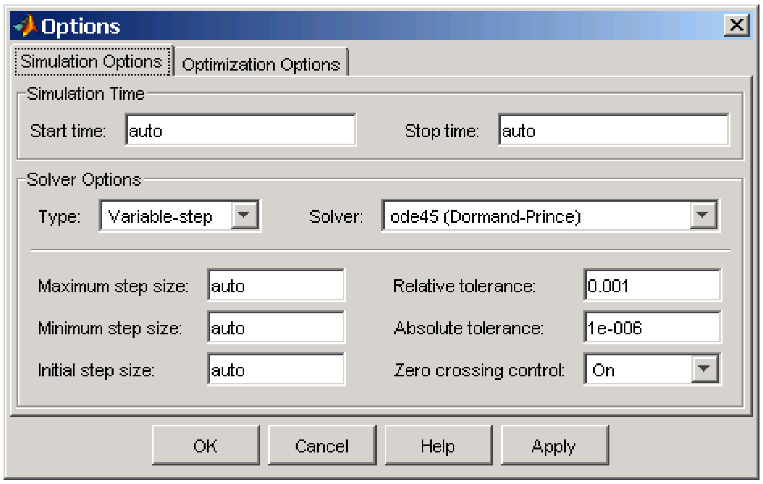
Specifying the Cost Function

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 (steady-state error), which uses a least-squares approach.
- **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.

Setting Options for the Simulation

To optimize the response signals of a model, Simulink Response Optimization runs simulations of the model. You can set options for these simulations by selecting **Optimization -> Simulation Options** in the **Signal Constraint** window.



Simulation Time

Start time: auto
Stop time: auto

Solver Options

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

Maximum step size: auto
Minimum step size: auto
Initial step size: auto
Relative tolerance: 0.001
Absolute tolerance: 1e-006
Zero crossing control: On

By default, the **Start time** and **Stop time** are automatically computed based on the start and stop times used in the model. To specify alternative start and stop times for the response optimization, enter them under **Simulation Time**.

Selecting Solvers

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

Variable-Step Solvers

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

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

See the Simulink documentation for information on these solvers.

Variable-Step Solver Options

When you select **Variable-step**, 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 non-smooth and the output depends on when a signal crosses the *x*-axis, such as absolute values.

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

Fixed-Step Solvers

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

- **discrete (no continuous states)**

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

See the Simulink documentation for information on these solvers.

When you select **Fixed-step** is selected 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.

Adaptive Look-Up Tables

Lookup Tables (p. 3-2)	A brief description of the lookup table concept
Adaptive Lookup Tables (p. 3-3)	More details on adaptive lookup tables
Implementation of Adaptive Lookup Tables (p. 3-4)	What adaptive lookup tables look like in Simulink
Example: N-D Adaptive Lookup Table (p. 3-9)	An example using an multidimensional adaptive lookup table

Lookup Tables

Lookup tables are used to store numeric data in a multi-dimensional 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 pre-determined set of data points, which are called the *breakpoints*. The set of all breakpoints corresponding to an indexing variable is called a *grid*. So, a system with M inputs is girded by M sets of breakpoints. Given the input data, the breakpoints are then used to locate the array elements, where the output data of the system are stored. For a system with N outputs, N array elements are located and the corresponding data are stored at these locations.

Once a lookup table is created using the input and output measurements as described above, the corresponding multi-dimensional array of values can be used in applications without the need of re-measuring 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 pre-determined array locations.

Adaptive Lookup Tables

The generation of lookup tables as described above establishes a permanent and static mapping of input-output behavior of a physical system. Statically defined lookup tables cannot accommodate the *time-varying* behavior (characteristics) of a physical plant. On the other hand, it is well known that the behavior of actual physical systems often vary with time due to wear, environmental conditions, and manufacturing tolerances. Under such variations, the static mapping of input-output behavior of a plant described by the lookup table may no longer provide a valid representation of the plant characteristics.

Adaptive lookup tables, on the other hand, incorporate the time-varying behavior of physical plants into the lookup table generation and maintenance process while providing all of the functionality of a regular lookup table.

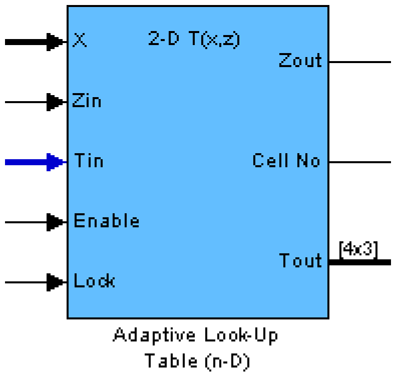
The adaptive lookup table receives the input and output measurements of a plant's behavior, which are then used to dynamically create and update the content of the underlying lookup table. In addition to requiring the input data to create the lookup table, the adaptive lookup table also uses the output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

The input measurements are used to locate the array elements by comparing these input values with the breakpoints defined for each indexing variable. Next, the output measurements are used to recalculate the numeric value stored at these array locations. However, unlike a regular table, which only stores the array data before the actual use of the lookup table, the adaptive table continuously improves the content of the lookup table. This continuous improvement of the table data is referred to as the adaptation or learning process.

The adaptation process involves statistical and signal processing algorithms to recapture the input-output behavior of the plant. The adaptive lookup table always tries to provide a valid representation of the plant dynamics even though the plant behavior may be time varying. The underlying signal processing algorithms are also robust against reasonable measurement noise and they provide appropriate filtering of noisy output measurements.

Implementation of Adaptive Lookup Tables

The adaptive lookup tables are implemented as Simulink blocks. They create multi-dimensional lookup tables from measured or simulated data. The inputs and outputs of a 2-D Adaptive Lookup Table block are shown below.



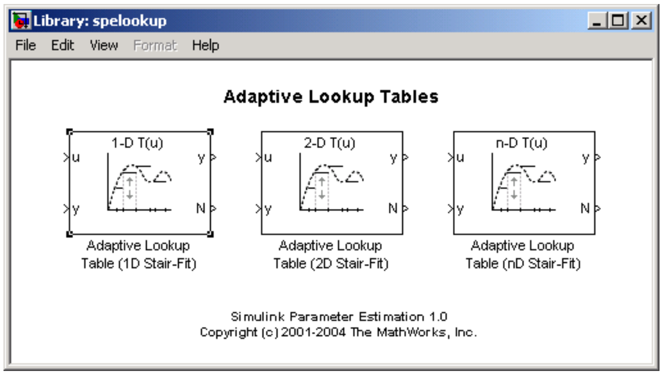
Adaptive Lookup Table Block Showing Inputs and Outputs]

The following are descriptions of the input and output parameters:

- The *inputs* `X` and `Zin` 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, `X = [rpm, pressure]` and `Zin = [efficiency]`.
- The *initial table* data may be entered either as a dialog parameter (by double-clicking on the block) or as an input port (i.e., the input port `Tin` in the figure). You can start/stop/reset the adaptation through the `Enable` input port.
- The *outputs* of the adaptive lookup table block include the value of the currently adapted table cell (`Zout`), the number (`Cell No`) of that cell (which may be specified through the block dialog), and if required, the whole adapted table data (`Tout`).

Adaptive Lookup Table Library

There are three adaptive lookup tables available in Simulink Parameter Estimation.

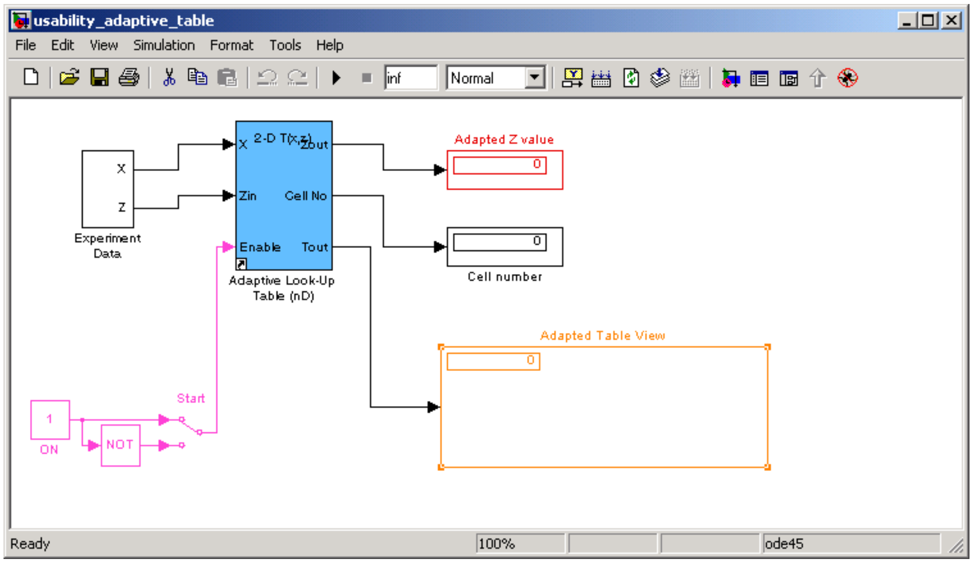

Library: spelookup
File Edit View Format Help
Adaptive Lookup Tables
1-D T(u)
u
y
y
N
Adaptive Lookup Table (1D Stair-Fit)
2-D T(u)
Adaptive Lookup Table (2D Stair-Fit)
n-D T(u)
Adaptive Lookup Table (nD Stair-Fit)
Simulink Parameter Estimation 1.0
Copyright (c) 2001-2004 The MathWorks, Inc.

The three blocks are:

- "Adaptive Look-Up Table (1-D)" on page 5-2 — One dimensional adaptive lookup
- "Adaptive Look-Up Table (2D)" on page 5-5 — Two-dimensional adaptive lookup
- "Adaptive Look-Up Table (n-D)" on page 5-8 — Multidimensional adaptive lookup (use this for dimension 3 or higher)

Using Adaptive Lookup Tables in Simulink Models

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



Simulink Diagram Using an Adaptive Lookup Table

In this figure, the Experimental 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.

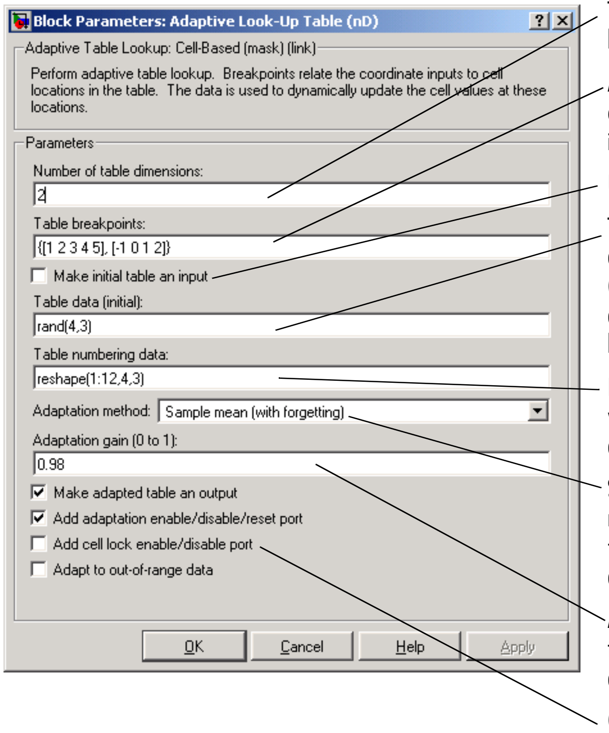
Real-Time Lookup Tables

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

The adaptive lookup table block may also be used in real-time environment, where some time-varying properties of a system need to be captured. This can be done by generating C code using the Real-Time Workshop, which can then be run in xPC or dSpace environment. Since the adaptation may be started/stopped/reset if desired, some logic may be used to adapt the table data only when it is desired. The cell number output, and the **Enable** and **Lock** inputs facilitate this process. The Enable input is used to start/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 number output provide the means for updating only the desired table cells during a simulation run.

Setting Adaptive Lookup Table Parameters

Adaptive lookup tables are highly configurable, as shown below.



Block Parameters: Adaptive Look-Up Table (nD)

Adaptive Table Lookup: Cell-Based (mask) (link)

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

Parameters

Number of table dimensions: 2

Table breakpoints: {[1 2 3 4 5], [-1 0 1 2]}

☐ Make initial table an input

Table data (initial): rand(4,3)

Table numbering data: reshape(1:12,4,3)

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.98

☑ Make adapted table an output

☑ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

☐ Adapt to out-of-range data

The number of dimensions for the adaptive look-up table.

A set of one-dimensional vectors that contains possible block input values for the input variables.

Use this port to input table data.

The initial table output values. This (n-D) array must be of size (n-1)-by-(n-1)... -by-(n-1), (D times) where D is the number of dimensions and n is the number of input breakpoints.

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

Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data.

A number between 0 and 1 that regulates the weight given to new data during the adaptation.

Checkboxes for customizing the I/O channels of the block and allowing adaptation to out-of-range data.

n-D Adaptive Lookup Table Dialog Box

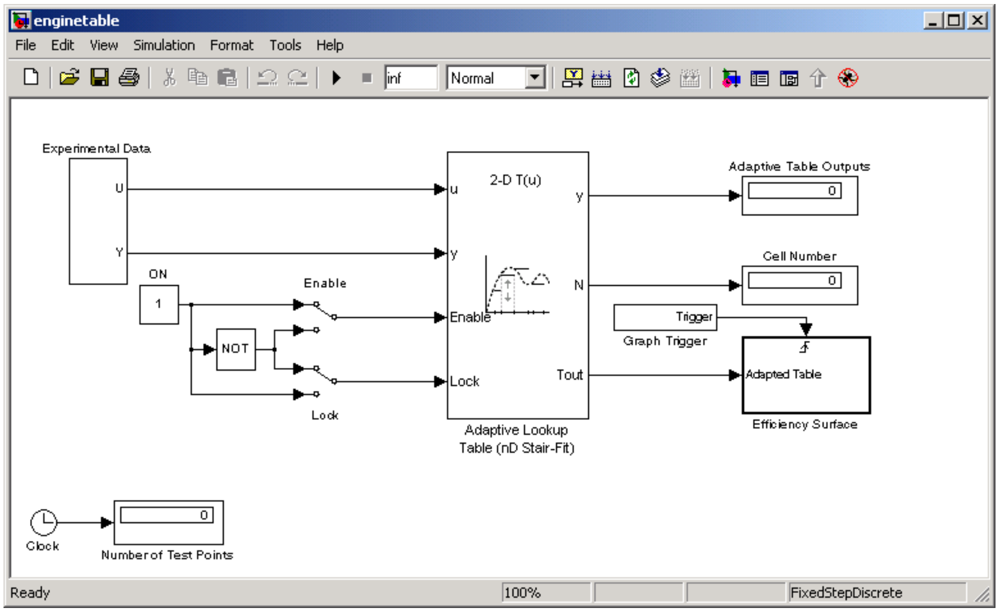
For details on how to set these parameters, see the individual reference pages.

Example: N-D Adaptive Lookup Table

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

```
enginetable
```

at the MATLAB prompt to open this model.



This model has several key features:

- Input —The adaptive lookup table input is the experimental data. It is also possible to make the original table itself an input.
- An enable feature—You can turn the adaptation on and off during the estimation to see how the basic features work.
- A lock feature—You can lock the table so that only one cell is adapting. This is useful if you have one section in your data that is highly erratic or otherwise difficult for the algorithm to handle.
- Output—Adaptive lookup breakpoints are the output data

Running the Example

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

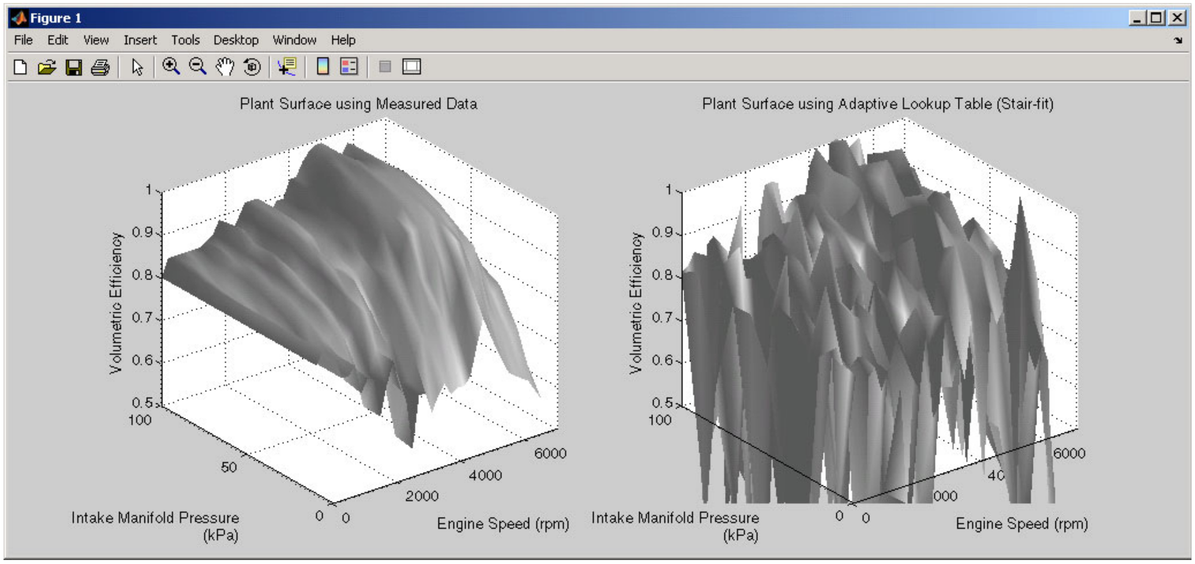


Figure 1
File Edit View Insert Tools Desktop Window Help
Plant Surface using Measured Data
Volumetric Efficiency
1
0.9
0.8
0.7
0.6
0.5
100
50
0
Intake Manifold Pressure (kPa)
0
2000
4000
6000
Engine Speed (rpm)
Plant Surface using Adaptive Lookup Table (Stair-fit)
Volumetric Efficiency
1
0.9
0.8
0.7
0.6
0.5
100
0
Intake Manifold Pressure (kPa)
0
000
40
6000
Engine Speed (rpm)

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

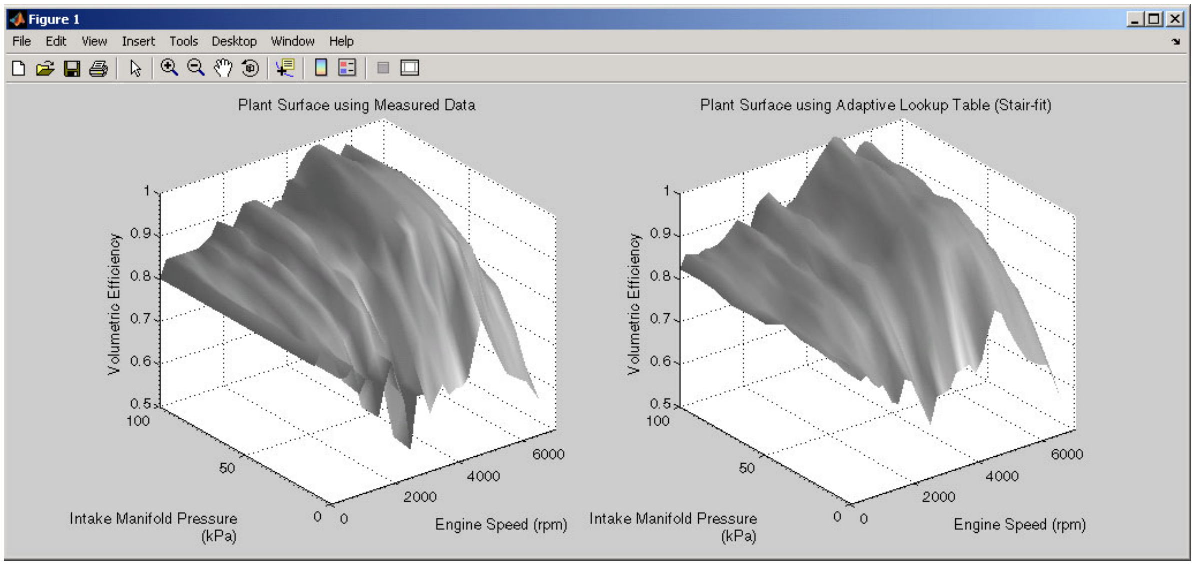


Figure 1
File Edit View Insert Tools Desktop Window Help
Plant Surface using Measured Data
Volumetric Efficiency
Intake Manifold Pressure (kPa)
Engine Speed (rpm)
Plant Surface using Adaptive Lookup Table (Stair-fit)
Volumetric Efficiency
Intake Manifold Pressure (kPa)
Engine Speed (rpm)

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

Estimating From the Command Line

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

Introduction (p. 4-2)	A brief discussion of the estimation problem in an object-oriented context
Example: Estimating Parameters and Initial Conditions of the F14 Model (p. 4-4)	How to create and simulate an estimation project from the command line
Creating and Customizing Estimation Projects (p. 4-12)	Using properties and methods to specify features of the estimation project
“Creating a Transient Data Object” on page 4-13	How to instantiate and use transient data objects, which contain input and output data.
“Creating Parameter Objects” on page 4-19	How to instantiate and use parameter objects, which maintain data about parameters you want to estimate.
“Creating State Data Objects” on page 4-23	How to instantiate and use state data objects, which contain information about states in your Simulink model.
“Creating Transient Experiment Objects” on page 4-26	How to instantiate and use transient experiment objects.

Introduction

In addition to the Parameter Estimation editor, Simulink Parameter Estimation provides a collection of functions for performing parameter and state estimation. These functions perform the same tasks as the editor, but have the advantages of command-line execution. When you perform a state or parameter estimation using the Simulink Parameter Estimation GUI, Simulink Parameter Estimation creates MATLAB objects for all the states and parameters of your model. If you have a large number of states or parameters, this can use up large chunks of memory and cause computational delays. Using 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 out large numbers of models.

Note Simulink Parameter Estimation uses MATLAB objects to perform estimation tasks. This chapter discusses what you need to know about object-oriented programming for using Simulink Parameter Estimation, but see "MATLAB Classes and Objects" for a detailed description of the rules of object-oriented programming in MATLAB. Also, see *** for a discussion of dot notation if you are unfamiliar with this feature.

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

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

The following sections discuss these topics:

- "Example: Estimating Parameters and Initial Conditions of the F14 Model" — How to perform the five steps using command-line functions

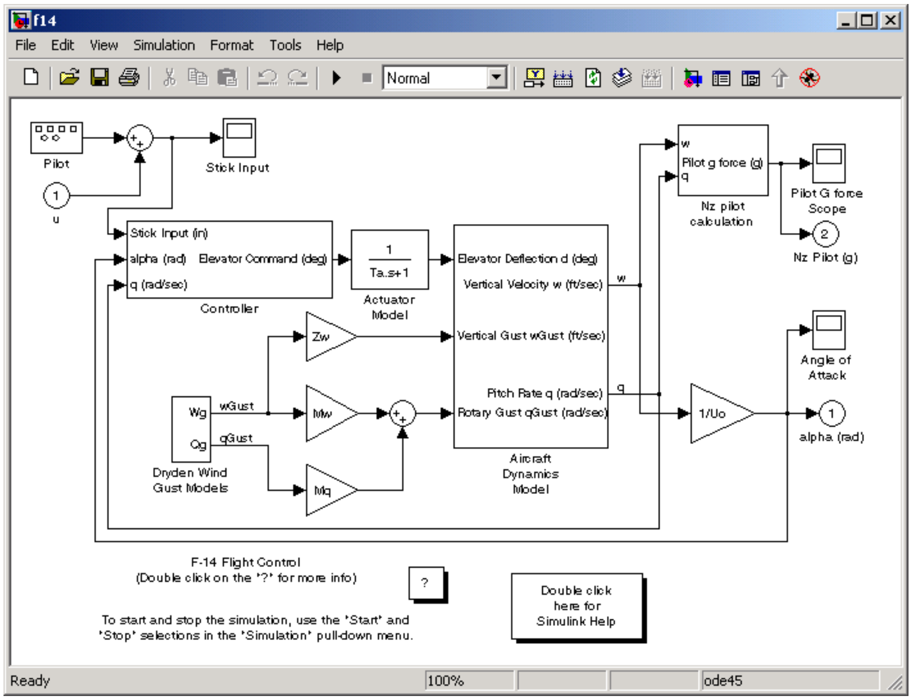
- "Creating and Customizing Estimation Projects" — How to use methods and properties to customize your estimation project's features

Example: Estimating Parameters and Initial Conditions of the F14 Model

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

```
f14
```

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



F14 Fighter Model

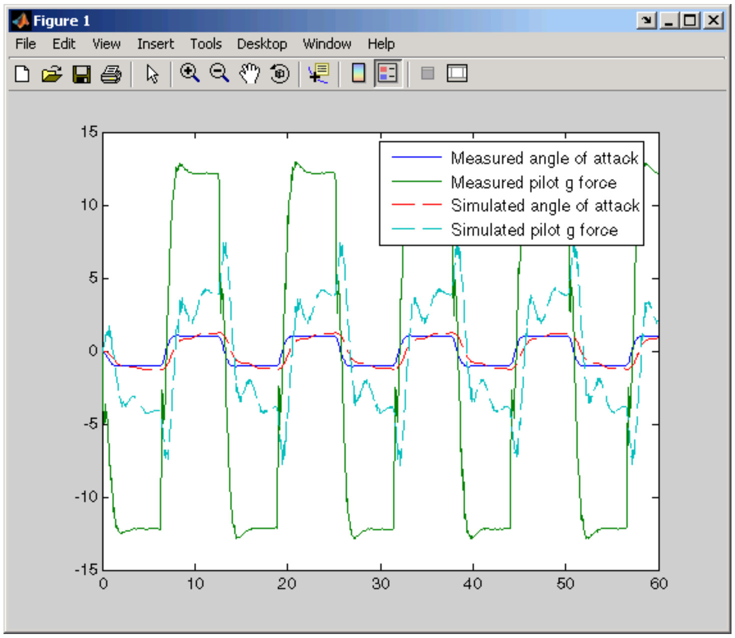
This example outlines the basics of constructing an estimation project using object oriented code. Only what you need to run the example is presented; see “Creating and Customizing Estimation Projects” on page 4-12 for details on all the properties and methods associated with parameter estimation.

Baseline Simulation

Prior to running an estimation, you will 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`, Then use the code below to run the Simulink F14 model. Note that this is standard Simulink code and does not involve Simulink Parameter Estimation in any way. See `sim` for information about running Simulink models from the MATLAB command line.

```
%% Open the model and load experimental data.
open_system('f14')
load f14_estim_data % 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');
```

This figure appears.



Baseline Comparison of Measured and Simulated F14 I/O Data

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

Creating a Transient Experiment Object

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

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

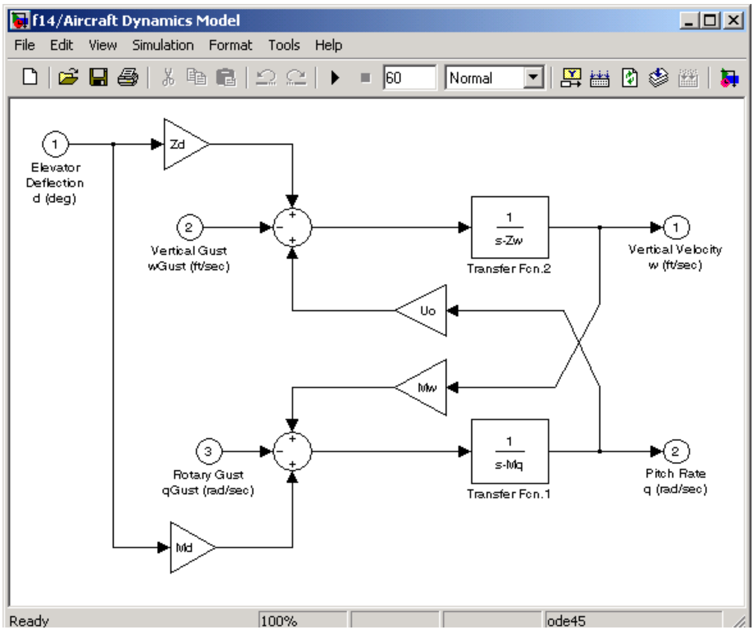
```
h = ParameterEstimator.TransientExperiment('f14');
```

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
```

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 the `Ta`, the actuator time constant, and `Zd` and `Md`, the vertical velocity and pitch rate gains, respectively. The Zd and Kf gains are located in the f14: Aircraft Dynamics subsystem.


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

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 estimated 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;
```

Assigning Experimental Data to Inputs and Outputs of the Model

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

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

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

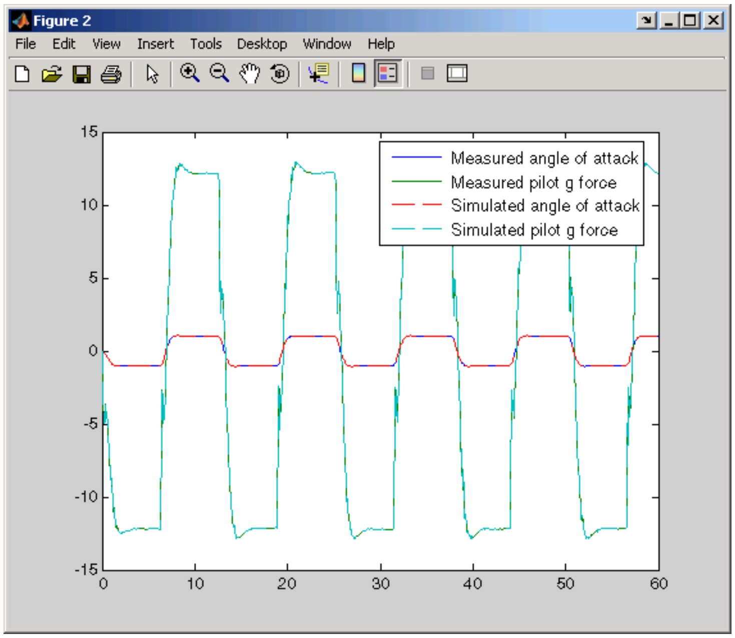
Creating an Estimation Object and Running the Estimation

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

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

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

%% Run the estimation
estimate(hEst);
```

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

This figure shows the results of the estimation.



The measured and simulated outputs now appear to be a close match. Next, look at the estimated values to see how they compare with the default values of the F14 model.

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

MATLAB responds with

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

       Parameter value : 0.05
         Initial guess : 0.5

             Estimated : true

          Referenced by:

(2) Parameter data for 'Md':

       Parameter value : -6.884
         Initial guess : -1

             Estimated : true

          Referenced by:

(3) Parameter data for 'Zd':

       Parameter value : -63.99
         Initial guess : -80

             Estimated : true

          Referenced by:
```

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

```
clear all
f14
whos
```

Creating and Customizing Estimation Projects

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

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 a Transient Data Object

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

```
h = ParameterEstimator.TransientData( block ); % I/O port block
h = ParameterEstimator.TransientData( block , portnumber); %
Internal block
h = ParameterEstimator.TransientData( block , data, time);
h = ParameterEstimator.TransientData( block , data, Ts);
h = ParameterEstimator.TransientData( block , portnumber, data,
time);
h = ParameterEstimator.TransientData( block , portnumber, data,
Ts);
h = ParameterEstimator.TransientData( block , ...,  Property ,
Value, ...)O.
```

Properties of Transient Data Objects

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

Transient Data Object Properties

Property	Description
`Block`	Name of the Simulink block with which the Data is associated. Must be a string.
`PortType`	The type of the 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 one.

Transient Data Object Properties (Continued)

Property	Description
`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 $N_s\ m$, where N_s is the number of data samples, and m is the number of channels in the signal. • Multi-dimensional data (matrix and higher dimensions). The Data is of the form $m_1 \ldots\ m_n$ N_s, where N_s is the number of data samples, and m_i is the number of channels in the ith dimension of the signal. • For missing or unspecified data NaNs are used.

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

Transient Data Object Properties (Continued)

Property	Description

`Time`	The time data in column vector format. The length of `Time` must be consistent with the number of samples in `Data`.
	• For a non-uniformly spaced `Time` vector, its length should match the length of `Data`.
	• Otherwise, `Time` is automatically adjusted based on the length of `Data`.
	Modifying `Ts` will reset `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` will set
	`Ts = NaN`
	`Tstart = Time(1)`
	If the time vector is uniformly spaced, a sample time `Ts` is calculated.
	• Modifying `Tstart` translates Time forward or backward.
	• Modifying `Ts` sets `Time = []` internally and generates it when required by the simulation.

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

Modifying Transient Data Object Properties

Once a transient data object is created, you can modify its properties using this syntax.

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

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

Example: Assigning Input Port Data

To assign data to an Input port with 23 port dimensions, use

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

MATLAB responds with

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

Using Class Methods

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

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

  ```
  in2 = select(in1,  Sample , 10:100); % 91 samples

  in3 = select(in1,  Range , [1 4]); % Samples for 1<t<4
  % ... or an alternative

  in3 = select(in1,  Sample , find(in1.Time > 1 & in1.Time < 4));
  ```

  To extract data from a subset of available channels, use

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

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

- `update`—Updates the object after the Simulink model has been modi.ed in some way. If the `Dimensions` property value changes then the other properties are reset to their default values.

Helper Functions: Seven Ways to Represent Sampling Instants

Time is represented in `@TransientData` objects ins seven different ways. An efficient storage method is required for uniformly spaced data (e.g., using sampling and start times only) as well as the ability to store a non-uniform time vector.

Only `Time`, `Ts`, and `Tstart` are writable time properties. However, there are other ways of specifying time data. Given a data array of length `n` and a combination of at most two of the following parameters

- Sampling time, *Ts*
- Start time, *ti*
- Final time, *tf*
- Time vector, *(t0, t1, . . . , tn-1)*

you can calculate the corresponding `Time` vector of length *n*. The combinations are

1 Given Ts $t = \{kTs, k = 0, . . . , n . 1\}$

2 Given Ts, ti $t = \{ti + kTs, k = 0, . . . , n . 1\}$

3 Given Ts, tf $t = \{tf . (n . 1 . k)Ts, k = 0, . . . , n . 1\}$

4 Given t_i, t_f $t = \{t_i + kT_s, k = 0, . . . , n . 1, T_s = t_f\text{-}t_i n\text{-}1\}$

5 Given t_f $t = \{kT_s, k = 0, . . . , n . 1, T_s = t_f n\text{-}1\}$

6 Given $(t_0, t_1, . . . , t_{n-1})$ $t = \{t_k, k = 0, . . . , n . 1\}$

7 Given t_i, $(t_0 = 0, t_1, . . . , t_{n-1})$ $t = \{t_i + t_k, k = 0, . . . , n . 1\}$.

Method for Logging Internal Block Signals.

When a data object represents an internal block signal, signal logging is used to access the output signal of that block. The following piece of code, which is mainly implemented in estimation methods, illustrates this process.

```
% Create data object
sig1 = ParameterEstimator.TransientData(gcb, 2);
% ...
% Configure port for signal logging before simulation
% Output port handle
h = get_param( sig1.Block, 'Object' );
hPort = handle( h.PortHandles.Outport(sig1.PortNumber) );
% Signal name
Name = [ sig1.Block '/' int2str(sig1.PortNumber) ];
Name = regexprep(Name, '\n|\n\r', ' ');
Name = regexprep(Name, ' ', '_');
Name = regexprep(Name, '/', '_')
% Configure signal logging properties
hPort.Name = 'SPE_signal'; % Doesn t need to be unique
hPort.TestPoint = 'on';
hPort.DataLogging = 'on';
hPort.DataLoggingName = Name; % Should be unique
hPort.DataLoggingNameMode = 'Custom';
% ...
% Extract signal data after simulation
% Signal logging name
LoggingName = get_param(gcs, 'SignalLoggingName');
hLog = evalin('base', LoggingName);
sig1_data = hLog.extract(Name);
```

Creating Parameter Objects

The `@Parameter` object refers to the parameters of the Simulink model marked for estimation. Some of the Simulink model parameters are to be estimated and storage is required for the initial values, current values, ranges, etc. One `@Parameter` object corresponds to each parameter in the Simulink model to be potentially estimated. These objects represent estimation parameters of any type such as scalars, vectors, and multi-dimensional 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);
h = ParameterEstimator.Parameter('Name', ..., 'Property', Value,
...);
```

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

Properties of Parameter Objects

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

Table 4-1: Parameter Object Properties

Property	Description
`Name`	Parameter name. The parameter can be a multi-dimensional array of any size.
`Dimensions`	Dimensions of the value of the parameter. This is the de.ning 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.

Table 4-1: Parameter Object Properties (Continued)

Property	Description
`Estimated`	A boolean array of the same size as that of `Value`. Depending on the value of the elements of the `Estimated` property, the behavior of the corresponding elements of `Value` are 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.
`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` are 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` will not be used in any way.

Table 4-1: Parameter Object Properties (Continued)

Property	Description
`Minimum`, `Maximum`	Parameter range
`TypicalValue`	The typical values of the parameters. This property is used in estimations for scaling purposes. The default value is one

Example: F14 Model

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

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

Example: Gain Matrix

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

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

Modifying Properties

Once a parameter object is created, its properties can be modi.ed using the following syntax:

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

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

Using Class Methods

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

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

Creating State Data Objects

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

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

This is the syntax for creating this object.

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

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

Properties of the State Data Object

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

Table 4-2: Properties of the State Data Object

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

Table 4-2: Properties of the State Data Object (Continued)

Property	Description
`Data`	Column vector to store the initial value of the state for the block speci.ed 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 a different initial values.
`Ts`	Sampling time of discrete blocks. Set to zero for continuos blocks. This property is read-only and is currently used for information only.
`BlockInfo`	Structure to hold data about SimMechanics or SimPower Systems 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

Once a state data object is created, its properties can be modi.ed using the following syntax.

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

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

Using Class Methods

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

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

Creating Transient Experiment Objects

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

The syntax to create a transient experiment object is

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

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

h = ...
ParameterEstimator.TransientExperiment('model', ...
{'in1', ...}, {'out1', ...}, {'ic1', ...});


h = ParameterEstimator.TransientExperiment('model', ...,
'Property', Value, ...);
```

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

Properties of Transient Experiment Objects

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

Table 4-3: Properties of Transient Experiment Objects

Property	Description
`Model`	Simulink model with which this experiment is associated
`InputData`, `OutputData`	Transient data objects associated with appropriate i/0 blocks in the Model. Blocks with unassigned objects or objects with no data will not be 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 f14 Experiment Using Block Names

To create an empty transient experiment where data is available only for the first output and the Actuator Model block, use

```
exp1 = ParameterEstimator.Experiment('f14', {}, ...
{'f14/alpha (rad)'},
{'f14/Actuator Model'})
Experimental (Transient) data set for the model 'f14':
Outputs
(1) f14/alpha (rad)
Inputs(none)
Initial States
(1) f14/Actuator Model
```

Example: Creating Van der Pohl Experiment From User Objects

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

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

Modifying Properties

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

Using Class Methods

The description of one important method is given below:

- `update`: Updates the object after the Simulink model has been modi.ed in some way. The object listed in `InputData`, `OutputData`, `InitialStates` properties are updated in turn.

Block Reference

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

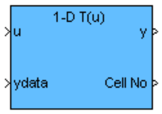
- `Adaptive Look-Up Table (1-D)` on page 5-2
- `Adaptive Look-Up Table (2D)` on page 5-5
- `Adaptive Look-Up Table (n-D)` on page 5-8

Adaptive Look-Up Table (1-D)

Purpose

Perform a one-dimensional adaptive table lookup

Description



The Adaptive Look-Up Table (1-D) 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.

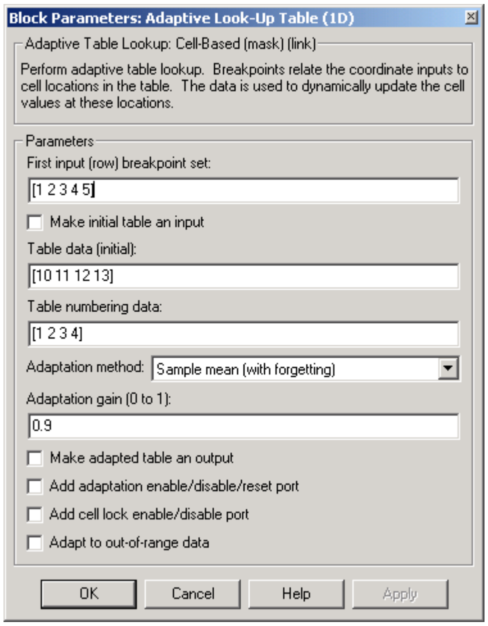
Each indexing parameter U may take a value within a set of adapting data points, which are called *breakpoints*. Two breakpoints in each dimension define a *cell*. The set of all breakpoints in one of the dimensions defines a *grid*. In the one-dimensional case, each cell has two breakpoints, and the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

You can use the Adaptive Look-Up Table (1-D) to model time-varying systems.

Data Type Support

Doubles only

Dialog Box



Block Parameters: Adaptive Look-Up Table (1D)

Adaptive Table Lookup: Cell-Based (mask) (link)

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

Parameters

First input (row) breakpoint set: [1 2 3 4 5]

☐ Make initial table an input

Table data (initial): [10 11 12 13]

Table numbering data: [1 2 3 4]

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

☐ Adapt to out-of-range data

First input (row) breakpoint set

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

Make initial table an input

Selecting this box forces the Adaptive Look-Up Table (1-D) block to ignore the **Table data (initial)** parameter. Instead, a new port appears with **Tin** next to it. Use this port to input table data.

Table data (initial)

The initial table output values. This vector must be of size N-1, where N is the number of breakpoints.

Table numbering data

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

Adaptation method

Choose **Sample mean** or **Sample mean with forgetting**. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter.

Adaptation gain (0 to 1)

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

Make adapted table an output

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

Add adaptation enable/disable/reset port

Add an input port that enables, disables, or resets the adaptive look-up 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.

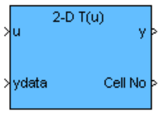
Adapt to out-of-range data

Extrapolate beyond the extreme breakpoints.

Purpose

Perform two-dimensional adaptive table lookup

Description

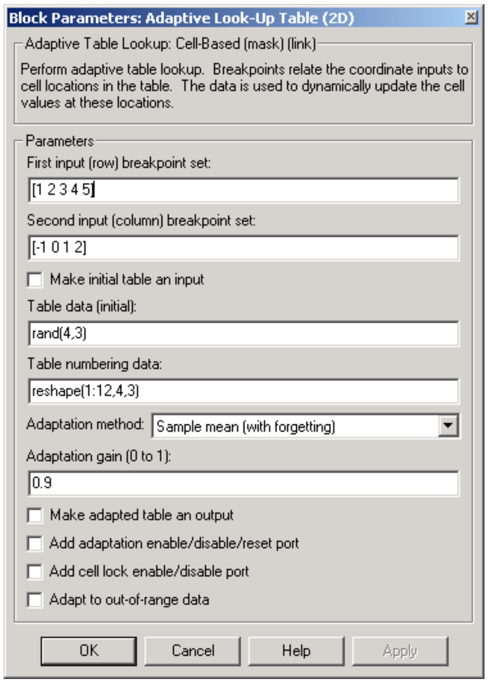


The Adaptive Look-Up Table (2-D) block creates a two-dimensional adaptive lookup table by dynamically updating the underlying look-up table. The block uses the outputs (*ydata*) of your system to do the adaptations.

Each indexing parameter U may take a value within a set of adapting data points, which are called *breakpoints.* wo 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 Look-Up Table (2-D) to model time-varying systems.

Dialog Box



Block Parameters: Adaptive Look-Up Table (2D)

Adaptive Table Lookup: Cell-Based (mask) (link)

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

Parameters

First input (row) breakpoint set: [1 2 3 4 5]

Second input (column) breakpoint set: [-1 0 1 2]

☐ Make initial table an input

Table data (initial): rand(4,3)

Table numbering data: reshape(1:12,4,3)

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

☐ Adapt to out-of-range data

First input (row) breakpoint set

The vector of values containing possible block input values for the first input variable. The first input vector must be monotonically increasing.

Second input (column) breakpoint set

The vector of values containing possible block input values for the second input variable. The second input vector must be monotonically increasing.

Make initial table an input

Selecting this box forces the Adaptive Look-Up Table (2-D) 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.

Adaptation gain (0 to 1)

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

Make adapted table an output

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

Add adaptation enable/disable/reset port

Add an input port that enables, disables, or resets the adaptive look-up table.

Add cell lock enable/disable port

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

Adapt to out-of-range data

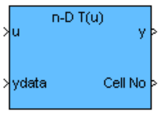
Extrapolate beyond the extreme breakpoints.

Adaptive Look-Up Table (n-D)

Purpose

Create an adaptive lookup table of arbitrary dimension

Description

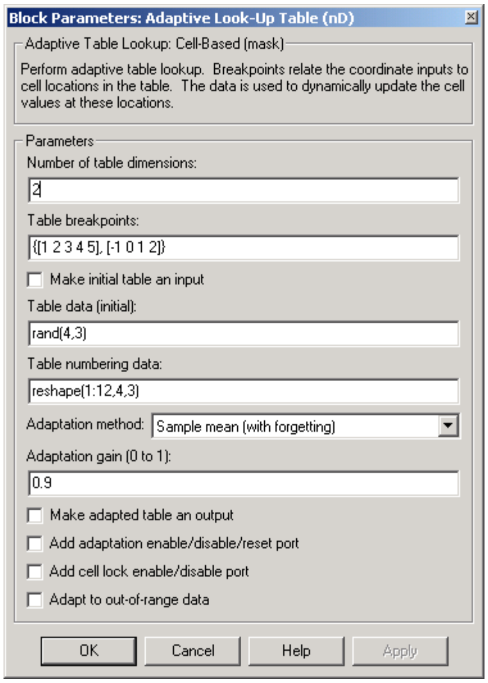


The Adaptive Look-Up Table (n-D) block creates an adaptive lookup table of arbitrary dimension by dynamically updating the underlying lookup table. The block uses the outputs of your system to do the adaptations.

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 Look-Up Table (n-D) to model time-varying systems.

Dialog Box



Block Parameters: Adaptive Look-Up Table (nD)

Adaptive Table Lookup: Cell-Based (mask)

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

Parameters

Number of table dimensions: 2

Table breakpoints: {[1 2 3 4 5], [-1 0 1 2]}

☐ Make initial table an input

Table data (initial): rand(4,3)

Table numbering data: reshape(1:12,4,3)

Adaptation method: Sample mean (with forgetting)

Adaptation gain (0 to 1): 0.9

☐ Make adapted table an output

☐ Add adaptation enable/disable/reset port

☐ Add cell lock enable/disable port

☐ Adapt to out-of-range data

Number of table dimensions

The number of dimensions for the adaptive look-up table.

Table breakpoints

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

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

Make initial table an input

Selecting this box forces the Adaptive Look-Up Table (n-D) 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. Because of Simulink's current limitations,

Table data (initial)

The initial table output values. This (n-D) array must be of size (n-1)-by-(n-1) ... -by- (n-1), (D times) where D is the number of dimensions and n is the number of input breakpoints.

Table numbering data

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

Adaptation method

Choose **Sample mean** or **Sample mean with forgetting**. Sample mean averages all the values received within a cell. Sample mean with forgetting gives more weight to the new data. How much weight is determined by the **Adaptation gain** parameter.

Adaptation gain (0 to 1)

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

Make adapted table an output

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

Add adaptation enable/disable/reset port

Add an input port that enables, disables, or resets the adaptive look-up table.

Add cell lock enable/disable port

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

Index

A
acker 5-8