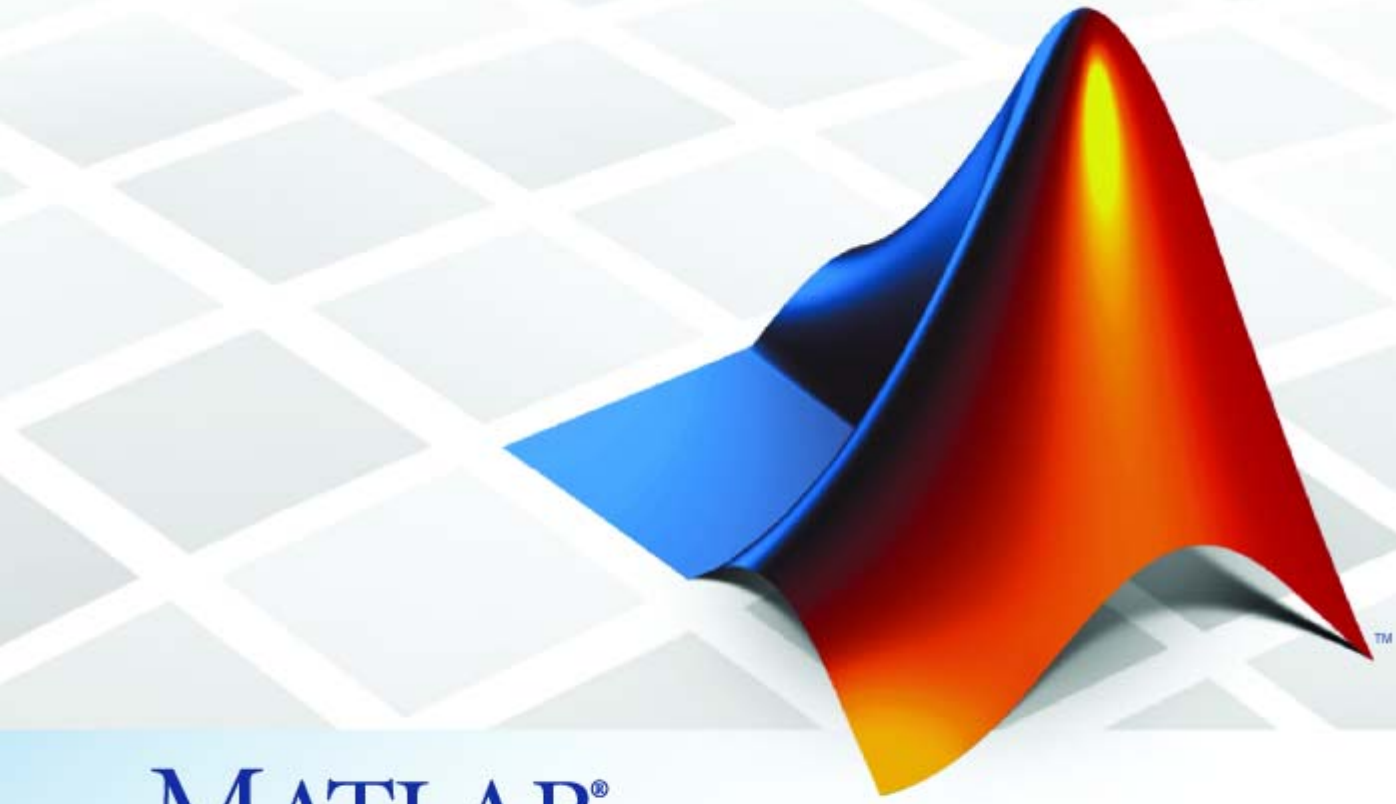
Model-Based Calibration Toolbox™ 3

CAGE 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.

Model-Based Calibration Toolbox™ CAGE User's Guide

© COPYRIGHT 2001–2009 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

December 2001	Online only	New for Version 1.0 (Release 12.1)
August 2002	Online only	Revised for Version 1.1 (Release 13)
May 2003	Online only	Revised for Version 2.0 (Release 13+)
June 2004	Online only	Revised for Version 2.1 (Release 14)
June 2004	Online only	Revised for Version 2.1.1 (Release 14+)
November 2005	Online only	Revised for Version 3.0 (Release 14SP3+)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.4 (Release 2008a)
October 2008	Online only	Revised for Version 3.4.1 (Release 2008a+)
October 2008	Online only	Revised for Version 3.5 (Release 2008b)
March 2009	Online only	Revised for Version 3.6 (Release 2009a)

Contents

1 Getting Started

What Is CAGE? 1-2

Navigating CAGE 1-4
How to Select CAGE Views 1-4
CAGE Views and Processes 1-6

How to Use This Manual 1-9

2 Variables and Models

Setting Up Variable Items 2-2
Introducing the Variable Dictionary View 2-2
Importing and Exportin g a Variable Dictionary 2-4
Adding and Editing Variable Items 2-5
Using the Variable Menu 2-7
Using Aliases 2-8

Setting Up Models 2-10
Introducing the Models View 2-10
Importing Models 2-13
Adding New Function Models 2-15
Renaming and Editing Models 2-17

Model Properties 2-20
How To Open The Model Properties Dialog Box 2-20
Model Properties: General 2-21
Model Properties: Inputs 2-22
Model Properties: Model 2-23
Model Properties: Information 2-24

CAGE Import Tool 2-25

Specifying Locations of Files 2-29

3 Tables

Setting Up Tables 3-2

Adding, Duplicating and Deleting Tables 3-3
- Adding Tables 3-3
- Duplicating Tables 3-4
- Deleting Tables 3-4

Editing Tables 3-6
- Introducing the Table View 3-6
- Viewing and Editing a Table 3-10
- Using the Graph of the Table 3-11
- Filling a Table From a Model 3-12
- Filling a Table by Extrapolation 3-12
- Table Menu 3-13
- Arithmetic Operations On Table Values 3-15

Filling a Table From a Model 3-17

Using the History Display 3-20
- Introducing the History Display 3-20
- Resetting to Previous Versions 3-21
- Comparing Versions 3-23

Calibration Manager 3-24
- Introducing the Calibration Manager 3-24
- Setting Up Tables from a Calibration File 3-24
- Setting Up Tables Manually 3-28
- Copying Table Data from Other Sources 3-28

Table Properties 3-30
- Opening the Table Properties Dialog Box 3-30

Table Properties: General Tab 3-30
Table Properties: Table Values Precision Tab 3-30
Table Properties: Inputs Tab 3-36

About Normalizers 3-37

Normalizer View 3-39
Introducing the Normalizer View 3-39
Editing Breakpoints 3-41
Input/Output Display 3-42
Normalizer Display 3-42
Breakpoint Spacing Display 3-43

Inverting a Table 3-46
Overview of Inverting Tables 3-46
Inverting One-Dimensional Tables 3-48
Inverting Two-Dimensional Tables 3-50

Importing and Exporting Calibrations 3-53
Formats 3-53
Importing Calibrations 3-53
Exporting Calibrations 3-55

4 Feature Calibrations

Perform ing Feature Calibrations 4-2

Settin g Up a Feature Calibration 4-5
Procedure Overview 4-5
Addin g a Feature 4-7
Assigning a Model 4-7
Setti ng Up Your Strategy 4-7

Cali brating the Normalizers 4-16
Overview of Calibrating Normalizers 4-16
Init ializing Breakpoints 4-17
Fill ing Breakpoints 4-18

Optimizing Breakpoints 4-22
Viewing the Normalizer Comparison Pane 4-26

Calibrating the Tables 4-29
Overview of Calibrating Feature Tables 4-29
Initializing Table Values 4-30
Filling Table Values 4-31
Comparing the Strategy and the Model 4-33
Filling the Table by Extrapolation 4-35

Calibrating the Feature Node 4-38
How to Calibrate the Feature 4-38
Initializing the Feature 4-38
Feature Fill Wizard 4-40
Feature View 4-47
Feature Menu 4-48

5 Tradeoff Calibrations

Performin g a Tradeoff Calibration 5-2

Setting U p a Tradeoff Calibration 5-5
Overview of Setting Up a Tradeoff 5-5
Adding a Tradeoff 5-6
Adding T ables to a Tradeoff 5-6
Display ing Models in Tradeoff 5-9

Calibr ating Tables in a Tradeoff Calibration 5-11
Procedure For Calibrating Tradeoff Tables 5-11
Settin g Values of Other Variables 5-14
Deter mining a Value at a Speci Þc Operating Point 5-16
Trade off Table Menus 5-18

Usin g Regions 5-22
What Are Regions? 5-22
DeÞning a Region 5-23
Clearing a Region 5-23

Point-by-Point Model Tradeoffs 5-25
What Is A Point-by-Point Model Tradeoff? 5-25
Adding a Point-by-Point Model Tradeoff 5-26
Calibrating Using a Point-by-Point Model Tradeoff 5-29

Automated Tradeoff 5-32
Using Automated Tradeoff 5-32
What Are Appropriate Optimizations? 5-34

6 Optimization

Using the Optimization View 6-2

Optimization Problems You Can Solve with CAGE 6-4
Point-by-Point Optimization Problems 6-4
Sum Optimization Problems 6-5

Creating an Optimization 6-7
Overview of Creating Optimizations 6-7
Setting Up Point-by-Point Optimizations 6-8
Optimization Wizard 6-8
Optimization View Toolbar 6-15
Setting Up Sum Optimizations 6-16
Parallel Computing in Optimization 6-25

De Þning Variable Values 6-27
What Are Variable Values? 6-27
DeÞne Variables Manually 6-27
Import from a Data Set 6-29
Import from Output 6-31
Import from Table Grid 6-34
Import from Table Values 6-35

Objectives and Constraints 6-36
Overview of Objectives and Constraints 6-36
Edit Objective 6-37
Edit Constraint 6-39

Running Optimizations 6-44

Using the Optimization Parameters Dialog Box 6-46
Overview of the Optimization Parameters Dialog Box 6-46
foptcon Optimization Parameters 6-46
NBI Optimization Parameters 6-49
GA Optimization Parameters 6-54
Pattern Search Optimization Parameters 6-57
Scale Optimization 6-60

Optimization Output Views 6-61
Using the Optimization Output View 6-61
Using Acceptable Solutions 6-63
Solution Slice: Optimization Results Table 6-65
Solution Slice: Results Surface and Results Contour Views 6-67
Pareto Slice 6-72
Weighted Objective Pareto Slice 6-74
Selected Solution Slice 6-76
Objective Slice Graphs 6-78
Objective Contour Plot 6-79
Pareto Front Graphs 6-80
Constraint Slice Graphs 6-81
Constraint Summary Table 6-83

Using Optimization Output 6-89
Create Sum Optimization from Point-by-Point Optimization Output 6-89
Exporting to a Data Set 6-90
Filling Tables from Optimization Results 6-92
Custom Fill Function Structure 6-95

Analyzing Point-by-Point Optimization Output 6-98
Process for Analyzing Optimization Results 6-98
Detecting Local Optima 6-101
Investigating Early Termination of Optimization 6-105
Handling Flat Optima 6-110

Interpreting Sum Optimization Output 6-114
Operating Point Indices 6-114
Optimization Results Table 6-115
Objective Graphs 6-116

Objective Contour Plot 6-117
Constraint Graphs 6-118
Constraint Summary 6-119
Table Gradient Constraint Output 6-120

User-De Þned Optimization 6-128
Introducing User-De Þned Optimization 6-128
Implementing Your Optimization Algorithm in CAGE . . . 6-129
About the Worked Example Optimization Algorithm 6-131
Checking User-De Þned Optimizations into CAGE 6-133

Optimization Function Reference 6-136
Methods of cgoptimoptions 6-136
Methods of cgoptimstore 6-138

Functions — Alphabetical List 6-141

7 Data Sets

Data Sets Views 7-2

Setting Up Data Sets 7-4
How to Set Up Data Sets 7-4
Importing Experimental Data 7-5
Importing Data from a Table in Your Session 7-7
Merging Data Sets 7-8
Specifying the Factors Manually 7-8
Creating a Factor from the Error Between Factors 7-12

Viewing Data in a Table 7-13

Plotting Outputs 7-15

Using Color to Display Information 7-18

Linking Factors in a Data Set 7-23

Assigning Columns of Data 7-25

Manipulating Models in Data Set View 7-26

Filling Tables from Experimental Data 7-27
How to Fill Tables from Experimental Data 7-27
Creating Rules 7-30

8 Surface Viewer

The Surface Viewer in CAGE 8-2

Viewing a Model or Strategy 8-3

Setting Variable Ranges 8-5
Displaying Point-by-Point Models in the Surface Viewer . . 8-6

Displaying the Model or Feature 8-8
Using Display Options 8-8
Surface 8-9
Contour 8-11
Line 8-12
Single Value 8-12
Multiline 8-13
Table 8-13

Making Movies 8-15

Displaying Errors 8-17
Introducing Error Displays 8-17
Feature Error Data 8-17
Prediction Error Data 8-17

Printing and Exporting the Display 8-19

1

Getting Started

This section includes the following topics:

- “What Is CAGE?” on page 1-2
- “Navigating CAGE” on page 1-4
- “How to Use This Manual” on page 1-9

What Is CAGE?

CAGE (CAlibration GEneration) is an easy-to-use graphical interface for calibrating lookup tables for your electronic control unit (ECU).

As engines get more complicated, and models of engine behavior more intricate, it is increasingly difficult to rely on intuition alone to calibrate lookup tables. CAGE provides analytical methods for calibrating lookup tables.

CAGE uses models of the engine control subsystems to calibrate lookup tables. With CAGE you fill and optimize lookup tables in existing ECU software using models from the Model Browser part of the Model-Based Calibration Toolbox™ product. From these models, CAGE builds steady-state ECU calibrations.

CAGE also compares lookup tables directly to experimental data for validation.

Feature Calibration

A feature calibration compares a model of an estimated signal with a lookup table (or algebraic collection of tables) that estimates the same signal in the ECU. CAGE finds the optimum calibration for the lookup table(s).

For example, a typical engine subsystem controls the spark angle to produce the peak torque; that is, the Maximum Brake Torque (MBT) spark. Using the Model Browser, you can build a statistically sound model of MBT spark, over a range of engine speeds and relative air charges, or loads. Use the feature calibration to fill a lookup table by comparing the table to the model.

Tradeoff Calibration

A tradeoff calibration fills lookup tables by comparing models of different engine characteristics at key operating points.

For example, there are several models of important engine characteristics, such as torque and nitrous oxides (NOX) emissions. Both models depend on the spark angle. At a particular operating point, a slight reduction of torque can result in a dramatic reduction of NOX emissions. Thus, the calibrator

uses the value of the spark angle that g ives this reduction in NOX emissions instead of the spark angle that generates maximum torque.

Optimization

CAGE can optimize calibrations wit h reference to models, including single- and multi-objective optimiz ations, sum optimizations, user-de Þned optimizations, and automated tradeoff.

Comparing Calibrations to Data

You can compare your calibrations to experimental data for validation.

For example, after completing a calib ration, you can import experimental data from a spreadsheet. You can use CAGE to compare your calibration to the data.

Starting the CAGE Browser

To start the application, type

```
cage
```

at the MATLAB ® command prompt.

Navigating CAGE

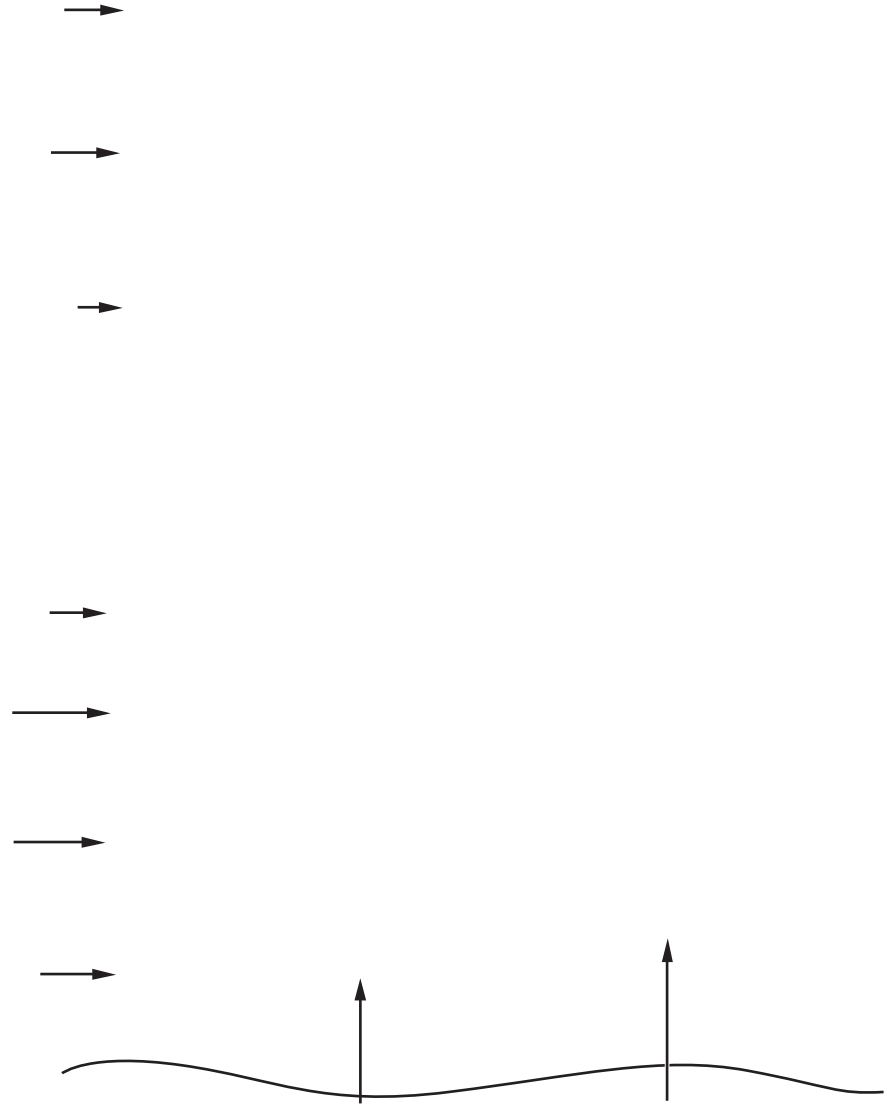
In this section...
“How to Select CAGE Views” on page 1-4
“CAGE Views and Processes” on page 1-6

How to Select CAGE Views

The view of CAGE depends on two things:

- Which button you select in the Processes and Data Objects panes
- The item you highlight in the tree display

When you open CAGE, it looks like this.



CAGE includes a Processes pane and a Data Objects pane to help you identify the type of calibration you want to do and the data objects that you intend to use. Use the buttons in these panes to navigate between the different sections of functionality in CAGE.

CAGE Views and Processes

The Processes pane has three buttons:

- Feature shows the Feature view, with the tables and strategies that are associated with that feature. See “Feature View” on page 4-47.

  A feature is a strategy (or collection of tables) and a model used to calibrate those tables. In the Feature view, you can fill tables by comparing a strategy to a model. See Chapter 4, “Feature Calibrations”. You can import existing strategies or construct new ones using Simulink® software from the feature view.

  From the feature node in the tree display, you can access the Surface Viewer to examine the strategy or model or both. See Chapter 8, “Surface Viewer”.

- Tradeoff shows the Tradeoff view, with a list of the tables and models to display. Here you can see graphically the effects of manually altering variables to trade off different objectives (such as maximizing torque while minimizing emissions). At the tradeoff node, you can calibrate table values to achieve the best compromise between competing objectives. You can calibrate using single or multimodel tradeoffs. See Chapter 5, “Tradeoff Calibrations”. You can also use the optimization functionality of CAGE to run automated tradeoffs, described in the Optimization section (see below).

- Optimization shows the Optimization view. From here you can set up and run optimizations, including automated tradeoffs. There are standard routines available and also templates provided so you can write your own optimization routines. You can find full instructions in Chapter 6, “Optimization”.

  You can reach the Calibration Manager from the Feature and Tradeoff process views, and from the Tables view, but not Optimization. In the Calibration Manager you can set up the size and contents of tables (manually or using existing calibration files) and edit the precision used for values (to match the kind of electronic control unit you are going to use). See “Calibration Manager” on page 3-24.

The Data Objects pane has four buttons:

- Variable Dictionary stores all the variables, constants, and formulas in your session. Here you can view, add, and edit any variables in any part of your session. See “Setting Up Variable Items” on page 2-2.
- Tables enables you to see all the tables and normalizers in your session. You can also calibrate tables manually here if you want. You can add and delete tables from the project. From any table display (here, or in other views) you can access the History Display to manage changes in your tables and normalizers. You can use the History Display to reverse changes. See “Setting Up Tables” on page 3-2.
- Models stores all the models in your session. Here you can view a graphical display of these models, including a diagram of the model’s input structure. This is useful because a model can have other models as inputs. You can change the inputs here. For example, you can change your model’s input Spark to be connected to a model for Spark rather than to the variable Spark. You can also access the surface viewer here to examine models. See “Setting Up Models” on page 2-10 and Chapter 8, “Surface Viewer”.

- Data Sets enables you to evaluate your models and features over a custom set of input values. Here you can create and edit a set of input values and view several models or features evaluated at these points. You can compare your tables and models with experimental data to validate your calibrations. You can also fill tables directly from experimental data by loading the experimental data as a new data set. See Chapter 7, “Data Sets”.

How to Use This Manual

This manual is the CAGE User's Guide. See also the Model Browser User's Guide for information on the other main interface of the Model-Based Calibration Toolbox product.

Learning CAGE

For tutorials and case studies, see Getting Started in the Model-Based Calibration Toolbox Getting Started Guide .

Using CAGE

- Chapter 2, "Variables and Models" describes how to set up CAGE sessions before performing calibrations and gives an overview of where in CAGE to find all the functionality for different processes.
- Chapter 3, "Tables" describes how to create and use tables and normalizers, including using the Calibration Manager and History Viewer.
- Chapter 4, "Feature Calibrations" describes how to calibrate lookup tables by reference to models built using the model browser.
- Chapter 5, "Tradeoff Calibrations" describes how to calibrate lookup tables by adjusting many values to fill different objectives.
- Chapter 6, "Optimization" describes how to use the optimization functions, including automated tradeoffs, and describes all the functions available for user-defined optimizations.
- Chapter 7, "Data Sets" describes how to use CAGE to compare calibrations to experimental data, and how to use experimental data to fill lookup tables.
- Chapter 8, "Surface Viewer" describes how to use the Surface Viewer.

Variables and Models

The following sections describe how t o set up variables and models before performing calibrations.

- “Setting Up Variable Items” on page 2-2
- “Setting Up Models” on page 2-10
- “Model Properties” on page 2-20
- “CAGE Import Tool” on page 2-25
- “Specifying Locations of Files” on page 2-29

Setting Up Variable Items

In this section...

“Introducing the Variable Dictionary View” on page 2-2

“Importing and Exporting a Var iable Dictionary” on page 2-4

“Adding and Editing Variable Items” on page 2-5

“Using the Variable Menu” on page 2-7

“Using Aliases” on page 2-8

Introducing the Variable Dictionary View

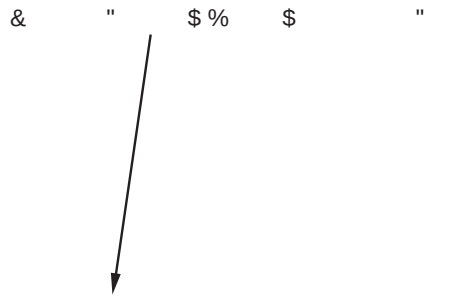
The Variable Dictionary is a store for all the variables, constants, and formulae in your session.

To view or edit the items in the Variabl e Dictionary, click the button, shown, in the Data Objects pane.

Selecting the Variable Dictionary view displays the variables, constants, and formulae in the current project.

Note that if you have existing CAGE p rojects you can use the “CAGE Import Tool” on page 2-25 to import variable items and other CAGE items directly from other projects.

Following is an example of the Variable Dictionary view.



The upper pane shows a list of all the current variables, constants, and formulas. The lower pane displays edit boxes so you can specify the settings of the selected variable, constant, or formula.

Different Variable Dictionary Items

- Variables — standard items that feed into models, strategies and tables, and define ranges for these items
- Constant — used for inputs that you do not want to change
- Formulae — used when you want a variable item to depend on another

Importing and Exporting a Variable Dictionary

A variable dictionary contains all the variable items for your calibrations. You can set up your variable dictionary once, and use it in many calibrations.

If you import a model, it has variables associated with it, in which case you might not have to import a variable dictionary.

Importing a Variable Dictionary

To import a dictionary of variables from an .xml file,

1 Select File > Import > Variable Dictionary.

2 Select the correct dictionary file.

Note you can also import variable items directly from other CAGE projects using the “CAGE Import Tool” on page 2-25.

Exporting a Variable Dictionary

After setting up a variable dictionary, you can save the dictionary for use in many different calibrations.

To export a dictionary of variables to an .xml file,

1 Select File > Export > Variable Dictionary.

2 Select a suitable name for the dictionary file.

See Also

- “Setting Up Variable Items” on page 2-2
- “Adding and Editing Variable Items” on page 2-5

Adding and Editing Variable Items

To add variable items you can use the Variable Dictionary toolbar, shown, or you can select items from the File -> New -> Variable Items menu.

Adding a Variable

To add a variable,

1. Select File > New > Variable Item > Variable .

   A new variable is added to the variable dictionary.

2. Select Edit > Rename to alter the name of the variable.

3. Specify the Minimum and Maximum values of the variable in the edit boxes in the lower pane.

4. Specify the value of the Set Point in the edit box.

Using Set Points in the Variable Dictionary. The set point of a variable is a point that is of particular interest in the range of the variable. You can edit set points in the variable di ctionary or the models view.

For example, for the air/fuel ratio variable, AFR, the range of values is typically 11 to 17. However, whenever only one value of AFR is required, it is preferable to choose 14.3, the stoichiom etric constant, over any other value. So enter 14.3 as the Set Point .

CAGE uses the set point as the default value of the variable wherever one value from the variable range is requi red. For instance, CAGE uses the set point when evaluating a model over the range of a different variable.

For example, a simple model for torq ue depends on AFR, engine speed, and relative air charge. CAGE uses the set point of AFR when it calculates the values of the model over the ranges of the engine speed and relative air charge.

Adding a Constant

To add a constant,

1 Select File > New > Variable Item > Constant .

A new constant is added to t he variable dictionary.

2 Select Edit > Rename to alter the name of the constant.

3 Specify the value of the constant in the Set Point edit box, in the lower pane.

Adding Formulas

You might want to add a formula to your session. For example, the formula

$$= \frac{\text{afr}}{\text{stoich}}$$

where afr is the air/fuel ratio and stoich is the stoichiom etric constant.

To add a formula,

1 Select File > New > Variable Item > Formula .

The Add Formula dialog box appears.

2 In the dialog, enter the right side of the formula, as in this example afr/stoich . Note it is normal to create inputs to a formula Þrst. If you do not use pre-existing variable names then those inputs are created, so be careful to get input names exactly c orrect. Follow these requirements for a valid formula string:

- A formula can only have exactly one variable input
- No formulae as inputs
- Not circular (i.e. self referencing)
- Must not error when evaluated
- Must produce a vector for a vector input
- Must be invertible

Click OK and a new formula is added to the variable dictionary.

3 Select Edit -> Rename to alter the name of the formula.

See Also

- “Setting Up Variable Items” on page 2-2
- “Adding and Editing Variable Items” on page 2-5

Using the Variable Menu

The Variable menu in the variable dictionary enables you to alter variable items. These choices are also available in the right-click context menu on the list view.

Change item to:

- Alias

  Changes the selected item to be an alias of another item in the current project. For example, if you have two variables, engine_speed and n, you can change n to be an alias of engine_speed, with its maximum and minimum values. For more information, see the next section, “Using Aliases” on page 2-8.

- Formula

  Changes a variable or constant into a formula. You have to define the right side of the formula, and you can select the check box to calculate the range.

- Constant

  Changes a variable or formula into a constant. The value of the constant is the set point of the old item.

- Variable

  Changes a constant or formula into a variable. The range is from 0 to twice the constant's value (negative values have a maximum of 0).

See Also

- "Setting Up Variable Items" on page 2-2
- "Using Aliases" on page 2-8

Using Aliases

The variable dictionary enables you to use the same set of variables, constants, and formulas with many different models and calibrations.

Why Use Aliases?

It is possible that in one model or strategy the engine speed has been defined as N, and in another it has been defined as rpm. The alias function enables you to automatically link inputs with various names to a single CAGE variable when you import models and strategies.

Creating an Alias

For example, in a variable dictionary there are two variables:

- N, with a range of 500 to 6500
- rpm, with a range of 2500 to 3500

To set rpm to be an alias of N,

1. Highlight the variable rpm.
2. Select Variable > Change item to > Alias .
3. In the dialog, choose N from the list.

This eliminates the variable rpm from your variable dictionary, and every model and calibration that refers to rpm now refers to N instead.

Note If N is made an alias of rpm in the preceding example, the range of N is restricted to the range of rpm, 2500 to 3500.

You can also add aliases to existing items by entering a list of names in the Alias edit box.

See Also

- “Setting Up Variable Items” on page 2-2

Setting Up Models

In this section...

“Introducing the Models View” on page 2-10

“Importing Models” on page 2-13

“Adding New Function Models” on page 2-15

“Renaming and Editing Models” on page 2-17

Introducing the Models View

CAGE generally calibrates lookup tables by reference to models. The Models view is a storage place for all the models in your session.

To view and edit the models in your session, select Models by clicking the button shown in the Data Objects pane.

The Models view displays the following:

- A list of all the models in the current project.
- The model connections. That is, which constants, variables, and models are inputs to the selected model. You can use the View menu or the right-click context menu on the graph to zoom in and out, zoom to fit, and reset.
- An image of the response surface of the selected model; you can select factors to display. Use the View menu to choose between:
  - No Constraint Display — Shows entire model surface.
  - Show Constraint — Areas outside the boundary constraint model (if any) are yellow.
  - Clip to Constraint — The surface is only shown within the boundary constraint model.

View > Edit Input Set Points opens a dialog box where you can edit the set points of your model variables. This setting alters the model display and also any calculations involving the set points throughout CAGE. Altering this setting is the same as altering the set points in the Variable Dictionary, see “Using Set Points in the Variable Dictionary” on page 2-5.

Following is an example of the Models display.

The icons in the Models list indicate the type of model, as listed in the Type column. As shown in the following illustration, a model can be a Model Browser statistical model, the boundary of a model, the prediction error

variance (PEV) of a model, a user-de Þned function model, or a feature model (converted from a feature).

You can use the “Model Properties” on page 2-20 dialog to switch a model output between the model value and th e boundary or PEV of the model. For function models see “Adding New Function Models” on page 2-15. You can convert a feature to a model by selecting Feature > Convert to Model .

Importing Models

CAGE enables you to calibrate loo kup tables by referring to models constructed in the Model Browser.

CAGE can only open Model-Based Calibration Toolbox model Þles. You can import models from project Þles (.mat , .cag) and from exported model Þles (.exm).

Import Mod els From Project

You can use the CAGE Import Tool to select models to import from any Model-Based Calibra tion Toolbox project Þle produced in CAGE or the Model Browser (.mat or .cag). You can replace suitable models in your current CAGE project (note that Model Browser models must have exactly the same input names as the CAGE model you are replacing).

See “CAGE Import Tool” on page 2-25 for instructions.

Import Ex ported Models File

To import models from a Model Browser exported models Þle (.exm):

1 Select File > Import > Model .

2 A Þle browser dialog opens. Locate the desired Þle or Þles. You can select multiple Þles. Examples can be found in matlab/toolbox/mbc/mbctraining . You can select MBC Model (*.exm) to Þlter for .exm Þles.

Click to select the model Þle, then click Open .

This opens the Model Import Wizard.

3 Select the models that you want to import by highlighting the models from the list, or click Select All if you want every model.

4 Either:

- Select the check box Automatically ass ign/create inputs , then you can click Finish .
- Alternatively to match inputs up manually, instead click Next .

5 Associate the model factors with th e available inputs in your session.

For example, to associate the model factor spark with the variable spk in your session,

a Highlight a Model Input , spark , in the list on the left and the corresponding variable, spark , in the list of Available Inputs on the right.

b Click the Assign Input button.

c Repeat a and b for all the model factors.

6 Click Finish to close the wizard and return to the Models view.

Note You can skip steps 5 and 6 by selecting the Automatically assign/create inputs box at step 6.

You can now see a display of the model surface and the model connections (inputs).

See Also

- “Setting Up Models” on page 2-10
- “Adding New Function Models” on page 2-15
- “Renaming and Editing Models” on page 2-17

Adding New Function Models

A function model is a model that is expressed algebraically. The function can be any MATLAB function (including user-defined functions). The only restriction is that the function must be vectorized, that is, take in column vectors and return a column vector of the same size, as in this example:

```
function y = foo(x1, x2)
y = x1 .* x2;
```

Once you have a function like this, you can create a function model applying it to any models or variables in your session, like the following example.

```
foo(NOX, SPK)
```

For example, you might want to view the behavior of torque efficiency. So you create a function model of torque efficiency = torque/peak torque.

To add a function model to your session,

1 Select File > New > Function Model.

This opens the Function Model Wizard.

2 In the dialog box, enter the formula for your function model. For example, enter torque_efficiency=torque/peak_torque .

3 Press Enter . CAGE checks that the function is recognized; if so, you can click Next . If the function is incorrect ly entered, you cannot click Next .

4 Select the models that you want to import by highlighting the models from the list.

5 Click Next .

6 You can select the check box to Automatically assign/create inputs and click Finish to close the wizard and return you to the Models view, or you can click Next and go to the next screen. Here you can manually associate the model factors with the available inputs as follows:

a Highlight a Model Input , e.g., peak_torque , in the list on the left and the corresponding model, peak_torque , in the Available Inputs list on the right.

b Click the Assign input button.

Repeat a and b for all the model factors. Click Finish to close the wizard and return you to the Models view.

You can now see a display of the model and its connections (inputs).

See Also

- “Setting Up Models” on page 2-10
- “Importing Models” on page 2-13
- “Renaming and Editing Models” on page 2-17

Renaming and Editing Models

Renaming Models

To rename a model,

1 Highlight the model that you want to rename.

2 Select Edit > Rename .

3 Enter the new name for the model and press Enter .

You can also rename the model by select ing a model and clicking the name, or pressing F2.

Editing Model Inputs

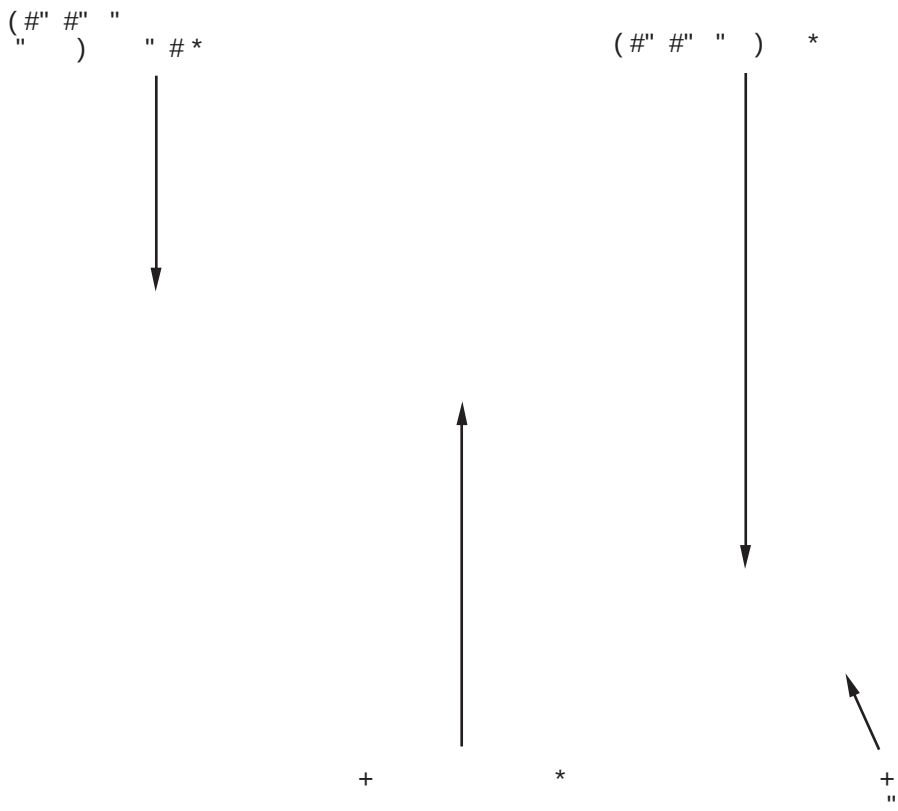
You can adjust a model so that variables, formulas, or other models are the factors of the model. For example, a model of torque depends on the spark angle. In place of the spark angle variable, you can use a model of the maximum brake torque (MBT) as the spark input.

To edit the inputs of a model,

1 Highlight the model.

2 Select Model > Edit Inputs .

This opens the Edit Inputs dialog box, shown.



3 Highlight the Model Input that you want to edit, in the list on the left.

4 Highlight the new input for that factor, in the Available Inputs list on the right.

5 Click the Assign Input button.

6 To close the dialog box, click Finish .

Note If you want to change the range of a variable in the session, change the range in the variable dictionary. For more information, see “Using the Variable Menu” on page 2-7.

Model Properties

In this section...

"How To Open The Model Properties Dialog Box" on page 2-20
"Model Properties: General" on page 2-21
"Model Properties: Inputs" on page 2-22
"Model Properties: Model" on page 2-23
"Model Properties: Information" on page 2-24

How To Open The Model Properties Dialog Box

Select Model > Properties (or right-click) to view information about the selected model. This opens the Model Properties dialog box where you can see the model type, definition, inputs, availability of PEV and constraints, creation date, user name, and toolbox version on the following tabs: General, Inputs, Model, and Information.

Model Properties: General

Here you can see the model type (such as MBC model or function model), the number of inputs, and the availability of constraints and Prediction Error.

You can use the radio buttons to select the Output Quantity to be the

- Model Value
- Prediction error variance of model
- Boundary constraint of model

The Output Quantity is the model value used everywhere in CAGE (surface plots, optimization objectives or constraints, tradeoff, etc.).

Choose one of the last two options if you want to use a model's prediction error variance (PEV) or boundary as a switching input to a function model. You

can duplicate the original model, choose the PEV output quantity, and feed it in to your switch function.

The option Boundary constraint of model evaluates only the boundary of the model output. Any boundary information from the inputs is ignored (e.g., if inputs are also models with boundary models).

You can enter values in the Output saturation limits edit boxes to set bounds on the model output values.

Model Properties: Inputs

Here you can view all the immediate inputs and variable dependencies of your model. For some models the two lists will be the same; in the example shown one of the inputs is another model (MBT) so the variable dependencies list also shows the variable inputs for that model. This information is shown graphically in the Connections pane.

Model Properties: Model

Here you can view the model definition, the project file, and the model path. Function model definitions are shown here. For MBC models the model definition (showing the parameters and coefficients of the model formula) is the same information you would see in the Model Browser part of the toolbox when selecting View > Model Definition.

Model Properties: Information

Here you can see the user name associated with the model, the date of creation and the version number of the Model-Based Calibration Toolbox product used to create the model. If you added any comments to the export information in the Model Browser Export Models dialo g this information also appears here.

CAGE Import Tool

You can use the CAGE Import Tool to select items to import from any Model-Based Calibration Toolbox project file produced in CAGE or the Model Browser (.mat or .cag). This can greatly simplify setting up new projects, and also making changes to existing projects, for example to make use of new models in an existing optimization and calibration.

You can import Model Browser models from any project file or direct from the Model Browser when it is open. You can import the following CAGE items from any CAGE project: models (including feature and function models), variables, normalizers, tables, features, optimizations, data sets and tradeoffs.

You can replace suitable items in your current CAGE project with imported items. You can see if an item is replaceable in the Import dialog, where the Replace action becomes available.

Note that Model Browser models (but not CAGE models) must have exactly the same input names as the CAGE model you want to replace. You can replace models, variables, normalizers, tables and features. You cannot replace optimizations, datasets or tradeoffs. You cannot replace tables used in tradeoffs with tables of a different size.

To use the CAGE Import Tool:

1 Select File > Import From Project.

  The CAGE Import Tool appears.

2 You can choose a project file or import directly from the Model Browser if it is open.

  - If the Model Browser is open, the list is automatically populated with a list of available items in the open project.
  - To import from a file, click the Import From Project File button.

    A file browser dialog opens. Locate the desired file and click Open.

3 The CAGE Import Tool displays the available items. Select the items you want to import from the list. Press Ctrl+A to select all items, or Ctrl+click or Shift+click to select multiple items in the list.

You can use the Find and Type controls to filter the item list:

- If you are importing from a Model Browser project you can select Response, Switch, Datum or Response Feature from the Type list to display a single model type only.
- If you are importing from a CAGE project you can select Variable, Model, Normalizer, Table, Feature, Optimization, Dataset, or Tradeoff from the CAGE items in the Type list. For models the Subtype column displays whether a model item is an MBC model, function model or feature model.
- Enter text in the Find edit box to find particular item names. You can also select the box to Match case

4 Click the Import Selected Items button.

5 The Import dialog opens displaying the items you selected for import.

- To edit item names, double-click the column cells of the CAGE Item Name (or CAGE Model Name if importing models).
- If it is not possible to replace items in the current CAGE session then Create new is displayed in the Action column. If it is possible to replace an item in the current CAGE session with an imported item, the Action column cell becomes a drop-down menu where you can select Replace or Create new. If an exact name match item is available to be replaced the Action drop-down menu automatically displays Replace. Change this to Create new if you do not want to replace the existing item.
- When replacing items, double-click the CAGE Item Name column cells to open a dialog to select the correct item to replace.
- Clear the View new item check box if you do not want CAGE to switch to the appropriate view for the top item in the import list when you dismiss the dialog. The CAGE Import Tool remains open either way.
- Click OK to import the items.

6 Click the Close button to close the CAGE Import Tool when you have finished importing items.

See also:

- “Importing and Exporting a Variable Dictionary” on page 2-4

- “Import Exported Models File” on page 2-13

Specifying Locations of Files

You can specify preferred locations of project and data files, using File > Preferences.

Project files have the file extension .cag and store entire CAGE sessions.

Data files are the files that form part of the CAGE session. For example, the following is a list of some of the data files used in CAGE:

- Simulink diagrams (.mdl)
- Experimental data (.xls, .csv, or .mat)
- Variable dictionaries (.xml)
- Models (.exm)

To specify preferred locations for files,

1 Select File > Preferences. This opens the dialog box shown.

2 Enter the directory or directories where your CAGE files are stored. Alternatively, click to browse for a directory. You can specify directories for projects, data files, model files and strategy files.

3 Click OK.

3

Tables

This section includes the following topics:

- “Setting Up Tables” on page 3-2
- “Adding, Duplicating and Deleting Tables” on page 3-3
- “Editing Tables” on page 3-6
- “Filling a Table From a Model” on page 3-17
- “Using the History Display” on page 3-20
- “Calibration Manager” on page 3-24
- “Table Properties” on page 3-30
- “About Normalizers” on page 3-37
- “Normalizer View” on page 3-39
- “Inverting a Table” on page 3-46
- “Importing and Exporting Calibrations” on page 3-53

Setting Up Tables

Select the Tables view by clicking the Tables button. It opens automatically if you add a table using the File > New > Table menu items.

The Tables view lists all the tables and normalizers in the current CAGE session.

Here you can add or delete tables and normalizers, and you can calibrate them manually. Once you have added new tables here you can also fill them using experimental data by going to the Data Sets view.

The next sections cover:

- "Adding, Duplicating and Deleting Tables" on page 3-3
- "Editing Tables" on page 3-6
- "Using the History Display" on page 3-20
- "Calibration Manager" on page 3-24
- "About Normalizers" on page 3-37

You can use the History display (from any other table or normalizer view in CAGE) to view and reverse changes and revert to previous versions of your tables. Use the Calibration Manager to set up tables manually or from calibration files.

See also

- "Editing Tables" on page 3-6 for information on using the table view functionality once you have added tables to your project

Adding, Duplicating and Deleting Tables

In this section...

“Adding Tables” on page 3-3

“Duplicating Tables” on page 3-4

“Deleting Tables” on page 3-4

Adding Tables

To add or delete tables, you can first select the Tables view, or CAGE automatically switches to this view if you add a table using the File > New menu items.

The Tables view lists all the tables and normalizers in the current CAGE session.

To add a table to a session,

1 Decide whether you want to add a one- or a two-dimensional table.

   For example, if you want to add a modifier table to account for the variation in exhaust gas recirculation, add a one-dimensional table (which has one input). If, however, you want to add a table with speed and load as its normalizer inputs, then add a two-dimensional table.

2 Select File > New > 1D Table or File > New > 2D Table as appropriate.

   Adding new tables automatically switches you to the Tables view.

3 In the Table Setup dialog you can enter the table name, number of rows and columns and initial value, and select the input variable (or variables) from the drop-down menus.

4 Click OK to add the new table. CAGE automatically initializes the normalizers of the table by spacing th e breakpoints evenly over the ranges of the selected input variables.

Note You can also select Tools > Calibration Manager to change the size of a table. For information, see “Setting Up Tables” on page 3-2.

You can rename tables by Þrst selecting the table, then

- Press F2, or
- Select Edit > Rename .

You can manually calibrate by entering values in any table. You can also Þll tables using experimental data or optimization output by going to the Data Sets view; see “Tutorial: Filling Tables from Data” in the Getting Started documentation.

Duplicating Tables

To copy a table or a normalizer from a session,

1 Select the Tables view.

2 Highlight the required table or normalizer.

3 Select Edit > Duplicate table_name (‘table_name’ is the currently selected table).

See also “CAGE Import Tool” on page 2-25 to add existing tables from other CAGE project Þles.

Deleting Tables

When you are calibrating a collection of tables using either Feature or Tradeoff calibrations, you cannot easily delete tables without affecting the entire calibration. When deleting items, you must delete from the highest level down. For example, you cannot delete a table that is part of a feature; you must delete the feature Þrst.

To delete a table or a normalizer from a session,

1 Select Tables view.

2 Highlight the required table or normalizer.

3 Click ; or press Delete ; or select Edit > Delete table_name (‘table_name’ is the currently selected table).

Editing Tables

In this section...

“Introducing the Table View” on page 3-6
“Viewing and Editing a Table” on page 3-10
“Using the Graph of the Table” on page 3-11
“Filling a Table From a Model” on page 3-12
“Filling a Table by Extrapolation” on page 3-12
“Table Menu” on page 3-13
“Arithmetic Operations On Table Values” on page 3-15

Introducing the Table View

When you select a table in the tree (under feature or tables), you see the Table view.

Note For feature calibration (Þlling and optimizing table values by comparing a strategy or collection of tables with a model), see “Performing Feature Calibrations” on page 4-2. To Þll a single table with model values, see “Filling a Table From a Model” on page 3-17.

In CAGE, a table is de Þned to be either a one-dimensional or a two-dimensional lookup table. One-di mensional tables are sometimes known as characteristic lines or functions. T wo-dimensional tables are also known as characteristic maps or tables. CAGE regards them both as similar objects.

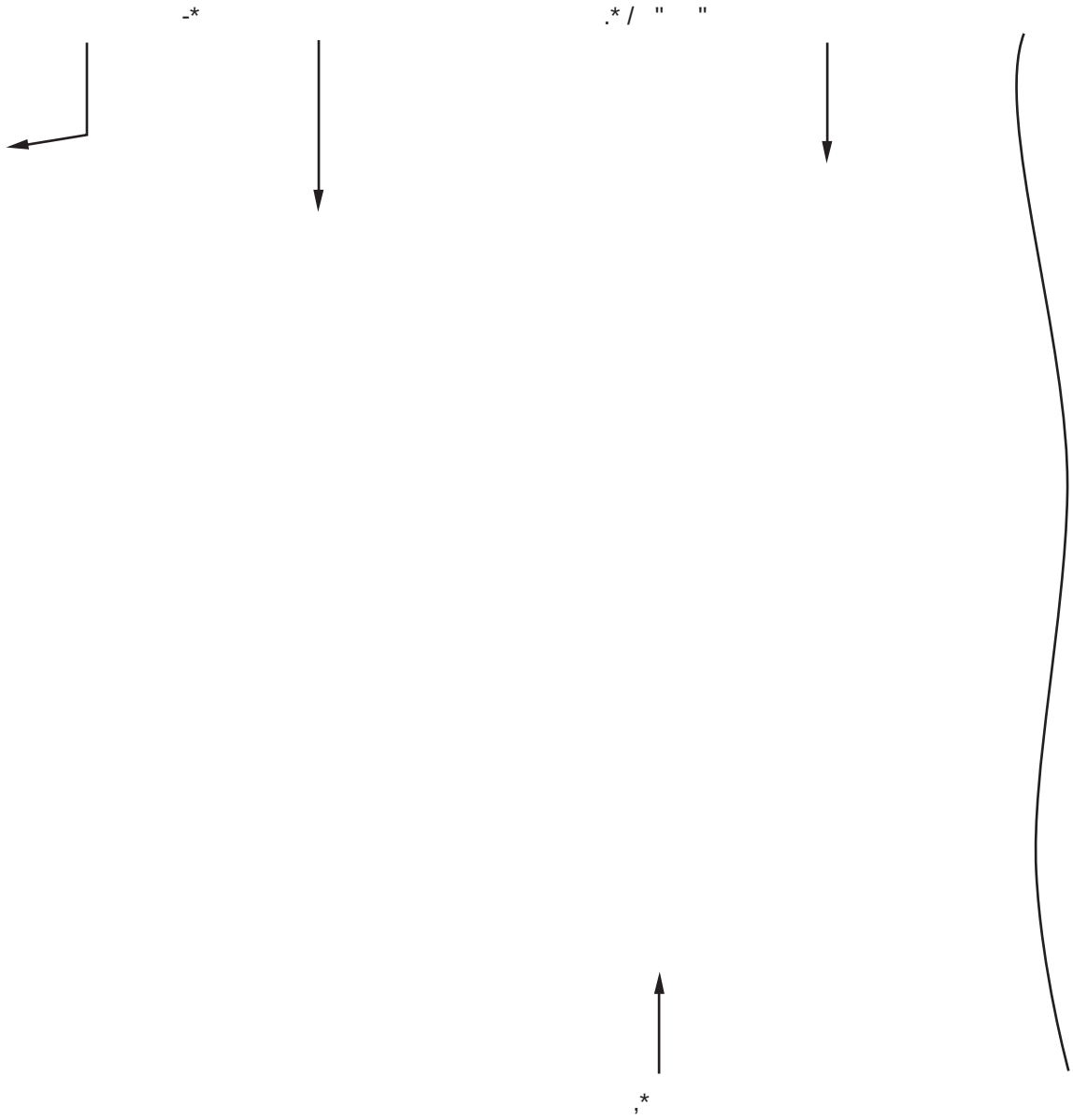
Each lookup table has either one or two axes associated with it. These axes are normalizers. See “About Normalizers” on page 3-37.

For example, a simple MBT feature has two tables:

- A two-dimensional table with speed and relative air charge as its normalizers

- A one-dimensional table with AFR as its normalizer

The example following is a feature view. In the Tables view for manual calibration, you do not see the lower comparison pane because you are not comparing tables with a model.



The parts of the display are numbered and labeled as follows:

1 The table displays the values of the breakpoints and the values of the table.

  The table breakpoint values are not ne cessarily identica l to the normalizer breakpoints. When you create a table the breakpoint values are the same as the normalizer values. If you delete breakpoints from the normalizers the table size does not change, so the table column and row breakpoint values are interpolated between the remaining normalizer breakpoints. (See “Viewing and Editing a Table” on page 3-10.)

2 The graph of the table pane displays the table values graphically. (See “Using the Graph of the Table” on page 3-11.)

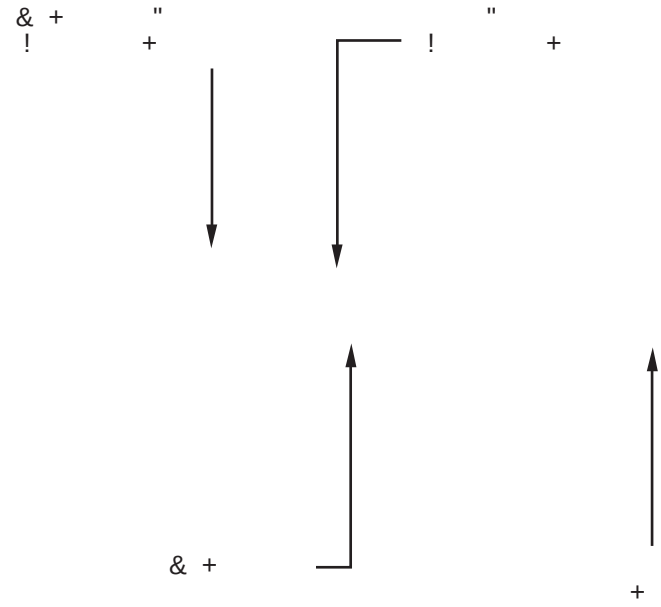
3 The comparison-of-results pane disp lays a comparison between the current output of the strategy and the feature model. (Only visible when calibrating a feature, see “Inverting a Table” on page 3-46.)

Note You can view and revert table changes in the History display by selecting View > History . For information, see “Using the History Display” on page 3-20.

This section describes each of these parts in detail.

Viewing and Edit ing a Table

The table displa ys the values of your lookup table and displays the breakpoints of t he normalizers. For example, the following table shows a lookup table wit h speed and relative air charge (load) as its normalizers.



To edit a value in the table, double-click the cell, then you can enter a value. Selected cells are blue except for the fo cussed cell which is white and outlined (typing edits the focussed c ell). You can right-click to Copy or Paste values. You can also edit table values using the table graph, see below.

See also “Filling a Table by Extrapolation” on page 3-12, and “Arithmetic Operations On Table Values” on page 3-15 for information on applying arithmetic operations to select ed cell values or whole tables.

Note You can revert table changes in the History display. Select View > History . See “Using the History Display” on page 3-20.

Locking and Unlocking Cell Values

When you are satis Þed with a region of the table, you might want to lock the cell values in that region, to ensure that those values do not change.

To lock or unlock a cell value, right-cl ick the cell and select from the menu. Locked cells have a padlock icon in the display. You can also lock an entire table using the Table menu.

Using the Graph of the Table

The table view displays both the table values and a graph of the table. This gives a useful display of the table's b ehavior. Shown is an example of a graph in dragging and rotation mode.

&)" " % " #

- In the default mode, you can rotate the graph of the table by clicking and dragging the axes.
- Select View > Edit Table Surface to alter values in the table by clicking and dragging vertically any point. In this mode, when you click a point, a

blue line indicates the selected point in the table. To return to table rotation mode without altering table values, select View > Rotate Table Surface .

Note When editing the table surface you may drag a value unintentionally - to return to previous table values, use the History display. See “Using the History Display” on page 3-20.

Filling a Table From a Model

To Þll a single table with model values, see “Filling a Table From a Model” on page 3-17.

Filling a Table by Extrapolation

Filling a table by extrapolation Þlls the table with values based on the values already placed in the extrapolation mask. Using the extrapolation mask is described below.

To Þll a table by extrapolating over a preselected mask, click or select Table > Extrapolate .

This extrapolation does one of the following:

- If the extrapolation mask has only one value, all the cell values change to the value of the cell in the mask.
- If the extrapolation mask has two or mo re collinear value s, the cell values change to create a plane parallel to the line of values in the mask.
- If the extrapolation mask has three or more coplanar values, the cell values change to create that plane.
- If the extrapolation mask has four or more ordered cells (in a grid), the extrapolation routine Þlls the cells by a grid extrapolation.
- If the extrapolation mask has four o r more unordered (scattered) cells, the extrapolation routine Þlls the cell values using a thin plate spline interpolant (a type of radial basis function).

Using the Extrapolation Mask

The extrapolation mask de Þnes a set of cells that form the basis of any extrapolation.

For example, a speed-load (or relative air charge) table has values in the following ranges that you consider to be accurate:

- Speed 3000 to 5000 rpm
- Load 0.4 to 0.6

You can deÞne an extrapolation mask to include all the cells in these ranges. You can then Þll the rest of your table based on these values.

To add or remove a cell from the extrapolation mask,

1 Right-click the table.

2 Select Add To Mask or Remove From Mask from the menu.

Cells included in the extrapolation mask are colored yellow.

Cells that are locked and in the extrapolation mask are yellow and have a padlock icon.

When using feature calibration you can a lso generate the extrapolation mask from the boundary model or from the predicted error of the model. See “Filling the Table by Extrapolation” on page 4-35.

Table Menu

All the toolbar button functions are also found in the table menu: Initialize , Fill , Extrapolate , Fill by Inversion . For information on these see “Calibrating the Tables” on page 4-29.

The Table menu contains the following other options

- Adjust Cell Values . This opens a dialog where you can specify an arithmetic operation to apply to either the whole table or only the cells currently selected. Arguments to op erations can be numeric (plus 10) or

percentages (minus 5%). You can set the selected cells to a value or to the mean. You can also apply user-de Þned functions. See “Arithmetic Operations On Table Values” on page 3-15. This function is also in the table context menu.

- Extrapolation Mask

  The following items are also in the table context menu:

  - Add Selection — Adds selected cells to the extrapolation mask.
  - Remove Selection — Removes selected cells from the extrapolation mask.
  - Clear Mask — This ensures that none of the cells are in the extrapolation mask.
  - Generate From PE — Generate extrapolation mask depending on the value of prediction error (PE). On ly available for tables in feature calibration, as you must have a model to calculate PE. A dialog opens where you can specify the threshold value of PE below which you want to include cells in the mask. The dia log contains information about the range and mean of prediction error for the model to help you select a threshold.
  - Generate From Boundary Model — Generate extrapolation mask to include only cells within the boundary model. Only available for tables in feature calibration, as you must have a boundary model.

- Extrapolate — Extrapolates values from the cells in the extrapolation mask to Þll the whole table. Also in the toolbar.
- Table Cell Locks The following items are also in the table context menu:
  - Lock Selection — Locks the selected cells and a padlock icon appears..
  - Unlock Selection — Unlocks the selected cells.
  - Lock Entire Table — Locks every cell in the current table.
  - Clear All Locks — Unlocks all cells in the table.
- Convert to Model . This option converts a table directly to a model.
- Properties . This opens the Table Properties dialog where you can set the precision type of the table data. You can also reach this from the Calibration Manager. See “Table Properties” on page 3-30.

Arithmetic Operations On Table Values

The Table menu item Adjust Cell Values (also a right-click context menu item) opens a dialog where you can specify an arithmetic operation to apply to either the whole table or only the cells currently selected. Arguments to operations can be numeric (plus 10) or percentages (minus 5%). You can set the selected cells to a value or to the mean. You can also apply user-defined functions.

1. Right-click the table or select Table > Adjust Cell Values. The Adjust Cell Values dialog box appears.
2. Select the operation to apply from the list - plus, minus, times, divide, set to value, set to mean, or custom operation. Use the custom operation to specify your own function in an M-file.
3. Use the Value edit box to enter an argument. All operators accept a numeric argument (e.g. operator = plus, value = 10). You can also enter a percentage for the operators plus, minus, and set to value (e.g. 'minus' '1%).
4. Select the radio buttons to apply the operation to either the whole table or only the cells currently selected, and click OK.

You can use the custom operation option to apply user-defined functions.

The custom function is called in this way:

```
newvalues = customfcn( currentvalue, selectedregion )
```

Where currentvalue is the matrix of table values and selectedregion is a logical matrix the same size as the table, that is "true" where a cell is selected by the user, and false otherwise.

The newvalues matrix should be the same size as currentvalue, and these numbers are put straight into the table.

EXAMPLES:

```
function table = addOne( table, region )
table(region) = table(region) + 1;
return;
```

```
function table = randomtable( table, region )
table( region ) = rand( nnz( region ), 1 );

function table = saturate( table, region )
maxValueAllowed = 150;
table( region & table>maxValueAllowed ) = maxValueAllowed;
minValueAllowed = 100;
table( region & table<minValueAllowed ) = minValueAllowed ;
return
```

As an illustration, to use the saturate example:

1 Save the function text in an M-file named saturate.m .

2 Click and drag to select a region of cells in a CAGE table.

3 Right-click and select Adjust Cell Values .

4 In the dialog:

- Select custom operation from the Operation list
- Enter saturate in the Value edit box (the first function of that name found on the MATLAB path will be used), or click the browse button to locate the M-file.
- Select the radio button to Apply to selected table cells , and click OK .

The selected table cells are saturated between the ranges specified in the function M-file (between 100-150).

Filling a Table From a Model

To Þll a table with values from a model ev aluated at the table breakpoints:

1 Open the Table Fill Wizard.

- With the table selected in the Tables view, select Table > Fill .
- Alternatively, click the Fill toolbar button ( ).

The Table Fill Wizard opens.

2 Select a model by clicking the Select Model button.

The Select Model dialog box opens.

Select the model you want to fill the table with, and click OK. You return to the Table Fill Wizard.

Note The subsequent screens of the Table Fill Wizard are identical to steps 2–4 of the Feature Fill Wizard. See “Feature Fill Wizard” on page 4-40 for details. This following procedure describes small differences in usage for the Table Fill Wizard.

---

3 (Optional) Change constraint, create links, or both. For table fill, you cannot use gradient constraints.

Click Next.

4 (Optional) Change variable values from the defaults. By default the table's normalizer breakpoints and the set points of other variables are selected, so the number of grid points equals the number of table cells.

Click Next.

5 Click the Fill Tables button. The Progress graph shows the change in RMSE as the optimization progresses.

- Smoothing does not affect table Þlling (no gradient constraints with Table Fill), and the surface pl ot check boxes are not enabled.
- The table Bounds (speciÞed in the Table Properties) constrain table values. The toolbox clears any previous extrapolation mask and automatically extrapolates the ne w table. When using the Feature Fill Wizard, you can control these options on the Þrst screen of the wizard. With the Table Fill Wizard, you cannot control these options as you start on screen 2.

Click Finish to exit the wizard.

After you exit the wizard, the plots with selected check boxes appear. You can then view your Þlled table values and surface plot in the Tables view of the CAGE Browser.

Using the History Display

In this section...
“Introducing the History Display” on page 3-20
“Resetting to Previous Versions” on page 3-21
“Comparing Versions” on page 3-23

Introducing the History Display

The History display enables you to view the history of any table or normalizer in a CAGE session.

The History display lets you

- Revert to previous versions of tables and normalizers (See “Resetting to Previous Versions” on page 3-21.)
- Compare different versions of tabl es and normalizers (See “Comparing Versions” on page 3-23.)

You can view the History display of a table or normalizer by selecting View > History .

The upper pane of the History display lis ts all the versions of the highlighted object.

The lower pane displays the normalizer or table of the highlighted version.

Resetting to Previous Versions

To reset the normalizer or table to a previous version, select View > History to open the History display.

1 Highlight the previous version that you want to revert to.

2 Click Reset .

3 Click Close to see the updated table view.

Note Tables are independent of normalizers, so if you reset a table to a previous version you must also reset the normalizers to that version (if they have changed).

To remove previous versions of the object or comments,

1 Highlight the version that you want to remove.

2 Click Remove .

Adding and Editing Comments About Versions

To add comments,

1 Click Add .

2 In the dialog box enter your comment.

3 Click OK . A new History set point is added when you add a comment.

To edit comments,

1 Select the comment that you want to edit.

2 Click Edit comment .

3 In the dialog box, edit the comment.

4 Click OK .

Comparing Versions

To compare two different versions of a no rmalizer or table, highlight the two versions using Ctrl +click. Note the following:

- The lower pane shows the difference between the later and the earlier versions.
- Cells that have no entries have no difference.
- Cells that have red entries have a higher value in the later version.
- Cells that have blue entries have a lower value in the earlier version.

Calibration Manager

In this section...

“Introducing the Calibra tion Manager” on page 3-24

“Setting Up Tables from a Cal ibration File” on page 3-24

“Setting Up Tables Manually” on page 3-28

“Copying Table Data from Other Sources” on page 3-28

Introducing the Calibration Manager

To change the size of tables in CAGE, you use the Calibration Manager dialog box. Open this tool by selecting Tools > Calibration Manager or by clicking the Calibration Manager button on the toolbar.

You can either set up your tables manually or from a calibration Þle. You can also copy table data from other sources.

You can enter the required inputs, number of rows and columns and an initial value for table cells when you add a new table. Use the File > New menu items to make new tables. See “Adding, Duplicating and Deleting Tables” on page 3-3. You can use the Calibration Manager to change the sizes, values and precision of tables.

Setting Up Tables from a Calibration File

Setting up tables with a calibration Þle involves two steps:

- “Importing Calibration Files into th e Calibration Manager” on page 3-24
- “Importing Calibration File Values into a Table” on page 3-25

Importing Calibration Files i nto the Calibration Manager

You can import calibration Þles from the CAGE Browser by selecting Select File > Import > Calibration > File or ATI Vision . Your selected Þle opens in the Calibration Manager.

You can also open the Calibration Manager and import calibration Þles from within the Calibration Manager window by using the following procedure:

1 In the Calibration Manager, open the Þle by clicking the Open Calibration File button in the toolbar.

The Import Cali bration Data dialog box opens.

2 Select whether you want to import from File or from ATI Vision.

- If importing from ATI Vision, use the Connection Manager dialog box to select the required calibration. See “Importing and Exporting Calibrations” on page 3-53 for instructions.
- If importing from Þle, browse to the calibration Þle, select it, and click Open .

In the Calibration Ma nager, review the Þles you have imported:

- Your imported calibration Þle items appear in the Calibration File Contents pane at the top right.
- The tables, normalizers, and other items in your project appear on the left in the Project Calibration Items pane.
- The values of the currently selected item appear in the lower pane, so you can inspect the value s in your calibration Þle and current project tables. Because the import process Þlters out empty data, any empty variables will not appear.

Note You can Þnd an example calibration Þle, tutorialcal.mat , in the mbctraining folder.

Importing Calibration File Values into a Table

To import the data in your calibration Þle into a table in your project:

1 Click to select both the table in the Calibration File Contents pane and the table in the Project Calibration Items pane with which you want to associate it.

2 Associate these two items by clicking the button (with tooltip “Set Up From Selected Calibration File Item”). The values in the calibration table load into the table. You can inspect the values in the lower pane by clicking to select the table in the Project Calibration Items pane.

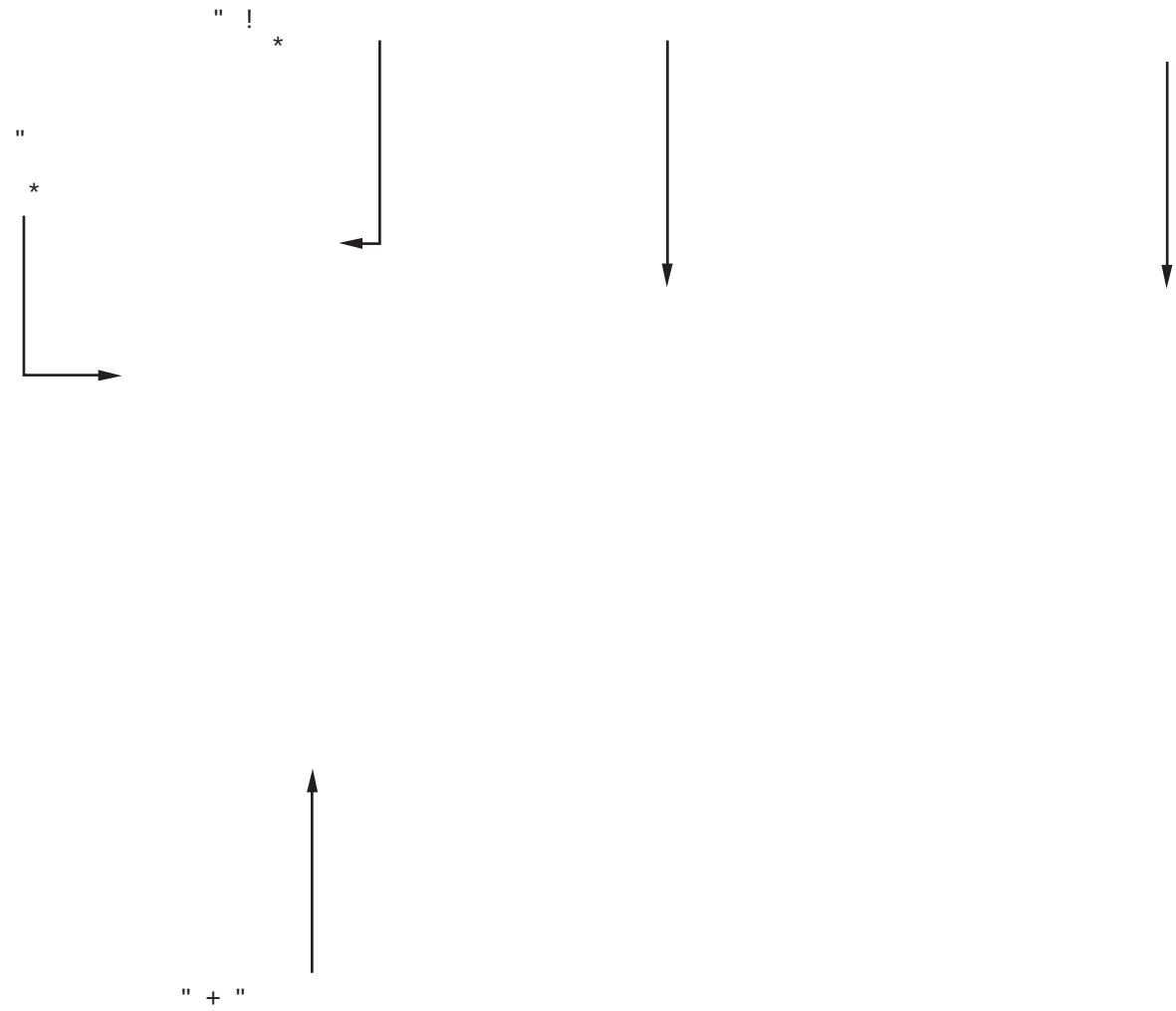
   To associate all the items listed in the Project Calibration Items pane with items that have the same names listed in the Calibration File Contents pane, click the Auto button (with tooltip “Set Up All Matching Calibration File Items”).

   To find particular names in a large calibration table, you can click the Calibration File Contents list and type the first few letters of the item that you want to find. The cursor moves to the letters specified as you type.

3 Check the display of your table, and then click Close.

When you close the Calibration Manager you can view your updated tables in CAGE. If you want to compare or revert to an earlier version of your table, select View > History. See “Using the History Display” on page 3-20.

The following figure shows the Calibration Manager.



Note You can add additional Þle formats to con Þgure CAGE to work with your processes.

Contact The MathWorks for details about adding Þle formats at http:/www.mathworks.com/products/mbc/ .

Setting Up Table s Manually

1 Select the norma lizer or table to set up from the list on the left.

2 Enter the number of rows and columns in the edit boxes on the left and select initial values for each cell in the table.

3 Click Apply .

Note When initializing tables for a feature calibration (comparing a model to a strategy) you should think about your strategy. CAGE cannot Þll those tables if you try to divide by zero. Modi Þer tables should be initialized with a value of 1 for all cells if they are multipliers, and a value of 0 if they are to be added to other tables. See “Initializing Table Values” on page 4-30.

4 Check the display of your table, then click Close .

Copying Table Data from Other Sources

You can paste table values from other app lications, such as Excel, by copying the array in the other application and clicking Paste in the Calibration Manager:

1 Open the desired Þle and copy the array that you want to import.

2 In the Calib ration Manager dialog box, click Paste .

You can also set up a table from a text Þle:

1 Click Set Up From ASCII File in the toolbar.

2 Select the desired Þle, then click Open .

Note If the size of the table is different from the Þle that you are copying, CAGE changes the size of the table in the session.

Table Properties

In this section...

"Opening the Table Properti es Dialog Box" on page 3-30

"Table Properties: General Tab" on page 3-30

"Table Properties: Table Values Precision Tab" on page 3-30

"Table Properties: Inputs Tab" on page 3-36

Opening the Table Properties Dialog Box

In the Tables view, to reach the Table Properties dialog,

- Right-click a table node and select Properties .
- Select a table, then select Table > Properties

Table Properties: General Tab

The selected table name, type and number of inputs are displayed.

Use the Table value limits edit boxes to set a range of values restricting the values in the table.

When you are done, click OK .

Table Properties: Table Values Precision Tab

The Table Values Precision tab contains the same settings as the Edit Precision dialog box (reached by clicking the Edit Precision button in the Calibration Manager dialog box).

These settings allows you to edit the precision of the number in selected tables and normalizers according to the way tabl es are implemented in the electronic control unit (ECU). The ECU designer chooses the type of precision for each element to make best use of available memory or processor power.

To edit the precision of a table or normalizer,

1 Clear the Read-only check box to make the precision writable.

2 Select the Precision type you require for the table:

- Floating Point (See “Floating-Point Precision” on page 3-31.)
- Polynomial Ratio, Fixed Point (See “Polynomial Ratio, Fixed Point” on page 3-32.)
- Lookup Table, Fixed Point (See “Lookup Table, Fixed Point” on page 3-35.)

Floating-Point Precision

The advantage of using ßoating-point precision is the large range of numbers that you can use, but that makes the computation slower.

There are three types of ßoating-point precision that you can choose from:

- IEEE double precision (64 bit)
- IEEE single precision (32 bit)
- Custom precision

If you choose Custom precision , you must specify the following:

- Number of mantissa bits
- Number of exponent bits

See Also.

- For more information on IEEE double precision in MATLAB, see Moler, C., “Floating points,” The MathWorks Company Newsletter , 1996.

Polynomial Ratio, Fixed Point

The advantage of using fixed-point precision is the reduction in computation needed for such numbers. However, it restricts the numbers available to the user.

For example, the polynomial ratio is of the form (see the ratio shown)

To edit the polynomial ratio,

1 Select the Numerator Coefficients edit box and enter the coefficients. In the preceding example, enter 50 0.

The number of coefficients determines the order of the polynomial, and the coefficients are ordered from greatest to least.

2 Select the Denominator Coefficients edit box and enter the coefficients. In the preceding example, enter 0 255.

3 To edit the size of the precision, choose from

- BYTE (8 bits)
- WORD (16 bits)
- LONG (32 bits)
- CUSTOM (Enter the number of bits in the edit box)

4 Select the Signed check box if you want the numbers to be negative and positive.

Lookup Table, Fixed Point

The advantage of using fixed-point precision is the reduction in computation needed for such numbers. However, it restricts the numbers available to the user.

For example, consider using a lookup table for the physical quantity spark advance for maximum brake torque (MBT spark) . Typically, the range of values of MBT spark is 0 to 50 degrees. This is the physical data. The ECU can only store bytes of information and you want to restrict the hardware store to a range of 0 to 8, with at most one decimal place stored.

To adjust the fixed-point precision of the lookup table:

1 Select the Physical Data edit box and enter the range of the physical data.

2 Select the Hardware Data and enter the range to store.

3 To edit the size of the precision, choose from

- BYTE (8 bits)
- WORD (16 bits)
- LONG (32 bits)
- CUSTOM (Enter the number of bits in the edit box)

4 Select the Signed check box if you want the numbers to be negative and positive.

In the example shown, the hardware is restricted to 8 bytes and to one decimal place.

Table Properties: Inputs Tab

This tab displays the inputs and variable dependencies for the selected table.

About Normalizers

What are normalizers? A normalizer is the axis of your lookup table. It is the same as the collection of the breakpoints in your table.

For information on using the controls, see “Normalizer View” on page 3-39

CAGE distinguishes between the normalizers and the tables that they belong to. Using models to calibrate lookup tables enables you to perform analysis of the models to determine where to place the breakpoints in a normalizer. This is a very powerful analytical process.

Note For information on optimizing breakpoints with reference to a model (in feature calibration), see “Calibrating the Normalizers” on page 4-16.

It is important to stress that in CAGE a lookup table can be either one-dimensional or two dimensional. One-dimensional tables are sometimes known as characteristic lines or functions. Two-dimensional tables are also known as characteristic maps or tables. This is important because normalizers are very similar to characteristic lines.

For example, a simple strategy to calibrate the behavior of torque in an engine might have a two-dimensional table in speed and relative air charge (a measure of the load). Additionally, this strategy might take into account the factors of air/fuel ratio (AFR) and spark angle. Each of these compensating factors is accounted for by the use of a simple characteristic line. In CAGE, these characteristic lines are one-dimensional tables. In the example strategy, there are the following tables and normalizers:

- One characteristic map: the torque table
- Six characteristic lines:
  - Two tables: one for AFR and one for spark angle
  - Four normalizer functions: speed, load, AFR, and spark angle

Notice also that a breakpoint is a point on the normalizer where you set values for the lookup table.

Thus, when you calibrate a normalizer you place the individual breakpoints over the range of the table's axis.

Normalizer View

In this section...
“Introducing the Normalizer View” on page 3-39
“Editing Breakpoints” on page 3-41
“Input/Output Display” on page 3-42
“Normalizer Display” on page 3-42
“Breakpoint Spacing Display” on page 3-43

Introducing the Normalizer View

The normalizer node shows the Normalizer view, which displays

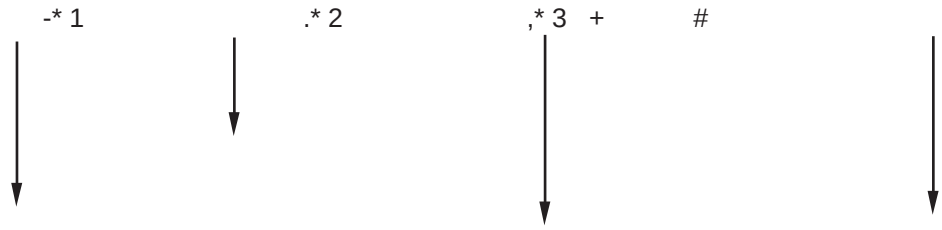
- One normalizer if the table selected is one-dimensional
- Both normalizers if the table is two-dimensional

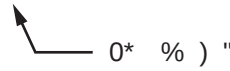
Note If the table has two normalizers, both are displayed, the normalizer for the table columns at the top, the normalizer for the table rows below. This is true whichever normalizer on the tree is highlighted.

See “Editing Breakpoints” on page 3-41.

The parts of the display as shown in the example below are:

- “Input/Output Display” on page 3-42. This shows the breakpoints of the normalizer.
- “Normalizer Display” on page 3-42. Th is is a graphical representation of the Input Output display.
- “Breakpoint Spacing Display” on page 3-43. This shows a slice of the model (in feature calibration) over the range of the breakpoints.
- The comparison pane (for feature calibr ation with reference to a model). For information, see “Viewing the Normalizer Comparison Pane” on page 4-26.


-* 1
.* 2
,* 3 +
#


0* %) "

Editing Breakpoints

To edit breakpoints:

- Double-click on a cell in the Input or Output column and edit the value.
- Click and drag a breakpoint in the Normalizer Display graph or the Breakpoint Spacing display.

To view the history of the normalizer function, select View > History from the menu. This opens the History dialog box where you can view and revert to previous versions. For a more detailed description of the History dialog box, see “Using the History Display” on page 3-20.

Locking and Unlocking Breakpoints

Locking breakpoints ensures that the locked breakpoint does not alter. You might want to lock a breakpoint when you are satisfied that it has the correct value.

To lock a breakpoint, do one of the following:

- Right-click the selected breakpoint in the Input/Output display and select Lock. Locked breakpoint cells have padlock icons.
- Right-click the selected breakpoint in the Normalizer Display or Breakpoint Spacing display and select Lock Breakpoint. Locked breakpoints are black.

Similarly use the right-click context menus to unlock breakpoints.

Deleting Breakpoints

Deleting breakpoints removes them from the normalizer table. There are still table values for the deleted breakpoints: CAGE determines the positions of the deleted breakpoints by spacing them linearly by interpolation between the nondeleted breakpoints.

Deleting breakpoints frees ECU memory. For example, a speed normalizer runs from 500 to 5500 rpm. Six breakpoints are spaced evenly over the range of speed, that is, at 500, 1500, 2500, 3500, 4500, and 5500 rpm. If you delete all the breakpoints except the endpoints, 500 and 5500 rpm, you reduce the

amount stored in the ECU memory. The ECU calculates where to place the breakpoints by linearly spacing the breakpoints between the 500 rpm breakpoint and the 5500 rpm breakpoint.

To delete a breakpoint, right-click the breakpoint and select Delete Breakpoint.

Deleted breakpoints are green in the Breakpoint Spacing display. You can restore them by right-clicking and selecting Add Breakpoint.

Input/Output Display

The table consists of the breakpoints of the normalizer function.

The table has inputs and outputs:

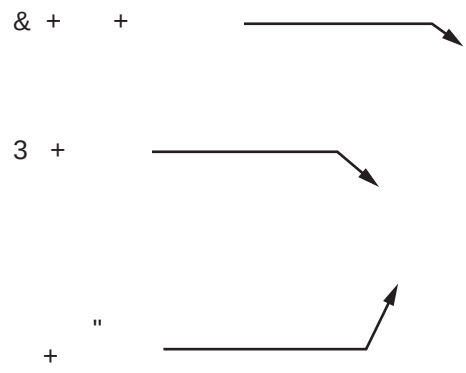
- The inputs are the values of the breakpoints.
- The outputs refer to the row/column indices of the attached table.

To change values of the normalizers in the Input Output display, double-click a cell in the Input column and change its value.

Normalizer Display

This displays the values of the breakpoints plotted against the marker numbers of the table (that is, the inputs against the outputs).

Click and drag the breakpoints to move them.

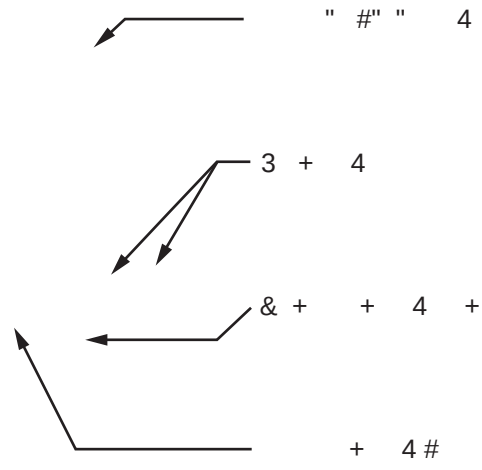


Breakpoint Spacing Display

The Breakpoint Spacing display shows

- A slice through the model in blue (when feature calibrating with reference to a model)
- The breakpoints in red

To move breakpoints, click and drag.



Show the Model's Curvature

You might want to view the curvature of the model to manually move breakpoints to where the model's curvature is greatest.

To display the model slice as its second-order derivative, the curvature of the model,

- Right-click the model in the Breakpoint Spacing display and select Display > Model Curvature ..

You can revert to displaying the model by selecting Display > Model from the right-click menu.

Multiple Slice View

By default the Breakpoint Spacing display shows one slice through the model, shown.

Viewing many slices of the model gives a b etter impression of the curvature of the model. For example, see the following Þgure.

To view multiple slices through the model,

- Right-click the model slice in the Breakpoint Spacing display and select Number of Lines and choose the number of slices that you want to view from the list.

Inverting a Table

In this section...

“Overview of Inverting Tables” on page 3-46

“Inverting One-Dimensional Tables” on page 3-48

“Inverting Two-Dimensional Tables” on page 3-50

Overview of Inverting Tables

You can use CAGE to produce a table that is the inverse of another table. This involves swapping a table input with a table output, and you can invert 1-D or 2-D tables.

Inverting a table allows you to link a forward strategy to a backward strategy; that is, swapping inputs and outputs. This process is desirable when you have a "forward" strategy, for example predicting torque as a function of speed and load, and you want to reverse this relationship in a "backward strategy" to find out what value of load would give a particular torque at a certain speed.

Normally you fill tables in CAGE by comparing with data or models. Ideally you want to fill using the correct strategy, but that might not be possible to find or measure. If you only have a forward strategy but want a backward one, you can fill using the forward strategy (tables or model) and then invert the table.

For example, to fill a table normally from a model, you need the model response to be the table output, and the model inputs to be a function of the table inputs (or it should be possible to derive the input – for example, air

mass from manifold pressure). If the ava ilable model is “inverted“(the model response is a table input and the table output is a model input) and you cannot change the model, you can invert the table in CAGE.

In the diagram of a table shown, the x- and y-axes represent the normalizers (which you want to be spark and load) and the z-axis is the output at each breakpoint (torque). To Þll this table correctly from the model is a two-step process. First you need to Þll a table that has the same input and output as the model, and then Þll a second table by inversion.

For the inversion to be deterministic and accurate, the table to be inverted must be monotonic; that is, always incre asing or decreasing. This requirement is explained by the following one-dimensional example. Every point on the y-axis must correspond to a unique point on the x-axis. The same problem applies also to two-dimensional t ables: for any given output in the Þrst table there must be a unique input condition; that is, every point on the z-axis should correspond to a unique point in the x-y plane. Some table inversions have multiple values and so do not meet this requirement, just as the square root function can take either positive or negative values. You can use the inversion wizard in CAGE to handle t his problem; you can control the inversion process and determine what to do in these cases.

The following example illustrates a tabl e with multiple values. There are two solutions for a single value of torque. CAGE has a table inversion wizard that can help overcome this problem. You can specify whether you want to use the upper or lower values for Þlling certain parts of the table; this allows you to successfully invert a multiple-valued fu nction. See the inversion instructions for 1-D and 2-D tables in the next sections.

The process of inverting a one-dimensio nal table is different from the process of inverting a two- dimensional table.

Inverting One-Dimensional Tables

To invert a one-dimensional table,

1 Ensure that your session contains two tables:

   a The Þrst table from your forward strategy, Þlled

   b The second table from your backwa rd strategy, which you want to Þll

2 Highlight the second table.

3 Click or select Table > Fill by Inversion .

   The lower pane now acts as a wizard.

4 In the lower pane, highlight the table that you want to invert. Click Next .

5 The next page asks what CAGE should do if it encounters multiple values. The options are

   • Minimum selects the lower of the two if a given number has two possible inverses (like selecting the negative square root of a number).

- Maximum selects the uppermost range if a given number has two possible inverses (like selecting the positive square root of a number).
- Intermediate selects the middle range if a given number has more than two possible inverses.
- Automatic selects the range that produces the least error (see below; the last page of the wizard plots the error metric).

For example, the function $y = x^2$ is impossible to invert over the range -1 to 1. You can specify to invert the range from 0 to 1, sacrificing the inversion in the lower range, or the reverse. To select the range from 0 to 1, highlight Maximum.

The display shows a comparison between the table (green) and the function $x = f^{-1}(f(x))$.

Choose one of these options, then click Next.

6 The last page of the wizard has a comparison plot that shows how successful the inversion has been. If your forward function is $y = f(x)$, and your inverse function is $x = g(y)$, then, combining these, in an ideal world, you should have $x = g(f(x))$. The plot then displays a red line showing x against x and a green line showing x against $g(f(x))$. The closeness of these two lines indicates how good the inversion has been: a perfect inverse would show the lines exactly on top of each other.

In the following example, the lines are together and then diverge; this plot can show you which part of your table has not successfully inverted and where you should try a different routine.

Note The automatic inversion routine tries to minimize the total distance between these lines. This can sometimes lead to unexpected results. For example, given the function f(x) = x^2 between -1 and 1, if you select either positive or negative square root as the inverse, this induces a large error in the combined inverse. If you choose g(y) = sqrt(y), then g(f(-1)) = 1, an error of 2. To minimize this, the automatic routine might choose to send everything to zero and accept a mediu m error over the whole range rather than a large error over half the range. The more knowledge you have of the form of the "forward" table, the more you can make an informed choice about which routine to select.

7 Click Finish to accept the inversion or Cancel to ignore the result and return to the original table.

Inverting Two-Dimensional Tables

To invert a two-dimensional table,

1 Ensure that your session contains two tables:

a The Þrst table from your forward strategy, Þlled

b The second table from your backwa rd strategy, which you want to Þll

2 Highlight the second table.

3 Click or select Table > Fill by Inversion .

The lower pane now acts as a wizard.

4 In the lower pane , highlight the table that you want to invert and click Next .

5 Identify the corresponding signals.

The forward table and backward table share a common input. This page of the wizard lists all possible combinations of inputs into the forward and backward tables and asks you to highlight the combination that gives the two common inputs. To illustrate this, if the forward table gives torque in terms of the variables engine speed and load, whereas you want the backward table to give load in terms of RPM and Tq, then the list would read

- RPM and engine speed
- RPM and load
- Tq and engine speed
- Tq and load

In this case, you would select the Þrst option.

Highlight the part of the table to invert, then click Next .

6 CAGE asks what to do if it encounters multiple values. The choices are

- Maximum selects the uppermost range (l ike choosing a positive square root of a number).
- Minimum selects the lower value if there are two choices (like choosing a negative square root of a number).
- Intermediate selects the middle range when there are more than two choices.
- Automatic selects the range that produces the least error. CAGE tries to choose values to put in the inverse table that minimize an

error metric similar to the error metric for 1-D tables (see “Inverting One-Dimensional Tables” on page 3-48).

Choose one of these options and click Next .

7 The last page of the wizard has a comparison plot that shows how successful the inversion has been. If the forward f unction is z = f(x,y), and the inverse function is x = g(y,z), then, combining these, in an ideal world you should have x = g(y,f(x,y)). The plot then dis plays a plane showing x plotted against x and y, and a colored surface showing g (y,f(x,y)) plotted against x and y. The closeness of these two planes indicates how good the inversion is.

Following is an example. In this case, the forward table is a quadratic (z = y^2); the backward table is inverted using the positive square root of z (maximum range). As you can see, thi s leads to large errors at negative values of y, but good inversion for positive values of y.

Click Finish to accept the result or Cancel to ignore the result and return to the original table.

Importing and Exporting Calibrations

In this section...

“Formats” on page 3-53
“Importing Calibrations” on page 3-53
“Exporting Calibrations” on page 3-55

Formats

You can import and export calibrations in various formats.

- You can import the following File formats:
  - Simple M Þle
  - Simple MAT Þle
  - ATI Vision MAT Þle
  - ETAS INCA DCM Þle (version 1)
- Or directly to/from ATI Vision (supported versions: 3.5, 3.4 and 2006 SP2.1).

  To use Vision, a license is required for the "Horizon Scripting/Remote API Toolkit".

Note Note to use the Vision interface you must Þrst enter mbcconfig -visioninterface at the command line.

You can export all the same formats, plus the simple CSV Þle format.

Importing Calibrations

1 Select File > Import > Calibration > File or ATI Vision .

Similarly, from the Calibration Manage r, if you click Open Calibration File in the toolbar, you can select File or ATI Vision in the dialog and proceed to import in the same way.

2 If importing a Þle, a Þle browser dialog opens.

a Select the type of Þle you want from the Files of type drop-down list, or leave the default All files (*.*) and CAGE will try to load the Þle based on the Þle extension.

b Browse to the Þle and click Open to import.

If importing from .dcm , .mat or .m Þle, the Calibration Manager opens. See “Setting Up Tables from a Calibration File” on page 3-24 for instructions.

If importing from ATI Vision, the ATI Vision Connection Manager dialog appears.

a The Computer Þeld is optional. Leave this Þeld blank if you are using Vision on the local machine. If you want to connect to a remote machine, you can enter a machine name or an IP address.

b Click Connect .

If Vision is already running on the machine that you try to connect to, MATLAB connects to Vision. If Vision is not running then it is

launched, typically with no projec t loaded and with the application window invisible.

c If there is a project (.prj Þle) currently loaded in Vision it appears in the Project Þeld. If this Þeld is blank then there is no project loaded. Type a project Þle name to load that project. Note that the project Þle path is relative to the machine on which Vision is running.

d Select the appropriate Vision Device , Strategy and Calibration within your project, and click OK to import.

The Calibration Manager opens . See “Setting Up Tables from a Calibration File” on page 3-24 for instructions.

Exporting Calibrations

1 Select File > Export > Calibration > Selected Item or All Items .

2 The Export Calibration dialog appears. Select the format you want to export to:

- ATI Vision
- ATI Vision MAT Þle
- INCA DCM Þle
- Simple CSV Þle
- Simple MAT Þle
- Simple M Þle

Click OK .

3 If you select ATI Vision , the ATI Vision Connection Manager dialog appears, as for importing calibrations.

If you select a Þle format, a Þle browser appears. Choose a location and Þlename and click Save .

If you choose All Items , all tables, normalizers, curves and constants in the project are exported.

What you export when you choose Selected Item depends on which node is highlighted:

- Selecting a Normalizer node outputs the values of the normalizer.
- Selecting a Table node outputs the values of the table and its normalizers.
- Selecting a Feature or Tradeoff node outputs the whole feature or tradeoff (all tables, normalizers, curves and constants).

When exporting to an existing calibration file, the exported items replace the existing items. (There is no merging of existing items and new exported items.)

When exporting to Vision, the items in the CAGE project are matched by name with the items in the Vision calibration and the values are replaced. It is not possible to add new items to a Vision project by exporting from CAGE.

Feature Calibrations

This section includes the following topics:

- “Performing Feature Calibrations” on page 4-2
- “Setting Up a Feature Calibration” on page 4-5
- “Calibrating the Normalizers” on page 4-16
- “Calibrating the Tables” on page 4-29
- “Calibrating the Feature Node” on page 4-38

Performing Feature Calibrations

A 'feature' calibration is the process of calibrating lookup tables and their normalizers by comparing an ECU strategy (represented by a Simulink diagram) to a model.

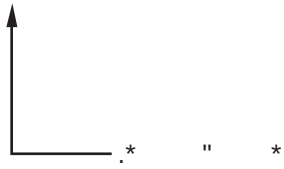
The strategy is an algebraic collection of lookup tables. It is used to estimate signals in the engine that cannot be measured and that are important for engine control.

CAGE calibrates an electronic control unit (ECU) subsystem by directly comparing it with a plant model of the same feature.

There are advantages to feature calibration compared with simply calibrating using experimental data. Data is noisy (that is, there is measurement error) and this can be smoothed by modeling; also models can make predictions for areas where you have no data. This means you can calibrate more accurately while reducing the time and effort required for gathering experimental data.

The basic procedure for performing feature calibrations is as follows:

1. Set up the variables and constants. (See "Setting Up Variable Items" on page 2-2.)
2. Set up the model or models. (See "Setting Up Models" on page 2-10.)



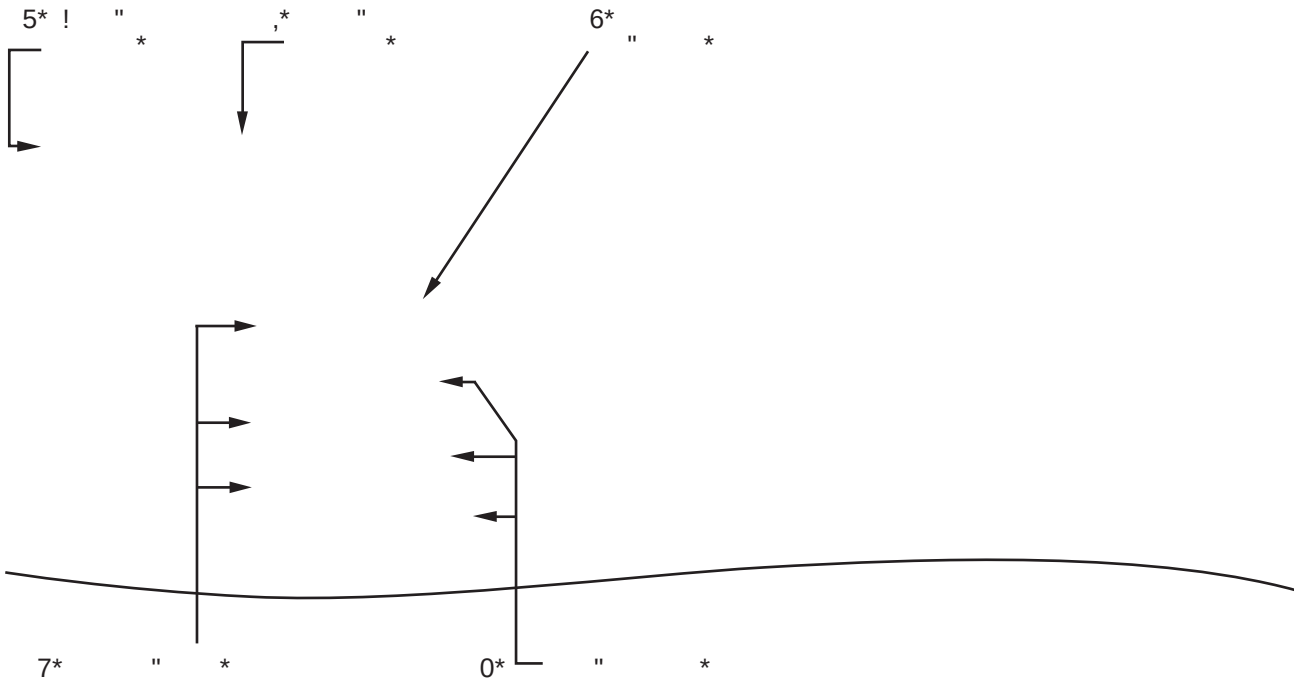
3 Set up the feature calibration. (See “Setting Up a Feature Calibration” on page 4-5.)

4 Calibrate the normalizers. (See “Calibrating the Normalizers” on page 4-16.)

5 Calibrate the tables. (See “Calibrating the Tables” on page 4-29.)

6 Calibrate and view the entire feature. (See “Calibrating the Feature Node” on page 4-38.)

7 Export the normalizers, tables, and features. (See “Importing and Exporting Calibrations” on page 3-53.)



The normalizers, tables, and features form a hierarchy of nodes, each with its own view and toolbar. Th e feature view is shown.

Setting Up a Feature Calibration

In this section...

“Procedure Overview” on page 4-5

“Adding a Feature” on page 4-7

“Assigning a Model” on page 4-7

“Setting Up Your Strategy” on page 4-7

Procedure Overview

A feature calibration is the process of calibrating lookup tables and their normalizers by comparing a collect ion of lookup tables to a model. The collection of lookup tables is determined by a strategy.

A feature refers to the object that con tains the model and the collection of lookup tables. For example, a simple fea ture for calibrating the lookup tables for the maximum brake to rque (MBT) consists of

- A model of MBT
- A strategy that adds the two following tables:
  - A speed (N), load (L) table
  - A table to account for the behavior of the air/fuel ratio (A)

Having already set up your variable items and models, you can follow the procedure below to set up your feature calibration:

1 Add a feature. This is described in the next section, “Adding a Feature” on page 4-7.

2 Assign a model. This is described i n “Assigning a Model” on page 4-7.

3 Set up your st rategy. This is described in “ Setting Up Your Strategy” on page 4-7.

4 Set up the tables. This is described in “Setting Up Tables” on page 3-2.

This section describes steps 1, 2, and 3 in turn.

When you have completed these four steps, you are ready to calibrate the normalizers, tables, and features.

Adding a Feature

A feature consis ts of a model and a collection of lookup tables, organized in a strategy.

To add a feature t o your session, select File -> New -> Feature . This automatically switches you to the Feature view and adds an empty feature to your session.

An incomplete f eature is a feature that does not contain both an assigned model and a stra tegy. If a feature is incomplete, it is displayed as in the tree display. I f a feature is complete, it is displayed as in the tree display.

Assigning a Mo del

Having alread y added a feature and a model to your session, you can assign a model to your f eature.

To assign a model to your feature,

1 Highlight th e top feature node in the tree display.

2 Click Select Model to select the model you want to work with. A dialog box appears.

3 Highlight the correct model to a ssign to your feature and click OK . You will see the model name and inputs appear above the Select Model button.

Setting Up Your Strategy

A strategy is an algebraic collection of tables, and forms the structure of the feature.

For example, a simple strategy to calibrate a feature for MBT adds two tables:

- A table ranging over the variables speed and load
- A table to account for the behavior of the model as the AFR varies

To evaluate the feature side by side with the model, you need to have a strategy that takes some or all of the same variables as the model. The

strategy is expressed using Simulink diagrams. You can either import a strategy or you can construct a strategy.

The following topics are described next:

- “Importing a Strategy” on page 4-8
- “Constructing a Strategy” on page 4-9
- “Exporting Strategies” on page 4-13

Importing a Strategy

To import a Simulink strategy,

1. Highlight the top feature node in the tree display.
2. Select File > Import > Strategy.
3. Select the appropriate .mdl file. CAGE checks the strategy for more than one outport.
4. Select the outport that you want to use.

   If there is more than one outport to your strategy, a Simulink window opens. Double-click the correct blue outport to parse (or import) the strategy to your feature.

   If there is only one outport to your strategy, a dialog box opens.

   - Click Automatic to parse the strategy without viewing it.
   - Click Manual to edit the strategy. When you are finished editing double-click the blue outport circle to parse the strategy to your feature. The Simulink windows close and parse this strategy to your feature.

To view a representation of your strategy, select the Feature node. Your strategy is represented in the Strategy pane. Select View > Full Strategy Display to switch between the full description and the simplified expression. You can select and copy the strategy equation to the clipboard.

For information about using Simulink to amend strategies, see “Constructing a Strategy” on page 4-9.

Example. In the matlab\toolbox\mbc\mbctraining directory, there is a Simulink diagram called tutorial.mdl . If you import this and click Manual in the dialog box, you see the following diagram.

Double-click the Torque-Output outport to parse the strategy into the Strategy pane.

Constructing a Strategy

For you to perform a feature calibration, the strategy and the model must have some variables in common.

To construct a strategy using Simulink,

1 Highlight the correct feature by clicking the Feature node.

2 Select Feature > Graphical Strategy Editor or press Ctrl +E.

Three Simulink windows open:

- The strategy window for editing your strategy, like the following example.

- A library window, cgeqlib , with all the blocks available for building a strategy.

- A library window with all the existing blocks in your session, organized in libraries.

3 In the strategy window, build your str ategy using the blocks available in the library windows.

4 Double-click the blue outport circle to parse the strategy into the CAGE session.

Note This closes all three Simulink windows and parses your strategy into the feature.

About Feature Parsing

The library, cgeqlib contains all the blocks available for building a strategy, as described in "Constructing a Strategy" on page 4-9. Tables in the current CAGE session are shown in dark green. A black table block signifies that a new table will be created in CAGE. If you copy a CAGE table block, a new table is created and the block color changes to black to signify this.

Normalizers in the current CAGE session are shown in light blue. Normalizer blocks must be inputs to tables. A black Normalizer block signifies that a new normalizer will be created in CAGE. If you copy a CAGE table block a new normalizer is created and the block color changes to black to signify this.

Some rules for feature parsing:

- You can create 1D tables with or without normalizers. If you add a Function block with no normalizer, a 1D lookup table with an internal normalizer is created on parsing. If the Function block has a normalizer as its input, then you can use shared normalizers (from the list of available normalizers in CAGE). After creation you cannot change from using shared normalizers to internal normalizers. You must set normalizer size and values using the Calibration Manager, after parsing the strategy.
- 2D lookup tables always have shared normalizers. If the input to the Table block is not a normalizer, then a normalizer is created when the strategy is parsed.
- If you change the name of inports, table or normalizer blocks for blocks associated with existing CAGE items, then the CAGE item's name is changed. The name is unique for the current CAGE project (suffixes ("_1") are added if necessary to create a unique name).
- Prelookup tables must feed into an Interpolation block using a Prelookup block.

- Constant values are read in from the block.
- Only scalar inputs are supported with the exception of the Fcn block. The function block accepts multiple inputs as inputs to the expression (e.g., u(1)+u(2)).

If the parsing of the Simulink diagram fails, you see an error message in a dialog box, and any renamed items or changed connections in the current feature are restored. You can then co rrect the Simulink diagram and reparse the diagram.

For more information about using Sim ulink to build your strategy, see Simulink Help.

If you need to parse strategies older than Release 2008a, you can use the function cgStrategyTables to set the style for strategy parser behavior for 1D tables. Pre-R2008a, the CAGE strategy parser interpreted 1D tables as normalizers if they fed into a lookup table. In R2008a, a normalizer block was added to the cgeqlib library, and lookup table blocks are now always interpreted as lookup tables. This chan ge creates an incompatibility for any older strategies saved as mdl Þles and reimported into CAGE. If you need to parse pre-R2008a strategies , use the function as follows:

```
cgStrategyTables('Pre-R2008a')
```

This reverts the parser behavior to the pre-R200 8a interpretation of 1D lookup tables. A warning is issued when a 1D table is converted to a normalizer. You can turn the warning off as follows:

```
warning off mbc:cgslparser:ObsoleteNormalizer
```

Reset the parser style to Release 2008a and later as follows:

```
cgStrategyTables('R2008a')
```

You can query the current strategy table style by entering: Style = cgStrategyTables .

Exporting Strategies

Simulink strategies can be exported. For example, you might want to

- Include a strategy in a Simulink vehicle model
- Compile the strategy using Real-Time Workshop ® software to produce C code
- Evaluate the strategy using Simulink

To export a strategy from CAGE,

1 Highlight the Feature node that contains the strategy that you want to save.

2 Select File > Export > Strategy .

3 Assign a name for your strategy.

The strategy is saved as a Simulink model (.mdl) Þle.

Calibrating the Normalizers

In this section...

“Overview of Calibrating Normalizers” on page 4-16

“Initializing Breakpoints” on page 4-17

“Filling Breakpoints” on page 4-18

“Optimizing Breakpoints” on page 4-22

“Viewing the Normalizer Com parison Pane” on page 4-26

Overview of Calibrating Normalizers

Select a normalizer in the tree display. This displays the Normalizer view, where you can calibrate the normalizers.

This section describes how you can use CAGE to space the breakpoints over the range of the normalizers.

To space the breakpoints, either click the buttons on the toolbar or select from the following options on the Normalizer menu:

- Initialize

  This spaces the breakpoints evenly along the normalizer. For more information, see “Initializing Breakpoints” on page 4-17.

- Fill

  This spaces the breakpoints by refe rence to the model. For example, you can place more breakpoints where th e model curvature is greatest. For more information, see “Filling Breakpoints” on page 4-18.

- Optimize

  This moves the breakpoints to minimize the least square error over the range of the axis. For more information, see “Optimizing Breakpoints” on page 4-22.

The next sections describe each of these in detail.

Note Fill and Optimize are only available when you are calibrating with reference to a model, when you are performing Feature calibrations.

For more information about the Normalizer view controls, see “Normalizer View” on page 3-39.

Initializing Breakpoints

Initializing the breakpoints places the breakpoints at even intervals along the range of the variable defined for the normalizer. When you add a table and specify the inputs in the Table Setup dialog, CAGE automatically initializes the normalizers of the table by spacing the breakpoints evenly over the ranges of the selected input variables. If you have edited breakpoints you can return to even spacing by using the Initialize function.

To space the breakpoints evenly,

1 Click on the toolbar or select Normalizer > Initialize.

2 In the dialog box, enter the range of values for the normalizer.

3 Click OK.

For example, for a torque table with two normalizers of engine speed and load, you can evenly space the breakpoints of both normalizers over the range 500 rpm to 6500 rpm for speed and 0.1 to 1 for the relative air charge. To do this, in the dialog box you enter 500 6500 for the speed normalizer, N, , and 0.1 1 for the load normalizer, L.

Filling Breakpoints

Filling breakpoints spaces the breakpoints by reference to the model. For example, one method places the majority of the breakpoints where the curvature of the model is greatest. This option is only available when you are performing Feature calibrations.

For example, a model of the spark angle that produces the maximum brake torque (MBT) has the following inputs: engine speed N, relative air charge L, and air/fuel ratio A. You can space the breakpoints for engine speed and relative air charge over the range of these variables by referring to the model.

To space the breakpoints based on model curvature, perform the following steps:

1 Click or select Normalizer > Fill .

The Breakpoint Fill Options dialog box opens.

2 Choose the appropriate method to space your breakpoints, from the drop-down menu in the dialog box.

The preceding example shows ShareAveCurv . For more information about the methods for spacing the breakpoints, see “Filling Methods” on page 4-19.

3 Enter the ranges of the values for the normalizers.

The preceding example shows Range N 500 6500 , and Range L , 0.1 1 .

4 Enter the ranges of the other model variables.

CAGE spaces the breakpoints by refe rence to the model. It does this at selected points of the other model variables. The example shows 11 17 for the Range of A and 2 for the Number of points . This takes two slices through the model at A = 11 and A = 17. Each slice is a surface in N and L. That is, MBT (N, L, 11) and MBT (N, L, 17).

CAGE computes the average value of these two surfaces to give an average model $MBT_{AV}(N, L)$.

If you set Number of points to one, and specify a range, then the mean of the range is chosen as the evaluation point.

5 Click OK .

Note If any of the breakpoints is locked, each group of unlocked breakpoints is independently spaced according to the selected algorithm.

If you increase the number of slices through the model, you increase the computing time required to calculate where to place the breakpoints.

Filling Methods

This section describes in detail the methods for spacing the breakpoints of your normalizers in CAGE.

- For one-dimensional tables, the two Þll methods are
  - ReduceError
  - ShareAveCurv
- For two-dimensional tables, the two Þll methods are
  - ShareAveCurv
  - ShareCurvThenAve

ReduceError

Spacing breakpoints using ReduceError uses a greedy algorithm:

1. CAGE locks two breakpoints at the extremities of the range of values.
2. Then CAGE interpolates the function between these two breakpoints.
3. CAGE calculates the maximum error between the model and the interpolated function.
4. CAGE places a breakpoint where the error is maximum.
5. Steps 2, 3, and 4 are repeated.
6. The algorithm ends when CAGE locates all the breakpoints.

ShareAveCurv and ShareCurvThenAve

Consider calibrating the normalizers for speed, N, and relative air-charge, L, in the preceding MBT model.

In both cases, CAGE approximates the $MBT_{AV}(N, L)$ model, in this case using a fine mesh.

The breakpoints of each normalizer are calibrated in turn. In this example, these routines calibrate the normalizer in N first.

Spacing breakpoints using ShareAveCurv or ShareCurvThenAve calculates the curvature, K, of the model $MBT_{AV}(N, L)$,

as an approximation for

Both routines calculate the curvature for a number of slices of the model at various values of L. For example, the figure shown has a number of slices of a model at various values of L.

Then

- ShareAveCurv averages the curvature over the range of L, then spaces the breakpoints by placing the i^{th} breakpoint according to the following rule.
- ShareCurvThenAve places the i^{th} breakpoint according to the rule, then finds the average position of each breakpoint.

Rule for Placing Breakpoints. If j breakpoints need to be placed, the i^{th} breakpoint, N_i, is placed where the average curvature so far is

Essentially this condition spaces out the breakpoints so that an equal amount of curvature (in an appropriate metric) occurs in each breakpoint interval. The breakpoint placement is optimal in the sense that the maximum error between the lookup table estimate and the model decreases with the optimal convergence rate of $O(N^{-2})$. This compares with an order of $O(N^{-1/2})$ for equally spaced breakpoints.

The theorem for determing the position of the unequally spaced breakpoints is from the Þeld of Approximation Theory — page 46 of the following reference: de Boor, C., A Practical Guide to Splines , New York, Springer-Verlag, 1978.

Optimizing Breakpoints

Optimizing breakpoints alters the posi tion of the table normalizers so that the total square error between the model and the table is reduced.

This routine improves the Þt between your strategy and your model. The following illustration shows how the op timization of breakpoint positions can reduce the difference between the mod el and the table. The breakpoints are moved to reduce the peak error between breakpoints. In CAGE this happens in two dimensions across a table.

To see the difference between optimizing breakpoints and optimizing table values, compare with the illustratio n in “Optimizing Table Values” on page 4-31.

See “Filling Methods” on page 4-19 for details on how the optimal breakpoint spacing is calculated.

For an example of breakpoint optimization, say you have a model of the spark angle that produces the MBT (maximu m brake torque). The model has the following inputs: engine speed, N, relative air charge, L, and air/fuel ratio, A. You can optimize the breakpoints for N and L over the ranges of these variables.

To optimize the breakpoints, perform the following steps:

1 Ensure that the optimization routine works over reasonable values for the table by choosing one of these methods:

   a Select Normalizer > Initialize .

   b Select Normalizer > Fill .

2 Click on the toolbar or select Normalizer > Optimize .

This opens the following dialog box.

3 Enter the ranges for the normalizers.

The example shows 0.2 0.811 for the Range of L, and 750 6500 for N.

4 Enter the appropriate number of grid points for the optimization.

This defines a grid over which the optimization works. In the preceding example, the number of grid points is 36 for both L and N. This mesh is combined using cubic splines to approximate the model.

5 Enter ranges and numbers of points for the other model variables.

The example shows a Range of A of 14.3 and the Number of points is 1.

6 Decide whether or not to reorder deleted breakpoints, by clicking the radio button.

If you choose to reorder deleted breakpoints, the optimization process might redistribute them between other nondeleted breakpoints (if they are more useful in a different position).

For information about deleting breakpoints, see “Editing Breakpoints” on page 3-41.

7 Click OK.

CAGE calculates the table filled with the mesh at the current breakpoints. Then CAGE calculates the total square error between the table values and the mesh model.

The breakpoints are adjusted until this error is minimized, using nonlinear least squares optimization (lsqnonlin).

When optimizing the breakpoints, it is worth noting the following:

- The default range for the normalizer variable is the range of the variable.
- The default value for all other model variables is the set point of the variable.
- The default number of grid points is three times the number of breakpoints.

See Also

- Reference page for lsqnonlin

Viewing the Normalizer Comparison Pane

To view or hide the comparison pane, select View > Feature/Model Comparison. Alternatively, click , the “snapper point” at the bottom of the normalizer display panes.

The comparison pane displays a comparison between the following:

- A full factorial grid filled using these breakpoints
- The model

Note This is not a comparison between the current table values and the model. To compare the current table values and the model, see “Comparing the Strategy and the Model” on page 4-33.

To make full use of the comparison pane,

1. Adjust the ranges of the variables that are common to the model and table.
2. Adjust the values selected for any variables in the model that are not in the selected table.

The default for this is the set point of the variable, as specified in the variable dictionary. For more information, see “Using Set Points in the Variable Dictionary” on page 2-5.

3 Check the number of points at which the display is calculated.

4 Check the comparison between the table and the model.

Right-click the comparison graph to view the error display.

5 Check some of the error statistics for the comparison, and use the comparison to locate where improvements can be made.

Error Display

The comparison pane can also be used to display the error between the model and the ’generated table’ (gridfilled using these breakpoints).

To display the error, select one of the Error items from the Plot type drop-down list.

This changes the graph to display the error between the model and the table values at these breakpoints.

You can display the error data in one of the following ways:

- Error (Table Model) . This is the difference between the feature and the model.
- Squared Error . This is the error squared.
- Absolute Error . This is the absolute value of the error.
- Relative Error . This is the error as a percentage of the value of the table.
- Absolute Relative Error (%) . This is the absolute value of the relative error.

See Also

- “Comparing the Strategy and the Model” on page 4-33

This describes the comparison made when a table node is selected in the tree display.

Calibrating the Tables

In this section...

"Overview of Calibrating Feature Tables" on page 4-29

"Initializing Table Values" on page 4-30

"Filling Table Values" on page 4-31

"Comparing the Strategy and the Model" on page 4-33

"Filling the Table by Extrapolation" on page 4-35

Overview of Calibrating Feature Tables

After you set up your session and your t ables, you can calibrate your tables.

Highlight a table in the tree disp lay to see the Table view. For more information about the Table view, s ee "Editing Tables" on page 3-6.

In CAGE, a table is de Þned to be either a one-dimensional or a two-dimensional lookup table. One-di mensional tables are sometimes known as characteristic lines or functions. T wo-dimensional tables are also known as characteristic maps or tables.

Each lookup table has either one or two axes associated with it. These axes are normalizers. See "About Normalizers" on page 3-37 for more information.

For example, a simple MBT feature has two tables:

- A two-dimensional table with speed and relative air charge as its normalizer inputs
- A one-dimensional table wit h AFR as its normalizer input

Before you can calibrate your tables, you must calibrate your normalizers. For information, see "Calibrati ng the Normalizers" on page 4-16.

This section describes how you can use CAGE to Þll your lookup tables by reference to a model.

To fill the table values, either click the buttons in the toolbar, , or select from the following options in the Table menu:

- Initialize Table

  Sets each cell in the lookup table to a specified value. For information, see "Initializing Table Values" on page 4-30.

- Fill Table

  Fills and optimizes the table values by reference to the model. For information, see "Filling Table Values" on page 4-31.

- Fill by Inversion

  Fills the table by creating an inversion of another table. For information, see "Inverting a Table" on page 3-46.

- Fill by Extrapolation

  Fills the table values based on the cells specified in the extrapolation mask. You can choose values in cells that you trust to define the extrapolation mask and fill the rest of the table using only those cells for extrapolation. For information, see "Filling the Table by Extrapolation" on page 4-35.

The next sections describe each of these toolbar options in detail. See the "Table Menu" on page 3-13 for other menu options.

Initializing Table Values

Initializing table values sets the value of every cell in the selected table to a constant. You can do this when you set up a table (see "Adding, Duplicating and Deleting Tables" on page 3-3) or later.

To initialize the values of the table,

1. Click or select Table > Initialize .
2. In the dialog box that appears, select the constant value that you want to insert into each cell.

When initializing tables, you should think about your strategy. Filling with zeros can cause a problem for some strategies using "modifier" tables. For

example, your strategy might use several speed-load tables for different values of AFR, or you might use an AFR table as a "modifier" to add to a single speed-load table to adjust for the effects of different AFR levels on your torque output.

Be careful not to initialize modifier tables with 0 if they are multipliers in your strategy. In this case, solving results in trying to divide by zero. This operation will fail. If your table is a modifier that is added to other tables, you should initially fill it with zeros; if it is a modifier that multiplies other tables, you should fill it with 1s.

Filling Table Values

To fill and optimize the table values by reference to the model,

- Click or select Table > Fill.

  This opens the Feature Fill Wizard. You can fill multiple tables at once using the wizard, and you can Fill from the top feature node or from any table node in a feature. See “Feature Fill Wizard” on page 4-40 for instructions.

Optimizing Table Values

The Feature Fill Wizard optimizes the table values to minimize the current total square error between the feature values and the model.

This routine optimizes the fit between your strategy and your model. Using Fill places values into your table. The optimization process shifts the cell values up and down to minimize the overall error between the interpolation between the model and the strategy.

This process is illustrated by the following example; the green shaded areas show the error between the mesh model (evaluated at the number of grid points you choose) and the table values.

To see the difference between optimizing table values and optimizing the positions of breakpoints, compare with the illustration in “Optimizing Breakpoints” on page 4-22.

CAGE evaluates the model over the number of grid points specified in the Feature Fill Wizard, then calculates the total square error between this mesh model and the feature values. CAGE adjusts the table values until this error is minimized, using `lsqnonlin` if there are no gradient constraints, otherwise `fmincon` is used with linear constraints to specify the gradient of the table at each cell.

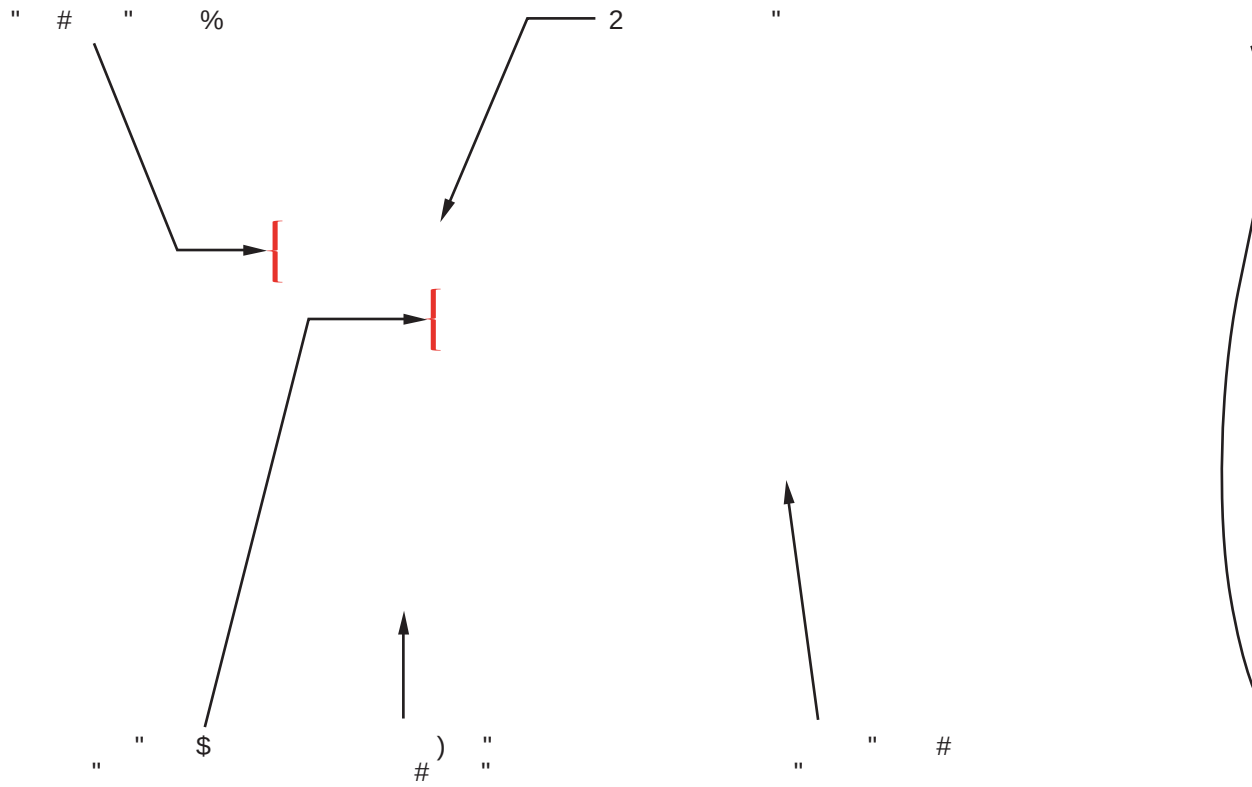
See Also.

- Reference page for `lsqnonlin`
- “Calibrating the Tables” on page 4-29

Comparing the St rategy and the Model

When you calibra te a strategy, or collection of tables, by reference to a model, it is useful to co mpare the strategy and the model. The comparison pane provides a graph ical tool for investigating this, as shown in the following example.

Note This is a co mparison between the current strategy values and the model, unlike t he comparison pane from the normalizer node, which compares the model and a f ull factorial grid Þlled using the breakpoints.



To make full use of the comparison-of-results pane,

1 Check the ranges of the variables that are common to the model and table. For each variable check the number of points at which the display is calculated. Double-click to edit any variable range or number of points.

2 Check the values selected for any vari ables in the model that are not in the selected table. The default for this is the set point of the variable's range. Double-click to edit.

3 Check the comparison between the table and the model. You can rotate this comparison by clicking and dragging, so that you can view all parts of the comparison easily.

4 Use the Plot Type drop-down menu to display the error statistics for the comparison.

Error Display

The comparison-of-results pane can also be used to display the error between the model and the strategy.

To display the erro r, select one of the Error options from the Plot Type drop-down menu. This changes the graph to display the error between the model and the strategy.

You can display the error data in one of the following ways:

- Error (Feature-Model) . This is the difference between the feature and the model.
- Squared Error . This is the error squared.
- Absolute Error . This is the absolute value of the error.
- Relative Error (%) . This is the error as a percentage of the value of the model.
- Absolute Relative Error (%) . This is the absolute value of the relative error.

When you have completed a calibration, you can export your feature. For information, see “Exporting Calibrations” on page 3-55.

Filling the Table by Extrapolation

Filling a table by extrapolation Þlls the table with values based on the values already placed in the extrapolation mask. The extrapolation mask is described below. You can also choose to extrapolate auto matically after Þlling cells in the mask in the “Feature Fill Wizard” on page 4-40.

To Þll a table by extrapolating over a preselected mask, click or select Table > Extrapolate .

This extrapolation does one of the following:

- If the extrapolation mask has only one value, all the cell values change to the value of the cell in the mask.
- If the extrapolation mask has two or mo re colinear values, the cell values change to create a plane parallel to the line of values in the mask.
- If the extrapolation mask has three or more coplanar values, the cell values change to create that plane.

- If the extrapolation mask has four or more ordered cells (in a grid), the extrapolation routine fills the cells by a grid extrapolation.
- If the extrapolation mask has four or more unordered (scattered) cells, the extrapolation routine fills the cell values using a thin plate spline interpolant (a type of radial basis function).

Using the Extrapolation Mask

The extrapolation mask de Þnes a set of cells that form the basis of any extrapolation.

For example, a speed-load (or relative air charge) table has values in the following ranges that you consider to be accurate:

- Speed 3000 to 5000 rpm
- Load 0.4 to 0.6

You can deÞne an extrapolation mask to include all the cells in these ranges. You can then Þll the rest of your table based on these values.

To add or remove a cell from the extrapolation mask,

1 Right-click the table.

2 Select Add To Extrapolation Mask or Remove From Extrapolation Mask from the menu.

Cells included in the extrapolation mask are colored yellow.

Creating a Mask from the Boundary Model or Predicted Error

You can automatically generate an extrapolation mask based on the boundary model or prediction error. Prediction error (PE) is the standard deviation of the error between the model and the data used to create the model.

To generate a mask automatically,

1 Select Table > Extrapolation Mask > Generate From Boundary Model or Generate From PE

2 If you select PE, a dialog appears where you must set the PE threshold to apply, and click OK.

The cells in the table either within the boundary model or where the prediction error is within the thresh old now form the extrapolation mask, and thus are colored yellow.

Calibrating the Feature Node

In this section...

"How to Calibrate the Feature" on page 4-38
"Initializing the Feature" on page 4-38
"Feature Fill Wizard" on page 4-40
"Feature View" on page 4-47
"Feature Menu" on page 4-48

How to Calibrate the Feature

Selecting a Feature node displays the Feature view. For more information about the Feature view, see "Feature View" on page 4-47.

The Feature view enables you to calibrate the entire feature, that is, Þll all the table values by referring to a model.

To calibrate the feature, either click the buttons on the toolbar, , or select from the following options on the Feature menu described in the following sections: "Initializing the Feature" on page 4-38 and "Feature Fill Wizard" on page 4-40.

Initializing the Feature

For example, a simple feature for maxi mum brake torque (MBT) consists of the following tables:

- A speed (N), load (L) table
- A table to account for the behavior of air/fuel ratio (A)

Initializing this feature sets the valu es of the normalizers for speed, load, and AFR over the range of each variable and put speci Þed values into each cell of the two tables.

A table that is already initialized provides a useful starting point for a more detailed calibration.

To initialize the feature, p erform the following steps:

1 Click . This opens the Feature Initialization Options dialog box, as shown.

2 Enter t he ranges for the breakpoints in your normalizers. In the preceding example, these are the breakpoint ranges:

- L has ra nge 0.2 0.811.
- N has ra nge 750 6500.
- A has ra nge 11 17.6.

3 Enter the initial table value for each cell in each table. Above, the cell values are

- Table_NL has initial value 0.
- Fn_A has initial value 0.

4 Click OK to initialize the feature.

Note The default values in this dialog box are taken from the variable dictionary. If you clear any Enable box, the associated table or normalizer is left unchanged.

Feature Fill Wizard

Use the Feature Fill Wizard to Þll and optimize the values in tables by reference to the model. You can Þll multiple tables at once using the wizard, and you can Fill from the top feature node or from any table node in a feature.

Note you could also optimize the break points for the normalizers before using the Feature Fill Wizard. (See “Filling Breakpoints” on page 4-18 and “Optimizing Breakpoints” on page 4-22.)

This section describes how to use the Feature Fill Wizard. For a detailed description about the Þlling processes, see “Filling Table Values” on page 4-31.

To Þll feature tables, perform the following steps:

1 Click . This opens the Feature Fill Wizard.

Screen 1: Select tables to Þll.

Select the check boxes of the tables you want to Þll. For each table you can set the following options:

- Clear Mask — select this check box to clear any table mask and Þll all unlocked table cells (locked cells a re never altered). Clear this check box to Þll unlocked cells in the current ex trapolation mask only, or all unlocked cells if there is no mask.
- Extrapolate — select this to extrapolate across the whole table after Þlling cells. The extrapolation is based on the Þlled cells in the mask and any locked cells.
- Table Bounds — enter values here to set bounds on the table values
- Gradient Bounds — enter values here to set bounds on the gradient (slope) between rows (left edit box) and between columns (right edit box). For example, entering 0 Inf in the left edit box imposes the constraint that the gradient must be positive (in creasing) between successive rows.

When you have selected Þlling options for each table, click Next .

2 Choose models and links.

- Click Select Model to choose a model to Þll the tables from. The feature Þller adjusts the table cells so that the value of the feature across the range of inputs best matches the value of this model.
- Click Select Constraint to choose a constraint to use in the Þlling process. You can use Linear, 1–D table, 2–D table, ellipsoid and model constraints (see “Edit Constraint” on page 6-39). The feature Þller limits its activity to within this constraint, for example, the boundary constraint of a model. While boundary models are often used as model constraints in this setting you can use any model. For example, you can use a function that returns a logical output (true for valid, false for invalid) by setting up the model constraint >=0.5.
- Click Link to associate a model, feature or table (selected on the right side) with a variable (selected on the left side). Linking replaces the variable inputs to any relevant models and features with the linked item. This enables useful operations such as feeding a table into a model, for example, an optimal cam schedule into a torque model, without needing

to make a separate function model. Click Unlink to disassociate any pair.

Click Next .

3 Set variable values.

By default the table's normalizer breakpoints and the set points of other variables are selected, so the number of grid points is the number of table cells. To increase the grid size you can enter more points for variables by editing the Value fields, or you can interleave values between breakpoints (see below). Increasing the number of grid points increases the quality of the approximation and minimizes interpolation error, but also increases the computation time.

- You can edit normalizers manually, or you can click the Initialize From Normalizer button to reach a dialog box where you can select normalizers and interleave values between breakpoints. Interleaving values can minimize interpolation error by adding values between each normalizer value. In this way you can create a grid of more points than table cells to optimize over. Select normalizers in the dialog box to use those breakpoints as a variable's value.

In this dialog box, you can enter a value in the Number of values between breakpoints edit box to add values between breakpoints. By default, the feature Þller compares the feature and model at the table breakpoints. Choose a positive value to compare the feature and model on a Þner grid. A positive value further enhances the comparison between feature and model to account also for errors introduced by linear interpolation in the table (see “Optimizing Table Values” on page 4-31). A value of 1 inserts one grid point between each pair of breakpoints, and so on. Click OK to return to the Feature Fill Wizard.

- Edit set point values in the Value Þelds to optimize over a range rather than at a single point. If you choose a range of values the table will be Þlled using the average model valu e at each cell. For example, if you enter -5:5:50 for the variable spark, the optimization of table values will be carried out at values of spark between -5 and 50 in steps of 5 degrees.

Click Next .

4 Fill Tables. Click Fill Tables to Þll the tables.

CAGE evaluates the model over the number of grid points speci Þed, then calculates the total square error between this mesh model and the feature values. CAGE adjusts the table values until this error is minimized, using lsqnonlin if there are no gradient constraints, otherwise fmincon is used with linear constraints to specify the gradient of the table at each cell.

The graph shows the change in RMSE as the optimization progresses.

- You can enter a value in the Smoothing edit box to apply a smoothing penalty to the optimization. The Smoothness penalty uses the second derivative to avoid steep jumps between adjacent table values. There is a penalty as smoothing trades smoother tables for increased error. Enter a smoothing factor (0–Inf) and click Fill Tables to observe the difference in the resulting RMSE and the table shape. Keep increasing the value until you reach the required smoothness. If you go too far the results will be a ßat plane.
- Select the Create dataset check box to create a dataset containing the output values at each speciÞed grid point.
- Select the Feature model check box to create a feature model (on Þnishing the feature Þll wizard) that is a static snapshot of the feature with its links included inside the feature model. If these links are features then the link is bundled up within the feature model of the feature being Þlled.
- Select the Fill model with links check box to create a model (on Þnishing the feature Þll wizard) that is a static snapshot of the Þll model

with its links connected to the model inputs (visible in the Connections diagram, in the Models view).

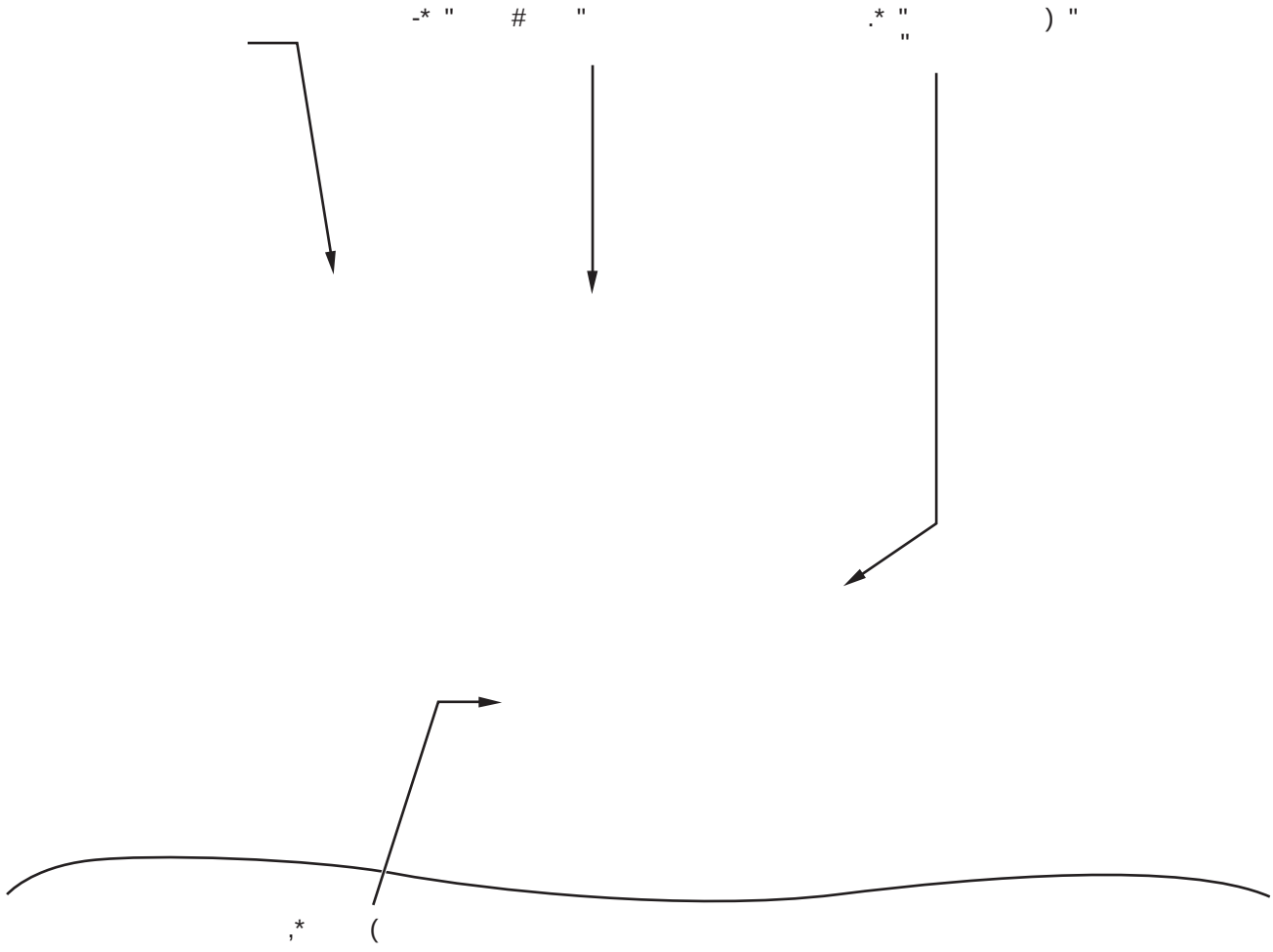
- Select the remaining check boxes to display plots when you close the Wizard. You can see plots of error against all the variables (Plot), error between feature and model (Error), table surface and error surface.

You can click Back to return to previous screens and fill more tables, or you can click Finish. When you click Finish to dismiss the wizard, the plots with selected check boxes appear.

When you have completed a calibration, you can export your feature. For information, see “Importing and Exporting Calibrations” on page 3-53.

Feature View

As you select a Feature node you see the Feature view, shown. This section describes the Feature view and the Feature menu options.



The parts of the Feature view include

1 The strategy for the selected feature. This is the algebraic collection of the tables that you are using to calibrate the selected feature.

2 The model associated with the selected feature.

3 The Feature History pane, which displays the history of the feature.

Feature Menu

The Feature menu has the following options:

- Select Model

  Use this to select the correct model for your feature.

- Deselect Model

  Use this to clear the current model from your feature.

- Convert to Model

  Takes the current feature and converts it to a model, which you can view by clicking the Model button.

- Graphical Strategy Editor

  Opens your current strategy for editing. For more information, see “Setting Up Your Strategy” on page 4-7.

- Parse Strategy Diagram

  Performs the same function as double-clicking the blue outport of your strategy diagram. For more information, see “Setting Up Your Strategy” on page 4-7.

- Clear Strategy

  Clears the current strategy from your feature.

- Initialize

  Initializes the feature; also in the toolbar. See “Initializing the Feature” on page 4-38 for details.

- Fill

  Fills and optimizes the feature; al so in the toolbar. See “Feature Fill Wizard” on page 4-40 for details.

5

Tradeoff Calibrations

This section includes the following topics:

- “Performing a Tradeoff Calibration” on page 5-2
- “Setting Up a Tradeoff C alibration” on page 5-5
- “Calibrating Tables in a Trade off Calibration” on page 5-11
- “Using Regions” on page 5-22
- “Point-by-Point Model Tradeoffs” on page 5-25
- “Automated Tradeoff” on page 5-32

Performing a Tradeoff Calibration

A tradeoff calibration is the process of calibrating lookup tables by adjusting the control variables to result in table values that achieve some desired aim.

For example, you might want to set the spark angle and the air/fuel ratio (AFR) to achieve the following objectives:

- Maximize torque
- Restrict CO emissions

The data in the tradeoff is presented in such a way as to aid the calibrator in making the correct choices. For example, sometimes the model is such that only a slight reduction in torque results in a dramatic reduction in CO emissions.

The basic procedure for performing tradeoff calibrations is as follows:

1 Set up the variables and constants. (See “Setting Up Variable Items” on page 2-2.)

2 Set up the model or models. (See “Setting Up Models” on page 2-10.)

3 Set up the tradeoff calibration. (See “Setting Up a Tradeoff Calibration” on page 5-5.)

4 Calibrate the tables. (See “Calibrating Tables in a Tradeoff Calibration” on page 5-11.)

5 Export the normalizers, tables, and tradeoffs. (See “Exporting Calibrations” on page 3-55.)

You can also use regions to enhance your calibration. (See “Using Regions” on page 5-22.)

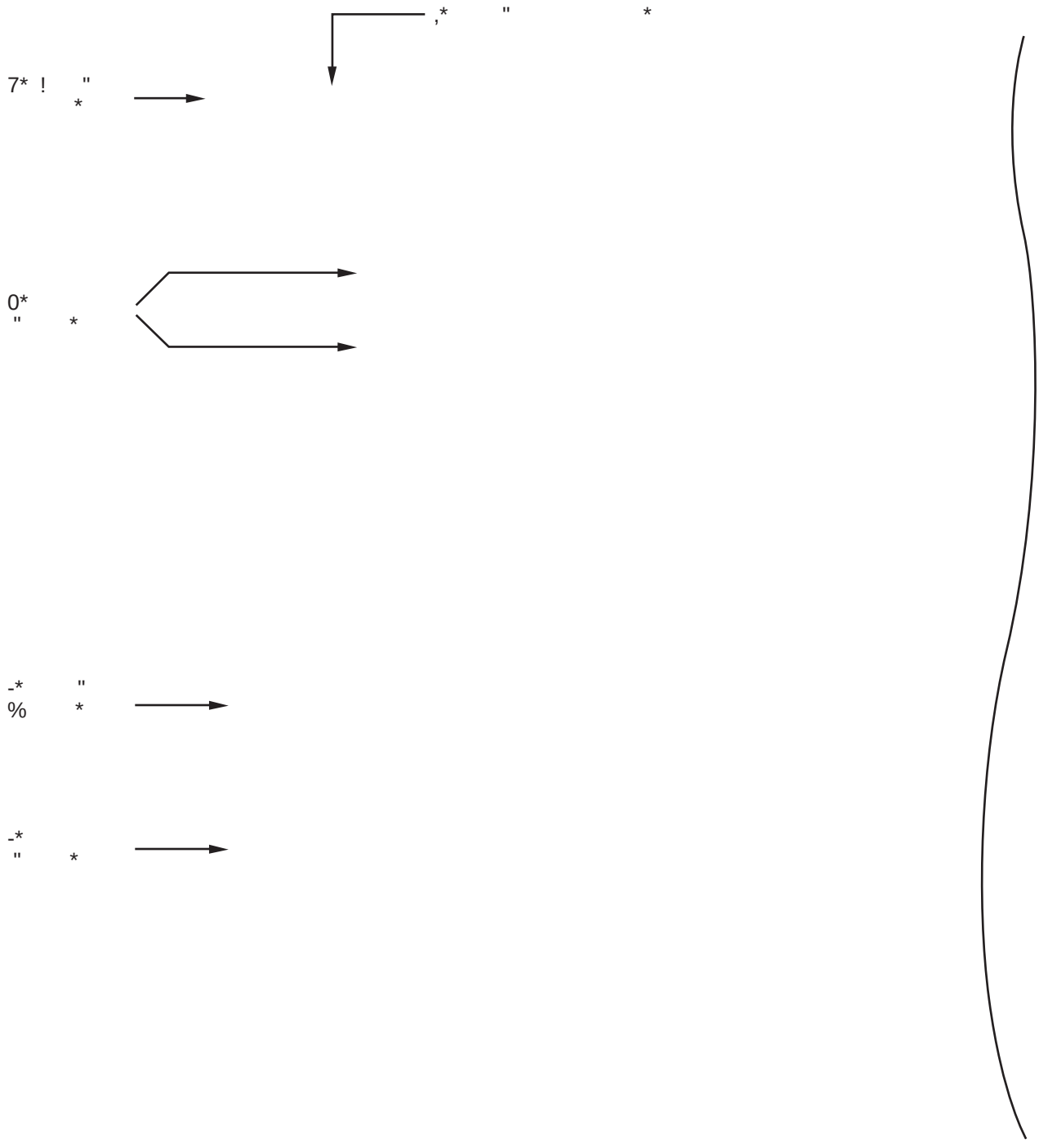
See also

- “Tutorial: Tradeoff Calibration” in the Getting Started documentation.

  This is a tutorial giving an example of how to set up and complete a simple tradeoff calibration.

- “Automated Tradeoff” on page 5-32 is a guide to using the optimization functionality in CAGE for tradeoffs.

The normalizers, tables, and tradeoff form a hierarchy of nodes, each with its own view and toolbar.


,* " *
7* ! "
*
0*
" *
-*
% *
-*
" *

Setting Up a Tradeoff Calibration

In this section...

"Overview of Setting Up a Tradeoff" on page 5-5

"Adding a Tradeoff" on page 5-6

"Adding Tables to a Tradeoff" on page 5-6

"Displaying Models in Tradeoff" on page 5-9

Overview of Setting Up a Tradeoff

A tradeoff calibration is the process of Þlling lookup tables by balancing different objectives.

Typically there are many different and con ßicting objectives. For example, a calibrator might want to maximize torqu e while restrictin g nitrogen oxides (NOX) emissions. It is not possible to achieve maximum torque and minimum NOX together, but it is possible to trade off a slight reduction in torque for a reduction of NOX emissions. Thus, a calibrator chooses the values of the input variables that produce this slight loss in torque instead of the values that produce the maximum value of torque.

A tradeoff also refers to the object that contains the models and tables. Thus, a simple tradeoff can involve balancing the torque output while restricting NOX emissions.

After you set up your variable items and models, you can follow the procedure below to set up your tradeoff calibration:

1 Add a tradeoff. This is described in the next section, "Adding a Tradeoff" on page 5-6.

2 Add tables to the tradeoff. This is described in "Adding Tables to a Tradeoff" on page 5-6.

3 Display the models. This is describe d in "Displaying Models in Tradeoff" on page 5-9.

This section describes steps 1, 2, and 3 in turn.

When you Þnish these steps, you are ready to calibrate the tables.

Adding a Tradeoff

To add a tradeoff to your session, select File > New > Tradeoff . This automatically switches you to the Tra deoff view and adds an empty tradeoff to your session.

An incomplete tradeoff is a tradeoff that does not contain any tables. If a tradeoff is incomplete, it is displayed as in the tree display. If a tradeoff is complete, it is displayed as in the tree display.

After you add a tradeoff you must add tables to your tradeoff.

Adding Tables to a Tradeoff

1 Add a table by selecting Tradeoff -> Add New Table or click in the toolbar. You can also add existing tables from your CAGE session; see “Adding Existing Tables” on page 5-9.

   Note that you must select the top tra deoff node in the tree display to use the Tradeoff menu. This is automatically selected if your tradeoff has no tables yet (it is the only node). You must also add at least three variables (in the variable dictionary) to your project before you can add a table, because CAGE needs a variable to Þll the table and two more variables to deÞne each of the two normalizers.

A dialog box opens.

2 Enter the name for the table.

If your tradeoff already contains one or more tables, when you add additional tables they must be the same size and have the same inputs (and therefore have the same normalizers). So if your tradeoff has existing tables, you can only enter the new table name and the initial value.

For the first table in a tradeoff, you must set the normalizer inputs and sizes:

a Edit the names for the X and Y normalizer inputs (the first two variables in the current variable dictionary are automatically selected here).

b Enter sizes for each of the normalizers (Y input = rows, X input = columns)

3 Enter an initial value to fill the table cells, or leave this at zero.

4 Click Select to choose a Þlling item for a table. A dialog opens where you can select from the models and variables in your session.

a Depending on what kind of input you want, click the radio buttons to display models or variables or both. You can choose to also show items that are Þlling another table by clearing the check box.

b Select the Þlling item for the table and click OK.

5 Click OK to dismiss the Table Setup dialog and create the new table.

CAGE adds a table node to the tradeoff tree. Note you can still change the input for the table as follows. Double-click the new table in the list under Tables In Tradeoff, or click to select the table (it is selected automatically if it is the only table in the tradeoff) and then click Change Filling Item ( ) in the toolbar. This is also in the Tradeoff menu and the right-click context menu.

The Select Filling Item dialog appears where you can select inputs to Þll the table, as described above.

6 Repeat this procedure for each new table you want to add. Each additional table in the tradeoff must have the same normalizers as the first table, so you do not have to select normalizer inputs and sizes repeatedly. For each new table you only have to enter the name and initial value.

Adding Existing Tables

1 Add a table by selecting Tradeoff > Add Existing Tables or click in the toolbar.

A dialog appears where you can select from a list of tables in the current session.

2 Select a table and click OK. It may be helpful to first select the check box to only show suitable tables that can be added to the tradeoff.

Displaying Models in Tradeoff

To display models when viewing tables in the tradeoff display,

1 Highlight the tradeoff node in the tree.

2 From the Available Models list, select the one you want to display.

Models that are filling a table are automatically displayed.

3 Click Add Model to Display List in the toolbar or in the Additional Display Models pane to move the selected model into the Display Models pane. To quickly add all available models to the display list, click the display button repeatedly and each successive model will be added.

4 Repeat steps 2 and 3 to add all the models you want to the display list.

Removing a Model

1 In the Display Models list, select the model that you want to remove.

2 Click in the toolbar, or in the Display Models pane, to move the selected model into the Available Models pane.

3 Repeat until you have cleared all the appropriate models.

Once you have displayed all the models that you want to work with, you are ready to calibrate your tables.

Calibrating Tables in a Tradeoff Calibration

In this section...

“Procedure For Calibrating Tradeoff Tables” on page 5-11

“Setting Values of Other Variables” on page 5-14

“Determining a Value at a Specific Operating Point” on page 5-16

“Tradeoff Table Menus” on page 5-18

Procedure For Calibrating Tradeoff Tables

Selecting a table node in the tree display enables you to view the models that you have displayed and calibrate that table.

To calibrate the tables,

1 Select the table that you want to calibrate.

2 Highlight one operating point from the table.

3 Set the values for other input variables.

For information, see “Setting Values of Other Variables” on page 5-14.

4 Determine the value of the desired operating point.

For instructions, see “Determining a Value at a Specific Operating Point” on page 5-16.

5 Click to apply this value to the lookup table.

This automatically adds the point to the extrapolation mask.

6 Repeat steps 2, 3, 4, and 5 at various operating points.

7 Extrapolate to fill the table by clicking in the toolbar.

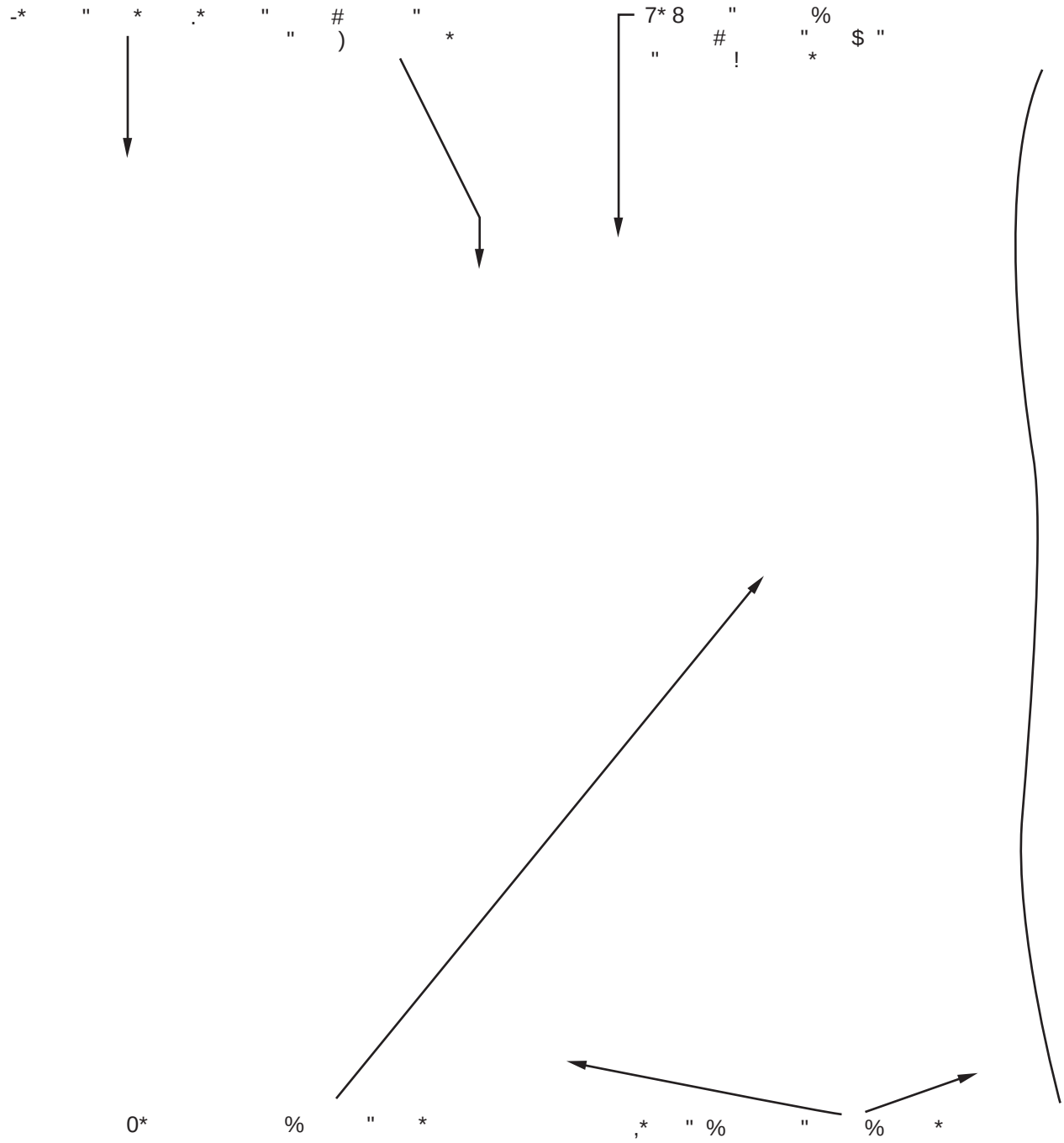
For information, see “Filling the Table by Extrapolation” on page 4-35.

8 You can also edit table cell values manually by typing values, or right-click to Copy or Paste values.

After you complete all these steps you can export your calibration. For information, see “Exporting Calibrations” on page 3-55.

Notice that the graphs colored green ind icate how the highlighted table will be Þlled:

- If a row of graphs is highlighted, the table is being Þlled by the indicated model evaluation (the value shown at the left of the row).
- If the column of graphs is green, the table is being Þlled by the indicated input variable (shown in the edit box below the column).

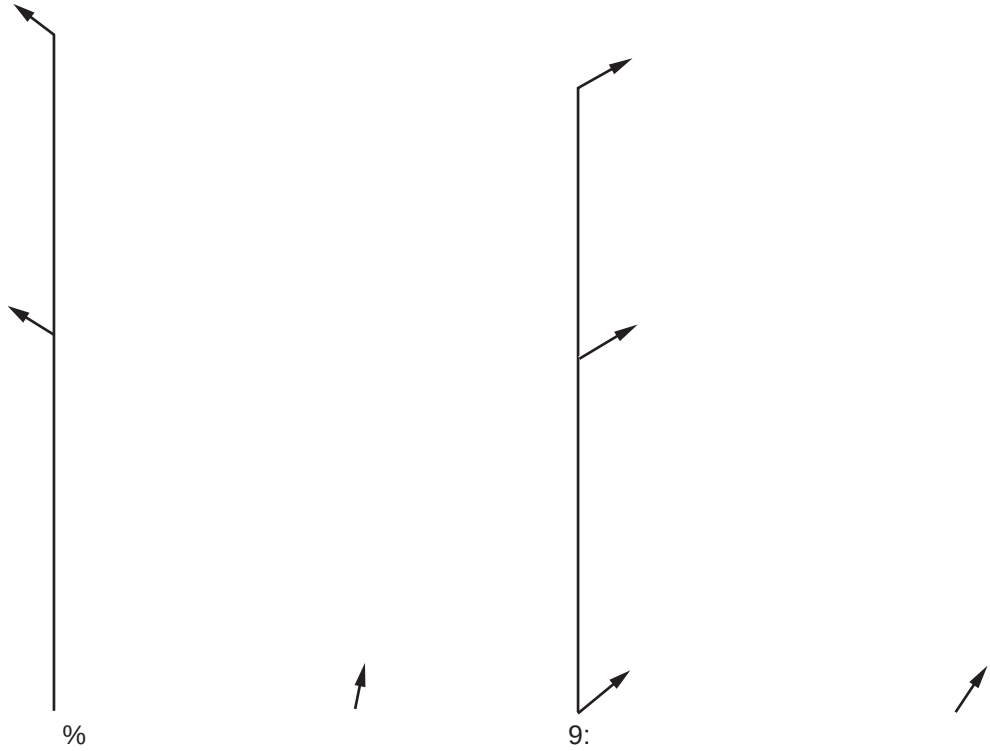


The next sections describe the following in detail:

- “Setting Values of Other Variables” on page 5-14
- “Determining a Value at a Speci Þc Operating Point” on page 5-16

Setting Values of Other Variables

Typically the models that you use to perform a tradeoff calibration have many inputs. When calibrating a table of just one input, you need to set values for the other inputs.



Setting Values for Individual Operating Points

To set values for inputs at individual operating points,

1. Highlight the operating point in the lookup table.
2. Use the edit boxes or drag the red bars to specify the values of the other variables.

In the preceding example, the spark table is selected (the SPK graph is colored green). You have to specify the values of AFR (A) and EGR (E) to be used, for example:

1. Select the spark table node.
2. Click in the edit box for A and set its value to 14.3 .
3. Click in the edit box for E and set its value to 0.

The default values are the set points of variables, which you can edit in the Variable Dictionary.

Setting Values for All Operating Points

For example, if you are using a tradeof f to calibrate a table for spark angle, you might want to set the initial values f or tables of air/fuel ratio (AFR) and exhaust gas recycling (EGR).

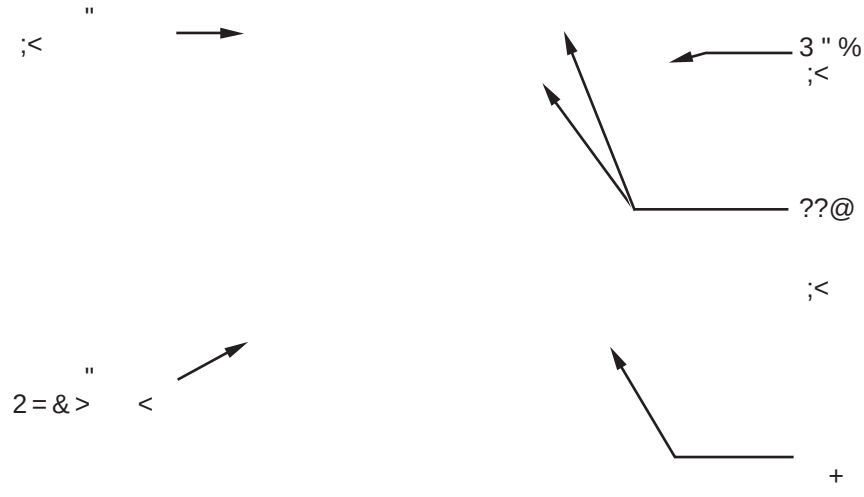
To set constant values for all the operating points of one table,

1. Highlight the table in the tree display.
2. Select one operating point in the table.
3. Enter the desired value of the cell.
4. Right-click and select Extrapolation Mask > Add Selection .

   This adds the cell to the extrapolation mask.
5. Click to extrapolate over the entire table.

This Þlls the table with the value of the one cell.

Determining a Value at a Specific Operating Point



Performing a tradeof f calibration necessarily inv olves the comparison of two or more models. For example, in this case , the tradeoff allows a calibrator to check that a value of spark that gives p eak torque also gives an acceptable value for the NOX ßow model.

1 To select a value of an input, do one of the following:

- Drag the red line.
- Right-click a graph and select Find the minimum, maximum, or turning point of the model as appropriate (also in the toolbar and Inputs menu).

- Click the edit box under the graph as shown above and enter the required value.

2 Once you are satisfied with the value of your variable at this operating point, you apply this value to the table by doing one of the following:

- Press Ctrl+T.
- Click (Apply Table Filling Values) in the toolbar.
- Select Tables > Apply > Fill to Table.

Right-Click Menu

Right-clicking a graph enables you to

- Find minimum of model output with respect to the input variable
- Find maximum of model output with respect to the input variable
- Find turning point of model with respect to the input variable

  These first three options are also in the Inputs menu.
- Reset graph zooms (also in the View menu)

There are also toolbar buttons to find the minimum, maximum and turning point of the selected model graph.

Using Zoom Controls on the Graphs

To zoom in on a particular region, shift-click or click with both mouse buttons simultaneously and drag to define the region as a rectangle.

To zoom out to the original graph, double-click the selected graph, or use the right-click Reset Graph Zooms option (also in the View menu).

Note Zooming on one graph adjusts other graphs to the same scale.

Tradeoff Table Menus

View Menu

Selecting the View menu offers you the following options:

- Table History

  This opens the History display. For information, see “Using the History Display” on page 3-20.

- Con Þgure Hidden Items

  This opens a dialog box that allows you to show or hide models and input variables. Select or clear the check b oxes to display or hide items. This is particularly useful if you are trading off a large number of models or models that have a large number of factors.

- Display Con Þdence Intervals

  When you select this, the graphs display the 99% con Þdence limits for the models.

- Display Common Y Limits

  Select this to toggle a common y-axis on and off for all the graphs. You can also press CTRL +Y as a shortcut to turn common Y limits on and off.

- Display Constraints

  Select this to toggle constraint di splays on and off. Regions outside constraints are shown in yellow on the graphs, as elsewhere in the toolbox.

- Graph Size

  Select from the following options for number and size of graphs:

  - Display All Graphs
  - Small
  - Medium
  - Large

- Large Graph Headers

  Select this to toggle graph header size. The smaller size can be useful when you need to display many models at once.

- Reset Graph Zooms

  Use this to reset if you have zoomed in on areas of interest on the graphs. Zoom in by shift-clicking (or clicking both buttons) and dragging. You can also reset the zooms by double-clicking , or by using the right-click context menu on the graphs.

- Display Table Legend

  Select this to toggle the table legend display on and off. You might want more display space for table cells on ce you know what the legend means. The table legend tells you how to interpret the table display:

  - Cells with a tick contain saved values that you have applied from the tradeoff graphs (using the 'Apply table Þlling values' toolbar or menu option).
  - Yellow cells are in the extrapolation mask.
  - Blue cells are in a region mask.
  - Yellow and blue cells with rounded corners are both in a region and the extrapolation mask.
  - Cells with a padlock icon are locked.

Tables Menu

- Apply Fill to Table

  Select this option to apply the valu es from the tradeoff graphs to the selected table cell. This option is also in the toolbar, and you can use the keyboard shortcut CTRL +T.

  Note that the corresponding cell in all tables is Þlled with the appropriate input, not just the cell in the currently displayed table. For example if you have graphs for spark and EGR inputs, selecting Apply Fill to Table Þlls the spark table cell with the spark value in the graphs, and the EGR table cell with the EGR value.

- Extrapolation Mask — Also available in the toolbar and the context menu (by right-clicking a table cell). Use these options to add and remove cells from the mask for Þlling tables by extrapolation. Note that cells Þlled by applying values from the tradeoff graphs (using the Apply

Fill To Table toolbar and menu option) are automaticall y added to the extrapolation mask.

- Add Selection
- Remove Selection
- Clear Mask

- Extrapolation Regions — Also available in the toolbar and the context menu (by right-clicking a table cell). Use these options to add and remove cells from regions. A region is an area that de Þnes locally where to extrapolate before globally extrapolating over the entire table. Use regions to deÞne high-priority areas for use when Þlling tables by extrapolation. See “Using Regions” on page 5-22.
  - Add Selection
  - Remove Selection
  - Clear Regions
- Extrapolate — This option (also in the toolbar) Þlls the table by extrapolation using regions (to de Þne locally where to extrapolate before globally extrapolating) and the cells de Þned in the extrapolation mask.
- Extrapolate (Ignore Regions) — This option Þlls the table by extrapolation only using cells in the extrapolation mask.
- Table Cell Locks — Also available in the context menu by right-clicking a table cell. Use these options to lock or unlock cells; locked cells are not changed by extrapolating.
  - Lock Selection
  - Unlock Selection
  - Lock Entire Table
  - Clear All Locks

Inputs Menu

- Reset to Last Saved Values — This option resets all the graph input values to the last saved value. Also in the toolbar.

- Set to Table Value — This option sets the appropriate input value on the graphs to the value in the table.

The following three options are only enabled if a graph is selected (click to select, and a blue frame appears around the selected graph). They are also available in the right-click context menu on the graphs.

- Find Minimum of model vs input factor
- Find Maximum of model vs input factor
- Find Turning Point of model vs input factor

  where model and input factor are the model and input factor displayed in the currently selected graph, for example, TQ_model vs Spark.
- Automated Tradeoff — Use this option once you have set up an optimization, to apply that optimization to the selected region of your tradeoff table. See “Automated Tradeoff” on page 5-32 for information.

Tools Menu

- Calibration Manager — opens the Calibration Manager. See “Calibration Manager” on page 3-24.
- Surface Viewer — Opens the Surface Viewer. See Chapter 8, “Surface Viewer”.

Using Regions

In this section...

“What Are Regions?” on page 5-22

“DeÞning a Region” on page 5-23

“Clearing a Region” on page 5-23

What Are Regions?

A region is an area that de Þnes locally where to extrapolate before globally extrapolating over the entire table.

For example, consider Þlling a large table that has twenty breakpoints for each normalizer by extrapol ation. Two problems arise:

- To have meaningful results, you need to set values at a large number of operating points.
- To set values at a large number of ope rating points takes a long time.

To overcome this problem, you can

1 DeÞne regions within the lookup table.

2 In each region, set the values of some operating points.

3 Click to Þll the table by extrapolation.

Each region is Þlled by extrapolation in turn. Then the rest of the table is Þlled by extrapolation. The advantage of using regions is that you can have more meaningful results by setting values fo r a smaller number of operating points.

Cells are colored

- Yellow if they form part of the extrapolation mask
- Blue if they are part of a region
- Yellow and blue with rounded corners i f they are part of the extrapolation mask and part of a region

Defining a Region

1 Click and drag to highlight the rectangle of cells in your table.

2 To deÞne th e region, click in the toolbar, or right-click and select Extrapola tion Regions > Add Selection , or select the menu option Tables > E xtrapolation Regions > Add Selection .

The cells in the region are colored blue.

Clearing a Region

1 Highligh t the rectangle of cells in your table.

2 To clear the regi on, click in the toolbar, or right-click and select Extrapolation R egions > Remove Selection , or select the menu option Tables > Extrapo lation Regions > Remove Selection .

You can clear all regions at once by selecting Clear Regions from the Extrapolation Regions submenu.

Point-by-Point Model Tradeoffs

In this section...

“What Is A Point-by-Point Mo del Tradeoff?” on page 5-25

“Adding a Point-by-Point Model Tradeoff” on page 5-26

“Calibrating Using a Point-by-P oint Model Tradeoff” on page 5-29

What Is A Point-by-Point Model Tradeoff?

There are two types of tradeoff that you can add to your session, a tradeoff of independent models, as described earlier (see “Performing a Tradeoff Calibration” on page 5-2), or a tradeoff of interconnected models: a point-by-point model (or multimodel) tradeoff.

A point-by-point model tradeoff is a spe cially built collection of models from the Model Browser.

You can build a series of models so that each operating point has a model associated with it. In the Model Browser, you can export models for a point-by-point model tradeoff from the test plan node. The models must be two-stage and must have exactly two global inputs. You can use the point-by-point test plan template to create these models. For more information see “How to Set Up a Poin t-by-Point Model” and “Exporting Point-by-Point Models to CAGE”.

The procedure for calibrating by using a po int-by-point model tradeoff follows:

1 Add the point-by-point model tradeoff. (See the following section, “Adding a Point-by-Point Model Tradeoff” on page 5-26.)

2 Calibrate the tables. (See “Calibrating Using a Point-by-Point Model Tradeoff” on page 5-29.)

3 Export your calibration. (See “Impor ting and Exporting Calibrations” on page 3-53.)

The point-by-point model is only de Þned for certain cells in the tradeoff tables. These are the operating points that we re modeled using the Model Browser

part of the toolbox. These cells have model icons in the table. At each of these operating points, you can use the model to trade off, and by doing this you can adjust the value in the table. The point-by-point model is not de Þned for all other cells in the table and so you cannot use models to tradeoff. You can edit these cells and they can be Þlled by extrapolation. You trade off values at each of the model operating points in exactly the same way as when using independent models, as described in “Determining a Value at a SpeciÞc Operating Point” on page 5-16. When you have determined table values at each of the model operating points, you can Þll the whole table by extrapolation by clicking . See “Filling the Table by Extrapolation” on page 4-35.

Adding a Point-by-Point Model Tradeoff

To add a point-by-point model tradeoff to your session, you can either:

- Automatically create the tradeoff by exporting directly from the Model Browser test plan. See “Exporting Point-by-Point Models to CAGE”.
- In CAGE, create a new tradeoff and then import the point-by-point models from a Þle.

To create a new tradeoff from scratch:

1 Select File > New > Tradeoff . CAGE switches to the tradeoff view and creates a new empty tradeoff.

2 Select File > Import > Point-by-Point Model Tradeoff File .

The Þle must have been exported from the Model Browser, from the test plan. See “Exporting Point-by-Point Models to CAGE”.

3 Select the correct Þle to import and click Open . This opens a dialog box.

4 In the left Model sites list, you can clear the check boxes for any models at operating points that you do not want to import.

Notice that the operating points are displayed graphically at the top. If an operating point is deselected, it is displayed as gray here, rather than blue.

CAGE will create tables for all the models and input variables, with breakpoints at all the model operating points. You can choose not to create

all the tables; click Select Tables to choose from the list which tables you want.

5 Choose the normalizers (axes) of the tables by using the X- and Y-axis input drop-down menus.

6 You can adjust the number of breakpoints in the following ways:

- Leave the Automatic breakpoint settings radio button selected and edit the relative tolerances around the model sites. Use the tolerance edit boxes in the model setup pane. You can observe the effects of altering the tolerances on the number of breakpoint dotted lines drawn on the top graphic. Initially each model site has a breakpoint. If operating points are close together, you can increase the tolerances to decrease the number of breakpoints.

  For example, if several close points may all have been intended to run at exactly the same point, you might want to adjust the tolerances until those model points (displayed as blue dots) only have one breakpoint line. The number of rows and columns that will be created is displayed in the edit boxes on the right.

- Alternatively you can select the Manual breakpoint settings radio button and enter the number of rows and columns in the edit boxes, and you can directly edit the values of the breakpoints.

7 Click OK.

When you click OK, CAGE creates all the tables for the multimodel tradeoff, with breakpoints at the values you have selected.

Note When you calibrate the tables, you can only use models to tradeoff at the operating points defined for the models. These cells have model icons in the table. You can edit other cells, but they have no models to tradeoff associated with them.

You can now calibrate your tables. See the next section, “Calibrating Using a Point-by-Point Model Tradeoff” on page 5-29.

Calibrating Using a Point-by-Point Model Tradeoff

Each editable operating point in your tables has a model icon in the cell, like this example cell.

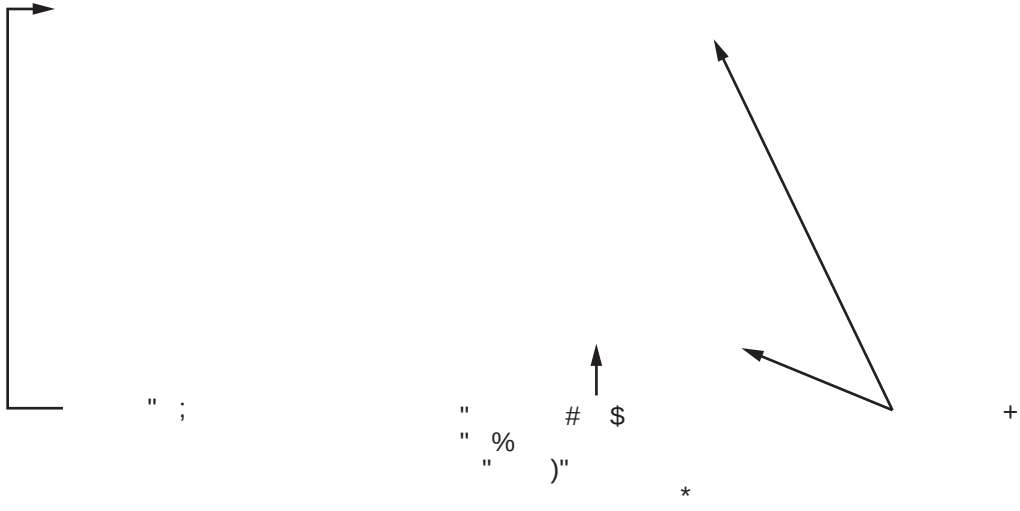
These cells have a model defined at that point. You use the display of these models to help you trade off values at these points to fulfill your aims in exactly the same way as when using independent models in "ordinary" tradeoff mode, as described in "Determining a Value at a Specific Operating Point" on page 5-16.

1. Change input values by dragging the red lines on the graphs or by typing directly into the edit boxes above the graphs. Use the context menu, toolbar or Inputs menu to find the maximum, minimum, or turning point of a model if appropriate.

2. Look at the model evaluation values (to the left of each row of graphs) and the input variable values (in the edit boxes below the graphs) to see if they meet your requirements.

   Remember that the green highlighted graphs indicate how the selected table is filled: if a row is green, the model evaluation value (to the left) fills the table at that operating point; if a column is green, the input variable value (in the edit box below) fills the table.

See the example following; the SPK column of graphs is green, so the value of SPK in the edit box is entered in the table when you click the Apply Table Filling Values button ( ).



3 When you are satis Þed with the tradeoff given by the value of your variable at this operating point, you apply this value to the table by pressing Ctrl +T, selecting Tables -> Apply Fill to Table , or clicking in the toolbar.

4 When you have determined table values at each of the model operating points, you can Þll the whole table by extrapolation by clicking . See “Filling the Table by Extrapolation” on page 4-35.

You can then export your calibratio n; see “Importing and Exporting Calibrations” on page 3-53. An example point-by-point tradeoff is shown following.

Automated Tradeoff

In this section...

"Using Automated Tradeoff" on page 5-32

"What Are Appropriate Optimizations?" on page 5-34

Using Automated Tradeoff

You can use automated tradeoff to run an optimization routine and fill your tradeoff tables. Once you have set up an optimization and a tradeoff, you can run an automated tradeoff. As with any other tradeoff you need at least one table. You can apply an optimization to a cell or region of a tradeoff table, or the whole table, and the tradeoff values found are used to fill the selected cells. If only filling selected cells you can then fill the entire table by extrapolation.

You must first set up an optimization to use automated tradeoff.

There is an example automated tradeoff in the optimization tutorial chapter, "Tutorial: Optimization and Automated Tradeoff" in the Getting Started documentation.

1 You need a CAGE session with some models and a tradeoff containing some tables.

- See Chapter 5, "Tradeoff Calibrations" for instructions on setting up a tradeoff. You could use the tradeoff tutorial to generate a suitable example session.

You also need to set up an optimization before you can run an automated tradeoff. Objectives and constraints must be set up.

- For an example work through the step-by-step tutorial to set up some optimizations and then apply them to a tradeoff table. See "Tutorial: Optimization and Automated Tradeoff" in the Getting Started documentation.

2 Go to the tradeoff table you want to automate. You can select some table cells to apply the optimization to, or use the whole table, or fill only previously saved tradeoff points. Note that if you define a large region with many cells or a whole table it can take a long time to complete the

optimization. You can select individual cells, or click and drag to select a rectangle of cells. The selected cells do not have to be adjacent. Try a small region (say up to six cells) to begin with. Right-click selected cells and select Extrapolation Regions -> Add Selection or use the toolbar button (to add selection to extrapolation regions).

3 To apply optimization: click in the toolbar, or select Inputs -> Automated Tradeoff .

- A dialog appears that allows an appropriate (de Þned below) optimization to be selected from the current project.

Note You must set up an optimization to run before you can perform an automated tradeoff. You do this in the Optimization view. See also “Setting Up Point-by-Point Optimizations” on page 6-8.

The set of cells in the region you have selected becomes the operating point set for the optimization. The cell/region breakpoint values are used to replace the Þxed variable values in the selected optimization. Note that the existing Þxed variable values are reset to their previous state at the end of the automated tradeoff.

If previous tradeoff values have been applied to a cell, those values are used for free variable initial values and non-table-axis Þxed variables; otherwise the set points are used.

4 The optimiz ation is run as if you were clicking Run from the Optimization view. See “Running Optimizations” on page 6-44.

Results are placed in the tradeoff obj ect, that is, values for the tables involving the free variables or valu es for the tables for constraint or objective models. If the routine appl ied gives more than one solution, for example, an NBI optimization, then a solution which tries to trade off all objectives is placed in the tradeoff tables. Every cell in the de Þned region is Þlled.

5 The cells of the region become part of the extrapolation mask (as if apply point has been applied); so if you want you can then click Extrapolate in the toolbar to Þll the rest of the table from your o ptimized automa ted tradeoff.

What Are Appropriate Optimizations?

The list of all optimizations in the project is filtered. To be eligible for selection,

- The optimization must be ready to run (toolbar button enabled).
- The variables in the axes of the tradeoff tables must not be free variables in the optimization. For example, if one of the axes is speed, then speed cannot be a free variable.
- The fixed variables must be a subset of the variables in the axes of the tradeoff tables. For example, if the optimization requires variables Speed and Load, then these must be the axes variables in the tradeoff table.
- The optimization must either have N runs with all variables of length 1, or a single run with all variables of length N.

Multimodel Tradeoff

For a multimodel tradeoff, things work slightly differently. The multimodel is only defined for certain cells in the tradeoff tables. These are the operating

points that were modeled using the Model Browser part of the toolbox. Such cells are marked with a model icon as shown in the example, and you should select these for running the automated tradeoff. You can select any region, but the optimization can only find values for the operating points defined by the multimodel.

6

Optimization

This section includes the following topics:

- “Using the Optimization View” on page 6-2
- “Optimization Problems You Can Solve with CAGE” on page 6-4
- “Creating an Optimization” on page 6-7
- “DeÞning Variable Values” on page 6-27
- “Objectives and Constraints” on page 6-36
- “Running Optimizations” on page 6-44
- “Using the Optimization Parameters Dialog Box” on page 6-46
- “Optimization Output Views” on page 6-61
- “Using Optimization Output” on page 6-89
- “Analyzing Point-by-Point Optimization Output” on page 6-98
- “Interpreting Sum Optimization Output” on page 6-114
- “User-DeÞned Optimization” on page 6-128
- “Optimization Function Reference” on page 6-136
- “Functions — Alphabetical List” on page 6-141

Using the Optimization View

Optimization functionality is one of the CAGE processes. The Optimization button can be found in the left Processes pane.

To reach the Optimization view, click the Optimization button.

Here you can set up and view optimizations. As with other CAGE processes, the left Optimization pane shows a tree hierarchy of your optimizations, and the right panes display details of th e optimization selected in the tree. When you Þrst open the Optimization view both panes are blank until you create an optimization.

As for other CAGE processes, you must set up your session for an optimization. For any optimization, you need one or more models. You can run an optimization at a single point, or you can supply a set of points to optimize. The steps required are

1 Import a model or models.

2 Set up a new optimization.

Optimization functionality in CAGE is described in the following sections:

• “Optimization Problems You Can Solve with CAGE” on page 6-4

- The steps for setting up and running optimizations are described in these sections:
  - “Creating an Optimization” on page 6-7
    - “Setting Up Point-by-Point Optimizations” on page 6-8
    - “Setting Up Sum Optimizations” on page 6-16
  - “Objectives and Constraints” on page 6-36
  - “Running Optimizations” on page 6-44
  - “Using the Optimization Parameters Dialog Box” on page 6-46
- “Optimization Output Views” on page 6-61 describes using the views to analyze your results.
- After you set up an optimization, you can apply it to a region in a set of tradeoff tables. See “Automated Tradeoff” on page 5-32.
- You can deÞne your own optimization functions for use in CAGE. See “User-DeÞned Optimization” on page 6-128.

There are tutorial examples to gu ide you through the optimization functionality. See “Tutorial: Optim ization and Automated Tradeoff”, and see the optimization sections in “ Gasoline Engine Calibration Case Study” and “Diesel Engine Calibratio n Case Study”, in the Getting Started documentation.

Optimization Problems You Can Solve with CAGE

In this section...

"Point-by-Point Optimization Problems" on page 6-4

"Sum Optimization Problems" on page 6-5

Point-by-Point Optimization Problems

CAGE provides a flexible optimization environment in which many automotive optimization problems can be solved. These problems can be divided into two main groups, point-by-point and sum problems. This section describes point-by-point problems.

In a point-by-point problem, a single optimization run can determine optimal control parameter values at a single operating point. To optimize control parameters over a set of operating points, an optimization can be run for each point.

Examples of point-by-point problems that CAGE can be used to solve are described below:

- Find the optimal spark timing (SPK), intake valve timing (INTCAM) and exhaust valve timing (EXHCAM) at each point of a lookup table whose axes are engine speed (N) and relative load (L).

  Optimized values of the control parameters are determined by running the following optimization at each point of the lookup table:

  Objective: Maximize engine torque, TQ = TQ(N, L, SPK, EXHCAM, INTCAM)

  Constraints:

  - Residual fraction <= 17% at each (N, L) operating point
  - Exhaust temperature <= 1290°C at each (N, L) operating point
  - Engine to be operated inside the operating envelope of the engine
- Find the optimal mass of fuel injected (F), rail pressure (P), pilot timing (PT) and main timing (MT) at each point of a lookup table whose axes are engine speed (N) and engine torque (TQ).

Optimized values of the control parameters are determined by running the following optimization at each point of the lookup table:

Objective: Minimize brake specific fuel consumption, BSFC = BSFC(N, TQ)

Constraints:

- Engine out NOx <= 0.001 kg/s at each (N, TQ) operating point
- Engine out Soot emissions <= 0.0001 kg/s at each (N, TQ) operating point

- Find the optimum spark timing (SPK) and exhaust gas recirculation (EGR) at each point of a set of operating points defined by engine speed (N), engine load (L) pairs. Optimized values of SPK and EGR are determined by running the following optimization at each point:

  Objective: Maximize engine torque, TQ = TQ(N, L, SPK, EGR)

  Constraints: Engine out NOx <= 400 g/hr at each (N, L) operating point

- For a new engine, find out the optimal torque versus NOx emissions curve for this engine over the operating range of the engine. This is a multi-objective optimization, and CAGE Optimization contains an algorithm (NBI) to solve these problems.

  For this example, the optimal torque-NOx curve is determined by solving the following optimization problem for optimal settings of spark timing (SPK) and exhaust gas recirculation (EGR):

  Objectives:

  - Maximize engine torque, TQ = TQ(N, L, SPK, EGR)
  - Minimize engine out NOx = NOx(N, L, SPK, EGR)

To find out more about solving these types of problems in CAGE, see “Setting Up Point-by-Point Optimizations” on page 6-8.

Sum Optimization Problems

In a sum optimization, a single optimization run can determine the optimal value of control parameters at several operating points simultaneously. All the control parameters for the operating points are optimized by calling the algorithm once (there’s only one call to `foptcon` per run for a sum optimization). This approach contrasts with a point-by-point optimization,

which has to make a call to the algorithm for every point to find the optimal settings of the control parameters.

- Find the optimal spark timing (SPK), intake valve timing (INTCAM) and exhaust valve timing (EXHCAM) at each point of a look-up table whose axes are engine speed (N) and relative load (L).

  Optimized values of the control parameters are determined by running the following optimization once:

  Objective: Maximize weighted sum of engine torque, TQ = TQ(N, L, SPK, EXHCAM, INTCAM) over the (N, L) points of a look-up table.

  Constraints:

  - Difference in INTCAM between adjacent cells is no greater than 5°.
  - Difference in EXHCAM between adjacent cells is no greater than 10°.
  - At each table cell, residual fraction <= 17%
  - At each table cell, exhaust temperature <= 1290°C

- Find the optimal start of injection (SOI), basefuelmass (BFM), fuel pressure (P), turbo position (TP) and lift of the EGR valve (EGR) at a set of mode points defined by engine speed (N), engine torque (TQ) pairs.

  Optimized values of the control parameters are determined by running the following optimization once:

  Objective: Maximize weighted sum of brake specific fuel consumption, BSFC = BSFC(SOI, BFM, P, TP, EGR, N, TQ) over the (N, TQ) mode points.

  Constraints:

  - Weighted sum of brake specific NOx must be less than a legislated maximum
  - At each mode point, air fuel ratio must be greater than a specified minimum
  - At each mode point, turbo speed must not exceed a specified maximum

To find out more about solving these types of problems in CAGE, see “Setting Up Sum Optimizations” on page 6-16.

Creating an Optimization

In this section...

"Overview of Creating Optimizations" on page 6-7
"Setting Up Point-by-Point Optimizations" on page 6-8
"Optimization Wizard" on page 6-8
"Optimization View Toolbar" on page 6-15
"Setting Up Sum Optimizations" on page 6-16
"Parallel Computing in Optimization" on page 6-25

Overview of Creating Optimizations

To create a new optimization, select File > New > Optimization .

This takes you to the Optimization Wizard, which leads you through the steps of choosing the optimization to run, specifying the number of variables to optimize over (unless this is predefined by the function), and linking the variables referenced in the optimization to CAGE variables.

For guidance, see the following sections:

1. "Setting Up Point-by-Point Optimizations" on page 6-8 describes steps to set up point-by-point optimizations with links to instructions.
2. "Optimization Wizard" on page 6-8. You use the Optimization Wizard to create an optimization, including choosing your algorithm, algorithm options, and free variables. You can set up objectives and constraints either in the wizard or from the main Optimization view.
3. After using the Optimization Wizard, you can use the optimization toolbar for common tasks for setting up optimizations. See "Optimization View Toolbar" on page 6-15.
4. "Setting Up Sum Optimizations" on page 6-16 describes steps to set up sum optimizations with links to instructions, including how to use the Number of Values length controls.

5 “Parallel Computing in Optimization” on page 6-25 describes how to set up your optimization to use distributed computing for running.

Setting Up Point-by-Point Optimizations

Use the following process to set up a point-by-point optimization:

1 Use the “Optimization Wizard” on page 6-8 to create your optimization.

It is optional whether you set up your objectives in the wizard or later in the Optimization view.

2 For simple model constraints it is optional whether you set them up in the wizard or later in the Optimization view. To apply other types of constraints (more flexible model constraints, linear, ellipsoid, 1–D table, 2–D table, and range) you must use the Optimization view. See “Edit Constraint” on page 6-39 for details of all these constraints.

3 Set variable values for the points where you want the optimization to run. See “Defining Variable Values” on page 6-27. You can enter values manually or by importing from data sets or the output of existing optimizations.

4 Run the optimization. See “Running Optimizations” on page 6-44.

5 View the results. See “Optimization Output Views” on page 6-61.

Optimization Wizard

You use the Optimization Wizard to:

1 Choose algorithm

2 Set up free variables, objectives, and constraints options — “Optimization Wizard Step 2” on page 6-10

3 Select free variables — “Optimization Wizard Step 3” on page 6-12

The last 3 steps you can do in the wizard or in the Optimization view:

4 Set up objectives — “Optimization Wizard Step 4” on page 6-13

5 Set up model constraints — “Optimization Wizard Step 5” on page 6-14

6 Set up data sets (user-defined optimizations only) — “Optimization Wizard Step 6” on page 6-15

Step 1. First you must choose your algorithm. The first screen of the Optimization Wizard is shown below.

The first four algorithm choices in the list are standard routines you can use for constrained single and multiobjective optimization.

- fgoalattain is a single-objective optimization subject to constraints. This function uses the MATLAB fmincon algorithm from the Optimization Toolbox™ product.
- NBI stands for Normal Boundary Intersection algorithm, which is multiobjective and can also be subject to constraints.
- ga and patternsearch are only available if you have the Genetic Algorithm and Direct Search Toolbox™ product installed.
  - ga stands for Genetic Algorithm, for single-objective optimization subject to constraints. This function uses the MATLAB ga algorithm from the Genetic Algorithm and Direct Search Toolbox product. See “Getting Started with the Genetic Algorithm”.

- patternsearch is another algorithm for single-objective optimization subject to constraints, from the Genetic Algorithm and Direct Search Toolbox product. See “Getting Started with Direct Search”.

In many cases these standard routines are sufficient to allow you to solve your optimization problem. Sometimes, however, you might need to write a customized optimization algorithm; to do this you can use the supplied template to modify for your needs. Any optimization functions that you have checked into CAGE appear in this list. See “User-Defined Optimization” on page 6-128 for information. The Worked Example option is designed to show you how to use the modified template. For step-by-step instructions, see the optimization tutorial section “Worked Example Optimization” in the Getting Started documentation.

Note If you choose a user-defined optimization function at step 1, all choices in subsequent steps depend on the settings defined by that function. When writing user-defined optimizations you can choose to set predetermined algorithm options or allow the user to make selections on any subsequent screen of the Optimization Wizard.

Optimization Wizard Step 2

Here you select algorithm options for numbers of free variables, objectives, and constraints. The optimization tries to find the best values of the free variables. The options available depend on your selected algorithm.

- If in step 1 you select the foptcon algorithm and click Next, you get the following choices:

The foptcon algorithm can only have a single objective, so this control is not enabled. Choose the number of free variables and constraints you require. You can also add constraints later.

- If in step 1 you select the algorithm NBI, and click Next , you see this:

NBI must have a minimum of two objectives, and you can choose as many free variables and constraints as you like. You can add constraints later if required.

Click Next to proceed to setting up free variables.

Optimization Wizard Step 3

You must select variables to link with the free variables used in your optimization.

Use this screen to associate the variables from your CAGE session with the free variable(s) you want to use in the optimization. Select the correct pair in the right and left lists by clicking, then click the large button as indicated in the Þgure.

Once you have assigned your free variables here you can either click Next or Finish. This also applies to all later steps in the Optimization Wizard.

- If you click Next you proceed to further screens of the Optimization Wizard where you can set up objectives and constraints.

- If you click Finish you return to the Optimization view in CAGE. You can set up your objectives and constraints from the Optimization view instead of using the Optimization Wizard. You cannot run your optimization until objectives (and constraints if required) have been set up.

Optimization Wizard Step 4

You can set up your objectives here or you can set them up at the Optimization view in CAGE. See “Edit Objective” on page 6-37.

Here you can select which models from your session you want to use for the optimization, and whether you want to maximize or minimize the model output. The foptcon algorithm is for single objectives, so you can only maximize or minimize one model. The NBI algorithm can evaluate multiple objectives. For example, you might want to maximize torque while minimizing NOX emissions. Remember you can also de Þne constraints later, for example, using emissions requirements.

You can also include ’helper’ models in your user-de Þned optimizations, so you can view other useful information to help you make optimization decisions (this is not enabled for NBI or foptcon).

- Click Next to proceed to setting up constraints.

- Click Finish to complete the Optimization Wizard and return to the Optimization view. Note you can only set up point objectives in the wizard, but you can also set up sum objectives in the main Optimization view. See “Objectives and Constraints” on page 6-36.

Optimization Wizard Step 5

You can use models to define constraint regions that restrict free variables. If you want to use constraints you can select them here, or add them in the Optimization view in CAGE. You can also add other types of constraints in the Optimization view. See “Edit Constraint” on page 6-39.

Select a model for each constraint by selecting a CAGE model and a model constraint and clicking the button to match them up.

For each constraint, either:

- Enter a value in the edit box to define the bound. Select the operator to define whether the optimization output should be constrained to be greater than or less than the value.
- Alternatively, select the radio button to use the Boundary of model as the constraint.

- Click Finish to complete the Optimization Wizard and return to the Optimization view.
- You can only click Next to proceed to setting up any data sets if required by your user-defined optimization.

Optimization Wizard Step 6

If your user-defined optimization allows you to add a data set you can select it on step 6 of the Optimization Wizard. You can use data sets to evaluate models over a different set of operating points during an optimization run. As an example, you could run an optimization at the points (N1, L1), (N2, L2), but an important quantity to monitor and possibly act upon is, say, temperature at points (N3, L3), (N4, L4). You can monitor the temperature at these points by using data sets, to help you select optimization results. You can set up data sets in Step 6 of the wizard or in the Optimization view in CAGE (select Optimization > Edit Data Sets).

Data sets are not enabled for foptcon and NBI optimizations.

Click Finish to return to the Optimization view in CAGE. Your new optimization appears as a new node in the tree pane on the left, and the setup details appear on the right.

Optimization View Toolbar

Common tasks are available in the toolbar:

- Add Objective — Adds an objective to your optimization (if enabled; remember foptcon can only have a single objective). You must double-click the new objective to open the Edit Objective dialog box, select a model, and set whether to maximize or minimize. See “Edit Objective” on page 6-37.
- Add Constraint — Adds a constraint to your optimization. You must double-click the new constraint (in the list of constraints) to open the

Constraint Editor and set up the constraint. See “Edit Constraint” on page 6-39

- Import from a data set, import from optimization output, import from table grid, import from table values — You can use these to populate the Variable Values panes by importing values — See “Defining Variable Values” on page 6-27.
- Set Up Optimization, Set Up and Run Optimization — Both these options open the Optimization Parameters dialog box, where you can change optimization settings such as tolerances and number of solutions. When you close the dialog box the settings are saved (and the optimization runs in the case of Set Up and Run). See “Using the Optimization Parameters Dialog Box” on page 6-46.
- Run Optimization — Starts the optimization. See “Running Optimizations” on page 6-44.

Setting Up Sum Optimizations

CAGE can solve sum type optimizations. These optimizations find the optimal settings of control parameters at several operating points simultaneously. Sum optimizations are useful for solving drive cycle problems where the constraints have to be applied across the whole cycle, e.g. weighted engine out brake specific NOx <= 3 g/kWh.

To set up a sum optimization:

1 Use the first 3 steps of “Optimization Wizard” on page 6-8 to create your optimization, defining the algorithm, number of objectives and free variables.

2 For simple model constraints it is optional whether you set them up in the wizard or later in the Optimization view. To apply other types of constraints you must use the Optimization view. You can apply linear, ellipsoid, 1-D table, 2-D table, and range constraints, and some constraints are specific to sum optimizations—sum constraints and table gradient constraints.

See “Edit Constraint” on page 6-39 for details of all these constraints.

3 For a sum optimization, it is highly likely that the objectives are sum objectives. For sum objectives you must configure your objectives in the

Optimization view, not in the optimization wizard. To configure the objectives to be sum objectives, follow the instructions in “Sum Objectives” on page 6-38.

4 Set variable values for the points where you want the optimization to run. See “Defining Variable Values” on page 6-27. You can enter values manually, or by importing from data sets or the output of existing optimizations.

For sum optimizations you must also use the length controls when defining the variable values. See “Using Variable Values Length Controls” on page 6-19.

5 Run the optimization. See “Running Optimizations” on page 6-44.

6 View the results (see “Optimization Output Views” on page 6-61). For descriptions of optimization output specific to sum problems, see “Interpreting Sum Optimization Output” on page 6-114.

Example Sum Optimization

The following sections describe the controls and outputs for sum optimizations using the following example problem for illustration.

Say you have created models for torque (TQ), residual fraction (RESIDFRAC) and exhaust temperature (EXTEMP) for a gasoline engine.

The inputs to these models are

- Spark advance, S
- Intake cam timing, INT
- Exhaust cam timing, EXH
- Engine speed, N
- Relative load, L

You need to set up an optimization to calculate optimal settings of S, INT and EXH for the following operating points:

N	L
1000	0.3
1100	0.2
1250	0.31
1500	0.25
1625	0.18

The objective for this optimization is:

Maximize the weighted sum of TQ over the operating points.

The constraints for this optimization are:

- Constraint 1: EXTEMP <= 1290°C at each operating point
- Constraint 2: RESIDFRAC <= 17% at each operating point
- Constraint 3: Change in INT is no more than 5.5° per 500rpm change in N and 5.5° per 0.1 change in L, evaluated over a 3-by-3 (N, L) table.
- Constraint 4: Change in EXH is no more than 5.5° per 500rpm change in N and 5.5° per 0.1 change in L, evaluated over a 3-by-3 (N, L) table.

You can use the foptcon algorithm in CAGE to solve this problem.

This example is used to explain the con trols and outputs in the following sections, “Using Variable Values L ength Controls” on page 6-19 and “Interpreting Sum Optimization Output” on page 6-114.

What Is a Run?. Sum type optimizations determine optimal settings of operating points simultaneously. Thus, one call to the algorithm determines the optimal settings of the control p arameters at each operating point.

Each call to the optimization algorithm is known in CAGE as a run . The number of runs that CAGE will perform is indicated in the Number of runs control in the Input Variable Values pane. See the next section, “Using Variable Values Length Controls” on page 6-19.

See “Algorithm Restrictions” on page 6-22 for details on the optimization algorithm restrictions in CAGE.

Using Variable Values Length Controls

You use the Input Variable Values pane to set variable values for the points where you want the optimization to run. (See “De Þning Variable Values” on page 6-27). You can enter values manua lly or by importing from data sets or the output of existing optimizations.

For sum optimizations you must also use the Number of Values length controls when de Þning the variable values.

At the optimization node the Input Variable Values pane has Number of Values controls for each free and Þxed variable. Use these controls to increase the number of operating points per optimization run. If you leave all the Number of Values set to one, each row in the values panes represents one optimization run. See “What Is a Run?” on page 6-18.

- You can edit the Number of Values directly, or you can select Optimization > Set Variable Lengths to change all variable lengths at once.
- You can quickly toggle between N runs of one point and a single run of N points (which can be used as a drive cycle for sum optimization problems) using the Optimization menu items Convert to Single Run and Convert to Multiple Runs . You can also use the Number of Values controls to de Þne your sum optimization runs.

If you increase the Number of Values of a Þxed or free variable, then the number of operating points within ea ch run increases, as shown in the following example.

The input variable values are con Þgured for the example problem, showing a single run (left column under Number of Values shows 1) of Þve operating points (as shown in th e right column under Number of Values). The

optimizer simultaneously finds the optimal settings of S, EXH and INT at all the operating points, starting at the initial values shown in the Free Variables table for each point.

The index of each operating point is indicated by the number in brackets in the right column under Number of Values, for example the third operating point is N=1250, L=0.31.

When objectives or constraints require weights or bounds you can enter them in the Input Variable Values pane. In the example problem, the objective requires specified weights for the weighted sum of torque, so the column Objective1_weights appears in the Fixed Variables pane, where you can enter weights for each point. For an example see "Setting Weights for the Sum Objective and Constraint" in the diesel case study.

You can also run a sum optimization over different sets of operating points. Consider the following example, an optimization of the weighted sum of fuel consumption over two different drive cycles.

The preceding figure shows an optimization that runs twice (Number of runs has been set to 2, and the left column under Number of Values shows 2 runs). Each run contains five operating points (as shown in brackets in the right column under Number of Values).

The optimization algorithm will be called twice (two runs). In the first run, optimal settings of S, EXH and INT will be simultaneously calculated for each point in the first drive cycle, as shown in the following table.

N	L
1000	0.3
1100	0.2
1250	0.31
1500	0.25
1625	0.18

In the second run, optimal settings of S, EXH and INT will be calculated for each point in the second drive cycle, as shown in the following table.

N	L
5000	0.55
5214	0.5
5564	0.6
5847	0.64
6000	0.7

In the previous examples, the number of values for each variable is identical. It is also possible to specify a mixture of scalars and vectors for each variable, as shown in the following example.

The Number of Values controls are independent for each variable. In the preceding figure:

- S Number of Values = 1
- EXH Number of Values = 5
- INT Number of Values = 1

In this case, the single initial value of S is used for every drive cycle point in the optimization, and similarly for INT (and the optimizer will return a single value for S and INT for the run).

Algorithm Restrictions

Each run of a CAGE Optimization makes a call to the algorithm you have chosen to use. This algorithm needs to evaluate the objectives and constraints (probably several times) to allow it to determine the optimal settings of the free variables. Optimization algorithms typically have restrictions on the number of objective and constraint outputs they can handle. The following table details the restrictions on the two algorithms provided in CAGE.

Algorithm Name	Objectives	Constraints
Foptcon	One output	Any number of outputs
NBI	Two or more outputs	Any number of outputs

When each objective and constraint is evaluated during a run, the number of outputs it returns depends on the maximum number of values of all of its inputs. The following table details the number of outputs each objective type returns as a function of the maximum number of values of all of its inputs.

Objective Type	Maximum Number of Values of All Inputs to the Objective	Number of Outputs	Reason
Point	N	N	A point objective is evaluated at each operating point within a run, and all the values are returned.
Sum	N	One	A sum objective evaluates a model at every operating point

Objective Type	Maximum Number of Values of All Inputs to the Objective	Number of Outputs	Reason
			and returns one value, which is the weighted sum of the model evaluations.

Similarly, the following table details the number of outputs each constraint type returns as a function of the maximum number of values of all of its inputs.

Constraint Type	Maximum Number of Values of All Inputs to the Constraint	Number of Outputs	Reason
Linear	N	N	These constraints are evaluated at every operating point within a run, and all values are returned.
Ellipsoid	N	N	
1D Table	N	N	
2D Table	N	N	
Model	N	N	
Range	N	0, N or 2N	A range constraint evaluates an expression at each operating point within a run. The constraint returns two values for each point, the distance from the lower and upper bound. In this case 2N outputs are returned. If one of the bounds is infinite, then only the distance to the finite bound is returned for each point, and N outputs are returned. If both bounds are

Constraint Type	Maximum Number of Values of All Inputs to the Constraint	Number of Outputs	Reason
			in Þnite then 0 outputs will be returned.
Sum	N	1	A sum constraint evaluates a model at every operating point and returns the difference between the weighted sum of the model and a bound.
Table	N	>=8 (dependent on settings)	A table gradient constraint constrains the gradient of a free variable over a grid. The number of outputs returned depends on the dimensions of the grid.

You can use these three tables to check whether the problem set up satis Þes the algorithm restrictions. As an example, the following table checks whether the example problem (detailed in “Example Sum Optimization” on page 6-17) satis Þes the restriction of the algorithm chosen to solve it, `foptcon` .

Objective	Maximum Number of Values of All Inputs	Number of Outputs
Weighted sum of TQ over the drive cycle points	5	1 (using the Objective table)

Constraint	Maximum Number of Values of All Inputs	Number of Outputs
EXTEMP <= 1290°C at each drive cycle point	5	5 (using the Constraint table)
RESIDFRAC <= 17% at each drive cycle point	5	5 (using the Constraint table)
Change in INT is no more than 5.5° per 500 rpm and 5.5° per 0.1 change in L	5	24 (this value is the number of table gradient constraint outputs generated from a 3-by-3 table)
Change in EXH is no more than 5.5° per 500 rpm and 5.5° per 0.1 change in L	5	24 (this value is the number of table gradient constraint outputs generated from a 3-by-3 table)

Thus, the example problem has 1 object ive output and 58 constraint outputs. This satis Þes the restrictions of the foptcon algorithm and so the algorithm can be used.

Parallel Computing in Optimization

If you have the Parallel Computing To olbox™ product available, you can distribute optimization runs to a clu ster of computers. The optimization runs are then executed in parallel. This option can signi Þcantly reduce the computation time for larger problems where each run is taking a lot longer than the time it takes to send the problem to another computer.

This functionality only appears in the menu if you have the Parallel Computing Toolbox product installed.

To use distributed computing in your optimizations:

1 You Þrst need to set up a con Þguration that de Þnes a scheduler for distributed computing.

After you have set up a configuration, in CAGE select Optimization > Distributed Computing > Select Scheduler. The Parallel Computing Toolbox Scheduler dialog box appears. Select the configuration in the list that defines your scheduler, and click OK. You only need to do this once per user per machine.

2 To use distributed computing, select Optimization > Distributed Computing > Distribute Runs. A tick appears next to the menu item, and the Optimization Information pane shows Distributed runs: On. This setting is saved with your optimization. If you try to run the same optimization on a machine without distributed computing you see a warning.

3 If your optimization requires additional files (such as user defined optimization scripts, function model M-files, user-defined models) you must also distribute these to the workers. To specify these, select Optimization > Distributed Computing > Set Job Parameters. In the dialog box, add files and paths required on the workers. Paths must be relative to the worker.

4 When you run the optimization, each run is performed on a worker. Running the optimization creates a distributed computing job, that distributes a task for each run.

CAGE displays a modal status dialog box, displaying progress messages until the job is completed. If the job is being held in the scheduler's execution queue, you see the message Waiting for job to be started. While the job runs, the progress bar tells you how many tasks are complete and how many tasks are currently running.

For more information about these terms and settings, see Parallel Computing Toolbox on the MathWorks Web site.

Note Opening matlabpool may prevent other jobs (e.g., distributed optimizations in CAGE) from being processed. See "Parallel Model Building".

Defining Variable Values

In this section...

“What Are Variable Values?” on page 6-27
“DeÞne Variables Manually” on page 6-27
“Import from a Data Set” on page 6-29
“Import from Output” on page 6-31
“Import from Table Grid” on page 6-34
“Import from Table Values” on page 6-35

What Are Variable Values?

In the optimization view, you can use the Variable Values panes to de Þne a set of operating points for the optimization. You do not have to choose a set of operating points; if you do not, howev er, the optimization runs at a single point of your choosing (the set po ints of variables is the default).

Running the optimization requires the selected models to be evaluated (many times over) and hence values are requi red for all the model input factors. The default values for the Þxed variables are their set points, as shown in the Fixed Variables pane. You chose one or more free variables, so the optimization chooses different values for those free variables in trying to Þnd the best value of the objectives. The default initial value for a free variable is the set point, as shown in the Free Variables pane.

To deÞne the set of operating points for the optimization, you can de Þne variables manually, or you can import values from these sources: data set, optimization output, table grid, or table values.

Define Variables Manually

To deÞne values manually:

1 In the Input Variable Values pane, increase the Number of runs . New rows appear for both Þxed and free variables, all containing the default

set point values of each variable. Each row de Þnes an operating point for an optimization run.

2 Edit the values in the Fixed Variables pane to deÞne the points where you want to run the optimization.

- You can copy and paste values from other parts of CAGE (existing optimizations or data sets etc.), or from the Help Browser or other documents.
- You can select Optimization > Import From Data Set if you have suitable variables to import.
- You can select Optimization > Import From Output if you have suitable optimization outputs.

An example is shown in the following Þgure.

3 Edit the values in the Free Variables pane in a similar way, if you want to deÞne the starting values of the free variables, or you can leave these at the default.

- For foptcon optimizations you can specify a number of initial starting values per run, see “foptcon Optimization Parameters” on page 6-46.
- If you wish to restrict the range of the free variables, you can select Optimization > Edit Free Variable Ranges . The default is the range of the variable as de Þned in the Variable Dictionary.

4 Use the right-click context menu to duplicate or delete runs, or select Fill All Runs to copy the selected run’s values to all other runs.

The Number of Values controls are for sum optimizations. See “Setting Up Sum Optimizations” on page 6-16.

Import from a Data Set

1 Select Optimization > Import From Data Set (or use the toolbar button) to deÞne the operating points for an opt imization from a data set, if you have suitable variables to import. The Import From Data Set dialog box appears.

2 Select a data set.

3 Select data set columns to import.

4 Choose whether you want a run per data set row (alter number of runs), or each imported variable to have th e same length as the number of data set rows (alter length). For information on altering the length of variables

(for sum optimizations only), see “Using Variable Values Length Controls” on page 6-19.

5 Click OK to import the variable values.

Import from Output

1 Select Optimization > Import From Output to import starting values from the output values of a previous optimization. The Import From Output dialog box appears.

2 Select the desired optimization output.

3 Select the columns from the output you want to import.

4 Choose the runs from the optimization output that you want to use. The Selection within output controls allow you to choose a subselection. If the number of values per run differs between current inputs and selected outputs, the inputs are altered to match.

- Select the option button All to import all runs.
- Select the option button Selection to import a subset of runs. You can enter a vector specifying the runs you want to import (e.g., 1 3 6:9), or click the button Select in Table to open a dialog box and select runs manually.
- Select the option button Acceptable to use only the runs with a selected Accept check box. See “Using Acceptable Solutions” on page 6-63. Click the button Select in Table to open a dialog box and view or edit the selection.
- Select the option button Unacceptable to use only the runs without a selected Accept check box. Click the button Select in Table to open a dialog box and view and edit the selection.
- For multiobjective optimizations you can choose to use the selected solutions or a solution number.

5 Use the Import options buttons to choose whether you want a run per output row (alter number of runs), or each imported variable to have the same length as the number of output rows (alter length).

If you click the button Select in Table you see the following dialog box.

Highlight cells in the table (Shift+click , Ctrl+click , or click and drag) to select runs to import.

If you chose a subselection on the parent dialog box (e.g., a vector of runs or an acceptable status), the table appears pre Þltered with runs selected by those choices. You can Þlter again for acceptable status on this dialog box: select Acceptable or Unacceptable from the drop-down list and click the Select button.

If there are multiple solutions in the output you can browse them with the Solution controls.

When you are satis Þed with the selected runs, click OK to return to the Import From Output dialog box. Click OK to import the runs.

Import from Table Grid

1 Select Optimization > Import From Table Grid to import starting values from the breakpoint values of a table. The Import From Table Grid dialog box appears.

2 Select the desired table in the list.

3 Use the Import options buttons to choose whether you want a run per table cell (alter number of runs), or each imported variable to have the same length as the number of table cells (alter length).

4 Click OK .

When you click OK , values for each table cell are imported into the optimization input variable values pane, e.g., for a 10 by 10 table, 100 starting points are imported.

Import from Table Values

1 Select Optimization > Import From Table Values to import starting values from the evaluation of a table. The Import From Table Values dialog box appears.

2 For each input you want to import, select the appropriate table from the Fill Input With list.

The check box for an input is automatically selected when you select a table for it.

You cannot choose to fill an input with a table that depends on it.

3 Click OK.

When you click OK, your selected optimization inputs are replaced with an evaluation of the table at the current optimization values. Other inputs are not altered.

Objectives and Constraints

In this section...

"Overview of Objectives and Constraints" on page 6-36

"Edit Objective" on page 6-37

"Edit Constraint" on page 6-39

Overview of Objectives and Constraints

You can set up objectives and constraints from the main CAGE Optimization view, as well as within the Optimization Wizard.

You can perform the following tasks by using the right-click context menu or Optimization menu (if allowed by the algorithm— foptcon can only have a single objective):

- You can Add , Edit , Rename , or Delete objectives and constraints.
- For objectives, if your objective model has a boundary model, you can select Add modelname Boundary to Constraints . This shortcut allows you to set up a boundary constraint without needing to open the Edit Constraint dialog box.
- For constraints, you can select Disable to remove constraints without deleting them, and use Enable to reapply them.

Double-click to edit existing objectives and constraints in the Objectives or Constraints panes. This opens the Edit Objective or Edit Constraint dialog boxes.

You can run two types of optimizations, point optimizations and sum optimizations. Point optimizations look for the optimal values of each objective function at each point of an operating point set. A sum optimization finds the optimal value of a weighted sum of each objective function. The weighted sum is taken over each point, and the weights can be edited. For an example, see the tutorial section "Sum Optimization" in the Getting Started documentation.

You need to use the Edit Objective dialog box and Edit Constraint dialog box to set up sum objectives and model sum constraints. You must do this to run weighted sum optimizations. You cannot set these up from the Optimization Wizard.

You can also set up linear, 1- and 2-D table, and ellipsoid constraints in the Edit Constraint dialog box, as for designs in the Model Browser part of the Model-Based Calibration Toolbox product.

Edit Objective

Double-click or right-click objectives to open the Edit Objective dialog box.

You can select Point objective or Sum objective from the Objective type drop-down menu. Use sum objectives only for weighted sum optimizations; otherwise, use point objectives.

You can rename the objective by editing the Objective name edit box, to aid analysis in the Optimization views. This may be disabled for user-defined optimizations.

Point Objectives

The preceding example shows the point objective controls. Select which models from your session you want to use for the optimization, and whether you want to maximize or minimize the model output. The foptcon algorithm is for single objectives, so you can only maximize or minimize one model. The NBI algorithm can evaluate multiple objectives. For example, you might want to maximize torque while minimizing NOX emissions.

You can also include 'helper' models in your user-defined optimizations, so you can view other useful information to help you make optimization decisions (this is not enabled for NBI or foptcon).

These are the same options you can choose in the Optimization Wizard. See "Optimization Wizard Step 4" on page 6-13.

Sum Objectives

For weighted sum optimizations, the objectives are typically sum objectives. See the following example.

As for point objectives, select which models from your session you want to use for the optimization, and whether you want to maximize or minimize the model output.

You can edit weights in the Optimization view, to make certain operating points more important, giving more flexibility to solutions for other points. You can edit the weights in the Fixed Variables pane. This is the same process as selecting weights for the Weighted Pareto View. See “Weighted Objective Pareto Slice” on page 6-74.

For a tutorial example of a sum optimization, see “Sum Optimization” in the Getting Started documentation.

Edit Constraint

You can rename the constraint by editing the Constraint name edit box, to aid analysis in the Optimization views. This may be disabled for user-defined optimizations.

Select a Constraint type in the drop-down menu. The first four choices are the same as the following design constraint types:

- “Linear Constraints”
- “Ellipsoid Constraints”
- “1-D Table Constraints”
- “2-D Table Constraints”

These are the same constraints you can apply to designs in the Model Browser part of the Model-Based Calibration Toolbox product.

In the context of optimization you can select constraint inputs on the additional Inputs tab. You can select any variable or model as an input into constraints. The default selects the free variables where possible. Models are treated as nonlinear functions, so if you choose to feed a model into a linear constraint it will make that constraint nonlinear. You are not able to access it as a linear constraint in user-defined optimization scripts.

For optimization constraints you can also select the following constraint types:

- “Model Constraints” on page 6-40
- “Range Constraints” on page 6-41
- “Sum Constraints” on page 6-42
- “Table Gradient Constraints” on page 6-42

Model Constraints

To construct a model constraint:

1 Select an Input model in the left list.

2 You can use the Evaluate quantity drop-down list to choose Evaluation value, Boundary constraint, or PEV value (model prediction error variance) to define your constraint.

3 Choose the appropriate option button to either enter a value in the Constant edit box, or to select a CAGE item from the list of models or variables.

4 Select the Constraint type operator to define whether the optimization output should be constrained to be greater than or less than the constant or item value specified on the right.

5 Check the displayed Constraint description, and click OK.

The model constraint settings are shown in the following figure.

Range Constraints

You can specify an upper and lower bound to constrain expressions (which can be variables, models or tables). You can specify bounds with constants, vectors, variables, models, or tables.

1. Select a CAGE item to constrain on the Bound Expression tab. Use the drop-down menu to switch between variables, models, or tables, and then select the item to constrain. For appropriate models you can also choose to constrain either the PEV or evaluation value.
2. On the Lower Bound tab, select an option button to choose whether to use a constant, vector, or CAGE item to specify the bound.
   - For constants, enter a value.
   - For vectors, you can enter the lower bound for each point in the Input Variable Values pane in the Optimization view after you close the Edit Constraint dialog box.

- For CAGE items, use the drop-down menu to switch between variables, models, or tables, and then select the item to specify the lower bound. For appropriate models you can also choose to use either the PEV or evaluation value.

3 Specify the upper bound on the Upper Bound tab in the same way as you specified the lower bound on the Lower Bound tab.

4 Check the displayed Constraint description, and click OK.

For a detailed explanation of range constraint outputs, see “Range Constraint Output” on page 6-84.

Sum Constraints

Use these for weighted sum optimizations. Choose a model, constraint bound value and an operator.

You can have a mixture of point and sum constraints.

See the tutorial “Sum Optimization” in the Getting Started documentation for a step-by-step example, and for descriptions of optimization output specific to sum problems, see “Interpreting Sum Optimization Output” on page 6-114.

Table Gradient Constraints

Table Gradient constraints allow you to constrain the gradient of a free variable over a grid of fixed variables. These constraints are most useful in ’sum’ problems. Unless you are using a user-defined optimization, you should normally use a sum objective (and therefore runs normally have multiple values).

1 Select a free variable to constrain.

2 Specify one or two fixed variables, and a grid of points either manually or by selecting table axes.

3 Enter values in the edit boxes to specify the maximum change in the free variable per amount of fixed variable change between cells. For example, enter 5 and 1000 to specify 5 degrees maximum change in cam angle per 1000 rpm.

4 Check the displayed Constraint description , and click OK .

For a detailed explanation of table gradient outputs, see “Table Gradient Constraint Output” on page 6-120.

Running Optimizations

When you click Finish to complete the Optimization Wizard, you return to the Optimization view in CAGE. Your new optimization appears as a new node in the tree pane on the left, and the setup details appear on the right. An example follows:

If your optimization is ready to run you can click Run Optimization in the toolbar to proceed. You may want to define variable values before running the optimization. If you need to set up any objectives or constraints Run will not be enabled. If your optimization is ready to run you can also click Set Up and Run Optimization if you want to change algorithm-specific settings such as number of required solutions and tolerances for termination.

- If you click Set Up and Run Optimization, you can change settings in the Optimization Parameters dialog box. Then when you click OK the optimization process begins. See "Using the Optimization Parameters Dialog Box" on page 6-46.
- If you click Run Optimization instead, you do not see the optimization settings, but go straight to running the optimization.

You will see a progress bar as the optimization proceeds. When it is finished, a new Output node appears under your Optimization node in the tree and the view automatically switches to this node where you can analyze the results. An example tree is shown in the following figure. See "Optimization Output Views" on page 6-61.

Using the Optimization Parameters Dialog Box

In this section...

"Overview of the Optimization Parameters Dialog Box" on page 6-46
"foptcon Optimization Parameters" on page 6-46
"NBI Optimization Parameters" on page 6-49
"GA Optimization Parameters" on page 6-54
"Pattern Search Optimization Parameters" on page 6-57
"Scale Optimization" on page 6-60

Overview of the Optimization Parameters Dialog Box

The settings in the Optimization Parameters dialog box are algorithm specific.

If you edit these settings and later want to return to the defaults, select Optimization > Reset Parameters. If you add parameters to user-defined optimization scripts, you may need to use this reset option to make all new parameters appear in the dialog box.

foptcon Optimization Parameters

The foptcon optimization algorithm in CAGE uses the MATLAB fmincon algorithm from the Optimization Toolbox product. foptcon wraps up the fmincon function so that you can use the function for maximizing as well as minimizing. For more information, see the fmincon reference page in the Optimization Toolbox documentation, fmincon.

- Displa y — choose none , iter , or final . This setting determines the level of diag nostic information display ed in the MATLAB workspace.
  - none — No information is displayed.
  - iter — Displays statistical information every iteration.
  - final — Displays statistical information at the end of the optimization.
- Maximu m iterations — Choose a positive integer.

  Maximu m number of iterations allowed
- Maximu m function evaluations — Choose a positive integer.

  Maximu m number of function evaluations allowed

- Variable tolerance — Choose a positive scalar value.

  Termination tolerance on the free variables

- Function tolerance — Choose a positive scalar value.

  Termination tolerance on the function value

- Constraint tolerance — Choose a positive scalar value.

  Termination tolerance on the constraint violation

- Minimum/maximum change in variables for gradient

  Choose a positive scalar to control the input step size that is taken when gradients are being calculated. The default settings should work for the majority of problems.

- Number of start points — Choose a positive integer, N. (N-1) start points per run are generated in addition to the starting value specified in the Input Variable Values pane.

  The optimization runs from each of the N start points (possibly subject to feasibility, see Run from feasible start points only option) and the best solution is chosen.

  The N-1 extra start points are generated as follows:

  1 Generate a 10000 point Halton set design, D, over the free variables.

  2 Evaluate the objectives and constraints over D.

  3 Return the N-1 feasible points with the lowest objective value.

    If there are not N-1 feasible points, fill the remaining starting values with the points with the lowest maximum constraint violation.

Note For point optimization problems, it is strongly recommended that you set Number of start points to either 1 or 2.

- Run from feasible start points only — Select this option to terminate all runs that start with an initial value that does not satisfy the constraints. If this condition is not met this is reported in Output message, in the Solution Information pane of the Optimization Output view.

- Interface version — This option is only enabled when a user-defined optimization script does not specify a version to use. Some existing user-defined optimization scripts may require setting the interface version as 2 or 3, according to the toolbox version. Version 3 is preferable, but may not work with all old scripts. See setRunInterfaceVersion for details.

NBI Optimization Parameters

- "Background on the NBI (Normal Boundary Intersection Algorithm)" on page 6-50
- "NBI Options" on page 6-53
- "NBI Output Messages" on page 6-54

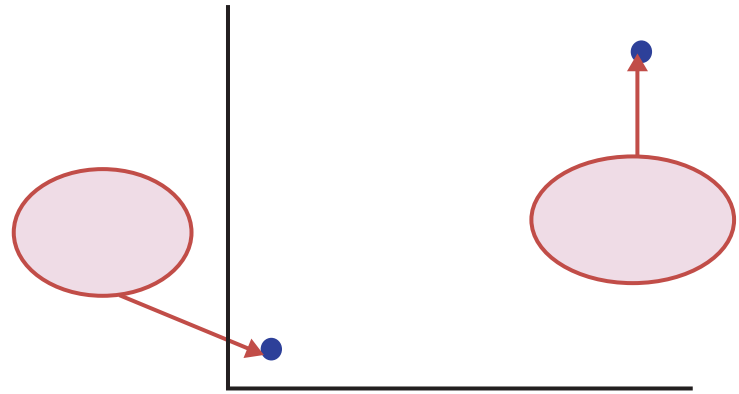
The example following shows the NBI options in the Optimization Parameters dialog box.

Background on the NBI (Normal Boundary Intersection Algorithm)

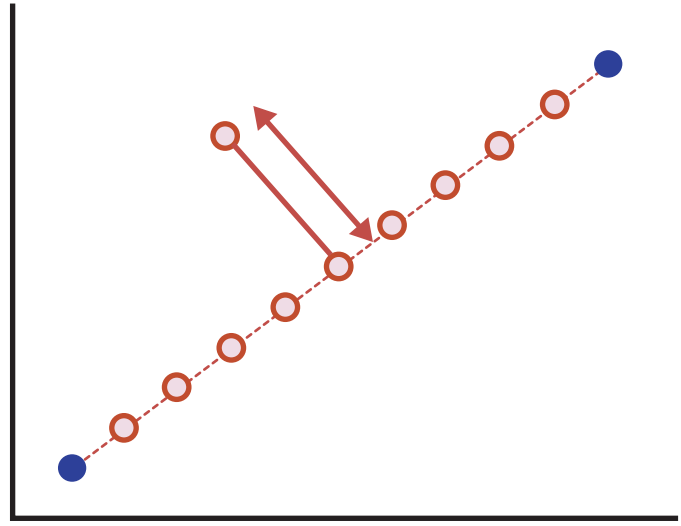
To understand the options for the NBI al gorithm, some limited understanding of the algorithm is required. For more information on the NBI algorithm, see the following reference:

Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria O ptimization Problems, I. Das and J.E. Dennis, SIAM J. on Optimization. 8(3), 631-657 (1998).

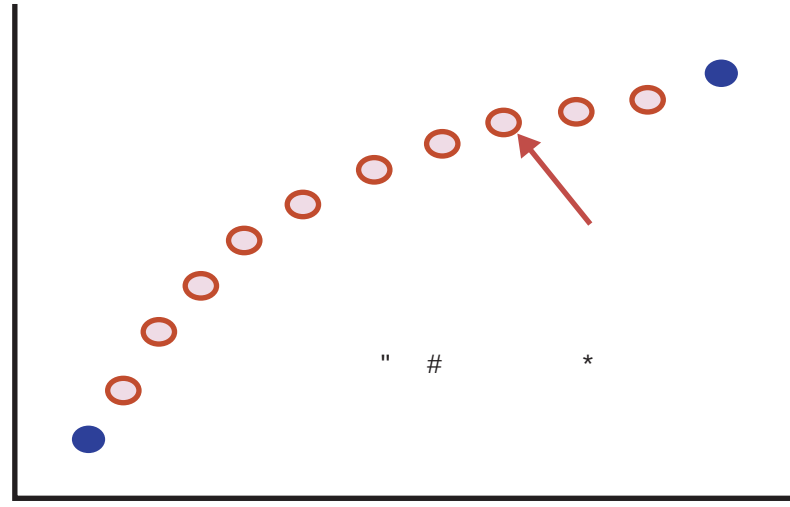
The NBI algorithm is performed in two steps. The first step is to find the global of each objective individually. This is called the shadow minima problem, and is a single-objective problem for each objective function. The MATLAB routine fmincon is used to find these. Once these are found, they can be plotted against each other. For example, consider an NBI optimization that simultaneously maximizes TQ and minimizes NOX emissions. A plot of the against each other might resemble the following.



The second step is to find the "best" set of tradeoff solutions between your objectives. To do this, the NBI algorithm spaces Npts start points in the (n-1) hypersurface, S, that connects the shadow. In the above example, S is the straight line that connects the points N and T. For each of the Npts points on S, the algorithm tries to maximize the distance along the normal away from this surface (this distance is labeled L in the following figure). This is called the NBI subproblem. For each of the points, the NBI subproblem is a single-objective problem and the algorithm uses the MATLAB fmincon routine to solve it. This is illustrated below for the TQ-NOX example.



The Þgure above shows spacing of the points between the along the (n-1) surface. The algorithm tries to maximize the distance L along the normal away from the surface. The following Þgure shows the Þnal solution found by the NBI algorithm.



NBI Options

- Tradeoff points per objective pair (Np)

  The number of tradeoff solutions betw een your objectives that you want to Þnd, Npts , is determined by the following formula:

  where

  - Np is the number of points per objective pair.
  - n is the number of objective functions.

Note the following:

- For problems with two objectives (n = 2),

- For problems with three objectives (n = 3),

- Shadow minima options and NBI subproblem options

  The NBI algorithm uses the MATLAB fmincon algorithm to solve the shadow minima problem and the NBI subproblems, the options available are similar to those for the foptcon library function. For more information on these options, see the previous section, “foptcon Optimization Parameters” on page 6-46.

NBI Output Messages

The NBI algorithm provides exit messages that can be seen in the Optimization output view, in the Solution Information pane, for the currently selected run. Check these messages to check for problems with your optimization.

All possible exit flags and messages are shown in the following table.

Exit flag	Message
6	The shadow minima do not differ from one another. This suggests that all objectives can be minimized simultaneously. Check that the objectives are competing or alter tolerances.
1	All shadow and NBI subproblems converged to a solution.
0	At least one of the NBI subproblems is infeasible.
0	The maximum number of function evaluations was reached in at least one of the shadow or NBI subproblems.
-1	Optimization terminated prematurely by the user.
-2	At least one of the shadow problems is infeasible.
-7	At least one of the Pareto solutions is dominated.

GA Optimization Parameters

The ga optimization algorithm in CAGE uses the MATLAB ga algorithm from Genetic Algorithm and Direct Search Toolbox product. In CAGE, ga wraps up the ga function from this toolbox so that you can use the function for

maximizing as well as minimizing. If you have Genetic Algorithm and Direct Search Toolbox product installed, see "Getting Started with the Genetic Algorithm".

- Display — choose none, iter, final, or diagnose. This setting determines the level of diagnostic information displayed in the MATLAB workspace.

  - none — No information is displayed.
  - iter — Displays statistical information every iteration.
  - final — Displays statistical information at the end of the optimization.
  - diagnose — Displays information at each iteration. In addition, the diagnostic lists some problem information and the options that have been changed from the defaults.

- Crossover function — Choose a function to use to generate new population members from the existing GA population by crossover. For more information on each function, see the Crossover Options section in the Genetic Algorithm and Direct Search Toolbox documentation. It is recommended not to use a heuristic crossover function for nonlinearly constrained problems.
- Crossover fraction — Choose a scalar in the range [0 1]. This parameter specifies the fraction of the next generation, other than elite children, that is produced by crossover.
- Mutation function — Choose a function to use to generate new population members from the existing GA population by mutation. The fraction of the next generation, other than elite children, that is produced by mutation is (1 minus Crossover fraction). Also, for nonlinearly constrained problems, the mutation function must be set to adaptfeasible.
- Selection function — Choose a function to use to select the population members that will be used as the parents for the crossover and selection functions.
- Population size — Choose a positive integer value. Number of population members used by the algorithm. See the Genetic Algorithm and Direct Search Toolbox documentation for guidelines on setting the population size.
- Generations – Choose a positive integer value. The algorithm stops when the number of generations reaches the value of Generations.
- Hybrid function — Choose an optimization function that will run after the GA has terminated to try to improve the value of the objective function. Note that if the algorithm has nonlinear constraints, the hybrid function cannot be fminunc or fminsearch. If either of these algorithms is selected in this case, the hybrid algorithm switches to fmincon.

- Stall generations — Choose a positive integer value. The algorithm stops when the weighted average change in the objective function over Stall generations is less than Function tolerance.
- Stall time limit – Choose a positive scalar value. The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to Stall time limit.
- Function tolerance — Choose a positive scalar value. The algorithm runs until the weighted average change in the fitness function value over Stall generations is less than Function tolerance.
- Constraint tolerance – Choose a positive scalar value. This tolerance determines whether a population member is feasible with respect to the nonlinear constraints.
- Time limit – Choose a positive scalar value. The algorithm stops after running for an amount of time in seconds equal to Time limit.

Pattern Search Optimization Parameters

The patternsearch optimization algorithm in CAGE uses the MATLAB patternsearch algorithm from Genetic Algorithm and Direct Search Toolbox product. In CAGE, patternsearch wraps up the patternsearch function from this toolbox so that you can use the function for maximizing as well as minimizing. If you have the Genetic Algorithm and Direct Search Toolbox product installed, see “Getting Started with Direct Search”.

- Display — Choose none , iter , final , or diagnose . This setting determines the level of diagnostic information displayed in the MATLAB workspace.
  - none — No information is displayed.
  - iter — Displays statistical information at every iteration.

  - final — Displays statistical information at the end of the optimization.
  - diagnose — Displays information at each iteration. In addition, the diagnostic lists some problem information and the options that have been changed from the defaults.

- Time limit – Choose a positive scalar value. The algorithm stops after running for an amount of time in seconds equal to Time limit.
- Maximum number of iterations — Choose a positive scalar value. This parameter specifies the maximum number of iterations performed by the algorithm.
- Maximum function evaluations — Choose a positive integer value. The algorithm stops if the number of function evaluations reaches this value.
- Variable tolerance — Choose a positive scalar value. The algorithm stops if the distance between two consecutive free variable values is less than the variable tolerance.
- Function tolerance — Choose a positive scalar value. The algorithm stops if the distance between two consecutive objective function values and the mesh size are both less than Function tolerance.
- Constraint tolerance – Choose a positive scalar value. Determine feasibility with respect to the nonlinear constraints.
- Mesh tolerance — Choose a positive scalar value. The algorithm stops if the mesh size is smaller than Mesh tolerance.
- Initial mesh size — Choose a positive scalar value. Sets the initial size of the mesh for the pattern search algorithm. Do not set this value too small, as insufficient size may lead to the algorithm getting trapped in local optima.
- Poll method – Choose a poll method from the drop-down list. This parameter sets the polling strategy that will be used by the pattern search algorithm. Generally, the GPSPositiveBasis2N and MADSPositiveBasis2N methods will be slower than the GPSPositiveBasisNp1 and MADSPositiveBasisNp1 methods. However, the former methods perform a more thorough search. For more information on these methods, consult the Genetic Algorithm and Direct Search Toolbox documentation.
- Search method — Choose a search method from the drop-down list. This parameter selects a function that will perform a search in addition to that

performed by the pattern search algo rithm. For automotive problems, searchlhs tends to perform well. For more information on the possible search methods, consult the Genetic Algorithm and Direct Search Toolbox documentation.

Scale Optimization

The Optimization menu contains the option to Scale Optimization Items — Select this to toggle scaling on and off. When you select scaling on, objective and constraint evaluations are (approx imately) scaled onto the range [-1 1]. With scaling off, when you run the optimi zation the objective and constraint evaluations return their raw numbers.

Try running your optimization with scaling off, which is the default setting, to see if it converges to a satisfactory solution (check the output ßags and the contour view). If your optimization solution is unsatisfactory, check to see if the objective and constra int functions have vastly different scales. In this case, try turning scaling on, because these optimization problems may bene Þt from objective and constraint evalua tions being scaled to a common scale.

The output view always shows the solutions in raw, unscaled values, whether or not you use scaling to evaluate the problem.

Optimization Output Views

In this section...

“Using the Optimization Output View” on page 6-61
“Using Acceptable Solutions” on page 6-63
“Solution Slice: Optimization Results Table” on page 6-65
“Solution Slice: Results Surface and Results Contour Views” on page 6-67
“Pareto Slice” on page 6-72
“Weighted Objective Pareto Slice” on page 6-74
“Selected Solution Slice” on page 6-76
“Objective Slice Graphs” on page 6-78
“Objective Contour Plot” on page 6-79
“Pareto Front Graphs” on page 6-80
“Constraint Slice Graphs” on page 6-81
“Constraint Summary Table” on page 6-83

Using the Optimization Output View

When you have run an optimization an Output node appears in the optimization tree and the Optimization Output views appear. Use the toolbar buttons shown in the following Þgures to determine what is displayed in the table and the graph views. The Þrst default view is the Solution Slice table and the Objective Slice Graphs.

Use these toolbar buttons or the View menu to select the following Table Views:

- “Solution Slice: Optimization Results Table” on page 6-65 and “Solution Slice: Results Surface and Results Contour Views” on page 6-67— See also “Using Acceptable Solutions” on page 6-63
- “Pareto Slice” on page 6-72

- “Weighted Objective Pareto Slice” on page 6-74
- “Selected Solution Slice” on page 6-76

Use these toolbar buttons to sele ct the following Graph Views:

- “Objective Slice Graphs” on page 6-78
- “Objective Contour Plot” on page 6-79
- “Pareto Front Graphs” on page 6-80
- “Constraint Slice Graphs” on page 6-81
- “Constraint Summary Table” on page 6-83
- Free Variable Values Table — displays the values of the free variables for the currently selected solution.
- Solution Information Table — displays information about the currently selected solution, including the Accept status, the algorithm exit ßag and exit message, and other algorithm deta ils such as the number of iterations.

  Hover the mouse pointer over the Exit message to see the whole message. This message can tell you, for example, if an foptcon optimization run terminated because no feasi ble start point was found.

You can split and add these views as in the Design, Data and Boundary Editors. Use the right-click context menu, the View menu, or the buttons in the view title bars to do so.

The last four toolbar buttons are also in the Solution menu:

- Select solution — This option is for multiobjective optimization, used for choosing your preferred solution for each operating point. See “Selected Solution Slice” on page 6-76.
- Edit pareto weights — This option is used for evaluating weighted sums. See “Weighted Objective Pareto Slice” on page 6-74.
- Export to data set — This option exports the table visible in the current view only to a new or existing data set. See “Using Optimization Output” on page 6-89.
- Fill tables using optimal solutions — This option opens the Table Filling From Optimization Results Wizard. See “Using Optimization Output” on page 6-89.
- The Solution menu also has Retain Output (also in the context menu when you right-click an optimization output node). If you select this option, the output node is retained, so if you rerun the optimization you get additional output nodes.

Note For help understanding your results, see “Analyzing Point-by-Point Optimization Output” on page 6-98 or “Interpreting Sum Optimization Output” on page 6-114.

Using Acceptable Solutions

CAGE automatically selects successful optimization solutions and highlights unsuccessful solutions for you to investigate. These selections are shown in the icons and check boxes next to the Run column in the Optimization Results table, and shown in the Results Surface and Results Contour views. You can change the selections using the check boxes for each solution, or right-click to change acceptable status of solutions in the graphical views.

You can use these selections to choose solutions within the table for use in:

- “Filling Tables from Optimization Results” on page 6-92
- “Exporting to a Data Set” on page 6-90

- Importing to other optimization star ting values: “Import from Output” on page 6-31.

Accept status is shown in the following ways:

- CAGE automatically selects the Accept check boxes for solutions where the algorithm exit ßag indicates success (>0). These solutions show a green square icon next to the check box. Typically constraints are met within tolerance.

- Solutions with a red round icon indicate that the algorithm exit ßag does not report success (<0). Some constraints may not be met.

- Solutions with an orange triangula r icon indicate that the algorithm exit ßag is zero. Some constraints may not be met. An exit ßag of zero indicates the algorithm failed because it exceeded limits on the amount of computation allowed (e.g., the algor ithm ran out of iterations or function evaluations). You could decide to accept these solutions or you could try changing tolerances and optimizing again.

- Solutions where you have altered th e check box status show an asterisk.

- Violated constraints are shown by yellow cells with cross icons in the table. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance .

  It is possible to have highlighted constraints within green accept status solutions. The algorithm can report s uccess if constraints are met within tolerance on scaled values. The const raint display applies a tolerance to raw values, and you can also edit this t olerance to help you analyze results.

If you are viewing constraints wi th more than one value and have the view set to Compact , the cell is yellow if any of the individual values are infeasible.

- You can view the algorithm output ßag in a tooltip by hovering the mouse pointer over each colored accept status icon, or click to select a solution and you can view the algorithm Exit flag , Exit message and other details in the Solution Information table.

  The icon and (editable) Accept status check box are also shown at the top right for the currently selected solution.

Note For help understanding your results, see “Analyzing Point-by-Point Optimization Output” on page 6-98.

Solution Slice: Optimization Results Table

The Solution S lice view (click) shows a table with one solution at all operating poi nts and all runs. The solution is shown in both tabular and graphical fo rms — see “Solution Slice: Results Surface and Results Contour Views” on pag e 6-67 for information on the graphical views.

The followin g example shows a Solution Slice table display.

The Solution Slice view shows a table of one solution at all operating points and all runs in the problem. For single-objective optimizations there is only one solution per operating point, so the Solution Slice is the only useful view. For multiobjective optimizations with more than one solution per run, you can scroll through the solutions using the arrows or edit box at the top.

The table shows the selected solution at all operating points. The Optimization Results pane shows the fixed variable settings, the optimal free variable settings, and the evaluation of objectives and constraints at the optimal free variable settings.

Click inside the table to make the graph views (objective slice, constraint slice and pareto front) display the selected operating point.

- The “Objective Slice Graphs” on page 6-78 show the objective functions at the operating point selected in the table, with the solution value in orange.

- If you have constraints you can also choose to display the "Constraint Slice Graphs" on page 6-81. These show the constraint functions at the selected operating point with the solution value in orange.
- If you are viewing a multiobjective optimization you can also choose to display the "Pareto Front Graphs" on page 6-80, which show the available solutions with the current selection highlighted in red.
- You can also display the "Constraint Summary Table" on page 6-83, which details the distance to each constraint edge for the selected operating point in the table. This table can be useful to see at a glance if a solution met all the constraints. If there are many constraints it can be time-consuming to use the constraint graphs to verify that the constraints are met.

Before you run an NBI optimization you can specify how many solutions you want the optimization to find, using the Set Up and Run Optimization toolbar button.

For information on selecting best solutions at each operating point for subsequent export to a data set or plotting tables, see "Selected Solution Slice" on page 6-76.

Solution Slice: Results Surface and Results Contour Views

Surface View of Optimization Results

The Results Surface view shows a 3D plot of one solution at all operating points and all runs in the problem. Use the axes popup controls to change what is plotted on each axis. You can plot the following against each other:

- Fixed variable settings
- Optimal free variable settings
- Evaluation of objectives at the optimal free variable settings

The optimization results are plotted as points, and an extrapolation surface (of the z-axis quantity as a function of the x and y-axis quantities) is also displayed.

Each optimization result is displayed using the Accept icon, as shown in the Optimization Results table:

- Successful result
- Failed result
- Problem result
- User-altered accept status.
- Currently selected result (black outline). Select results by clicking an icon in the plot or a value in the table. Changing the currently selected result in the Results Surface view also updates the result selected in the table, and updates any plots displayed in the lower half of the output view.

Left-click anywhere except an icon to rotate the plot.

Use the right-click context menu to control these options:

- Results to Display
  - All – Show all optimization results for this solution
  - Acceptable – Show only the acceptable results for this solution
  - Green – Show the results with a positive exit flag
  - Orange – Show the results with a zero exit flag

  - Red – Show the results with a negative exit ßag

- Set Acceptable — mark an optimization result as acceptable if it is currently marked as unacceptable.

  Any results whose acceptability has been changed are shown as stars in the plot

- Set Unacceptable — mark an optimization result as unacceptable.
- Extrapolate All — toggles extrapolation fro m acceptable solutions only (default) to using all res ults for extrapolation.
- Surface Options
  - Reset Axes Orientation — Reset the axes orientation to the default.
  - Show Axes Grid — Toggle whether the axes grid is displayed or not.
  - Show Axes Box — Toggle whether the axes box is displayed or not.
  - Hide Surface — Toggle whether the surface is visible or not.
  - Show Stems — Use this option to a dditionally display stems projected from the data to the surface. These stems can be useful to show the location of results that are not used i n the extrapolation and are hidden by the surface.

Contour View of Optimization Results

The Results Contour view shows a contour plot of one solution at all operating points and all runs in the problem. You can plot the following against each other: Þxed variable settings, optima l free variable settings and evaluation of objectives at the o ptimal free variable settings.

The optimization results are plotted as points in the contour plot and extrapolation contours (of the z-axis q uantity as a function of the x and y-axis quantities) are also displayed. The accept icon for each result is plotted as for the Results Surface.

Rotati on is not permitted in the contour view.

The rig ht-click context menu shares these options with the Results Surfac e view: Results to Display , Set Acceptable /Unacceptable , and Extra polate All . Some additional items for the contour view:

- Conto ur Options
  - Label Contour Lines
  - Fill C ontours

- Contour Levels — These contour options are identical to those for the objective contour view.
- Show Axes Grid — Toggle whether the axes grid is displayed or not.
- Hide Contour — Toggle whether the contour is hidden or not.
- Display Contour — Toggle whether the contour is displayed or not.

Pareto Slice

The Pareto Slice table view (click) is for multiobjective optimization where there is more than one solution at each run. The Pareto Slice shows a table of all solutions at one run; you can scroll through the runs using the arrows or edit box at the top.

To collect best solutions across different runs, you need to use the Select Solution function in the toolbar.

To select a solution for each run:

1 Enable the Selected Solutions view. Select Solution > Selected Solution > Initialize.

The Create Selected Solution dialog box appears.

The default 1 initializes the first solution for each run as the selected solution. You can edit the solution number here if you want. For example, if you select 4, solution number 4 is initialized as the best solution for every run. When you click OK, the toolbar buttons for the Selected Solution Slice view and Select Solution are enabled.

2 Decide which solution you want to use for the currently selected run. Use these tools to help you:

- Display the “Par eto Front Graphs” on page 6-80 (click in the toolbar) which s how the available solutions with the current selection highlighted in r ed.
- Use the pareto fr ont graphs together with the “Objective Slice Graphs” on page 6-78 to select the best solution for the run. If you have constraints yo u can also use the “Constraint Slice Graphs” on page 6-81 and “Constrain t Summary Table” on page 6-83 to help you decide which solution to ch oose for each run.

3 When you have decided which solution you want to use for the currently selected run, you can select it as be st by clicking Select Solution ( ) in the toolbar. You can also select best solutions in the Solution Slice view, see “Solution Slice: Optimization Results Table” on page 6-65 .

4 Scroll throu gh the runs and select a best solution for each. These selections are collect ed in the Selected Solutions Slice, where you can use them to Þll tables or ex port to a data set. You can also import them to an optimization. See “Select ed Solution Slice” on page 6-76.

Before you r un an NBI optimization you can specify how many solutions you want the op timization to Þnd, using the Set Up and Run Optimization toolbar button.

As in the other table views, you can use the Accept check boxes to choose a selection of rows within the table. In this table view, you can only use this to select solutions within a single run. Each different solution has a check box and colored icon for “Acceptable” status. There is only one exit flag per run, so all solutions have either red or green Accept status. You can override these selections using the check boxes if you want to choose solutions within a run, for use when importing to other optimizations, or for future reference. See “Using Acceptable Solutions” on page 6-63.

Weighted Objective Pareto Slice

The Weighted Objective Pareto Slice view (click) shows a weighted sum Pareto solution. This table shows a weighted sum of the objective values over all runs for each solution. For a single objective optimization there is a single cell, which is the sum of the objective across all runs.

In the following multiobjective example, the value in the Objective1 column in the first row shows the sum of the solution 1 values of the first objective across all runs. The second row shows the sum of solution 2 Objective1 values across all runs, and so on for all ten solutions in this case. This information can be useful, for example, for evaluating total emissions across a drive cycle. The default weights are unity (1) for each run.

You can change the weights; for example, if you need a weighted sum of emissions over a drive cycle, you might want to give a higher weight to the value at idle speed. You can alter weights by clicking Edit Pareto Weights ( ) in the toolbar. The Pareto Weights Editor appears.

In this dialog box, you can select objectives to sum, and select weights for any run by clicking and editing, as shown in the previous example. The same weights are applied to each solution to calculate the weighted sums. Click OK to apply new weights, and the weighted sums are recalculated.

You can also specify weights with a MATLAB vector or any column in the optimization output by selecting the other option buttons. If you select Output column you can also specify which solution; for example, you could choose to use the values of spark from solution 5 at each operating point as weights. Click Table Entry again, and you can then view and edit these new values.

Note Weights applied in the Weighted Pareto View do not alter the results of your optimization as seen in other views. You can use the weighted sums to investigate your results only. You need to perform a sum optimization if you want to optimize using weighted operating points.

The Accept check box is disabled in this view. The exit flag is the minimum of all of the runs that are summed over, so the Accept status can only go green if all runs are green.

Selected Solution Slice

In a multiobjective optimization, there is more than one possible optimal solution at each run. You can use the Selected Solutions view to collect and export those solutions you have decided are optimal at each run.

After you enable the Selected Solution view, you can use the plots and table views to help you select best solutions for each run. These solutions are saved in the Selected Solutions view. You can then export your chosen optimization output for each point from the Selected Solutions view to a data set, or use your optimization output to fill tables or import to another optimization.

You cannot select best solutions until you have enabled the Selected Solutions view.

1 Select Solution > Selected Solution > Initialize .

A dialog box called Create Selected Solution appears. The default 1 initializes the first solution for each run as the selected solution.

2 Edit the solution number in this dialog box if you want. For example if you select 4, solution number 4 is initialized as the best solution for every run. When you click OK, the toolbar buttons for the Selected Solutions view and Select Solution are enabled.

3 After you enable the Selected Solutions view, you can use the table views and the plots in the graphs (Objective Slice, Pareto Front, and Constraint Slice graphs) to help you select best solution for each run.

   a Click in the "Pareto Slice" on page 6-72 (or Solution Slice) table to select a point to display in the graphs until you can decide which solution you want for a point.

   b Click Select Solution ( ) in the toolbar to select the current solution as best.

   Repeat steps a and b until you have selected solutions for all points.

These solutions are saved in the Selected Solutions view. This view collects all your selected solutions together in one place. For example, you might want to select solution 7 for the first run, and solution 6 for the second, and so on. You can then use your chosen optimization output for each point to fill tables (see "Filling Tables from Optimization Results" on page 6-92), or choose the Export to Data Set toolbar and Solution menu option (see "Exporting to a Data Set" on page 6-90), or use these solutions as starting points in another optimization (see "Import from Output" on page 6-31).

An example of the Selected Solutions view is shown. It looks similar to the Solution Slice view, except the solution controls at the top are not enabled. You cannot change solution number in this view. The solution chosen as best (in the other views) for the currently selected run is displayed in the grayed-out edit box.

As in the other table views you can use the Accept check boxes to choose a selection of rows within the table. See "Using Acceptable Solutions" on page 6-63.

Objective Slice Graphs

The objective slice graphs are displayed by default for optimization output views, or you can select in the toolbar.

The objective slice graphs show the objective functions at the point selected in the table, with the solution value in orange. Whether the table is displaying a solution slice or pareto slice, the cell you select in the table is always displayed

in the graphs. The objective graphs sh ow cross section plots of the objective function against each free variable in the problem.

The yellow areas show a region outside a constraint tolerance (such as a boundary constraint exported from the Model Browser part of the Model-Based Calibration Toolbox p roduct, or any other optimization constraint). All constraint regions in optimization displays (as in the rest of the toolbox) are shown in yellow.

Use the right-click context menu to togg le constraint display and alter graph size.

Objective Contour Plot

The Objective Contour Plot (click) shows the contours of the objective against any p air of control parameters, at the run selected in the table, with the solution value at the center of the orange cross-hairs. Yellow areas show a region outs ide a constraint tolerance (see the following Þgure). This view can be useful for exploring objective functions—a visual way to help avoid local minim a.

Select parameters to plot in the drop- down lists, and if you have more than one objective you can select from the Objective drop-down list.

Use the right-click context menu to togg le constraint display, contour labels, Þll contours, and colorbar, and control other options such as number and placing of contour levels.

Pareto Front Graphs

The Pareto Fron t Graphs (click) are for multiobjective optimization where there is more th an one solution at each run. The Pareto Front graphs show the avail able solutions for the selected r un with the current selection highlighted i n red. Click in the tables or graphs to select solutions. The selected solution is displayed in all other graphs (objective and constraint).

Before you ru n an NBI optimization you can specify how many solutions you want the opti mization to Þnd, using the Set Up and Run Optimization toolbar button to acc ess the Optimization Parameters dialog box.

You can use the Pareto Front graphs, shown in the preceding Þgure, in combination with the table views (Solution Slice and Pareto Slice) and the other plots in the graphs (Objective Slice and Constraint Slice graphs) to help you select best solutions for each run. You can collect these solutions together in the “Selected Solution Slice” on page 6-76.

Constraint Slic e Graphs

The Constraint Slice graphs (click) show the constraint functions at the selected operating point with the solution value in orange. Click inside the tables to select solutions to display. Y ellow areas on the graphs show a region outside a constraint tolerance, as shown in the following Þgure.

This example shows the constraint EXTEMP ” 1290° C.

The constraint graphs (the blue lines) show how the Left Value of each output of a constraint (in this case, the EXTEMP model) depends on the free variables in the optimization (in this case S, EXH and INT). The Left Value is compared with a plot of the Right Value output (in this case, 1290° C) on the same axes.

The red horizontal line denotes the Right Value (i.e., the upper bound on EXTEMP which in this case is 1290° C). Because this value is an upper bound, the yellow region above the red line shows where the constraint is infeasible. Yellow is shown above the Right Value plus the tolerance — on many graphs the distance is too small to see between the red line and the tolerance line where the yellow begin s. By default, this tolerance is taken from the optimization constraint toler ance. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance .

The vertical orange lines show the optimal values of the free variables; the intersection of these with the blue lines is marked with a blue triangle on the

Constraint1 axis—this intersection is the Left Value (1290° C) at the optimal settings. These are the Left and Right values in the Constraint Summary table for Constraint1. See “Constraint Summary Table” on page 6-83.

Note Use the right-click context menu to alter graph size.

If a constraint is violated at the solution value, the Y axis is highlighted in yellow, as shown in Constraint 2 in the following example. If constraint values are greater than the tolerance, the row is highlighted in yellow. By default, this tolerance is taken from the optimization constraint tolerance. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance.

See also “Range Constraint Output” on page 6-84 for an explanation of range constraint graphs, and “Constraint Graphs” on page 6-118 for specific sum optimization features, such as a table gradient constraints.

Constraint Summary Table

The Constraint Summary Table (click) view displays the constraint values for the selected solution in the table. This view can be useful to see at a glance if a solution met all the constraints. If there are many constraints it can be time-consuming to use the constraint graphs for verification. If you are using equality constraints or tight table gradient constraints, the graphs can appear entirely yellow and you can only see whether a feasible solution has been found by looking at the Constraint Summary Table, shown in the following figure.

Constraint values greater than the tolerance appear in bold, and the row is highlighted in yellow. By default, this tolerance is taken from the optimization constraint tolerance. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance . These results should be checked as they may show the optimization failed to find a solution within the constraint, or they may be within tolerance (very close to zero). Constraint values less than zero are within the constraint.

Constraints are evaluated as inequalities, e.g., the first constraint, RESIDFRACatMBT, as shown in the preceding figure, is RESIDFRACatMBT ” 25%. The Left Value shows the left side of the inequality at the optimal settings of the free variables (in this case, the output of the residual fraction model (RESIDFRACatMBT), which is 21.652). The Right Value shows the right side of the inequality (in this case, the upper bound, 25%). The constraint value is the difference between the Left and Right values, and the distance to the constraint edge.

In this case, the second constraint, MBT_Boundary, is violated, so the row is yellow, and the positive Constraint Value is highlighted in bold.

For additional information on working with constraints, see the following topics:

- “Range Constraint Output” on page 6-84 for an explanation of range constraints in the summary table.
- “Constraint Summary” on page 6-119 for speci Þc sum optimization features, such as table gradient constraint outputs.

Range Constraint Output

The range constraint output is best explained using an example problem.

Control parameters or free variables: S, EXH, INT

Fixed variables: N, L

Objective: Maximize TQ(S, EXH, INT, N, L) at the Þxed values shown in the following table.

Run	N	L
1	3000	0.5
2	4000	0.6

Constraint: Restrict S between an upper and lower bound shown in the following table.

Run	N	L	Min S	Max S
1	3000	0.5	20	30
2	4000	0.6	30	40

When the optimization is run the optimizer returns the following optimal values of S, EXH and INT, as the following table shows.

Run	N	L	Optimal S	Optimal EXH	Optimal INT
1	3000	0.5	21.33	8.593	29.839
2	4000	0.6	30	5	7.767

Range constraints implemen t the following expression:

Lower Bound (LB) ” Expression ” Upper Bound (UB)

In CAGE, this expression is implemented as two upper-bound constraints, namely:

$$\begin{matrix} \text{RangeConLeft}(1) \\ \text{RangeConLeft}(2) \end{matrix} = \begin{matrix} \text{–Expression} \\ \text{Expression} \end{matrix} \begin{matrix} \text{–LB} \\ \text{UB} \end{matrix} = \begin{matrix} \text{RangeConRight}(1) \\ \text{RangeConRight}(2) \end{matrix}$$

A range constraint returns two values at each operating point within a run, as shown in the following expression:

$$\begin{matrix} \text{RangeConOut}(1) \\ \text{RangeConOut}(2) \end{matrix} = \begin{matrix} \text{–Expression} + \text{LB} \\ \text{Expression} - \text{UB} \end{matrix}$$

The two values that the range constraint returns are the distance from the lower bound, RangeConOut(1), and the distance from the upper bound, RangeConOut(2), respectively.

The constraint in the example problem is

LB(N,L) ” S ” UB(N,L)

CAGE implements this constraint as

$$\begin{matrix} -S \\ S \end{matrix} \quad \begin{matrix} -\text{LB}(N,L) \\ \text{UB}(N,L) \end{matrix}$$

and returns the following two values at each operating point within a run to the optimizer (in this point-by-point example there is only one point per run):

$$\begin{matrix} \text{RangeConOut}(1) \\ \text{RangeConOut}(2) \end{matrix} = \begin{matrix} -S + \text{LB}(N,L) \\ S - \text{UB}(N,L) \end{matrix}$$

The Optimization Results pane shows the fixed variable settings, the optimal free variable settings, and the evaluation of objectives and constraints at

the optimal free variable settings. In this example, the output of the range constraint at the optimal free variable settings is shown in the Constraint1 column. For each operating point in a run, two values are returned from the range constraint.

Looking at the first run:

Optimal S value = 21.33°

To calculate the distances returned from the range constraint:

Distance from lower bound: RangeConOut(1) = –21.33°+20° = –1.33°

Distance from upper bound: RangeConOut(2) = 21.33°–30° = –8.67°

These are the values shown in the Constraint1 column. Remember that negative constraint values mean that the constraint is feasible. The same values appear in the Constraint Summary Table for the selected run, in the Constraint Value column, as shown in the following figure.

The Constraint Value gives a measure of the distance to the constraint boundary for each constraint output. If the Left Value > Right Value and greater than the tolerance for any of the constraint outputs, the constraint value is bold and the row is highlighted yellow. By default this tolerance is taken from the optimization constraint tolerance. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance . This means that this constraint distance should be checked to see if the constraint is feasible at that point.

The Objective Graphs show cross-sect ion plots of the objective function against each free variable in the problem. The left plot is a plot of the objective function against S, with EXH and INT at their optimal values, for the second run. The range constraint for the second operating point (30 ” S ” 40) can be seen; within the constraint region is white, and all other regions outside the constraint are yellow.

The constraint graphs for a range constraint shows how the Left Value of each output of a range constraint depends on the free variables in the optimization. The Left Value is compared with a plot of the Right Value output on the same axes. This comparison is illustrated for the example problem at the second run, as shown in the top left graph.

Constraint1(1) is the first Left Value of the range constraint, RangeConLeft(1), for the first run in the example problem. The top-left graph shows a blue line, which is a plot of RangeConLeft(1) against S (the constrained variable) with all other free variables set to their optimal values. The red horizontal line denotes the Right Value (RangeConRight(1), i.e., the upper bound on S) which in this case is -20°. Because this value is an upper bound, the yellow region above the red line shows where the table gradient constraint is infeasible. The vertical orange line shows the optimal value of S; the intersection of this line with the blue line is marked with a blue triangle on the Constraint1(1) axis—the triangle marks the Left Value (-21.3°) at the optimal settings. These are the Left and Right values in the Constraint Summary table for Constraint1(1).

Constraint1(2) is the second Left Value of the range constraint, RangeConLeft(2), for the first run in the example problem. The bottom left graph shows a blue line plot of RangeConLeft(2) against S with all other free variables set to their optimal values. The horizontal red line denotes the Right Value (RangeConRight(2)) which in this case is 30°. Because this value is an upper bound, the yellow region above the red line denotes where the table gradient constraint is infeasible. The vertical orange line shows the optimal value of S; the intersection of this with the blue line is marked with a blue triangle on the Constraint1(2) axis—the triangle marks the Left Value (21.3°) at the optimal settings. These are the Left and Right values in the Constraint Summary table for Constraint1(2).

In this example, the range constraint does not depend on EXH or INT, so the constraint graphs against these variables are blank.

Using Optimization Output

In this section...

“Create Sum Optimization from Point- by-Point Optimization Output” on page 6-89

“Exporting to a Data Set” on page 6-90

“Filling Tables from Optimi zation Results” on page 6-92

“Custom Fill Function Structure” on page 6-95

Create Sum Optimization from Point-by-Point Optimization Output

Many users employ a point-by-point optimization work ßow to Þnd good initial values for a sum optimization. To make this work ßow easier and faster, you can use a utility to create a sum optimization from your point-by-point optimization output.

From your point-by-point opti mization output node, select Solution > Create Sum Optimization .

CAGE creates a new optimization (called Sum_myOptimizationName). The optimization has these characteristics:

- The objective matches your original optimization but converted to a sum objective.
- The new optimization has identical constraints to your original optimization. Edit or add to these as usual if desired.
- Your original Þxed and free variables are converted to a sum optimization (a single run with multiple values).
  - The new optimization uses only acce pted solutions from your original optimization output (all runs with a selected Accept check box).
  - Therefore, the number of accepted solutions you had in the original optimization determines the number of values within the sum optimization run.

- The free variable initial values and fixed variable values are populated from your point-by-point optimal results (accepted solutions only).
- The fixed variables have a Weights column with every value set to 1.

Exporting to a Data Set

You can export the optimization output results to new data sets or existing data sets.

Note In a single objective optimization there is only one solution for each operating point, so this is exported. Use the Accept check boxes to choose a subset of results for export. See “Using Acceptable Solutions” on page 6-63.

Multiobjective optimizations produce more than one solution per point, so you must first select your preferred solutions before you can export to a data set. See “Selected Solution Slice” on page 6-76.

To export to a data set:

1 Select Solution > Export to Data Set or use the toolbar button. The Export to Data Set dialog box appears.

2 If exporting to a New data set (the default), you can edit the name in the edit box.

3 If you want to overwrite or add to an existing data set:

a Select the option button Modify existing .

b Select the desired data set in the list.

c Choose from Action list:

- Append adds the data to the chosen data set
- Overwrite replaces all data in the data set with the new data

4 By default, the check box Use acceptable solutions only is selected. Optimization results with selected Accept check boxes will be exported.

Clear the Use acceptable solutions only check box if you want to export all the optimization results. See “Using Acceptable Solutions” on page 6-63.

5 Click OK and the data is exported to the data set.

Export Rules

All fixed and free variables are exported where possible.

No models are exported to the data set. If you want to evaluate a model at the variable values, add the model to the data set in the Data Sets view.

When appending, the rules are the same as when merging data sets:

- Columns of inputs are appended to columns with names that match in the data set you are appending to.
- Outputs (models) and any other co lumns without matching names are not appended.
- The values for any unmatched columns in the data set are set to the set point if possible, or zero otherwise.

Filling Tables from Optimization Results

There are two methods for filling tables with optimization results.

- “Table Filling From Optimization Results Wizard” on page 6-92
- “Filling Tables Via Data Sets” on page 6-94

Table Filling From Optimization Results Wizard

In the Optimization output view, you can use the Table Filling wizard as follows.

1 At the Optimization_Output node, select Solution > Fill Tables , or click the toolbar button .

The Table Filling wizard appears.

2 Select the tables to Þll, and click the button to add them to the list of tables to be Þlled. Click Next .

Note Fill settings are remembered between optimization runs and saved with the CAGE project.

3 Select a table to be Þlled, then select the corre ct variable or model output from the list of optimization results, and click the button to match them, as shown in this example. You can also double-click in the results column to match to the currently selected ta ble. Repeat for other tables.

In a single objective optimization there is only one solution for each operating point. In a multiobjective optimization there is more than one solution per point, so you must Þrst select the preferred solutions before you can use the Table Filling wizard. To collect your preferred solutions you must use the “Selected Solution Slice” on page 6-76, then you can use this wizard to Þll tables with the selected solutions.

4 Select a Fill Method .

- Extrapolate Fill — Uses the optimization results to ﬁll the whole table by extrapolation.
- Direct Fill — Fills only those table cells whose breakpoints exactly match the optimization points.
- Custom Fill — You can write your own table ﬁlling algorithm and use the ﬁle browser to select it. See “Custom Fill Function Structure” on page 6-95.

5 Use acceptable solutions only — select this check box to use only optimization results marked as ’a cceptable’. See “Using Acceptable Solutions” on page 6-63.

6 Add to extrapolation mask — when this check box is selected, ﬁlled table cells are added to the extrapolation mask.

If you use the wizard to repeatedly ﬁll a table any existing extrapolation mask is added to. As an example, consider ﬁlling multiple zones of a table using results from different optimiz ations. All zones are cumulatively added to the mask. If there is overlap with previous ﬁlls cells are overwritten unless they are locked. Note that locked cells are never altered by table ﬁlling.

7 Click Finish to ﬁll the tables.

You will see a dialog box reporting w hich tables have been successfully ﬁlled. Switch to the Tables view to examine the tables.

Filling Tables Via Data Sets

The other method of ﬁlling tables with optimizati on output uses Data Sets. This can be useful to see the optimization results and the ﬁlled table surface on the same plot. In Data Sets you can al so manually edit the results before ﬁlling, and compare results with external data.

1 From the Optimization_Output optimization output node, click Export to Data Set ( ) in the toolbar (or select Solution > Export to Data Set). The Export to Data Set dialog box appears. See “Exporting to a Data Set” on page 6-90 for instructions.

2 Go to the Data Sets view (click the Data Sets button in the Data Objects pane) to see that the table of optimization results is contained in the new data set.

   You can now use this data set (or any optimization results) to fill tables, as you can with any data set.

3 Select the data set and click (Fill Table From Data Set) in the toolbar.

4 Clear the check box to Show table history after fill.

5 Choose to fill a table with the desired optimization output by selecting them in the two lists, then click the button Fill Table at the bottom right.

6 Right-click the display and select Surface to see the filled table surface and the optimization output values.

See also

- "Tutorial: Filling Tables from Data" in the Getting Started documentation for more details on using data sets to fill tables.

Custom Fill Function Structure

It can be useful to create your own function to fill tables from the results of an optimization, for example to implement alternative fill methods, smoothing strategies, or to customize output.

The input/output structure of a custom fill function resembles that of the MATLAB interpolation routines INTERP1 and INTERP2. To see the structure of the function it is best to look at an example:

1 Locate and open the file griddataTableFill.m in the mbctraining directory.

2 Type the following at the command line to open the example:

```
edit griddataTableFill
```

   There are instructions for using this example in the optimization tutorial, "Using a Custom Fill Routine to Fill Tables", in the Getting Started

documentation. This function is an example of a function that will Þll 2-D tables from optimization results.

All 2-D custom Þll functions must take the following six inputs, which will be supplied to it by CAGE when the function is called:

Input	Description
col	Column coordinate of op timization results (NF-by-1)
row	Row coordinate of optimization results (NF-by-1)
filldata	Optimized results at (row, col) points (NF-by-1)
colaxis	Column breakpoints of table to be Þlled (1-by-NCOL)
rowaxis	Row breakpoints of table to be Þlled (NROW-by-1)
currtabdata	Existing table values of table to be Þlled (NROW-by-NCOL)

The function must pass three output arguments back to CAGE, to allow CAGE to Þll the table:

Output	Description
ok	Boolean ßag to indicate success of the table Þll (TRUEor FALSE)
tabval	New table values of table to be Þlled (NROW-by-NCOL)
fillmask	Logical matrix to indicate cells to be added to the extrapolation mask as a consequence of the table being Þlled (NROW-by-NCOL)

In the above speci Þcations:

- NF is the number of points from the optimization results that will be used to fill your tables
- NCOL is the number of column breakpoints in the table
- NROW is the number of row breakpoints in the table

Note that your function should handle the cases when the table fill is successful or not. In griddataTableFill, this is handled using the try-catch construct around the call to griddata. If griddata should fail, then the ok flag is set to false and the function returns.

Custom Fill Function for 1–D Tables

You can also write custom fill functions to fill 1–D tables. In this case the input and output specifications are as follows:

Input	Description
row	Row coordinate of optimization results (NF-by-1)
filldata	Optimized results at (row, col) points (NF-by-1)
rowaxis	Row breakpoints of table to be filled (NROW-by-1)
currtabdata	Existing table values of table to be filled (NROW-by-1)

Output	Description
ok	Boolean flag to indicate success of the table fill (TRUE or FALSE)
tabval	New table values of table to be filled (NROW-by-1)
fillmask	Logical matrix to indicate cells to be added to the extrapolation mask as a consequence of the table being filled (NROW-by-1)

Analyzing Point-by-Point Optimization Output

In this section...
“Process for Analyzing Optimi zation Results” on page 6-98
“Detecting Local Optima” on page 6-101
“Investigating Early Terminati on of Optimization” on page 6-105
“Handling Flat Optima” on page 6-110

Process for Analyzing Optimization Results

This topic describes a process for analyz ing the results from single-objective optimizations (e.g., maximizing torque vs. spark, ICP, ECP at an engine operating point, using foptcon , ga and patternsearch algorithms).

For each run of an optimization, the aim is to Þnd the optimal solution. The Optimization Output View provides gr aphical tools to help you determine whether an optimal solution has been found for a given run. This view provides a table with icons that indicate the status of each optimization run.

Optimization Results Table Icons

Icon	Description
Green square Accept icon	Indicates su ccess (algorithm exit ßag > 0).
Orange trian gle Accept icon	Indicates the optimization terminated early (exit ßag = 0). This situation typically occurs when the optimizer has reached some form of time limit. Examples of this include exceeding a number of iterations or function evaluation limit. In such cases, the optimization was in progress but was forced

Optimization Results Table Icons (Continued)

Icon	Description
	to terminate before the optimal solution had been found.
Red circle Accept icon	Indicates failure (an exit ßag < 0). Typically this occurs due to the problem being over constrained for this run.

The process for analyzing point-by-poi nt optimization results comprises the following tasks:

- “Analyzing Output for All Runs” on page 6-99
- “Adjusting Settings To Improve Results” on page 6-100

Analyzing Output for All Runs

1 Switch to the Optimization Output view for the optimization.

2 Analyze all runs with green square Accept icons ( ). For each run:

   a Inspect the Objective Graphs.

   b Inspect Objective Contour plots for as many pairs of free variables as possible. You can conÞgure the optimization output view to display multiple contour plots simultaneously.

   Has the solution found a local optimu m? Many optimiza tion algorithms are designed to locate local optima (e.g., foptcon in CAGE). Check each successful run to ensure that the optimizer has found the best solution possible. See “Detecting Local Optima” on page 6-101 for more information and examples.

   Does the optimization appear to have terminated early? In some cases an optimization appears to return sub-optimal results even though the optimizer has returned a positive exit ßag. Investigate such cases. See “Investigating Early Terminati on of Optimization” on page 6-105.

3 Repeat steps 2a and 2b to analyze all runs with orange triangle Accept icons ( ) that indicate the optimization terminated early. See “Investigating Early Ter mination of Optimization” on page 6-105 for more information and examples.

4 Repeat steps 2a and 2b to analyze all runs with red circle Accept icons ( ) that indicate failures. These runs have typically failed to meet constraints. Inspect the plots and determine if it is acceptable to relax any of the constraints.

Adjusting Settings To Improve Results

After you investigate your results to id entify problems, use these suggestions to try to improve your optimization results:

1 If you detect local optima, try running the optimization again to locate the best optimum.

- Edit the initial condition manually for this optimization operating point and rerun.
- For point optimizations that use the foptcon algorithm, set the Number of start points to be greater than 1 and rerun. In this case, CAGE performs the optimization more than once for each run. To save time, you might want to only repeat the offending runs in this way.
- Use an alternative algorithm on the runs that have found a local optimum. For example, you could try the ga or patternsearch algorithms in CAGE (if you tried the foptcon algorithm Þrst).

2 If the optimization terminates early:

- In cases where the optimizer ru ns out of iterations/function evaluations/time and t he solution returned is feasible, determine whether the solution is acceptable to you.
  - To accept the solution, select the A ccept check box on the Optimization Results table.
  - If you reject the solution, rerun the optimization with modi Þed parameter settings. In this case, if foptcon or patternsearch is being used, it is advisable to start the optimizer from the solution that has just been found.

- In cases where the optimizer ru ns out of iterations/function evaluations/time and the solution returned is infeasible, you can try rerunning the optimization from di fferent initial conditions (for foptcon or patternsearch) or different parameter settings (all algorithms). If this approach does not resolve the problem, determine if any constraint has been violated. Investigate violated constraints, to determine whether they can be relaxed. If they can, rerun the optimization with the relaxed constraints; if not, leave the check box unselected to indicate the solution is unacceptable.

3 See also “Handling Flat Optima” on page 6-110.

Detecting Local Optima

The following Þgure shows views for an optimization which has found the optimal solution. The objective is to maximize Torque (Objective1) against spark angle (S), Exhaust valve closing (ECP) and Intake valve opening (ICP).

This result is taken from the Gasoline case study (see “Gasoline Engine Calibration Case Study” in the Model-Based Calibration Toolbox Getting Started Guide).

When you analyze the optimization results, look for results that have located the best optimum against the free variables.

In this case, an individual plot can on ly show it is highly likely rather than deÞnitely the optimal value because there are more than two free variables. For problems with more than two free variables, the Objective Graphs and Contours views cannot guarantee that an optimal solution has been found because they provide projections of the model.

For further con Þrmation, you should inspect the Objective Contour view for as many pairs of free variables as you have time to analyze.

The following example shows the algorithm has found a local maximum (marked by the orange cross). You can see the global maximum for this optimization in the lower-rig ht corner of the contour plot.

A constraint, such as a table gradient co nstraint, could cause a local maximum result. This result could be desirable, however, because it may be preferable for table smoothness to Þnd a local maximum with a slight loss of torque compared to the global maximum (in this case, about 1.3 NM of torque (1%) which is within model accuracy).

To inspect contour plots for many pairs of free variables, you can con Þgure the optimization output view to display mul tiple contour plots simultaneously. Simultaneous display can help locat e those runs that have converged to a local optimum.

The following Þgure simultaneously shows contour plots for all pairs of free variables for the gasoline case study.

Investigating Early Termination of Optimization

Inspect the Objective Graphs and Cont our Views to check for optimizations that have terminated early. Early ter mination typically occurs with runs that have warning orange triangle Acce pt icons, but can also occur when the optimizer has returned a success ful green square Accept icon.

The following Þgure shows an optimization run with a warning orange triangle Accept icon that has been force d to terminate because it exceeded the iterations limit.

In this case, the optimizer has almost found the optimal solution for this run. If this optimizer has taken a long time to run, then as this solution is almost optimal it is probably worth ma rking as acceptable (select the Accept box in the Optimization Results table for this run).

The following Þgure shows another example where an optimization terminated early because it ex ceeded the iterations limit.

In this case , the problem appears to be over constrained because the plots are entirely sh aded yellow. You can check the constraint summary table or the output tab le to identify if constraints are met. Also inspect the constraint summary an d constraint graphs.

Note Solut ions on the constraint boundary a nd table gradient constraints often cause all objective and contour plots to be yellow (see “Table Gradient Constrai nt Output” on page 6-120).

The constraint graphs for this case are shown in the following Þgure.

These constraint views con Þrm that Constraint2 is violated for this run. Therefore, this solution is probably best left as unacceptable. In cases like this, if it is not already marked as unacceptable, clear the Accept box in the Optimization Results table for this run.

The following Þgure shows an optimization that appears to have terminated early despite return ing a positive exit ßag. You can see that the optimizer has not located the maximum. You should investigate cases like this.

There are many reasons why an optimization appears to terminate early. Two common causes and possible resolutions are discussed in this section.

Poor algorithm parameter settings

Foptcon may not return a local optimum if the following parameter values are too high:

- Variable tolerance
- Function tolerance
- Constraint tolerance

In this case try reducing the values of these parameters to improve performance. However, do not reduce these parameter values too low

(less than ~10^{-10}) to avoid internal issues with foptcon. Models that have nonphysical nonlinearity can also cause failure.

Some nongradient-based algorithms may not return an optimum solution. An example of this is the genetic algorithm (ga) optimization in CAGE. A poor choice of parameters for such algorithms can lead to early termination of the optimization. For example, setting the Crossover Fraction parameter of the ga algorithm to 1 can lead to a situation where the algorithm prematurely converges. In this case, try rerunning the optimization at alternative parameter settings. For best results, rerun the algorithm with a Crossover Fraction lower than 1 (the default is 0.8).

Using foptcon with noisy models

Optimizations can terminate early because the models are noisy and you used a gradient based algorithm (foptcon) to solve the optimization problem.

If the contour plots or any results are suspicious you should always investigate model trends to check if they are sensible and not over fitting. Examine models in the CAGE Surface Viewer or the Model Browser response surface view. You may need to remodel.

To check whether your model is noisy, zoom in on a line plot of the model in the CAGE Surface viewer. Following is a plot of Objective1 against x around the value of x returned by the optimizer.

You can see that the model is noisy and the optimizer has (correctly) returned a local maximum of the model. However, this result is a maximum of the noise component in the model and not the physical component. If the noise is not behavior of the physical system, then you should remodel the noisy models in the Model Browser. The CAGE Import tool can be used to replace the noisy models with the results of the remodeling and the optimization can be rerun.

Handling Flat Optima

Functions that are flat in the vicinity of their optima can be difficult to optimize. The following figure shows an example of such a function, $g(x, y) = (x^2 + y^2 + xy)^4$, and its surface plot.

This function has a global minimum at (0, 0) and is very ßat in the vicinity of the optimal solution.

Using the foptcon algorithm in CAGE to Þnd the minimum of this function (from initial conditions of $(x, y) = [0.5, 0.5]$) produces the result shown in the following Þgure. The optimizer Þnds a solution at $(x, y) = [-0.113, -0.113]$, which is not optimal. In the following plots, you can clearly see that the optimizer has not located the minimum at (0, 0).

To adjust the optimizer to find the minimum, you can take one of several approaches:

- Change the initial conditions.

  For a gradient-based algorithm (foptcon in CAGE), changing the initial conditions can help the optimizer locate a minimum where the objective function is flat in the vicinity of the minimum. In the example shown in the previous figure, changing the initial conditions to (x,y) = (1,1) leads to foptcon finding the minimum at (0, 0).

- Rescale the objective function.

  Rescale the objective function with an operation that does not change the location of any optimal solutions, e.g. , try taking a square root, fourth root or log, or multiplying by a positive scalar. Check that the position of the optimum is not changed. When an objective function is flat in the vicinity of an optimum, rescaling the objectiv e function can help gradient-based optimization algorithms such as foptcon in CAGE. In the example shown

in the previous figure, when foptcon in CAGE is used to minimize $10^{12} g(x, y)$, the minimum at (0, 0) is located.

- Use a non-gradient based algorithm.

  Try either the pattern search or genetic algorithm options. As these algorithms do not use gradient information, they can perform better when used on optimization problems with flat minima. In the example shown in the previous figure, the pattern search algorithm in CAGE located the minimum using the default settings.

- Run the optimization from several initial condition values.

  If you are using foptcon then another possible workaround is to set the Number of Start Points parameter to be greater than 1. This setting runs foptcon the specified number of times from different start conditions. Use this option only for the affected runs as it can be time consuming.

- Change tolerances.

  For a gradient-based algorithm (foptcon in CAGE), changing the variable or function tolerances can help the optimizer locate a minimum where the objective function is flat in the vicinity of the minimum. Reducing the variable and function tolerances may improve the convergence to the optimum value in this case.

Interpreting Sum Optimization Output

Some features of the output node are speci Þc to sum optimizations. Using the Example Problem (see “Example Sum Optimization” on page 6-17) for reference these features are described in the following sections:

In this section...
“Operating Point Ind ices” on page 6-114
“Optimization Results Table” on page 6-115
“Objective Graphs” on page 6-116
“Objective Contour Plot ” on page 6-117
“Constraint Graphs ” on page 6-118
“Constraint Summary” on page 6-119
“Table Gradient Constraint Output” on page 6-120

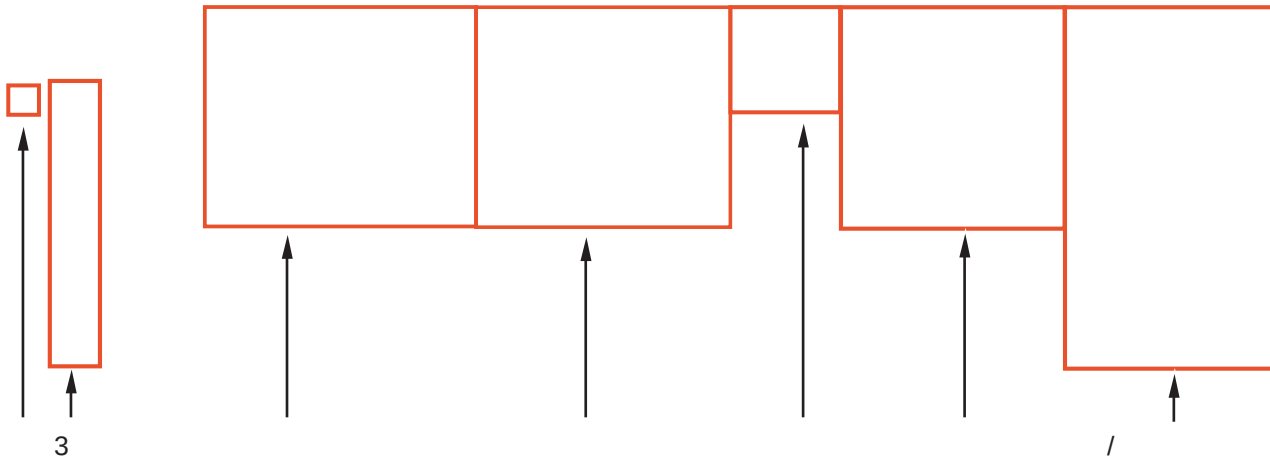
Operating Point Indices

As in the Input Variable Values pane in the Optimization view, in the output view, the index of the operating point within a run is denoted by the number in brackets. The following Þgures provide examples.

In the Optimization Results table, the i ndex of the operating point within the run is shown in brackets. In the Free Variable Values table and graphical displays, the input variable at the i-th operating point within a run is denoted by InputVariableName (i) , for example, S(4) is the spark value at the 4th operating point, EXH(1) is the value of exhaust cam phasing at the Þrst operating point.

Optimization Results Table

Features of the Optimization Results table are labeled in the following Þgure.



3 /

Key to Optimization Results Table

- A: The run index — Index into the set of operating points that is being displayed.
- B: The quantity index.
  - For Þxed and free variables this index corresponds to the index of the operating point within the run.
  - For objectives this corresponds to the index of the output for the speci Þc labeled objective.

- For constraints this corresponds to the index of the output for the specific labeled constraint.

- C: Optimal Free Variable Settings — The optimal settings in this case of S, EXH and INT at each operating point in the run. For example, the optimal settings of S, EXH and INT at the third operating point in this run 1 are S=18.565°, EXH= 24.801°, INT = 41.214°
- D: Fixed Variable Settings — These settings define the operating points for the run and other fixed variables (such as weights) required for objectives and constraints. These values were set up before the optimization was run. For information on the set up of these values, see “Using Variable Values Length Controls” on page 6-19.
- E: Optimal objective outputs — The optimal values of any objective outputs are displayed here, e.g., the optimized value of the weighted sum of TQ (115.002 Nm) over the 5 operating points shown in this case.
- F,G: Constraint outputs at optimized control parameter settings — The value of constraint outputs are displayed here. For the example problem, the model constraint outputs are displayed in the section labeled F. Note that the number of constraint outputs matches the number of operating points. The table gradient constraint outputs are displayed in the section labeled G. The number of values returned by the table gradient constraint is dependent on the internal settings of that constraint (see information see “Table Gradient Constraint Output” on page 6-120). For more information on the number of values returned by objectives and constraints, see “Algorithm Restrictions” on page 6-22.

Objective Graphs

The objective graphs for sum objective problems show the objective cross section plots as in the point-by-point case. However, plots are now displayed against each control parameter at each point in the set of operating points within each run. In the following figure, the weighted sum of TQ is plotted against the spark values at the first four operating points in run 1.

Objective Contour Plot

The objective contour plot for sum objective problems shows the contours of the objective as in the point-by-point case. However, plots can now be displayed against any pair of control pa rameters chosen from all the control parameters at each point in the set of ope rating points within each run. In the following Þgure, a contour plot of the weighted sum of TQ is plotted against the value of exhaust valve timing for the third operating point, EXH(3) and the value of intake valve timing for the third operating point, INT(3) .

Constraint Graphs

The constraint graphs for sum objective problems show the cross section plots of the left side of the constraints as in the point-by-point case. However, in the sum case there are several more inputs and outputs that can be plotted. Specifically, each constraint can return several outputs (see “Algorithm Restrictions” on page 6-22 for more detail) and these can be displayed against each control parameter at each point in the set of operating points within each run.

In the example problem, the exhaust temperature and residual fraction constraints have 5 outputs, one for each operating point. In the graphs shown, one output of the exhaust temperature and residual fraction constraints is displayed against four free variables. Specifically, the exhaust temperature model evaluated at the fifth operating point in run 1 (Constraint1(5)) and the residual fraction model evaluated at the first operating point in run 1 (Constraint2(1)) is plotted against the values of exhaust valve timing at operating points 4 and 5 (EXH(4) and EXH(5)) plus the values of intake valve timing at operating points 1 and 2 (INT(1) and INT(2)).

See also “Table Gradient Constraint Output” on page 6-120.

Constraint Summary

The constraint summary for sum optimizations shows a summary of all the constraint outputs for each constraint at the optimized control parameter settings for the selected run. The constraint summary table for the Example problem is shown in the following figure.

A summary of the first constraint, EXTEMP <= 1290°C at each operating point (Constraint1), is shown in the first five rows of the table. In this case, each of the rows corresponds to an evaluation of the constraint at each operating point within the run. For example, the second row of Constraint1 details an evaluation of EXTEMP <= 1290°C at the second operating point in the set of operating points in the run, as indicated in the Description: EXTEMP(S(2), N(2), EXH(2), INT(2))<= 1290.

The summary for the table gradients (Constraint3 and Constraint4) is shown. For a detailed explanation of table gradient outputs, see the next section, “Table Gradient Constraint Output” on page 6-120.

Table Gradient Constraint Output

The table gradient constraint output is best explained using an example problem.

Control parameters/free variables: SPK, EXH, INT

Fixed variables: N, L

Objective: Maximize Weighted sum of TQ(SPK, EXH, INT, N, L) over the points shown in the following table (with unit weights at each point):

N	L
3000	0.5
3000	0.6

N	L
4000	0.5
4000	0.6

Table Gradient Constraint: Maxi mum change in EXH is bounded by the following speci Þcations:

- No more than 5° per 1000rpm change in N
- No more than 4° per 0.1 change in L
- Over the following 2-by-2 table: N breakpoints = [3000 4000]; L breakpoints =[0.5 0.6]

  In this case, the optimization operating points are the same as the selected table breakpoints for the table gradient constraint, but these are not necessarily always the same.

When the optimization has run, the following optimal values of EXH are returned from the optimizer, as shown in the following tables.

N/L	L(1)	L(2)
N(1)	EXH(1)	EXH(2)
N(2)	EXH(3)	EXH(4)

The values for all these items are shown in the following table.

N/L	0.5	0.6
3000	2.225	0
4000	-2.775	–5

Table gradient constraints calculate the gradient between the values of speciÞed free variable at the speci Þed table points speci Þed by the constraint. In the example problem, the table gr adient constraint returns a set of constraint values as follows.

The table gradient constraint takes the values of EXH from the optimizer, and then determines the value of EXH at the grid points de Þned in the table gradient constraint. In this case, those grid points are the same, so this

is identical to the preceding table. In cases where the grid points in the optimization do not match those in the table gradient constraint, a radial basis function interpolant is used to estimate the constrained variable on the table gradient grid points.

The table gradient constraint takes the grid of EXH values and calculates row and column gradients. Row gradients in the direction of increasing N, rg^{inc}, are calculated on the grid as follows:

rg_1^{inc} = (EXH(3)–EXH(1))/(N(2)–N(1))

= (–2.775–2.225)/1000

= –0.005

rg_2^{inc} = (EXH(4)–EXH(2))/(N(2)–N(1))

= (–5–0)/1000

= –0.005

The table gradient constraint restricts the row and column gradients in each direction. Row gradients in the direction of decreasing N, rg^{dec}, are calculated on the grid as follows:

$rg_1^{dec} = -rg_1^{inc} = 0.005$

$rg_2^{dec} = -rg_2^{inc} = 0.005$

Column gradients in the direction of increasing L, cg^{inc}, are calculated on the grid as follows:

cg_1^{inc} = (EXH(2)–EXH(1))/(L(2)–L(1))

= (0–2.225)/0.1

= –22.25

cg_2^{inc} = (EXH(4)–EXH(3))/(L(2)–L(1))

= (–5–(–2.775))/0.1

= –22.25

Similarly, column gradients in the direction of decreasing N, rg^{dec}, are calculated on the grid as follows:

$cg_1^{dec} = -cg_1^{inc} = 22.25$

$cg_2^{dec} = -cg_2^{inc} = 22.25$

The table gradient constraint implements the following:

rg_1^{inc}	5 / 1000
rg_2^{inc}	5 / 1000
rg_1^{dec}	5 / 1000
rg_1^{dec}	5 / 1000
cg_1^{inc}	4 / 0.1
cg_2^{inc}	4 / 0.1
cg_1^{dec}	4 / 0.1
cg_1^{dec}	4 / 0.1

This equation can be rewritten as Left Value <= Right Value. In each row the Left Value must be smaller than the R ight Value to meet the constraint.

The Constraint Value numbers returned to the optimizer are calculated as follows: Constraint Value = Left Value – Right Value.

$$\begin{matrix} rg_1^{inc} \\ rg_2^{inc} \\ rg_1^{dec} \\ rg_1^{dec} \\ cg_1^{inc} \\ cg_2^{inc} \\ cg_1^{dec} \\ cg_1^{dec} \end{matrix} \begin{bmatrix} 5/1000 \\ 5/1000 \\ 5/1000 \\ 5/1000 \\ 4/0.1 \\ 4/0.1 \\ 4/0.1 \\ 4/0.1 \end{bmatrix} = \begin{bmatrix} -0.005 \\ -0.005 \\ 0.005 \\ 0.005 \\ -22.25 \\ -22.25 \\ 22.25 \\ 22.25 \end{bmatrix} \begin{bmatrix} 0.005 \\ 0.005 \\ 0.005 \\ 0 \\ 40 \\ 40 \\ 40 \\ 40 \end{bmatrix} = \begin{bmatrix} -0.01 \\ -0.01 \\ 0 \\ 0 \\ -62.25 \\ -62.25 \\ -17.75 \\ -17.75 \end{bmatrix}$$

These constraint values are shown in the Optimization Results table. Negative constraint values mean the constraint is feasible, and infeasible constraints are highlighted yellow. In the following figure, these values appear in the Constraint4 column. The Optimization Results pane also shows the fixed variable settings, the optimal free variable settings, and the evaluation of objectives and constraints at the optimal free settings.

The constraint graphs for a table gradient constraint show how the Left Value of each output of a table gradient constraint depends on the free variables in the optimization. These graphs for the example problem appear in the following figure.

The Left Value is compared with a plot of the Right Value output on the same axes. This comparison is illustrated for the table gradient example problem. Consider the top-left graph in the figure shown. Constraint4(1) is the first Left Value (rg_1^{inc}) of the table gradient constraint in the example problem. Recall that this can be written as

$$rg_1^{inc} = (EXH(3)–EXH(1))/(N(2)–N(1))$$

The top left graph shows a plot of rg_1^{inc} against EXH(1) with all other free variables set to their optimal values, i.e.,

$$rg_1^{inc} = (2.775–EXH(1))/1000$$

which is the blue line shown in the top left graph. The horizontal red line shows the Right Value (i.e., the upper bound on rg_1^{inc}). Because this value is an upper bound on the allowable gradient, the yellow region above the line shows where the table gradient constraint is infeasible. The vertical orange line shows the optimal value of the free variable, EXH(1). The blue marker on the Constraint4(1) axis marks the Left Value (the value of rg_1^{inc}) at the intersection of the optimal EXH(1) value and the blue line.

The graph of Constraint4(1) against EXH(2) shows a flat line. The flat line indicates that there is no dependence of rg_1^{inc} on EXH(2), as it is calculated as (EXH(3)–EXH(1))/(N(2)–N(1)).

The other constraint graphs can be analyzed in a similar way.

Note If you are using table gradient constraints the solution may appear infeasible upon inspection of the objective and constraint graphs (the graphs may appear to be entirely yellow). There are cases when the solution is actually feasible in this case. This appearance of infeasibility often arises in sum problems which have tight table gradient constraints. In such cases, you should check the Solution Information pane and the Constraint Summary Table to check whether a feasible solution has been found.

A summary of the table gradient constraint output is shown in the Constraint Summary table, as shown following.

The maximum gradient in the row and column direction (if it is a 2-D table gradient constraint) is shown in the table. In the example shown, observe the maximum column gradient of EXH Recall previously that the cg (column gradient) values were calculated to be –22.25, –22.25, 22.25 and 22.25. The maximum column gradient is 22.25, shown in the Left Value column in the Constraint Summary table. The bound at the maximum value of the column gradient is 40, shown in the Right Value column in the table. The Constraint Value column shows the value of Left Value minus Right Value, which is –17.75, so the constraint has been met.

The Constraint Value gives a measure of the distance to the constraint boundary for each constraint output. If the Left Value > Right Value and greater than the tolerance for any of the constraint outputs, the constraint

value is bold and the row is highlighted yellow. By default this tolerance is taken from the optimization constraint tolerance. You can control the value used for this highlighting by selecting View > Edit Constraint Tolerance . The highlighting indicates that this constraint distance should be checked to see if the constraint is feasible at that point.

User-Defined Optimization

In this section...

"Introducing User-Defined Optimization" on page 6-128

"Implementing Your Optimization Algorithm in CAGE" on page 6-129

"About the Worked Example Optimization Algorithm" on page 6-131

"Checking User-Defined Optimizations into CAGE" on page 6-133

Introducing User-Defined Optimization

User-defined optimizations are described in the following sections:

- "Implementing Your Optimization Algorithm in CAGE" on page 6-129 describes how to customize the optimization template to use your optimization routines in CAGE.
- There is a step-by-step guide to using the example provided to help you understand how to modify the template file to use your own optimization functions. See the optimization tutorial section "Worked Example Optimization" in the Getting Started documentation.

In many cases the standard routines supplied for constrained single objective (`foptcon`, `ga`, and `patternsearch`) and multiobjective optimization (`NBI`) are sufficient to allow you to solve your optimization problem. Sometimes, however, you need to write a customized optimization algorithm. This can be useful in many situations, for example,

- For an expert to capture an optimization process to solve a particular problem, for example, determination of optimal spark angle and exhaust gas recirculation rate on a port-fuel injection engine
- To implement an alternative optimization algorithm to those supplied
- To implement a complex constraint or objective that is only possible through writing M-code
- To produce custom output graphics

User-defined optimization functions in CAGE allow advanced users to write their own optimization routines that can access current CAGE data. In order to access the user function from CAGE, you must register the M-file with CAGE and place it on the MATLAB path. It is crucial that this function conforms to the template specified. The following sections describe this process.

Implementing Your Optimization Algorithm in CAGE

At some point a CAGE optimization function calls on an algorithm to optimize the objective functions over the free variables. You can implement the algorithm in the CAGE optimization function as an external M-file. Use the template file as a basis for your optimization function. The best way to understand how to alter the template file to implement your own optimization algorithms is to compare it with the worked example, as described in the optimization tutorial.

- See the following optimization tutorial sections in the Getting Started documentation:
  - "Worked Example Optimization" describes the process of using the worked example
  - "Creating an Optimization from Your Own Algorithm" describes in detail the steps necessary to use an example optimization algorithm in CAGE
- "About the Worked Example Optimization Algorithm" on page 6-131

  examines the coding involved in implementing an external optimizer in a CAGE optimization M-file
- "Checking User-Defined Optimizations into CAGE" on page 6-133 explains how to check in your optimization function so you can use it in CAGE

Optimization Function Structure

The optimization function M-files have two sections. To compare these sections in the worked example with the template file on which it is based:

1. Locate and open the file mbcOStemplate in the mbctraining directory
2. Type the following at the command line to open the example:

```
edit mbcOSworkedexample
```

The two sections are the Options section and Evaluate section.

1 The Options function section contains the settings that define your optimization. Here you can set up these attributes:

- Name
- Description
- Free variables
- Objective functions
- Constraints
- Helper data sets
- Optimization parameters

CAGE interacts with the cgoptimoptions object, where all these settings are stored.

See “Methods of cgoptimoptions” on page 6-136 for information about setting up the options section.

If you leave the cgoptimoptions function unchanged, your optimization function must be able to support the default options. That is, your optimization will have:

- One objective
- Any number of constraints (selected by the user in CAGE)

2 The Evaluate function section contains your optimization routine. CAGE calls this section when the Run button is clicked.

Place your optimization routine under this section, interacting with CAGE (obtaining inputs and sending outputs) via the cgoptimstore object. Your optimization must conform to the following syntax:

```
optimstore = <Your_Optimization> (optimstore)
```

where <Your_Optimization> is the name of your optimization function.

Any subfunctions called by your optimization routine should also be placed at the bottom of this section.

See “Methods of cgoptimstore” on page 6-138.

Note Be careful not to overwrite the worked example and template Þles when you are trying them out — save them under a new name when you make changes.

There is a step-by-step guide describing how to modify the template using the worked example optimi zation function in the optimization tutorial. See “Worked Example Optimization” in t he Getting Started documentation.

About the Worked Example Optimization Algorithm

mbcweoptimizer is an example of a user-speci Þed optimization that solves the following problem:

max TQover (AFR, SPK).

- [bestafr, bestspk] = mbcweoptimizer(TQ) Þnds a maximum (bestafr , bestspk) to the function TQ.

  TQ must be a function (or a function handle) where the Þrst two input arguments are AFR and SPK respectively. TQ functions with more parameters can be used. The extra pa rameters to these functions can be speciÞed using anonymous functions. For example if a TQ model has N and L inputs, you can use the following call to mbcweoptimizer :

  ```
  [bestafr, bestspk] = mbcweoptimizer (@(afr, spk) TQ(afr, spk, N, L))
  ```

- [bestafr, bestspk]=mbcweoptimizer (TQ, afrrng, spkrng) Þnds a maximum (bestafr,bestspk) to the function TQ.

  afrrng and spkrng are 1-by-2 row vectors containing search ranges for those variables.

- [bestafr, bestspk]=mbcweoptimizer (TQ, afrrng, spkrng, res) Þnds a maximum (bestafr ,bestspk) to the function TQ.

This optimization is performed over a res -by-res grid of (AFR, SPK) values. If res is not speciÞed, the default grid resolution is 25.

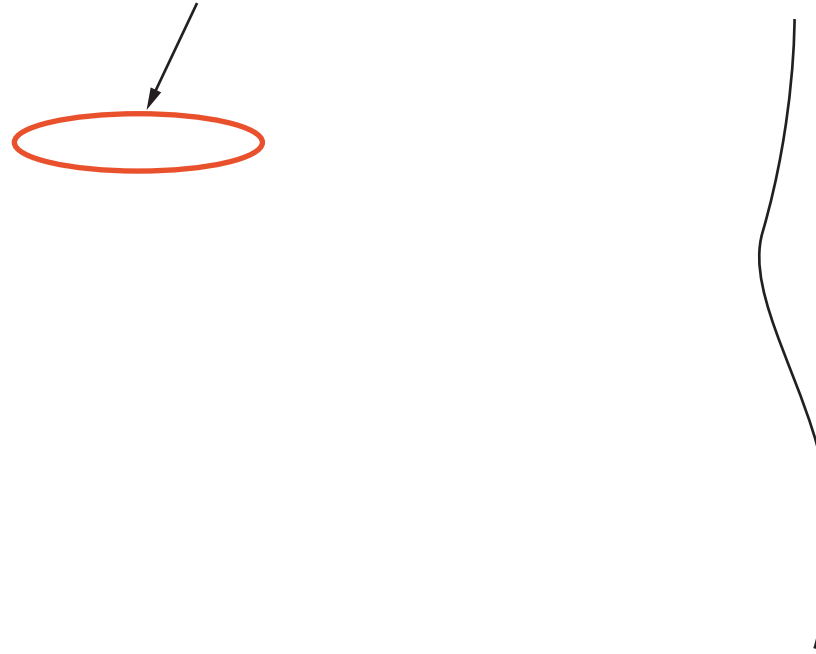
The Structure of the Worked Example

The best way to understand how to impl ement an external optimizer in a CAGE optimization function is to study the details of the example.

- To view the whole worked example M- Þle, at the command line, type

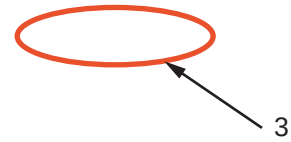
  edit mbcOSworkedexample

The following code section is taken from the Evaluate section of the worked example Þle as an example.



The code fragment above is in the i_Evaluate subfunction. This subfunction is called once for each run of the script. The line of code labeled A above calls the worked example optimization algo rithm external to the optimization

function. As with functions in the Optimization Toolbox product, the first argument to the call to the optimizer is a function handle that evaluates the objectives at a given input point. We recommend you place the function pointed at by the function handle in the optimization file. If you do not place them in the same file you must make sure the evaluate function M-file is on the MATLAB path. As an example, the optimization evaluation function in the worked example optimization is shown in the code fragment following.



The inputs to n_evalTQ are the required inputs for the torque (in this case) model. To evaluate the objective, the evaluate method from the optimstore object is used. In the above example, the line of code referenced by B evaluates the torque model in the worked example at the (afr, spk) input points. The values of (N, L) at the current run are used in the evaluation of the torque model. CAGE retrieves these values from optimstore when the torque model is evaluated.

The two subfunctions presented above are an example of how to implement an external optimizer in a CAGE optimization M-file.

See also the optimization tutorial section “Creating an Optimization from Your Own Algorithm” in the Getting Started documentation, which describes in detail the steps involved in incorporating an example algorithm into a CAGE optimization M-file.

Checking User-Defined Optimizations into CAGE

When you have modified the template to create your own optimization function, you must check it into the Model-Based Calibration Toolbox product in order to use the function in CAGE. Once you have checked in

your optimization function it appears in the Optimization Wizard . See “Optimization Wizard” on page 6-8.

To check a user-deÞned optimization into CAGE,

1 Select File -> Preferences .

2 Click the Optimization tab and click Add... to browse to your M- Þle. Select the Þle and click Open . This registers the optimization function with CAGE. You need to do this when you customize your own optimizations.

The example shows the worked exam ple function, which is already registered with CAGE for use in the optimization tutorial.

3 You can click Test to check that the optimiza tion function is correctly set up. This is a very useful function when you use your own functions; if anything is incorrectly set up the test results tell you where to start correcting your function.

You can see an example of this by saving a copy of the worked example Þle and changing one of the variable names (such as afr) to a number. Try to check this altered function into CAGE and the Test button will return an informative error specifying the line you have altered.

4 Click OK to dismiss the CAGE Preferences dialog box and return to the CAGE browser.

Registered optimizations appear in the Optimization Wizard when you set up a new optimization.

Optimization Function Reference

In this section...
“Methods of cgoptimo ptions” on page 6-136
“Methods of cgoptimstore” on page 6-138

Methods of cgoptimoptions

You use these functions to set up all your optimizati on settings in the Options section of the Þle. You can set up any or all of these seven attributes:

- Name
- Description
- Free variables
- Objective functions
- Constraints
- Helper data sets
- Optimization parameters

The following methods are available:

addFreeVariable	Add free variable to optimization
addLinearConstraint	Add line ar constraint to optimization
addModelConstraint	Add model constraint to optimization
addObjective	Add objec tive to optimization
addOperatingPointSet	Add operating point set to optimization
addParameter	Add parameter to optimization
getConstraints	Return information about all optimization constraints
getConstraintsMode	Return current usage of constraints

getDescription	Get current description for optimization function
getEnabled	Get current enabled status for optimization
getFreeVariables	Return optimization free variable labels
getFreeVariablesMode	Return current usage of free variables
getLinearConstraints	Get linear constraint placeholder information
getModelConstraints	Get mo del constrain t placeholder information
getName	Get current name label for optimization function
getNonlcon	Get nonlinear constraint information
getObjectives	Return information about optimization objectives
getObjectivesMode	Return current usage of objective functions
getOperatingPointSets	Return information about optimization operating point sets
getOperatingPointsMode	Retu rn current usage of operating point sets
getParameters	Return information about optimization parameters
getRunInterfaceVersion	Get preferred interface to provide evaluation function
removeConstraint	Remove constraint from optimization
removeFreeVariable	Remove free variable from optimization
removeObjective	Remove objective from optimization

removeOperatingPointSet	Remove operating point set from optimization
removeParameter	Remove parameter from optimization
setConstraintsMode	Set how op timization constraints are to be used
setDescription	Provide description for optimization function
setEnabled	Set enabled status for optimization function
setFreeVariablesMode	Set how optimization free variables are used
setName	Provide name label for optimization function
setObjectivesMode	Set how optimization objective functions are used
setOperatingPointsMode	Set how optimization operating point sets are used
setRunInterfaceVersion	Get preferred interface to provide evaluation function

Methods of cgoptimstore

The following methods are available:

evaluate	Evaluate optimization objectives and constraints
evaluateConstraint	Evaluate optimization constraints
evaluateNonlcon	Evaluate optimization nonlinear constraints
evaluateObjective	Evaluate optimization objectives
get	Get optimization properties

getA	Get linear inequality constraint matrix.
getB	Get linear inequality constraint target values.
getConstraint	Return constraint labels
getDataset	Retrieve data from data set
getFreeVariables	Get optimal values of free variables
getInitFreeVal	Get initial free values for optimization
getLB	Get free variable lower bounds
getLcon	Return linear constraint labels
getNumConstraint	Return number of constraints per label
getNumConstraintLabels	Retu rn number of constraint labels
getNumLcon	Return number of linear constraints per label
getNumLconLabels	Return number of linear constraint labels
getNumNonlcon	Return number of nonlinear constraints per label
getNumNonlconLabels	Return number of nonlinear constraint labels
getNumObjectiveLabels	Return number of objective labels
getNumObjectives	Return number of objectives per label
getNumRowsInDataset	Get number of rows in optimization data set
getObjectives	Return objective labels for optimization
getObjectiveType	Return objective type
getOptimOptions	Retrieve optimization options object

getOutputInfo	Get output information for optimization
getParam	Get optimization parameter
getStopState	Current stop state for optimization
getUB	Get free variable upper bounds
gridEvaluate	Grid evaluation of optimization objectives and constraints
gridPevEvaluate	Grid evaluation of prediction error variance (PEV)
isScalarFreeVariables	Return whether all free variables are scalars
nEvaluate	Natural evaluation of optimization objectives and constraints
nEvaluateConstraint	Natural evaluation of optimization constraints
nEvaluateNonlcon	Natural evaluation of optimization nonlinear constraints
nEvaluateObjective	Natural evaluation of optimization objectives
optimset	Create/alter optimization OPTIONS structure
pevEvaluate	Evaluate pre diction error variance (PEV)
setExitStatus	Set exit status information for optimization
setFreeVariables	Set optimal values of free variables
setOutput	Set diagnostic information for optimization
setOutputInfo	Set output information for optimization
setStopState	Set current stop state for optimization

Functions — Alphabetical List

addFreeVariable

Purpose

Add free variable to optimization

Syntax

```
options = addfreeVariable (options, label)
```

Description

A method of `cgoptimoptions`. Adds a placeholder for a free variable to the optimization. The string `label` is used to refer to the variable in CAGE.

See Also

`setFreeVariablesMode`, `getFreeVariablesMode`, `getFreeVariables`, `removeFreeVariable`

Purpose

Add linear constraint to optimization

Syntax

```
options = addLinearConstraint(options, label, A, B)
```

Description

A method of cgoptimoptions . Adds a placeholder for a linear constraint to the optimization. The string label is used to refer to the constraint in the CAGE GUI. Linear constraints can be written in the form

```
A(1)X(1) + A(2)X(2) + ... + A(n)X(n) <= b
```

where X(i) is the i th free variable, A is a vector of coefficients, and b is a scalar bound.

Examples

```
% Add SPK and EGR variables to an optimization
opt = addFreeVariable(opt, 'SPK');
opt = addFreeVariable(opt, 'EGR');
% Add a linear constraint such that 3*SPK - 2*EGR <= 30
opt = addLinearConstraint(opt, 'newCon', [3 -2], 30);
```

See Also

getLinearConstraints , addModelConstraint , setConstraintsMode , removeConstraint

addModelConstraint

Purpose

Add model constrai nt to optimization

Syntax

```
options=addModelConstraint(options, label, boundtype, bound)
```

Description

A method of cgoptimoptions . Adds a placeholder for a model constraint to the optimization. The string label is used to refer to the constraint in CAGE.

boundtype can be set either to the string ' greaterthan' or 'lessthan' .

bound must be a scalar real.

If boundtype = 'greaterthan ', the model constraint takes the following form:

CAGE model >= bound

Similarly, if boundtype = 'lessthan ', the model constraint takes the form

CAGE model <= bound

Examples

An optimization requires a c onstraint where a user-de Þned function must be less than 500. The following code line adds a placeholder for this constraint that is labeled ' mycon':

```
opt = addModelConstraint(opt, 'mycon', 'lessthan', 500);
```

See Also

getModelConstraints , addLinearConstraint , setConstraintsMode , removeConstraint

Purpose

Add objective to optimization

Syntax

```
options = addObjective(options, label, typestr)
```

Description

A method of cgoptimoptions . Adds a placeholder for an objective function to the optimization. The string label is used to refer to the constraint in CAGE.

typestr can take one of four values, ' max', 'min ', 'min/max ', or 'helper '.

Examples

```
opt = addObjective(opt, 'newObj', 'max')
```

Adds an objective function labeled newObj to the optimization and indicates that it is to be maximized.

```
opt = addObjective(opt, 'newObj', 'min/max')
```

Adds an objective function labeled newObj to the optimization and indicates that the user should be allowed to choose whether it is minimized or maximized from CAGE.

```
opt = addObjective(opt, 'newObj2', 'helper')
```

Adds an objective function labeled newObj2 to the optimization. The string ' helper ' indicates that the function is used as part of the determination of the cost function but is not directly minimized or maximized.

See Also

getObjectives , setObjectivesMode , getObjectivesMode , removeObjective

addOperatingPointSet

Purpose

Add operating point set to optimization

Syntax

```
options = addOperatingPointSet(options, label, vars)
```

Description

A method of cgoptimoptions . options = addOperatingPointSet(options, label, vars) Adds a placeholder for an additional operating point set to the optimization.

The string label is used to refer to the constraint in CAGE. vars is a (1-by-N) cell array of strings where N >= 1. Each element of vars is a label for a CAGE variable that must appear in the operating point set that the user chooses.

See Also

getOperatingPointSets , setOperatingPointsMode , getOperatingPointsMode , removeOperatingPointSet

Purpose

Add parameter to optimization

Syntax

```
options = addParameter(options, Label, Type, Value)
options = addParameter(options, Label, Type, Value,
    DisplayName)
```

Description

A method of cgoptimoptions .

options = addParameter(options, Label, Type, Value) adds a parameter to the optimization. The string Label is used to refer to the parameter in the Evaluate section of your script. You must specify a default value in Value . The table below lists the parameter types that are supported along with how to specify their Type and Value .

Parameter Type	Type	Value
Real number	'number'	Real scalar
Integer	'integer'	Integer scalar
Enumerated list	{'list', {list items}}	One of {list items}
Boolean	'boolean'	true or false

Note: The {list items} cell array for an enumerated list must be a cell array of strings, one for each list member.

You can restrict a numeric parameter ('number' or 'integer') to a valid range. To do this, specify a cell array for Type from the following:

Range type	Type
Positive	{TYPESTR, 'positive'}
Negative	{TYPESTR, 'negative'}
User deÞned	{TYPESTR, [a b]}

where TYPESTRis either 'number' or 'integer' . Note that the user-deÞned range type strictly includes the limits, whereas the positive

and negative range types exclude zero. Furthermore, the default Value must lie in the specified range.

options = addParameter(options, Label, Type, Value, DisplayName) allows you to add a more descriptive label for the parameter in the CAGE Optimization Parameters GUI. Note that you still must refer to the parameter by label in the Evaluate section of your script.

See Also

getParameters , getParam , removeParameter

Purpose

Evaluate optimization ob jectives and constraints

Syntax

```
Y = evaluate(optimstore, X)
```

Description

A method of cgoptimstore .

Evaluate optimization obje ctives and constraints.

Y = evaluate(optimstore, X) evaluates all of the optimization objectives and constraints at the free variable values X. X is a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization.

Evaluation over data sets is only su pported when the free variables are scalar, that is, you cannot perform evaluation over a data set for "sum" optimizations.

Examples

```
Y = evaluate(optimstore, X, itemnames)
```

evaluates the objectives and constraints speci Þed in the cell array of strings, itemnames , at the free variable values X. The values of the objectives and constraints are returned in Y, which is of size (NPoints-by-NItems) where NIte ms is the number of objectives and constraints listed in itemnames . Note that the evaluation of Y is scaled onto [-1 1].

```
Y = evaluate(optimstore, X, itemnames, datasetname)
```

evaluates the speci Þed objectives and constraints at the operating points in the data set speci Þed by the string datasetname . X must be a (Nrows-by-NfreeVar) matrix, where Nrows is the number of rows in the data set.

```
Y = evaluate(optimstore, X, itemnames, datasetname, rowind)
```

evaluates the speci Þed objectives and constraints at the points of datasetname given by rowind . X must be a (NRows-by-NFreeVar)

matrix where NRows is the length of ROWIND. ROWIND must be a list of integer indices in the range [1 NumRowsInDataset]. Y is a (Nrows-by-NItems) matrix.

See Also

nEvaluate , pevEvaluate

Purpose

Evaluate optimization constraints

Syntax

```
Y = evaluateConstraint(optimstore, X)
```

Description

A method of cgoptimstore.

Y = evaluateConstraint(optimstore, X) evaluates all of the optimization constraints at the free variable values X. X must be a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization. The values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of constraints in the optimization.

If you enable scaling of the optimization items, then the evaluation of Y is approximately scaled onto [-1 1]. See “Scale Optimization” on page 6-60 for more information on scaling.

Negative values of Y imply X is feasible.

Examples

```
Y = evaluateConstraint(optimstore, X, itemnames)
```

evaluates the constraints specified in the cell array of strings, itemnames, at the free variable values X. The values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of objectives listed in itemnames.

```
[Y, YG] = evaluateConstraint(optimstore, X, itemnames)
```

also evaluates the gradient of the specified constraints in YG (if itemnames is not specified, then the gradient of all constraints is returned). YG is of size NFreeVar-by-NItems-by-NPoints, where NFreeVar is the number of free variables in the optimization.

See Also

evaluateObjective, evaluateNonlcon

evaluateNonlcon

Purpose

Evaluate optimization nonlinear constraints

Syntax

```
[varargout] = evaluateNonlcon(optimstore, X, ItemNames)
```

Description

Evaluate optimization nonlinear constraints. A method of cgoptimstore.

`Y = evaluateNonlcon(optimstore, X)` evaluates all of the nonlinear constraints in the optimization at the free variable values `X`. `X` must be a (NPoints-by-NFreeVar) matrix where `NPoints` is the number of points to be evaluated and `NFreeVar` is the number of free variables in the optimization.

If you enable scaling of the optimization items, then the evaluation of `Y` is approximately scaled onto [-1 1]. See “Scale Optimization” on page 6-60 for more information on scaling.

`Y = evaluateNonlcon(optimstore, X, ItemNames)` evaluates the nonlinear constraints specified in the cell array of strings, `ItemNames`, at the free variable values `X`. The values of the nonlinear constraints are returned in `Y`, which is of size (NPoints-by-NItems) where `NItems` is the number of nonlinear constraints listed in `ItemNames`.

`[Y, YG] = evaluateNonlcon(optimstore, X, ItemNames)` also evaluates the gradient of the specified constraints in `YG` (if `ItemNames` is not specified, then the gradient of all constraints is returned). `YG` is of size NFreeVar-by-NItems-by-NPoints, where `NFreeVar` is the number of free variables in the optimization.

See Also

evaluateConstraint, evaluateObjective

Purpose

Evaluate optimization objectives

Syntax

```
varargout = evaluateObjective(optimstore, X, ItemNames)
```

Description

Evaluate optimization objectives. A method of cgoptimstore .

Y = evaluateObjective(optimstore, X) evaluates all of the optimization objectives at the free variable values X. X must be a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization. The values of th e objectives are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of objectives in the optimization.

If you enable scaling of the optimization items, then the evaluation of Y is approximately scaled onto [-1 1]. See “Scale Optimization” on page 6-60 for more information on scaling.

Y = evaluateObjective(optimstore, X, ItemNames) evaluates the objectives specified in the cell array of strings, ItemNames , at the free variable values X. The values of the objectives are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of objectives listed in ItemNames .

[Y, YG] = evaluateObjective(optimstore, X, ItemNames) also evaluates the gradient of the specified objectives in YG (if ItemNames is not specified, then the gradient of all objectives is returned). YG is of size NFreeVar-by-NItems-by-NPoints , where NFreeVar is the number of free variables in the optimization.

See Also

evaluateNonlcon

Purpose

Get optimization properties

Syntax

```
V = get(optimstore, 'PropertyName')
```

Description

Returns the value of the specified property in the optimization. A method of cgoptimstore.

get(optimstore) displays all property names and a description of each property for the OPTIMSTORE object.

S = get(optimstore) returns a structure where each field name is the name of a property of OPTIMSTORE and each field contains the description of that property.

Note This method is obsolete. Use the GETXXX methods instead.

See Also

See also cgoptimstore/GETXXX, for example getA, getB, etc.

Purpose

Get linear inequality constraint matrix.

Syntax

```
A = getA(optimstore)
```

Description

Get the linear inequality constraint matrix. A method of cgoptimstore .

`A = getA(optimstore)` returns the linear inequality constraint matrix used in the optimization. `A` is a `(NLINCON-by-NFreeVar)` matrix where `NFreeVar` is the number of free variables in the optimization and `NLINCON` is the number of linear inequality constraints.

The following code evaluates the li near inequality constraints in the optimization:

```
A = getA(optimstore);
b = getB(optimstore);
out = A*x - b;
```

where x is a column vector containing the current free variable values.

See Also

getB

getB

Purpose

Get linear inequality constraint target values.

Syntax

```
B = getB(optimstore)
```

Description

Get the linear inequality constraint target values. A method of cgoptimstore .

`B = getB(optimstore)` returns the linear inequality constraint target values used in the optimization. `B` is a (NLINCON-by-1) column vector where `NLINCON` is the number of linear inequality constraints.

The following code evaluates the li near inequality constraints in the optimization:

```
A = getA(optimstore);
b = getB(optimstore);
out = A*x - b;
```

where x is a column vector containing the current free variable values.

See Also

getA

Purpose

Return constraint labels

Syntax

```
conLabels = getConstraint(optimstore)
```

Description

Return the constraint labels. A method of cgoptimstore .

conLabels = getConstraint(optimstore) returns the labels for all the constraint functions in optimization. These labels are the those found in the CAGE GUI for the optimization constraints.

See Also

getNonlcon , getLcon

getConstraints

Purpose

Return information about all optimization constraints

Syntax

```
coninfo = getConstraints(obj)
```

Description

Return information about all optimization constraints. A method of cgoptimoptions.

coninfo = getConstraints(options) returns a structure array of information regarding the optimization constraint functions. coninfo(i).label contains the label for the i-th constraint. A string defining the type of the i-th constraint is stored in coninfo(i).typestr. The constraint parameters are stored in coninfo(i).pars.

See Also

addModelConstraint, addLinearConstraint

Purpose

Return current usage of constraints

Syntax

```
mode = getConstraintsMode(options)
```

Description

Returns a string describing how the optimization makes constraints available to the user. mode will be one of 'any' or 'fixed'.

See Also

setConstraintsMode

getDataset

Purpose

Retrieve data from data set

Syntax

```
V = getDataset(optimstore, datasetName, inputNames)
```

Description

Returns required data from a named data set. A method of `cgoptimstore`.

`PTS = getDataset(optimstore, datasetName)` returns all the data from the specified helper data set. If the data set cannot be found, `data` is returned as empty.

`PTS = getDataset(optimstore, datasetName, inputNames)` returns data from the specified helper data set. Data is retrieved for the columns of the data set with names that match those in `inputNames`. If the dataset cannot be found, `data` is returned as empty.

Examples

```
V = getdataset(optimstore, 'myDS', {'speed', 'afr'})
```

returns a `NPTS` by `2` matrix, `V`.

`NPTS` is the number of rows in the operating point set labeled `'myDS'`, `V(:, 1)` is the data for the variable labeled `'speed'`, `V(:, 2)` is the data for the variable labeled `'afr'`.

See Also

`addOperatingPointSet`

Purpose

Get current description f or optimization function

Syntax

```
desc = getDescription(options)
```

Description

A method of cgoptimoptions . Returns the description, desc , of the user-deÞned optimization function.

See Also

setDescription

getEnabled

Purpose

Get current enabled status for optimization

Syntax

```
en=getEnabled(options)
```

Description

A method of cgoptimoptions. Returns whether this user-defined optimization is available to be run. en is set to true or false. When an optimization is disabled, the user can still register it with CAGE but is not allowed to create new optimizations using it.

See Also

setEnabled

Purpose

Get optimal values of free variables

Syntax

```
data = getFreeVariables(obj)
```

Description

A method of cgoptimstore. Get the optimal values of the free variables.

Results = getFreeVariables(obj) returns the matrix of optimal values that has been set for the free variables. Results is a NSOL by NFREEVAR matrix containing many solutions for the optimal values of the free variables. NSOL is the number of solutions and NFREEVAR is the number of free variables.

See Also

setFreeVariables

getFreeVariables

Purpose

Return optimization free variable labels

Syntax

```
labels=getFreeVariables(options)
```

Description

A method of `cgoptimoptions`. Returns the current placeholder labels for the free variables in the optimization. The labels are returned in a (1-by-`NFreeVar`) cell array, `labels`, where `NFreeVar` is the number of free variables that have been added to the optimization.

See Also

`addFreeVariable`, `setFreeVariablesMode`, `getFreeVariablesMode`

Purpose

Return current usage of free variables

Syntax

```
mode= getFreeVariablesMode(options)
```

Description

A method of cgoptimoptions . Returns a string describing how the optimization makes free variables available to the user. mode is set to any or fixed .

See Also

setFreeVariablesMode

getInitFreeVal

Purpose

Get initial free values for optimization

Syntax

```
x0 = getInitFreeVal(cos)
```

Description

Get the initial free values for the optimization. A method of cgoptimstore .

`x0 = getInitFreeVal(optimstore)` returns the initial values of the free variables used in the optimization. `X0` is a (1-by-NFreeVar) matrix where `NFreeVar` is the number of free variables in the optimization.

See Also

setFreeVariablesMode

Purpose

Get free variable lower bounds

Syntax

```
LB = getLB(optimstore)
```

Description

Get the free variable lower bounds. A method of `cgoptimstore`.

`LB = getLB(optimstore)` returns the free variable lower bounds used in the optimization. `LB` is a `(1-by-NFreeVar)` vector where `NFreeVar` is the number of free variables in the optimization.

See Also

`getUB`

getLcon

Purpose

Return linear constraint labels

Syntax

```
conLabels = getLcon(optimstore)
```

Description

Return the linear constraint labels. A method of `cgoptimstore`.

`conLabels = getLcon(optimstore)` returns the labels for the linear constraints in the optimization. These labels are those found in the CAGE GUI for the optimization linear constraints.

See Also

`getObjectives`, `getNumNonlcon`

Purpose

Get linear constraint placeholder information

Syntax

```
out = getLinearConstraints(options)
```

Description

A method of cgoptimoptions. Returns a structure array of information regarding the linear constraints in the optimization. The structure has three fields: label, A, and b. See the help for addLinearConstraint for more information on these fields.

See Also

addLinearConstraint, setConstraintsMode

getModelConstraints

Purpose

Get model constraint placeholder information

Syntax

```
out = getModelConstraints (options)
```

Description

A method of cgoptimoptions. Returns a structure array of information regarding the model constraints in the optimization. The structure has three fields: label, boundtype, and bound. See the help for addModelConstraint for more information on these fields.

See Also

addModelConstraint, setConstraintsMode

Purpose

Get current name label for optimization function

Syntax

```
name=getName(options)
```

Description

A method of `cgoptimoptions`. Returns the current name label, `name`, for the user-defined optimization function.

See Also

`setName`

getNonlcon

Purpose

Get nonlinear constraint information

Syntax

```
out = getNonlcon(obj)
```

Description

Get nonlinear constraint information. A method of cgoptimoptions .

out = getNonlinearConstraints(options) returns a structure array of information regarding the nonlinear constraints in the optimization. The structure has three fields: label , type and pars . The label field contains the label used for the constraint in the CAGE GUI. The typestr field contains constraint type selected by the user. The pars field contains any parameters associated with the constraint.

See Also

getModelConstraints , getLinearConstraints

Purpose

Return number of constraints per label

Syntax

```
ncon = getNumConstraint(optimstore)
ncon = getNumConstraint(optimstore, conLabels)
```

Description

Return the number of constraints per label. A method of cgoptimstore .

ncon = getNumConstraint(optimstore) returns the number of constraints that will be returned from an evaluation of each labeled constraint. For example, consid er an optimization that has a sum constraint over a set of points, S, and a point constraint to be evaluated at each member of S. NCONwill return [1 r] , where r is the number of points in S.

ncon = getNumConstraint(optimstore, conLabels) returns the number of constraints from an evaluation of the de Þned constraints.

See Also

getNumNonlcon

getNumConstraintLabels

Purpose

Return number of constraint labels

Syntax

```
out = getNumConstraintLabels(optimstore)
```

Description

Return the number of constraint labels. A method of `cgoptimstore`.

`out = getNumConstraintLabels(optimstore)` returns the number of constraint labels in the optimization.

See Also

getNumObjectiveLabels

Purpose

Return number of linear constraints per label

Syntax

```
ncon = getNumLcon(optimstore)
ncon = getNumLcon(optimstore, conLabels)
```

Description

Return the number of linear constraints per label. A method of cgoptimstore .

ncon = getNumLcon(optimstore) returns the number of constraints that will be returned from an evaluation of each linear constraint.

ncon = getNumNonlcon(optimstore, conLabels) returns the number of constraints from an evaluation of the de Þned constraints.

See Also

getNumNonlcon , getNumConstraint

getNumLconLabels

Purpose

Return number of linear constraint labels

Syntax

```
numlab = getNumLconLabels(optimstore)
```

Description

Return the number of linear constraint labels. A method of cgoptimstore .

numlab = getNumLconLabels(optimstore) returns the number of linear constraint labels in the optimization.

See Also

getNumConstraintLabels

Purpose

Return number of nonlinear constraints per label

Syntax

```
ncon = getNumNonlcon(optimstore)
ncon = getNumNonlcon(optimstore, conLabels)
```

Description

Return the number of nonlinear constraints per label. A method of cgoptimstore .

ncon = getNumNonlcon(optimstore) returns the number of constraints that will be returned from an evaluation of each labeled constraint. For example, consid er an optimization that has a sum constraint over a set of points, S, and a point constraint to be evaluated at each member of S. NCONwill return [1 r] , where r is the number of points in S.

ncon = getNumNonlcon(optimstore, conLabels) returns the number of constraints type for the de Þned constraints.

See Also

getConstraints , getNumNonlconLabels

getNumNonlconLabels

Purpose Return number of nonlinear constraint labels

Syntax numlab = getNumNonlconLabels(optimstore)

Description Returns the number of nonlinear constraint labels in the optimization.
A method of cgoptimstore .

See Also getNumObjectiveLabels

Purpose

Return number of objective labels

Syntax

```
numlab = getNumObjectiveLabels(optimstore)
```

Description

Returns the number of objective labels in the optimization. A method of cgoptimstore .

See Also

getNumNonlconLabels

getNumObjectives

Purpose

Return number of objectives per label

Syntax

```
nobj = getNumObjectives(optimstore)
nobj = getNumObjectives(optimstore, objlabels)
```

Description

Return the number of objecti ves per label. A method of cgoptimstore .

nobj = getNumObjectives(optimstore) returns the number of objectives that will be returned fr om an evaluation of each objective label. For example, consider an optimization that has a sum objective over a set of points, S, and a point objective to be evaluated at each member of S. nobj will return [1 r], where r is the number of points in S.

nobj = getNumObjectives(optimstore, objlabels) returns the number of objectives that will be returned for the de Þned objective labels.

See Also

getObjectives ; getObjectiveType

Purpose

Get number of rows in optimization data set

Syntax

```
npts = getNumrowsInDataset(optimstore, datasetName)
```

Description

Returns the number of rows in the named data set. A method of cgoptimstore .

getObjectives

Purpose

Return objective labels for optimization

Syntax

```
objLabels = getObjectives(optimstore)
```

Description

A method of cgoptimstore . Returns the labels for the objective functions in optimization. These labels are those found in the CAGE GUI for the optimization objectives.

See Also

getLcon

Purpose

Return information about optimization objectives

Syntax

```
objinfo=getObjectives(options)
```

Description

A method of cgoptimoptions. Returns a structure array of information regarding the optimization objective functions. objinfo(i).label contains the label for the i^{th} objective. A string defining the type of the i^{th} objective (max, min, min/max, or helper) is stored in objinfo(i).type.

See Also

addObjective, setObjectivesMode, getObjectivesMode

getObjectivesMode

Purpose

Return current usage of o bjective functions

Syntax

```
mode = getObjectivesMode(options)
```

Description

A method of cgoptimoptions . Returns a string describing how the optimization makes objecti ves available to the user. mode will be one of 'multiple' , 'any', or 'fixed' .

See Also

setObjectivesMode

Purpose

Return information about opti mization operating point sets

Syntax

```
getOperatingPointSets(options)
```

Description

A method of cgoptimoptions . Returns a structure array of information regarding the optimization operating point sets. The structure has two Þelds, label and vars . See the help for addOperatingPointSet for more information on these Þelds.

See Also

addOperatingPointSet , setOperatingPointsMode , getOperatingPointsMode

getOperatingPointsMode

Purpose

Return current usage of operating point sets

Syntax

```
mode=getOperatingPointsMode(options)
```

Description

A method of cgoptimoptions . Returns a string describing how the optimization makes operating po int sets available to the user. mode will be one of 'default' , 'fixed' , or 'any' .

See Also

setOperatingPointsMode

Purpose

Return objective type

Syntax

```
objType = getObjectiveType(optimstore)
objType = getObjectiveType(optimstore, objLabels)
```

Description

Return the objective type. A method of cgoptimstore .

objType = getObjectiveType(optimstore) returns the objective type of all the objectives in the optimization. A 1-by-NOBJ cell array is returned, each element being 'min', 'max' or 'helper'.

objType = getObjectiveType(optimstore, objLabels) returns the objective type for the defined objectives.

See Also

getObjectives

getOptimOptions

Purpose

Retrieve optimizati on options object

Syntax

```
options = getOptimOptions(optimstore)
```

Description

A method of cgoptimstore . Returns the optimization con Þguration object. Information about the optimization set up can be retrieved from this object.

Purpose

Get output information for optimization

Syntax

```
[exitflag, msg, stats] = getOutputInfo(cos)
```

Description

Get output information for the optimization. A method of cgoptimstore .

[exitflag, termMsg] = getOutputInfo(optimstore) returns diagnostic output information from optimstore . exitflag indicates the success (exitflag > 0) or failure (exitflag <= 0) of the current optimization run. exitflag may also give some indication why the optimization terminated. Any termination message set by the optimization can be retrieved from termMsg .

[exitflag, termMsg, output] = getOutputInfo(optimstore) returns in addition a struc ture of algorithm-speci Þc information in output . For output to be non-empty, the user must create it in their algorithm. See the worked example a nd tutorial for more information on how to create output structures.

getParam

Purpose

Get optimization parameter

Syntax

property_value = getParam(obj, propertyname)

Description

Get optimization parameter. A method of cgoptimstore .

V = getParam(optimstore, 'Parameter_name') returns the value of the specified parameter in the optimization. These optimization parameters must be set up in the Options section of the user-defined script.

See Also

addParameter

See the example file mbcOSworkedexample , used in the optimization tutorial “Worked Example Optimization”.

Purpose

Return information about optimization parameters

Syntax

```
getParameters(options)
```

Description

A method of cgoptimoptions. Returns a structure array containing information about the parameters that are defined for the optimization. Parameter information is returned in a structure with fields label, typestr, value, and displayname. See the help for addParameter for more information on these fields.

See Also

addParameter, getParam

getRunInterfaceVersion

Purpose

Get preferred interface to pr ovide evaluation function

Syntax

```
ver = getRunInterfaceVersion(obj)
```

Description

Get the preferred interface to provide the evaluation function. A method of cgoptimoptions .

ver = getRunInterfaceVersion(options) returns the Model-Based Calibration Toolbox product Ve rsion that is emulated when the optimization func tion's evaluate op tion is called. If ver is set to 2, the interface provided by Model-Based Calibration Toolbox Version 2 software is activated. If ver is set to 3, the new interface, which Model-Based Calibration Toolbox Version 3 software de Þnes, is used.

See Also

setRunInterfaceVersion

Purpose

Current stop state for optimization

Syntax

```
stop= getStopState(opt)
```

Description

A method of cgoptimstore . stop= getStopState(optimstore) returns the current stop state for the o ptimization. The stop state could be set by the Stop button on the Running Optimization progress bar or via a call to setStopState within a script.

See Also

setStopState

getUB

Purpose

Get free variable upper bounds

Syntax

```
UB = getUB(optimstore)
```

Description

A method of cgoptimstore. Returns the free variable upper bounds used in the optimization. UB is a (1-by-NFreeVar) vector where NFreeVar is the number of free variables in the optimization.

See Also

getLB

Purpose

Grid evaluation of optimizati on objectives and constraints

Syntax

```
Y = gridEvaluate(optimstore, X)
Y = gridEvaluate(optimstore, X, objconname)
Y = gridEvaluate(optimstore, X, objconname, datasetname)
Y = gridEvaluate(optimstore, X, objconname, datasetname, rowind)
```

Description

A method of cgoptimstore .

Y = gridEvaluate(optimstore, X) evaluates all the objectives and constraints at the points X for the current run. This call produces identical results to the equivalent call to cgoptimstore/evaluate .

Y = gridEvaluate(optimstore, X, objconname) evaluates the objectives/constraints speci Þed in the cell array objconname as described above.

Y = gridEvaluate(optimstore, X, objconname, datasetname) evaluates all the objectives and co nstraints at all combinations of the points in datasetname with X. The return matrix, Y, is of size SIZE(X,1)-by-(NOBJ+NCON)-by-NPTS , where NOBJ is the number of objectives, NCON is the number of constraints and NPTS is the number of rows in P. Further, Y(I, J, K) is the value of the J-th objective/constraint at X(I, :) and P(K, :). Y is scaled on [-1 1].

Examples

Objectives : O1, O2

Constraints : C1, C2

Primary data set:

A	B
4	5
1	3

Free variables:

X1	X2	X3
2	4	8
1	9	3
6	2	7

X

In this case, the following command

```
Y = gridEvaluate(optimstore, X)
```

evaluates objectives and constr aints at the following points:

A	B	X1	X2	X3
4	5	2	4	8
4	5	1	9	3
4	5	6	2	7
1	3	2	4	8
1	3	1	9	3
1	3	6	2	7

Y is a 3-by-4-by-2 matrix where

Y(:, 1, 1) = Values of 01 at A = 4, B = 5

Y(:, 2, 1) = Values of 02 at A = 4, B = 5

Y(:, 3, 1) = Values of C1 at A = 4, B = 5

Y(:, 4, 1) = Values of C2 at A = 4, B = 5

Y(:, 1, 2) = Values of 01 at A = 1, B = 3

Y(:, 2, 2) = Values of 02 at A = 1, B = 3

Y(:, 3, 2) = Values of C1 at A = 1, B = 3

Y(:, 4, 2) = Values of C2 at A = 1, B = 3

```
Y = gridEvaluate(optimstore, X, objconname, datasetname, rowind)
```

evaluates the specified objectives/constraints at the points of datasetname given by rowind as described above. Y is a length (rowind) by length (objconname) by npts matrix.

See Also

evaluate

gridPevEvaluate

Purpose

Grid evaluation of prediction error variance (PEV)

Syntax

```
[y, ysums] = gridpevevaluate(optimstore, X)
Y = gridpevevaluate(optimstore, X, objconname)
Y = gridpevevaluate(optimstore, X, objconname, datasetname)
Y = gridpevevaluate(optimstore, X, objconname, datasetname, rowind)
```

Description

Warning

The evaluation of PEV is no longer supported in cgoptimstore and this method will return PEV values of zero (as detailed below) if called.

A method of cgoptimstore .

Y = gridpevevaluate(optimstore, X) produces identical results to the equivalent call to cgoptimstore/pevEvaluate

Y = gridpevevaluate(optimstore, X, objconname) returns PEV values of zero for the objectives/constraints specified in the cell array objconname .

Y = gridpevevaluate(optimstore, X, objconname, datasetname) returns PEV values of zero for the specified objectives/constraints. The return matrix, Y, is of size SIZE(X,1)-by-(NOBJCON)-by-NPTS , where NOBJCON is the number of specified objectives/constraints and NPTS is the number of rows in P.

Y = gridpevevaluate(optimstore, X, objconname, datasetname, rowind) returns PEV values of zero for the specified objectives/constraints. Y is a LENGTH(ROWIND) by LENGTH(OBJCONNAME) by NPTS matrix.

See Also

pevEvaluate

Purpose

Return whether all free variables are scalars

Syntax

```
stat = isScalarFreeVariables(optimstore)
```

Description

Return whether all the free variables are scalars. A method of cgoptimstore .

stat = isScalarFreeVariables(optimstore) returns TRUE if all the free variables are scalars and FALSE otherwise.

nEvaluate

Purpose

Natural evaluation of optimization objectives and constraints

Syntax

```
[y, ysums] = nEvaluate(optimstore, x)
Y = nEvaluate(optimstore, x, itemNames)
Y = nEvaluate(optimstore, x, itemNames, datasetName)
Y = nEvaluate(optimstore, x, itemNames, datasetName, rowind)
```

Description

Natural evaluation of optimization objectives and constraints. A method of cgoptimstore .

Y = nEvaluate(optimstore, x) evaluates the raw values of all of the optimization objectives and constra ints at the free variable values X. X is a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization.

Y = nEvaluate(optimstore, x, itemNames) evaluates the raw values of the objective s and constraints speci Þed in the cell array of strings, itemNames , at the free variable values X. The values of the objectives and cons traints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of objectives and constraints listed in itemNames .

Y = nEvaluate(optimstore, x, itemNames, datasetName) evaluates the speci Þed objectives and constraints at the operating points in the data set speci Þed by the string datasetName .

Y = nEvaluate(optimstore, x, itemNames, datasetName, rowind) evaluates the speci Þed objectives and constraints at the points of datasetName given by rowind . X must be a (NRows-by-NFreeVar) matrix where NRows is the length of rowind . rowind must be a list of integer indices in the range [1 NumRowsInDataset] . Y is a (Nrows-by-NItems) matrix.

See Also

evaluate

Purpose

Natural evaluation of optimization constraints

Syntax

```
Y = nEvaluateConstraint(optimstore, x)
Y = nEvaluateConstraint(optimstore, x, itemNames)
```

Description

A method of cgoptimstore .

Y = nEvaluateConstraint(optimstore, X) evaluates all of the optimization constraints at the free variable values x. X must be a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization. The raw values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of constraints in the optimization.

Y = nEvaluateConstraint(optimstore, X, itemNames) evaluates the constraints specified in the cell array of strings, itemNames , at the free variable values X. The raw values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of constraints listed in itemNames .

See Also

evaluateObjective , evaluateNonlcon

nEvaluateNonlcon

Purpose

Natural evaluation of optimization nonlinear constraints

Syntax

```
y = nEvaluateNonlcon(optimstore, x)
Y = nEvaluateNonlcon(optimstore, x, itemNames)
```

Description

Natural evaluation of optimization n onlinear constraints. A method of cgoptimstore .

Y = nEvaluateNonlcon(optimstore, x) evaluates all of the optimization nonlinear constraints at the free variable values X. X must be a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization. The raw values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of nonlinear constraints in the optimization.

Y = nEvaluateNonlcon(optimstore, x, itemNames) evaluates the nonlinear constraints speci Þed in the cell array of strings, itemNames , at the free variable values X. The raw values of the constraints are returned in Y, which is of size (NPoints-by-NItems) where NItems is the number of nonlinear constraints listed in itemNames .

See Also

evaluateObjective ; evaluateNonlcon

Purpose

Natural evaluation of optimization objectives

Syntax

```
y = nEvaluateObjective(optimstore, x)
Y = nEvaluateObjective(optimstore, x, itemNames)
```

Description

Natural evaluation of optimization objectives. A method of cgoptimstore .

`Y = nEvaluateObjective(optimstore, x)` evaluates all of the optimization objectives at the free variable values `X`. `X` must be a (NPoints-by-NFreeVar) matrix where `NPoints` is the number of points to be evaluated and `NFreeVar` is the number of free variables in the optimization. The raw values of the objectives are returned in `Y`, which is of size (NPoints-by-NItems) where `NItems` is the number of objectives in the optimization.

`Y = nEvaluateObjective(optimstore, x, itemNames)` evaluates the objectives specified in the cell array of strings, `itemNames` , at the free variable values `X`. The raw values of the objectives are returned in `Y`, which is of size (NPoints-by-NItems) where `NItems` is the number of objectives listed in `itemNames` .

See Also

evaluateObjective ; evaluateNonlcon

Purpose

Create/alter optimization OPTIONS structure

Syntax

```
options = optimset(optimstore)
options = optimset(optimfunction, optimstore)
options = optimset(optimfunction, optimstore)
options = optimset(..., 'param1',value1,...)
```

Description

Create/alter optimization OPTIONS structure. A method of cgoptimstore .

options = optimset(optimstore) creates an optimization options structure that can be used with Optimization Toolbox functions. with the named parameters altered with the speci Þed values. Any parameters speci Þed in the optimization that match (by name) those in the default options structure are copied into options .

options = optimset(oldopts, optimstore) creates a copy of oldopts and copies matching parameters from the optimization into it.

options = optimset(optimfunction, optimstore) creates an options structure with all the pa rameter names and default values relevant to the optimization function named in optimfunction and then copies matching parameters from the optimization into it.

options = optimset(..., 'param1',value1,...) sets the additional named parameters to the speci Þed values.

See Also

getParam

Purpose

Evaluate prediction error variance (PEV)

Syntax

```
Y = pevEvaluate(optimstore, X)
```

Description

Warning

The evaluation of PEV is no longer supported in cgoptimstore and this method will return PEV values of zero (as detailed below) if called.

A method of cgoptimstore .

```
Y = pevEvaluate(optimstore, X, itemnames)
```

returns PEV values of zero for objecti ves/constraints at the free variable values X. X is a (NPoints-by-NFreeVar) matrix where NPoints is the number of points to be evaluated and NFreeVar is the number of free variables in the optimization.

```
Y = pevevaluate(optimstore, X, objconname, datasetname)
```

returns PEV values of zero for the objectives/constraints at the operating points in the data set speci Þed by the string datasetname .

```
Y = pevevaluate(optimstore, X, objconname, datasetname, rowind)
```

returns PEV values of zero for the speci Þed objectives/constraints at the points of datasetname given by rowind . X must be a (NRows-by-NFreeVar) matrix where NRows is the length of rowind . rowind must be a list of integer indices in the range [1 NumRowsInDataset] .Y is a (Nrows-by-NItems) matrix.

See Also

gridPevEvaluate

removeConstraint

Purpose

Remove constraint from optimization

Syntax

```
obj = removeConstraint(obj, sLabel)
```

Description

Remove a constraint from the optimization. A method of cgoptimoptions .

obj = removeConstraint(options, label) removes the placeholder for the constraint referred to by the string label .

See Also

getModelConstraints , getLinearConstraints , addModelConstraint , addLinearConstraint

Purpose

Remove free variable from optimization

Syntax

```
obj = removeFreeVariable(obj, sLabel)
```

Description

Remove a free variable from the optimization. A method of cgoptimoptions .

options = removeFreeVariable(options, label) removes the placeholder for the free variable referred to by the string label .

See Also

getFreeVariables , addFreeVariable

removeObjective

Purpose

Remove objective from optimization

Syntax

```
obj = removeObjective(obj, sLabel)
```

Description

Remove an objective from the optimization. A method of cgoptimoptions .

options = removeObjective(options, label) removes the placeholder for the objecti ve referred to by the string label .

See Also

getObjectives , addObjective

Purpose

Remove operating point set from optimization

Syntax

```
obj = removeOperatingPointSet(obj, sLabel)
```

Description

Remove an operating point set fro m the optimization. A method of cgoptimoptions .

options = removeOperatingPointSet(options, label) removes the placeholder for the operating po int set referred to by the string label .

See Also

getOperatingPointSets , addOperatingPointSet

removeParameter

Purpose

Remove parameter from optimization

Syntax

```
obj = removeParameter(obj, sLabel)
```

Description

Remove a parameter from the optimization. A method of cgoptimoptions .

Removes the placeholder for the p arameter referred to by the string label .

See Also

getParameters , addParameter

Purpose

Set how optimization constraints are to be used

Syntax

```
options=setConstraintsMode(options, modestr)
```

Description

A method of `cgoptimoptions`. Sets the mode that governs how the user can set up constraints for the optimization in CAGE.

When `modestr = any`, the user can add any number of constraints.

When `modestr = fixed`, the user can only edit the constraints that are added by the user-defined optimization function.

See Also

`getConstraintsMode`, `addModelConstraint`, `addLinearConstraint`

setDescription

Purpose	Provide description for optimization function
Syntax	options=setDescription(options, desc)
Description	A method of cgoptimoptions . Sets the description for the optimization object to be the string desc .
See Also	getDescription

Purpose

Set enabled status for optimization function

Syntax

```
options = setEnabled(options, status)
```

Description

A method of `cgoptimoptions`. Sets the optimization function enabled status. `status` must be true or false. When an optimization is disabled, you can still register it with CAGE but are not allowed to create new optimizations using it.

See Also

getEnabled

setExitStatus

Purpose

Set exit status information for optimization

Syntax

```
optimstore = setExitStatus(optimstore, exitflag, termmsg)
```

Description

Set exit status information for the optimization. A method of cgoptimstore.

```
optimstore = setExitStatus(optimstore, exitflag, termmsg)
```

sets termination status information in the optimstore. exitflag is an integer which determines whether the optimization has terminated successfully. A value of exitflag > 0 indicates success, and exitflag <=0 indicates failure. In any event, a termination message can be passed back to the optimization through termmsg.

See Also

See the example mbcOSworkedexample, used in the optimization tutorial “Worked Example Optimization”.

Purpose

Set optimal values of free variables

Syntax

OUT = setFreeVariables(optimstore, results)

Description

Sets the optimal values of the free variables, as returned by the optimization, into the optimstore . A method of cgoptimstore .

results is a npts by nfreevar matrix containing the optimal values of the free variables. nsol is the number of solutions and nfreevar is the number of free variables.

Note This function must be called at the end of the optimization for the optimal values to be stored.

See Also

getFreeVariables

setFreeVariablesMode

Purpose

Set how optimization free variables are used

Syntax

```
options = setFreeVariablesMode(options, modestr)
```

Description

A method of cgoptimoptions. Sets the mode that governs how the user is allowed to set up free variables for the optimization in the CAGE GUI.

When modestr = 'any', the user is allowed to add any number of free variables.

When modestr = 'fixed', the user is only allowed to use the number of free variables that are added by the user-defined optimization function.

See Also

getFreeVariablesMode, addFreeVariable

Purpose

Provide name label for optimization function

Syntax

```
options = setName(options, name)
```

Description

A method of `cgoptimoptions`. Sets the name label for the optimization object to be the string `name`.

See Also

getName

setObjectivesMode

Purpose

Set how optimization obje ctive functions are used

Syntax

```
options = setObjectivesMode(options, modestr)
```

Description

A method of cgoptimoptions . Sets the mode that governs whether the user is allowed to set up objectives for the optimization in the CAGE GUI.

When modestr = 'any' , the user is allowed to add any number of objectives.

When modestr = 'fixed' , the user is only allowed to edit the objectives that are added by the user-de Þned optimization function.

When modestr = 'multiple' , the user is only allowed to run the optimization if he or she has de Þned two or more objectives.

See Also

getObjectivesMode , addObjective

Purpose

Set how optimization operating point sets are used

Syntax

```
options = setOperatingPointsMode(options, modestr)
```

Description

A method of cgoptimoptions. Sets the mode that governs how the user is allowed to set up operating point sets for the optimization in CAGE.

When modestr = 'any', the user is allowed to add any number of operating point sets.

When modestr = 'default', the user is allowed to optionally define a single operating point set to run the optimization over.

When modestr = 'fixed', the number of operating point sets required can be fixed by the optimization function and the user is not allowed to add or remove any using the CAGE GUI.

See Also

getOperatingPointsMode, addOperatingPointSet

setOutput

Purpose

Set diagnostic information for optimization

Syntax

optimstore = setOutput(optimstore, OUTPUT)

Description

Set diagnostic information for the optimization. A method of cgoptimstore.

optimstore = setOutput(optimstore, OUTPUT) sets diagnostic information for the optimization in optimstore. Any diagnostic information is passed to optimstore through the structure, OUTPUT. See the worked example for an example of creating an OUTPUT structure.

See Also

See the example file mbcOSworkedexample, used in the optimization tutorial “Worked Example Optimization”.

Purpose

Set output information for optimization

Syntax

```
optimstore = setOutputInfo (optimstore, exitflag, termmsg, output)
```

Description

Sets output information for the optimization in optimstore . A method of cgoptimstore .

The following information is set:

- exitflag : integer value status ßag indicating why the optimization has terminated. exitflag > 0 implies that the optimization has terminated successfully.
- termmsg : Message that is displayed a t termination of algorithm. Normally used for error messages.
- output : Structure of algorithm statistics for the optimization.

Note This method is obsolete. Use cgoptimstore/setExitStatus and cgoptimstore/setOutput instead.

See Also

setExitStatus , setOutput

setRunInterfaceVersion

Purpose

Get preferred interface to pr ovide evaluation function

Syntax

```
obj = setRunInterfaceVersion(obj, ver)
```

Description

Set the preferred interface to provide the evaluation function. A method of cgoptimoptions .

Sets the Model-Based Calibratio n Toolbox product Version that is emulated when the optimization function's evaluate option is called. If ver is set to 2, the interface provided by Model-Based Calibration Toolbox Version 2 software is activated. If ver is set to 3, the new interface, which Model-Based Calibration Toolbox Version 3 software deÞnes, will be used.

The interface version that the cu rrent version of t he Model-Based Calibration Toolbox product runs is s uperior in its capabilities, however it does contains some backwards incompatibilities with the interface used in version 2. You can use this function in old Model-Based Calibration Toolbox optimization Þles that fail to work with the newer interface.

See Also

getRunInterfaceVersion

Purpose

Set current stop state for optimization

Syntax

setStopState(opt,stop)

Description

Set current stop state for optimization. A method of cgoptimstore .

stop = setStopState(optimstore,stop) sets the current stop state (TRUE or FALSE) for the optimization. Note that this command does not stop an optimization, the opti mization script must do this.

See Also

getStopState

7

Data Sets

This section includes the following topics:

- “Data Sets Views” on page 7-2
- “Setting Up Data Sets” on page 7-4
- “Viewing Data in a Table” on page 7-13
- “Plotting Outputs” on page 7-15
- “Using Color to Display Information” on page 7-18
- “Linking Factors in a Data Set” on page 7-23
- “Assigning Columns of Data” on page 7-25
- “Manipulating Models in Data Set View” on page 7-26
- “Filling Tables from Experimental Data” on page 7-27

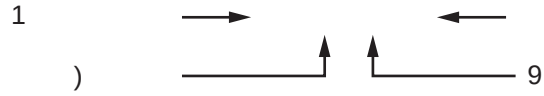
Data Sets Views

The Data Set view has these main functions:

- Validating calibrations with experimental data
- Filling tables by reference to a set of experimental data
- Constructing operating point sets for running optimizations
- Investigating optimization results and using them to Þll tables

For worked examples about data sets, see the Getting Started tutorials.

Data Sets consists of four views. These views display different aspects of the data set. Each view is accessible from the View menu or by clicking the appropriate button on the toolbar.



- Factor Information

  List of all available project expressions, which can be added to the data set for display and evaluation.

- View Data

  Displays the data in a table. Individual entries can be altered. Columns of data can be assigned to CAGE expressions.

- Plot Outputs

  Displays models and features evaluated at the data points (of the data set).

- Fill Table from Data Set

  This mode allows you to Þll tables by reference to experimental data.

Setting Up Data Sets

In this section...

“How to Set Up Data Sets” on page 7-4

“Importing Experimental Data” on page 7-5

“Importing Data from a Table in Your Session” on page 7-7

“Merging Data Sets” on page 7-8

“Specifying the Factors Manually” on page 7-8

“Creating a Factor from the Error Between Factors” on page 7-12

How to Set Up Data Sets

The Data Sets view displays the strategies, tables, and models, etc., as a list of factors in the default Data Set Factors view. You can also display the same factors as columns in a grid, with all factors displayed as columns in the list, by selecting the View Data toolbar button ( ). The data set works over a grid of values, which is not necessarily the same as the normalizers of any included tables in the data set.

You have to set the input factors and their values to define the grid in the data set. You can do this in one of these ways:

- Import experimental data. See “Importing Experimental Data” on page 7-5.
- Import the values from a table in your CAGE session. See “Importing Data from a Table in Your Session” on page 7-7.
- Merge data sets that share the same factors. See “Merging Data Sets” on page 7-8.
- Specify the factors and their values manually. See “Specifying the Factors Manually” on page 7-8.

The next sections describe each of these in detail.

Importing Exper imental Data

You can import ex perimental data to a data set, either to validate a calibration or to use it as the basis for a calibration.

You can import da ta that is stored in the following formats:

- Microsoft Excel spreadsheets
- Comma-separate d value Þles
- MAT- Þles

Importing from Excel or Comma-Separated Value

When you import d ata from either a Microsoft Excel spreadsheet or from a comma-separat ed value Þle, you must ensure that the data is organized in the following m anner:

- The Þrst row can either be column headers (text) or entries (numbers).
- The second row can be a row of units (text), as for importing into the Model Browser. CAGE ignores this row.
- All the other ro w and column entries must be numbers.

Note The Data E ditor can create a tailor-made Excel sheet for you to Þll with data and t hen import. This sheet will be in the format the Model-Based Calibration T oolbox product expects to import data. See “Tailor-Made Excel® Sheets” in the Model-Based Calibration Toolbox Model Browser User’s Guide .

Importing from MAT-files

When you impor t from a MAT- Þle, you must ensure that the Þle contains numbers only , that is, a double array.

To import exp erimental data,

1 Select File - > Import -> Data .

2 In the Þle browser, select the correct Þle to import. This opens the Loading Data from MAT-filename dialog. Use this dialog to select the data in the MAT Þle you wish to import into the data set. Click OK.

This opens the Data Set Import Wizard.

3 Discard any columns of data you do not want to import by selecting the column and clicking the button shown.

4 Click Next.

The following screen asks you to associate variables in your project with data columns in the data.

5 Highlight the variable in the Project Assignments column and the correspondin g data column in the Data Column, then click the assign button, shown.

6 Repeat step 5 until you are satis Þed that you have associated all the variables and data columns. Any una ssigned data columns are treated as output factors.

7 Click Finish to close the dialog box.

This imports your data into the data set. When you have imported your data, you can view your data set.

Importing Data from a Table in Your Session

To import data from a table,

1 Select Data -> Import -> Import from Table .

If your data set is not empty, a dialog box asks whether you want to Fill the data set from the table or Overwrite the data set from the table. Select Fill to use the table values to fill the factors in your data set. Select

Overwrite to disregard all factors in your data set and fill the data set with the input and output factors from the table. A dialog box opens.

2 Select the correct table from your session to import and click OK.

When you have imported your data, you are ready to view the data set.

Merging Data Sets

To merge another data set in your project with the currently selected data set,

1 Select Data -> Import -> Merge Data Set.

The Merge Data Sets dialog box appears containing a list of all data sets in your project.

2 Select the data set you want to merge with the current data set, and click OK.

Columns of inputs and external data are appended to columns with names that match in the current data set.

Outputs (models) and any other columns without matching names are not merged.

The values for any unmatched columns are set to the set point if possible, or zero otherwise.

Specifying the Factors Manually

1 Select the Data Set view by clicking the large Data Sets button in the Data Objects pane.

2 Add a data set to the project by selecting File -> New -> Data Set.

3 Select the factors. (See “Selecting the Factors” on page 7-9.)

4 Build the grid. (See “Manually Setting Values of the Input Variables” on page 7-11.)

Once you have completed these steps you can view the data set.

This section describes

- “Selecting the Factors” on page 7-9
- “Manually Setting Values of the Input Variables” on page 7-11

Selecting the Factors

Clicking the Factors View button in the toolbar ( ). This displays two list boxes.

- The upper list shows all factors within the data set. You can sort factors by clicking the column headings.
- The lower list shows CAGE project expressions.

You can use this view to add factors to or remove factors from the data set.

To add a factor to a data set,

- Right-click a factor and select Add to Data Set from the context menu.

- Alternatively, select the factor or factors that you want to add to the data set from the list in the lower Project Expressions pane, then select Data > Factors > Add to Data Set .

  To make multiple selections, use the standard Shift +click or Ctrl +click.

To remove a factor from a data set,

1 Select the factor or factors that you want to remove from the data set.

2 Right-click and select Remove from Data Set , or select the menu item Data -> Factors -> Remove From Data Set .

Note Links between the two lists are always preserved, so clicking load in the upper list also selects load in the lower list. In other words, you can copy or remove from either list and the relevant results appear in both.

Manually Setting Values of the Input Variables

Clicking the Build Grid toolbar button ( ) or selecting Data -> Build Grid enables you to set the values of the input variables for the data set.

To build a full factorial grid,

1 Select Data -> Build Grid .

2 Select the factor that you want to define a grid for.

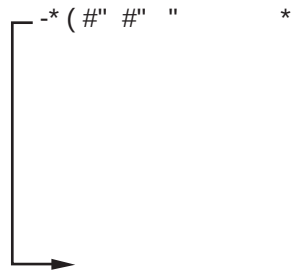
3 Set the grid for the factor.

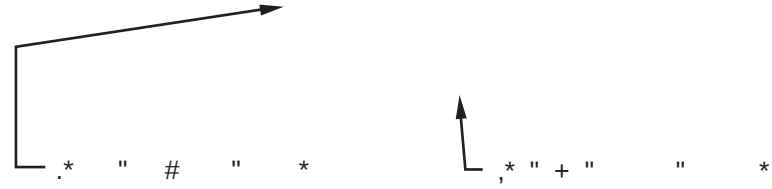
  To set a grid of 5, 10, 15, 20, 25, 30, input the following: 5:5:30 , where the first number is the minimum, the second is the step size, and the last number is the maximum value.

4 Check the size of the data set in the pane. The current size reported at the bottom of the dialog is the size if you click Cancel to leave the data set unchanged. The projected size is created if you click OK . In the following example, the projected size of 45 you can see is obtained by multiplying the number of points for each factor with a grid (in this case, 3 * 5 * 3).

5 Select the next factor that you want to de Þne a grid for.

6 When you have set the grids for all the factors, click OK .





Creating a Factor from the Error Between Factors

To create a factor that is the difference between two other factors,

1 Highlight the two factors, using Ctrl+click or Shift+click .

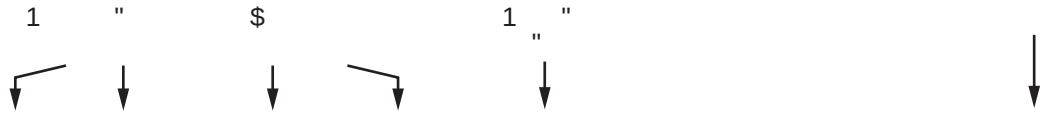
2 Select Create Error from the right-click menu on either column head.

This creates a new factor that is the difference between the two other factors.

Viewing Data in a Table

Click the View Data button ( ) in the toolbar or select View -> Data to display the data in tabular form and a list of the current items in the project.

Note that this view is only enabled if you have a grid of points at which to evaluate and display the models and variables. This grid is not necessarily derived from the normalizers of any t ables included in the data set. You can set the grid by importing experimenta l or table data, or by using the Build Grid toolbar b utton ( ). See “Setting Up Data Sets” on page 7-4.



Columns are color coded by factor type:

- Input factors are white.

- Output factors are gray.

Selecting an output column highlights the input columns associated with it by turning the header cells cream.

Standard editing facilities are available. Double-click an input cell to edit the value.

Cut and paste using the desktop clipboard. Cells, columns, and rows can be copied directly to and from other ap plications (for example, Excel).

Note You can only edit input values, not output values.

Plotting Outputs

Use this to plot the outputs of your data sets.

To view a plot,

1 Select View > Plot or click the toolbar button.

2 Select an expression from the list to view.

A plot of the selected output factor appears in the top pane.

3 Use the pop-up menus below the plot to change the factors displayed.

To zoom in on an area of interest,

- Press both mouse buttons simultaneously and drag a rectangle; double-click the graph to return to full size.

Plotting Multiple Selections

You can plot a multiple selection by using standard Ctrl+click and Shift+click operations.

A legend at the top of the screen displays the key to the graph.

When exactly two items are displayed, further plot options are available:

- Plot the Þrst item against the second item (X-Y Selection).
- Display the error using one of the following options:
  - Error
  - Absolute error
  - Relative error (%)
  - Absolute relative error (%)

Using Color to Display Information

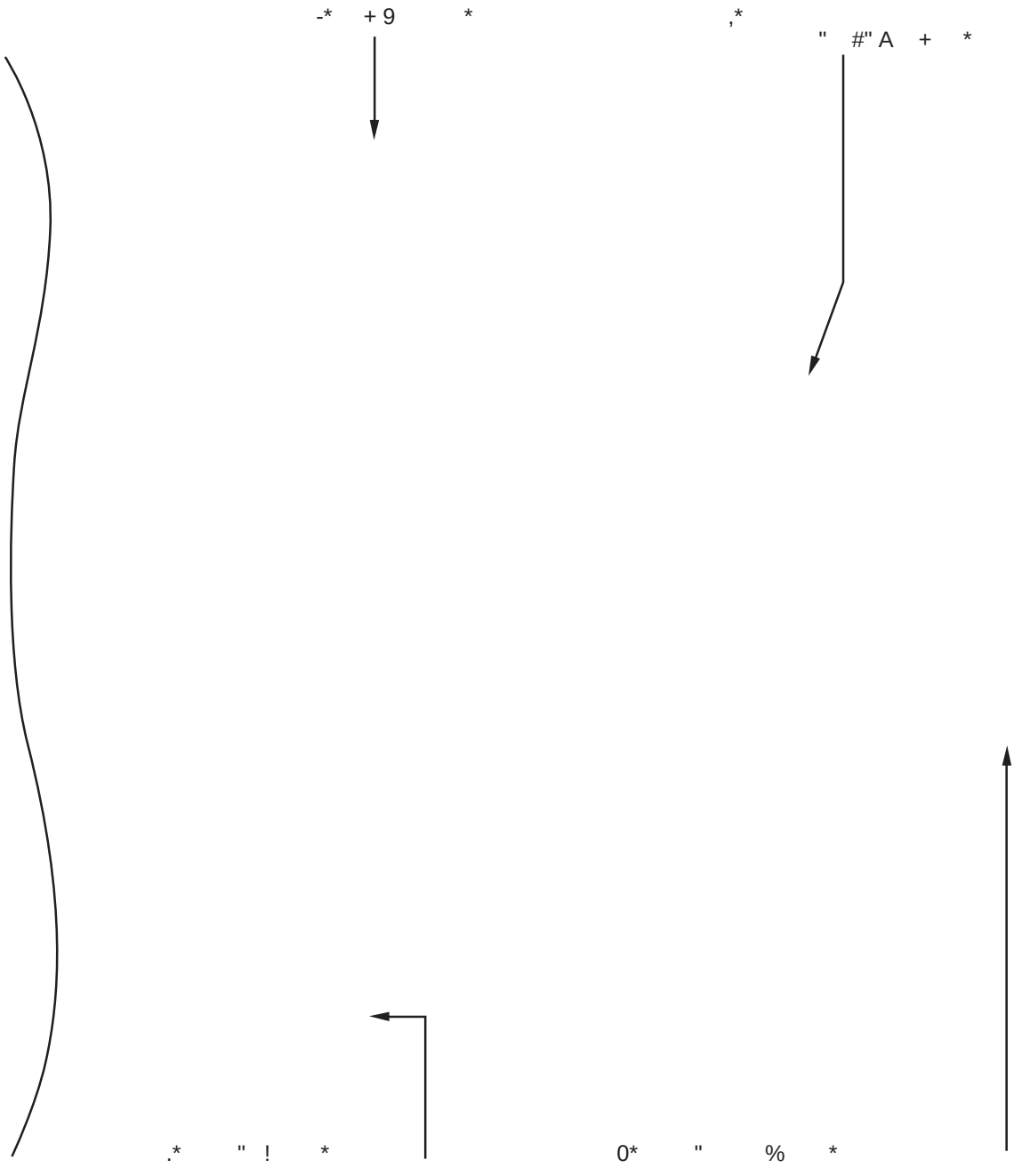
You can use the plot view to display more information by coloring the plots.

1 Select View > Plot or click .

2 Highlight the correct expression in the Output Expressions (Project and Data Set) pane.

3 Select Color by Value from the right-click menu of the plot.

4 Select from the pop-up menu the variable you want to use to color the plot.


-* + 9 *
,*
" #" A + *
.* " ! *
0* " % *

In the following figure, you can see

- A plot of the Sum vs Data Set Point (this is the strategy from a torque feature calibration).
- The points are colored by load.
- For this example it can be seen that, in general, the higher the load, the higher the value of torque.

Restricting the Color

You might be interested in only part of the display; for example, you might only be interested in points with a low engine speed. The various display options enable you to color only the points that you are interested in.

To restrict the color,

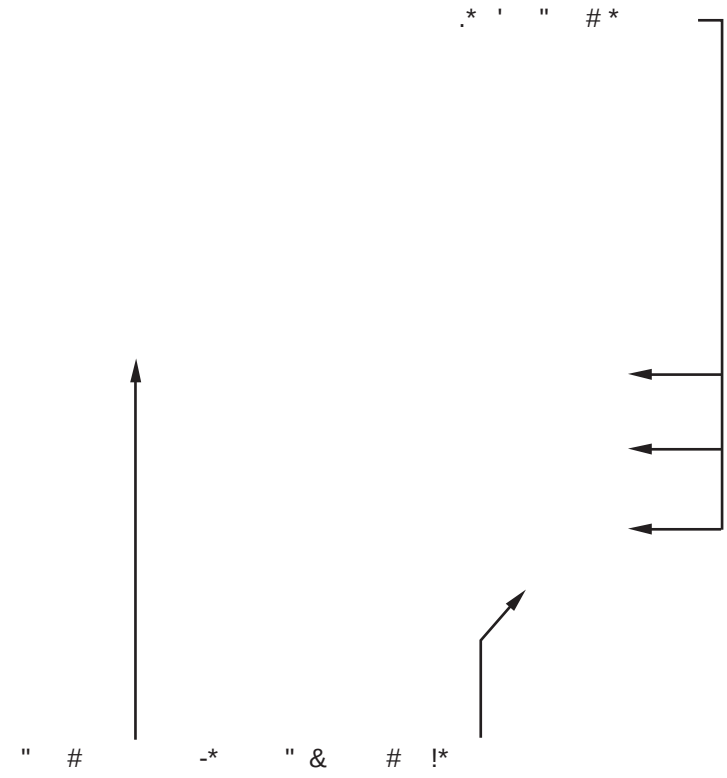
1 Select the Limit range box, or right-click the plot and select Limit Color Range.

Three limit markers appear in the color bar. The colors in the color bar are compressed within the limit markers. This increases the range of colors

over the range you are interested in (b etween the limits), making it easier to see the distribution of points.

2 Adjust the maximum, midpoint, and minimum of the range by dragging the limit markers on the color bar.

3 Examine the data points and those that are outside the range.

Use the right-click menu to alter the view of the points outside the range:

- Select Exclude to remove all points outside the limits from the display.
- Select Color Outside Limits to display all points in color, including those outside the limits. Points outside the l imits are still colored, but only dark red or dark blue, depending on w hich end of the range they are.
- Select No Color Outside Limits to display the points as in the example shown. Points outside the limi ts are plotted as empty circles.