Model-Based Calibration Toolbox

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Programming
- Simulation

CAGE User's Guide

Version 3

The MathWorks

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

Trademarks

Patents

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+)

Contents

1 Getting Started

2 Variables and Models

3 Tables

4 Feature Calibrations

5 Tradeoff Calibrations

6 Optimization

7 Data Sets

8 Surface Viewer

1

Getting Started

This section includes the following topics:

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. 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 gives this reduction in NOX emissions instead of the spark angle that generates maximum torque.

Optimization

CAGE can optimize calibrations with reference to models, including single- and multi-objective optimizations, sum optimizations, user-defined optimizations, and automated tradeoff.

Comparing Calibrations to Data

You can compare your calibrations to experimental data for validation.

For example, after completing a calibration, 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
```

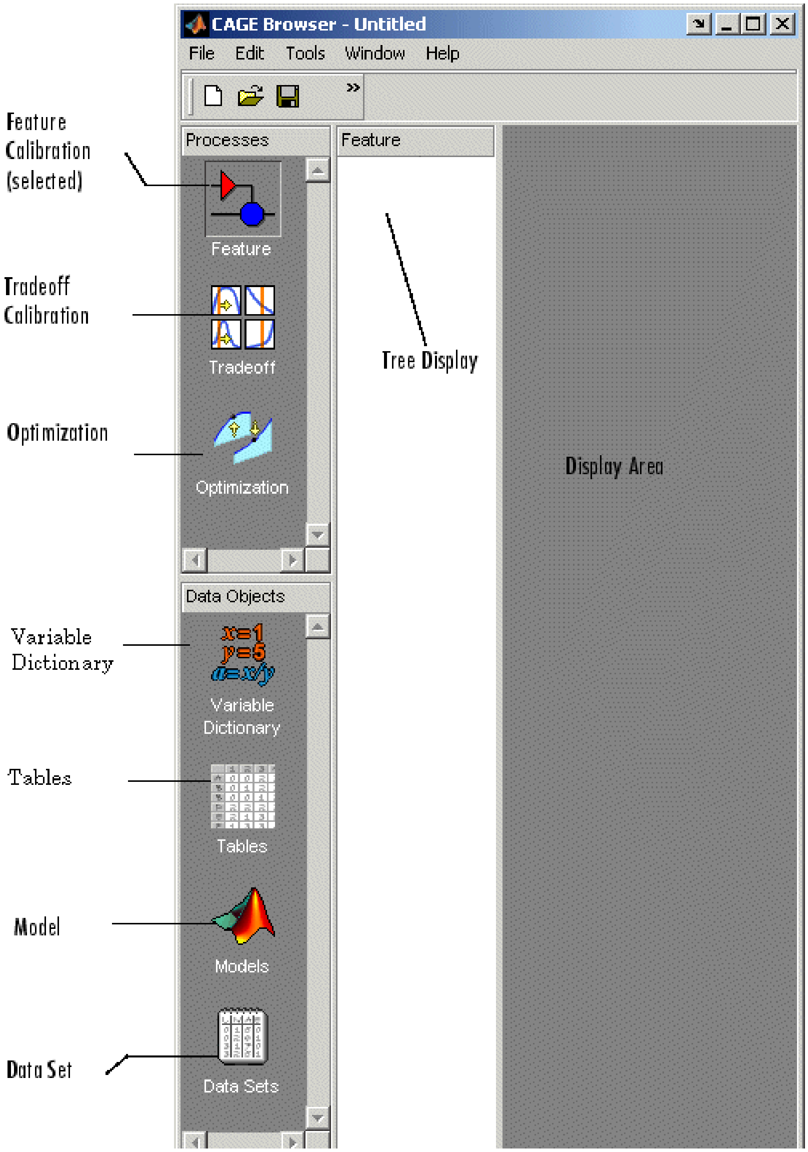
at the MATLAB® command prompt.

Navigating CAGE

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 4-40.

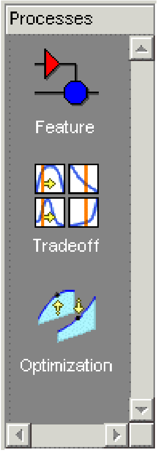
  A feature is a strategy (or collection of tables) and a model used to calibrate those tables. In the **Feature** view, you can fill tables by comparing a strategy to a model. See Chapter 4, “Feature Calibrations”. You can import existing strategies or construct new ones using Simulink® from the feature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

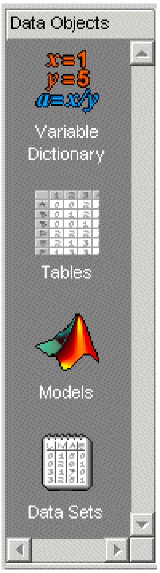
  You can reach the Calibration Manager from the **Feature** and **Tradeoff** process views, and from the **Tables** view, but not **Optimization**. In the Calibration Manager you can set up the size and contents of tables (manually or using existing calibration files) and edit the precision used for values (to match the kind of electronic control unit you are going to use). See “Calibration Manager” on page 3-20.



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 2-3.
- **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 3-3.
- **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 2-11 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.

Learning CAGE

See the Getting Started guide for tutorials and case studies.

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 fulfill 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.

2

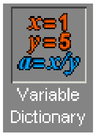
Variables and Models

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

Setting Up Variable Items

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

To view or edit the items in the Variable 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.

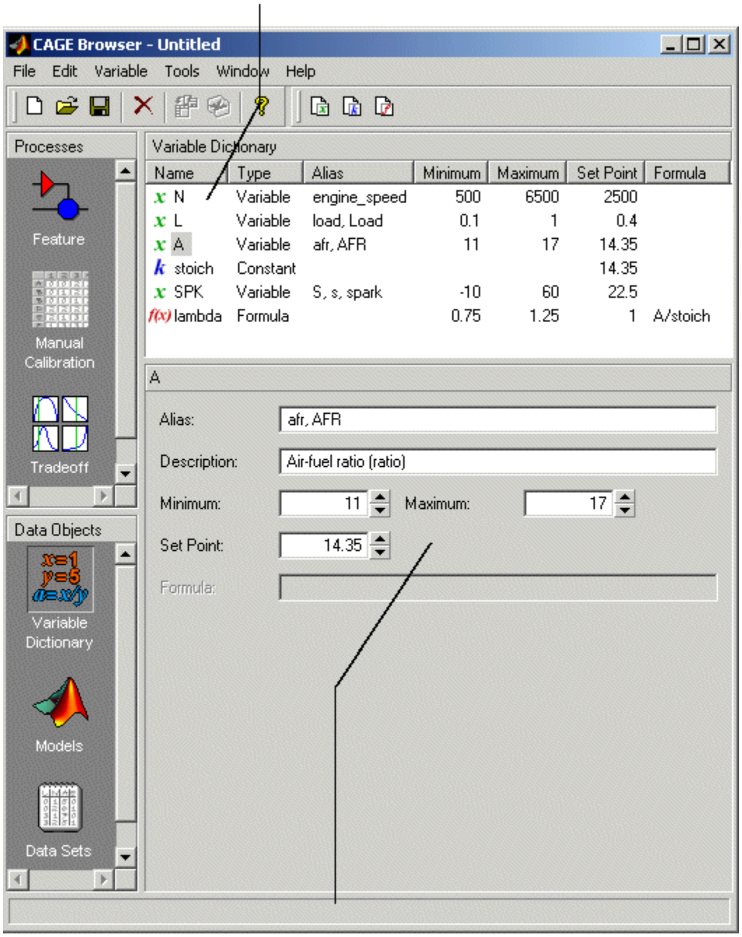
This section describes the following:

- “Importing and Exporting a Variable Dictionary” on page 2-5
- “Adding and Editing Variable Items” on page 2-6
- “Using the Variable Menu” on page 2-8
- “Using Aliases” on page 2-9

Note that if you have existing CAGE projects you can use the “CAGE Import Tool” on page 2-24 to import variable items and other CAGE items directly from other projects.

Following is an example of the **Variable Dictionary** view.

List of all the constants, variables, and formulas in the project



Name	Type	Alias	Minimum	Maximum	Set Point	Formula
x N	Variable	engine_speed	500	6500	2500	
x L	Variable	load, Load	0.1	1	0.4	
x A	Variable	afr, AFR	11	17	14.35	
k stoich	Constant				14.35	
x SPK	Variable	S, s, spark	-10	60	22.5	
f(x) lambda	Formula		0.75	1.25	1	A/stoich

A

Alias: afr, AFR

Description: Air-fuel ratio (ratio)

Minimum: 11 Maximum: 17

Set Point: 14.35

Formula:

Edit boxes to change the settings of the selected constant, variable, or formula

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 2-24.

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

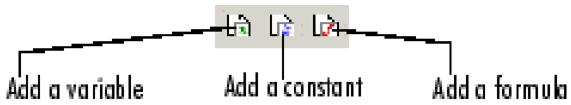
- "Setting Up Variable Items" on page 2-3

- “Adding and Editing Variable Items” on page 2-6

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.

Variable Dictionary Toolbar



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 dictionary 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 stoichiometric 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 required. For instance, CAGE uses the set point when evaluating a model over the range of a different variable.

For example, a simple model for torque 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 the 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

where `afr` is the air/fuel ratio and `stoich` is the stoichiometric 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 first. If you do not use pre-existing variable names then those inputs are created, so be careful to get input names exactly correct. 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 2-3
- "Adding and Editing Variable Items" on page 2-6

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 2-9.

- **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 2-3
- "Using Aliases" on page 2-9

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.

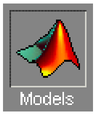
See Also

- "Setting Up Variable Items" on page 2-3

Setting Up Models

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.



This section describes the following:

- "Importing Models" on page 2-13
- "Adding New Function Models" on page 2-15
- "Renaming and Editing Models" on page 2-17

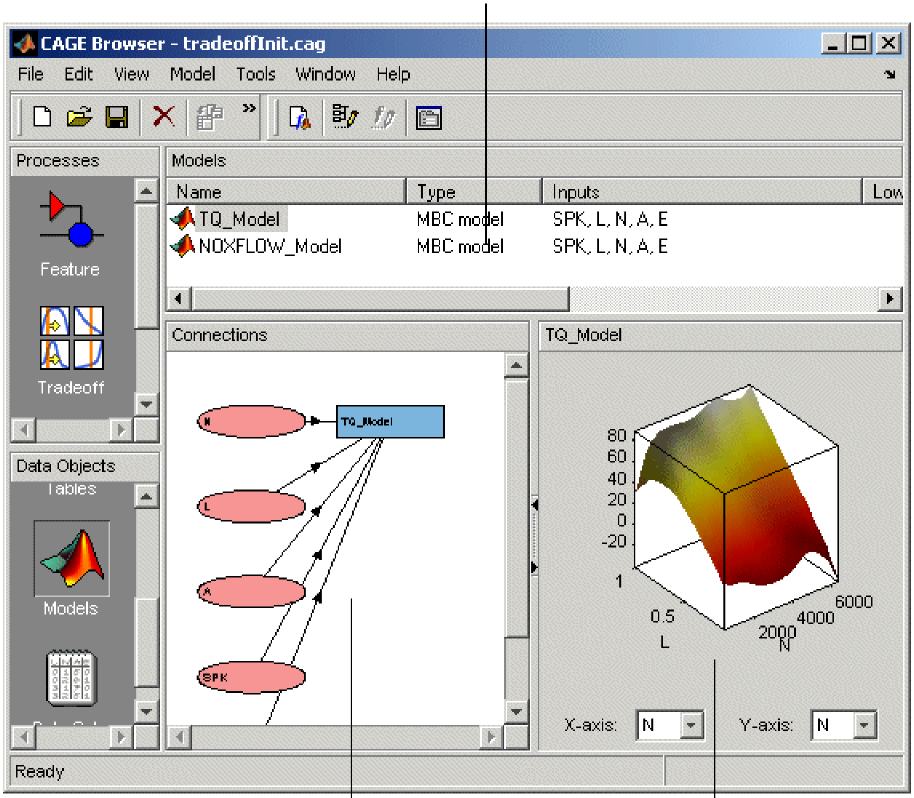
The **Models** view displays the following:

- 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 whether to display constraints and to edit input set points.

  View > Edit Input Set Points opens a dialog where you can edit the set points of your model variables. This alters the model display and also any calculations involving the set points throughout CAGE. This is the same as altering the set points in the Variable Dictionary, see "Using Set Points in the Variable Dictionary" on page 2-6.

Following is an example of the **Models** display.

List of the current models



Name	Type	Inputs	Low
TQ_Model	MBC model	SPK, L, N, A, E	
NOXFLOW_Model	MBC model	SPK, L, N, A, E	

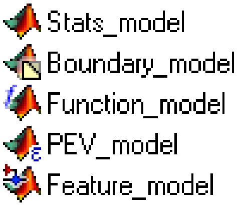
X-axis: N

Y-axis: N

Model connections display

Model 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-defined function model, or a feature model (converted from a feature).



Stats_model
Boundary_model
Function_model
PEV_model
Feature_model

You can use the “Model Properties” on page 2-20 dialog to switch a model output between the model value and the 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 lookup tables by referring to models constructed in the Model Browser.

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

Import Models From Project

You can use the CAGE Import Tool to select models to import from any Model-Based Calibration Toolbox project file 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 2-24 for instructions.

Import Exported Models File

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

1. Select **File > Import > Model**.

2. A file browser dialog opens. Locate the desired file or files. You can select multiple files. Examples can be found in `matlab/toolbox/mbc/mbctraining`.

a If the model is saved as an `.exm` file, select `MBC Model (*.exm)` from the drop-down menu.

b If the model is not saved as an `.exm` file, select `All files (*.*)` from the **Files of type** drop-down menu. For example, the file extension might be accidentally changed.

Click to select the model file then click **Open** .

This opens the Model Import Wizard.

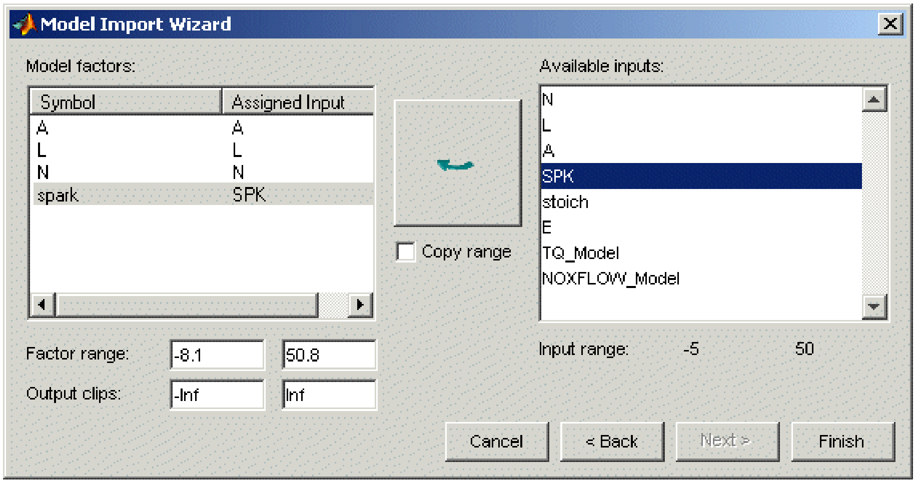
3 Select the models that you want to import by highlighting the models from the list, or click **Select All** if you want every model.

4 Either:

- Select the check box **Automatically assign/create inputs**, then you can click **Finish**.
- Alternatively if you do not want to automatically assign or create inputs, instead click **Next** to match these up manually.

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

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



Symbol	Assigned Input
A	A
L	L
N	N
spark	SPK

Available inputs: N, L, A, SPK, stoich, E, TQ_Model, NOXFLOW_Model

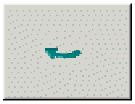
Factor range: -8.1 50.8

Output clips: -Inf Inf

Input range: -5 50

a. Highlight a model factor, `spark`, in the list on the left and the corresponding variable, `SPK`, in the list on the right.

b. Click the select input button, shown.



c. Repeat 5a and 5b for all the model factors.

6. Click **Finish** to close the wizard and return you 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 2-11
- "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**.

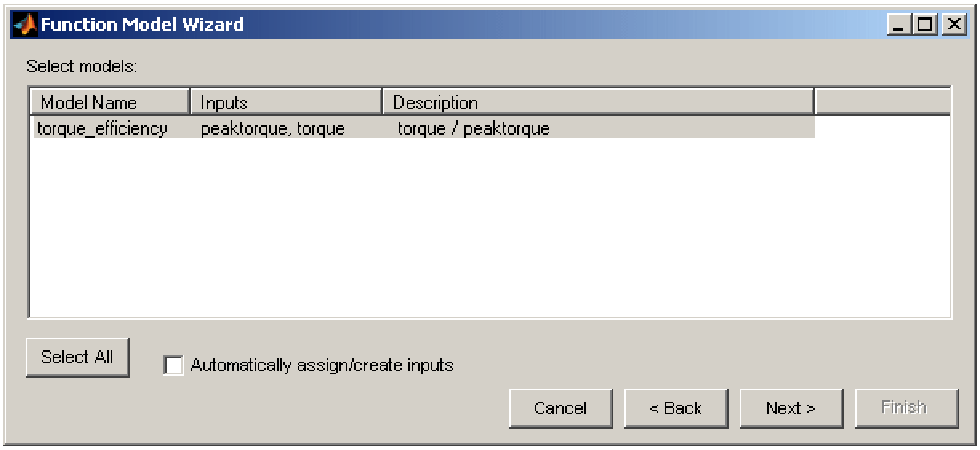
   This opens the Function Model Wizard.

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 incorrectly entered, you cannot click **Next**.

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

5 Click **Next**.



Function Model Wizard

Select models:

Model Name	Inputs	Description
torque_efficiency	peaktorque, torque	torque / peaktorque

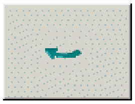
□ Automatically assign/create inputs

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 factor, e.g. `peak_torque`, in the list on the left and the corresponding model, `peak_torque`, in the list on the right.

b Click the select input button, shown.



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 2-11
- “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 selecting a model and clicking the name, or pressing **F2**.

Editing Model Inputs

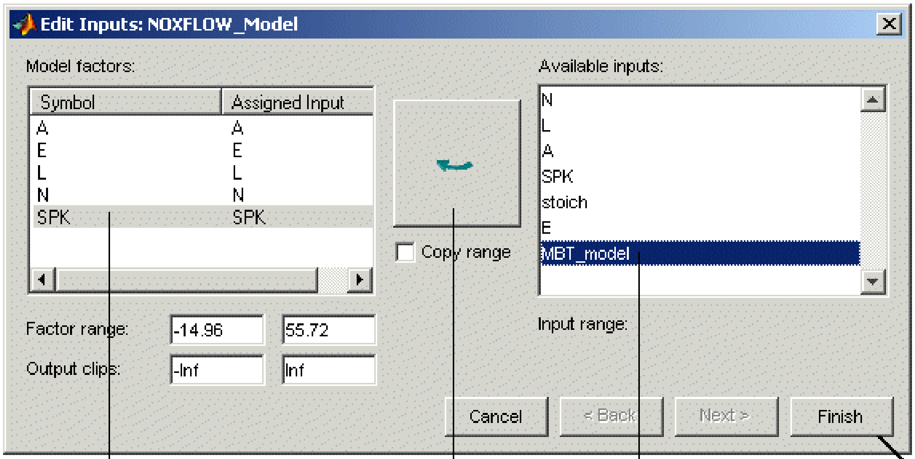
You can adjust a model so that variables, formulas, or other models are the factors of the model. For example, a model of torque depends on the spark angle. In place of the spark angle variable, you can use a model of the maximum brake torque (MBT) as the spark input.

To edit the inputs of a model,

1 Highlight the model.

2 Select **Model > Edit Inputs**.

This opens the Edit Inputs dialog box, shown.



Model factors:

Symbol	Assigned Input
A	A
E	E
L	L
N	N
SPK	SPK

Available inputs:

N
L
A
SPK
stoich
E
MBT_model

□ Copy range

Factor range: -14.96 55.72

Output clips: -Inf Inf

Input range:

Highlight the model factor that you want to change.

Click **Select Input**.

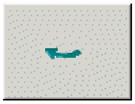
Highlight the new input.

Click **Finish**.

3 Highlight the model factor that you want to adjust, in the list on the left.

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

5 Click the **Select Input** button, shown.



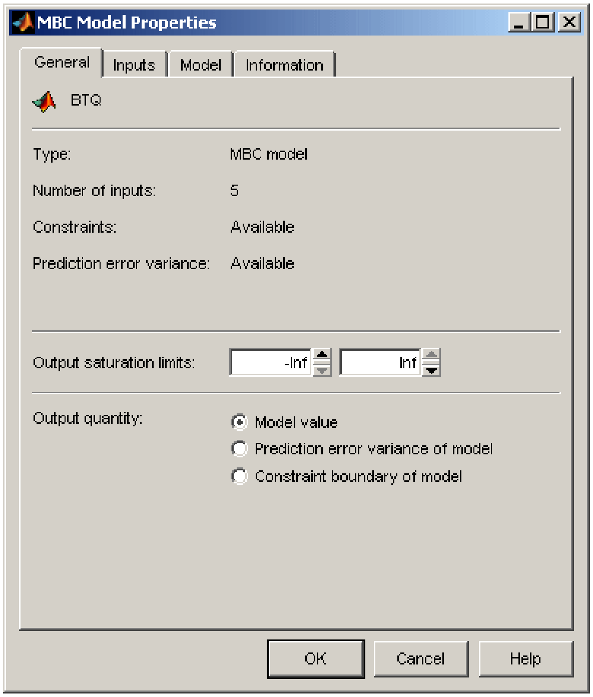
6 To close the dialog box, click **Finish**.

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 2-8.

Model Properties

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

Model Properties: General



BTQ

Type: MBC model

Number of inputs: 5

Constraints: Available

Prediction error variance: Available

Output saturation limits: -Inf Inf

Output quantity:
● Model value
○ Prediction error variance of model
○ Constraint boundary of model

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.

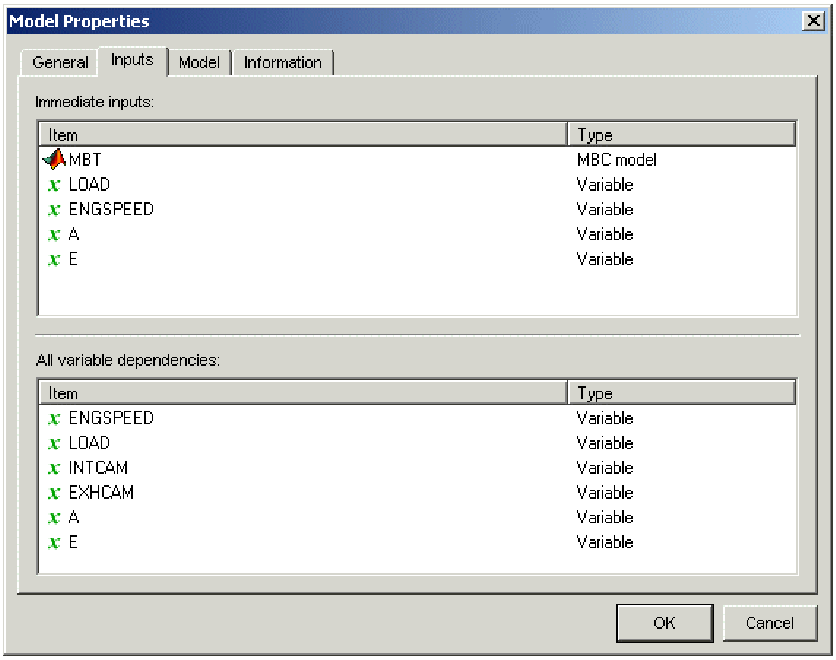
You can use the radio buttons to select the **Output Quantity** to be the

- **Model Value**
- **Prediction error variance of model**

- **Constraint boundary of model**

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

Model Properties: Inputs



Immediate inputs:

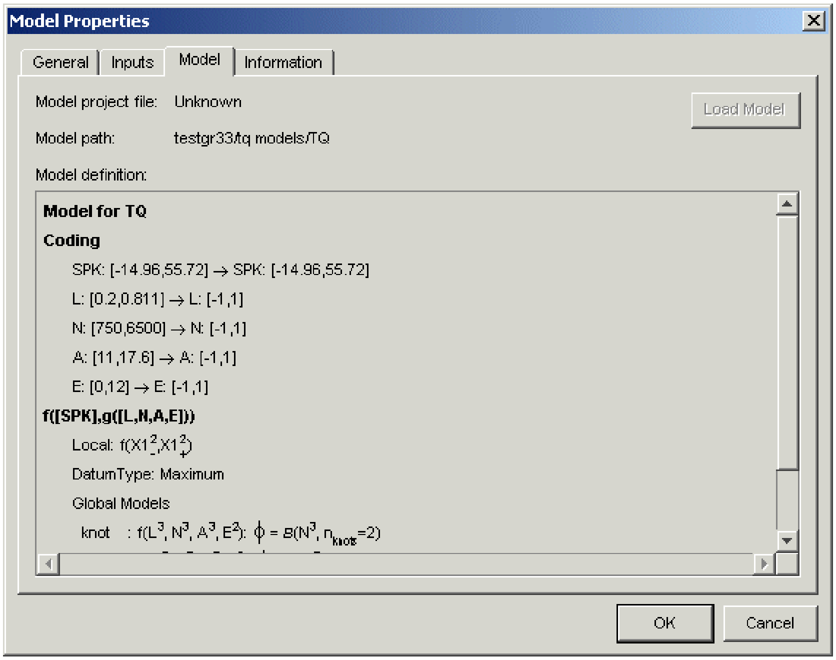
Item	Type
MBT	MBC model
LOAD	Variable
ENGSPEED	Variable
A	Variable
E	Variable

All variable dependencies:

Item	Type
ENGSPEED	Variable
LOAD	Variable
INTCAM	Variable
EXHCAM	Variable
A	Variable
E	Variable

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



Model project file: Unknown

Model path: testgr33/tq models/TQ

Model definition:

Model for TQ

Coding

SPK: [-14.96,55.72] → SPK: [-14.96,55.72]

L: [0.2,0.811] → L: [-1,1]

N: [750,6500] → N: [-1,1]

A: [11,17.6] → A: [-1,1]

E: [0,12] → E: [-1,1]

f([SPK],g([L,N,A,E]))

Local: $f(X1^2_-,X1^2_+)$

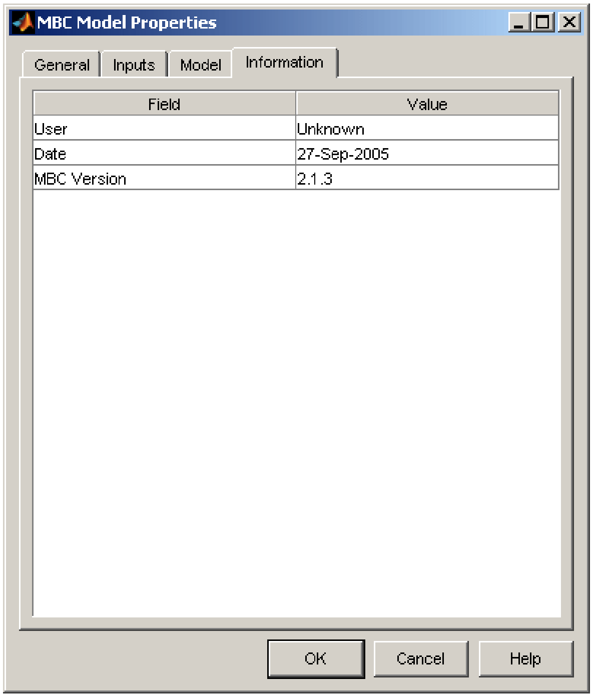
DatumType: Maximum

Global Models

knot : $f(L^3, N^3, A^3, E^2)$: $\phi = B(N^3, n_{knots}=2)$

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



Field	Value
User	Unknown
Date	27-Sep-2005
MBC Version	2.1.3

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 used to create the model. If you added any comments to the export information in the Model Browser Export Models dialog 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, datasets 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:

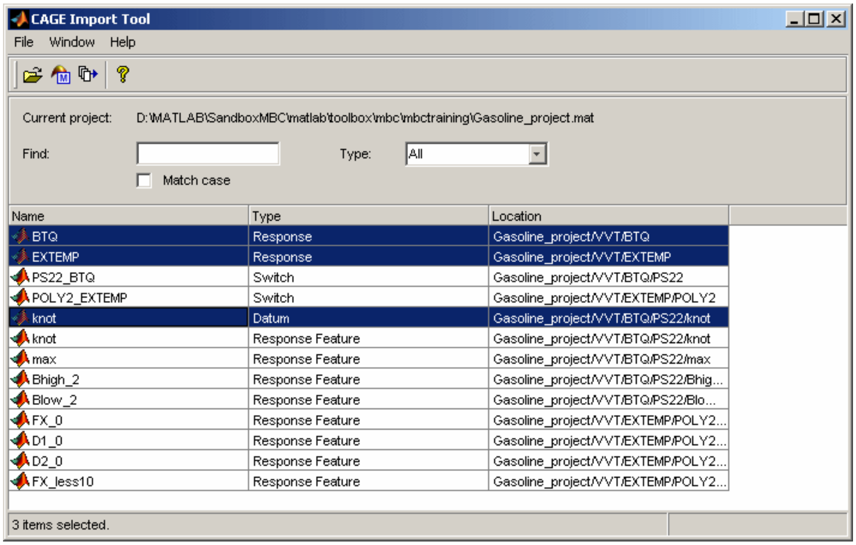
1. Select **File > Import > From Project**.

   The CAGE Import Tool appears.

2. You can choose a project file or import directly from the Model Browser if it is open. Use the toolbar buttons, or select **File > Select Project File**, or **File > Import From Model Browser**.

   If you are choosing a project file, 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.



Current project: D:\MATLAB\SandboxMBC\matlab\toolbox\mbc\mbctraining\Gasoline_project.mat

Find:

Type: All

□ Match case

Name	Type	Location
BTQ	Response	Gasoline_project/VVT/BTQ
EXTEMP	Response	Gasoline_project/VVT/EXTEMP
PS22_BTQ	Switch	Gasoline_project/VVT/BTQ/PS22
POLY2_EXTEMP	Switch	Gasoline_project/VVT/EXTEMP/POLY2
knot	Datum	Gasoline_project/VVT/BTQ/PS22/knot
knot	Response Feature	Gasoline_project/VVT/BTQ/PS22/knot
max	Response Feature	Gasoline_project/VVT/BTQ/PS22/max
Bhigh_2	Response Feature	Gasoline_project/VVT/BTQ/PS22/Bhig...
Blow_2	Response Feature	Gasoline_project/VVT/BTQ/PS22/Blo...
FX_0	Response Feature	Gasoline_project/VVT/EXTEMP/POLY2...
D1_0	Response Feature	Gasoline_project/VVT/EXTEMP/POLY2...
D2_0	Response Feature	Gasoline_project/VVT/EXTEMP/POLY2...
FX_less10	Response Feature	Gasoline_project/VVT/EXTEMP/POLY2...

3 items selected.

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

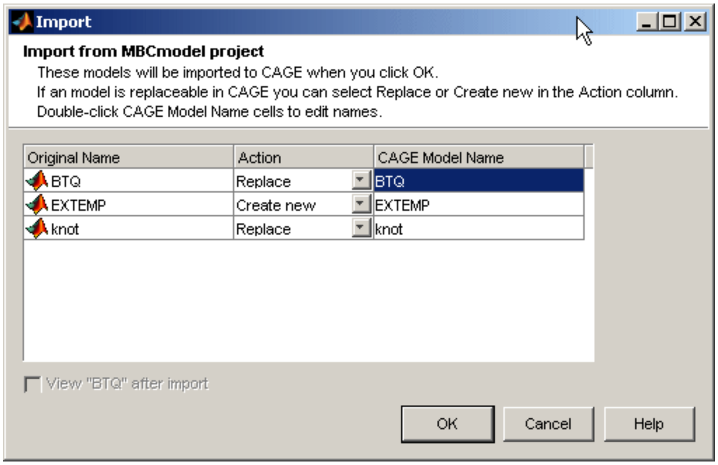
- If you are importing from a Model Browser project you can select `Response`, `Switch`, `Datum` or `Response Feature` from the **Type** list to display a single model type only.

- 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 model names. You can also select the box to **Match case**

4 Click the Import Selected Items toolbar button ( ) or select **File > Import Selected Items**.

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



Import from MBCmodel project

These models will be imported to CAGE when you click OK.
If an model is replaceable in CAGE you can select Replace or Create new in the Action column.
Double-click CAGE Model Name cells to edit names.

Original Name	Action	CAGE Model Name
BTQ	Replace	BTQ
EXTEMP	Create new	EXTEMP
knot	Replace	knot

□ View "BTQ" after import

- Double-click the **CAGE Item Name** column cells to edit item names.
- 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 or select **File > Close** to close the CAGE Import Tool when you have finished importing items.

See also:

- "Importing and Exporting a Variable Dictionary" on page 2-5
- "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**.

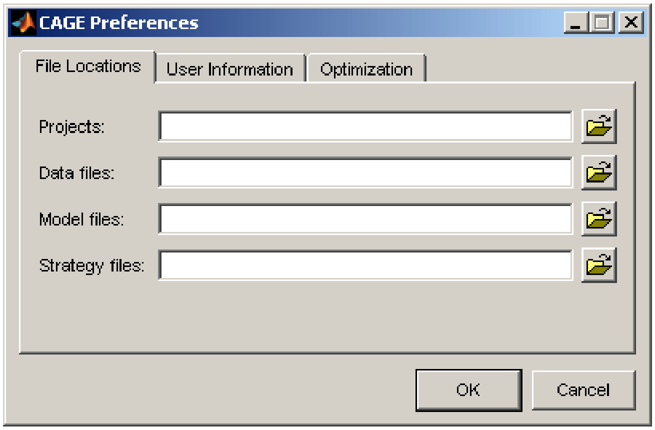
Project files have the file extension `.cag` and store entire CAGE sessions.

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.

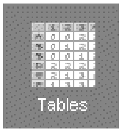
3. Click **OK**.

Tables

This section includes the following topics:

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 3-4
- “Using the History Display” on page 3-16
- “Calibration Manager” on page 3-20
- “About Normalizers” on page 3-30

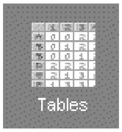
You can use the History display (from any other table or normalizer view in CAGE) to view and reverse changes and revert to previous versions of your tables. Use the Calibration Manager to set up tables manually or from calibration files.

See also

- “Table View” 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

To add or delete tables, you can first select the **Tables** view. 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.

Adding Tables

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 the 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 3-3.

You can rename tables by first selecting the table, then

- Press **F2**, or
- Select **Edit > Rename**.

You can manually calibrate by entering values in any table. You can also fill 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 2-24 to add existing tables from other CAGE project files.

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 first.

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).

Table View

When you select a table in the tree (under feature or tables), you see the **Table** view. The following sections describe:

- "Viewing and Editing a Table" on page 3-8
- "Using the Graph of the Table" on page 3-10
- "Filling a Table by Extrapolation" on page 3-11
- "Table Menu" on page 3-12

Note For feature calibration (filling and optimizing table values by comparing a strategy and a model), see "Calibrating the Tables" on page 4-24.

In CAGE, a table is defined to be either a one-dimensional or a two-dimensional lookup table. One-dimensional tables are sometimes known as characteristic lines or functions. Two-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 3-30.

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

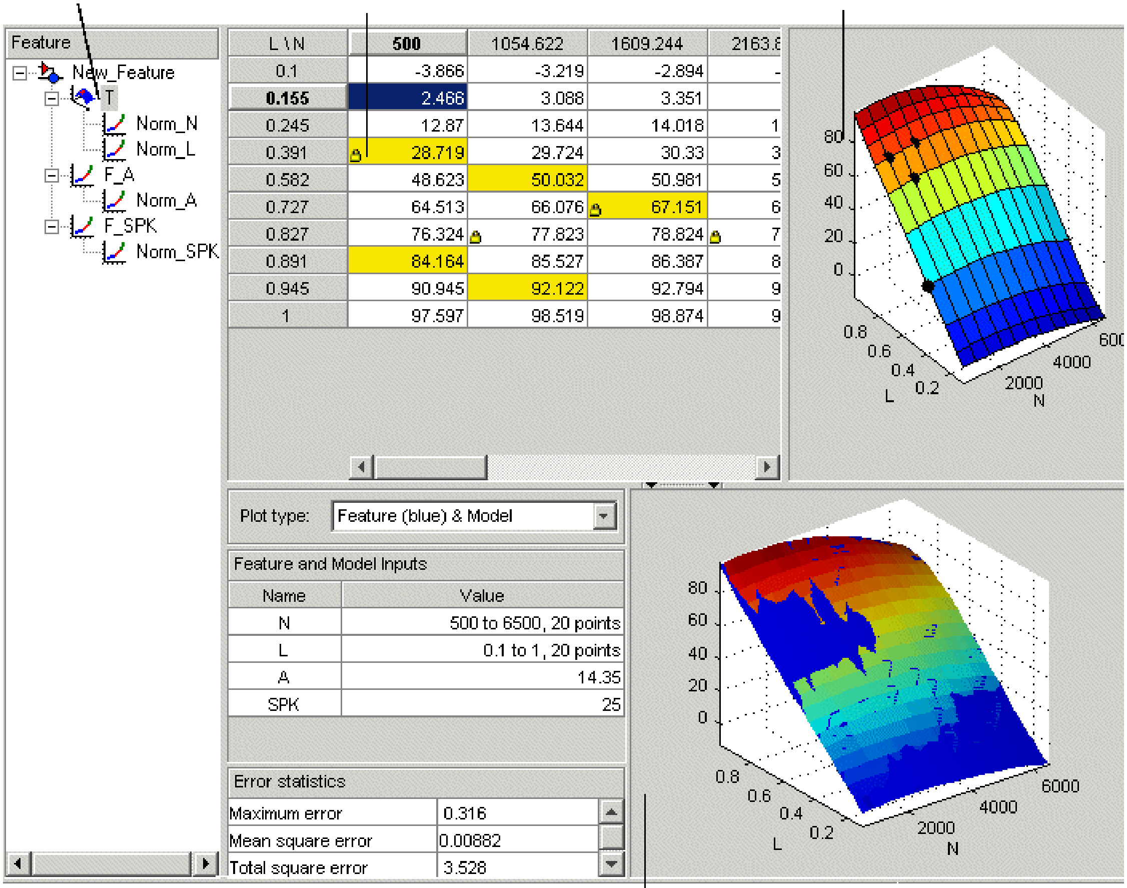
The example following is a feature view. In the Tables view for manual calibration, you do not see the lower comparison pane because you are not comparing tables with a model.

Table Display in Feature Calibration

Selected table node

1. Table

2. Graph of the table



L \ N	500	1054.622	1609.244	2163.8
0.1	-3.866	-3.219	-2.894	-
0.155	2.466	3.088	3.351	
0.245	12.87	13.644	14.018	1
0.391	28.719	29.724	30.33	3
0.582	48.623	50.032	50.981	5
0.727	64.513	66.076	67.151	6
0.827	76.324	77.823	78.824	7
0.891	84.164	85.527	86.387	8
0.945	90.945	92.122	92.794	9
1	97.597	98.519	98.874	9

Plot type: Feature (blue) & Model

Feature and Model Inputs

Name	Value
N	500 to 6500, 20 points
L	0.1 to 1, 20 points
A	14.35
SPK	25

Error statistics

Maximum error	0.316
Mean square error	0.00882
Total square error	3.528

3. Comparison of results

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 necessarily identical 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 3-8.)

2 The graph of the table pane displays the table values graphically. (See “Using the Graph of the Table” on page 3-10.)

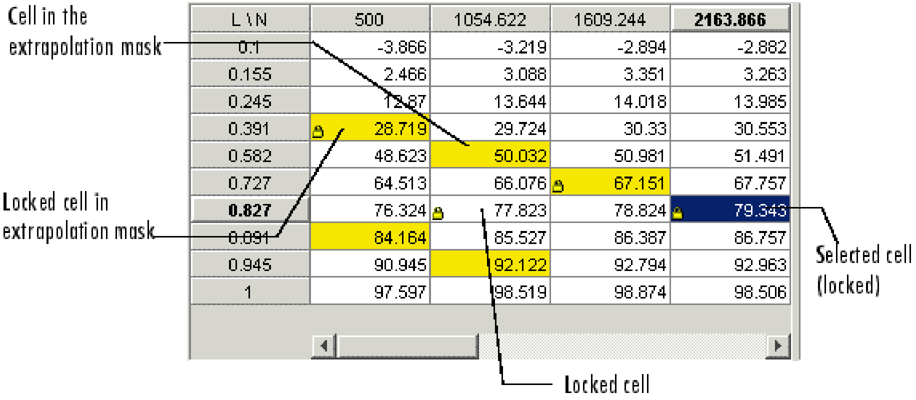
3 The comparison-of-results pane displays a comparison between the current output of the strategy and the feature model. (Only visible when calibrating a feature, see “Inverting a Table” on page 3-39.)

Note You can view the History display by selecting **View > History**. For information, see “Using the History Display” on page 3-16.

This section describes each of these parts in detail.

Viewing and Editing a Table

The table displays the values of your lookup table and displays the breakpoints of the normalizers. For example, the following table shows a lookup table with speed and relative air charge (load) as its normalizers.



L \ N	500	1054.622	1609.244	**2163.866**
0.1	-3.866	-3.219	-2.894	-2.882
0.155	2.466	3.088	3.351	3.263
0.245	12.87	13.644	14.018	13.985
0.391	🔒 28.719	29.724	30.33	30.553
0.582	48.623	50.032	50.981	51.491
0.727	64.513	66.076	🔒 67.151	67.757
0.827	76.324	🔒 77.823	78.824	🔒 79.343
0.891	84.164	85.527	86.387	86.757
0.945	90.945	92.122	92.794	92.963
1	97.597	98.519	98.874	98.506

Cell in the extrapolation mask

Locked cell in extrapolation mask

Selected cell (locked)

Locked cell

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

See also "Filling a Table by Extrapolation" on page 3-11, and "Adjust Cell Values" on page 3-13 for information on applying arithmetic operations to selected cell values or whole tables.

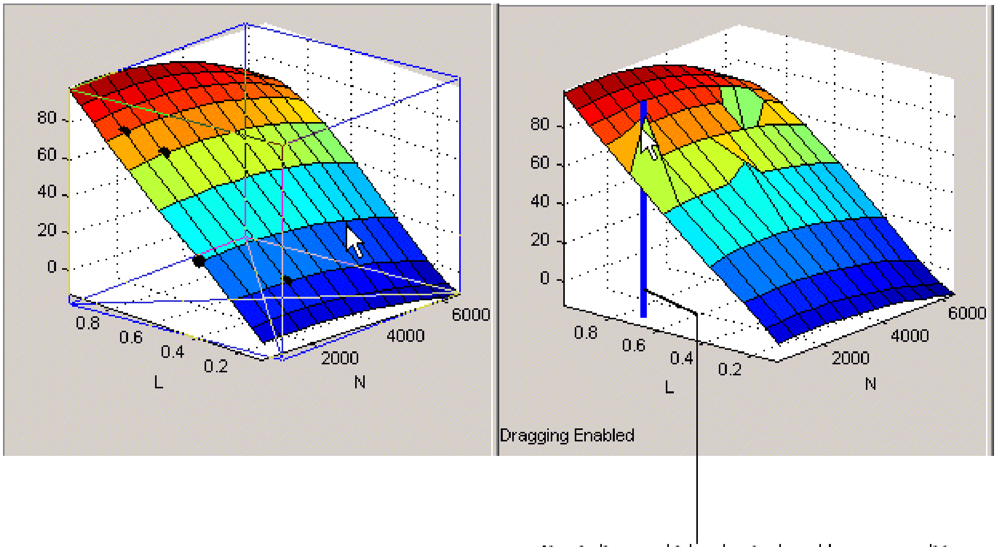
Locking and Unlocking Cell Values

When you are satisfied 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-click 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 behavior. Shown is an example of a graph in dragging and rotation mode.



Line indicates which value in the table you are editing.

- 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 3-16.

Filling a Table by Extrapolation

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

To fill 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 more collinear 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 defines 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 define an extrapolation mask to include all the cells in these ranges. You can then fill 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 also generate the extrapolation mask from the **boundary model** or from the **predicted error** of the model. See “Filling the Table by Extrapolation” on page 4-30.

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 4-24.

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 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. See “Adjust Cell Values” on page 3-13. 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). Only 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 dialog 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 fill 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 3-24.

Adjust Cell Values

This **Table** menu item (or right-click context menu item) opens a dialog where you can specify an arithmetic operation to apply.

1. 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.

2. 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%`’).

3. 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).

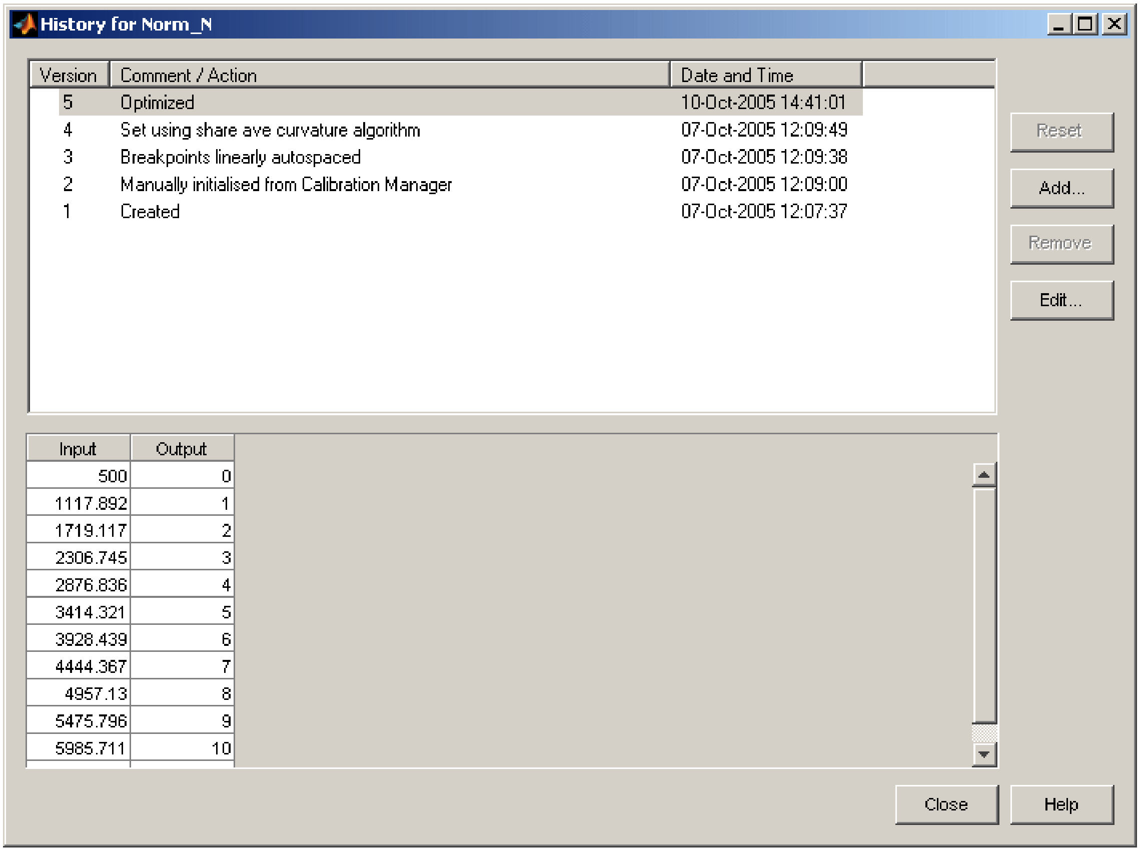
Using the History Display

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 3-17.)
- Compare different versions of tables and normalizers (See “Comparing Versions” on page 3-18.)

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



Version	Comment / Action	Date and Time
5	Optimized	10-Oct-2005 14:41:01
4	Set using share ave curvature algorithm	07-Oct-2005 12:09:49
3	Breakpoints linearly autospaced	07-Oct-2005 12:09:38
2	Manually initialised from Calibration Manager	07-Oct-2005 12:09:00
1	Created	07-Oct-2005 12:07:37

Input	Output
500	0
1117.892	1
1719.117	2
2306.745	3
2876.836	4
3414.321	5
3928.439	6
4444.367	7
4957.13	8
5475.796	9
5985.711	10

The upper pane of the History display lists 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**.

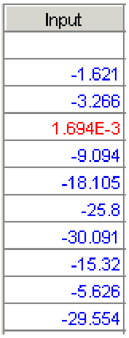
3 In the dialog box, edit the comment.

4 Click **OK**.

Comparing Versions

To compare two different versions of a normalizer 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.



Input
-1.621
-3.266
1.694E-3
-9.094
-18.105
-25.8
-30.091
-15.32
-5.626
-29.554

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 on the toolbar.

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

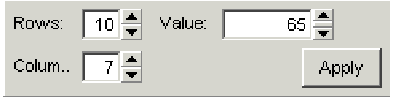
Note that you can enter the required inputs, number of rows and columns and an initial value for table cells when you add a new table using the **File > New** menu items. See “Adding, Duplicating and Deleting Tables” on page 3-4. You can use the Calibration Manager to change the sizes, values and precision of tables.

Setting Up Tables Manually

1 Select the normalizer 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**.



Rows: 10 Value: 65

Colum.. 7

Note When initializing tables for a feature calibration (comparing a model to a strategy) you should think about your strategy. CAGE cannot fill those tables if you try to divide by zero. Modifier 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 4-25.

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

Setting Up Tables Using an Existing Calibration File

1 Open the file by clicking 📂.

This opens the Import Calibration Data dialog box.

2 You can select whether you want to import from File or from ATI Vision. See “Importing and Exporting Calibrations” on page 3-46 for details.

3 If importing from file, browse to the calibration file, select it, and click **Open**. Note that empty data is filtered out and any empty variables will not appear.

Note `tutorialcal.mat` is an example calibration file in the `mbctraining` folder.

If importing from ATI Vision, use the Connection Manager dialog to select the required calibration. See “Importing and Exporting Calibrations” on page 3-46 for instructions.

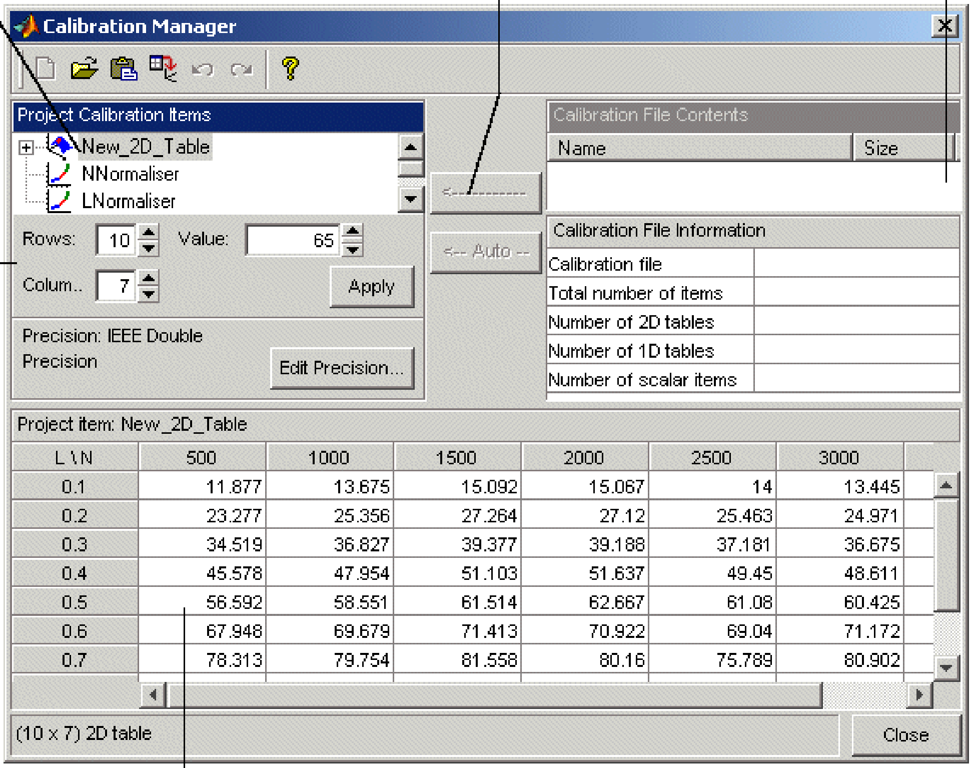
4 Highlight both the table in the **Calibration File Contents** pane and the table in the **Project Calibration Items** pane that you want to associate with it.

5 Associate these two items by clicking <----------.

To associate all the items listed in the **Project Calibration Items** pane with items having the same names listed in the **Calibration File Contents** pane, click <--- Auto ---.

6 To find particular names in a large calibration file, click the **Calibration File Contents** list, and type the first few letters of the item that you are searching for. The cursor moves to the letters specified as you type.

7 Check the display of your table, then click **Close**.



Association buttons

Contents of calibration file

Select the axis or table to be calibrated.

Manually set up the table or normalizer.

Check the display of your table.

Rows: 10 Value: 65

Colum.. 7

Precision: IEEE Double Precision

Calibration File Information	
Calibration file	
Total number of items	
Number of 2D tables	
Number of 1D tables	
Number of scalar items	

Project item: New_2D_Table

L \ N	500	1000	1500	2000	2500	3000
0.1	11.877	13.675	15.092	15.067	14	13.445
0.2	23.277	25.356	27.264	27.12	25.463	24.971
0.3	34.519	36.827	39.377	39.188	37.181	36.675
0.4	45.578	47.954	51.103	51.637	49.45	48.611
0.5	56.592	58.551	61.514	62.667	61.08	60.425
0.6	67.948	69.679	71.413	70.922	69.04	71.172
0.7	78.313	79.754	81.558	80.16	75.789	80.902

(10 x 7) 2D table

Note You can add additional file formats to configure CAGE to work with your processes.

Contact The MathWorks for details about adding file formats at `www.mathworks.com/products/mbc/`.

Copying Table Data from Other Sources

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

1. Open the desired file and copy the array that you want to import.
2. In the Calibration Manager dialog box, click Paste .

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

1. Click Set Up From ASCII File in the toolbar.
2. Select the desired file, then click **Open**.

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

Table Properties

Table properties allow you to edit the precision of the number in selected tables and normalizers according to the way tables 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. Click the **Edit Precision**button in the Calibration Manager dialog box.

   Alternatively, if you highlight a table in a calibration (in the Tables or Feature views), display the table properties by selecting **Table > Properties**.

2. Decide whether you want the precision to be writable, then either select or clear the **Properties Read-only** check box.

3. Select the **Precision type** you require for the table:

   - `Floating Point` (See “Floating-Point Precision” on page 3-25.)
   - `Polynomial Ratio, Fixed Point` (See “Polynomial Ratio, Fixed Point” on page 3-26.)
   - `Lookup Table, Fixed Point` (See “Lookup Table, Fixed Point” on page 3-28.)

The following sections describe these types of precision in detail.

Floating-Point Precision

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

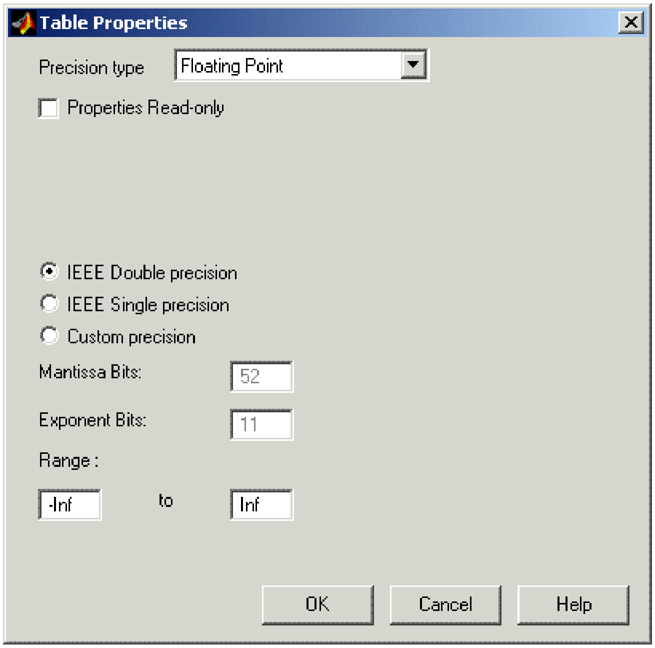


Table Properties

Precision type: Floating Point

□ Properties Read-only

● IEEE Double precision

○ IEEE Single precision

○ Custom precision

Mantissa Bits: 52

Exponent Bits: 11

Range : -Inf to Inf

There are three types of floating-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

You can use the **Range** edit boxes to set a range of values restricting the values in the table.

When you are done, click **OK**.

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)

$$y = \frac{50x + 0}{0 + 255}$$

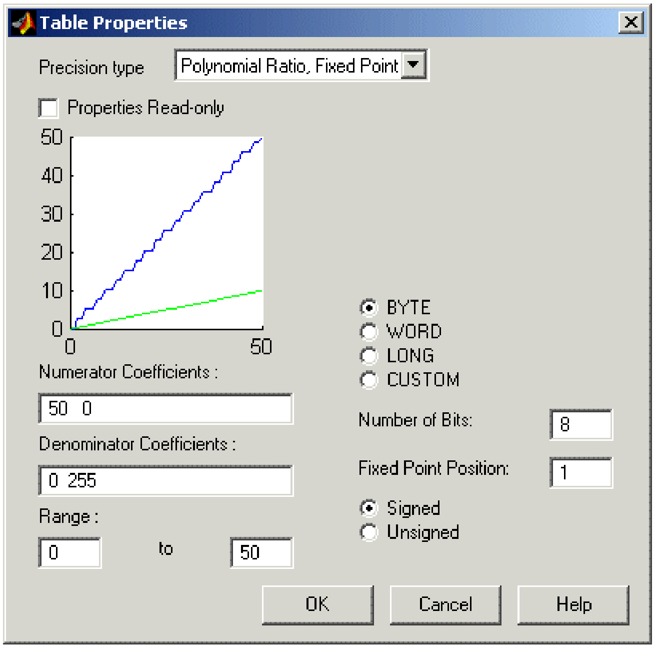


Table Properties

Precision type: Polynomial Ratio, Fixed Point

□ Properties Read-only

Numerator Coefficients : 50 0

Denominator Coefficients : 0 255

Range : 0 to 50

● BYTE
○ WORD
○ LONG
○ CUSTOM

Number of Bits: 8

Fixed Point Position: 1

● Signed
○ Unsigned

OK Cancel Help

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. Determine the range of values that you want to have in the table. In the preceding example, enter `0 50`.

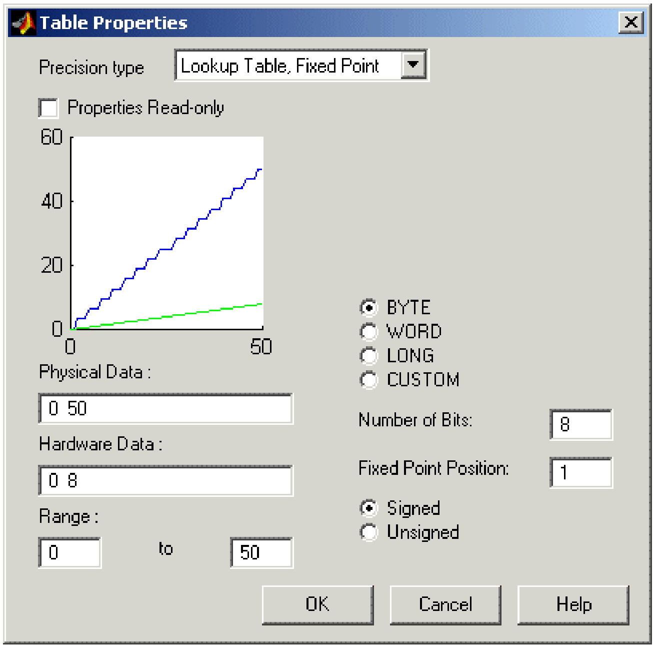
To edit the size of the precision, choose from

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

Next, select the radio button to determine whether you want the numbers to be **Signed** (negative and positive) or **Unsigned** (nonnegative).

You can use the **Range** edit boxes to set a range of values restricting the allowable values in the table, though there are also limits inherent in the storage format.

Lookup Table, Fixed Point



Precision type: Lookup Table, Fixed Point

☐ Properties Read-only

Physical Data : 0 50

Hardware Data : 0 8

Range : 0 to 50

● BYTE
○ WORD
○ LONG
○ CUSTOM

Number of Bits: 8

Fixed Point Position: 1

● Signed
○ Unsigned

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. In the example shown it is`0 50`.

2 Select the **Hardware Data** and enter the range to store. In the example shown it is `0 8`.

3 Determine the range of values that you want to have in the table. In the example shown it is`0 50`.

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)

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

Choose whether you want the numbers to be **Signed** (negative and positive) or **Unsigned** (nonnegative) by clicking the radio buttons.

You can use the **Range** edit boxes to set a range of values restricting the values in the 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 3-32

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 4-12.

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

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

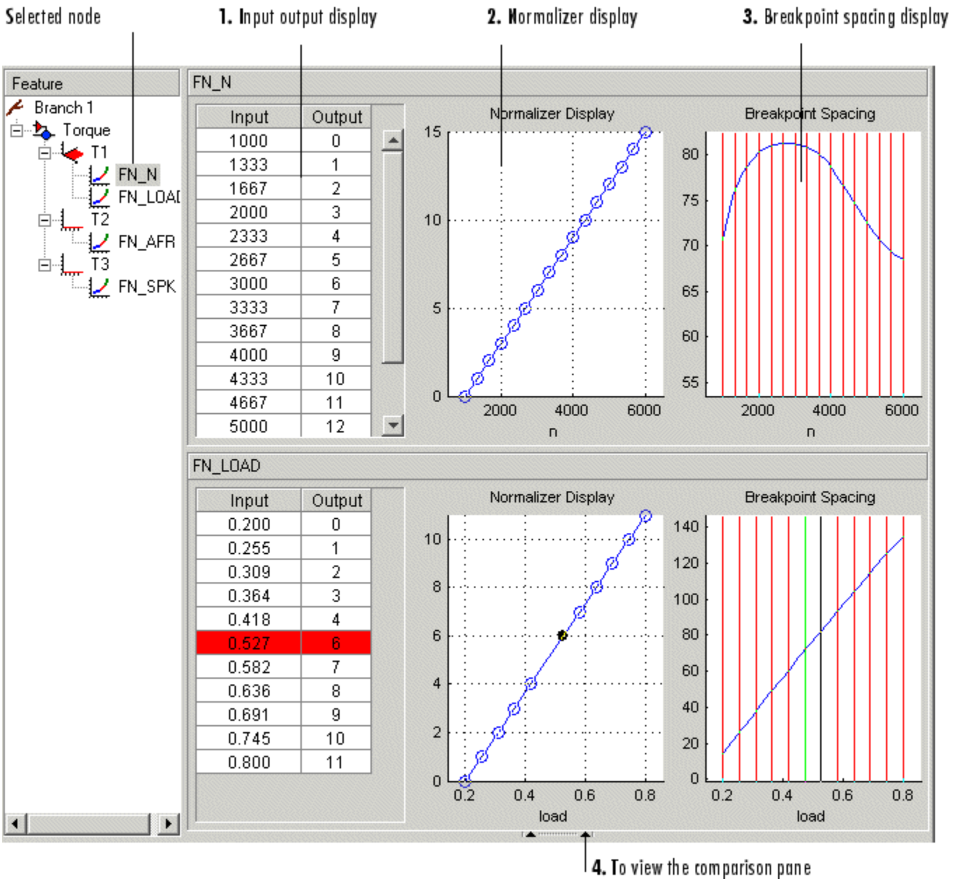
Note If the table has two normalizers, both are displayed, the normalizer for the table columns at the top, the normalizer for the table rows below. This is true whichever normalizer on the tree is highlighted.

See “Editing Breakpoints” on page 3-33.

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

- “Input/Output Display” on page 3-35. This shows the breakpoints of the normalizer.
- “Normalizer Display” on page 3-35. This is a graphical representation of the **Input Output** display.
- “Breakpoint Spacing Display” on page 3-36. This shows a slice of the model (in feature calibration) over the range of the breakpoints.
- The comparison pane (for feature calibration with reference to a model). For information, see “Viewing the Normalizer Comparison Pane” on page 4-21.

Normalizer View



Selected node

1. Input output display

2. Normalizer display

3. Breakpoint spacing display

4. To view the comparison pane

FN_N

Input	Output
1000	0
1333	1
1667	2
2000	3
2333	4
2667	5
3000	6
3333	7
3667	8
4000	9
4333	10
4667	11
5000	12

FN_LOAD

Input	Output
0.200	0
0.255	1
0.309	2
0.364	3
0.418	4
0.527	6
0.582	7
0.636	8
0.691	9
0.745	10
0.800	11

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 3-16.

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

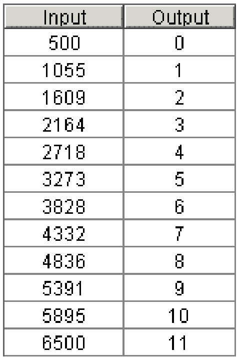
Deleting breakpoints removes them from the normalizer table. There are still table values for the deleted breakpoints: CAGE determines the positions of the deleted breakpoints by spacing them linearly by interpolation between the nondeleted breakpoints.

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



Input	Output
500	0
1055	1
1609	2
2164	3
2718	4
3273	5
3828	6
4332	7
4836	8
5391	9
5895	10
6500	11

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.

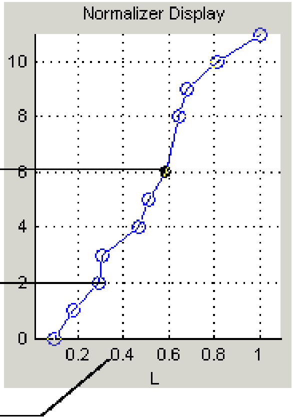
To change values of the normalizers in the **Input Output** display, double-click a cell in the **Input** column and change its value.

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.

Example of the Normalizer Display



A locked breakpoint

Breakpoint

Values of the breakpoints

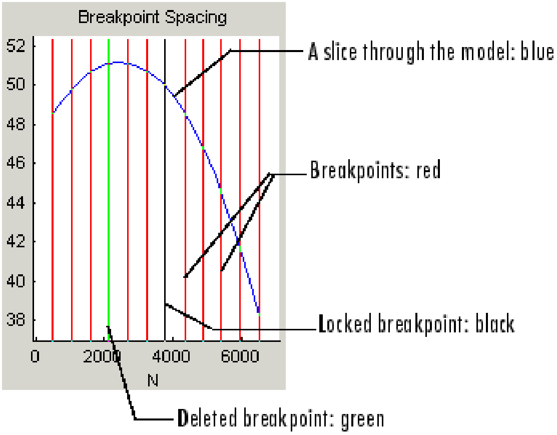
Breakpoint Spacing Display

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.

Example of the Breakpoint Spacing Display



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,

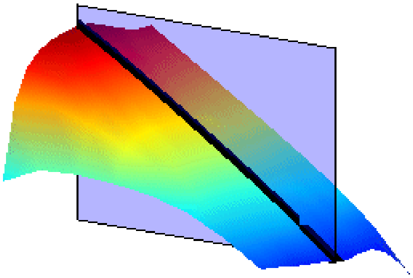
- Right-click the model in the **Breakpoint Spacing** display and select **Display > Model Curvature**..

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

Multiple Slice View

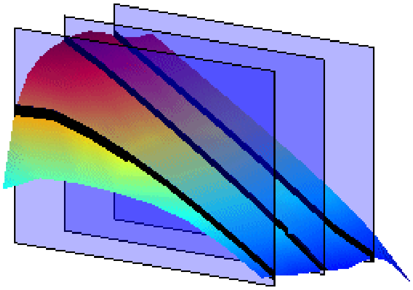
By default the **Breakpoint Spacing** display shows one slice through the model, shown.

Slice Through a Model Surface



Viewing many slices of the model gives a better impression of the curvature of the model. For example, see the following figure.

Many Slices Through a Model Surface

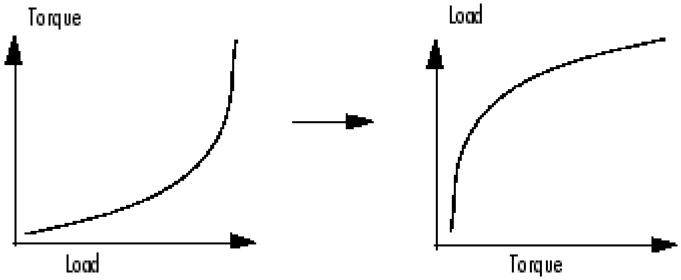


To view multiple slices through the model,

- 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

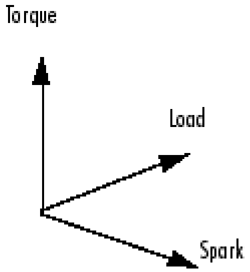
You can use CAGE to produce a table that is the inverse of another table. This involves swapping a table input with a table output, and you can invert 1-D or 2-D tables.



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 available 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.



Model

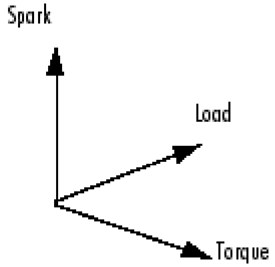
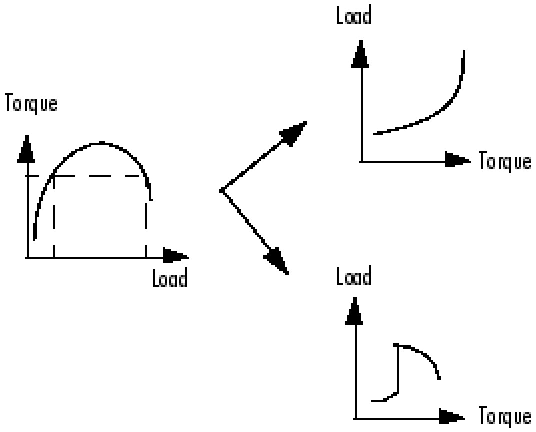


Table to fill

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 fill this table correctly from the model is a two-step process. First you need to fill a table that has the same input and output as the model, and then fill a second table by inversion.

For the inversion to be deterministic and accurate, the table to be inverted must be monotonic; that is, always increasing 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 tables: for any given output in the first 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 this problem; you can control the inversion process and determine what to do in these cases.

The following example illustrates a table 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 filling certain parts of the table; this allows you to successfully invert a multiple-valued function. See the inversion instructions for 1-D and 2-D tables in the next sections.



The process of inverting a one-dimensional 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 first table from your forward strategy, filled

   b The second table from your backward strategy, which you want to fill

2 Highlight the second table.

3 Click F 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`.

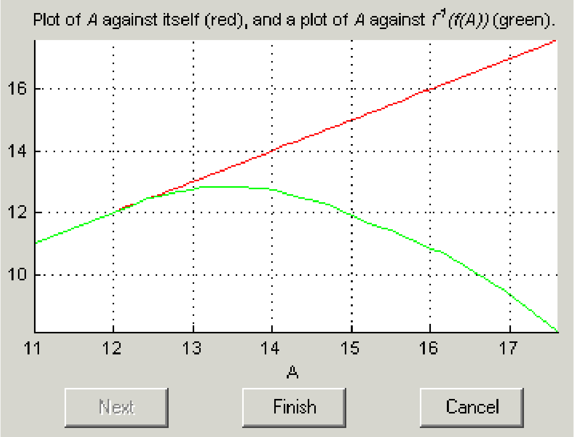
The display shows a comparison between the table (green) and the function $x = f^{-1}(f(x))$.

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.

Inverting a One-Dimensional Table



Plot of *A* against itself (red), and a plot of *A* against $f^{-1}(f(A))$ (green).

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 medium 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 first table from your forward strategy, filled

b. The second table from your backward strategy, which you want to fill

2. Highlight the second table.

3. Click F^{-1} 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 first 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

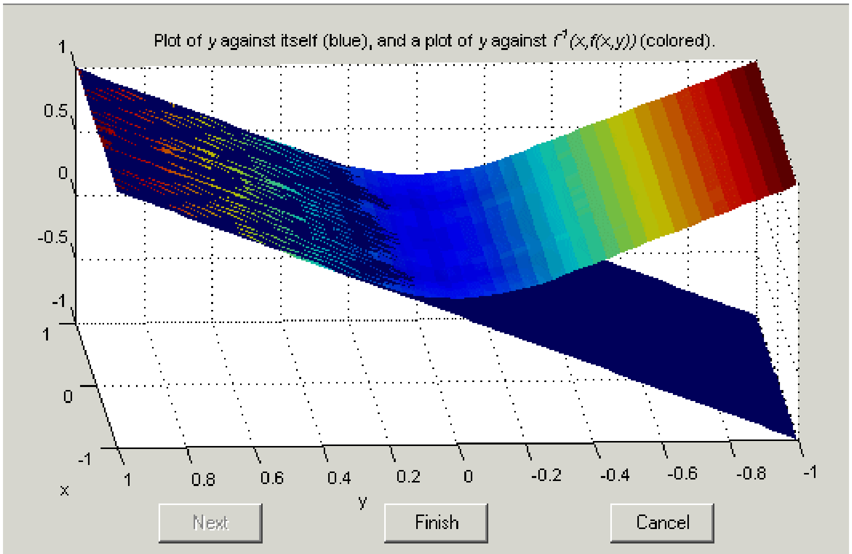
   - `Maximum` selects the uppermost range (like choosing a positive square root of a number).
   - `Minimum` selects the lower value if there are two choices (like choosing a negative square root of a number).
   - `Intermediate` selects the middle range when there are more than two choices.
   - `Automatic` 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 3-41).

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 function 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 displays 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, this leads to large errors at negative values of y, but good inversion for positive values of y.


Plot of *y* against itself (blue), and a plot of *y* against $f^{-1}(x,f(x,y))$ (colored).
1
0.5
0
-0.5
-1
1
0
-1
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
y
Next
Finish
Cancel

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

Importing and Exporting Calibrations

You can import and export calibrations in various formats.

- You can import/export the following File formats:
  - Simple CSV file
  - Simple M file
  - Simple MAT file
  - ATI Vision MAT file
  - ETAS INCA DCM file (version 1)
- Or directly to/from ATI Vision (Version 2.3.3).

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

Importing Calibrations

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

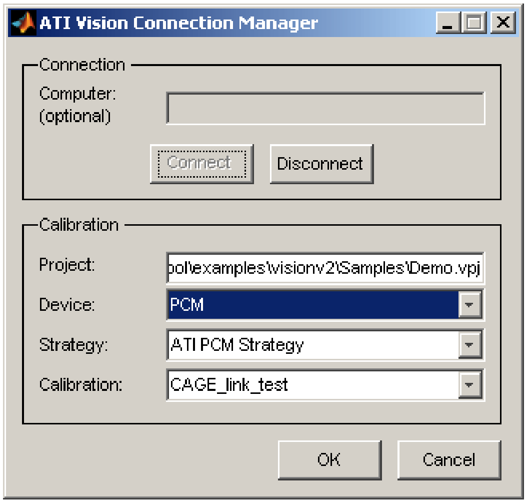
   Similarly, from the Calibration Manager, if you click Open Calibration File in the toolbar, you can select `File` or `ATI Vision` in the dialog and proceed to import in the same way.

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

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

   b. Browse to the file and click **Open** to import.

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



Computer: (optional)

Project: ol\examples\visionv2\Samples\Demo.vpj

Device: PCM

Strategy: ATI PCM Strategy

Calibration: CAGE_link_test

a The **Computer** field is optional. Leave this field 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 project loaded and with the application window invisible.

c If there is a project (`.prj` file) currently loaded in Vision it appears in the **Project** field. If this field is blank then there is no project loaded. Type a project file name to load that project. Note that the project file 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.

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 file
- INCA DCM file
- Simple CSV file
- Simple MAT file
- Simple M file

Click **OK**.

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

If you select a file format, a file browser appears. Choose a location and filename 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.)

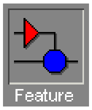
When exporting to Vision, the items in the CAGE project are matched by name with the items in the Vision calibration and the values are replaced. It is not possible to add new items to a Vision project by exporting from CAGE.

4

Feature Calibrations

This section includes the following topics:

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 2-3.)

2 Set up the model or models. (See “Setting Up Models” on page 2-11.)

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

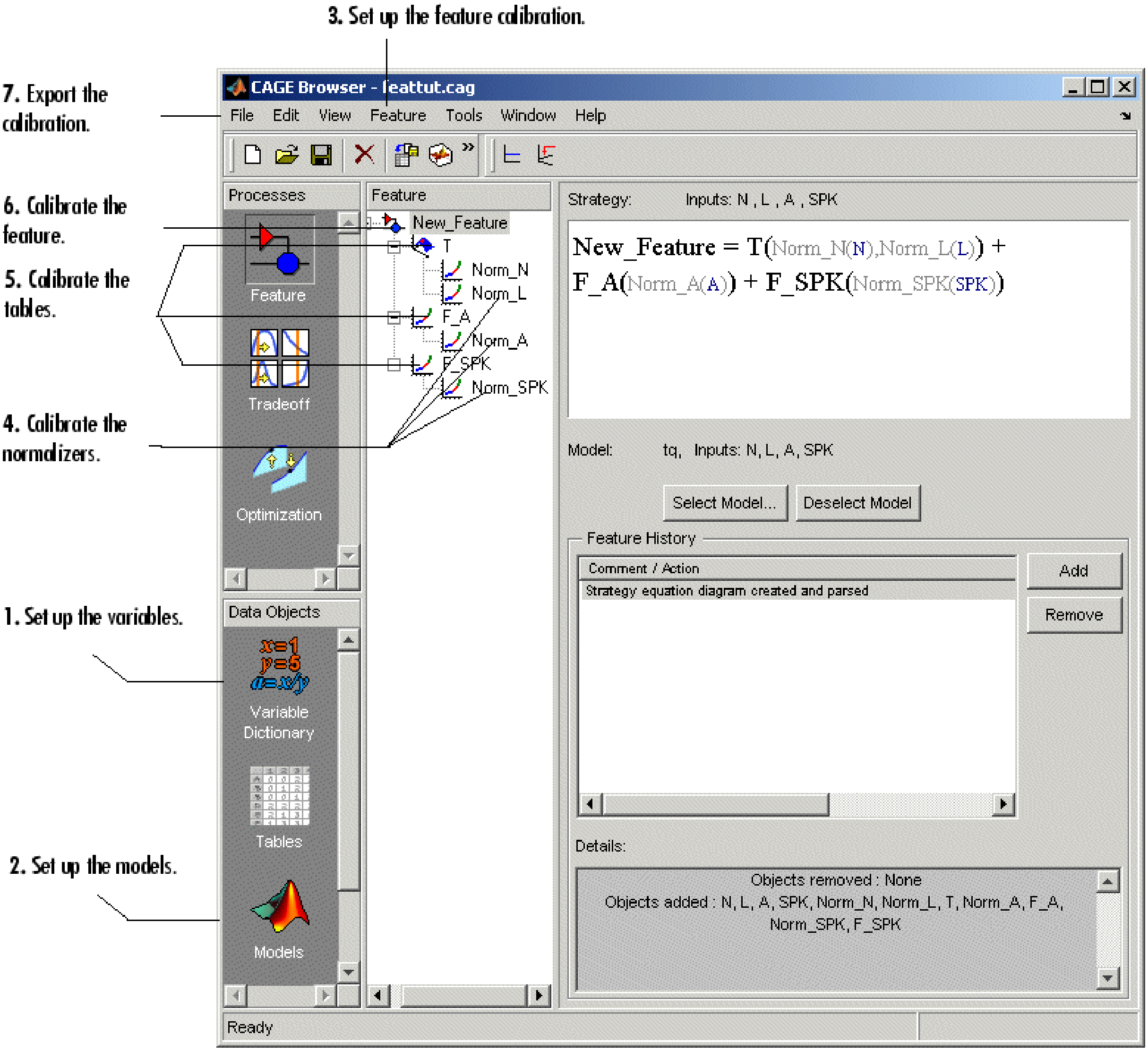
4 Calibrate the normalizers. (See “Calibrating the Normalizers” on page 4-12.)

5 Calibrate the tables. (See “Calibrating the Tables” on page 4-24.)

6 Calibrate and view the entire feature. (See “Calibrating the Feature Node” on page 4-32.)

7 Export the normalizers, tables, and features. (See “Importing and Exporting Calibrations” on page 3-46.)

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



3. Set up the feature calibration.

7. Export the calibration.

6. Calibrate the feature.

5. Calibrate the tables.

4. Calibrate the normalizers.

1. Set up the variables.

2. Set up the models.

CAGE Browser - feattut.cag

Strategy: Inputs: N , L , A , SPK

New_Feature = T(Norm_N(N),Norm_L(L)) + F_A(Norm_A(A)) + F_SPK(Norm_SPK(SPK))

Model: tq, Inputs: N, L, A, SPK

Feature History

Comment / Action
Strategy equation diagram created and parsed

Details:

Objects removed : None
Objects added : N, L, A, SPK, Norm_N, Norm_L, T, Norm_A, F_A, Norm_SPK, F_SPK

Ready

Setting Up a Feature Calibration

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

A feature refers to the object that contains the model and the collection of lookup tables. For example, a simple feature for calibrating the lookup tables for the maximum brake torque (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 4-5.
2. Assign a model. This is described in “Assigning a Model” on page 4-6.
3. Set up your strategy. This is described in “Setting Up Your Strategy” on page 4-6.
4. Set up the tables. This is described in “Setting Up Tables” on page 3-3.

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 consists of a model and a collection of lookup tables, organized in a strategy.

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

An incomplete feature is a feature that does not contain both an assigned model and a strategy. If a feature is incomplete, it is displayed as ◇ in the tree display. If a feature is complete, it is displayed as ◆ in the tree display.

Assigning a Model

Having already added a feature and a model to your session, you can assign a model to your feature.

To assign a model to your feature,

1 Highlight the 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 assign 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 4-7
- "Constructing a Strategy" on page 4-8
- "Exporting Strategies" on page 4-10

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.

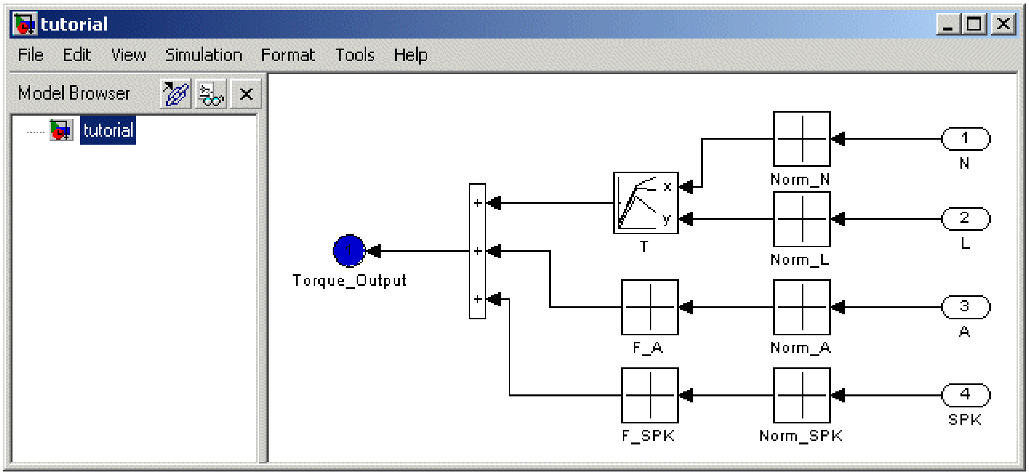
   If there is only one outport to your strategy, a dialog box opens.

   - 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 4-8.

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.

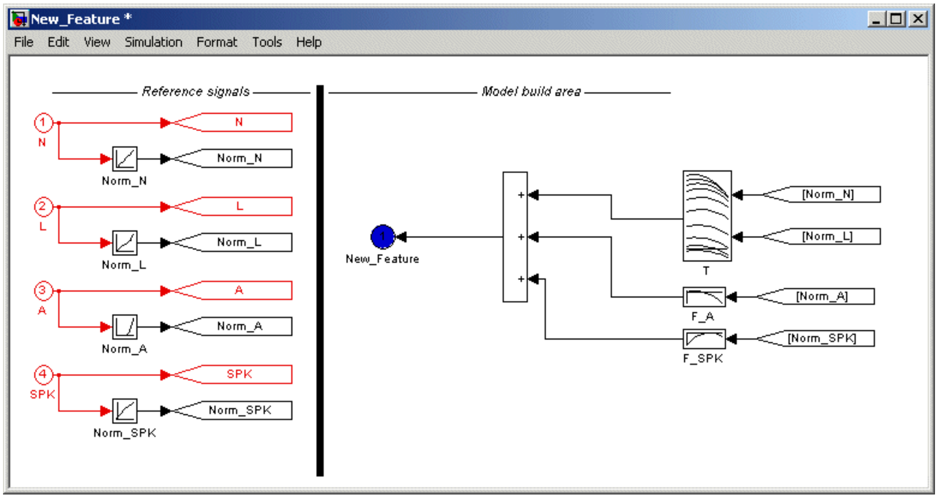
To construct a strategy using Simulink,

1 Highlight the correct feature by clicking the Feature node.

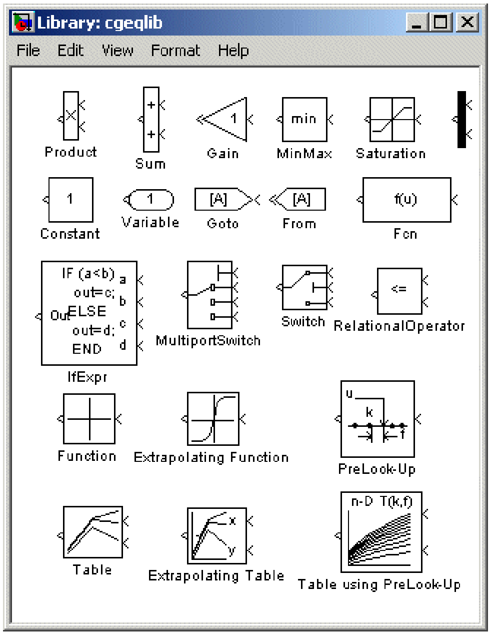
2 Select **Feature > Graphical Strategy Editor** or press **Ctrl**+E.

Three Simulink windows open:

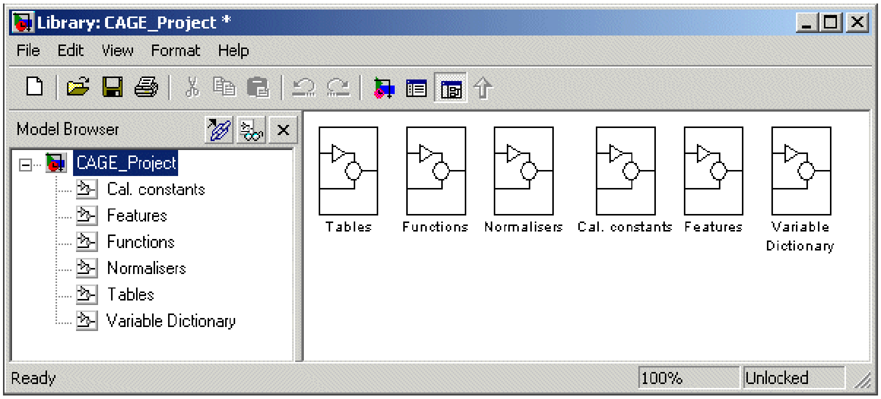
- The strategy window for editing your strategy, like the following example.



- A library window 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 strategy 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.

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

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® 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.

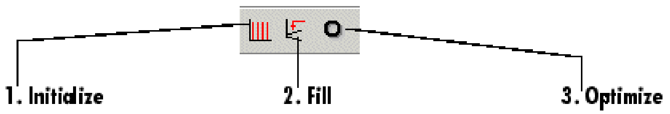
   The strategy is saved as a Simulink model (`.mdl`) file.

Calibrating the 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.

Normalizer Toolbar



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 4-13.

- **Fill**

  This spaces the breakpoints by reference to the model. For example, you can place more breakpoints where the model curvature is greatest. For more information, see “Filling Breakpoints” on page 4-13.

- **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 4-17.

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 3-32.

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

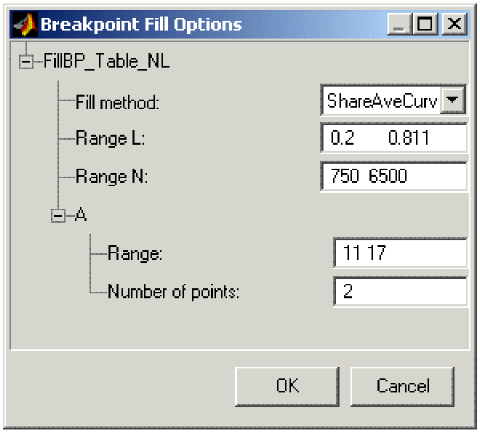
Filling breakpoints spaces the breakpoints by reference to the model. For example, one method places the majority of the breakpoints where the curvature of the model is greatest. This option is only available when you are performing Feature calibrations.

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,

1 Click or select **Normalizer > Fill**.

The Breakpoint Fill Options dialog box opens.



Breakpoint Fill Options

FillBP_Table_NL

Fill method: ShareAveCurv

Range L: 0.2 0.811

Range N: 750 6500

A

Range: 11 17

Number of points: 2

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 4-15.

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 reference 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 $MBT_{AV}(N, L)$.

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 fill methods are
  - `ReduceError`
  - `ShareAveCurv`
- For two-dimensional tables, the two fill 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 $MBT_{AV}(N, L)$,

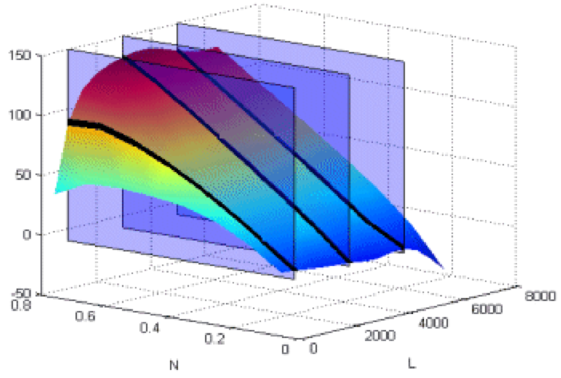
$$K = \sum_{i=1}^{\text{fine mesh}} (MBT_{AV}''(N, L))^{1/2}$$

as an approximation for

$$K = \int_{750}^{6000} |MBT_{AV}''(N, L)|^{1/2} dN$$

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.

Model Slices 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

$$\int_{750}^{N_i} \left|MBT_{AV}''(N,L)\right|^{1/2} dN = \frac{i-1}{j-1} \times K$$

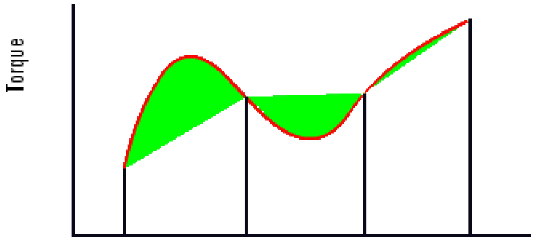
Essentially this condition spaces out the breakpoints so that an equal amount of curvature (in an appropriate metric) occurs in each breakpoint interval. The breakpoint placement is optimal in the sense that the maximum error between the lookup table estimate and the model decreases with the optimal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

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

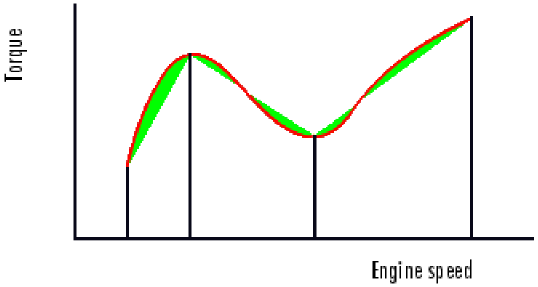
Optimizing Breakpoints

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

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



The green shaded areas show the error between the interpolated table values and the model using the initial breakpoints.



Optimizing the position of the breakpoints can greatly reduce the error between the interpolated table values and the model.

To see the difference between optimizing breakpoints and optimizing table values, compare with the illustration in “Optimizing Table Values” on page 4-26.

See “Filling Methods” on page 4-15 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 (maximum 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,

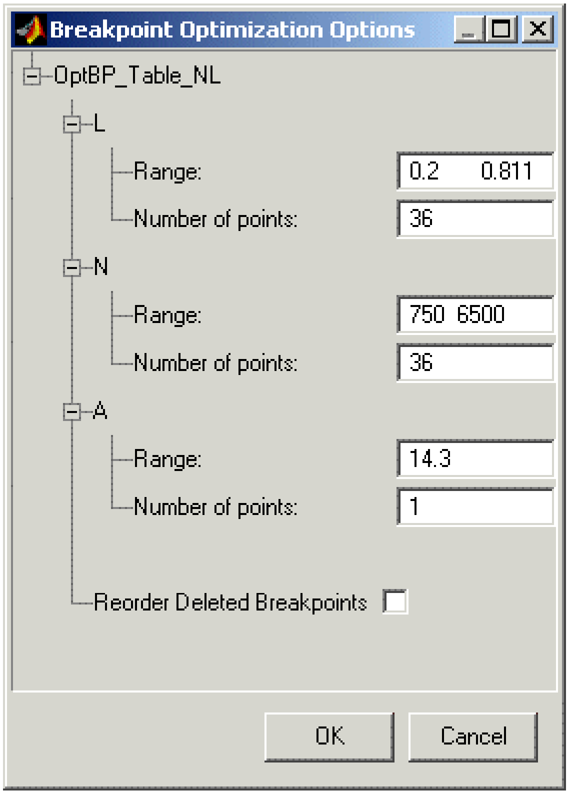
1 Ensure that the optimization routine works over reasonable values for the table by choosing one of these methods:

   a Select **Normalizer > Initialize**.

   b Select **Normalizer > Fill** .

2 Click O on the toolbar or select **Normalizer > Optimize**.

   This opens the following dialog box.



Breakpoint Optimization Options

OptBP_Table_NL

L

Range: 0.2 0.811

Number of points: 36

N

Range: 750 6500

Number of points: 36

A

Range: 14.3

Number of points: 1

Reorder Deleted Breakpoints □

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 3-33.

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:

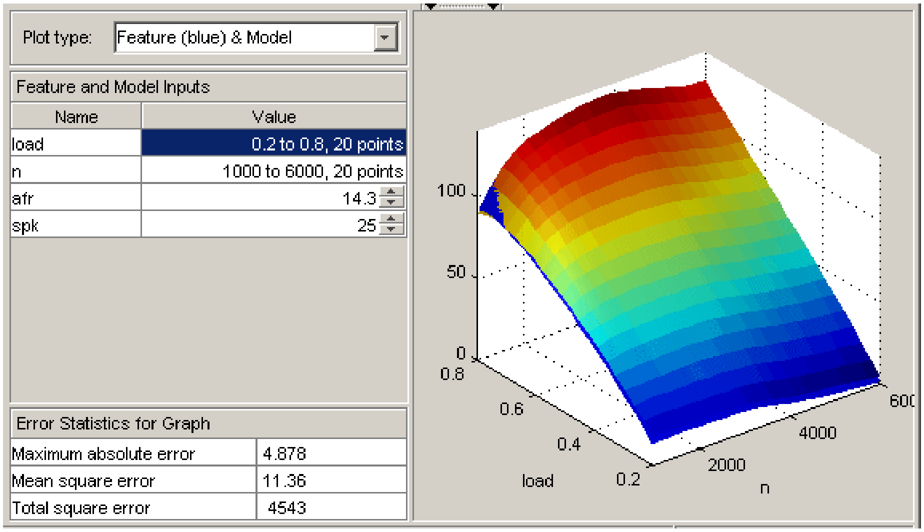
- The default range for the normalizer variable is the range of the variable.
- The default value for all other model variables is the set point of the variable.
- The default number of grid points is three times the number of breakpoints.

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.



Plot type: Feature (blue) & Model

Feature and Model Inputs

Name	Value
load	0.2 to 0.8, 20 points
n	1000 to 6000, 20 points
afr	14.3
spk	25

Error Statistics for Graph

Maximum absolute error	4.878
Mean square error	11.36
Total square error	4543

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 4-28.

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 2-6.

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

4 Check the comparison between the table and the model.

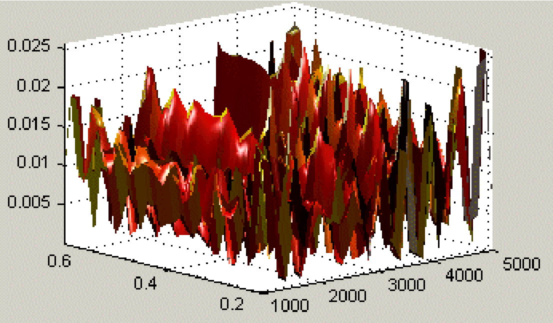
Right-click the comparison graph to view the error display.

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’ (grid filled using these breakpoints).

Error Display in the Comparison Pane



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 (TableModel)`. 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 4-28

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

Calibrating the Tables

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

Highlight a table in the tree display to see the Table view. For more information about the Table view, see “Table View” on page 3-6.

In CAGE, a table is defined to be either a one-dimensional or a two-dimensional lookup table. One-dimensional tables are sometimes known as characteristic lines or functions. Two-dimensional tables are also known as characteristic maps or tables.

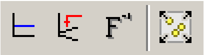
Each lookup table has either one or two axes associated with it. These axes are normalizers. See “About Normalizers” on page 3-30 for more information.

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 with AFR as its normalizer input

Before you can calibrate your tables, you must calibrate your normalizers. For information, see “Calibrating the Normalizers” on page 4-12.

This section describes how you can use CAGE to fill 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 4-25.

- **Fill Table**

  Fills and optimizes the table values by reference to the model. For information, see “Filling Table Values” on page 4-26.

- **Fill by Inversion**

  Fills the table by creating an inversion of another table. For information, see “Inverting a Table” on page 3-39.

- **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 4-30.

The next sections describe each of these toolbar options in detail. See the “Table Menu” on page 3-12 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 3-4) 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**.

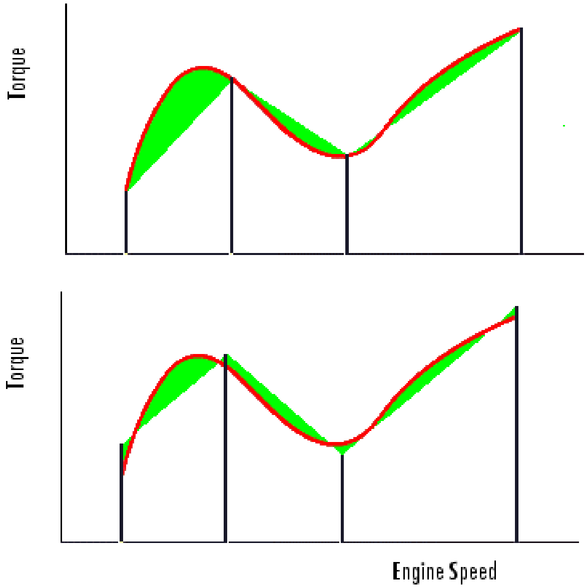
  This opens the Feature Fill Wizard. You can fill multiple tables at once using the wizard, and you can **Fill** from the top feature node or from any table node in a feature. See "Feature Fill Wizard" on page 4-34 for instructions.

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.



This shows the error when filling the table using breakpoints.

This shows the reduced error after optimizing table values using input values between the breakpoints.

To see the difference between optimizing table values and optimizing the positions of breakpoints, compare with the illustration in “Optimizing Breakpoints” on page 4-17.

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.

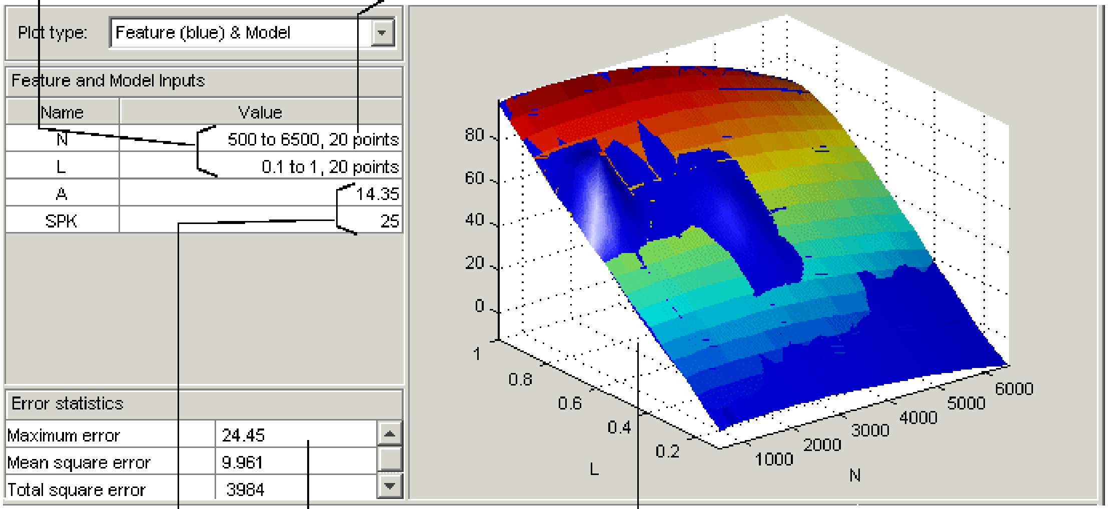
See Also.

- Reference page for `lsqnonlin`
- “Calibrating the Tables” on page 4-24

Comparing the Strategy and the Model

When you calibrate a strategy, or collection of tables, by reference to a model, it is useful to compare the strategy and the model. The comparison pane provides a graphical tool for investigating this, as shown in the following example.

Note This is a comparison between the current strategy values and the model, unlike the comparison pane from the normalizer node, which compares the model and a full factorial grid filled using the breakpoints.



Plot type: Feature (blue) & Model

Feature and Model Inputs

Name	Value
N	500 to 6500, 20 points
L	0.1 to 1, 20 points
A	14.35
SPK	25

Error statistics

Maximum error	24.45
Mean square error	9.961
Total square error	3984

The ranges of the common variables

Number of points in the comparison display

Variables in the model, not in the table

Error between the strategy and the model

Comparison of the strategy and the model

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.

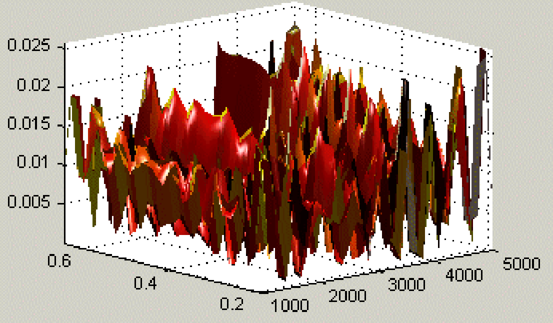
2. Check the values selected for any variables in the model that are not in the selected table. The default for this is the set point of the variable's range. Double-click to edit.

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 error, 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 erroras 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 3-47.

Filling the Table by Extrapolation

Filling a table by extrapolation fills 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 automatically after filling cells in the mask in the “Feature Fill Wizard” on page 4-34.

To fill 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 more 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 defines 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 define an extrapolation mask to include all the cells in these ranges. You can then fill 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 threshold now form the extrapolation mask, and thus are colored yellow.

Calibrating the Feature Node

Selecting a Feature node displays the Feature view. For more information about the Feature view, see “Feature View” on page 4-40.

The Feature view enables you to calibrate the entire feature, that is, fill 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 these sections:

1 “Initializing the Feature” on page 4-32

2 “Feature Fill Wizard” on page 4-34

Initializing the Feature

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

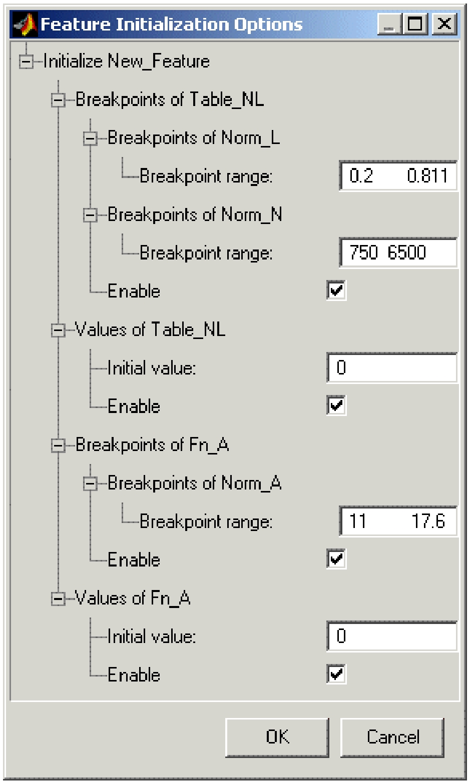
- A speed (N), load (L) table
- A table to account for the behavior of air/fuel ratio (A)

Initializing this feature sets the values of the normalizers for speed, load, and AFR over the range of each variable and put specified 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,

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



- Initialize New_Feature
  - Breakpoints of Table_NL
    - Breakpoints of Norm_L
      - Breakpoint range: 0.2 0.811
    - Breakpoints of Norm_N
      - Breakpoint range: 750 6500
    - Enable ☑
  - Values of Table_NL
    - Initial value: 0
    - Enable ☑
  - Breakpoints of Fn_A
    - Breakpoints of Norm_A
      - Breakpoint range: 11 17.6
    - Enable ☑
  - Values of Fn_A
    - Initial value: 0
    - Enable ☑

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

- L has range `0.2 0.811`.
- N has range `750 6500`.
- A has range `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

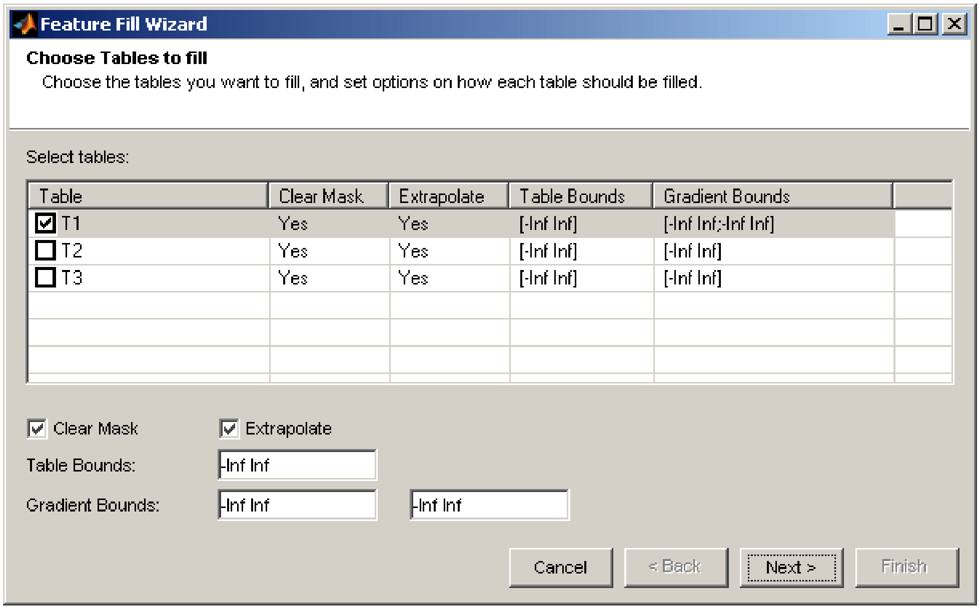
Use the Feature Fill Wizard to fill and optimize the values in tables by reference to the model. You can fill multiple tables at once using the wizard, and you can **Fill** from the top feature node or from any table node in a feature.

Note you could also optimize the breakpoints for the normalizers before using the Feature Fill Wizard. (See "Filling Breakpoints" on page 4-13 and "Optimizing Breakpoints" on page 4-17.)

This section describes how to use the Feature Fill Wizard. For a detailed description about the filling processes, see "Filling Table Values" on page 4-26.

To fill feature tables,

1 Click . This opens the Feature Fill Wizard.



Choose Tables to fill

Choose the tables you want to fill, and set options on how each table should be filled.

Select tables:

Table	Clear Mask	Extrapolate	Table Bounds	Gradient Bounds
☑ T1	Yes	Yes	[-Inf Inf]	[-Inf Inf;-Inf Inf]
□ T2	Yes	Yes	[-Inf Inf]	[-Inf Inf]
□ T3	Yes	Yes	[-Inf Inf]	[-Inf Inf]

☑ Clear Mask ☑ Extrapolate

Table Bounds: -Inf Inf

Gradient Bounds: -Inf Inf -Inf Inf

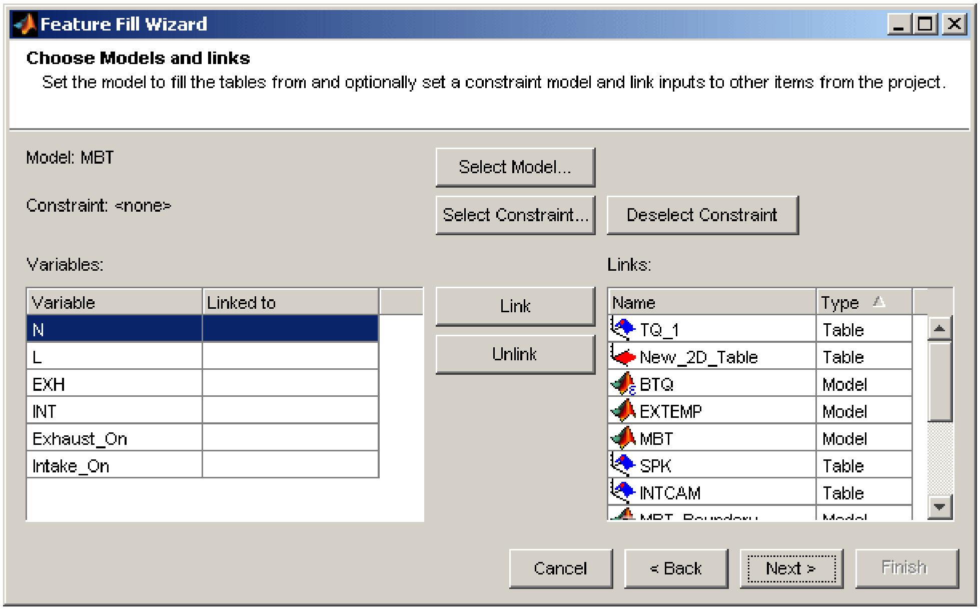
Screen 1: Select tables to fill.

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

- **Clear Mask** — select this check box to clear any table mask and fill all unlocked table cells (locked cells are never altered). Clear this check box to fill unlocked cells in the current extrapolation mask only, or all unlocked cells if there is no mask.
- **Extrapolate** — select this to extrapolate across the whole table after filling cells. The extrapolation is based on the filled 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 (increasing) between successive rows.

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

2 Choose models and links.



Choose Models and links

Set the model to fill the tables from and optionally set a constraint model and link inputs to other items from the project.

Model: MBT

Constraint: <none>

Variables:

Variable	Linked to
N	
L	
EXH	
INT	
Exhaust_On	
Intake_On	

Links:

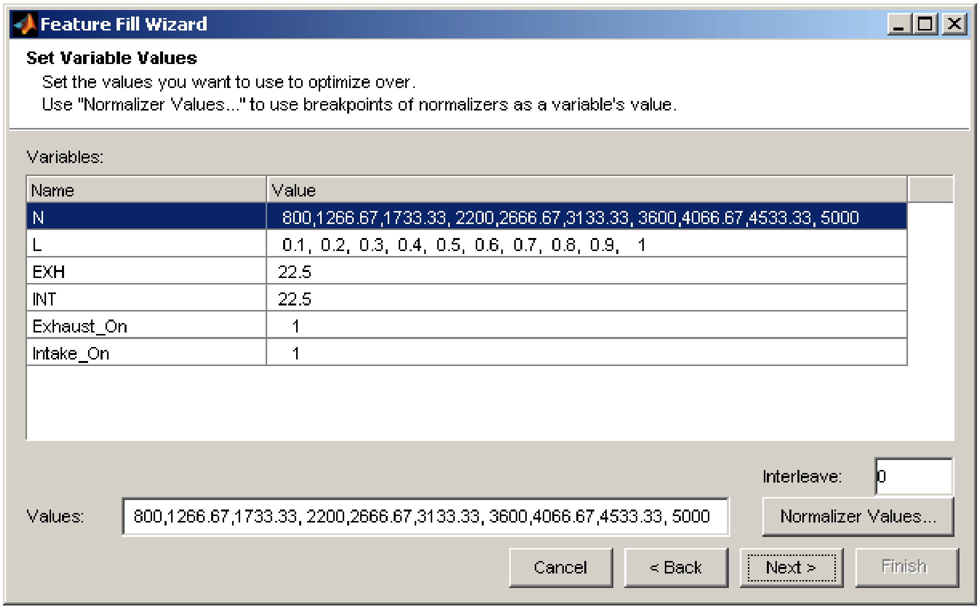
Name	Type
TQ_1	Table
New_2D_Table	Table
BTQ	Model
EXTEMP	Model
MBT	Model
SPK	Table
INTCAM	Table
MBT_Boundary	Model

- Click **Select Model** to choose a model to fill the tables from. The feature filler 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 model to use in the filling process. The feature filler limits its activity to within this constraint model, for example, the boundary constraint of a model. While boundary models are typically used as the constraint in this setting, any function model that returns a logical output (true for valid, false for invalid) may also be used.
- 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 feature filler compares the feature and model at the table breakpoints. It is also possible to compare the feature and model on a finer grid by choosing a positive value of **Interleave**. This 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 4-26). An **Interleave** value of 1 inserts one grid point between each pair of breakpoints, and so on.

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 the non-normalizer variables or you can interleave values between breakpoints. Increasing the number of grid points increases the quality of the approximation and minimizes interpolation error, but also increases the computation time.



Feature Fill Wizard

Set Variable Values

Set the values you want to use to optimize over.
Use "Normalizer Values..." to use breakpoints of normalizers as a variable's value.

Variables:

Name	Value
N	800,1266.67,1733.33, 2200,2666.67,3133.33, 3600,4066.67,4533.33, 5000
L	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
EXH	22.5
INT	22.5
Exhaust_On	1
Intake_On	1

Interleave: 0

Values: 800,1266.67,1733.33, 2200,2666.67,3133.33, 3600,4066.67,4533.33, 5000

- Click **Normalizer Values** to select normalizers to use those breakpoints as a variable's value.
- Enter a value in the **Interleave** edit box to add values between breakpoints, then click **Normalizer Values** to select a normalizer.

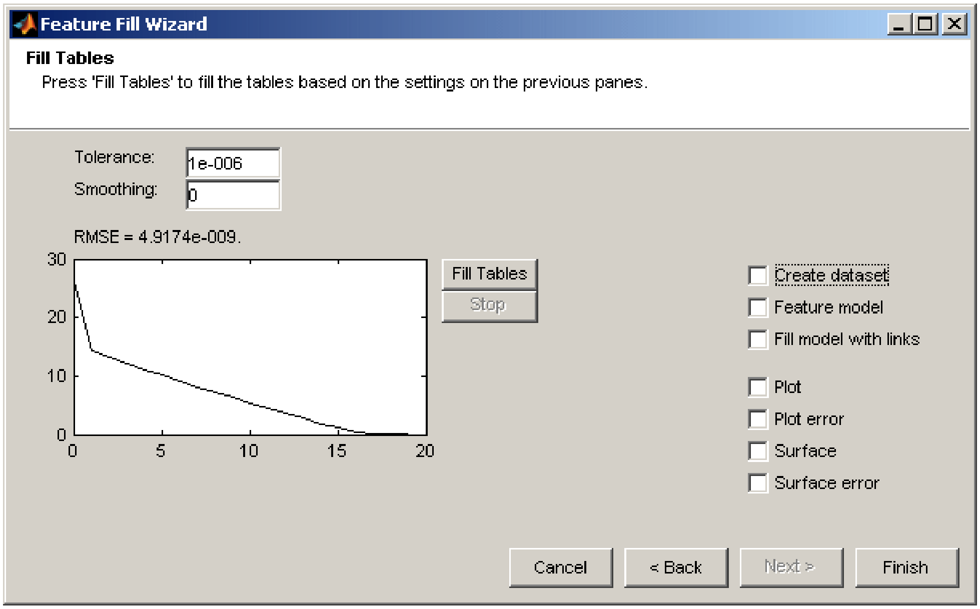
- Edit set point values in the **Values** edit box to optimize over a range rather than at a single point. If you choose a range of values the table will be filled using the average model value at each cell. For example, if you enter `-5:5:50` for the variable spark, the optimization of table values 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 fill the tables.

CAGE evaluates the model over the number of grid points specified, 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.



Feature Fill Wizard

Fill Tables

Press 'Fill Tables' to fill the tables based on the settings on the previous panes.

Tolerance: 1e-006

Smoothing: 0

RMSE = 4.9174e-009.

- 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 flat plane.

- Select the 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.
- Select the check box to create a dataset containing the output values at each specified grid point.

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 3-46.

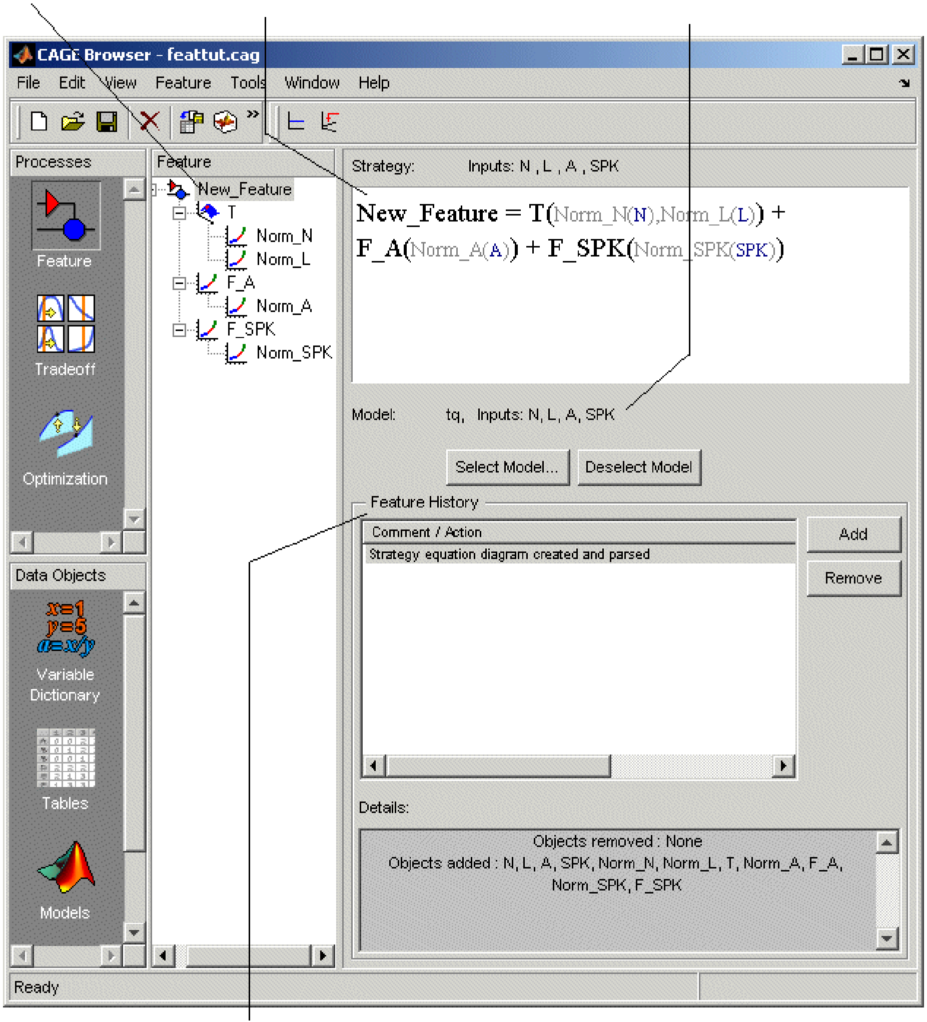
Feature View

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

Selected feature

1. The strategy for the selected feature

2. The model associated with the selected feature



Strategy: Inputs: N , L , A , SPK

New_Feature = T(Norm_N(N),Norm_L(L)) + F_A(Norm_A(A)) + F_SPK(Norm_SPK(SPK))

Model: tq, Inputs: N, L, A, SPK

Comment / Action
Strategy equation diagram created and parsed

Details:

Objects removed : None
Objects added : N, L, A, SPK, Norm_N, Norm_L, T, Norm_A, F_A, Norm_SPK, F_SPK

3. Feature History pane

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 4-6.

- **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 4-6.

- **Clear Strategy**

  Clears the current strategy from your feature.

- **Initialize**

  Initializes the feature; also in the toolbar. See "Initializing the Feature" on page 4-32 for details.

- **Fill**

Fills and optimizes the feature; also in the toolbar. See “Feature Fill Wizard” on page 4-34 for details.

Tradeoff Calibrations

This section includes the following topics:

Performing a Tradeoff Calibration

Tradeoff

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 2-3.)
2. Set up the model or models. (See “Setting Up Models” on page 2-11.)
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 5-10.)
5. Export the normalizers, tables, and tradeoffs. (See “Exporting Calibrations” on page 3-47.)

You can also use regions to enhance your calibration. (See “Using Regions” on page 5-21.)

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 6-46 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.

3. Set up the tradeoff calibration.

5. Export the calibration.

4. Calibrate the tables.

1. Set up the variables.

2. Set up the models.

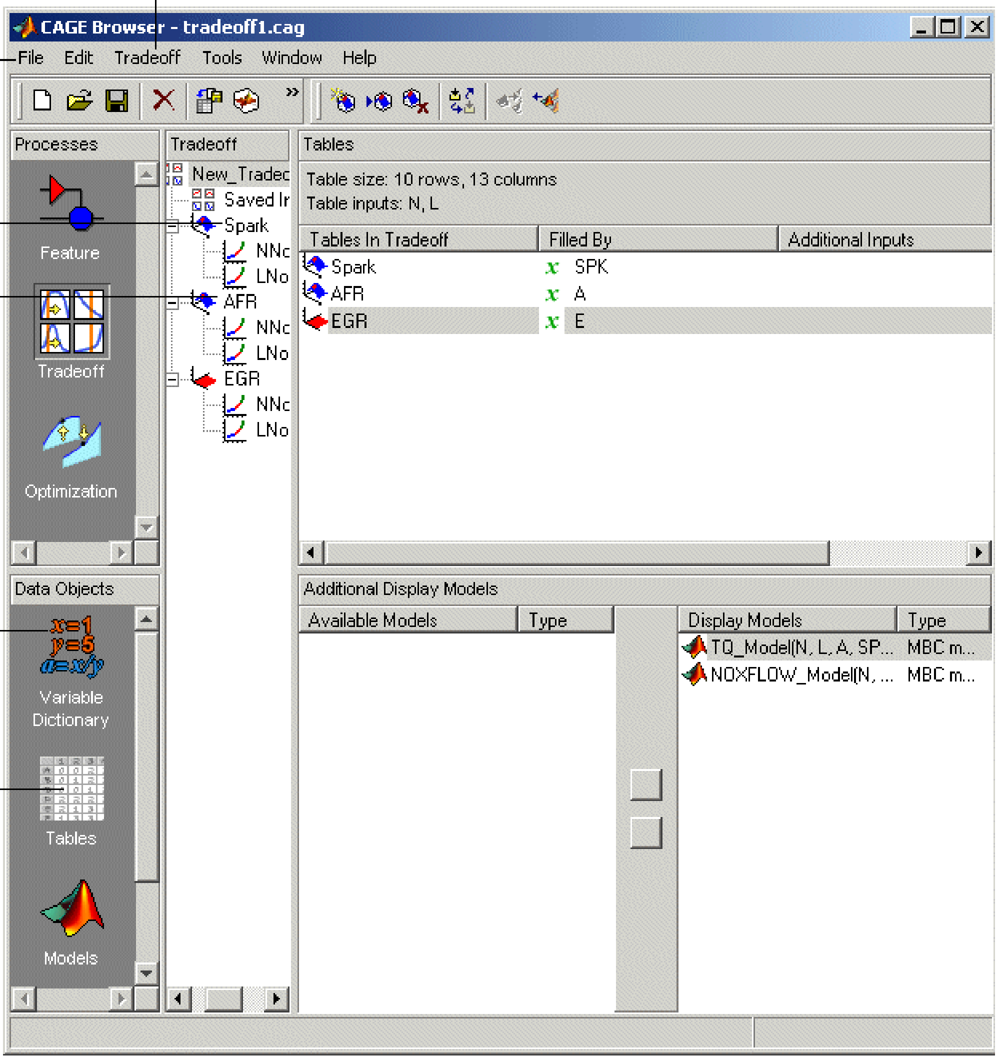


Table size: 10 rows, 13 columns
Table inputs: N, L

Tables In Tradeoff	Filled By	Additional Inputs
Spark	x SPK	
AFR	x A	
EGR	x E	

Display Models	Type
TQ_Model(N, L, A, SP...	MBC m...
NOXFLOW_Model(N, ...	MBC m...

Setting Up a Tradeoff Calibration

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

Typically there are many different and conflicting objectives. For example, a calibrator might want to maximize torque while restricting 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 5-5.

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 described in "Displaying Models in Tradeoff" on page 5-8.

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

When you finish 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 Tradeoff 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 5-8.

Note that you must select the top tradeoff 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 fill the table and two more variables to define each of the two normalizers.

A dialog box opens.

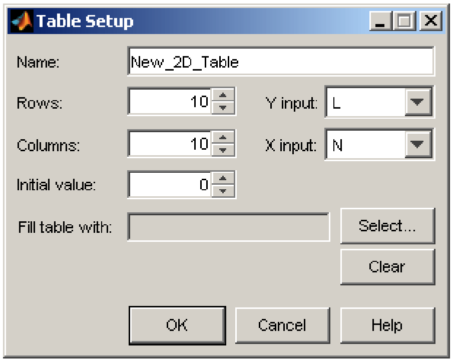


Table Setup

Name: New_2D_Table

Rows: 10 Y input: L

Columns: 10 X input: N

Initial value: 0

Fill table with:

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.

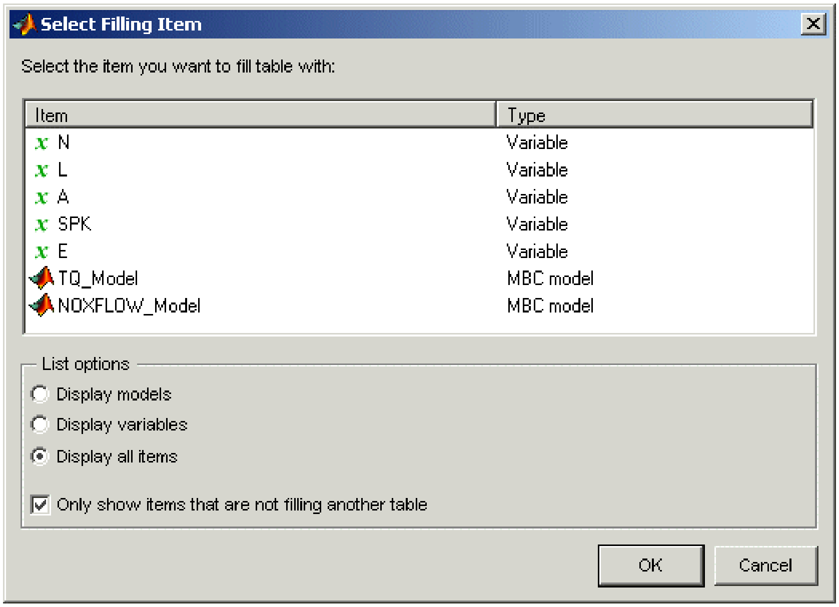
For the first table in a tradeoff, you must set the normalizer inputs and sizes:

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 filling item for a table. A dialog opens where you can select from the models and variables in your session.



Select Filling Item

Select the item you want to fill table with:

Item	Type
x N	Variable
x L	Variable
x A	Variable
x SPK	Variable
x E	Variable
TQ_Model	MBC model
NOXFLOW_Model	MBC model

List options

○ Display models

○ Display variables

● Display all items

☑ Only show items that are not filling another table

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 filling another table by clearing the check box.

b Select the filling 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 fill 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,

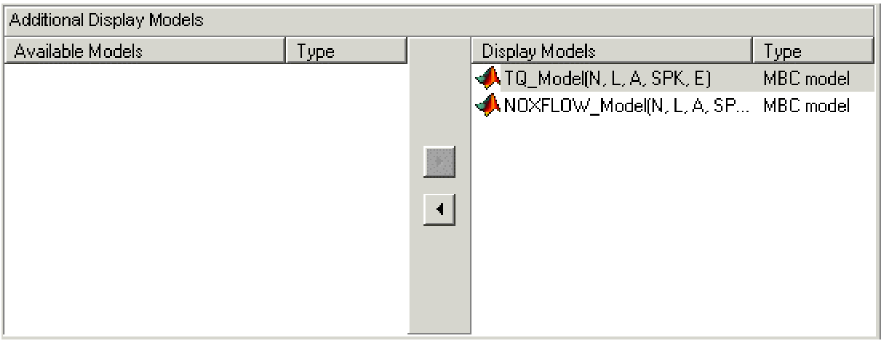
1 Highlight the tradeoff node in the tree.

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.



Additional Display Models

Available Models	Type

Display Models	Type
TQ_Model(N, L, A, SPK, E)	MBC model
NOXFLOW_Model(N, L, A, SP...	MBC model

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

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 5-13.

4. Determine the value of the desired operating point.

   For instructions, see “Determining a Value at a Specific Operating Point” on page 5-15.

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 4-30.

After you complete all these steps you can export your calibration. For information, see “Exporting Calibrations” on page 3-47.

Table View in a Tradeoff Calibration

1. Select the table.

2. Select the operating point in the table that you want to calibrate.

5. Repeat this process over a number of operating points in the table, then fill the table by extrapolation.

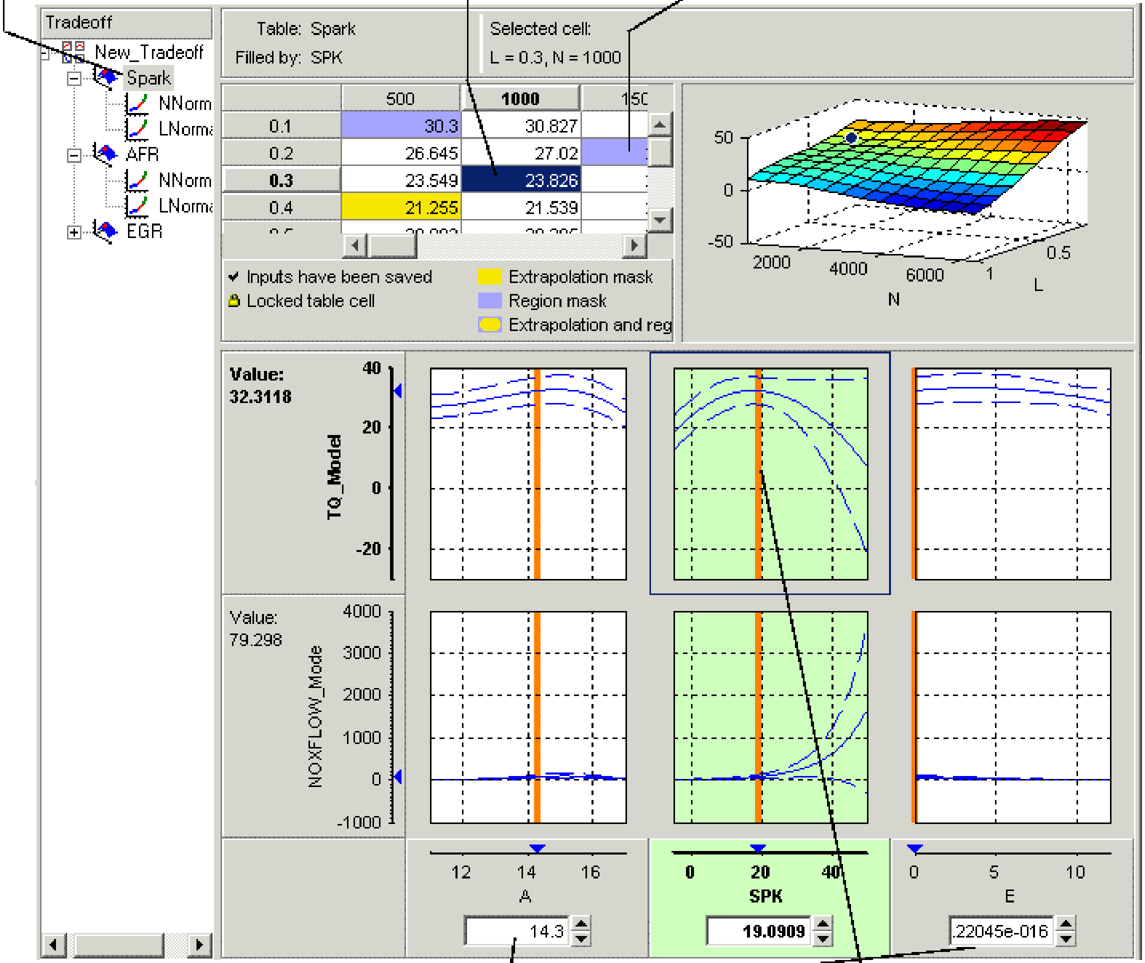


Table: Spark

Filled by: SPK

Selected cell:
L = 0.3, N = 1000

	500	1000	150
0.1	30.3	30.827	
0.2	26.645	27.02	
0.3	23.549	23.826	
0.4	21.255	21.539	

✓ Inputs have been saved

🔒 Locked table cell

🟨 Extrapolation mask

🟪 Region mask

🟨 Extrapolation and reg

Value:
32.3118

Value:
79.298

A: 14.3

SPK: 19.0909

E: 22045e-016

3. Set the values for other input variables.

4. Determine a suitable value for the point.

Notice that the graphs colored green indicate how the highlighted table will be filled:

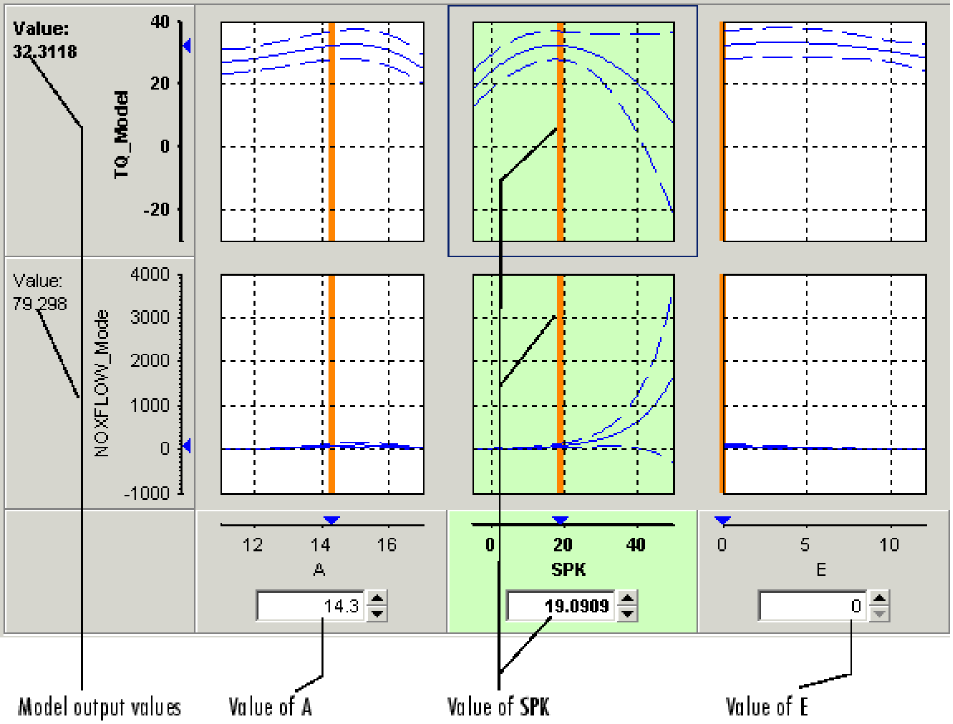
- If a row of graphs is highlighted, the table is being filled by the indicated model evaluation (the value shown at the left of the row).
- If the column of graphs is green, the table is being filled by the indicated input variable (shown in the edit box below the column).

The next sections describe the following in detail:

- "Setting Values of Other Variables" on page 5-13
- "Determining a Value at a Specific Operating Point" on page 5-15

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 tradeoff to calibrate a table for spark angle, you might want to set the initial values for 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.

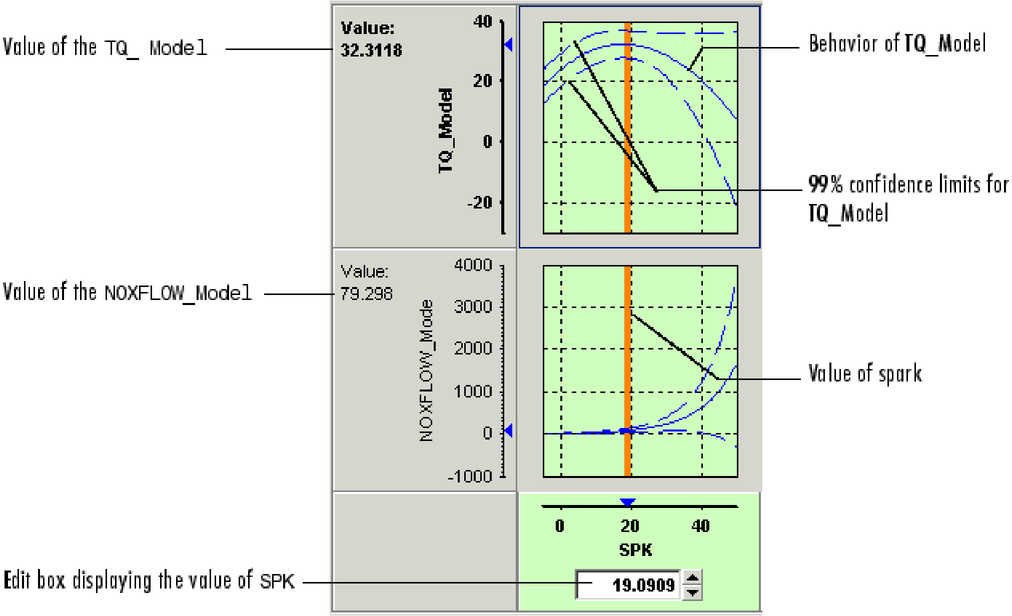
4 Right-click and select **Extrapolation Mask > Add Selection**.

This adds the cell to the extrapolation mask.

5 Click to extrapolate over the entire table.

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

Determining a Value at a Specific Operating Point



Value of the TQ_ Model

Value of the NOXFLOW_Model

Edit box displaying the value of SPK

Behavior of TQ_Model

99% confidence limits for TQ_Model

Value of spark

Value: 32.3118

Value: 79.298

19.0909

Performing a tradeoff calibration necessarily involves 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 peak torque also gives an acceptable value for the NOX flow 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 3-16.

- **Configure Hidden Items**

  This opens a dialog box that allows you to show or hide models and input variables. Select or clear the check boxes 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 Confidence Intervals**

  When you select this, the graphs display the 99% confidence 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 displays 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 once 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 filling 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 values 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 filled 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** fills 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 filling tables by extrapolation. Note that cells filled by applying values from the tradeoff graphs (using the **Apply Fill To Table** toolbar and menu option) are automatically 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 defines locally where to extrapolate before globally extrapolating over the entire table. Use regions to define high-priority areas for use when filling tables by extrapolation. See “Using Regions” on page 5-21.
  - **Add Selection**
  - **Remove Selection**
  - **Clear Regions**
- **Extrapolate** — This option (also in the toolbar) fills the table by extrapolation using regions (to define locally where to extrapolate before globally extrapolating) and the cells defined in the extrapolation mask.
- **Extrapolate (Ignore Regions)** — This option fills 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 6-46 for information.

Tools Menu

- **Calibration Manager** — opens the Calibration Manager. See "Calibration Manager" on page 3-20.
- **Surface Viewer** — Opens the Surface Viewer. See Chapter 8, "Surface Viewer".

Using Regions

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

For example, consider filling a large table that has twenty breakpoints for each normalizer by extrapolation. 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 operating points takes a long time.

To overcome this problem, you can

1 Define regions within the lookup table.

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

3 Click to fill the table by extrapolation.

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

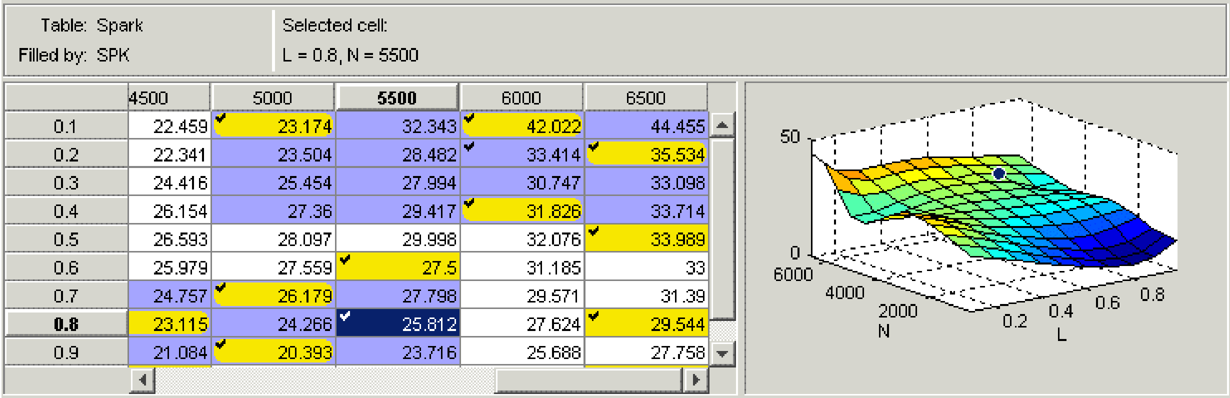


Table: Spark
Filled by: SPK

Selected cell:
L = 0.8, N = 5500

	4500	5000	5500	6000	6500
0.1	22.459	✓ 23.174	32.343	✓ 42.022	44.455
0.2	22.341	23.504	28.482	✓ 33.414	✓ 35.534
0.3	24.416	25.454	27.994	30.747	33.098
0.4	26.154	27.36	29.417	✓ 31.826	33.714
0.5	26.593	28.097	29.998	32.076	✓ 33.989
0.6	25.979	27.559	✓ 27.5	31.185	33
0.7	24.757	✓ 26.179	27.798	29.571	31.39
0.8	23.115	24.266	✓ 25.812	27.624	✓ 29.544
0.9	21.084	✓ 20.393	23.716	25.688	27.758

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 if 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 define the region, click in the toolbar, or right-click and select **Extrapolation Regions > Add Selection**, or select the menu option **Tables > Extrapolation Regions > Add Selection**.

The cells in the region are colored blue.

Clearing a Region

1. Highlight the rectangle of cells in your table.
2. To clear the region, click in the toolbar, or right-click and select **Extrapolation Regions > Remove Selection**, or select the menu option **Tables > Extrapolation Regions > Remove Selection**.

   You can clear all regions at once by selecting **Clear Regions** from the **Extrapolation Regions** submenu.

Multimodel Tradeoffs

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 multimodel tradeoff).

A multimodel tradeoff is a specially 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 multimodel tradeoff from the test plan node. The models must be two-stage and must have exactly two global inputs.

The procedure for calibrating by using a multimodel tradeoff follows:

1. Add the multimodel tradeoff. (See the following section, “Adding a Multimodel Tradeoff” on page 5-24.)

2. Calibrate the tables. (See “Calibrating Using a Multimodel Tradeoff” on page 5-27.)

3. Export your calibration. (See “Importing and Exporting Calibrations” on page 3-46.)

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. 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 multimodel is not defined for all other cells in the table and so you cannot use models to tradeoff. You can edit these cells and they can be filled 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 Specific Operating Point” on page 5-15. When you have determined table values at each of the model operating points, you can fill the whole table by extrapolation by clicking . See “Filling the Table by Extrapolation” on page 4-30.

Adding a Multimodel Tradeoff

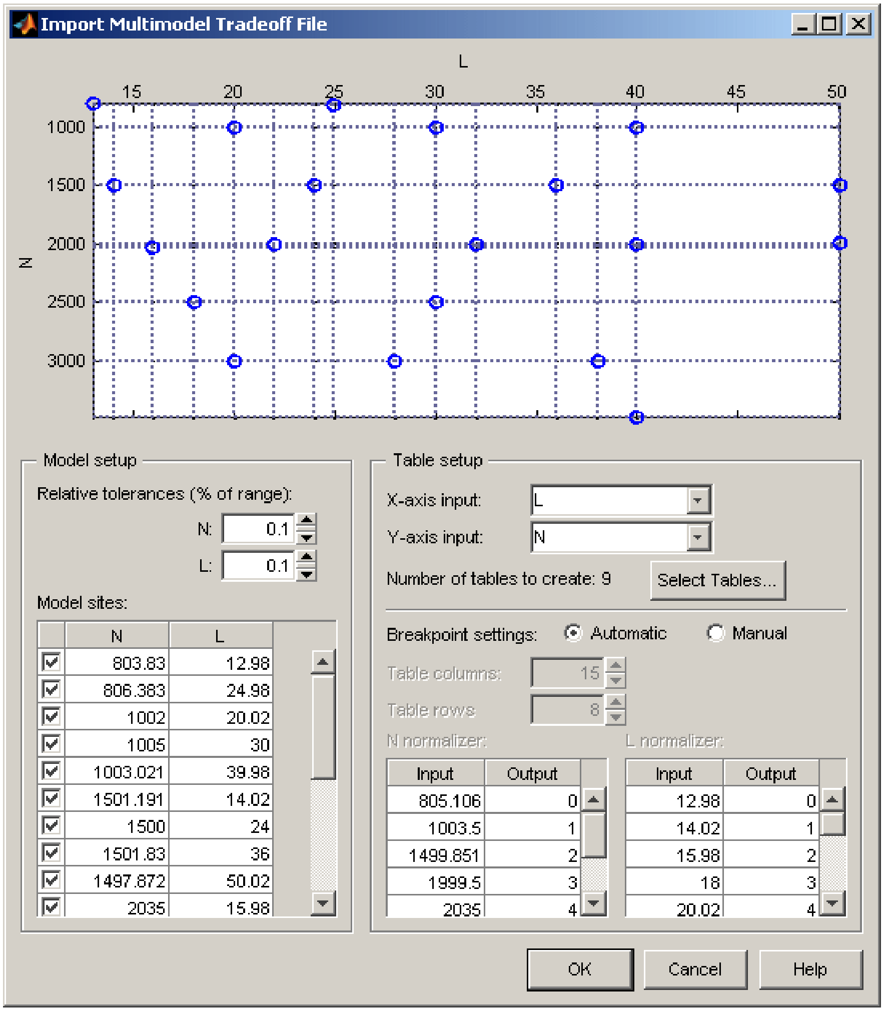
To add a multimodel tradeoff to your session,

1 Select **File > New > Tradeoff**. CAGE switches to the tradeoff view and creates a new empty tradeoff.

2 Select the new tradeoff in the tree, then select **File > Import > Multimodel Tradeoff**.

   The file must have been exported from the MBC Model Browser using the **Tradeoff** button (only enabled for two-stage models with exactly two global inputs). See “Multimodel Tradeoffs” on page 5-23.

3 Select the correct file to import and click **Open**. This opens a dialog box.



Model setup

Relative tolerances (% of range):

N: 0.1

L: 0.1

Model sites:

	N	L
☑	803.83	12.98
☑	806.383	24.98
☑	1002	20.02
☑	1005	30
☑	1003.021	39.98
☑	1501.191	14.02
☑	1500	24
☑	1501.83	36
☑	1497.872	50.02
☑	2035	15.98

Table setup

X-axis input: L

Y-axis input: N

Number of tables to create: 9 Select Tables...

Breakpoint settings: ● Automatic ○ Manual

Table columns: 15

Table rows: 8

N normalizer:

Input	Output
805.106	0
1003.5	1
1499.851	2
1999.5	3
2035	4

L normalizer:

Input	Output
12.98	0
14.02	1
15.98	2
18	3
20.02	4

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 Multimodel Tradeoff” on page 5-27.

Calibrating Using a Multimodel Tradeoff

Each editable operating point in your tables has a model icon in the cell, like this example cell.

4602.722

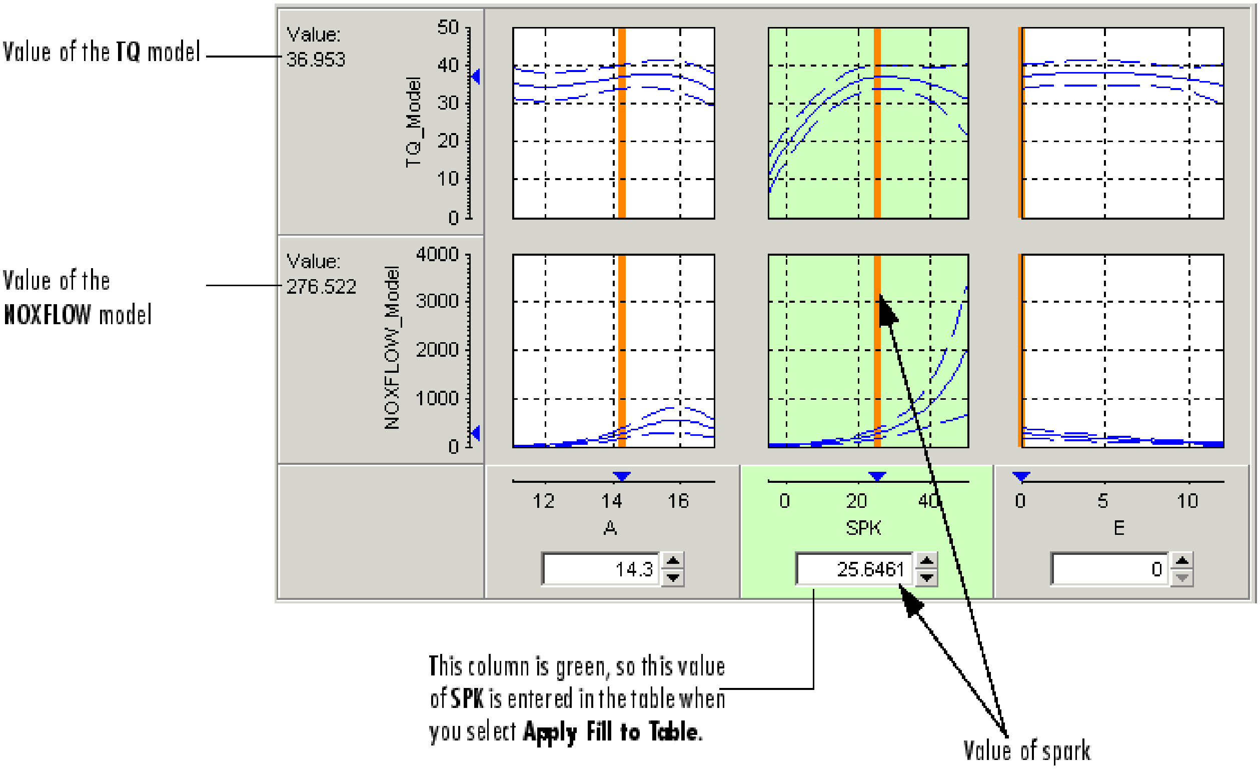
These cells have a model defined at that point. You use the display of these models to help you trade off values at these points to fulfill your aims in exactly the same way as when using independent models in "ordinary" tradeoff mode, as described in “Determining a Value at a Specific Operating Point” on page 5-15.

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 (▣).



Value of the **TQ** model

Value: 36.953

Value of the **NOXFLOW** model

Value: 276.522

A: 14.3

SPK: 25.6461

E: 0

This column is green, so this value of **SPK** is entered in the table when you select **Apply Fill to Table.**

Value of spark

3 When you are satisfied 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 fill the whole table by extrapolation by clicking ▣ . See “Filling the Table by Extrapolation” on page 4-30.

You can then export your calibration; see "Importing and Exporting Calibrations" on page 3-46. An example multimodel tradeoff is shown following.

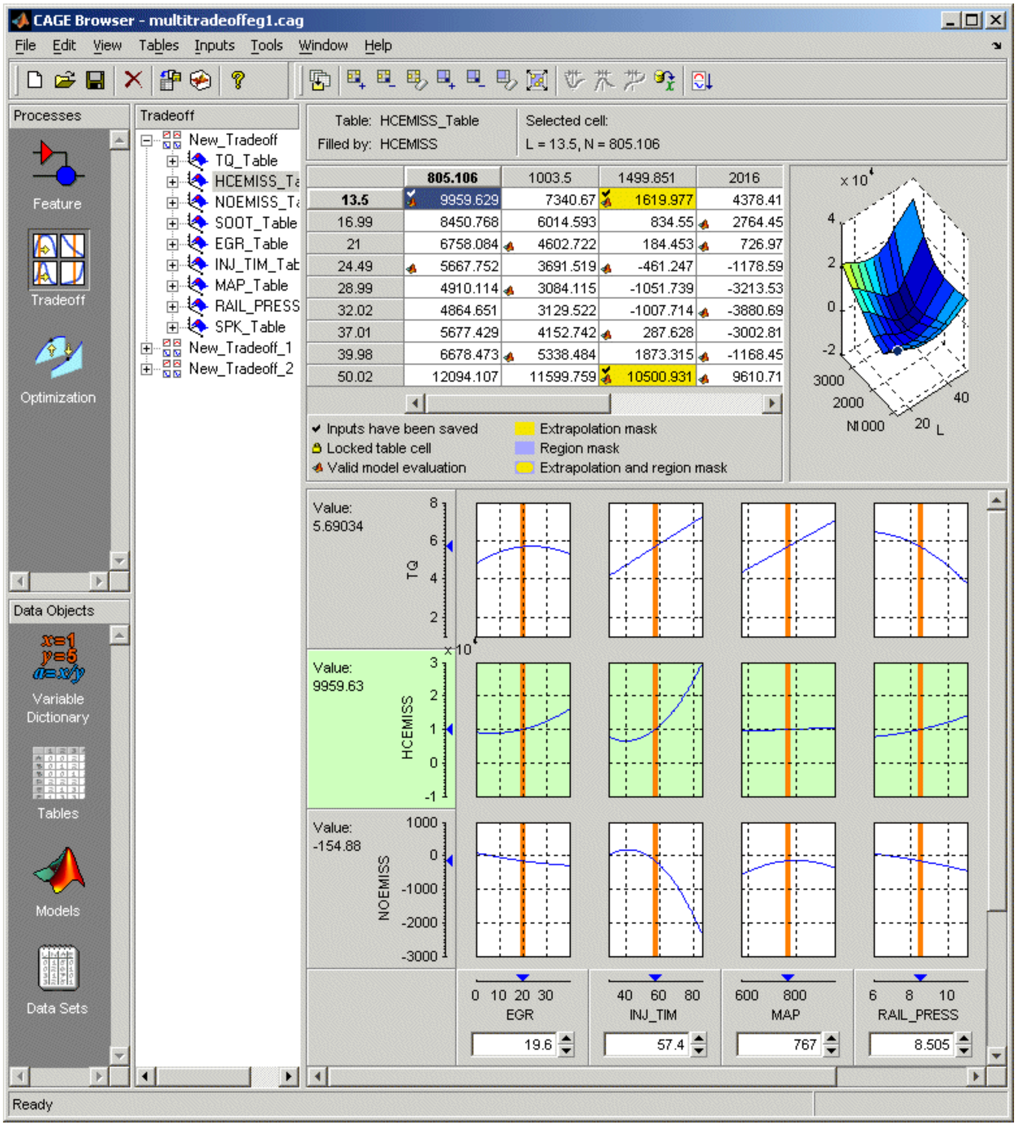


Table: HCEMISS_Table
Filled by: HCEMISS

Selected cell:
L = 13.5, N = 805.106

	805.106	1003.5	1499.851	2016
13.5	9959.629	7340.67	1619.977	4378.41
16.99	8450.768	6014.593	834.55	2764.45
21	6758.084	4602.722	184.453	726.97
24.49	5667.752	3691.519	-461.247	-1178.59
28.99	4910.114	3084.115	-1051.739	-3213.53
32.02	4864.651	3129.522	-1007.714	-3880.69
37.01	5677.429	4152.742	287.628	-3002.81
39.98	6678.473	5338.484	1873.315	-1168.45
50.02	12094.107	11599.759	10500.931	9610.71

✓ Inputs have been saved
🔒 Locked table cell
Valid model evaluation
🟨 Extrapolation mask
▨ Region mask
🟨 Extrapolation and region mask

Value: 5.69034

Value: 9959.63

Value: -154.88

EGR 19.6
INJ_TIM 57.4
MAP 767
RAIL_PRESS 8.505

Automated Tradeoff

You can use automated tradeoff to run an optimization routine and fill your tradeoff tables. Once you have set up an optimization 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 and the tradeoff values found are used to fill the selected cells. You can then fill the entire table by extrapolation.

You must first set up an optimization to use automated tradeoff.

See “Automated Tradeoff” on page 6-46 in the Optimization section for instructions.

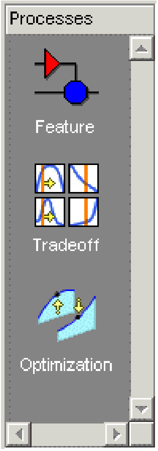
6

Optimization

This section includes the following topics:

Using the Optimization View

Optimization functionality is one of the CAGE processes. The **Optimization** button can be found in the left-hand **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 the optimization selected in the tree. When you first 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:

- “Setting Up Optimizations” on page 6-4
- “Optimization Output View” on page 6-30

  Once you have set up an optimization you can apply it to a region in set of tradeoff tables. See “Automated Tradeoff” on page 6-46

  You can define your own optimization functions for use in CAGE. See “User-Defined Optimization” on page 6-50

There is also a tutorial to guide you through the optimization functionality. See “Tutorial: Optimization and Automated Tradeoff” in the Getting Started documentation.

Setting Up Optimizations

The steps for setting up optimizations are described in the following sections:

- “Optimization Wizard” on page 6-4
- “Objectives and Constraints” on page 6-11
- “Defining Variable Values” on page 6-17
- “Running Optimizations” on page 6-22
- “Optimization View Toolbar” on page 6-23
- “Optimization Parameters Dialog” on page 6-24

For an example session you could open the `tradeoffInit.cag` file in the `mbctraining` folder.

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.

Optimization Wizard

You use the Optimization Wizard to:

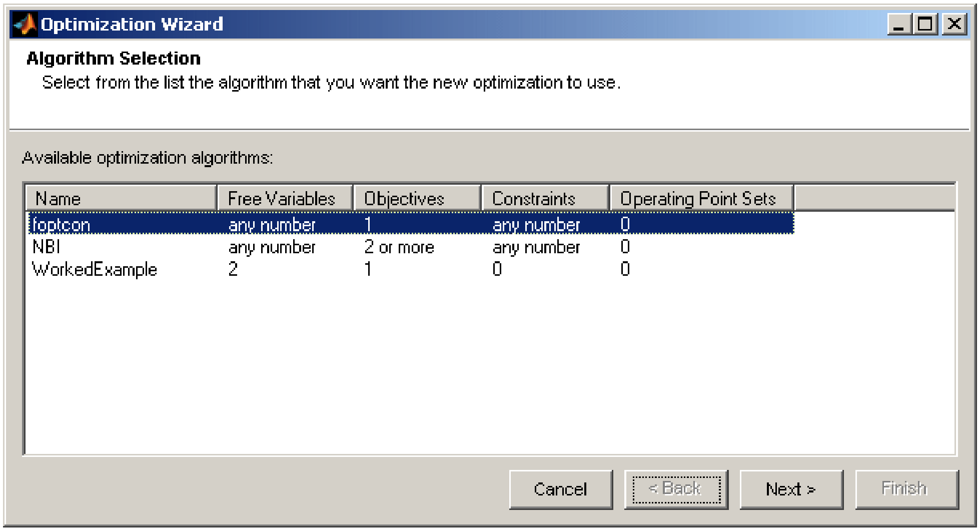
1. Choose algorithm
2. Set up free variables, objectives, and constraints options — “Optimization Wizard Step 2” on page 6-6
3. Select free variables — “Optimization Wizard Step 3” on page 6-8

   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 6-9
5. Set up model constraints — “Optimization Wizard Step 5” on page 6-10

6 Set up data sets (user defined optimizations only) — "Optimization Wizard Step 6" on page 6-11

Step 1. First you must choose your algorithm. The first screen of the Optimization Wizard is shown below.



Algorithm Selection

Select from the list the algorithm that you want the new optimization to use.

Available optimization algorithms:

Name	Free Variables	Objectives	Constraints	Operating Point Sets
foptcon	any number	1	any number	0
NBI	any number	2 or more	any number	0
WorkedExample	2	1	0	0

The first two algorithm choices in the list are standard routines you can use for constrained single and multiobjective optimization.

- `foptcon` is a single-objective optimization subject to constraints. This function uses the MATLAB `fmincon` algorithm from the Optimization Toolbox.
- `NBI` stands for Normal Boundary Intersection algorithm, which is multiobjective and can also be subject to constraints.

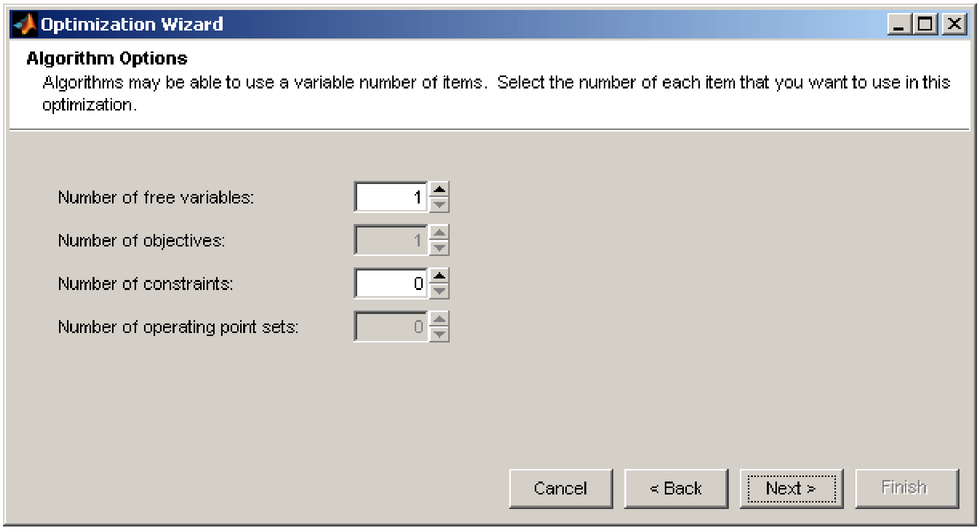
In many cases these standard routines are sufficient to allow you to solve your optimization problem. Sometimes, however, you might need to write a customized optimization algorithm; to do this you can use the supplied template to modify for your needs. Any optimization functions that you have checked into CAGE appear in this list. See "User-Defined Optimization" on page 6-50 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:



Optimization Wizard

Algorithm Options

Algorithms may be able to use a variable number of items. Select the number of each item that you want to use in this optimization.

Number of free variables: 1

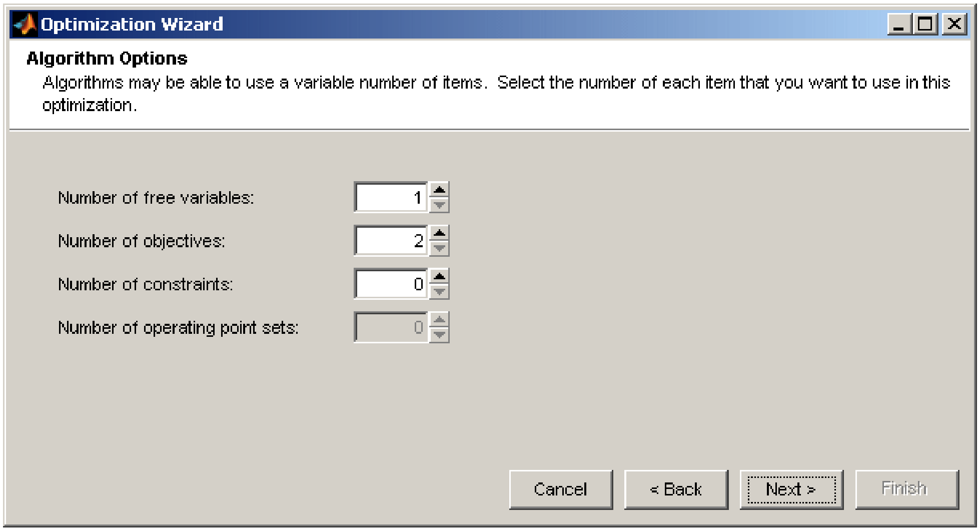
Number of objectives: 1

Number of constraints: 0

Number of operating point sets: 0

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:



Optimization Wizard

Algorithm Options

Algorithms may be able to use a variable number of items. Select the number of each item that you want to use in this optimization.

Number of free variables: 1

Number of objectives: 2

Number of constraints: 0

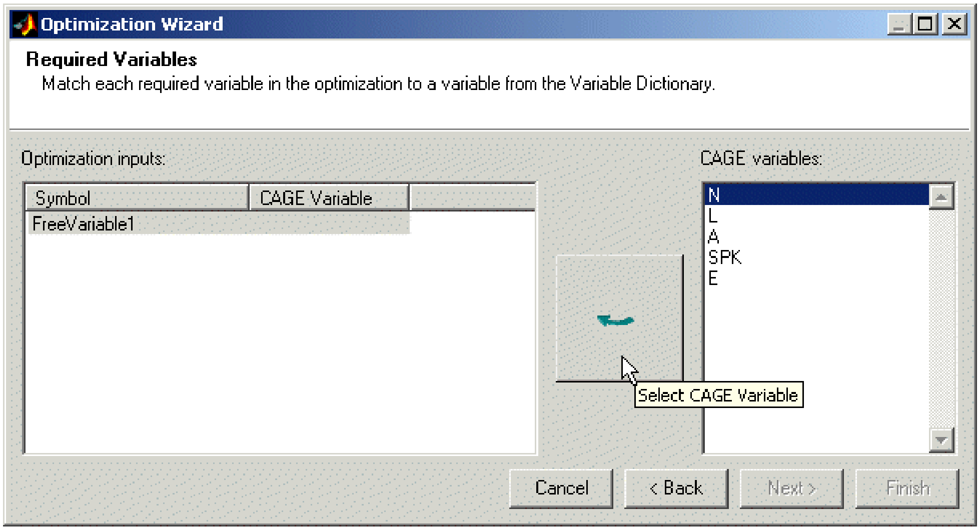
Number of operating point sets: 0

`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.



Optimization Wizard

Required Variables

Match each required variable in the optimization to a variable from the Variable Dictionary.

Optimization inputs:

Symbol	CAGE Variable
FreeVariable1	

CAGE variables:

N
L
A
SPK
E

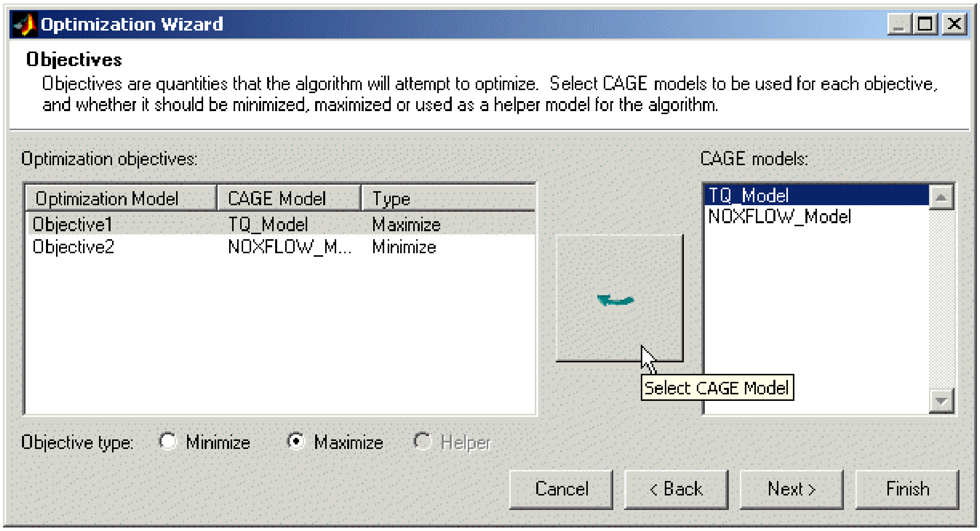
Use this screen to associate the variables from your CAGE session with the free variable(s) you want to use in the optimization. Select the correct pair in the right and left lists by clicking, then click the large button as indicated in the figure.

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 “Objective Editor” on page 6-12.



Optimization Wizard

Objectives

Objectives are quantities that the algorithm will attempt to optimize. Select CAGE models to be used for each objective, and whether it should be minimized, maximized or used as a helper model for the algorithm.

Optimization objectives:

Optimization Model	CAGE Model	Type
Objective1	TQ_Model	Maximize
Objective2	NOXFLOW_M...	Minimize

CAGE models:

- TQ_Model
- NOXFLOW_Model

Objective type: ○ Minimize ● Maximize ○ Helper

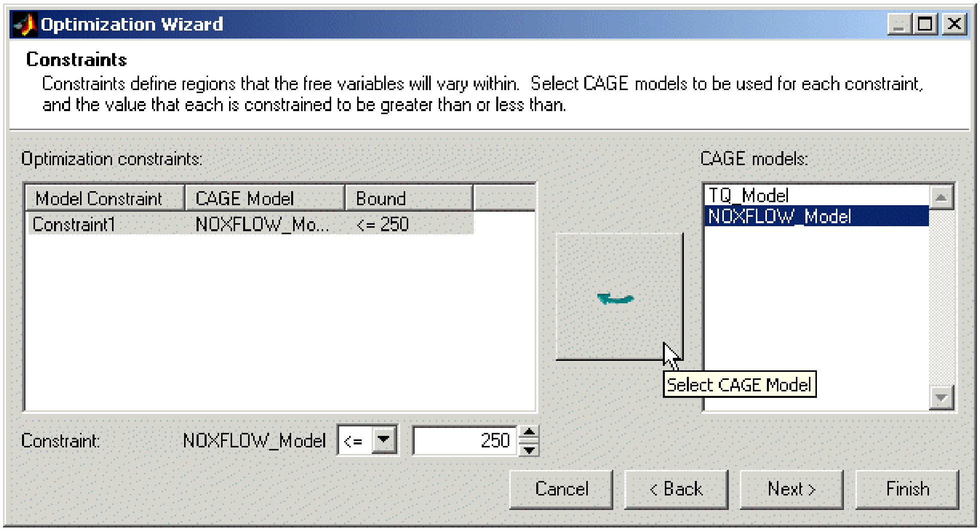
Here you can select which models from your session you want to use for the optimization, and whether you want to maximize or minimize the model output. The `foptcon` algorithm is for single objectives, so you can only maximize or minimize one model. The `NBI` algorithm can evaluate multiple objectives. For example, you might want to maximize torque while minimizing NOX emissions. Remember you can also define constraints later, for example, using emissions requirements.

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`).

- 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 6-11.

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 "Constraint Editor" on page 6-15.



Optimization Wizard

Constraints

Constraints define regions that the free variables will vary within. Select CAGE models to be used for each constraint, and the value that each is constrained to be greater than or less than.

Optimization constraints:

Model Constraint	CAGE Model	Bound
Constraint1	NOXFLOW_Mo...	<= 250

CAGE models:

TQ_Model
NOXFLOW_Model

Constraint: NOXFLOW_Model <= 250

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 enter a value in the edit box. Select the operator to define whether the optimization output should be constrained to be greater than or less than the value. The example shown is `NOXFLOW_Model <= 250`.

- Click **Finish** to complete the Optimization Wizard and return to the **Optimization** view. Note you can only set up point constraints in the wizard, but you can also set up sum constraints in the main **Optimization** view. See "Objectives and Constraints" on page 6-11.
- 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 this through the use of data sets to help you select optimization results. You can set up data sets here 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.

Objectives and Constraints

You can set up objectives and constraints from the main CAGE **Optimization** view, as well as within the Optimization Wizard.

To edit an objective or constraint,

- Double-click objectives and constraints in the **Objectives** or **Constraints** panes
- Right-click an objective or constraint and select **Edit**.
- Click to select an objective or constraint and select **Optimization > Objectives > Edit Objective**, or **Optimization > Constraints > Edit Constraint**.

You can also use the context and **Optimization** menus to **Add** or **Delete** objectives or constraints, if allowed by the algorithm (`foptcon` can only have a single objective).

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 Objective Editor and Constraint Editor 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.

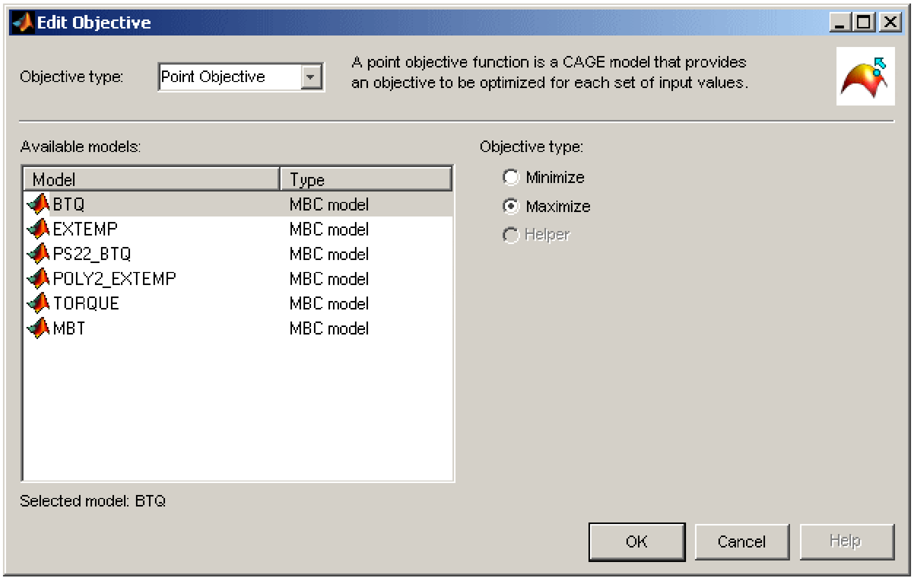
You can also set up linear, 1– and 2–D table, and ellipsoid constraints in the Constraint Editor, as for designs in the Model Browser part of the Model-Based Calibration Toolbox.

See

- “Objective Editor” on page 6-12
- “Constraint Editor” on page 6-15

Objective Editor

Double-click or right-click objectives to open the Edit Objectives dialog.



Edit Objective

Objective type: Point Objective

A point objective function is a CAGE model that provides an objective to be optimized for each set of input values.

Available models:

Model	Type
BTQ	MBC model
EXTEMP	MBC model
PS22_BTQ	MBC model
POLY2_EXTEMP	MBC model
TORQUE	MBC model
MBT	MBC model

Objective type:

○ Minimize
● Maximize
○ Helper

Selected model: BTQ

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.

Point Objectives

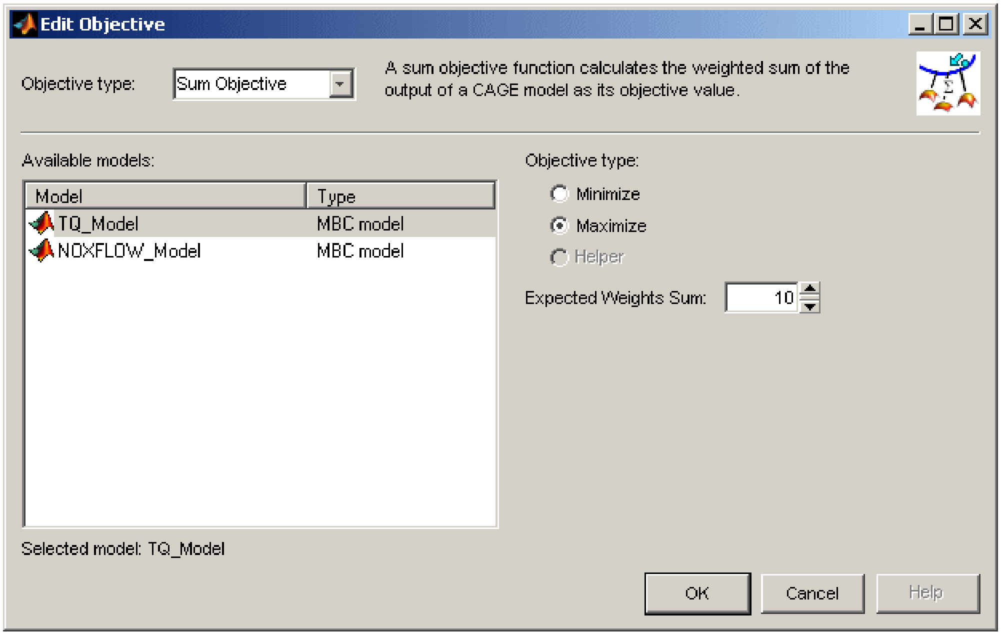
The preceding example shows the point objective controls. Select which models from your session you want to use for the optimization, and whether you want to maximize or minimize the model output. The `foptcon` algorithm is for single objectives, so you can only maximize or minimize one model. The `NBI` 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 6-9.

Sum Objectives

For weighted sum optimizations you must make all objectives sum objectives. See the following example.



Edit Objective

Objective type: Sum Objective

A sum objective function calculates the weighted sum of the output of a CAGE model as its objective value.

Available models:

Model	Type
TQ_Model	MBC model
NOXFLOW_Model	MBC model

Objective type:

○ Minimize
● Maximize
○ Helper

Expected Weights Sum: 10

Selected model: TQ_Model

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. The weights are applied to each solution to calculate the weighted sums. You can edit the weights in the **Fixed Variable Values** pane. This is the same process as selecting weights for the **Weighted Pareto View**. See “Weighted Objective Pareto Slice” on page 6-34.

In the Edit Objective dialog, the **Expected sum of weights** is optional. However if it is used (and set correctly), then it will improve the performance of the optimization. When the expected sum is set correctly, it scales the objective sum onto a range that is commensurate with the other optimization objectives and constraints.

The **Expected sum of weights** is a normalization value per run - all weights are divided by this, to try to make the normalized weights sum to 1 for each run. With this normalization, the objective sum should approximately vary on the same range as the other optimization objectives and constraints. For instance if you have two operating points per run, and the **Expected sum of weights** is 5, you can set the weights in the **Fixed Variable Values** pane to 2 and 3 for each operating point respectively, to make the normalized weights sum to 1. See "Using Variable Values Length Controls" on page 6-19 to set multiple operating points per run.

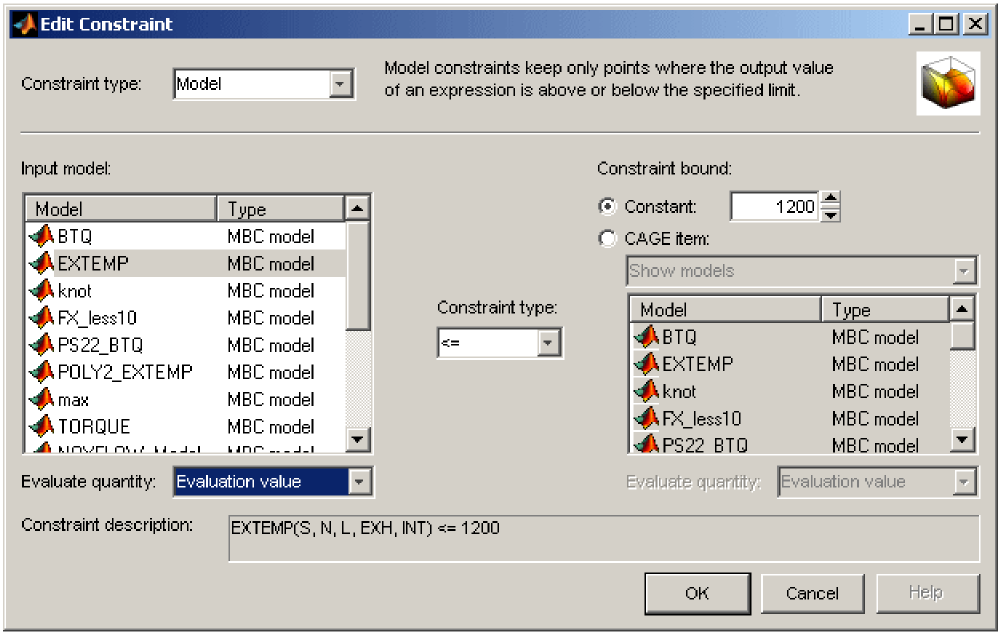
For a tutorial example of a sum optimization, see "Sum Optimization" in the Getting Started documentation.

Constraint Editor

Double-click or right-click constraints to open the Edit Constraints dialog. Here you can select `Linear`, `Ellipsoid`, `1D Table`, `2D Table`, `Model`, and `Sum Constraint` from the **Constraint type** drop-down menu. For a tutorial example of a sum optimization, see "Sum Optimization" in the Getting Started documentation.

You can have a mixture of point and sum constraints. For `fmincon` optimizations, if you have a weighted sum constraint, you must use sum objectives or the problem cannot be evaluated.

The model constraint settings are shown below.



Edit Constraint

Constraint type: Model

Model constraints keep only points where the output value of an expression is above or below the specified limit.

Input model:

Model	Type
BTQ	MBC model
EXTEMP	MBC model
knot	MBC model
FX_less10	MBC model
PS22_BTQ	MBC model
POLY2_EXTEMP	MBC model
max	MBC model
TORQUE	MBC model

Evaluate quantity: Evaluation value

Constraint type: <=

Constraint bound:

● Constant: 1200

○ CAGE item:

Show models

Model	Type
BTQ	MBC model
EXTEMP	MBC model
knot	MBC model
FX_less10	MBC model
PS22_BTQ	MBC model

Evaluate quantity: Evaluation value

Constraint description: EXTEMP(S, N, L, EXH, INT) <= 1200

For linear, ellipsoid and table constraints, see Constraint Types in the Designs chapter of the Model Browser documentation. These are the same constraints you can apply to designs in the Model Browser part of the Model-Based Calibration Toolbox. In the context of optimization you can also select any variable as a constraint input on the Input tab.

Model

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 radio 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**.

Sum Constraint

Use these for weighted sum optimizations. Choose a model, constraint bound value and operator. You can also select an **Expected sum of weights** normalization value per run to improve the performance of the optimization.

See the explanation under "Sum Objectives" on page 6-13.

See the tutorial "Sum Optimization" in the Getting Started documentation for a step-by-step example.

Defining Variable Values

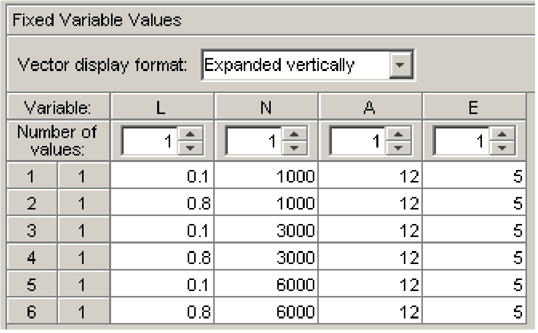
When you click **Finish** to complete the Optimization Wizard, you return to the Optimization view in CAGE. Your new optimization appears as a new node in the tree pane on the left, and the setup details appear on the right. You can now define variable values if you want.

In the optimization view you can use the Variable Values panes to define a set of operating points for the optimization. Note that you do not have to choose a set of operating points; if you do not, the optimization will run at a single point of your choosing (the set points of variables is the default).

Running the optimization requires the selected models to be evaluated (many times over) and hence values are required for all the model input factors. The default values for the fixed variables are their set points, as shown in the **Fixed Variable Values** pane. You have chosen one or more free variables, so the optimization will choose different values for those free variables in trying to find the best value of the objectives. The default initial value for a free variable is the set point, as shown in the **Free Variable Initial Values** pane.

To define the set of operating points for the optimization, 1 In the **Free Variable Initial Values** pane, increase the **Number of runs**. New rows appear in both fixed and free variable values panes, all containing the default set point values of each variable. Each row defines an operating point for an optimization run.

2 Edit the values in the **Fixed Variable Values** pane to define the points to run the optimization. You can select **Optimization > Import From Data Set** if you have suitable variables to import, or 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. An example is shown.



Fixed Variable Values

Vector display format: Expanded vertically

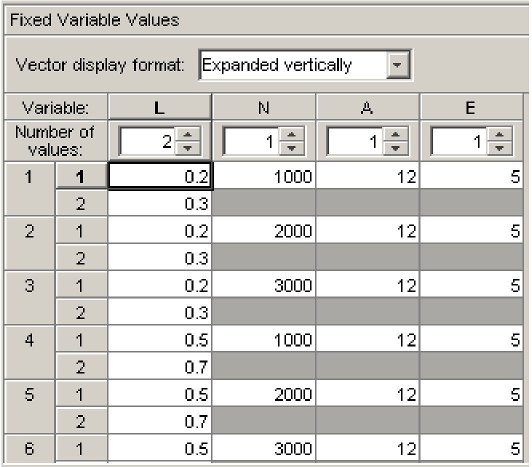
Variable:		L	N	A	E
Number of values:		1	1	1	1
1	1	0.1	1000	12	5
2	1	0.8	1000	12	5
3	1	0.1	3000	12	5
4	1	0.8	3000	12	5
5	1	0.1	6000	12	5
6	1	0.8	6000	12	5

3 Similarly you can edit the values in the **Free Variable Initial Values** pane to define the starting values of the free variables if you want, or you can leave these at the default.

Note for `foptcon` optimizations you can specify a number of initial starting values per run, see "foptcon Optimization Parameters" on page 6-24.

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 defined in the Variable Dictionary.

4 You can increase the number of values of any variable per run, as shown in the following example. You can edit the **Number of Values** directly or you can select **Optimization > Set Variable Length** to change all variable lengths at once. Use the **Vector display format** drop-down list to change how these are shown.



Fixed Variable Values

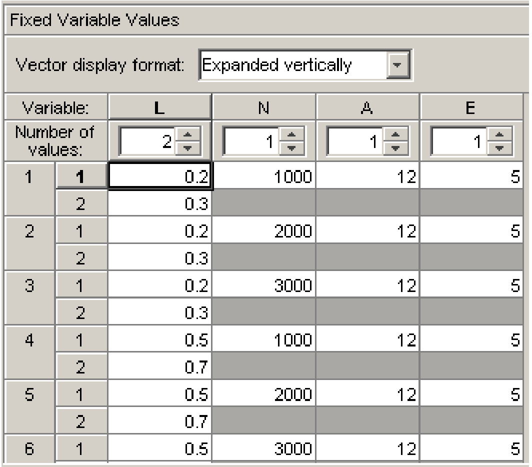
Vector display format: Expanded vertically

Variable:		L	N	A	E
Number of values:		2	1	1	1
1	1	0.2	1000	12	5
	2	0.3			
2	1	0.2	2000	12	5
	2	0.3			
3	1	0.2	3000	12	5
	2	0.3			
4	1	0.5	1000	12	5
	2	0.7			
5	1	0.5	2000	12	5
	2	0.7			
6	1	0.5	3000	12	5

Note an foptcon point optimization cannot have a **Number of values** greater than one. See the next section "Using Variable Values Length Controls" on page 6-19 for details.

Using Variable Values Length Controls

At the optimization node the fixed and free variable values panes have **Number of Values** controls for each 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. If you increase the **Number of Values** of a fixed or free variable, then the number of operating points within each run increases, as shown in the following example.



Fixed Variable Values

Vector display format: Expanded vertically

Variable:		L	N	A	E
Number of values:		2	1	1	1
1	1	0.2	1000	12	5
	2	0.3			
2	1	0.2	2000	12	5
	2	0.3			
3	1	0.2	3000	12	5
	2	0.3			
4	1	0.5	1000	12	5
	2	0.7			
5	1	0.5	2000	12	5
	2	0.7			
6	1	0.5	3000	12	5

In this example, each run contains two different values of L, so each run contains two operating points.

- N **Number of Values** = 1; L **Number of Values** = 2
- Run 1: N = 1000 L = [0.2;0.3]

  Run 2: N = 2000 L = [0.2;0.3] etc.

	N	L
Run 1, point 1	1000	0.2
Run 1, point 2	1000	0.3
Run 2, point 1	2000	0.2
Run 2, point 2	2000	0.3

For an example using these controls, see the tutorial section “Sum Optimization” in the Getting Started documentation.

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**.

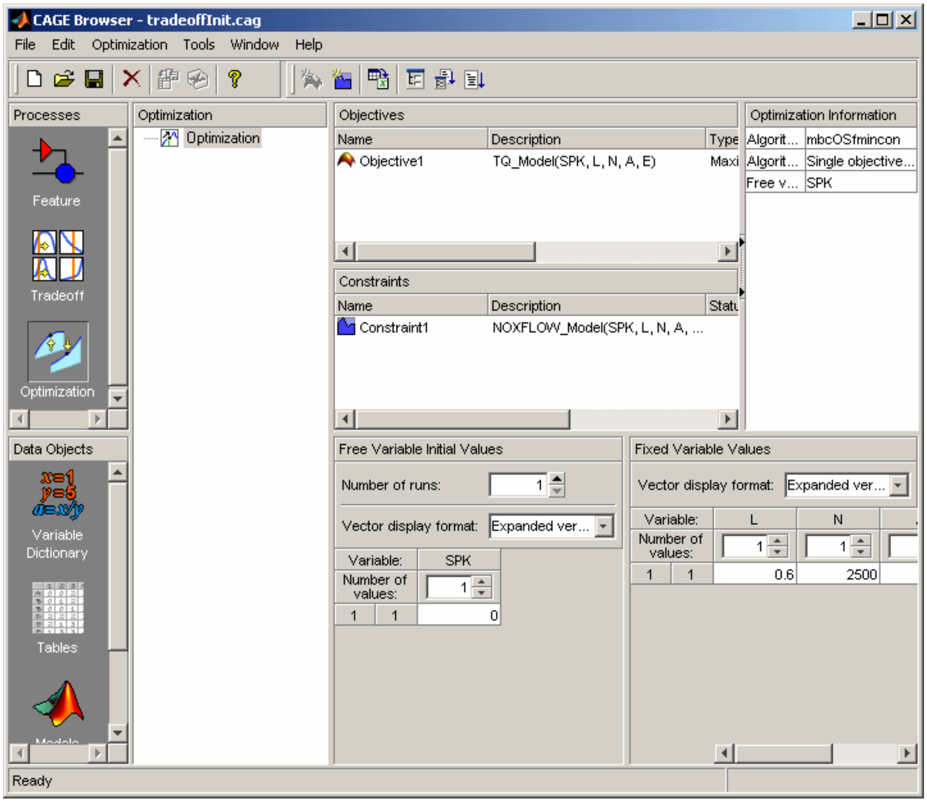
The following table shows the input/output relationships for objectives and constraints.

Maximum Input Length	Output length for objectives/constraints			
	Point objective	Sum objective	Sum Model Constraint	All other constraints (linear, 1D, 2D, model)
1	1	1	1	1
N (>1)	N (not allowed for foptcon)	1	1	N

An foptcon point objective can have only one output — therefore you cannot have a **Number of Values** greater than one for an foptcon single-objective optimization unless you use a sum objective (which returns a single output per run regardless of length). If the **Number of Values** is more than one for a sum objective, the result is the sum across all operating points within a run.

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:



Objectives

Name	Description	Type
Objective1	TQ_Model(SPK, L, N, A, E)	Maxi

Constraints

Name	Description	Statu
Constraint1	NOXFLOW_Model(SPK, L, N, A, ...	

Optimization Information

Algorit...	mbcOSfmincon
Algorit...	Single objective...
Free v...	SPK

Free Variable Initial Values

Number of runs: 1

Vector display format: Expanded ver...

	Variable:	SPK
	Number of values:	1
1	1	0

Fixed Variable Values

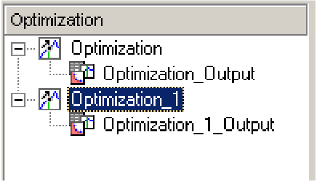
Vector display format: Expanded ver...

	Variable:	L	N
	Number of values:	1	1
1	1	0.6	2500

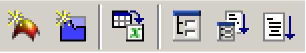
If your optimization is ready to run you can click **Run Optimization** in the toolbar to proceed. You may want to define variable values before running the optimization. If you need to set up any objectives or constraints **Run** will not be enabled. If your optimization is ready to run you can also 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" on page 6-24, then when you click **OK** the optimization process begins.
- 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. See "Optimization Output View" on page 6-30.



Optimization View 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 Objective Editor, select a model, and set whether to maximize or minimize.
- 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.
- Import initial data from a data set — you can populate the Variable Values panes by importing from a data set.
- Set Up Optimization — Opens the Optimization Parameters dialog where you can change optimization settings such as tolerances and number of solutions. When you close the dialog the settings are saved but the optimization does not run.

- Set Up and Run Optimization — Opens the Optimization Parameters dialog where you can change optimization settings such as tolerances and number of solutions. When you close the dialog the optimization requests starting values and then runs.
- Run Optimization — Starts the optimization. See "Running Optimizations" on page 6-22 above.

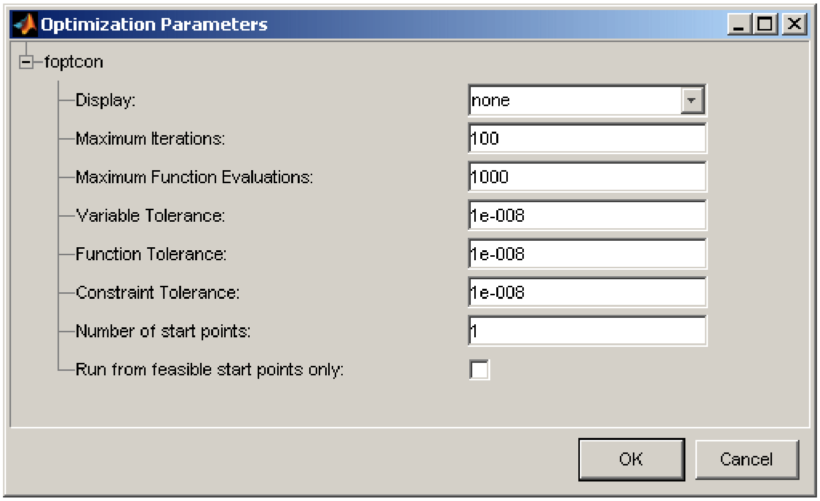
Optimization Parameters Dialog

The settings in this dialog are algorithm specific. Note if you edit these settings and later want to return to the defaults, select **Optimization > Reset Parameters**.

Following is an example showing the foptcon options. For the NBI options, see the next section, "NBI Optimization Parameters" on page 6-25.

foptcon Optimization Parameters

The foptcon optimization algorithm in CAGE uses the MATLAB fmincon algorithm from the Optimization Toolbox. 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.


Optimization Parameters

foptcon

Display: none

Maximum Iterations: 100

Maximum Function Evaluations: 1000

Variable Tolerance: 1e-008

Function Tolerance: 1e-008

Constraint Tolerance: 1e-008

Number of start points: 1

Run from feasible start points only: □

- **Display** — choose `none`, `iter`, or `final`. 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.
- **Maximum function evaluations** — Choose a positive integer.

  Maximum number of function evaluations allowed
- **Maximum iterations** — Choose a positive integer.

  Maximum number of iterations allowed
- **Variable tolerance** — Choose a positive scalar.

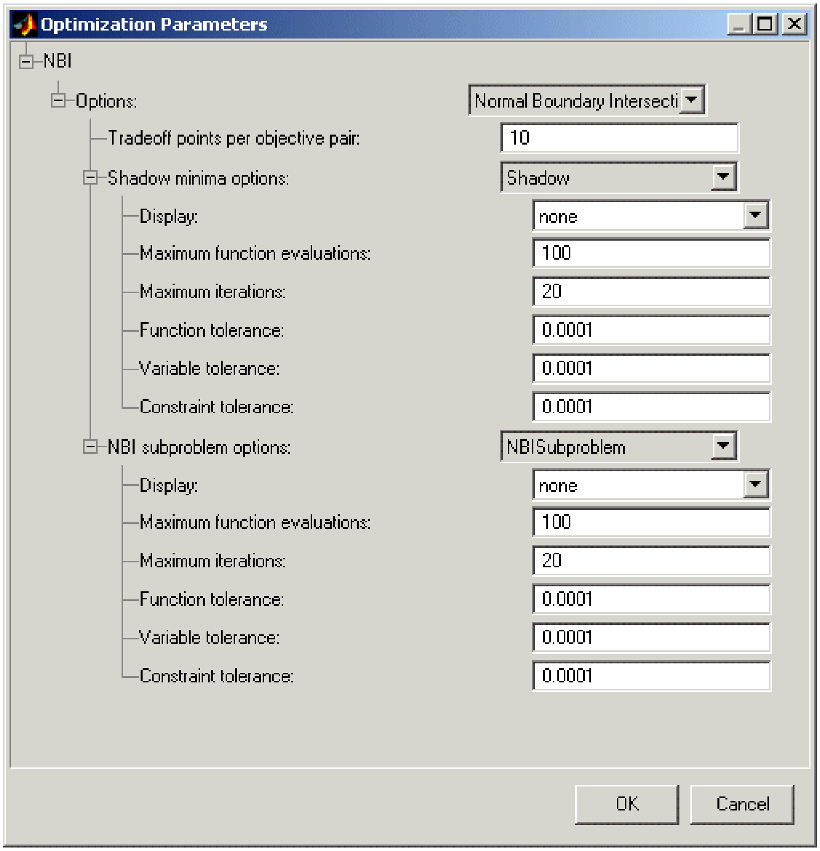
  Termination tolerance on the free variables
- **Function tolerance** — Choose a positive scalar.

  Termination tolerance on the function value
- **Constraint tolerance** — Choose a positive scalar.

  Termination tolerance on the constraint violation
- **Number of start points** — Choose a positive integer, N. (N-1) start points per run are chosen using a latin hypercube design to cover the space, in addition to the starting value specified in the Free Variable Initial Values pane.
- **Run from feasible start points only** — select this 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.

NBI Optimization Parameters

The example following shows the `NBI` options in the Optimization Parameters dialog.



- NBI
  - Options: Normal Boundary Intersecti
    - Tradeoff points per objective pair: 10
    - Shadow minima options: Shadow
      - Display: none
      - Maximum function evaluations: 100
      - Maximum iterations: 20
      - Function tolerance: 0.0001
      - Variable tolerance: 0.0001
      - Constraint tolerance: 0.0001
    - NBI subproblem options: NBISubproblem
      - Display: none
      - Maximum function evaluations: 100
      - Maximum iterations: 20
      - Function tolerance: 0.0001
      - Variable tolerance: 0.0001
      - Constraint tolerance: 0.0001

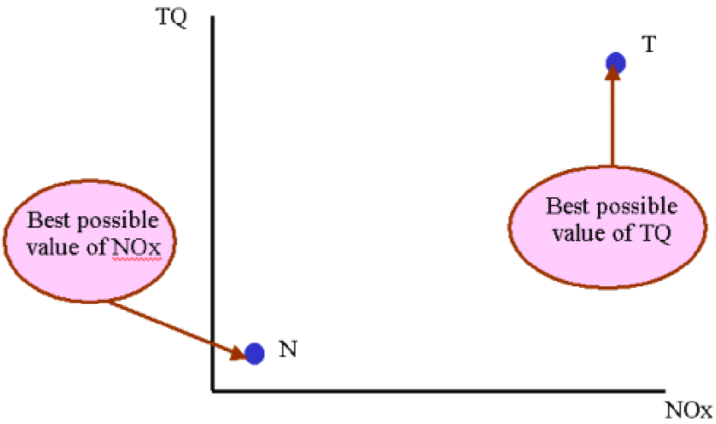
Background on the NBI (Normal Boundary Intersection Algorithm)

To understand the options for the NBI algorithm, some limited understanding of the algorithm is required. For more information on the NBI algorithm, see the NBI home page at the following URL:

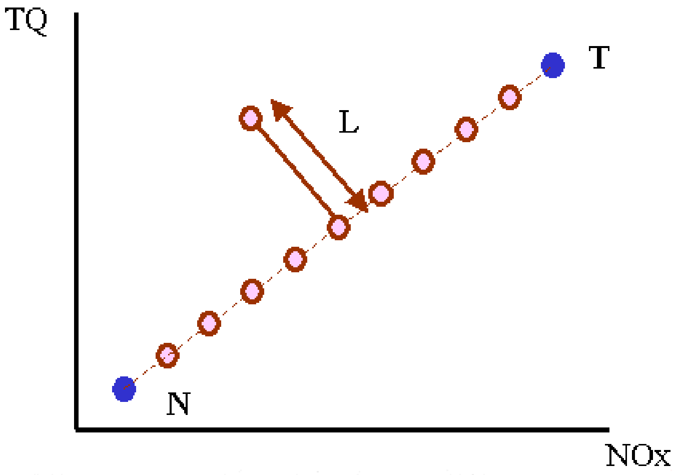
http://www.caam.rice.edu/~indra/NBIhomepage.html

The NBI algorithm is performed in two steps. The first step is to find the global extrema of each objective individually. This is called the shadow minima problem, and is a single-objective problem for each objective function.

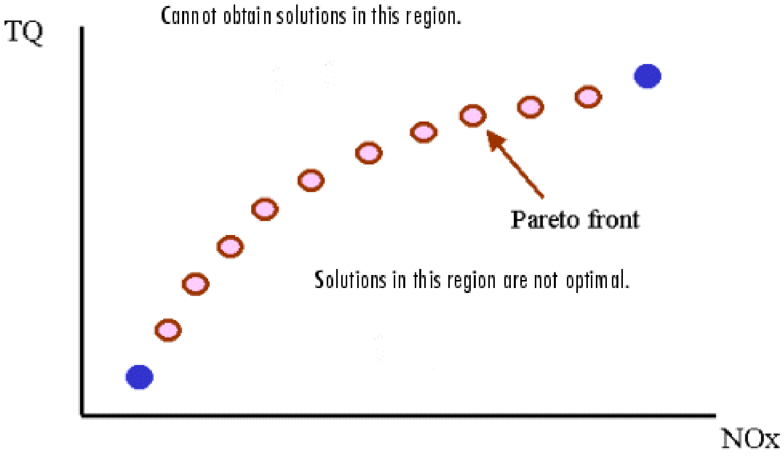
The MATLAB routine `fmincon` is used to find these extrema. Once these extrema are found, they can be plotted against each other. For example, consider an `NBI` optimization that simultaneously maximizes TQ and minimizes NOX emissions. A plot of the extrema against each other might resemble the following.



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 extrema. 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 figure above shows spacing of the points between the extrema along the (n-1) surface. The algorithm tries to maximize the distance `L` along the normal away from the surface. The following figure shows the final solution found by the NBI algorithm.


Cannot obtain solutions in this region.
TQ
Pareto front
Solutions in this region are not optimal.
NOx

NBI Options

- **Tradeoff points per objective pair** (Np)

The number of tradeoff solutions between your objectives that you want to find, Npts, is determined by the following formula

$$Npts = \binom{n + Np - 2}{Np - 1}$$

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),

  $$Npts = Np$$

- For problems with three objectives (n = 3),

  $$Npts = \frac{Np(Np + 1)}{2}$$

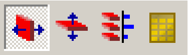
- **Shadow minima options** and **NBI subproblem options**

  As the NBI algorithm uses the MATLAB fmincon algorithm to solve the shadow minima problem and the NBI subproblems, the options here are similar to those for the foptcon library function. For more information on these options, see the previous section, "foptcon Optimization Parameters" on page 6-24.

Optimization Output View

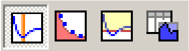
When you have run an optimization an `Output` node appears in the optimization tree and the **Optimization Output** views appear. Use the toolbar buttons to determine what is displayed in the table and the graph views. The first default view is the Solution Slice table and the Objective Slice Graphs.

Use these toolbar buttons or the **View** menu to select the following Table Views:



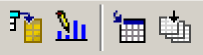
- “Solution Slice” on page 6-31
- “Pareto Slice” on page 6-33
- “Weighted Objective Pareto Slice” on page 6-34
- “Selected Solution Slice” on page 6-36

Use these toolbar buttons to select the following Graph Views:



- “Objective Slice Graphs” on page 6-38
- “Pareto Front Graphs” on page 6-39
- “Constraint Slice Graphs” on page 6-40
- “Constraint Summary Table” on page 6-40

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.



The last four toolbar buttons are also in the **Solution** menu:

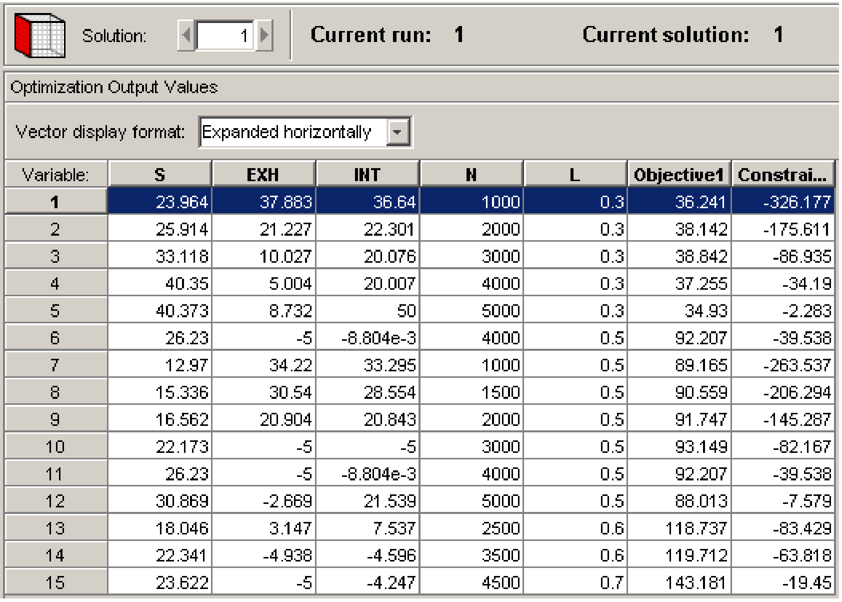
- Select solution — this is for multiobjective optimization, used for choosing your preferred solution for each operating point. See "Selected Solution Slice" on page 6-36.
- Edit pareto weights — used for evaluating weighted sums. See "Weighted Objective Pareto Slice" on page 6-34.
- Export to data set — exports the table visible in the current view only to a new data set (the name of the data set is taken from the name of the optimization node).
- Fill tables using optimal solutions — opens the Table Filling From Optimization Results Wizard. See "Filling Tables From Optimization Results" on page 6-40

Check the **Solution Information** pane for details such as the `Output message` for the selected solution. Here you can see, for example, if an foptcon optimization run terminated because no feasible start point was found.

Solution Slice

The Solution Slice view (click) shows a table with one solution at all operating points.

The following example shows a Solution Slice table display.



Solution: 1 Current run: 1 Current solution: 1

Optimization Output Values

Vector display format: Expanded horizontally

Variable:	S	EXH	INT	N	L	Objective1	Constrai...
1	23.964	37.883	36.64	1000	0.3	36.241	-326.177
2	25.914	21.227	22.301	2000	0.3	38.142	-175.611
3	33.118	10.027	20.076	3000	0.3	38.842	-86.935
4	40.35	5.004	20.007	4000	0.3	37.255	-34.19
5	40.373	8.732	50	5000	0.3	34.93	-2.283
6	26.23	-5	-8.804e-3	4000	0.5	92.207	-39.538
7	12.97	34.22	33.295	1000	0.5	89.165	-263.537
8	15.336	30.54	28.554	1500	0.5	90.559	-206.294
9	16.562	20.904	20.843	2000	0.5	91.747	-145.287
10	22.173	-5	-5	3000	0.5	93.149	-82.167
11	26.23	-5	-8.804e-3	4000	0.5	92.207	-39.538
12	30.869	-2.669	21.539	5000	0.5	88.013	-7.579
13	18.046	3.147	7.537	2500	0.6	118.737	-83.429
14	22.341	-4.938	-4.596	3500	0.6	119.712	-63.818
15	23.622	-5	-4.247	4500	0.7	143.181	-19.45

The **Solution Slice** view shows a table of one solution at all operating points in the set. For single-objective optimizations there is only one solution per operating point, so the Solution Slice is the only useful view and the **Solution** controls at the top are disabled. 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. Note that if you export the output to a data set (using the toolbar button) it is the current table that is exported.

Click in 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 6-38 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 6-40. 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 6-39 which show the available solutions with the current selection highlighted in red.
- You can also display the “Constraint Summary Table” on page 6-40 which details the distance to each constraint edge for the selected operating point in the table. This 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 this.

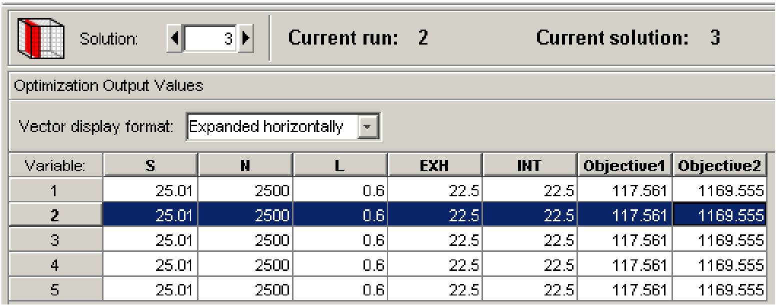
Note that before you run an NBI optimization you can specify how many solutions you want the optimization to find, using the Set Up and Run Optimization toolbar button.

For information on selecting best solutions at each operating point for subsequent export to a data set, see “Selected Solution Slice” on page 6-36.

Pareto Slice

The Pareto Slice table view (click) is for multiobjective optimization where there is more than one solution at each point. 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.

Note that 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.



Solution: 3 Current run: 2 Current solution: 3

Optimization Output Values

Vector display format: Expanded horizontally

Variable:	S	N	L	EXH	INT	Objective1	Objective2
1	25.01	2500	0.6	22.5	22.5	117.561	1169.555
2	25.01	2500	0.6	22.5	22.5	117.561	1169.555
3	25.01	2500	0.6	22.5	22.5	117.561	1169.555
4	25.01	2500	0.6	22.5	22.5	117.561	1169.555
5	25.01	2500	0.6	22.5	22.5	117.561	1169.555

You can also display “Pareto Front Graphs” on page 6-39 (click) which show the available solutions with the current selection highlighted in red.

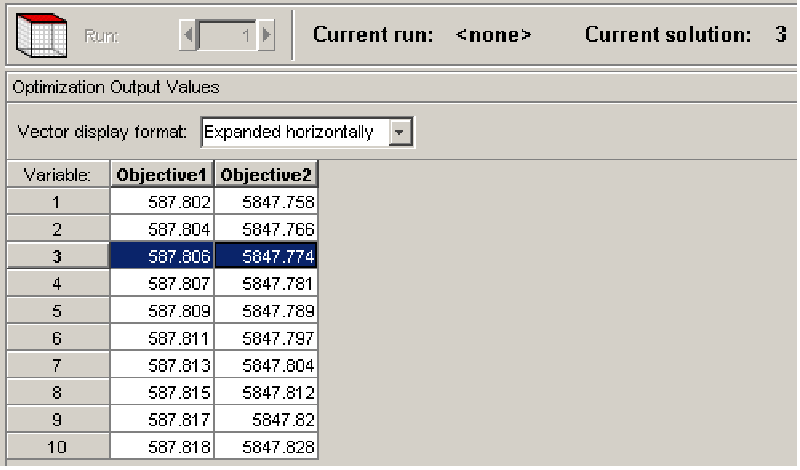
Use the pareto front graphs and “Objective Slice Graphs” on page 6-38 to select the best solution for the operating point. If you have constraints you can also use the “Constraint Slice Graphs” on page 6-40 and “Constraint Summary Table” on page 6-40 to help you decide which solution to choose for each run.

When you have decided which solution you want to use for the currently selected operating point you can select it as best by clicking Select Solution ( ) in the toolbar. You cannot select solutions until you enable the Selected Solutions view. See “Selected Solution Slice” on page 6-36. You can also select best solutions in the Solution Slice view, see “Solution Slice” on page 6-31 .

Weighted Objective Pareto Slice

The **Weighted Objective Pareto Slice** view (click) shows a weighted sum Pareto solution. This is a weighted sum of the objective values over all operating points for each solution. For a single objective optimization there is a single cell — 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 can be useful, for example, for evaluating total emissions across a drive cycle. The default weights are unity (`1`) for each run.



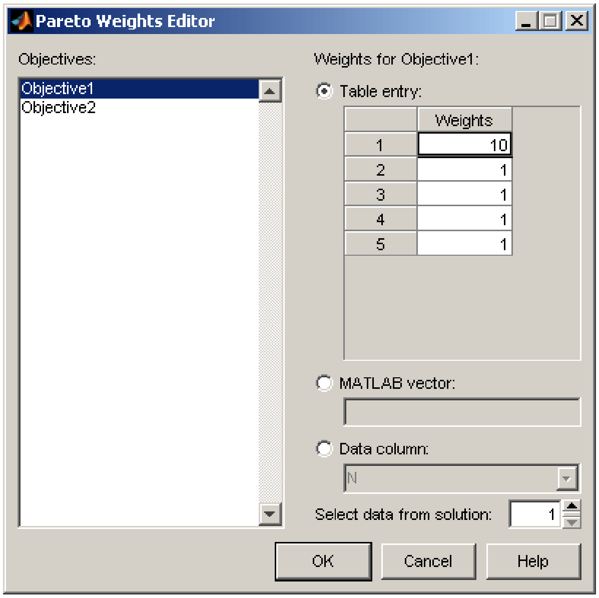
Current run: <none> Current solution: 3

Optimization Output Values

Vector display format: Expanded horizontally

Variable:	Objective1	Objective2
1	587.802	5847.758
2	587.804	5847.766
3	587.806	5847.774
4	587.807	5847.781
5	587.809	5847.789
6	587.811	5847.797
7	587.813	5847.804
8	587.815	5847.812
9	587.817	5847.82
10	587.818	5847.828

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.



Objectives:

Objective1
Objective2

Weights for Objective1:

● Table entry:

	Weights
1	10
2	1
3	1
4	1
5	1

○ MATLAB vector:

○ Data column: N

Select data from solution: 1

Here you can select models to sum, and select weights for any operating point by clicking and editing, as shown in the example above. 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 data column in your data set by selecting the other radio buttons. If you select **Data 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.

Selected Solution Slice

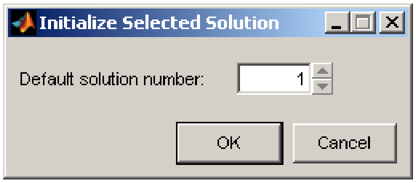
In a multiobjective optimization, there is more than one possible optimal solution at each operating point. You can use the **Selected Solutions** view to collect and export those solutions you have decided are optimal at each operating point.

Once you have enabled the **Selected Solution** view, you can use the plots in the Pareto Front Graphs view and Solution Slice table view to help you select best solutions for each operating point. 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, or use your optimization output to fill tables.

1 You cannot select best solutions until you have enabled the **Selected Solutions** view. Do this by selecting **Solution -> Selected Solution -> Initialize**.

2 A dialog called **Create Selected Solution** appears. The default 1 initializes the first solution for each operating point 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

operating point. When you click **OK**, the toolbar buttons for the **Selected Solutions** view and **Select Solution** are enabled.



Default solution number: 1

3 Once you have enabled the **Selected Solutions** view, you can use the table views (Solution Slice and Pareto Slice) and the plots in the graphs (Objective Slice, Pareto Front, and Constraint Slice graphs) to help you select best solution for each run.

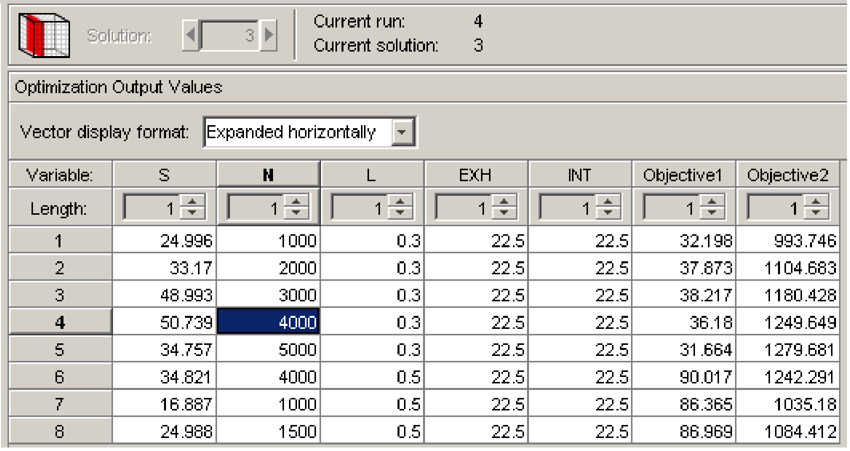
   a Click in the tables to select a point to display in the graphs until you can decide which solution you want for a point.

   b Click Select Solution ( ) in the toolbar to select the current solution as best.

   Repeat 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 operating point, and solution 6 for the second, and so on. You can then use your chosen optimization output for each point to fill tables, or choose the Export to Data Set toolbar and **Solution** menu option. See "Filling Tables From Optimization Results" on page 6-40.

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 here. The solution chosen as best (in the other views) for the currently selected operating point is displayed in the grayed out edit box.



Solution: 3

Current run: 4
Current solution: 3

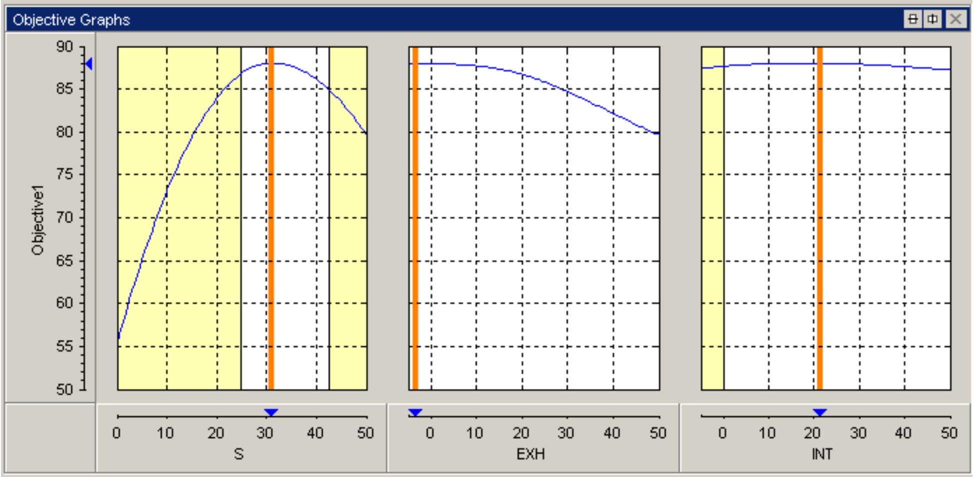
Optimization Output Values

Vector display format: Expanded horizontally

Variable:	S	N	L	EXH	INT	Objective1	Objective2
Length:	1	1	1	1	1	1	1
1	24.996	1000	0.3	22.5	22.5	32.198	993.746
2	33.17	2000	0.3	22.5	22.5	37.873	1104.683
3	48.993	3000	0.3	22.5	22.5	38.217	1180.428
4	50.739	4000	0.3	22.5	22.5	36.18	1249.649
5	34.757	5000	0.3	22.5	22.5	31.664	1279.681
6	34.821	4000	0.5	22.5	22.5	90.017	1242.291
7	16.887	1000	0.5	22.5	22.5	86.365	1035.18
8	24.988	1500	0.5	22.5	22.5	86.969	1084.412

Objective Slice Graphs

The objective slice graphs are displayed by default for optimization output views, or you can select in the toolbar.



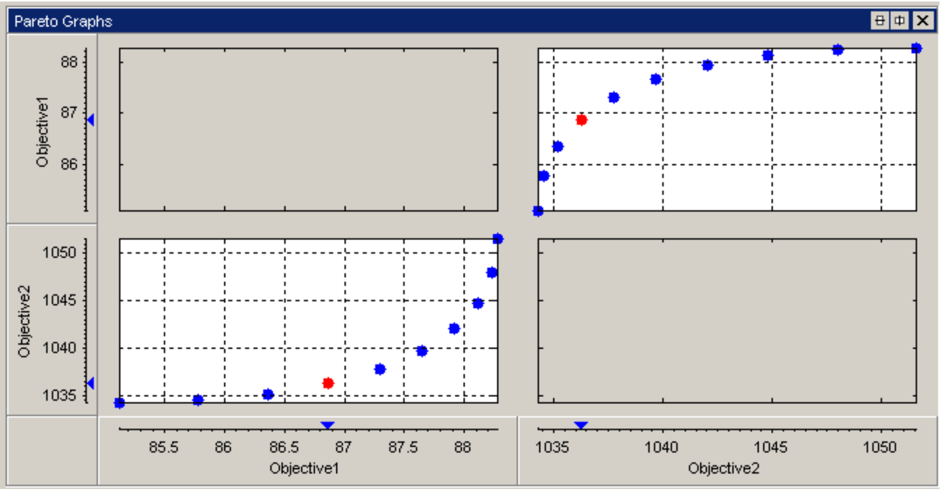
The objective slice graphs show the objective functions at the run selected in the table, with the solution value in orange. Whether the table is displaying a solution slice or pareto slice, the cell you select in the table is always displayed in the graphs.

The yellow areas show a region outside a constraint (such as a boundary constraint model exported from the Model Browser part of the Model-Based Calibration Toolbox, or any other optimization constraint). All constraint regions in optimization displays (as in the rest of the toolbox) are shown in yellow.

Pareto Front Graphs

The Pareto Front Graphs (click) are for multiobjective optimization where there is more than one solution at each point. The Pareto Front graphs show the available solutions for the selected run with the current selection highlighted in red. Click in the tables or graphs to select solutions. The selected solution is displayed in all other graphs (objective and constraint).

Note that 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.



You can use the Pareto Front graphs 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 6-36.

Constraint Slice Graphs

The Constraint Slice graphs (click) show the constraint functions at the selected operating point with the solution value in orange. Click in the tables to select solutions to display. Yellow areas show a region outside a constraint.

Constraint Summary Table

The Constraint Summary Table (click) view displays the distance to each constraint edge for the selected solution in the table — this 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 this. If you are using equality constraints the graphs can be entirely yellow (as there is only a solution at a single point) and you can only see whether a feasible solution has been found by looking at the Constraint Summary Table.

Filling Tables From Optimization Results

There are two methods for filling tables with optimization results.

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 fill and click the button to add them to the list of tables to be filled. Click **Next**.

3. For each table to be filled, select the correct variable or model output from the list of optimization results and click the button to match them, as shown in this example.

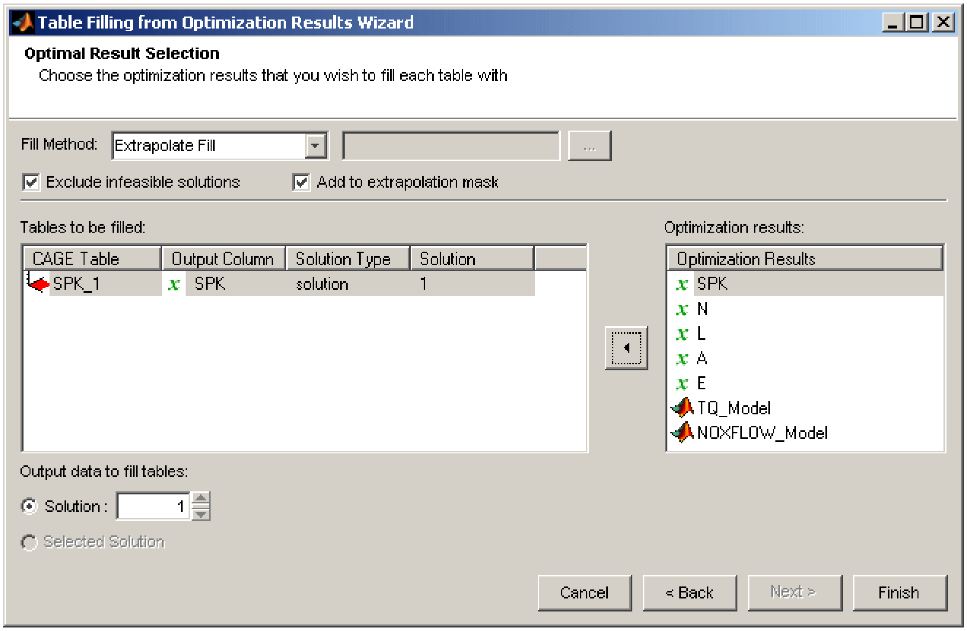


Table Filling from Optimization Results Wizard

Optimal Result Selection

Choose the optimization results that you wish to fill each table with

Fill Method: Extrapolate Fill

☑ Exclude infeasible solutions ☑ Add to extrapolation mask

Tables to be filled:

CAGE Table	Output Column	Solution Type	Solution
SPK_1	*x* SPK	solution	1

Optimization results:

Optimization Results
x SPK
x N
x L
x A
x E
TQ_Model
NOXFLOW_Model

Output data to fill tables:

● Solution : 1

○ Selected Solution

In a single objective optimization there is only one solution for each operating point so by default you will fill the table with solution `1`. In a multiobjective optimization there is more than one solution per point. You can select a solution number to use for every point, or you can select the other radio button to use your **Selected Solution** for each point. To collect your preferred solutions you must first use the “Selected Solution Slice” on page 6-36.

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 6-43.

5 **Exclude infeasible solutions** — select this check box to include only optimization results that meet the constraints. This does not depend on the exit status of the optimization (exit flag), only a check to see if constraints are met. To view the exit flag and algorithm termination messages check the **Solution Information** pane in the Optimization Output view.

6 **Add to extrapolation mask—** when this check box is selected, filled table cells are added to the mask only if their breakpoints exactly match points that have been run in the optimization.

   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 optimizations. 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 reporting successful table filling. Switch to the **Tables** view to examine the tables.

The other method of filling tables with optimization output uses Data Sets.

1 From the `Optimization_Output` optimization output node, click Export to Data Set ( ) in the toolbar (or select **Solution > Export to Data Set**)

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. The new data set takes the name of the optimization and the suffix identifies that which solution number you exported from the optimization view.

   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 fill 2-D tables from optimization results.

All 2-D custom fill 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 optimization results (`NF-by-1`)

Input	Description
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 filled (1-by-NCOL)
rowaxis	Row breakpoints of table to be filled (NROW-by-1)
currtabdata	Existing table values of table to be filled (NROW-by-NCOL)

The function must pass three output arguments back to CAGE, to allow CAGE to fill the table:

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-NCOL)
fillmask	Logical matrix to indicate cells to be added to the extrapolation mask as a consequence of the table being filled (NROW-by-NCOL)

In the above specifications:

- 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)

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.

There is an example automated tradeoff in the optimization tutorial chapter, "Tutorial: Optimization and Automated Tradeoff" in the Getting Started documentation.

Using Automated Tradeoff

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 (defined 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 Optimizations" on page 6-4.

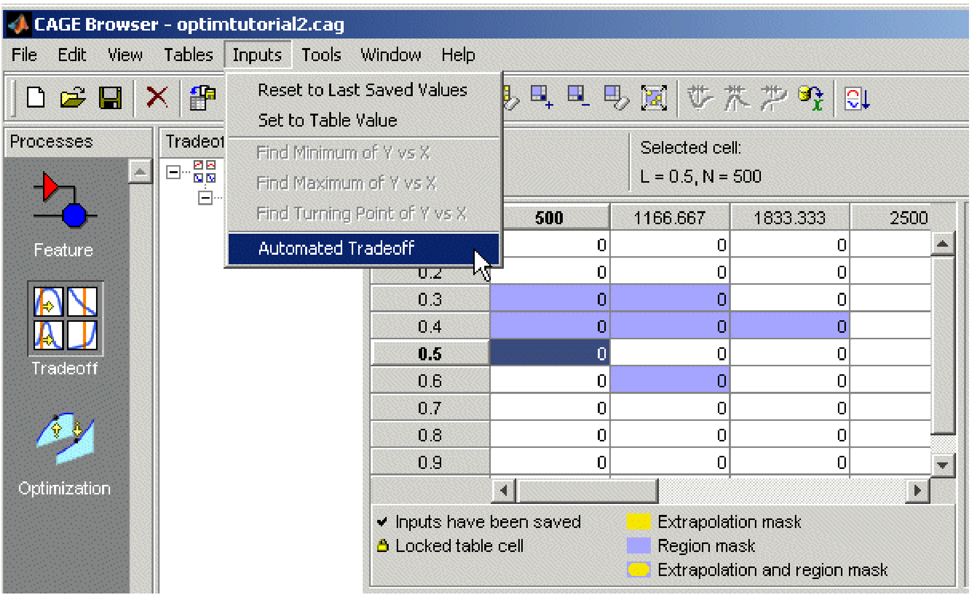
   The set of cells in the region you have selected becomes the operating point set for the optimization. The cell/region breakpoint values are used to replace the fixed variable values in the selected optimization. Note that the existing fixed variable values are reset to their previous state at the end 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 fixed variables; otherwise the set points are used.

4. The optimization is run as if you were clicking **Run** from the Optimization view. See "Running Optimizations" on page 6-22.

   Results are placed in the tradeoff object, that is, values for the tables involving the free variables or values for the tables for constraint or objective models. If the routine applied 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 defined region is filled.

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 fill the rest of the table from your optimized automated tradeoff.



Selected cell:
L = 0.5, N = 500

	500	1166.667	1833.333	2500
	0	0	0	
0.2	0	0	0	
0.3	0	0	0	
0.4	0	0	0	
0.5	0	0	0	
0.6	0	0	0	
0.7	0	0	0	
0.8	0	0	0	
0.9	0	0	0	

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.

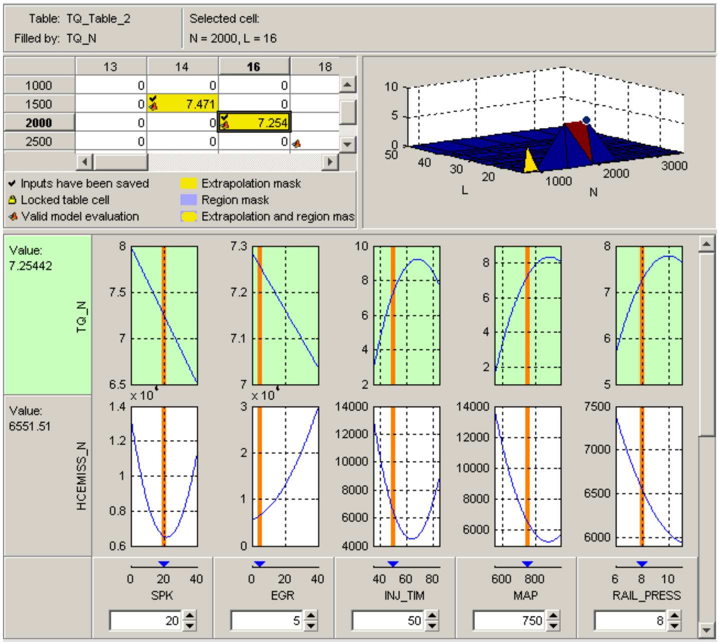


Table: TQ_Table_2
Filled by: TQ_N

Selected cell:
N = 2000, L = 16

	13	14	16	18
1000	0	0	0	
1500	0	7.471	0	
2000	0	0	7.254	
2500	0	0	0	

✓ Inputs have been saved
Locked table cell
Valid model evaluation

Extrapolation mask
Region mask
Extrapolation and region mas

TQ_N
Value:
7.25442

HCEMISS_N
Value:
6551.51

SPK 20
EGR 5
INJ_TIM 50
MAP 750
RAIL_PRESS 8

User-Defined Optimization

User-defined optimizations are described in the following sections:

- “Implementing Your Optimization Algorithm in CAGE” on page 6-50 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 and multiobjective optimization (`foptcon` and `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 6-53

  examines the coding involved in implementing an external optimizer in a CAGE optimization M-file
- "Checking User-Defined Optimizations into CAGE" on page 6-55 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 6-57 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 6-59.

Note Be careful not to overwrite the worked example and template files 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 optimization function in the optimization tutorial. See "Worked Example Optimization" in the Getting Started documentation.

About the Worked Example Optimization Algorithm

`mbcweoptimizer` is an example of a user-specified optimization that solves the following problem:

`max TQ` over (`AFR, SPK`).

- `[bestafr, bestspk] = mbcweoptimizer(TQ)` finds a maximum (`bestafr`, `bestspk`) to the function TQ.

  `TQ` must be a function (or a function handle) that depends on [AFR, SPK, any other variables] only. The function must depend on the variables in that order. This routine does no variable matching.

- `[bestafr, bestspk]=mbcweoptimizer(TQ, afrrng, spkrng)` finds 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)` finds 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 specified, the default grid resolution is 25.

- `[bestafr, bestspk]=mbcweoptimizer(TQ, afrrng, spkrng, res, optimstore)` finds a maximum (`bestafr`,`bestspk`) to the function `TQ` within a CAGE optimization function.

  `optimstore` is passed to this function when it is called from the `Evaluate` section subroutine of your optimization function. In this case, `TQ` must be a function handle that takes the inputs `AFR`, `SPK`, and `OPTIMSTORE`, in that order. Any other inputs for the TQ model will be set by CAGE.

- `[bestafr, bestspk] = mbcweoptimizer(TQ, afrrng, spkrng, res, optimstore, P1, P2, ...)` passes extra scalar arguments (in order) to the torque model when it is evaluated. In this case, the `optimstore` input is ignored.

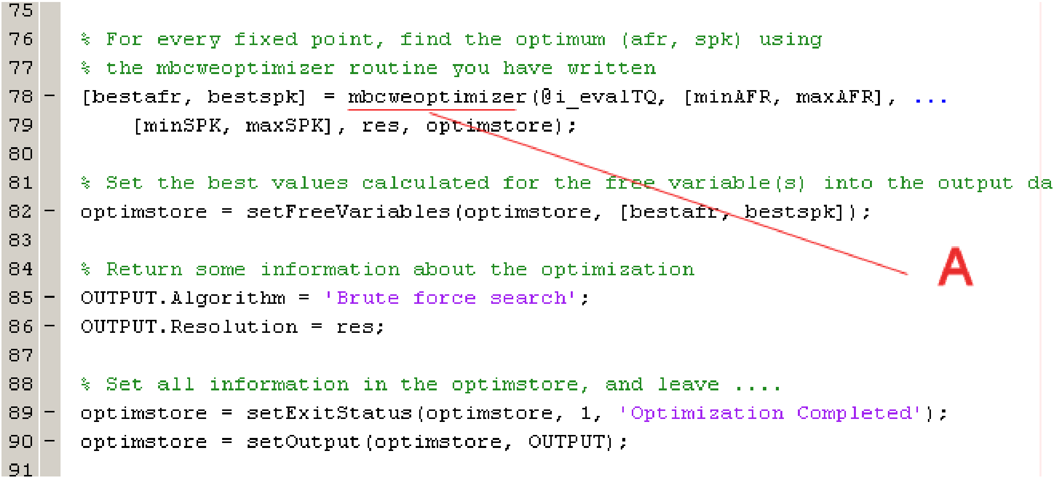
The Structure of the Worked Example

The best way to understand how to implement an external optimizer in a CAGE optimization function is to study the details of the example.

- To view the whole worked example M-file, at the command line, type

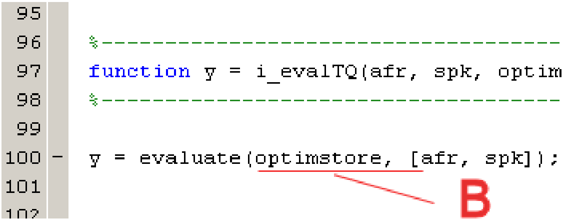
```
edit mbcOSworkedexample
```

The following code section is taken from the Evaluate section of the worked example file as an example.



```

% For every fixed point, find the optimum (afr, spk) using
% the mbcweoptimizer routine you have written
[bestafr, bestspk] = mbcweoptimizer(@i_evalTQ, [minAFR, maxAFR], ...
    [minSPK, maxSPK], res, optimstore);

% Set the best values calculated for the free variable(s) into the output data set
optimstore = setFreeVariables(optimstore, [bestafr, bestspk]);

% Return some information about the optimization
OUTPUT.Algorithm = 'Brute force search';
OUTPUT.Resolution = res;

% Set all information in the optimstore, and leave ....
optimstore = setExitStatus(optimstore, 1, 'Optimization Completed');
optimstore = setOutput(optimstore, OUTPUT);

```

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 algorithm external to the optimization function. As with functions in the Optimization Toolbox, 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.



```
%-----------------------------------------------------------------
function y = i_evalTQ(afr, spk, optimstore)
%-----------------------------------------------------------------

y = evaluate(optimstore, [afr, spk]);
```

The first two inputs to this function are the torque (in this case) model inputs. The final input is the `optimstore` object, where information about the optimization is stored. 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.

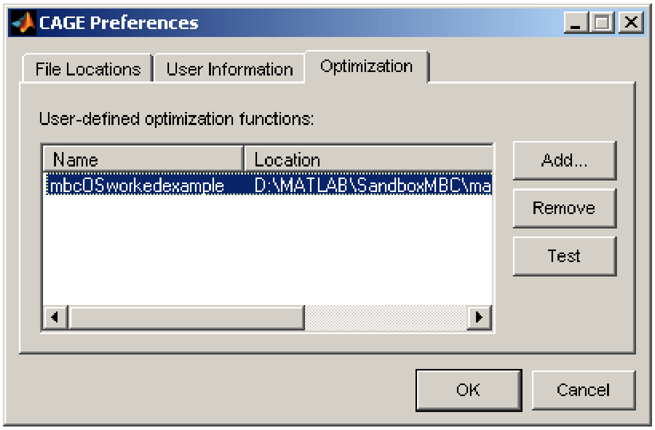
Checking User-Defined Optimizations into CAGE

When you have modified the template to create your own optimization function, you must check it into the Model-Based Calibration Toolbox 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 6-4.

To check a user-defined optimization into CAGE,

1 Select **File -> Preferences**.

2 Click the **Optimization** tab and click **Add...** to browse to your M-file. Select the file and click **Open**. This registers the optimization function with CAGE. You need to do this when you customize your own optimizations.



User-defined optimization functions:

Name	Location
mbcOSworkedexample	D:\MATLAB\SandboxMBC\ma

The example shows the worked example function, which is already registered with CAGE for use in the optimization tutorial.

3 You can click **Test** to check that the optimization 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 file 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 and return to the CAGE browser.

   Registered optimizations appear in the **Optimization Wizard** when you set up a new optimization.

Optimization Function Reference

Following are the reference pages for all the functions you can use in user-defined optimizations. See the following tables for a list of available functions by category:

- “Methods of cgoptimoptions” on page 6-57
- “Methods of cgoptimstore” on page 6-59

Methods of cgoptimoptions

You use these functions to set up all your optimization settings in the `Options` section of the file. 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 a free variable to the optimization
addLinearConstraint	Add a linear constraint to the optimization
addModelConstraint	Add a model constraint to the optimization
addObjective	Add an objective to the optimization
addOperatingPointSet	Add an operating point set to the optimization
addParameter	Add a parameter to the optimization

getConstraints	Return information about all optimization constraints
getConstraintsMode	Return the current usage of constraints
getDescription	Get the current description for the optimization function
getEnabled	Get the current enabled status for the optimization
getFreeVariables	Return the optimization free variable labels
getFreeVariablesMode	Return the current usage of free variables
getLinearConstraints	Get linear constraint placeholder information
getModelConstraints	Get model constraint placeholder information
getName	Get the current name label for the optimization function
getNonlcon	Get nonlinear constraint information
getObjectives	Return information about the optimization objectives
getObjectivesMode	Return the current usage of objective functions
getOperatingPointSets	Return information about the optimization operating point sets
getOperatingPointsMode	Return the current usage of operating point sets
getParameters	Return information about the optimization parameters
getRunInterfaceVersion	Get the preferred interface to provide the evaluation function

removeConstraint	Remove a constraint from the optimization
removeFreeVariable	Remove a free variable from the optimization
removeObjective	Remove an objective from the optimization
removeOperatingPointSet	Remove an operating point set from the optimization
removeParameter	Remove a parameter from the optimization
setConstraintsMode	Set how the optimization constraints are to be used
setDescription	Provide a description for the optimization function
setEnabled	Set the enabled status for this optimization function
setFreeVariablesMode	Set how the optimization free variables are used
setName	Provide a name label for an optimization function
setObjectivesMode	Set how the optimization objective functions are used
setOperatingPointsMode	Set how the optimization operating point sets are used
setRunInterfaceVersion	Get the preferred interface to provide the evaluation function

Methods of cgoptimstore

The following methods are available:

evaluate	Evaluate optimization objectives and constraints
evaluateNonlcon	Evaluate optimization nonlinear constraints
evaluateObjective	Evaluate optimization objectives
get	Get optimization properties
getA	Get the linear inequality constraint matrix.
getB	Get the linear inequality constraint target values.
getDataset	Retrieve data from a data set
getFreeVariables	Get the optimal values of the free variables
getInitFreeVal	Get the initial free values for the optimization
getLB	Get the free variable lower bounds
getLcon	Return the linear constraint labels
getNumNonlcon	Return the number of nonlinear constraints per label
getNumNonlconLabels	Return the number of nonlinear constraint labels
getNumObjectiveLabels	Return the number of objective labels
getNumObjectives	Return the number of objectives per label
getNumRowsInDataset	Get the number of rows in an optimization data set
getObjectives	Return the objective labels for the optimization
getObjectiveType	Return the objective type

getOptimOptions	Retrieve the optimization options object
getOutputInfo	Get output information for the optimization
getParam	Get optimization parameter
getStopState	Current stop state for optimization
getUB	Get the free variable upper bounds
gridEvaluate	Grid evaluation of optimization objectives and constraints
gridPevEvaluate	Grid evaluation of prediction error variance (PEV)
isScalarFreeVariables	Return whether all the free variables are scalars
nEvaluate	Natural evaluation of optimization objectives and constraints
nEvaluateNonlcon	Natural evaluation of optimization nonlinear constraints
nEvaluateObjective	Natural evaluation of optimization objectives
optimset	Create/alter optimization OPTIONS structure
pevEvaluate	Evaluate prediction error variance (PEV)
setExitStatus	Set exit status information for the optimization
setFreeVariables	Set the optimal values of the free variables
setOutput	Set diagnostic information for the optimization

setOutputInfo	Set output information for the optimization
setStopState	Set current stop state for optimization

Functions — Alphabetical List

addFreeVariable
addLinearConstraint
addModelConstraint
addObjective
addOperatingPointSet
addParameter
evaluate
evaluateNonlcon
evaluateObjective
get
getA
getB
getConstraints
getConstraintsMode
getDataset
getDescription
getEnabled
getFreeVariables
getFreeVariables
getFreeVariablesMode
getInitFreeVal
getLB
getLcon
getLinearConstraints
getModelConstraints
getName
getNonlcon
getNumNonlcon
getNumNonlconLabels
getNumObjectiveLabels
getNumObjectives
getNumRowsInDataset
getObjectives
getObjectives
getObjectivesMode

getObjectiveType
getOperatingPointSets
getOperatingPointsMode
getOptimOptions
getOutputInfo
getParam
getParameters
getRunInterfaceVersion
getStopState
getUB
gridEvaluate
gridPevEvaluate
isScalarFreeVariables
nEvaluate
nEvaluateNonlcon
nEvaluateObjective
optimset
pevEvaluate
removeConstraint
removeFreeVariable
removeObjective
removeOperatingPointSet
removeParameter
setConstraintsMode
setDescription
setEnabled
setExitStatus
setFreeVariables
setFreeVariablesMode
setName
setObjectivesMode
setOperatingPointsMode
setOutput
setOutputInfo
setRunInterfaceVersion
setStopState

Purpose

Add a model constraint to the 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 constraint where a user-defined 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

addLinearConstraint

Purpose

Add a linear constraint to the 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 ith 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

Purpose

Add a free variable to the 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`

addObjective

Purpose

Add an objective to the 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

Purpose

Add an operating point set to the optimization

Syntax

```
options = addOperatingPointSet(options, label, vars)
```

Description

A method of `cgoptimoptions`. 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 a parameter to the optimization

Syntax

```
options = addParameter(options, label, typestr, value)
```

Description

A method of `cgoptimoptions`. Adds a parameter to the optimization. The string `label` is used to refer to the parameter. The string `typestr` takes one of ’`number`’, ’`list`’, or ’`boolean`’. A default value for the parameter must be supplied in `value`. The form of `value` must be one of the following:

typestr	Value
`'number'`	Scalar, real number
`'list'`	Cell array of strings, one for each list member
`'boolean'`	True or false

`options = addParameter(options, label, typestr, value, displayName)` allows a more descriptive label to be used 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

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`

getLinearConstraints

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`

Purpose

Return the 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`

getDescription

Purpose Get the current description for the optimization function

Syntax

```
desc = getDescription(options)
```

Description A method of `cgoptimoptions`. Returns the description, `desc`, of the user-defined optimization function.

See Also `setDescription`

Purpose

Get the current enabled status for the 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`

getFreeVariables

Purpose

Return the 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 the 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`

Purpose Get the current name label for the 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`

Purpose Return information about the 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 the current usage of objective functions

Syntax

```
mode = getObjectivesMode(options)
```

Description A method of `cgoptimoptions`. Returns a string describing how the optimization makes objectives available to the user. `mode` will be one of ‘`multiple'`, ‘any’, or ‘`fixed'`.

See Also `setObjectivesMode`

Purpose

Return information about the optimization 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 fields, `label` and `vars`. See the help for `addOperatingPointSet` for more information on these fields.

See Also

`addOperatingPointSet`, `setOperatingPointsMode`, `getOperatingPointsMode`

getOperatingPointsMode

Purpose Return the current usage of operating point sets

Syntax

```
mode=getOperatingPointsMode(options)
```

Description A method of `cgoptimoptions`. Returns a string describing how the optimization makes operating point sets available to the user. `mode` will be one of ‘`default'`, ‘`fixed'`, or ‘`any'`.

See Also `setOperatingPointsMode`

Purpose

Return information about the 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`

Purpose

Set how the 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`

Purpose

Provide a description for the 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 the enabled status for this 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`

Purpose

Set how the 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`

setName

Purpose

Provide a name label for an 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 the optimization objective 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-defined optimization function.

When `modestr = 'multiple'`, the user is only allowed to run the optimization if he or she has defined two or more objectives.

See Also

`getObjectivesMode`, `addObjective`

setOperatingPointsMode

Purpose

Set how the 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`

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`

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

Get the preferred interface to provide the 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 Version that will be emulated when the optimization function's evaluate option is called. If `ver` is set to 2, the interface provided by the Model-Based Calibration Toolbox Version 2 will be activated. If `ver` is set to 3, the new interface that the Model-Based Calibration Toolbox Version 3 defines will be used.

See Also

`setRunInterfaceVersion`

Purpose

Remove a constraint from the 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 a free variable from the 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 an objective from the 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 objective referred to by the string `label`.

See Also

`getObjectives`, `addObjective`

Purpose

Remove an operating point set from the optimization

Syntax

```
obj = removeOperatingPointSet(obj, sLabel)
```

Description

Remove an operating point set from the optimization. A method of `cgoptimoptions`.

`options = removeOperatingPointSet(options, label)` removes the placeholder for the operating point set referred to by the string `label`.

See Also

`getOperatingPointSets`, `addOperatingPointSet`

removeParameter

Purpose

Remove a parameter from the optimization

Syntax

```
obj = removeParameter(obj, sLabel)
```

Description

Remove a parameter from the optimization. A method of `cgoptimoptions`.

Removes the placeholder for the parameter referred to by the string `label`.

See Also

`getParameters`, `addParameter`

Purpose

Get the preferred interface to provide the 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 Calibration Toolbox Version that will be emulated when the optimization function's evaluate option is called. If `ver` is set to 2, the interface provided by the Model-Based Calibration Toolbox Version 2 will be activated. If `ver` is set to 3, the new interface that the Model-Based Calibration Toolbox Version 3 defines will be used.

The interface version that the current version of Model-Based Calibration Toolbox runs is superior 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 files that fail to work with the newer interface.

See Also

`getRunInterfaceVersion`

evaluate

Purpose

Evaluate optimization objectives and constraints

Syntax

```
Y = evaluate(optimstore, X)
```

Description

A method of `cgoptimstore`.

Evaluate optimization objectives 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.

Examples

```
Y = evaluate(optimstore, X, itemnames)
```

evaluates the objectives and constraints specified 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 NItems 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 specified objectives and constraints at the operating points in the data set specified 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 specified 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 indicies in the range [1 `NumRowsInDataset`]. Y is a (`Nrows-by-NItems`) matrix.

See Also

`nEvaluate`, `pevEvaluate`

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 thename of a property of OPTIMSTORE and each field contains the description of that property.

This method is obsolete. Please use the GETXXX methods instead.

See Also

See also cgoptimstore/GETXXX, for example `getA`, `getB`, etc.

Purpose

Retrieve data from a 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 labelled `'myDS'`, `V(:, 1)` is the data for the variable labelled `'speed'`, `V(:, 2)` is the data for the variable labelled `'afr'`.

See Also

`addOperatingPointSet`

getNumRowsInDataset

Purpose

Get the number of rows in an optimization data set

Syntax

```
npts = getNumrowsInDataset(optimstore, datasetName)
```

Description

Returns the number of rows in the named data set. A method of `cgoptimstore`.

Purpose

Grid evaluation of optimization 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 specified in the cell array objconname as described above.

Y = gridEvaluate(optimstore, X, objconname, datasetname) evaluates all the objectives and constraints 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 constraints 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`

setFreeVariables

Purpose

Set the optimal values of the 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`

Purpose

Set output information for the 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 flag indicating why the optimization has terminated. `exitflag` > 0 implies that the optimization has terminated successfully.
- `termmsg`: Message that is displayed at termination of algorithm. Normally used for error messages.
- `output`: Structure of algorithm statistics for the optimization.

This method is obsolete. Use `cgoptimstore/setExitStatus` and `cgoptimstore/setOutput` instead.

See Also

`setExitStatus`, `setOutput`

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. Note that the evaluation of `Y` is scaled onto [-1 1].

`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

`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 the objectives are returned in `Y`, which is of size `(NPoints-by-NItems)` where `NItems` is the number of objectives in the optimization. Note that the evaluation of `Y` is scaled onto [-1 1].

`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`

getA

Purpose

Get the 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.

See Also

`getB`

Purpose

Get the 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.

See Also

`getA`

getFreeVariables

Purpose

Get the optimal values of the free variables

Syntax

```
data = getFreeVariables(obj)
```

Description

Get the optimal values of the free variables. A method of `cgoptimstore`.

`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`

Purpose

Get the initial free values for the 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`

getLB

Purpose

Get the 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`

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”.

getLcon

Purpose

Return the 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

Return the 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 labelled constraint. For example, consider 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`. `NCON` will return `[1 r]`, where `r` is the number of points in `S`.

`ncon = getNumNonlcon(optimstore, conLabels)` returns the number of constraints type for the defined constraints.

See Also

`getConstraints`, `getNumNonlconLabels`

getNumNonlconLabels

Purpose

Return the 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 the 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 the number of objectives per label

Syntax

```
nobj = getNumObjectives(optimstore)
nobj = getNumObjectives(optimstore, objlabels)
```

Description

Return the number of objectives per label. A method of `cgoptimstore`.

`nobj = getNumObjectives(optimstore)` returns the number of objectives that will be returned from 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 defined objective labels.

See Also

`getObjectives`; `getObjectiveType`

Purpose

Return the objective labels for the 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`

getObjectiveType

Purpose

Return the 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`

Purpose

Retrieve the optimization options object

Syntax

```
options = getOptimOptions(optimstore)
```

Description

A method of `cgoptimstore`. Returns the optimization configuration object. Information about the optimization set up can be retrieved from this object.

Purpose

Get output information for the 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 structure of algorithm-specific information in `output`. For `output` to be non-empty, the user must create it in their algorithm. See the worked example and tutorial for more information on how to create `output` structures.

Purpose

Get the 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`

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

Current stop state for optimization

Syntax

```
stop= getStopState(opt)
```

Description

A method of `cgoptimstore`. `stop= getStopState(optimstore)` returns the current stop state for the optimization. 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`

isScalarFreeVariables

Purpose

Return whether all the 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.

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 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 = nEvaluate(optimstore, x, itemNames) evaluates the raw values of the objectives and constraints specified 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 NItems is the number of objectives and constraints listed in itemNames.

Y = nEvaluate(optimstore, x, itemNames, datasetName) evaluates the specified objectives and constraints at the operating points in the data set specified by the string datasetName.

Y = nEvaluate(optimstore, x, itemNames, datasetName, rowind) evaluates the specified 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

nEvaluateNonlcon

Purpose

Natural evaluation of optimization nonlinear constraints

Syntax

```
y = nEvaluateNonlcon(optimstore, x)
Y = nEvaluateNonlcon(optimstore, x, itemNames)
```

Description

Natural evaluation of optimization nonlinear 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 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 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 specified values. Any parameters specified 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 parameter 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 specified 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 objectives/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 specified by the string `datasetname`.

```
Y = pevevaluate(optimstore, X, objconname, datasetname, rowind)
```

returns PEV values of zero for the specified 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`

setExitStatus

Purpose

Set exit status information for the 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. Note that `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 file `mbcOSworkedexample`, used in the optimization tutorial "Worked Example Optimization".

Purpose

Set diagnostic information for the 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”.

setStopState

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 optimization script must do this.

See Also

`getStopState`

7

Data Sets

This section includes the following topics:

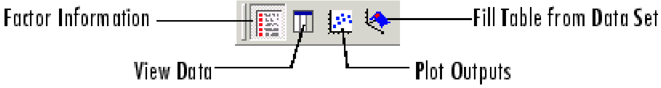
Data Sets Views

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 fill 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**

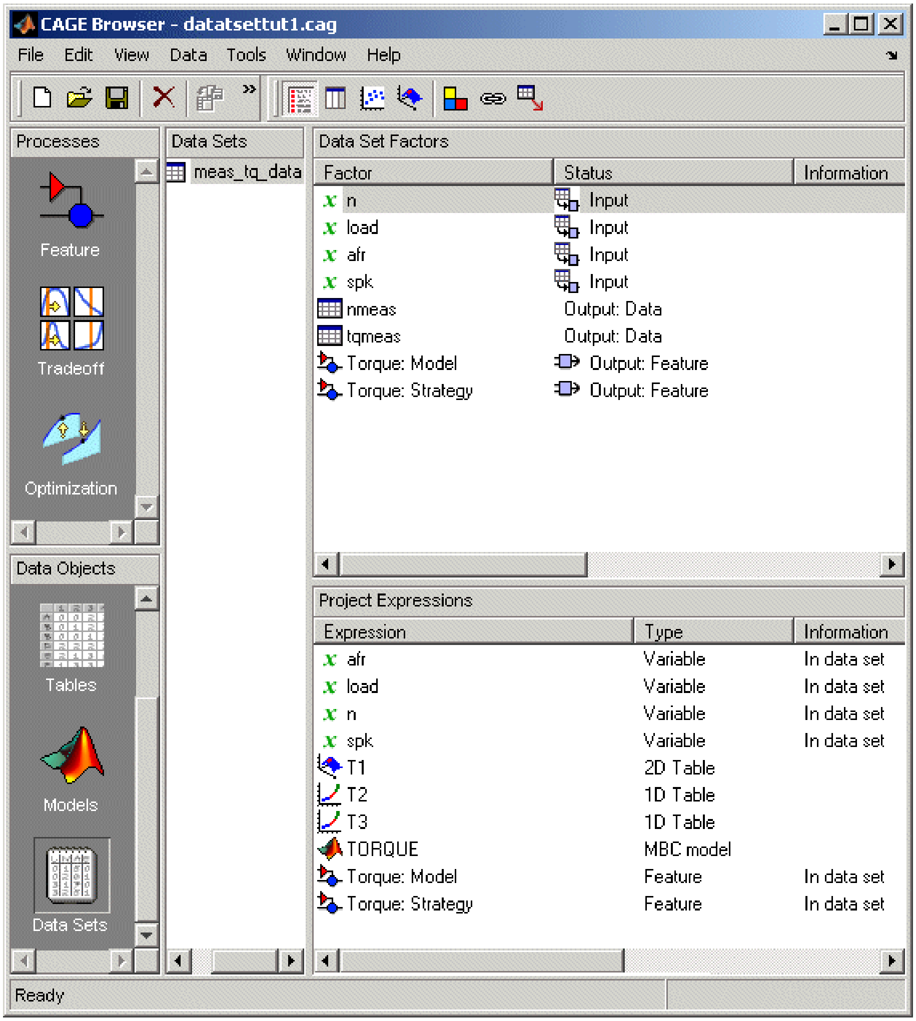
  Displays the data in a table. Individual entries can be altered. Columns of data can be assigned to CAGE expressions.

- **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 fill tables by reference to experimental data.



Factor	Status	Information
n	Input	
load	Input	
afr	Input	
spk	Input	
nmeas	Output: Data	
tqmeas	Output: Data	
Torque: Model	Output: Feature	
Torque: Strategy	Output: Feature	

Expression	Type	Information
afr	Variable	In data set
load	Variable	In data set
n	Variable	In data set
spk	Variable	In data set
T1	2D Table	
T2	1D Table	
T3	1D Table	
TORQUE	MBC model	
Torque: Model	Feature	In data set
Torque: Strategy	Feature	In data set

Setting 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 three ways:

- Import experimental data. (See “Importing Experimental Data” on page 7-4.)
- Import the values from a table in your CAGE session. (See “Importing Data from a Table in Your Session” on page 7-6.)
- Specify the factors and their values manually. (See “Specifying the Factors Manually” on page 7-7.)

The next sections describe each of these in detail.

Importing Experimental Data

You can import experimental data to a data set, either to validate a calibration or to use it as the basis for a calibration.

You can import data that is stored in the following formats:

- Microsoft Excel spreadsheets
- Comma-separated value files
- MAT-files

Importing from Excel or Comma-Separated Value

When you import data from either a Microsoft Excel spreadsheet or from a comma-separated value file, you must ensure that the data is organized in the following manner:

- The first column can either be row markers (text) or entries (numbers).
- The first row can either be column headers (text) or entries (numbers).
- All the other row and column entries must be numbers.

Importing from MAT-files

When you import from a MAT-file, you must ensure that the file contains numbers only, that is, a double array.

To import experimental data,

1 Select **File -> Import -> Data**.

2 In the file browser, select the correct file to import.

   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.

   ✕

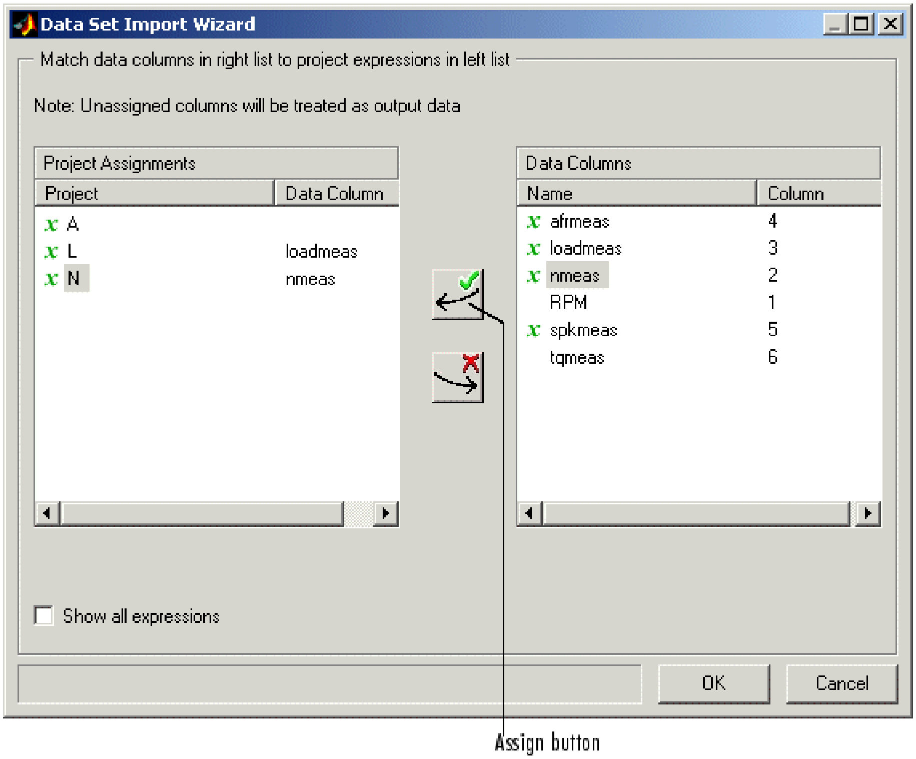
4 Click **Next**.

   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 corresponding data column in the **Data Column**, then click the assign button, shown.

   ↩✓

6 Repeat step 5 until you are satisfied that you have associated all the variables and data columns. Any unassigned data columns are treated as output factors.



Data Set Import Wizard

Match data columns in right list to project expressions in left list

Note: Unassigned columns will be treated as output data

Project Assignments

Project	Data Column
x A	
x L	loadmeas
x N	nmeas

Data Columns

Name	Column
x afrmeas	4
x loadmeas	3
x nmeas	2
RPM	1
x spkmeas	5
tqmeas	6

□ Show all expressions

Assign button

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.

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 7-7.)

4 Build the grid. (See "Manually Setting Values of the Input Variables" on page 7-9.)

Once you have completed these steps you can view the data set.

This section describes

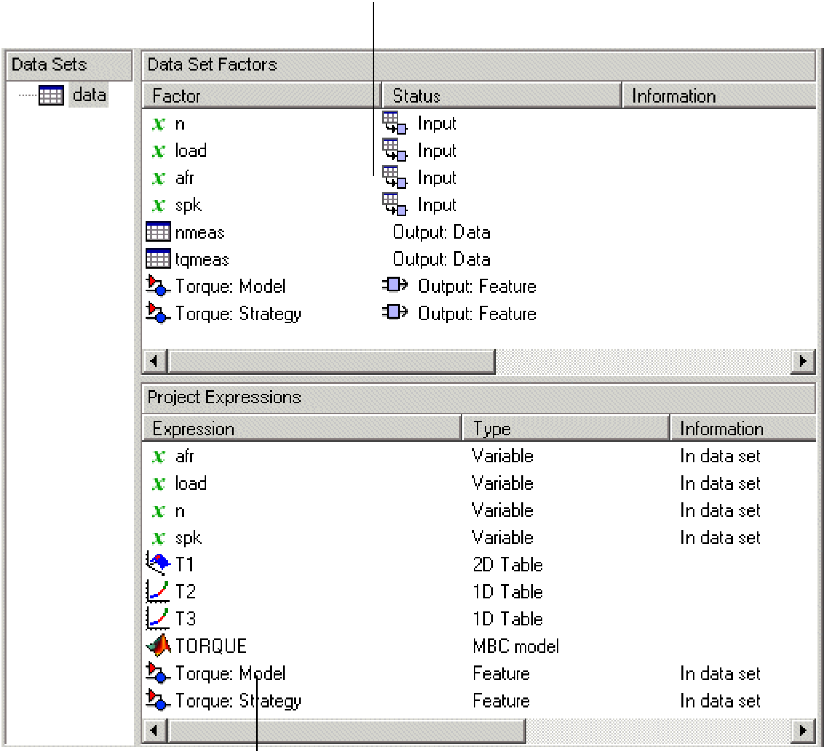
- "Selecting the Factors" on page 7-7
- "Manually Setting Values of the Input Variables" on page 7-9

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.

Factors in the current data set



Data Sets

- data

Data Set Factors

Factor	Status	Information
n	Input	
load	Input	
afr	Input	
spk	Input	
nmeas	Output: Data	
tqmeas	Output: Data	
Torque: Model	Output: Feature	
Torque: Strategy	Output: Feature	

Project Expressions

Expression	Type	Information
afr	Variable	In data set
load	Variable	In data set
n	Variable	In data set
spk	Variable	In data set
T1	2D Table	
T2	1D Table	
T3	1D Table	
TORQUE	MBC model	
Torque: Model	Feature	In data set
Torque: Strategy	Feature	In data set

Factors in the current project

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.

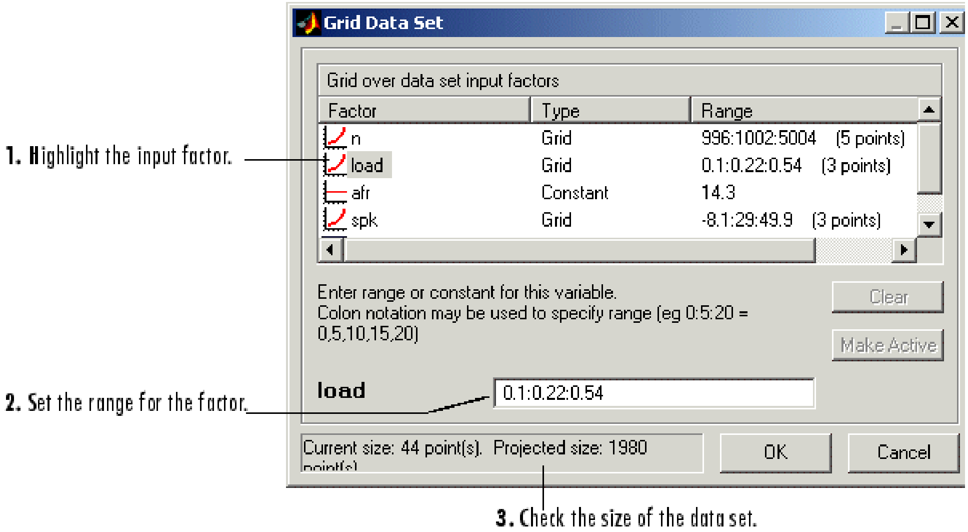
3 Set the grid for the factor.

   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 define a grid for.

6 When you have set the grids for all the factors, click **OK**.



Grid over data set input factors

Factor	Type	Range
n	Grid	996:1002:5004 (5 points)
load	Grid	0.1:0.22:0.54 (3 points)
afr	Constant	14.3
spk	Grid	-8.1:29:49.9 (3 points)

Enter range or constant for this variable.
Colon notation may be used to specify range (eg 0:5:20 = 0,5,10,15,20)

load 0.1:0.22:0.54

Current size: 44 point(s). Projected size: 1980 point(s)

1. Highlight the input factor.

2. Set the range for the factor.

3. Check the size of the data set.

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**.

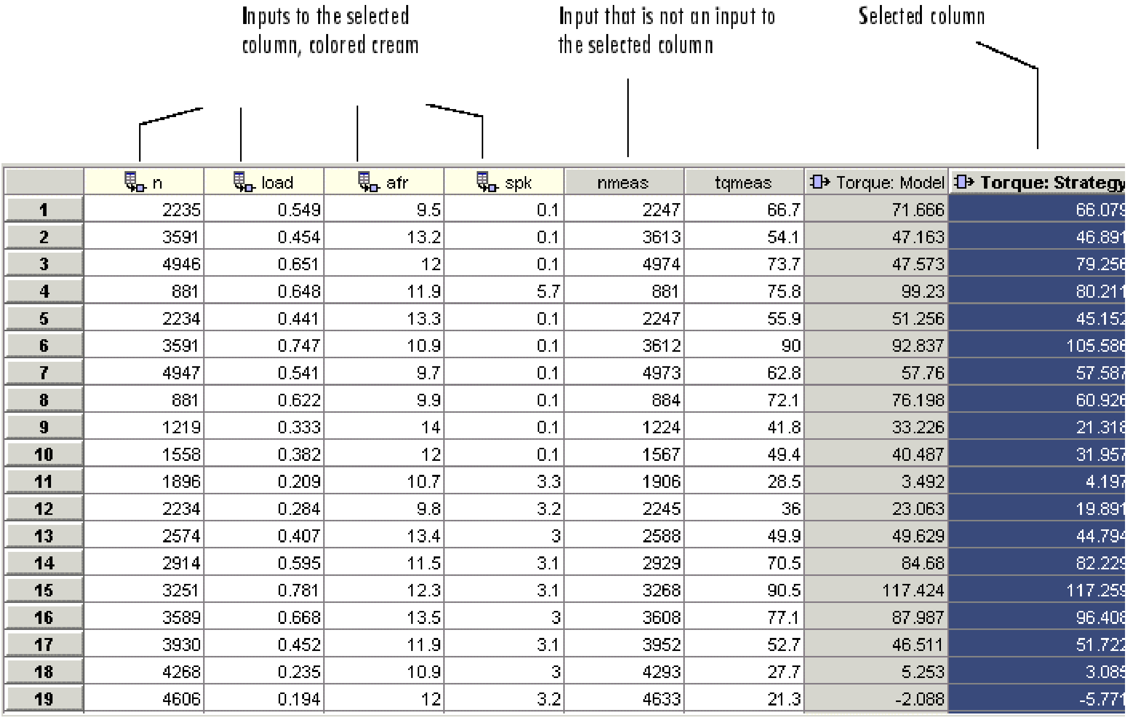
2 Select **Create Error** from the right-click menu on either column head.

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 tables included in the data set. You can set the grid by importing experimental or table data, or by using the Build Grid toolbar button (▦). See "Setting Up Data Sets" on page 7-4.



Inputs to the selected column, colored cream

Input that is not an input to the selected column

Selected column

	n	load	afr	spk	nmeas	tqmeas	Torque: Model	Torque: Strategy
1	2235	0.549	9.5	0.1	2247	66.7	71.666	66.079
2	3591	0.454	13.2	0.1	3613	54.1	47.163	46.891
3	4946	0.651	12	0.1	4974	73.7	47.573	79.256
4	881	0.648	11.9	5.7	881	75.8	99.23	80.211
5	2234	0.441	13.3	0.1	2247	55.9	51.256	45.152
6	3591	0.747	10.9	0.1	3612	90	92.837	105.586
7	4947	0.541	9.7	0.1	4973	62.8	57.76	57.587
8	881	0.622	9.9	0.1	884	72.1	76.198	60.926
9	1219	0.333	14	0.1	1224	41.8	33.226	21.318
10	1558	0.382	12	0.1	1567	49.4	40.487	31.957
11	1896	0.209	10.7	3.3	1906	28.5	3.492	4.197
12	2234	0.284	9.8	3.2	2245	36	23.063	19.891
13	2574	0.407	13.4	3	2588	49.9	49.629	44.794
14	2914	0.595	11.5	3.1	2929	70.5	84.68	82.229
15	3251	0.781	12.3	3.1	3268	90.5	117.424	117.259
16	3589	0.668	13.5	3	3608	77.1	87.987	96.408
17	3930	0.452	11.9	3.1	3952	52.7	46.511	51.722
18	4268	0.235	10.9	3	4293	27.7	5.253	3.085
19	4606	0.194	12	3.2	4633	21.3	-2.088	-5.771

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 applications (for example, Excel).

Note You can only edit input values, not output values.

Plotting Outputs

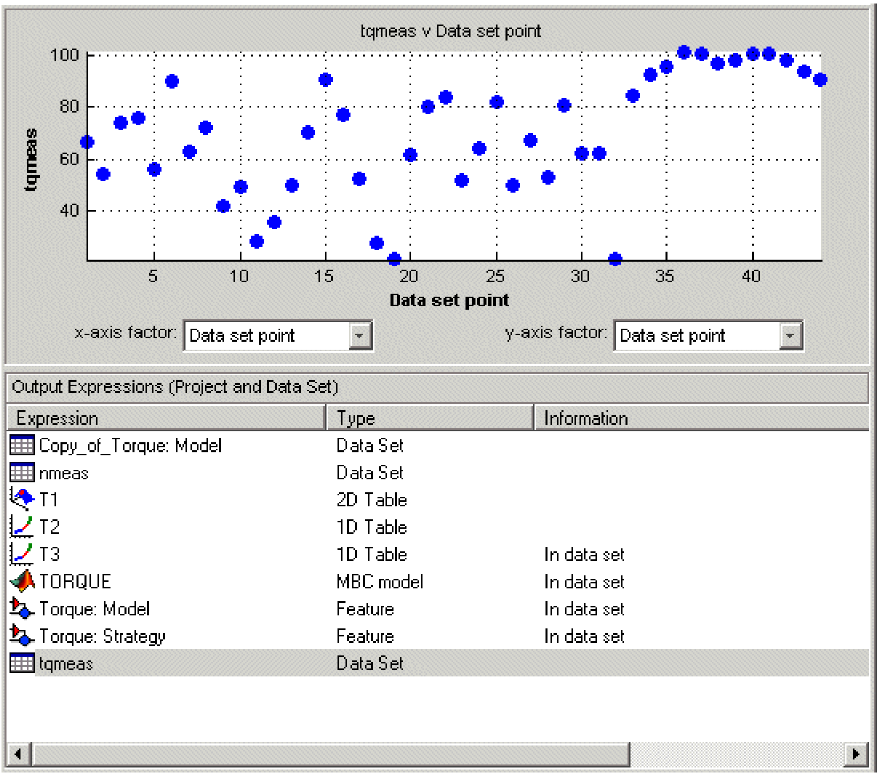
Use this to plot the outputs of your data sets.

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.



x-axis factor: Data set point

y-axis factor: Data set point

Expression	Type	Information
Copy_of_Torque: Model	Data Set	
nmeas	Data Set	
T1	2D Table	
T2	1D Table	
T3	1D Table	In data set
TORQUE	MBC model	In data set
Torque: Model	Feature	In data set
Torque: Strategy	Feature	In data set
tqmeas	Data Set	

3 Use the pop-up menus below the plot to change the factors displayed.

To zoom in on an area of interest,

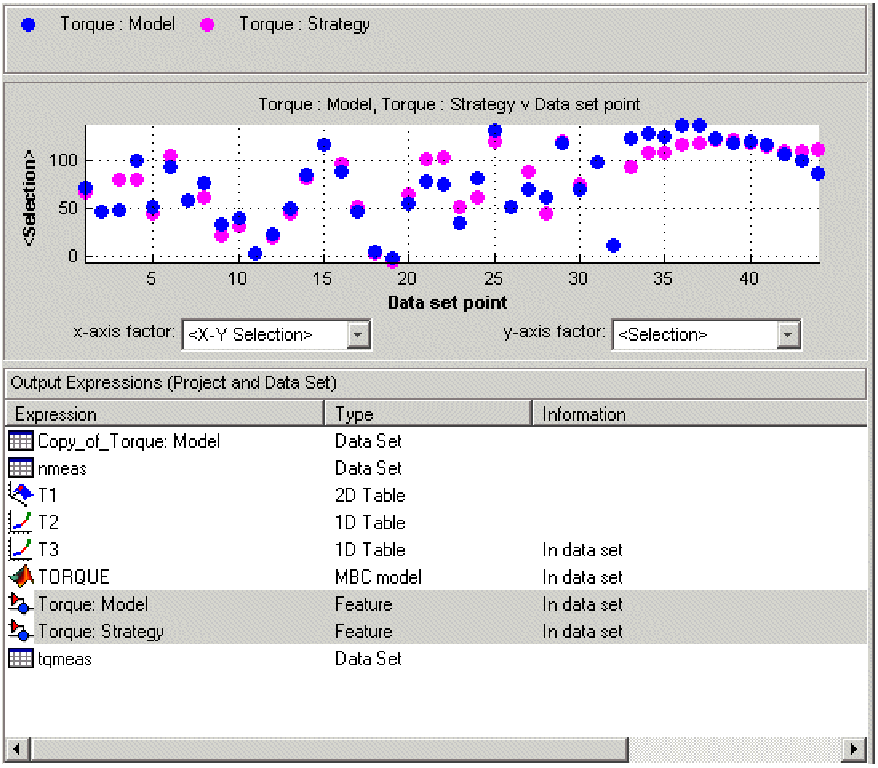
- Press both mouse buttons simultaneously and drag a rectangle; double-click the graph to return to full size.

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.

Multiple Plot Outputs



x-axis factor: <X-Y Selection>

y-axis factor: <Selection>

Output Expressions (Project and Data Set)

Expression	Type	Information
Copy_of_Torque: Model	Data Set	
nmeas	Data Set	
T1	2D Table	
T2	1D Table	
T3	1D Table	In data set
TORQUE	MBC model	In data set
Torque: Model	Feature	In data set
Torque: Strategy	Feature	In data set
tqmeas	Data Set	

When exactly two items are displayed, further plot options are available:

- Plot the first 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.

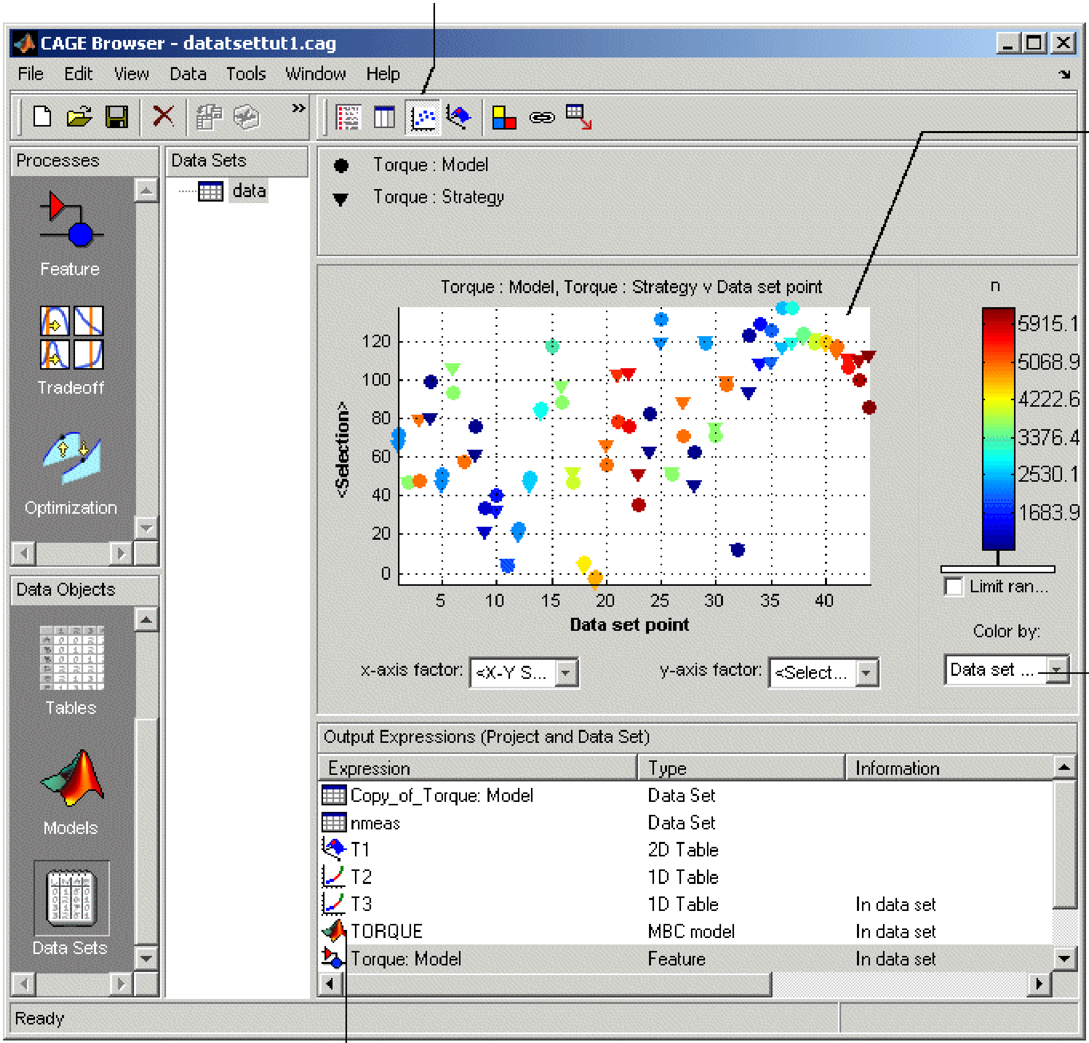
1 Select **View > Plot** or click 📊.

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.

1. Click Plot Outputs.



x-axis factor: <X-Y S...

y-axis factor: <Select...

Color by: Data set ...

Output Expressions (Project and Data Set)

Expression	Type	Information
Copy_of_Torque: Model	Data Set	
nmeas	Data Set	
T1	2D Table	
T2	1D Table	
T3	1D Table	In data set
TORQUE	MBC model	In data set
Torque: Model	Feature	In data set

3. Select **Color by Value** from the right-click menu.

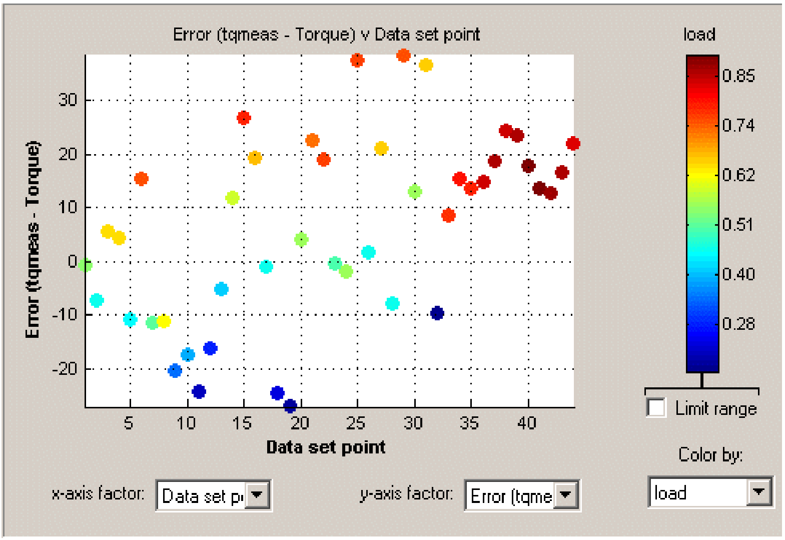
4. Select the correct variable.

2. Select the expression.

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.



□ Limit range

Color by: load

x-axis factor: Data set p

y-axis factor: Error (tqme

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**.

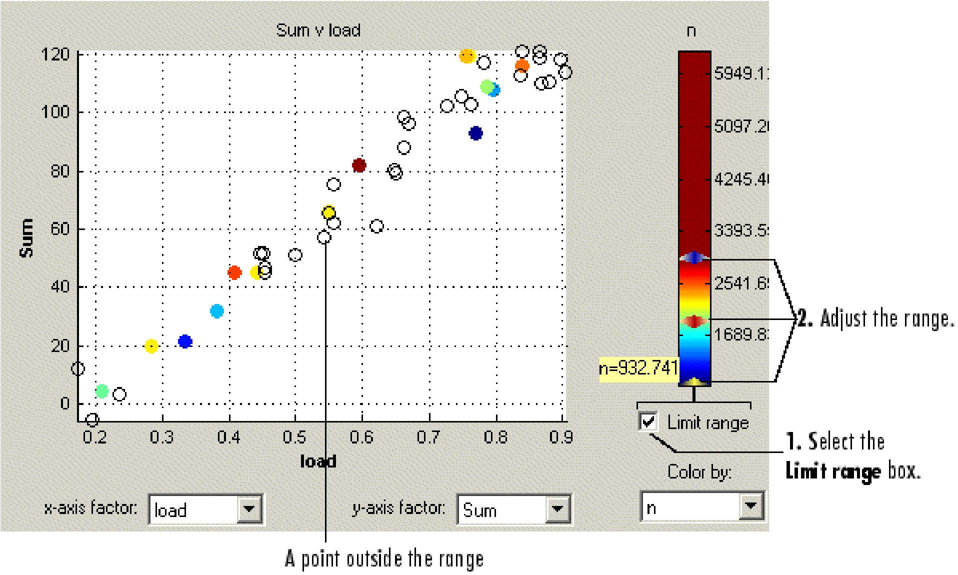
   Three limit markers appear in the color bar. The colors in the color bar are compressed within the limit markers. This increases the range of colors over the range you are interested in (between the limits), making it easier to see the distribution of points.

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 limits are still colored, but only dark red or dark blue, depending on which end of the range they are.
- Select **No Color Outside Limits** to display the points as in the example shown. Points outside the limits are plotted as empty circles.



☑ Limit range

Color by: n

x-axis factor: load

y-axis factor: Sum

2. Adjust the range.

1. Select the **Limit range** box.

A point outside the range

Linking Factors in a Data Set

A factor can be linked to another. The factor then takes on the values of that other factor, overwriting the original values.

For example, you might want to link a variable spark with a model for maximum brake torque (MBT) to evaluate a torque model.

To link two factors,

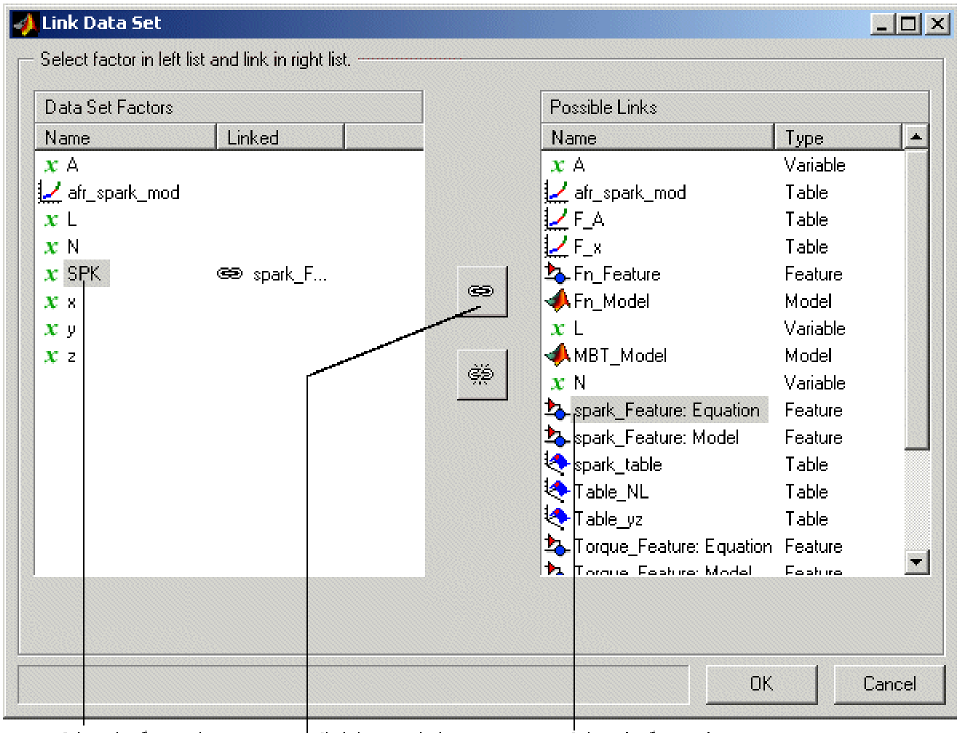
1 Select **Data -> Links**. This opens a dialog box.

2 Select the data set factor that you want to overwrite.

   CAGE generates a list of factors that you could possibly link to the selected factor. (For example, you cannot link to a factor that depends on the selected factor.)

3 Select the factor that you want to link the selected factor with.

4 Click 🔗 to link the two factors.



Select factor in left list and link in right list.

Data Set Factors

Name	Linked
A	
afr_spark_mod	
L	
N	
SPK	spark_F...
x	
y	
z	

Possible Links

Name	Type
A	Variable
afr_spark_mod	Table
F_A	Table
F_x	Table
Fn_Feature	Feature
Fn_Model	Model
L	Variable
MBT_Model	Model
N	Variable
spark_Feature: Equation	Feature
spark_Feature: Model	Feature
spark_table	Table
Table_NL	Table
Table_yz	Table
Torque_Feature: Equation	Feature
Torque_Feature: Model	Feature

2. Select the factor that you want to overwrite.

4. Click here to link the factors.

3. Select the factor that you want to link it with.

CAGE then overwrites the data set factor with the link.

To break a link and return to normal evaluation, click .

Once all the links have been created or broken as you want, click **OK** to exit the dialog.

See also:

- “Setting Up Data Sets” on page 7-4

Assigning Columns of Data

To analyze imported data, you need to assign columns of data to input factors in the CAGE data set.

Data can be imported into a data set from outside CAGE, for example, from an engine test cell. In many cases, this data contains a set of input points (or operating points) and the values of important measurable variables at those points. To compare data like this with models (and/or tables) in a CAGE data set, you have to assign columns of the data to the corresponding input factors in the data set.

To assign data,

1 Select **Data > Assign**.

2 In the dialog box, highlight the column that you want to assign and the variable that you want to assign it to.

3 Click ↩ to assign.

To unassign data,

1 Select **Data > Assign**.

2 In the dialog box, highlight the variable that you want to unassign.

3 Click ↪ to unassign.

Note Assigning data to a CAGE expression overwrites that expression in the data set. This does not affect the expression in the other parts of the CAGE project.

Manipulating Models in Data Set View

A model in a data set can be treated as either an input or an output. This is particularly useful when a model is used as an input to another model and you want to view specific values of the input model. For example, linking a model of MBT Spark to a Spark model allows the evaluation of a TQ model at MBT.

To change a model to an input,

1. Highlight the desired model in either the factor view or the table view.
2. Select **Treat as Input** from the right-click menu.

To revert a model to an output,

1. Highlight the desired model in either the factor view or the table view.
2. Select **Treat as Output** from the right-click menu.

Filling Tables from Experimental Data

Any table in the project whose axes (normalizers) exist as factors in the data set can be filled from imported experimental data (or any data set, such as optimization output).

CAGE extrapolates the values of the experimental data over the range of your table. Then it fills the table by selecting the values of the extrapolation at your breakpoints.

To fill the table with values based on the experimental data,

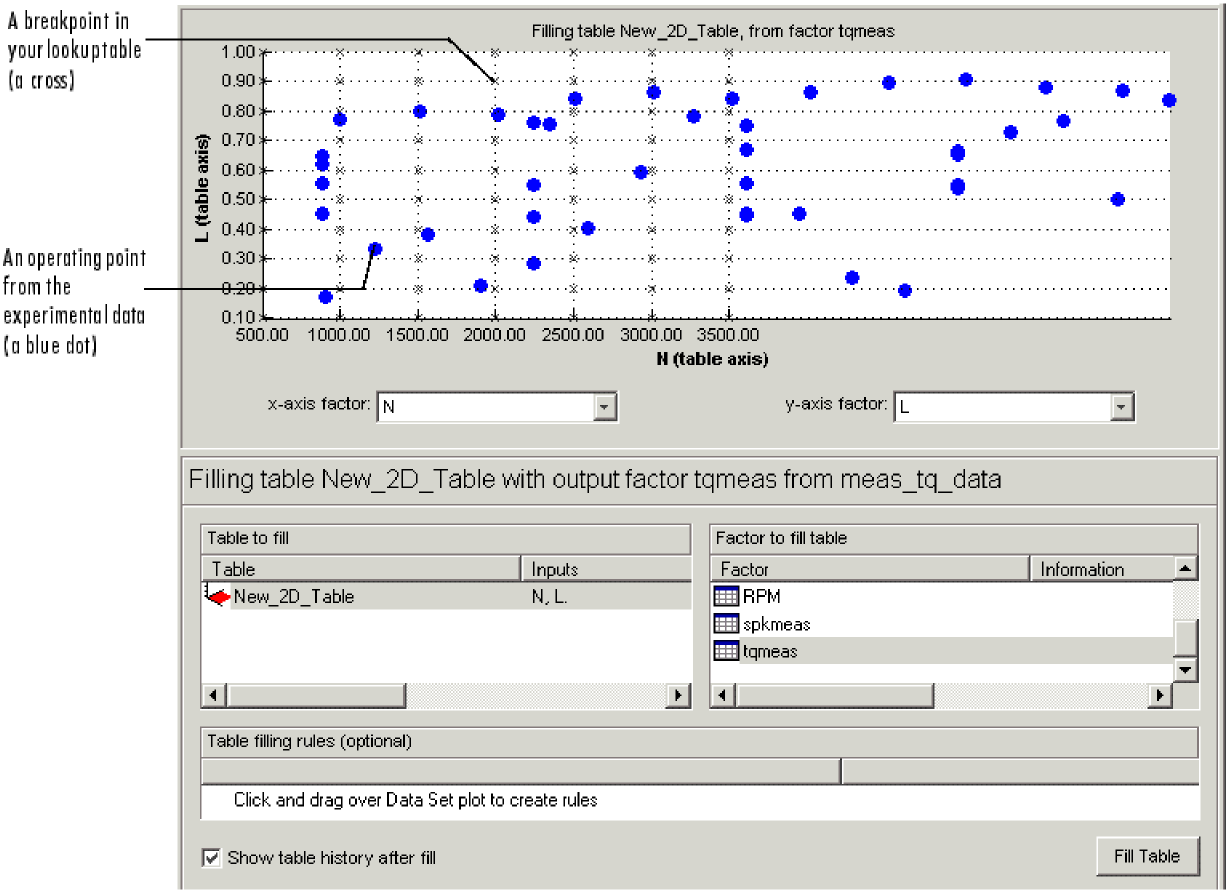
1 To view the **Table Filler** display, click (Fill Table From Data Set) in the toolbar; or select **View > Table Filler**.

   You can use this display to specify the table you want to fill and the factor you want to use to fill it.

2 In the lower pane, select the table from the **Table to fill** list. This is the table that you want to fill.

3 Select the experimental data from the **Factor to fill table** list. This is the data that you want to use to fill the table.

   For example, see the following display.



A breakpoint in your lookuptable (a cross)

An operating point from the experimental data (a blue dot)

x-axis factor: N

y-axis factor: L

Filling table New_2D_Table with output factor tqmeas from meas_tq_data

Table to fill

Table	Inputs
New_2D_Table	N, L.

Factor to fill table

Factor	Information
RPM	
spkmeas	
tqmeas	

Table filling rules (optional)

Click and drag over Data Set plot to create rules

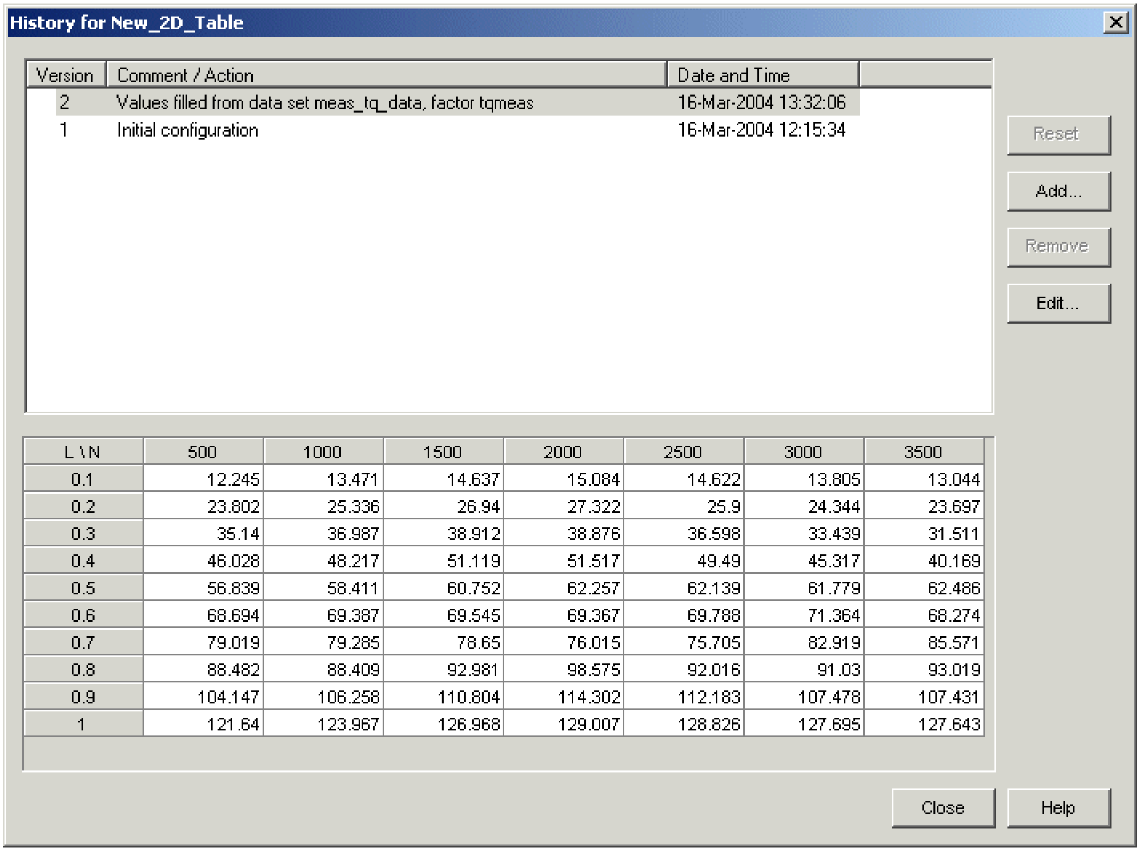
☑ Show table history after fill

The upper pane displays the breakpoints of your table as crosses and the operating points where there is data as blue dots. Data sets display the points in the experimental data, not the values at the breakpoints. You can inspect the spread of the data compared to the breakpoints of your table before you fill the table.

4 To view the table after it is filled, make sure the **Show table history after fill** box at the bottom left is selected. This is selected by default.

5 To fill the table, click **Fill Table**.

If the **Show table history after fill** box is selected, the **History** dialog box opens, similar to the one shown.

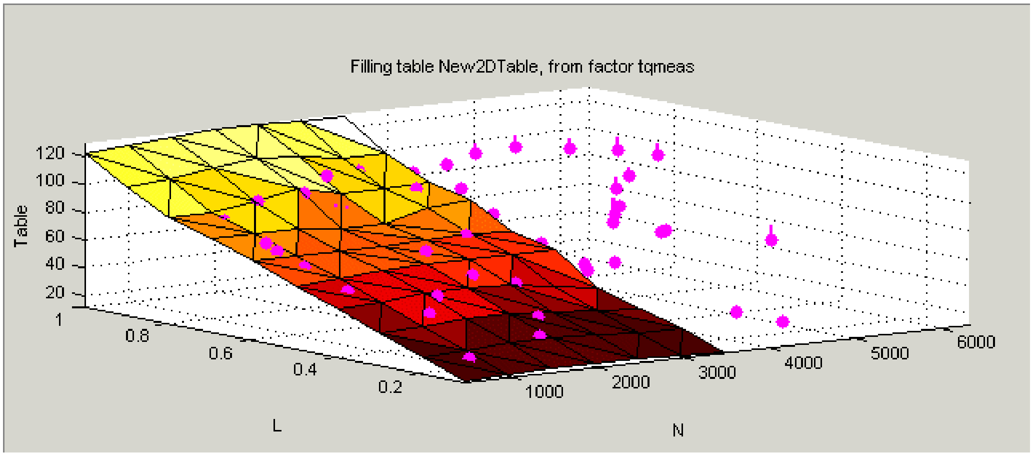


Version	Comment / Action	Date and Time
2	Values filled from data set meas_tq_data, factor tqmeas	16-Mar-2004 13:32:06
1	Initial configuration	16-Mar-2004 12:15:34

L \ N	500	1000	1500	2000	2500	3000	3500
0.1	12.245	13.471	14.637	15.084	14.622	13.805	13.044
0.2	23.802	25.336	26.94	27.322	25.9	24.344	23.697
0.3	35.14	36.987	38.912	38.876	36.598	33.439	31.511
0.4	46.028	48.217	51.119	51.517	49.49	45.317	40.169
0.5	56.839	58.411	60.752	62.257	62.139	61.779	62.486
0.6	68.694	69.387	69.545	69.367	69.788	71.364	68.274
0.7	79.019	79.285	78.65	76.015	75.705	82.919	85.571
0.8	88.482	88.409	92.981	98.575	92.016	91.03	93.019
0.9	104.147	106.258	110.804	114.302	112.183	107.478	107.431
1	121.64	123.967	126.968	129.007	128.826	127.695	127.643

6 Click **Close** to close the **History** dialog box and return you to the **Table Filler** display.

7 To view the graph of your table, select **Data > Plot > Surface**.



This display shows the table filled with the experimental points overlaid as purple dots.

Creating Rules

You can ignore points in the data set when you fill your lookup table.

By defining a region to include or exclude such points, you create a rule for the table filling.

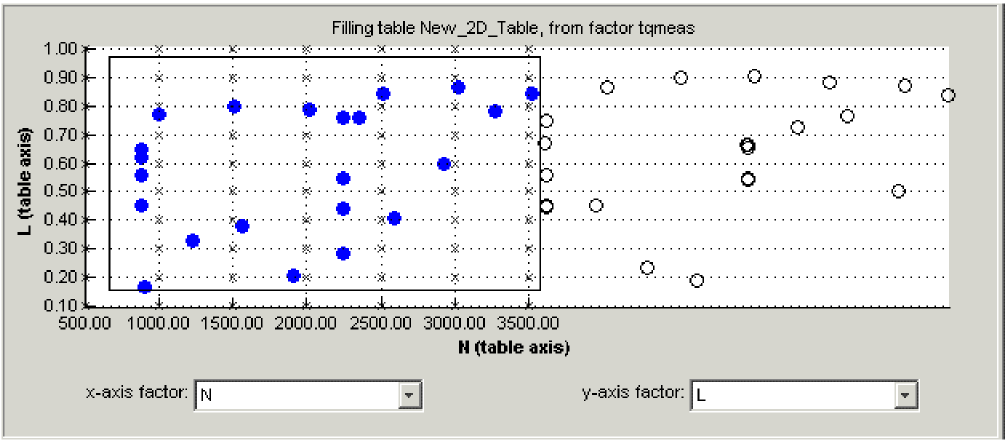
For example, you might want to fill a lookup table that has a range of operating points that is smaller than the range of the experimental data.

To ignore points in the data set,

1. Select **Data > Plot > Data Set**. This displays the view of where the breakpoints lie in relation to the experimental data.

2. To define the region that you want to include, left-click and drag the plot. For example, see the following display.

   This region defines a rule in the **Table filling rules** pane.



3 To fill the table based on an extrapolation over these data points only, click **Fill Table**.

The display of the surface now shows the table filled only by reference to the data points that are included in the range of the table.

You can now review your data set using the options in the **View** and **Plot** panes of **Data Sets**.

You can add any number of rules to follow when filling tables. For example, you might be aware that a particular test run included in the chosen area is not good data. You can click and drag to enclose any chosen point, then right-click that rule (in the **Table filling rules** pane) and select **Exclude Points**. You can set any number of rules to make sure you fill the table by using just the points you are interested in.

Right-Click Options

Select **Data -> Table Fill** to reach the following options:

- **Enable Rule**: Apply the rule to the data.
- **Disable Rule**: Do not apply the rule, but also do not delete it.
- **Exclude Points**: Do not include these points in table filling.
- **Include Points**: Include points in table filling.

- **Promote Rule**: Change order of rules.
- **Demote Rule**: Change order of rules.
- **Clear Rule**: Delete this rule.

You can use these options to enable an iterative process. You can fine-tune the selection of data points: try different selections of data to fill your tables, check the results, then reuse the same rules for the same or different tables.

Surface Viewer

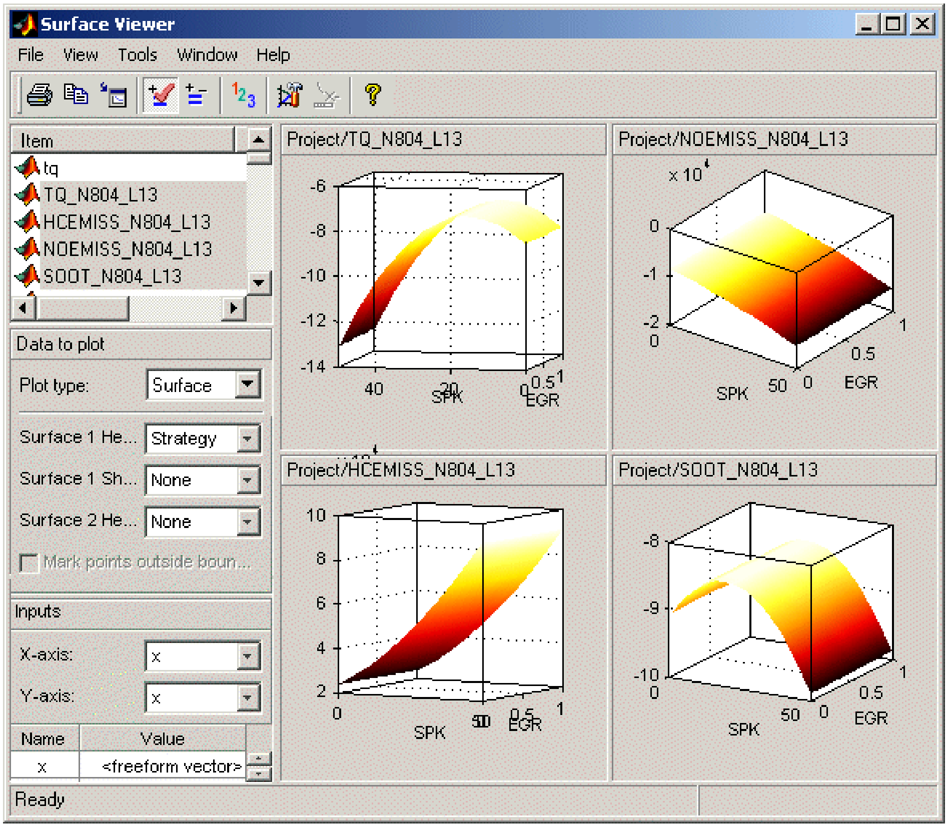
This section includes the following topics:

The Surface Viewer in CAGE

The **Surface Viewer** enables you to view the model or the feature as it varies over the ranges of its variables. You can automatically step through values of a variable, to make a movie of the behavior of the feature or model. You can view the model or feature using a variety of plot types.

Note The **Surface Viewer** is only available when you are viewing models, tradeoffs or the feature node of a feature calibration.

Following is an example of the **Surface Viewer** displays.



Plot type: Surface

Surface 1 He... Strategy

Surface 1 Sh... None

Surface 2 He... None

□ Mark points outside boun...

X-axis: x

Y-axis: x

Name	Value
x	<freeform vector>

Ready

Viewing a Model or Strategy

To access the surface viewer, select **Tools > Surface Viewer** or click on the toolbar.

These are the main steps to view the model or feature using the **Surface Viewer** dialog box:

1 The model or feature selected when you open the **Surface Viewer** is displayed in the plot. If you have more than one model or feature, select what to display from the top **Items** list.

   You can multiselect up to 4 items at once using **Ctrl+click** (the plot view on the right divides into a maximum of 4 plots). All the settings below the **Items** list apply to all plots. If one of the features selected in the **Items** list does not contain the appropriate input variables you select to plot, there will be no plot for that item.

2 Select the ranges for the variables. (See "Setting Variable Ranges" on page 8-5.)

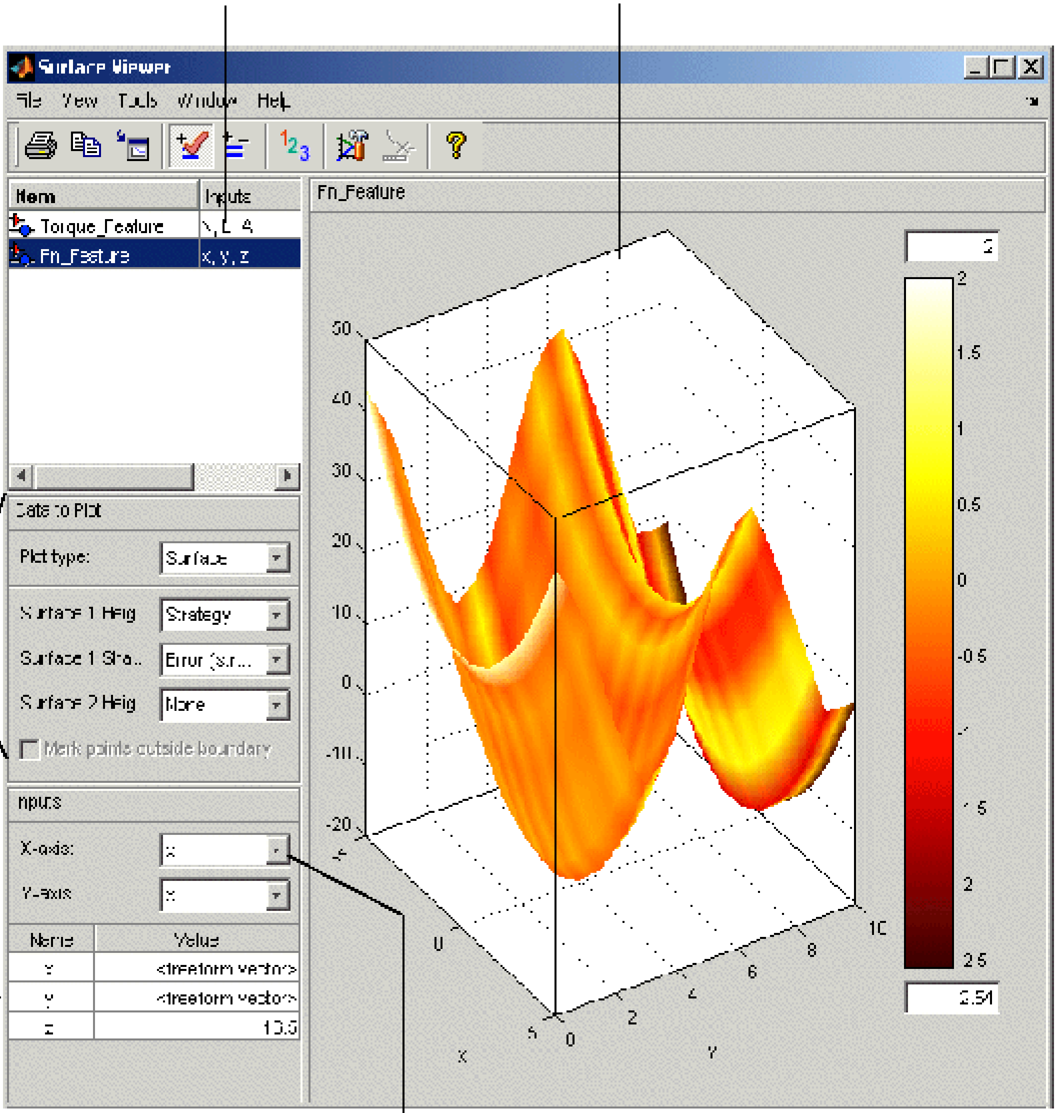
3 Choose the plot type to display. (See "Displaying the Model or Feature" on page 8-7.). You can view surfaces, contour plots, single and multilines, movies, tables, and single values.

   For example, as you view a feature, you can view either the strategy, the model associated with that feature, the error between the model and the strategy, or the prediction error if the model was imported from the Model Browser. You can also use one of these factors to shade the surface formed by one of the other factors, and you can select any two factors to display simultaneously as two surfaces.

- You can make a movie. (See "Making Movies" on page 8-14). This enables you to view the model or feature as it steps through several values of a variable. For example, if you want to view a feature calibrated for maximum brake torque (MBT) as it varies over exhaust gas recycling (EGR), you can make a movie of the feature.
- You can also print or export the display. (See "Printing and Exporting the Display" on page 8-18.)

Models or features in the project and their inputs

The model in the feature, shaded by the error (in this case)



Item	Inputs
Torque_Feature	N, L, A
Fn_Feature	x, y, z

Data to Plot

Plot type: Surface

Surface 1 Heig: Strategy

Surface 1 Sha..: Error (s.r...

Surface 2 Heig: None

□ Mark points outside boundary

Inputs

X-axis: x

Y-axis: y

Name	Value
x	<freeform vector>
y	<freeform vector>
z	13.5

3. Plot controls

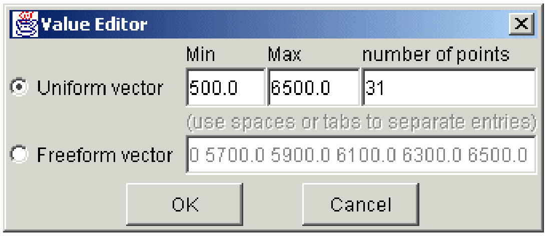
Variable ranges

Axes controls

Setting Variable Ranges

The **Surface Viewer** does not work over continuous ranges, only at discrete points. You must specify, for the model or feature, the discrete points you want to include in the display. You can display models or features over a range of points. To edit the displayed values of a variable, double-click in the value box for the appropriate variable.

- Variables not being used for the axes plotted have a single value for that plot; to edit the displayed value for these variables you can type directly into the edit box after double-clicking.
- For variables specified by the axes drop-down menus, the value box displays the range over which that variable is plotted and the number of points plotted across that range. To edit both the range and the number of points, double-click the value box. The **Value Editor** opens.



Value Editor

	Min	Max	number of points
● Uniform vector	500.0	6500.0	31

(use spaces or tabs to separate entries)

○ Freeform vector: 0 5700.0 5900.0 6100.0 6300.0 6500.0

Here you can indicate the points to include in the display. You can specify

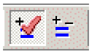
- The minimum and maximum values and the number of points across that range by choosing **Uniform Vector** and typing in the edit boxes **Min**, **Max**, and **Number of points**.
- Each discrete point at which you want to evaluate the model (or feature), by choosing **Freeform vector**, and then typing the required values.

  For example, if you want to display the variable *x* at 0, 1, 7, 30, and 50, enter the following in the **Freeform vector** edit box, separated by tabs or spaces:

  ```
  0 1 7 30 50
  ```

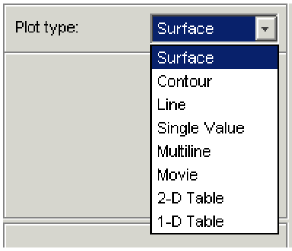
Click **OK** to apply your changes to the plot.

When you alter the variables, you can select whether you want the display to update automatically or not. You can toggle the automatic update on and off by selecting **Tools > Auto-Evaluate**. When you want to update the display, select **Tools > Evaluate Now** . Both of these options have equivalent toolbar buttons:



Displaying the Model or Feature

The **Plot Type** drop-down menu gives the options on how to display the model or feature, as shown below.



Use the options in this menu to display the model or feature in the following ways:

- "Surface" on page 8-8
- "Contour" on page 8-10
- "Line" on page 8-11
- "Single Value" on page 8-11
- "Multiline" on page 8-12
- "Table" on page 8-12
- Movie. See "Making Movies" on page 8-14.

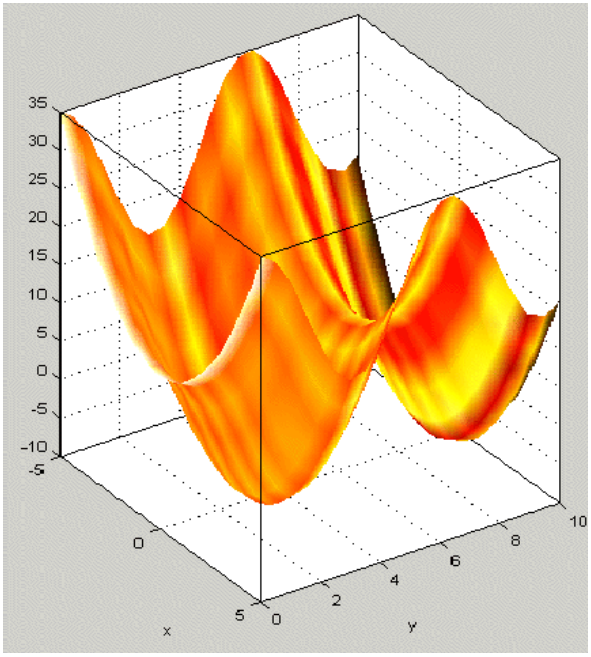
When plotting multiple models or features, it can be useful to link axes rotation or use common Y- or Z- ranges. Use the display options (toolbar button or **View** menu).

In any of these views you can select **View > Statistics**, or click the equivalent toolbar button. This opens a dialog box with a list of the summary statistics (mean, standard deviation, maximum, or minimum) of your currently selected model, strategy, or error for the current display.

For the plots (not movie, single value or tables) you can use the **File** menu or toolbar to print, copy to clipboard or print to figure. You can also export plot values to CSV file. See "Printing and Exporting the Display" on page 8-18.

You can alter display options for all plots (not value or tables) with the **View** menu or toolbar button.

Surface

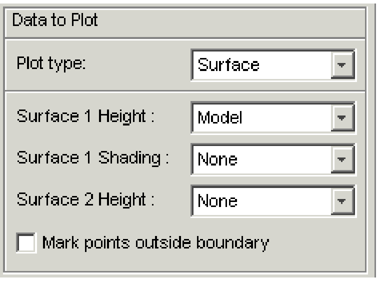


You can rotate the surface plots by left-clicking and dragging.

If you are using the surface viewer to view a feature, you can choose the following options to display:

- Model
- Strategy
- Prediction Error
- Error (between the model and the strategy)

When viewing models there are no strategy options. You can choose these options from the drop-down menus for **Surface 1 Height**, **Surface 1 Shading**, and **Surface 2 Height**, as illustrated below.



Data to Plot

Plot type: Surface

Surface 1 Height : Model

Surface 1 Shading : None

Surface 2 Height : None

□ Mark points outside boundary

You can view any of these options alone as a primary surface (by leaving the last two options set to **None**). You can add a second option to shade the primary surface, for example to color your model surface with the error between the model and the strategy, to highlight problem areas.

When you choose to shade a primary surface, a color bar appears to the right of the plot to show you the scale. You can change the maximum and minimum values of the shading factor by typing in the edit boxes above and below the color bar. You can see an example like this in “Viewing a Model or Strategy” on page 8-3.

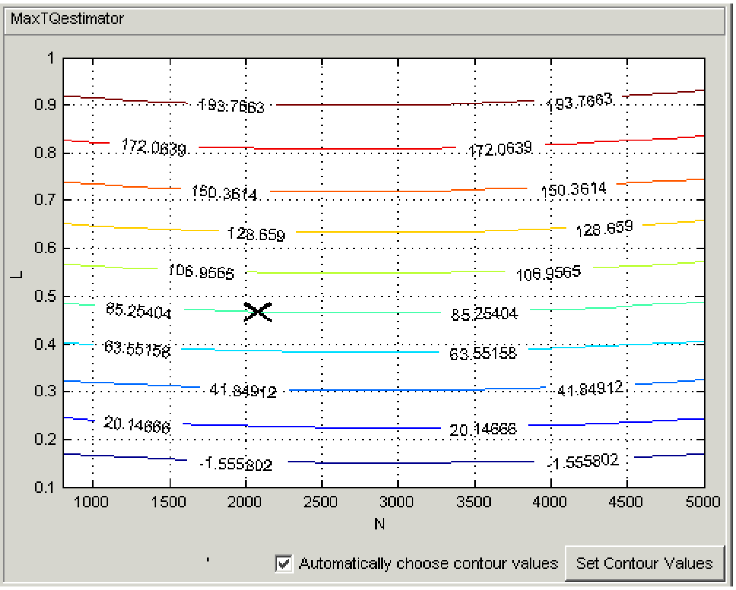
You can add a second surface to display any two of the options simultaneously, for example, your model and your strategy.

If you have a boundary model, you can display the boundary by selecting the check box.

Select the **Inputs** to plot from the **X-axis** and **Y-axis** drop-down lists, and specify the ranges of inputs in **Value** controls. See “Setting Variable Ranges” on page 8-5.

Note For information on the two different error displays available using the surface view, see the next section, “Displaying Errors” on page 8-16.

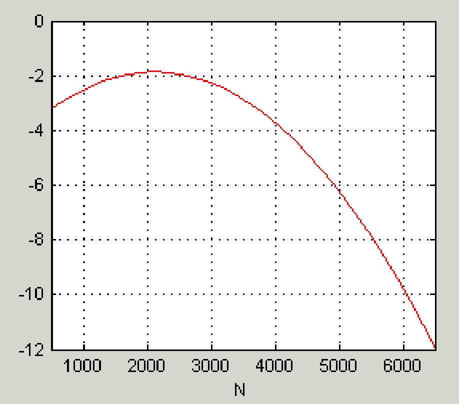
Contour



You can specify where you want contours by clicking **Set Contour Values**. Use the check box to return to automatic contour value selection. You can also control number of contours, filling and labels in the display options (toolbar or **View** menu).

You can enable **Cursor Mode** (use the **View** menu or toolbar button) and then click on the plot lines to display the values at a point (plotted with an X). The values are shown in the status bar.

Line



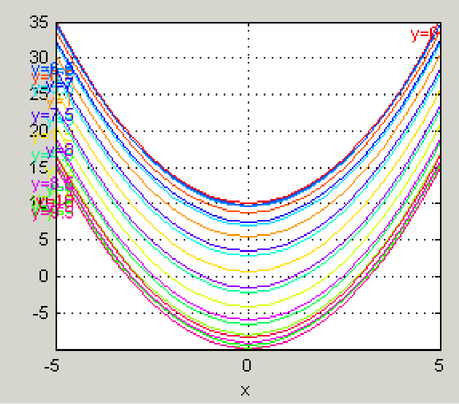
A line plot - you can display up to three different lines (strategy, model, prediction error and error between the model and strategy). Use the **Line** drop-down lists to select what to plot. You can select the check box to clip to a boundary if available.

You can enable **Cursor Mode** (use the **View** menu or toolbar button) and then click on the plot lines to display the values at a point (plotted with an X). The values are shown in the status bar.

Single Value

This displays the value of the model, strategy, prediction error or error at the point you have specified in the variable value boxes.

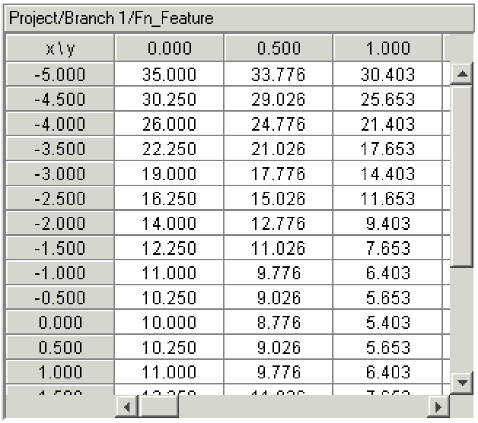
Multiline



Select the variables to plot from the **X-axis** and **Line colors** drop-down menus. Control the number of lines by altering the **Values**. You can use the check box to clip to a boundary if available.

You can enable **Cursor Mode** (use the **View** menu or toolbar button) and then click on the plot lines to display the values at a point (plotted with an X). The values are shown in the status bar.

Table



Project/Branch 1/Fn_Feature

x\y	0.000	0.500	1.000
-5.000	35.000	33.776	30.403
-4.500	30.250	29.026	25.653
-4.000	26.000	24.776	21.403
-3.500	22.250	21.026	17.653
-3.000	19.000	17.776	14.403
-2.500	16.250	15.026	11.653
-2.000	14.000	12.776	9.403
-1.500	12.250	11.026	7.653
-1.000	11.000	9.776	6.403
-0.500	10.250	9.026	5.653
0.000	10.000	8.776	5.403
0.500	10.250	9.026	5.653
1.000	11.000	9.776	6.403

You can select a 2-D or 1-D table to display. Select the check box to mark cells outside the boundary.

Choose variables to be the axes of your table and set the range and number of points in the same way as for all the plots. Set single values for any other variables. For more information, see “Setting Variable Ranges” on page 8-5.

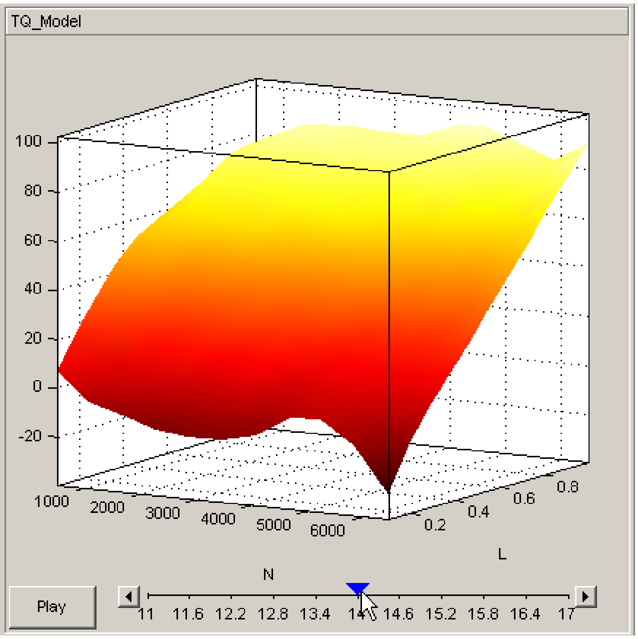
For 2–D tables you can use the **Cell values** drop-down menu to select whether to display the model output or the prediction error.

For 1-D tables you can select what to display in columns 1, 2 and 3: `Model`, `Prediction error`, `Strategy` or `Error (strategy  model)` (for features), or choose `None` for 2 and 3 to display only a single column. When viewing models there are no strategy options.

Making Movies

How to make a movie that allows you to see an evaluation over two variables at successive values of a third variable.

Choose **Movie** from the **Plot Type** drop-down menu in the **Data to Plot** pane.



The movie option allows you to see an evaluation over two variables at successive values of a third variable. For example, a model of torque might have speed (N), load (L), and air/fuel ratio (A) as inputs.

The movie option allows you to view how the torque model behaves over the ranges of speed and load for successive values of air/fuel ratio.

1 Select three variables from the **X-axis**, **Y-axis**, and **Time** drop-down menus, to indicate which variable you want to display. You can view the model surface plotted across the range of two variables, and define the third variable as "time" to see the model surface change across the third variable’s range.

2 Define the variable ranges using the **Value** boxes for the inputs. See "Setting Variable Ranges" on page 8-5.

3 Select the check box to mark boundaries if available.

4 Click **Play**.

5 You can click the buttons at each end of the progress bar under the plot to step through the movie, or click anywhere along the bar (or click and drag the blue pointer) to display a particular point in the movie. You can rotate the plot (including during play).

Displaying Errors

There are two different error displays available in the surface display options for primary and secondary surfaces and surface shading:

- Error between the model and the strategy (See “Feature Error Data” on page 8-16 following.)
- Prediction error of the model (See “Prediction Error Data” on page 8-16.)

Feature Error Data

When you are viewing a feature, this displays the error between the strategy and the model.

To display the error, select **Error (strategy-model)** from the drop-down menu for primary or secondary surface. You can also choose to shade your primary surface with the error by using the **Surface 1 Shading** menu.

To view the error statistics, select **View > Statistics**. This opens a dialog box with a list of the summary statistics for the error between model or feature.

Prediction Error Data

If the model is imported from the Model Browser, it is possible to display the *prediction error* (PE) data.

Prediction Error Variance (PEV) is a very useful way to investigate the predictive capability of your model. It gives a measure of the precision of a model’s predictions. PEV can also be examined in the Model Browser, both in the **Prediction Error Variance Viewer** and to shade surfaces in the **Model Selection** and **Model Evaluation** views. Here you can examine the PEV of designs and models. When you export the model to CAGE you can see this data in the **Surface Viewer** in the `Prediction Error` option. See the Model Browser GUI Reference and Technical Documents for details about the calculation of Prediction Error.

Viewing the Prediction Error

Select `Prediction Error` from the drop-down display menus for primary or secondary surfaces. You can also choose `Prediction Error` to shade your primary surface. As with all other plots, you can view the statistics for the `Prediction Error` displayed by selecting **View > Statistics**. The mean, standard deviation, and so on are calculated over the range specified in the variable value boxes.

Printing and Exporting the Display

To print the display, select **File -> Print**, or you can select Print to Figure. Selecting **File > Copy to Clipboard** copies the plot image to the clipboard. This is useful if you want to place plot images into other applications. These print options also have equivalent toolbar buttons.

You can also export the display data to a comma-separated variable file.

To export the display, select **File > Export to CSV**. The currently selected option is exported. The primary input to the first plot is exported (this is the top left if you have multiple plots). The output is the values at the grid of points specified by the current ranges and input values. The inputs for shading and secondary surfaces are not exported.

Note that you cannot print table plots, but you can click and drag to select cells and press **Ctrl-C** to copy the values to the clipboard, or you can export them to CSV files and then load them into Excel.

Index