Model-Based Calibration Toolbox™

Getting Started Guide

MATLAB&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Revision History

Contents

Multi-Injection Diesel Calibration

4

Model Quickstart

5

Design of Experiment

6

viii *Contents*

Introduction

The following sections introduce Model-Based Calibration Toolbox software.

- "Model-Based Calibration Toolbox Product Description" on page 1-2
- "What Is Model-Based Calibration?" on page 1-3

Model-Based Calibration Toolbox Product Description

Model and calibrate complex powertrain systems

Model-Based Calibration Toolbox provides apps and design tools for modeling and calibrating complex nonlinear systems. It can be used in a wide range of applications, including powertrain systems such as engines, electric machines, pumps, and fans, as well as nonautomotive systems such as jet engines, marine hydrofoils, and drilling equipment. You can define optimal test plans, automatically fit statistical models, and generate calibrations and lookup tables for high-degree-of-freedom systems that would require exhaustive testing using traditional methods. Using the toolbox apps or MATLAB® functions, you can automate the model fitting and calibration process.

Models created with Model-Based Calibration Toolbox can be exported to Simulink® to support control design, sensitivity analysis, hardware-in-the-loop testing, and other simulation activities. Calibration tables can be exported to ETAS INCA and ATI Vision.

What Is Model-Based Calibration?
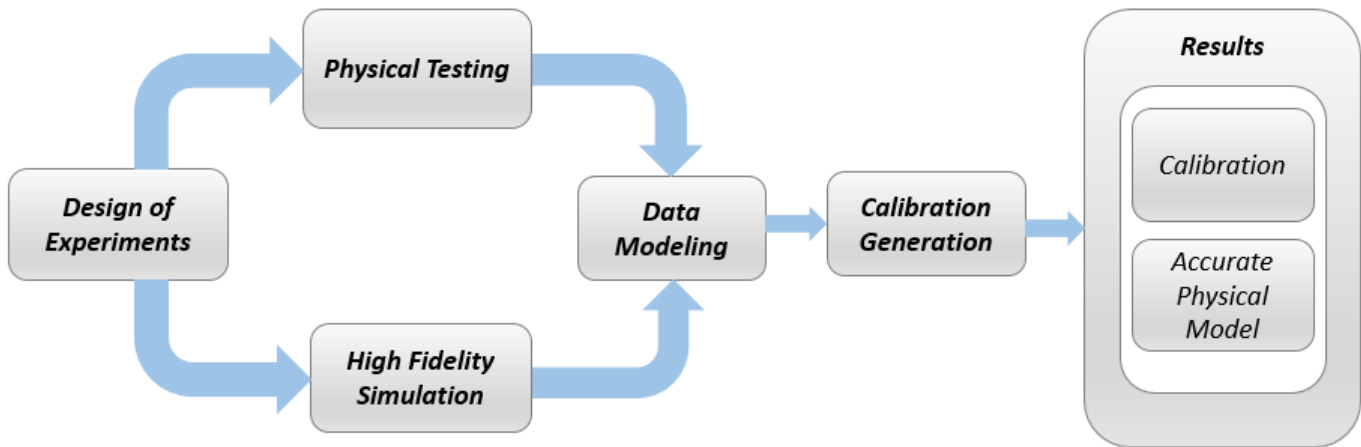
High accuracy engine models are a key component for reducing calibration effort and engine development time.

The time spent calibrating an engine control unit has been increasing, due to new control actuators. The new actuators give the potential for increased performance, reduced emissions, and improved fuel consumption. It is necessary to apply advanced modeling and optimization techniques to achieve the full benefits available from the addition of new actuators. Advanced modeling techniques offer increased understanding of the complex, nonlinear engine responses. High accuracy models can be used throughout the design process, including the calibration of base maps with the optimal settings for the control parameters, determined by constrained optimizations.

The toolbox has two main user interfaces for model-based calibration workflows:

- Model Browser for design of experiment and statistical modeling
- CAGE Browser for analytical calibration

The Model Browser part of the toolbox is a powerful tool for experimental design and statistical modeling. The models you build with the Model Browser can be imported into the CAGE Browser part of the toolbox to produce optimized calibration tables.

Designs and Modeling in the Model Browser

The Model Browser is a flexible, powerful, intuitive graphical interface for building and evaluating experimental designs and statistical models:

- Design of experiment tools can drastically reduce expensive data collection time.
- You can create and evaluate optimal, space-filling, and classical designs, and constraints can be designed or imported.
- Hierarchical statistical models can capture the nature of variability inherent in engine data, accounting for variation both within and between tests.
- The Model Browser has powerful, flexible tools for building, comparing, and evaluating statistical models and experimental designs.

- There is an extensive library of prebuilt model types and the capability to build user-defined models.
- You can export models to CAGE or to MATLAB or Simulink software.

Starting the Model Browser

To start the application, type

`mbcmodel`

at the MATLAB command prompt.

Calibration Generation in 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 Model Browser models. From these models, CAGE builds steady-state ECU calibrations.

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

Starting the CAGE Browser

To start the application, type

`cage`

at the MATLAB command prompt.

Gasoline Engine Calibration

Gasoline Case Study Overview

In this section...
"Gasoline Calibration Problem Definition" on page 2-2
"Case Study Example Files" on page 2-3

Gasoline Calibration Problem Definition

This case study demonstrates how to systematically develop a set of optimal steady-state engine calibration tables using the Model-Based Calibration Toolbox.

The engine to calibrate is a direct-injected 1.5L spark ignition turbocharged engine with dual cam-phasers and turbocharger wastegate.

The aim of the calibration is to maximize torque at specific speed/load values across the engine's operating range, and meet these constraints:

- Limit knock
- Limit residual fraction
- Limit exhaust temperature
- Limit calibration table gradients for smoothness.

The analysis must produce optimal engine calibration tables in speed and load for:

- Spark advance ignition timing (SA)
- Throttle position % (TPP)
- Turbo wastegate area % (WAP)
- Air/Fuel Ratio (Lambda: LAM)
- Intake cam phase (ICP)
- Exhaust cam phase (ECP)
- Torque (TQ)
- Brake-specific fuel consumption (BSFC)
- Boost (MAP)
- Exhaust temperature (TEXH)
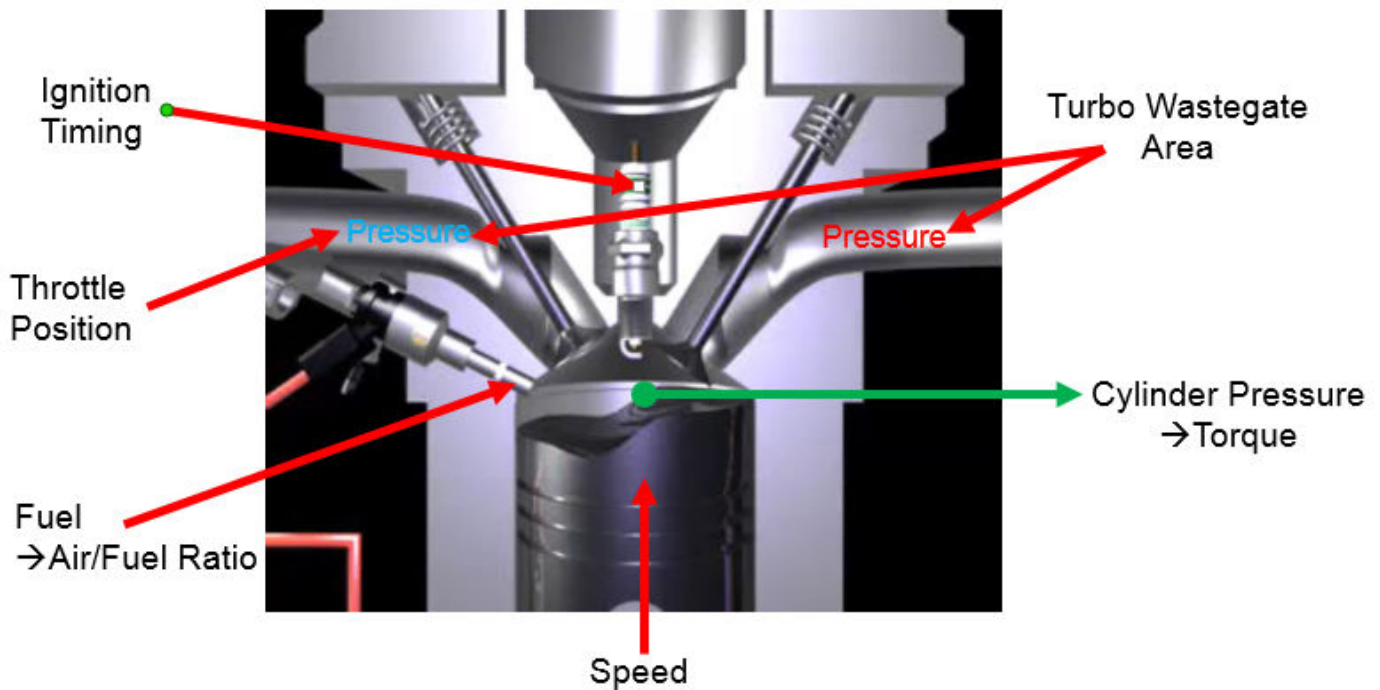
This case study illustrates the model-based calibration process.
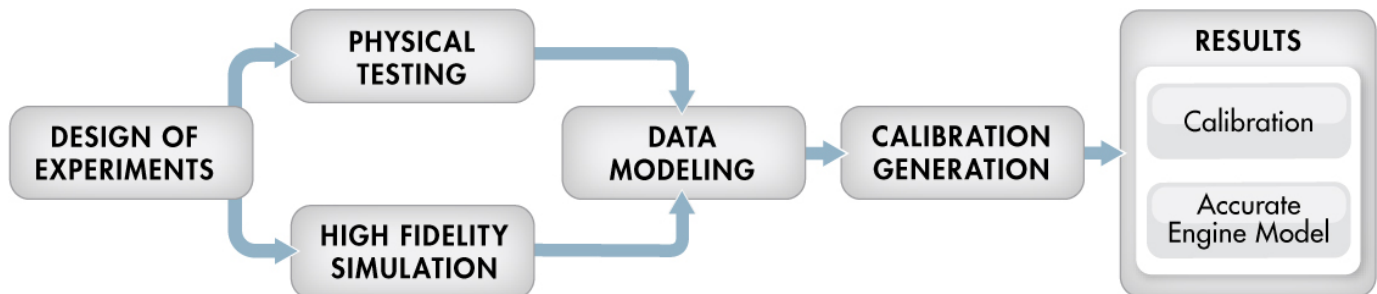
Case Study Example Files

The following sections guide you through opening example files to view each stage of the model-based calibration process. You can examine:

1 Designs, constraints, boundary model, and collected data, in topic "Design of Experiment" on page 2-5.

2 Finished statistical models, in topic "Empirical Engine Modeling" on page 2-11.

3 Optimization setup and results, and filled calibration tables, in topic "Optimization" on page 2-15.

Use these example files to understand how to set up systematic calibrations for similar problems. For next steps, see "Design of Experiment" on page 2-5.

Tip Learn how MathWorks® Consulting helps customers develop engine calibrations that optimally balance engine performance, fuel economy, and emissions requirements: see Optimal Engine Calibration.

Design of Experiment

Context

This topic describes design of experiments for the gasoline one-stage case study. To view the high-level workflow, see "Gasoline Case Study Overview" on page 2-2.

Benefits of Design of Experiment

You use design of experiment to efficiently collect engine data. Testing time (on a dyno cell, or as in this case, using high-fidelity simulation) is expensive, and the savings in time and money can be considerable when a careful experimental design takes only the most useful data. Dramatically reducing test time is increasingly important as the number of controllable variables in more complex engines is growing. With increasing engine complexity, the test time increases exponentially.

Power Envelope Survey Testing

The first stage to solve this calibration problem is to determine the boundaries of the feasible system settings. You need to generate the power envelope to constrain the design points. To do this, data was collected using simulation across a range of speed and torque. The initial survey determined boundaries that produce:

- Acceptable exhaust temperature (not too high to burn piston crowns)
- Achievable torque production
- Acceptable BSFC (not too high)
- Avoids knock

The envelope must include the idle region of low torque and speed, where the cams must be parked. The cam timings are set to zero in the idle design.

The initial survey design and test data provide information about the engine operating envelope. This information was used to create constraints for the final design, to collect detailed data about the engine behavior within those boundaries. You can then use this data to create response models for all the responses you need in order to create an optimal calibration for this engine.

The final design used 238 points for this engine with 4 inputs: speed, load, intake and exhaust cam. You can view the constraints defining the operating envelope by following the steps below.

Create Designs and Collect Data

You can use a space-filling design to maximize coverage of the factors' ranges as quickly as possible, to understand the operating envelope.

To create a design, you need to first specify the model inputs. Open the example file to see how to define your test plan.

1 Open MATLAB. On the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.
2 In the Model Browser home page, in the **Case Studies** list, select **Dual CAM gasoline engine with spark optimized during testing**. Alternatively, select **File > Open Project** and browse to the example file `gasolineOneStage.mat`, found in `matlab\toolbox\mbc\mbctraining`.

3 To view how to define your test plan design inputs, in the **All Models** tree, click the top project node, `gasolineOneStage`. In the **Common Tasks** pane, click **Design experiment**. In the New Test Plan dialog box, observe the inputs pane, where you can change the number of model inputs and specify the input symbols, signals and ranges. This example project already has inputs defined, so click **Cancel**.

4 Click the first test plan node in the **All Models** tree, `gasolineOneStageDoE`. The test plan view appears.

5 Observe the inputs listed on the test plan diagram. Double-click the **Inputs** block to view the ranges and names (symbols) for variables in the Input Factor Set Up dialog box. The design inputs are torque, speed, intake cam, and exhaust cam. The dynamometer test setup is in speed/torque and so the design is in speed/torque.

Close the dialog box by clicking **Cancel**.

6 After setting up inputs, you can create designs. In the **Common Tasks** pane, click **Design experiment**.

The Design Editor opens. Here, you can see how these designs are built.

7 Click the first design in the tree, `gasolineOneStageIdle`. If you do not see a 2D plot, select **View > Current View > 2D Design Projection**. Then select each design in the tree in turn.

- The first design in the tree, `gasolineOneStageIdle`, concentrates points in the idle area with cams parked. It uses the Sobol Sequence space-filling design type to maximize coverage. To park the cams, after creating the idle region space-filling design, the cam phaser positions are set to park position (0,0).
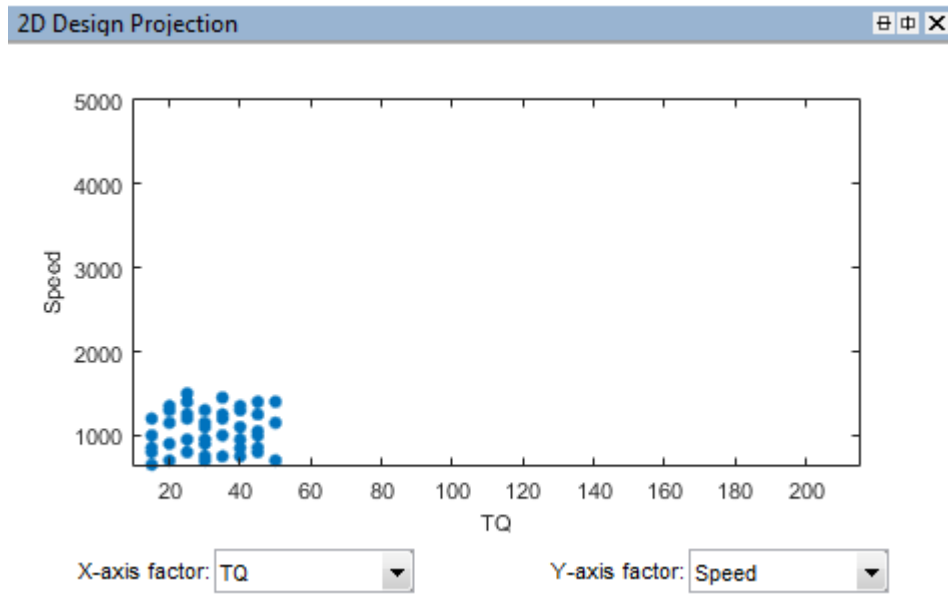
- The second design, `gasolineOneStageNonIdle`, is another Sobol Sequence space-filling design to span the non-idle engine operating envelope.
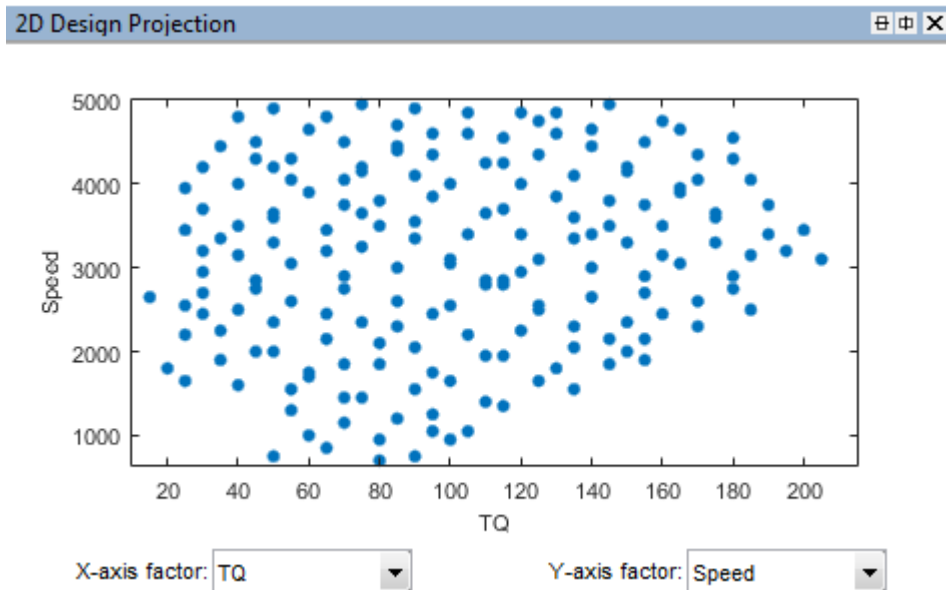
- The final design is called `gasolineOneStageMerged` because it contains the other two merged designs to cover the whole envelope.
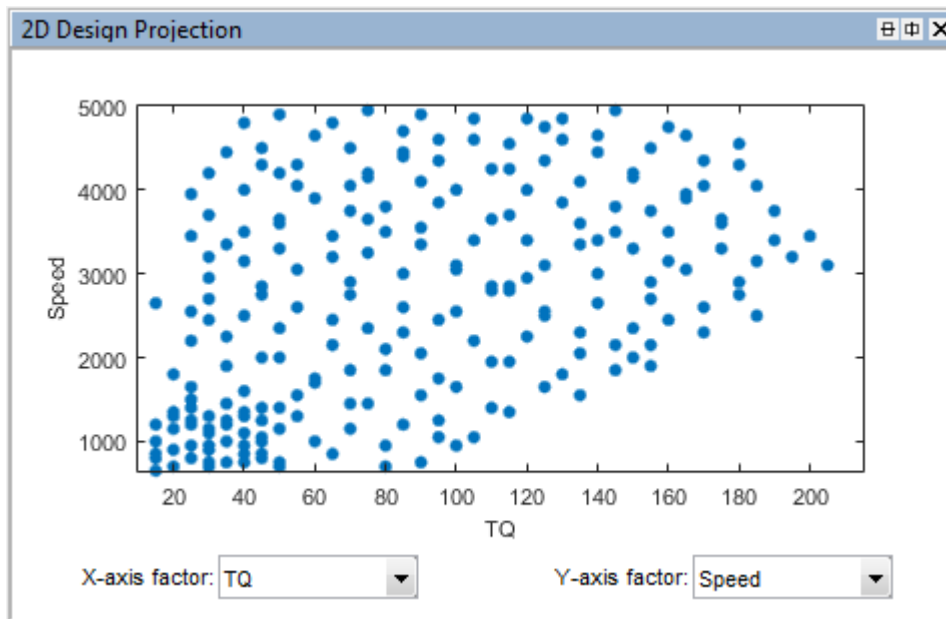
8   To see how the constraints are set up, select **Edit > Constraints**.

9   In the Constraints Manager dialog box, select each torque constraint in turn and click **Edit**. Observe the maximum and minimum torque constraints define the upper and lower boundary of the feasible operating envelope. These constraints were developed prior to this design in initial wide-open throttle and closed-throttle engine performance testing.
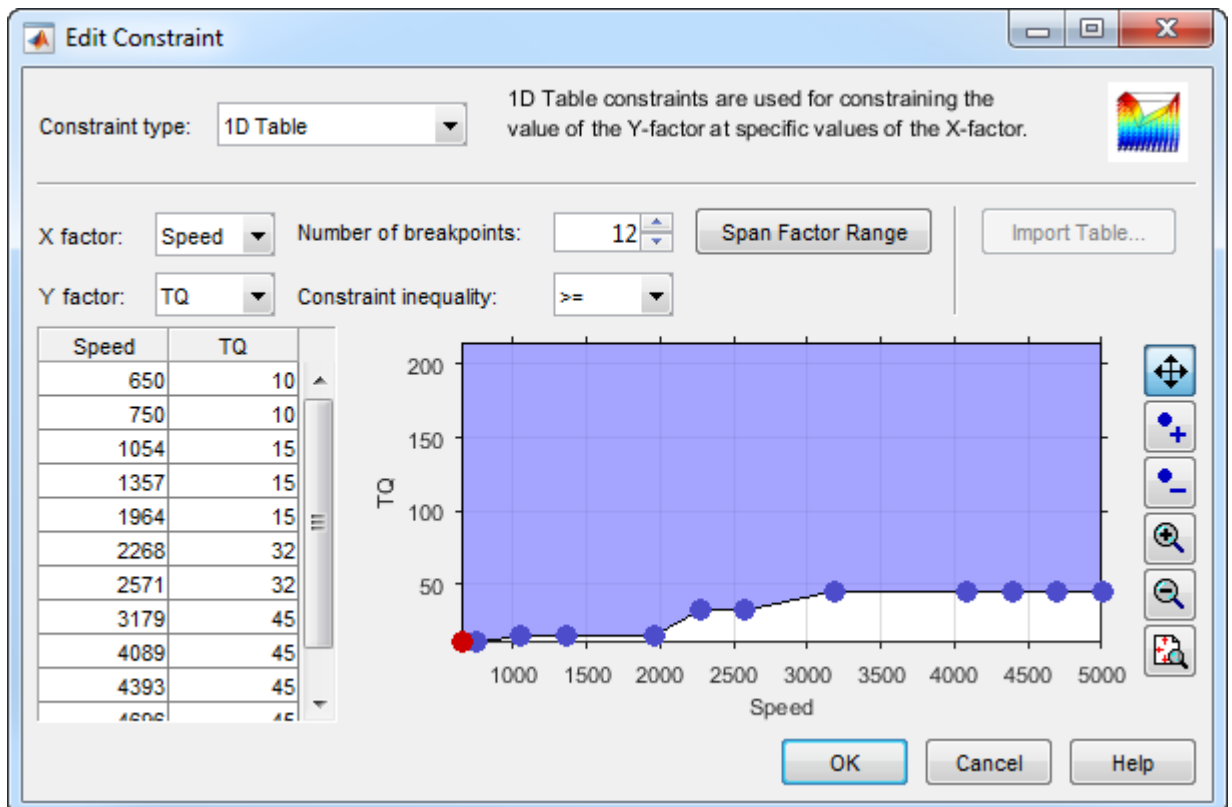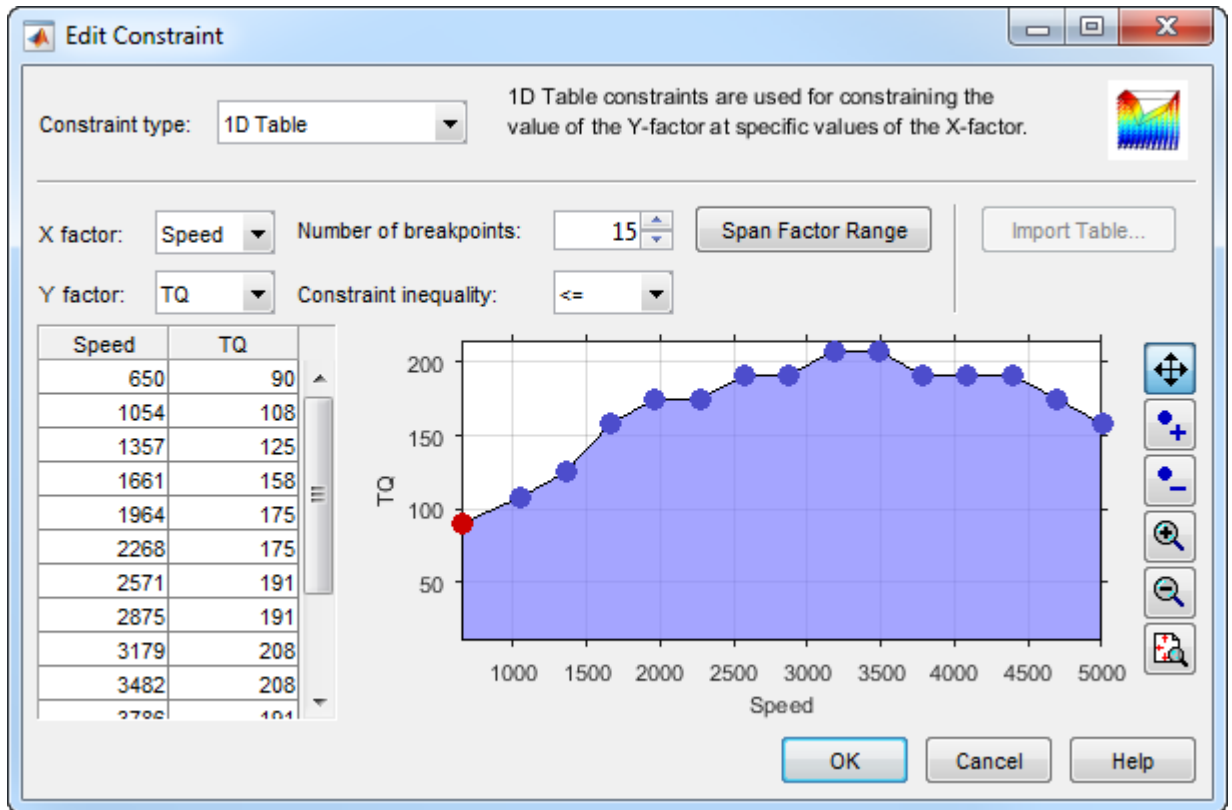
Observe that you can define areas to exclude by dragging points, typing in the edit boxes, or using the Table Editor tab.

To leave the constraint unchanged, click **Cancel**.

10 Observe the `Properties` of the selected `gasolineOneStageMerged` design under the tree, listing 2 constraints, and 238 points.

11 To experiment with a new design and avoid editing the prior designs, select the root `Designs` node and select **File > New Design**.

12 Add the constraints by selecting **Edit > Constraints**. In the Constraints Manager dialog box, click **Import**. Select the torque max and min constraints from the merged design and click **OK**. In the following dialogs, click **OK** to return to the Design Editor.

13 See how to construct a similar constrained space-filling design by selecting **Design > Space Filling > Design Browser**, or click the space-filling design toolbar button.

14 In the Space-Filling Design Browser, observe the design type is `Sobol Sequence`, and specify a **Number of points**. View the preview of design points. Click **OK**.
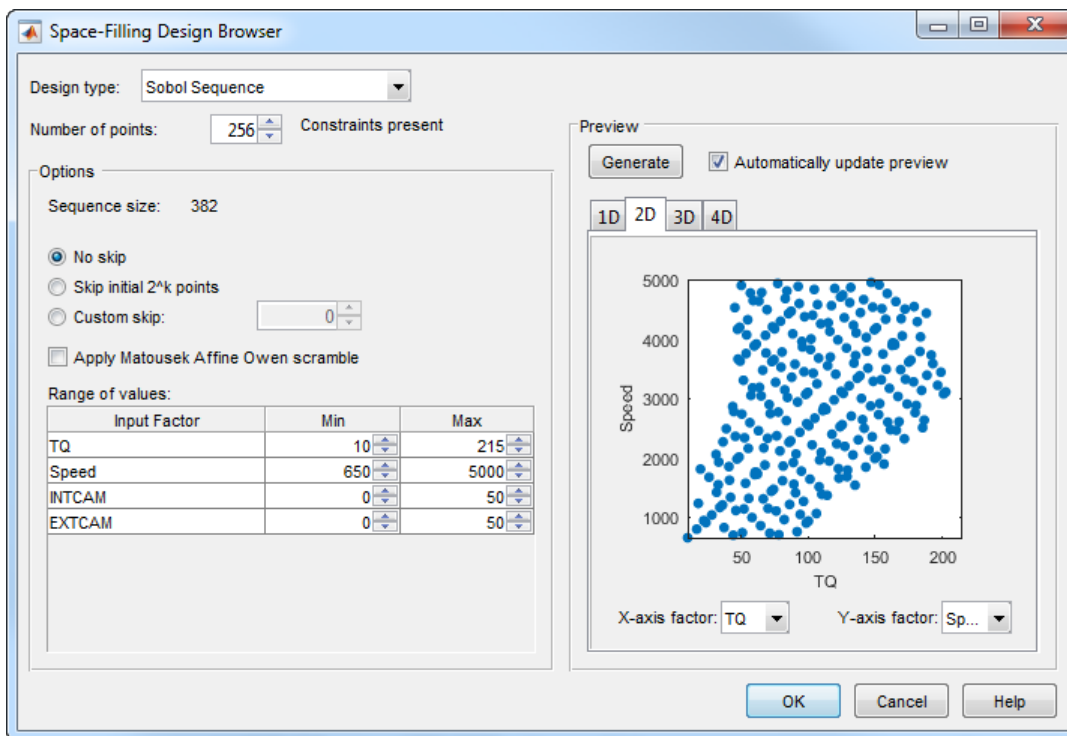
15 Click `gasolineOneStageMerged`. This design style is `Custom` because the points are rounded, using **Edit > Round Factor**. You might also sort design points to make data collection easier on a dyno. To preserve the space-filling sequence in case you want to add more points later, you can copy a design before rounding or sorting.

`gasolineOneStageMerged` is the final design used to collect data. To make it easier to collect the data, the points are rounded as follows:

- Intake and exhaust cam timings are rounded to 1 degree (2% of range)
- Speed is rounded to 50 RPM (1% of range)
- Torque is rounded to 5 Nm (3% of range)

You can export designs or copy and paste design points into other files to proceed to data collection.

16 Close the Design Editor.

The final `gasolineOneStageMerged` design was used to collect data from the GT-Power model with the Simulink and Simscape™ test harness. The example Model Browser project file `gasolineOneStage.mat` in the `mbctraining` folder contains this data, imported to the Model Browser after the data collection.

Data Collection and Physical Modeling

The toolbox provides the data in the projects for you to explore this calibration example.

MathWorks collected the data using simulation tools. Control and simulation models were constructed using a Simulink and Stateflow® test harness. Constrained experimental designs were constructed using Model-Based Calibration Toolbox. The points specified in the design were measured using the GT-Power engine simulation tool from Gamma Technologies (see https://www.gtisoft.com). The engine to calibrate is a direct-injected 1.5L spark ignition turbocharged engine with dual cam-phasers and turbocharger wastegate. This model is part of a GT-POWER engine library from Gamma Technologies.

To collect the data, Simulink and Stateflow controlled the GT-Power engine model to the desired Design of Experiments points.

Note Simulation time was reduced from days to minutes using Parallel Computing Toolbox™.

This simulation of 238 design points took 20 minutes to run in parallel on multiple machines in the cloud. Running on a single core, the same simulation takes 3 days. Parallel Computing Toolbox distributed the work to the 225 cores on a cloud computing cluster and showed that this problem scales linearly as you add workers.

The data was used in the next step of model-based-calibration, to create statistical models.
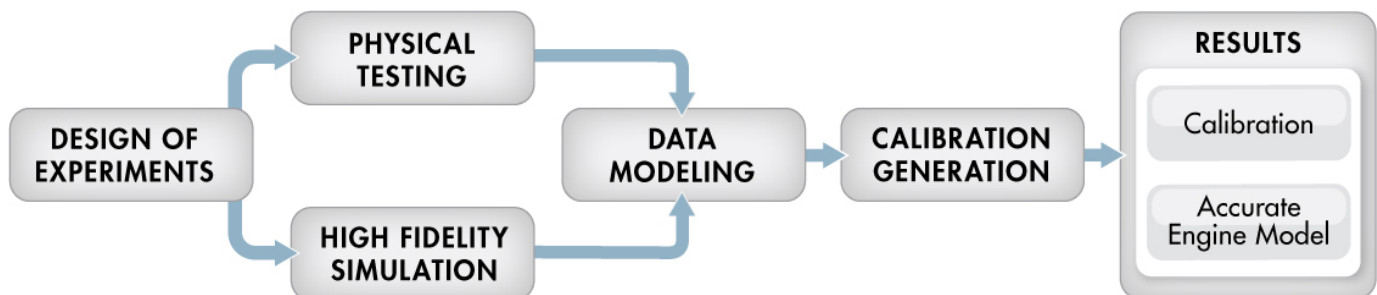
For next steps, see "Empirical Engine Modeling" on page 2-11.

Empirical Engine Modeling

After designing the experiments and collecting the data, you can fit statistical models to the data. Use the toolbox to generate accurate, fast-running models from the measured engine data.

The dynamometer test setup is in speed/torque and so the design is in speed/torque. The model is in speed/load because the production engine controller implementation uses load (derived from air flow) instead of torque tables. The controller uses load because airflow sensors are presently less expensive in mass production than torque meters.

Examine Response Models

1   Open MATLAB. On the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.

    In the Model Browser home page, in the **Case Studies** list, open **Dual CAM gasoline engine with spark optimized during testing**.

2   In the `gasolineOneStage.mat` project, click the second test plan node in the **All Models** tree, `gasolineOneStageModels`.

3   To assess high-level model trends, at the test plan node select the **Response Models** tab. After you fit models, the view at the test plan node displays the **Response Models** tab by default. View the cross-section plots of all the response models.
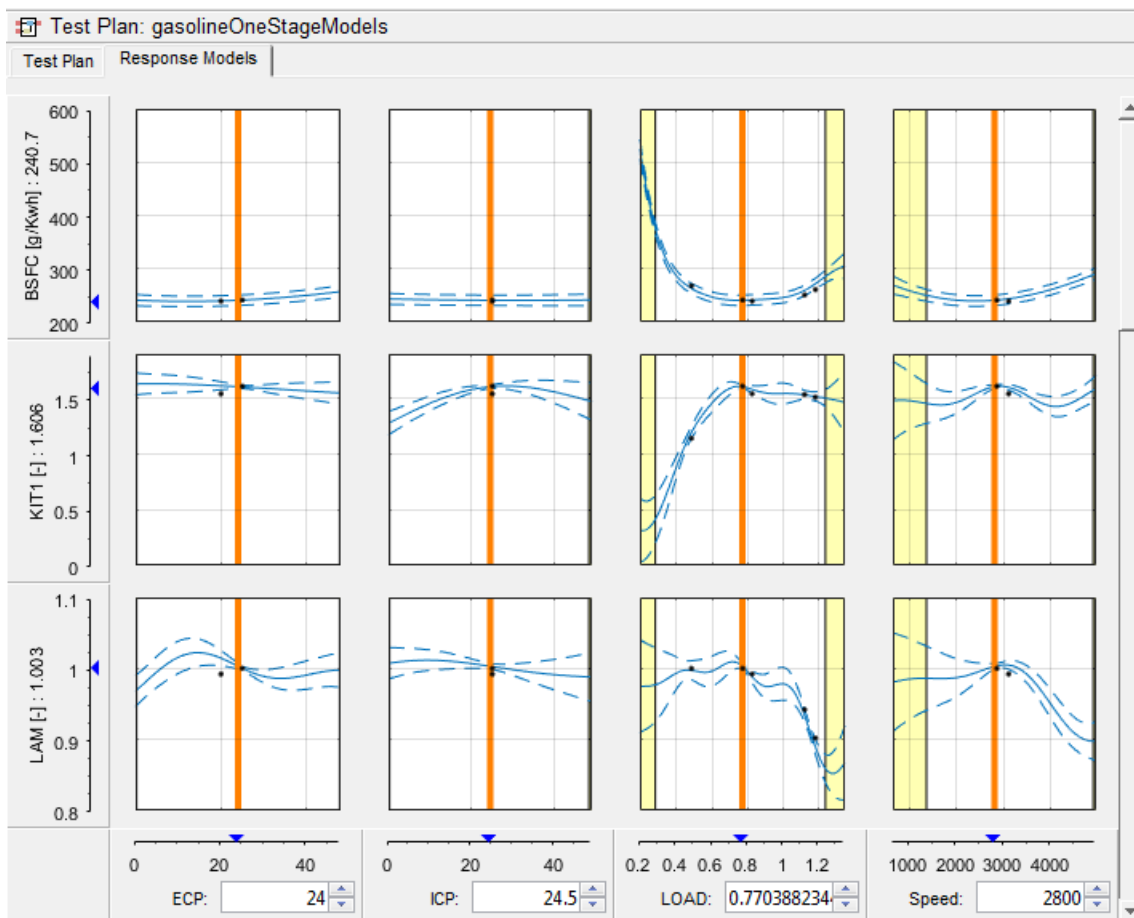
4   To view each response model in detail, expand `gasolineOneStageModels` test plan node in the
    All Models tree. Select the first response node under the test plan node, BSFC.

```
All Models
 gasolineOneStage
    gasolineOneStageDoE
    gasolineOneStageModels
        BSFC
        KIT1
        LAM
        MAP
        RF1
        SA
        TEXH
        TPP
        TSPEED
        Torque
        WAP
```
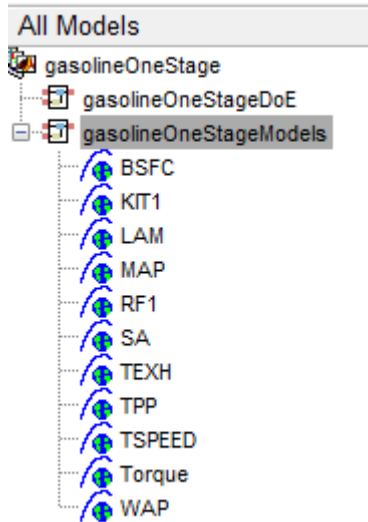
5   Examine the **Response Surface** plot and the **Diagnostic Statistics** plot.
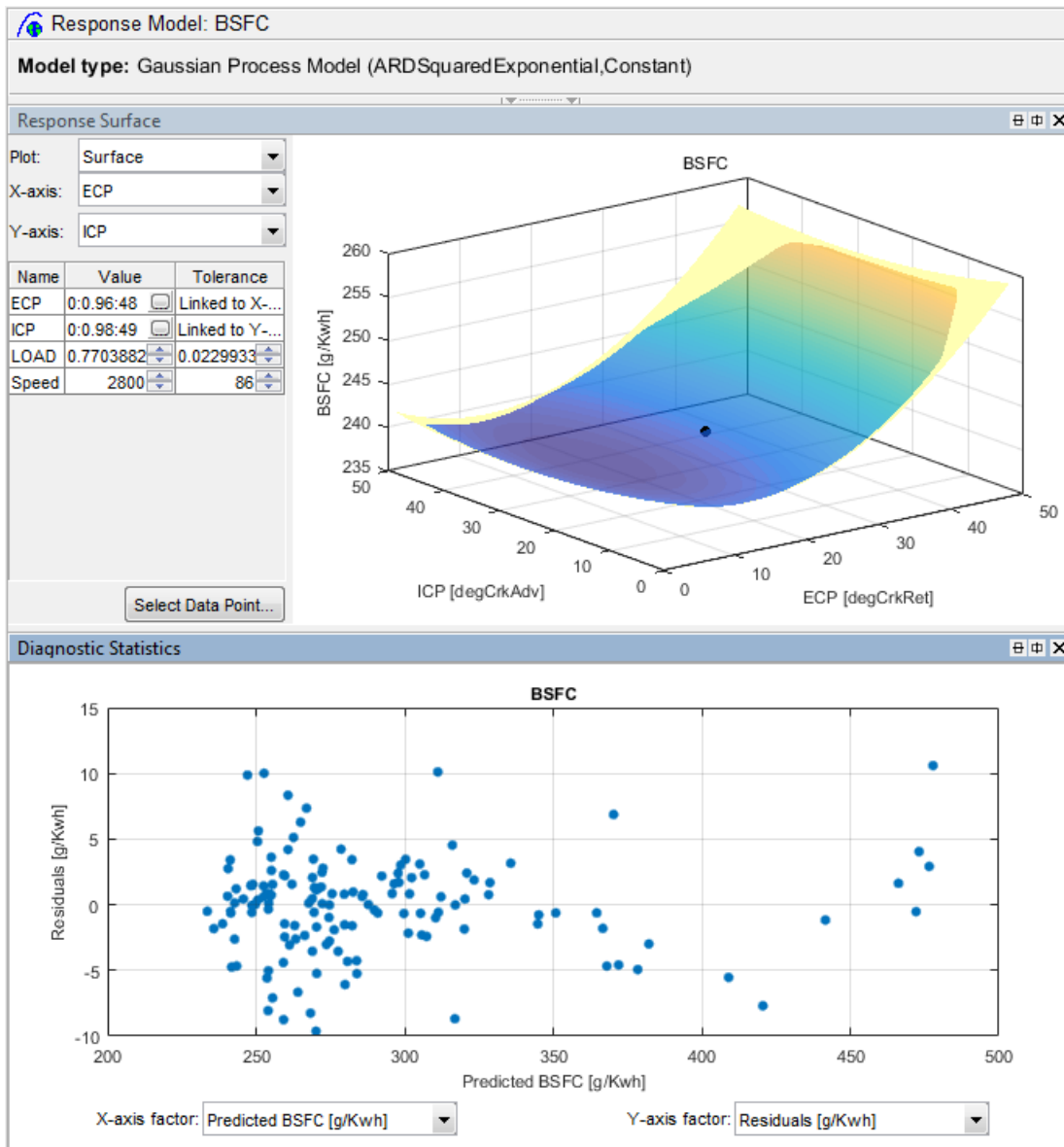
6 Examine the other responses in the All Models tree.

For details on using the plots and statistics to analyze models, see "Assess High-Level Model Trends" and "Assess One-Stage Models".

Examine the Test Plan

Examine the model setup.

1 At the `gasolineOneStageModels` test plan node, change to the test plan view if necessary by clicking the **Test Plan** tab. The Model Browser remembers selected views.

2 Observe the inputs and response model outputs listed on the test plan diagram.

3 Double-click the **Inputs** block to view the ranges and names (symbols) for variables on the Input Factor Set Up dialog box.

4 Double-click the **Model** block to view that the model class is the default for one-stage models, a `Gaussian Process Model`. When you use the **Fit models** button in the **Common Tasks** pane, and select a `One-stage` template, the toolbox sets the model type to a Gaussian Process Model.

For details on setting up one-stage models, see "Fit a One-Stage Model".

5 Click **Cancel** to close the Model Setup dialog box without altering your example models.

For next steps, see "Optimization" on page 2-15.

Optimization

Optimization Overview

After creating statistical models to fit the data, you can use them in optimizations. You can use the accurate statistical engine model to replace the high-fidelity simulation and run much faster, enabling optimization to generate calibrations in feasible times. You use the statistical models to find the optimal configurations of the engine that meet the constraints.

The statistical models described in "Empirical Engine Modeling" on page 2-11 were used in the next step of model-based-calibration, to create optimized calibration tables.
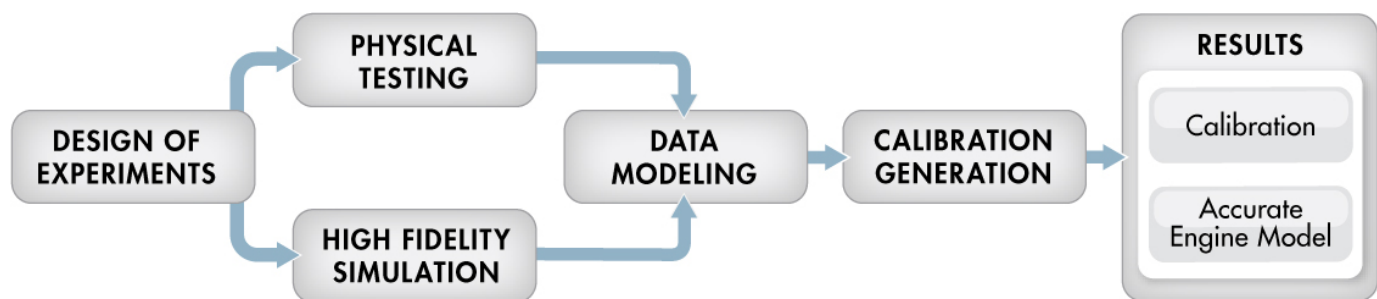
The aim of the calibration is to maximize torque at specific speed/load values across the engine's operating range, and meet these constraints:

- Limit knock (KIT1)
- Limit residual fraction
- Limit exhaust temperature
- Limit calibration table gradients for smoothness.

Turbocharger speed constraints are implicitly included in the boundary model which models the envelope.

The analysis must produce optimal engine calibration tables in speed and load for:

- Spark advance ignition timing (SA)
- Throttle position % (TPP)
- Turbo wastegate area % (WAP)
- Air/Fuel Ratio (Lambda: LAM)
- Intake cam phase (ICP)
- Exhaust cam phase (ECP)
- Torque (TQ)
- Brake-specific fuel consumption (BSFC)
- Boost (MAP)
- Exhaust temperature (TEXH)

View Optimization Results

1 Open MATLAB. On the **Apps** tab, in the **Automotive** group, click **MBC Optimization**.

2 In the CAGE Browser home page, in the **Case Studies** list, open **Dual CAM gasoline engine with spark optimized during testing**. Alternatively, select **File > Open Project** and browse to the example file `gasolineOneStage.cag`, found in `matlab\toolbox\mbc\mbctraining`.

3 In the **Processes** pane, click **Optimization**. Observe two optimizations, `Torque_Optimization` and `Sum_ Torque_Optimization`.

   Why are there two optimizations? To complete the calibration, you need to start with a point optimization to determine optimal values for the cam timings. You use the results of this point optimization to set up a sum optimization to optimize over the drive cycle and meet gradient constraints to keep the tables smooth.

4 In the Optimization pane, ensure `Torque_Optimization` is selected. In the Objectives pane, observe the description: maximize Torque as a function of ECP, ICP, LOAD and Speed.

5 In the Constraints pane, observe the constraints: the boundary model of Torque, knock, residual fraction, speed and exhaust temperature. If you want to see how these are set up, double-click a constraint and view settings in the Edit Constraint dialog box.

6 In the Optimization Point Set pane, observe the variable values defining the points at which to run the optimization.

7 To view the optimization results, in the Optimization pane expand the `Torque_Optimization` node and select `Torque_Optimization_Output`.
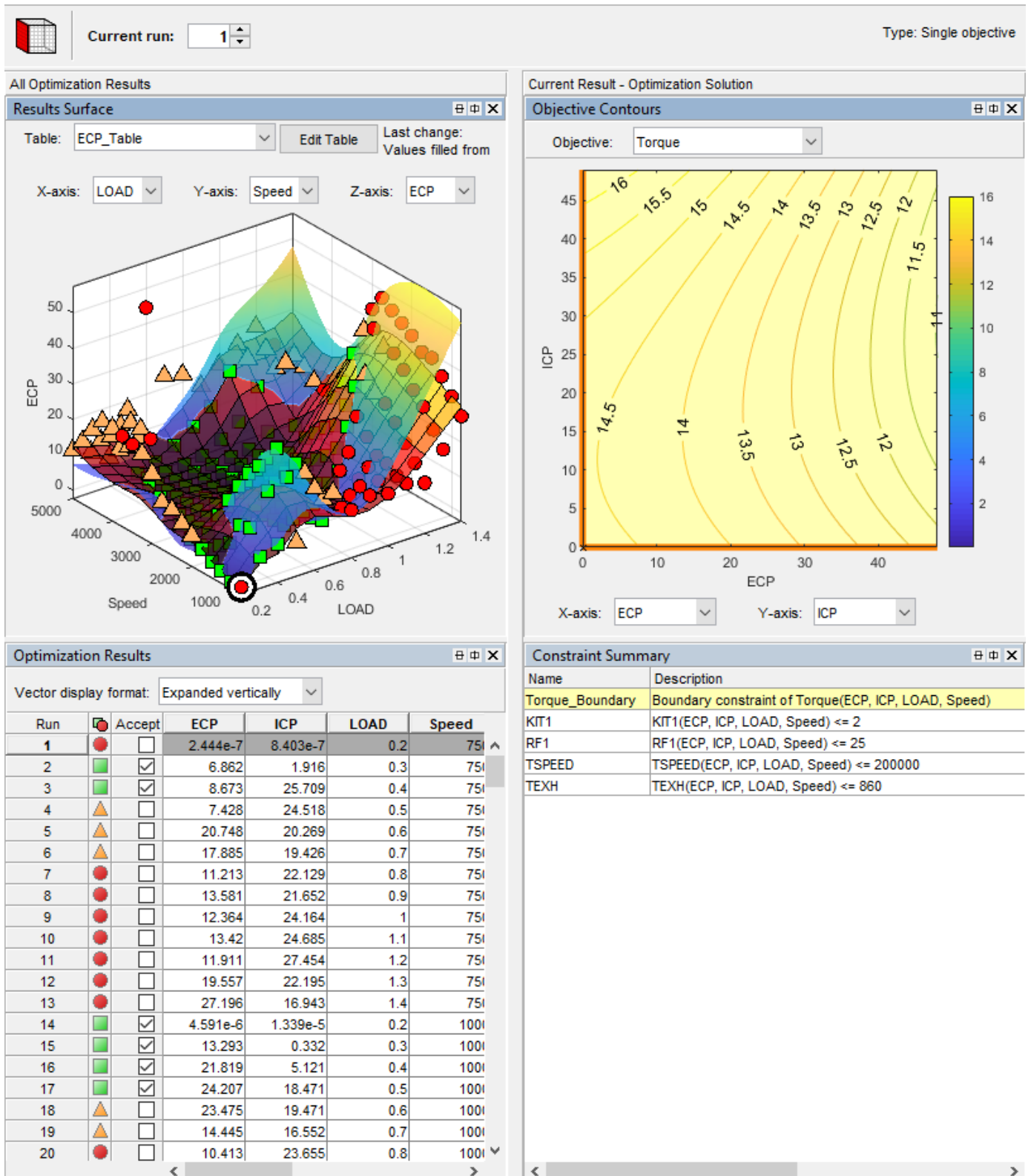
To learn about analyzing optimization results, see "Choosing Acceptable Solutions".

8 In the Optimization pane, select `Sum_Torque_Optimization` and compare with the previous point optimization setup.

Set Up Optimization

Learn how to set up this optimization.

1 To perform an optimization, you need to import the statistical models created in the Model Browser. To see how to import models, in CAGE, select **File > Import From Project**.

The CAGE Import Tool opens. Here, you can select models to import from a model browser project file or direct from the Model Browser if it is open. However, the toolbox provides an example file with the models already imported, so click **Close** to close the CAGE Import Tool.

2 To view the models, click **Models** in the left **Data Objects** pane.

3 You need tables to fill with the results of your optimizations. To see all the tables, select the **Tables** view.

4 To see how to set up these tables, select **Tools > Create Lookup Tables from Model**. The Create Lookup Tables from Model wizard appears.

a Select the model `Torque` and click **Next**.

b On the next screen you see the controls for specifying the inputs and size of tables. Click the **Edit breakpoints** buttons to see how to set up row and column values. Click **Next**.

   c On the next screen you can set limits on table values. To edit, double-click **Table Bounds** values.
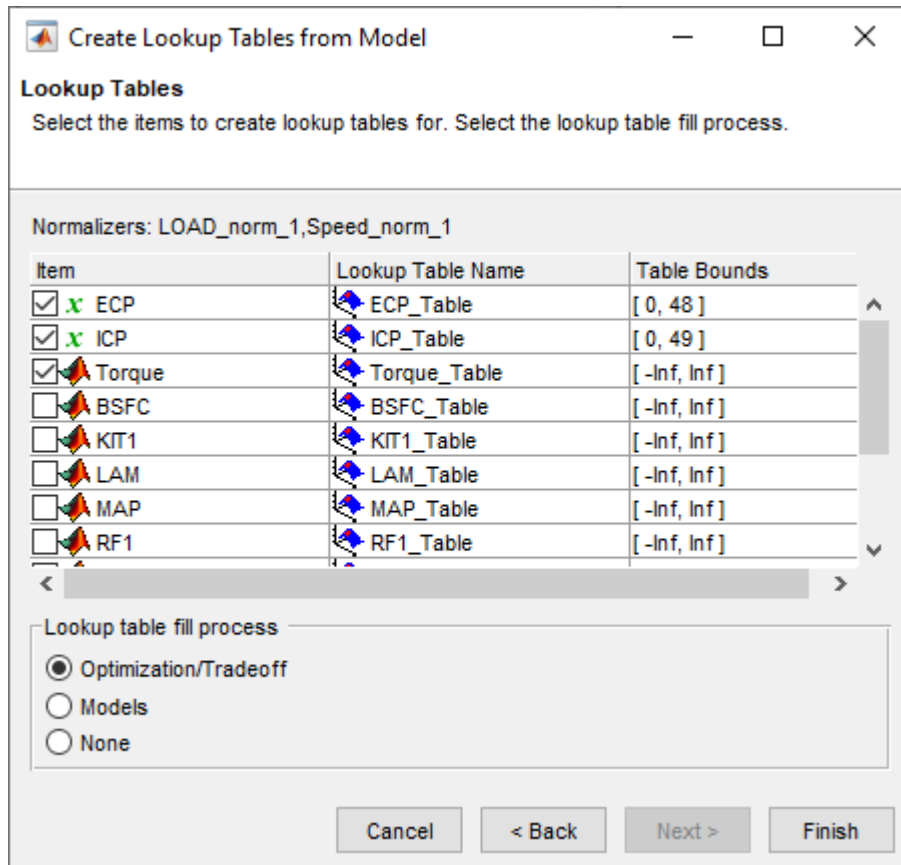
    Click **Cancel** to avoid creating new unnecessary tables in your example file.

5 Learn the easiest way to set up an optimization by selecting **Tools > Create Optimization from Model**.

   a Select the model `Torque` and click **Next**.

   b Observe that the default settings will create a point optimization to minimize `Torque`, using 4 free variables, and constrained by a model boundary constraint. To create this example optimization, edit the settings to maximize `Torque`, use the BSFC table breakpoints as a data source, and use only ECP and ICP as free variables. Click **Finish** to create the optimization.
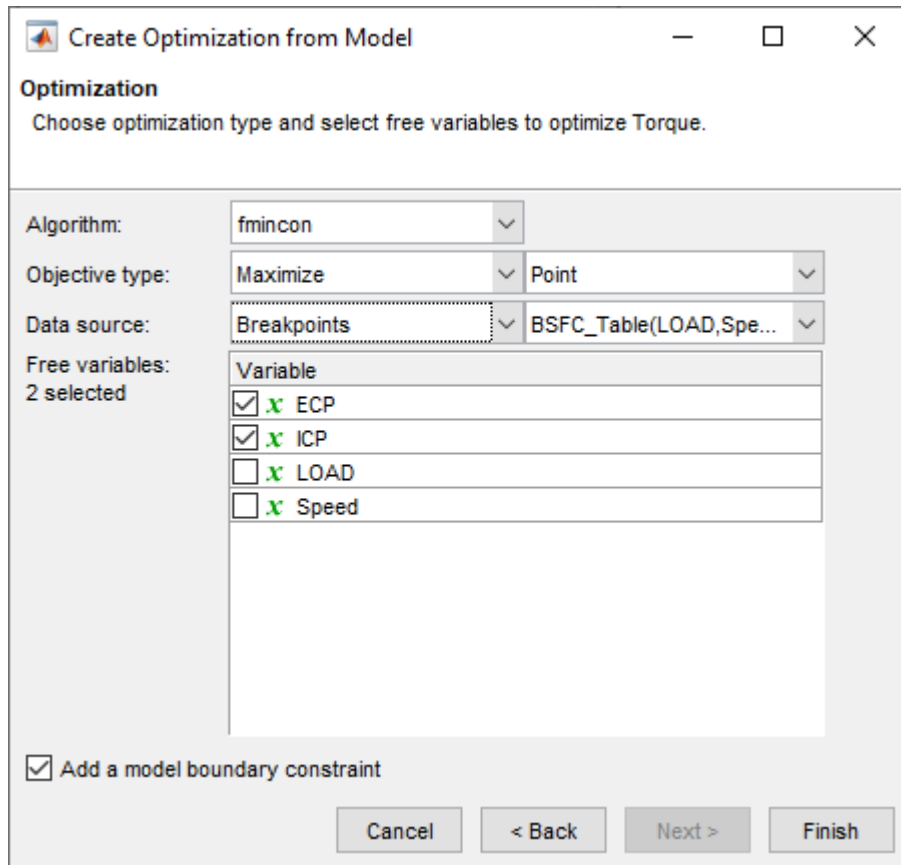
6 Compare your new optimization with the example `Torque_Optimization`. To finish the setup you need to add or import the rest of the constraints. In this case, select **Optimization > Constraints > Import Constraints**. Import the constraints from the `Torque_Optimization` optimization.

7 To learn how to set up the sum optimization, from the `Torque_Optimization_Output` node, select **Solution > Create Sum Optimization**. The toolbox creates a sum optimization for you.

8 Compare your new optimization with the example sum optimization, `Sum_Torque_Optimization`. The example shows you need to add the table smoothness constraints to complete the calibration. To see how to set up table gradient constraints, double-click the constraint `GradECP`.

To learn more about setting up optimizations and constraints, see:

- "Create Lookup Tables from a Model"
- "Creating Optimizations from Models"
- "Edit Objectives and Constraints"
- "Create Sum Optimization from Point Optimization Output"

.

Filling Tables From Optimization Results

CAGE remembers lookup table filling settings. To view how the example tables are filled:

1   Expand the example sum optimization node `Sum_Torque_Optimization` and select the `Sum_Torque_Optimization_Output` node.

2   Select **Solution > Fill Lookup Tables** to open the Lookup Table Filling from Optimization Results Wizard.

3   On the first screen, observe all the tables in the **CAGE tables to be filled list**. Note that the LAM (lambda) table is not filled from the optimization results, because this table was filled from the test data. Click **Next**.

4   On the second screen, observe all the tables are matched up with optimization results to fill them with. Click **Next**.

5   On the third screen, observe the lookup table filling options. You can either click **Finish** to fill all the tables, or **Cancel** to leave the tables untouched. The example tables are already filled with these settings.

The following plot shows similar calibration results for the ICP table. Observe that the idle region has locked cells to keep the cams parked at 0. If you want to lock values like this, to get smooth filled tables, lock the cells before filling from optimization results.
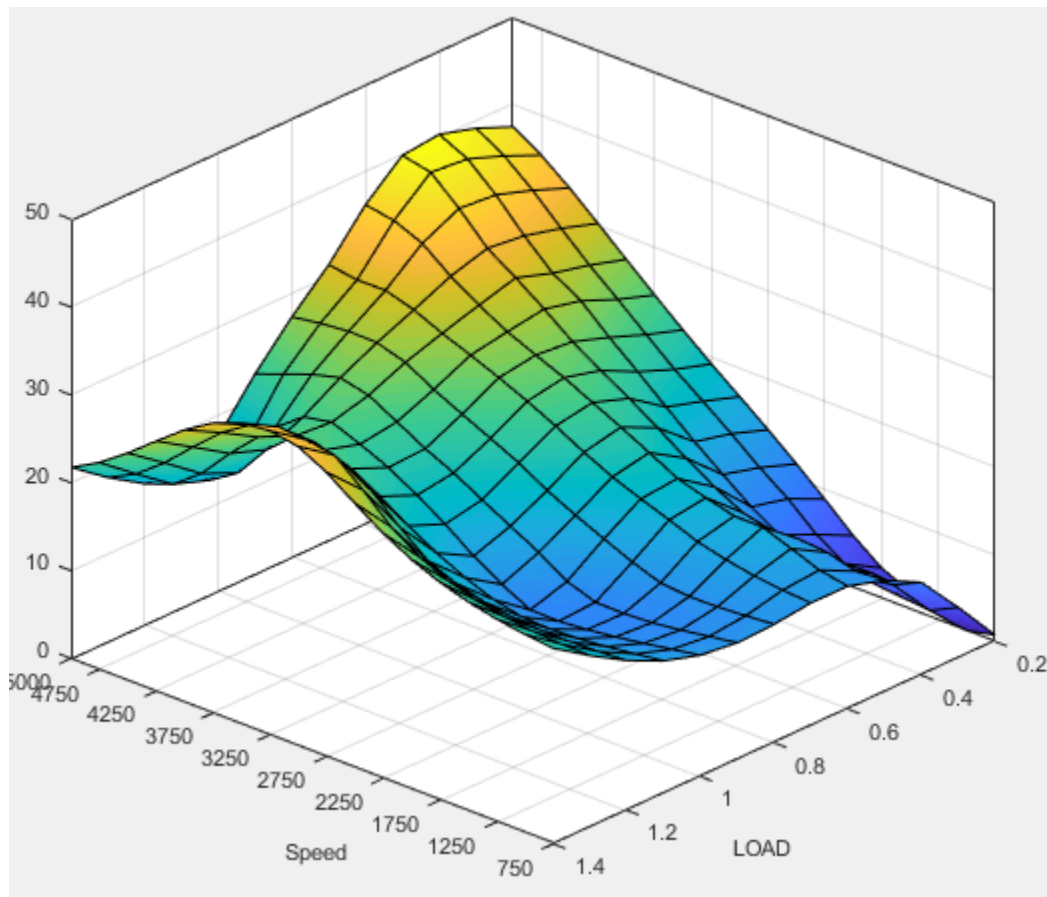
The following plot shows similar calibration results for the ECP table.
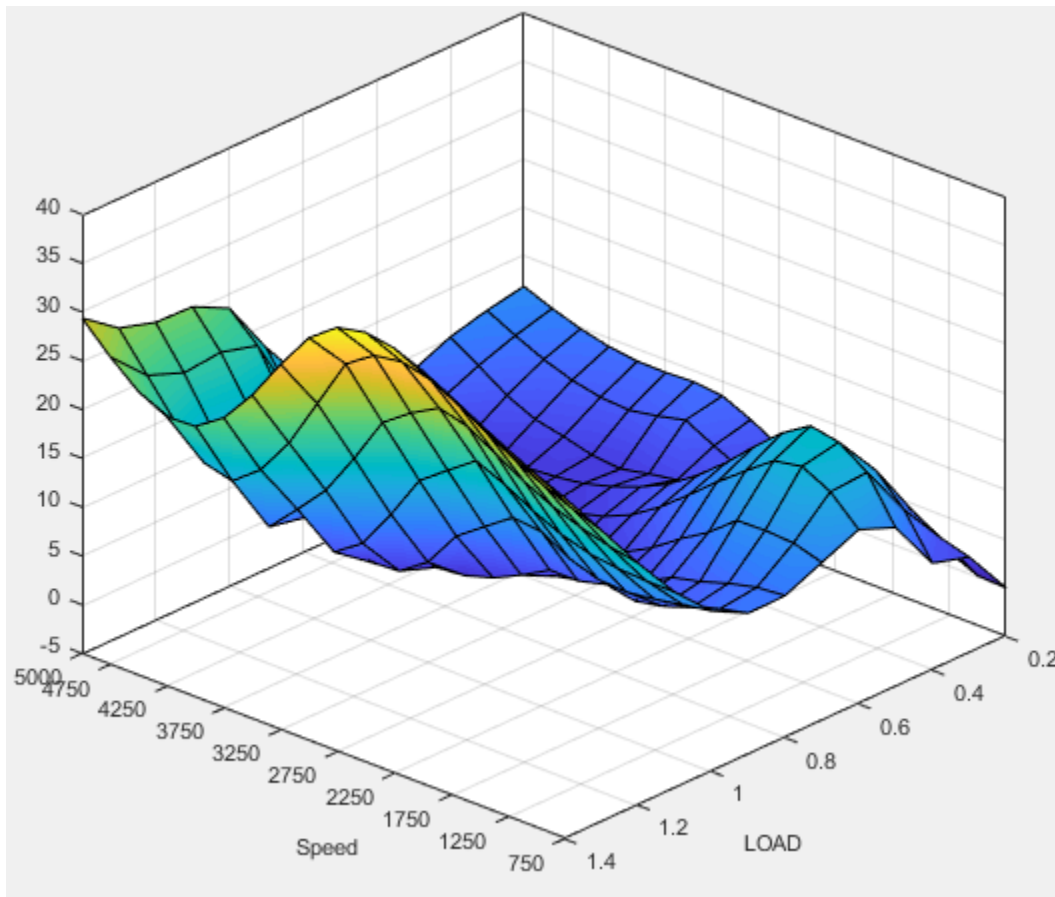
You can examine all the filled tables in the example project.

To learn more about analyzing and using optimization results, see "Optimization Analysis".

See Also

Related Examples
- "Create Lookup Tables from a Model"
- "Creating Optimizations from Models"
- "Edit Objectives and Constraints"
- "Choosing Acceptable Solutions"
- "Create Sum Optimization from Point Optimization Output"
- "Filling Tables from Optimization Results"

3

Design and Modeling Scripts

Introduction to the Command-Line Interface

The Model-Based Calibration Toolbox product is a software tool for modeling and calibrating powertrain systems. The command-line interface to the Model-Based Calibration Toolbox enables the design of experiments and modeling tools available in the toolbox to be accessible from the test bed.

You can use these commands to assemble your specific engine calibration processes into an easy to use script or graphical interface. Calibration technicians and engineers can use the custom interface without the need for extensive training. This system enables:

- Transfer of knowledge from the research and development engineers into the production environment
- Faster calibration
- Improved calibration quality
- Improved system understanding
- Reduced development time

Automate Design and Modeling With Scripts

In this section...
"Processes You Can Automate" on page 3-3
"Engine Modeling Scripts" on page 3-5

Processes You Can Automate

You can use these command-line functions to automate engine modeling processes.

Goal		Function
Create or load a project		• `CreateProject` • `Load`
Create a new test plan for the project using a template		`CreateTestplan`
Create designs that define data points to collect on the test bed		`CreateDesign`
Work with classical, space-filling or optimal designs		• `CreateConstraint` • `CreateCandidateSet` • `Generate` • `FixPoints` • `Augment`
Create or load a data object for the project and make it editable		• `CreateData` • `BeginEdit`
Load data from a file or the workspace		• `ImportFromFile` • `ImportFromMBCDataStructure`
Work with data	Examine	`Value`
	Modify	• `AddFilter` • `AddTestFilter`
	Add variables	`AddVariable`
	Add	`Append`
	Group	• `DefineTestGroups` • `DefineNumberOfRecordsPerTest`
	Export	`ExportToMBCDataStructure`
Save your changes to the data, or discard them		• `CommitEdit` • `RollbackEdit`
Designate which project data object to use for modeling in your test plan		`AttachData`
Create and evaluate boundary models, either in a project or standalone		"Boundary Model Scripting" on page 3-7

Goal		Function
Create models for the data; these can be one- or two-stage models and can include datum models		`CreateResponse`
Work with your models	Examine input data and response data	• `DoubleInputData` • `DoubleResponseData`
	Examine predicted values at specified inputs	• `PredictedValue` • `PredictedValueForTest`
	Examine Predicted Error Variance (PEV) at specified inputs	• `PEV` • `PEVForTest`
	Examine and remove outliers	• `OutlierIndices` • `OutlierIndicesForTest` • `RemoveOutliers` • `RemoveOutliersForTest` • `RestoreData`
	Create a selection of alternative models	`CreateAlternativeModels`
	Choose the best model by using the diagnostic statistics	• `AlternativeModelStatistics` • `DiagnosticStatistics` • `SummaryStatistics`
	Extract a model object from any response object	• `Model Object` • `fit` • `CreateModel` • `ModelSetup` • `Type (for models)` • `Properties (for models)` • `CreateAlgorithm` • `StepwiseRegression` • `Jacobian` • `ParameterStatistics` • `UpdateResponse`
For two-stage test plans, once you are satisfied with the fit of the local and response feature models, calculate the two-stage model		`MakeHierarchicalResponse`
Export models to MATLAB or Simulink		`Export`

Engine Modeling Scripts

To open examples scripts, in the Help browser, select the **Examples** tab. Run the scripts to learn about:

- Loading and Modifying Data
- Designing experiments and constraining designs
- Gasoline engine modeling script to automatically generate a project for the gasoline case study, including:

  - Grouping and filtering data
  - Boundary modeling
  - Response modeling
  - Removing outliers and copying outlier selections
  - Creating alternative models and selecting the best based on statistical results

- Point-by-point diesel engine modeling to automatically generate a project for the diesel case study, including:

  - Defining engine operating points
  - Creating designs for each operating point
  - Augmenting designs to collect more data
  - Building a point-by-point boundary model
  - Create response models using the local multiple model type

See Also

More About

- "Automation Scripting"

Understanding Model Structure for Scripting

In this section...
"Projects and Test Plans for Model Scripting" on page 3-6
"Response Model Scripting" on page 3-6
"Boundary Model Scripting" on page 3-7

Projects and Test Plans for Model Scripting

To use the Model Browser in the Model-Based Calibration Toolbox product, you must understand the structure and functions of the model tree to navigate the views. To use the command-line version of the toolbox, you must understand the same structure and functions, that is, how projects, test plans, and models fit together. The following sections describe the relationship between the different models that you can construct. The diagrams in the following section, "How the Model Tree Relates to Command-Line Objects" on page 3-9, illustrate these relationships.

- Projects can have one or more test plans.
- Projects can have one or more data objects.

- Test plans have no more than one data object.
- Test plans have response objects.

  - If a one-stage test plan, these are simply known as responses.
  - If two-stage test plan, these are hierarchical responses.
- Test plans have boundary tree objects.

Response Model Scripting

A response is a model fitted to some data. These are the types of responses:

- Hierarchical Response (Level 0)

  A hierarchical response (also known as a two-stage response) models a `ResponseSignalName` using a local response and one or more response features.

  A hierarchical response has one or more different local responses (accessible via the property `LocalResponses`) that provide different possible models of the `ResponseSignalName`. One of these must be chosen as the best, and that will then be the local response used subsequently. The response features of each of the local responses are available directly from those local response objects.
- Local Response (Level 1)

  The local response consists of models of the `ResponseSignalName` as a function of the local input factors. The local input factors are accessible via the `InputSignalNames` property.

  A local response has one or more response features (accessible via the property `ResponseFeatures`) containing the models fitted to those response features of the local model.
- Response (Level 1 or 2)

- For two-stage test plans, response objects model the response features of local responses (`ResponseSignalName` corresponds to the name of the response feature). In this case, the response has a level value of 2.
- For one-stage test plans, response objects simply model the `ResponseSignalName` as a function of the input factors. In this case, the response will have a level value of 1.

All responses can have zero or more alternative responses (accessible via the property `AlternativeResponses`) that provide different possible models of the `ResponseSignalName`. These all retain the same level as the response for which they are an alternative. One of these must be chosen as the best and that will then be the response used subsequently.

See the illustrations in the following section, "How the Model Tree Relates to Command-Line Objects" on page 3-9, for examples of different responses and how they relate to each other.

Note that each response contains a model object (`mbcmodel.model`) that can be extracted and manipulated independently of the project. You can change the model type and settings, fit to new data, examine coefficients, regression matrices and predicted values, and use stepwise functions to include or remove terms. You can change model type, properties and fit algorithm settings. To learn about what you do with a model object, see `Model Object`. If you change the model, you must use `UpdateResponse` to replace the new model type in the response object in the project. When you use `UpdateResponse` the new model is fitted to the response data. See `UpdateResponse`.

Boundary Model Scripting

You can create and evaluate boundary models either in a project or standalone. You can combine boundary models in the same way as when using the Boundary Editor GUI. You can use boundary models as design constraints.

In a project, the test plan has a `Boundary` property that can contain an `mbcboundary.Tree` object.

```
BoundaryTree = mbcmodel.testplan.Boundary
```

The `BoundaryTree` is a container for all the boundary models you create. The tree is empty until you create boundaries, and if you change the testplan data the toolbox deletes the boundaries.

You can fit boundary models in `mbcmodel` projects using the boundary tree class `mbcboundary.Tree`, or you can fit boundary models directly to data.

To create a boundary model outside of a project, you can either:

- Use the `CreateBoundary` package function:

  ```
  B = mbcboundary.CreateBoundary(Type,Inputs)
  ```
- Use the `fit` method to create and fit a boundary to some data X:

  ```
  B = mbcboundary.Fit(X,Type)
  ```

To create a boundary model within a project, use the `CreateBoundary` method of the boundary tree:

```
B = CreateBoundary(Tree,Type)
```

This creates a new boundary model, B, from the `mbcboundary.Tree` object, `Tree`. The test plan inputs are used to define the boundary model inputs. The new boundary model is not added to the tree, you must call `Add`.

To create a new boundary model from an existing boundary model, you can use the `CreateBoundary` method of all boundary model types:

```
B = CreateBoundary(B,Type)
```

You can combine boundary models by using the `InBest` property of the boundary tree. This corresponds to combining boundary models in best in the Boundary Editor GUI, as described in "Combining Best Boundary Models" in the Model Browser documentation. You can also combine boundary models with logical operators, for use as design constraints or outside projects.

You can change the `ActiveInputs`, `Evaluate`, and use as a `designconstraint`.

How the Model Tree Relates to Command-Line Objects

The tree in the Model Browser displays the hierarchical structure of models. This structure must be understood to use the command-line interface. The following examples illustrate the relationship between projects, test plans and responses in one-stage and two-stage models.

The following is an example of a two-stage model tree.
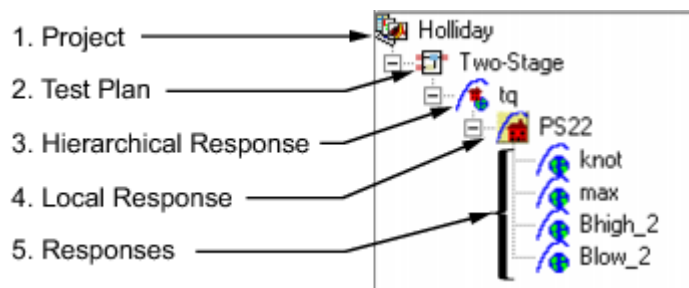
The elements of the tree correspond to the following objects in the command-line interface:

1   Project

2   Test Plan

3   Hierarchical Response

4   Local Response

5   Responses

The following example illustrates a project containing a one-stage test plan; in the command-line interface this corresponds to a project, one-stage test plan, and a response model.

Hierarchical responses can have multiple local responses, as shown in the following example from the Model Browser. In the command-line interface these are accessible via the property `LocalResponses` for a hierarchical response object (`mbcmodel.hierarchicalresponse`). In this example, the local responses are `PS22`, `PS32`, and `POLY2`.

Only one of these local responses can be chosen as best (in this example, `PS22`, indicated by the blue icon) and used to construct the hierarchical response, together with the associated response features of the local response. Each local response object has a set of responses, accessible by the property `ResponseFeatures(Local Response)`.
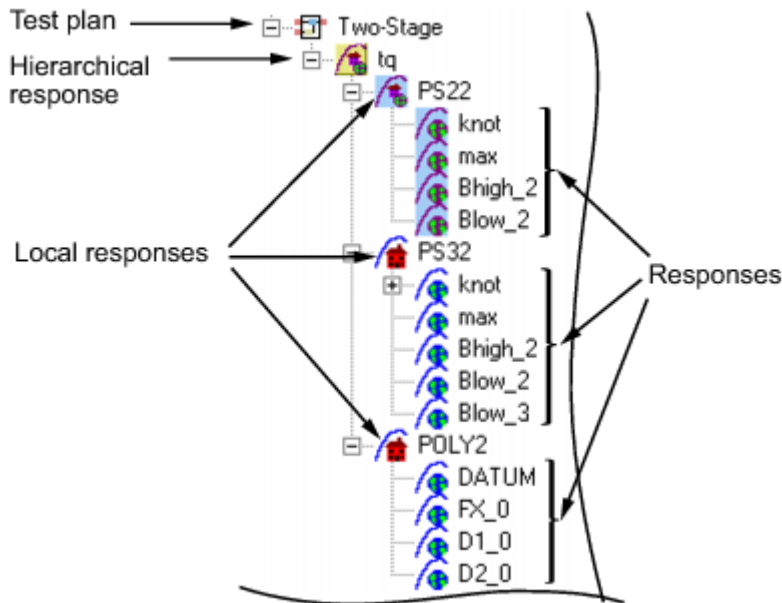
Responses can have zero or more alternative responses, as shown in the following model tree. You call the method `CreateAlternativeModels` on the command line to do the same.
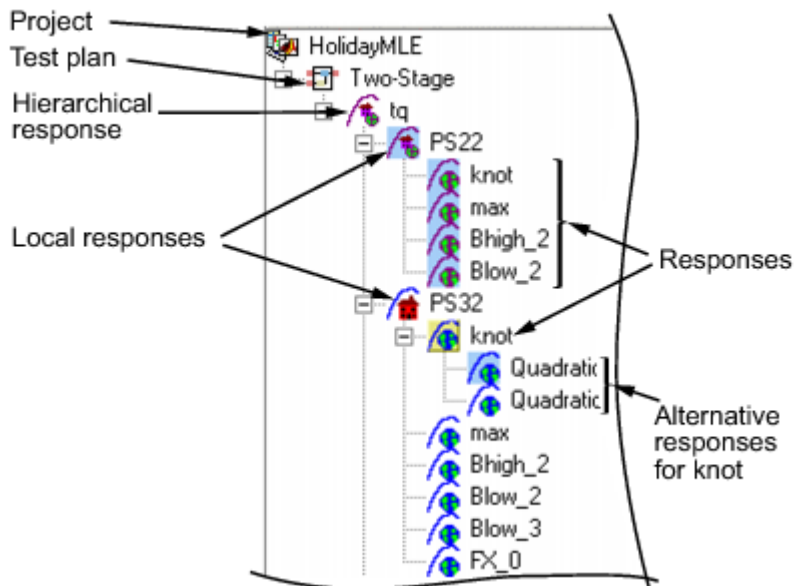
In this example, the alternative responses for the `knot` response are accessible via the property `AlternativeResponses`. You can create alternative responses for any response (including all one-stage responses).

You can use model templates to try alternative model types for several responses. The following example shows the results of using a model template for four alternative responses (`Linear-RBF`, `RBF-multiquadric`, `Cubic`, and `Quadratic`). The model template has been used to create alternative responses for the responses `knot` and `max`. You can call the method `CreateAlternativeModels` to do this in the command-line interface.
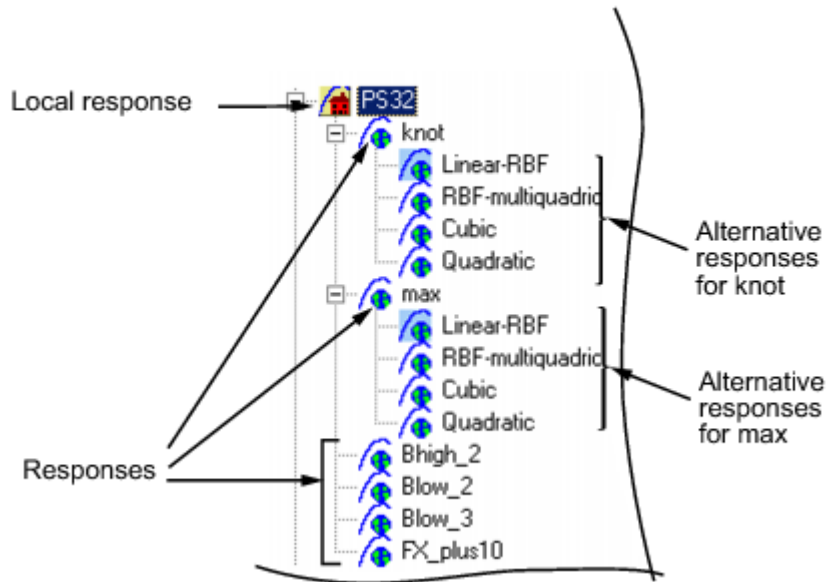
One of the alternative responses must be chosen as best for each response (call the method `ChooseAsBest`). In this example, when `Linear-RBF` is chosen as best from the alternatives for the `knot` response, then it is copied to `knot`.

4

Multi-Injection Diesel Calibration

Multi-Injection Diesel Calibration Workflow

In this section...
"Multi-Injection Diesel Problem Definition" on page 4-2
"Engine Calibration Workflow" on page 4-5
"Air-System Survey Testing" on page 4-6
"Multi-Injection Testing" on page 4-7
"Data Collection and Physical Modeling" on page 4-7
"Statistical Modeling" on page 4-8
"Optimization Using Statistical Models" on page 4-9
"Case Study Example Files" on page 4-14

Multi-Injection Diesel Problem Definition

This case study shows how to systematically develop a set of optimal steady-state engine calibration tables using Model-Based Calibration Toolbox.

The engine to be calibrated is a 3.1L multi-injection combustion ignition engine with common rail, variable-geometry turbocharger (VGT), and cooled exhaust gas recirculation (EGR).

The aim of the calibration is to minimize brake-specific fuel consumption (BSFC) at specific speed/load operating points across the engine's operating range, and meet these constraints:

- Limit total NOx emissions.
- Limit maximum turbocharger speed.
- Limit calibration table gradients for smoothness.

The analysis must produce optimal calibration tables in speed and torque for:

- Best main start of injection timing
- Best total injected fuel mass per cylinder per cycle
- Best pilot injection timing relative to main timing
- Best pilot injection fuel mass fraction of total injection mass
- Best exhaust gas recirculation (EGR) position
- Best variable geometry turbocharger (VGT) vane position
- Best fuel rail pressure relative to nominal pressure vs. engine speed

These sections explain the objectives of selecting best values for these calibration tables and the effects of these control variables on the engine:

- "Select Main Injection Timing for Efficiency" on page 4-3
- "Select Pilot Injection Timing to Control Noise" on page 4-4
- "Select Main Fuel Mass for Efficiency and Emissions" on page 4-4
- "Select Fuel Pressure for Efficiency and Emissions" on page 4-4
- "Select the Turbocharger Position to Control Air-Charge and EGR" on page 4-5

- "Select the EGR Valve Position to Control Air-Charge and Emissions" on page 4-5

Select Main Injection Timing for Efficiency

You select the injection timing of the main fuel injection to maximize engine efficiency. You aim to make peak cylinder pressure occur slightly after piston top center. You inject fuel just before top-center compression, as shown.
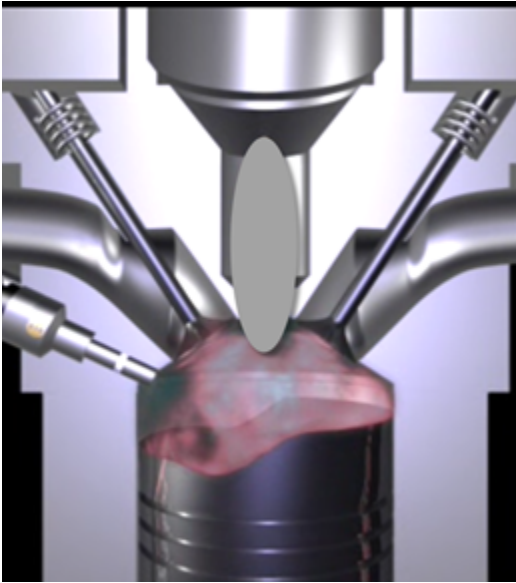
Then you can achieve peak combustion pressure just after top-center expansion.

You also need to adjust injection timing according to speed and other conditions.

- You need to advance (move earlier before piston top center) the start of injection timing with increasing speed and dilution (exhaust gas recirculation or EGR).

- You need to retard the start of injection timing with increased fresh air intake (load).

Select Pilot Injection Timing to Control Noise

You select the timing of the pilot fuel injection to start combustion early before the larger main fuel injection. The pilot fuel injection occurs well before top-center compression and before the main injection.

You can use pilot fuel injection to control combustion noise, because it affects the variability in cylinder pressure.

In this example, pilot fuel injection timing is defined as a crank-angle delta offset before the main injection, and is therefore a relative quantity.

Select Main Fuel Mass for Efficiency and Emissions

The air-fuel ratio (AFR) affects engine efficiency and emissions. A rich AFR causes high engine-out particulates and low engine-out NOx. You control AFR by changing the main fuel mass for optimal balance between power and emissions.

The AFR of the combustion mixture is determined by the main fuel injection mass for a given amount of fresh air. The amount of air results mainly from EGR valve position, VGT position, intake throttle position, and speed.

Select Fuel Pressure for Efficiency and Emissions

You can use fuel pressure to control fuel droplet size. Reduced fuel droplet size in the combustion chamber reduces particulates, releases more energy from the fuel, and achieves more stable combustion. High fuel pressure decreases fuel droplet size to improve efficiency and emissions.

At low loads, you can use lower fuel pressure to decrease fuel pump power losses without much effect on emissions and power efficiency.

In this example, fuel pressure is controlled relative to an engine-speed-dependent base level via a fuel pressure delta, and is therefore a relative quantity.

Select the Turbocharger Position to Control Air-Charge and EGR

You can use the variable-geometry turbocharger (VGT) position to balance fresh air and exhaust gas recirculation for optimal NOx control at a given power level.

You can change VGT vane position to increase cylinder fresh air due to the turbocharger speed increase. With the vanes closed, the turbocharger moves faster (high VGT speed) and sends a higher load (or boost) of air into the engine. Closing the vanes also increases exhaust gas recirculation (EGR) due to increased backpressure from the closed vanes.
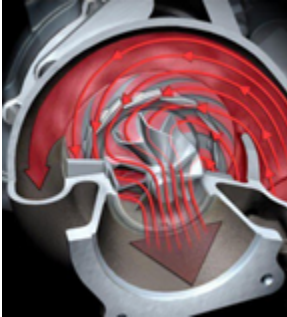
With the vanes open, VGT speed is low and passes through low load (or boost) to the engine.

Select the EGR Valve Position to Control Air-Charge and Emissions

You can use the EGR valve position to control the flow of burned exhaust gases back to the intake manifold.

Reburning exhaust gases decreases in-cylinder temperature, resulting in a significant decrease in NOx emissions.

If you select too much EGR for a given amount of injected fuel, then the air-fuel ratio will be rich, causing increased soot emissions. Therefore, you must balance these competing objectives.

In engines, timing is everything. Using the EGR valve and all the other control variables, controlling the engine's air flow is the key to optimizing fuel economy, reducing emissions, and increasing power density.

Engine Calibration Workflow

The following graphic illustrates the workflow for the model-based calibration process. The workflow can use a combination of tools: Model-Based Calibration Toolbox, Simulink, Stateflow, third-party high-fidelity simulation tools, and Hardware-in-the-Loop testing for fine tuning calibrations on ECUs.

MBC Toolbox	Simulink/Stateflow	GTPOWER
Design of Experiments	Automated Virtual Engine Mapping	High Fidelity Engine Model

MBC Toolbox	MBC Toolbox	ECU Calibrations
Model Fitting	Calibration Generation	
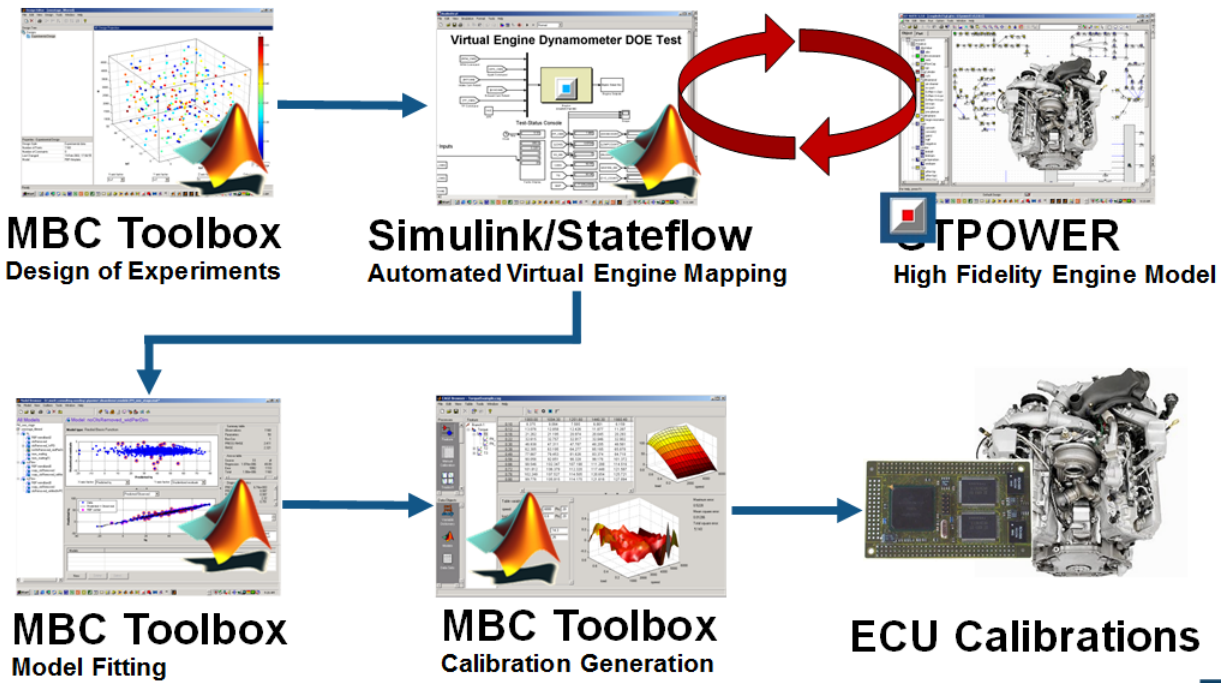
Air-System Survey Testing

The first step to solve this calibration problem is to determine the boundaries of the feasible air-system settings. To do this, you create an experimental design and collect data to determine air-system setting boundaries that allow positive brake torque production in a feasible AFR range.
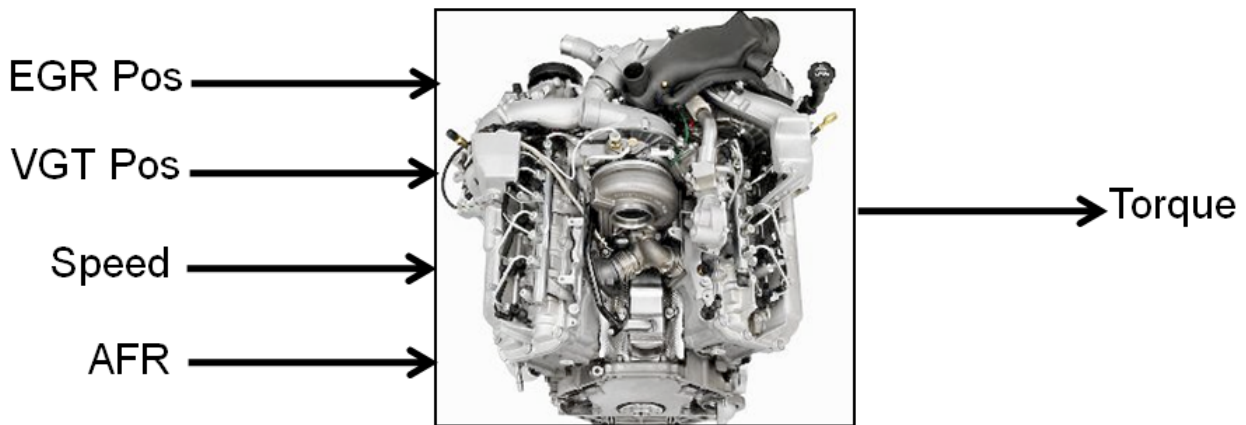
These simplifications were used to conduct the initial study:

- Pilot injection is inactive.
- Main timing is fixed.
- Nominal fuel pressure vs RPM.

- Main fuel mass is moved to match the AFR target.

Fit a boundary model to these design points.

Multi-Injection Testing

After the air-system survey, you have established the boundaries of positive brake torque air-system settings. Now, you can create an experimental design and collect data to gather fuel injection effects within those boundaries. You can then use this data to create response models for all the responses you need to create an optimal calibration for this engine.
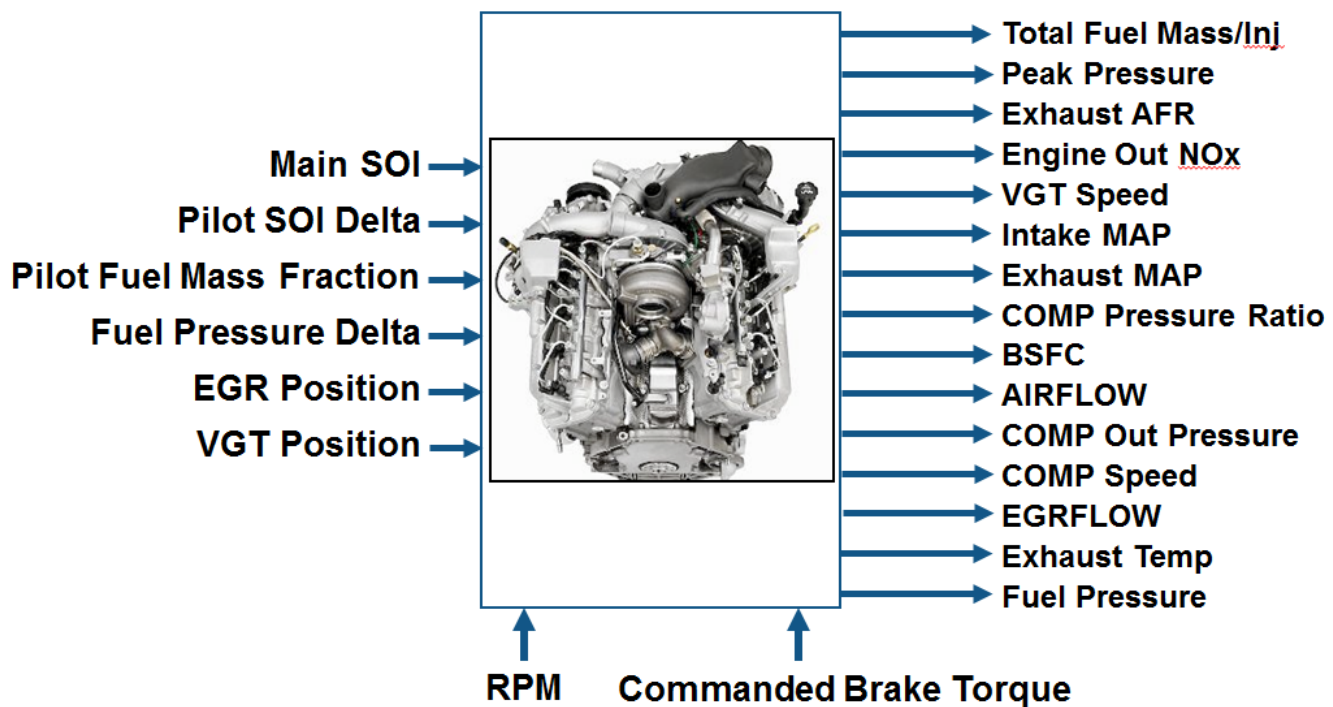
Data Collection and Physical Modeling

The toolbox provides the data for you to explore this calibration example.

MathWorks collected the data using simulation tools. Control and simulation models were constructed using Simulink and Stateflow. Constrained experimental designs were constructed using Model-Based Calibration Toolbox. The points specified in the design were measured using the GT-Power engine simulation tool from Gamma Technologies (see https://www.gtisoft.com).

To collect the data, Simulink and Stateflow controlled the torque output of the GT-Power engine model to the desired Design of Experiments points using total fuel mass. This graphic shows the virtual dynamometer test model.
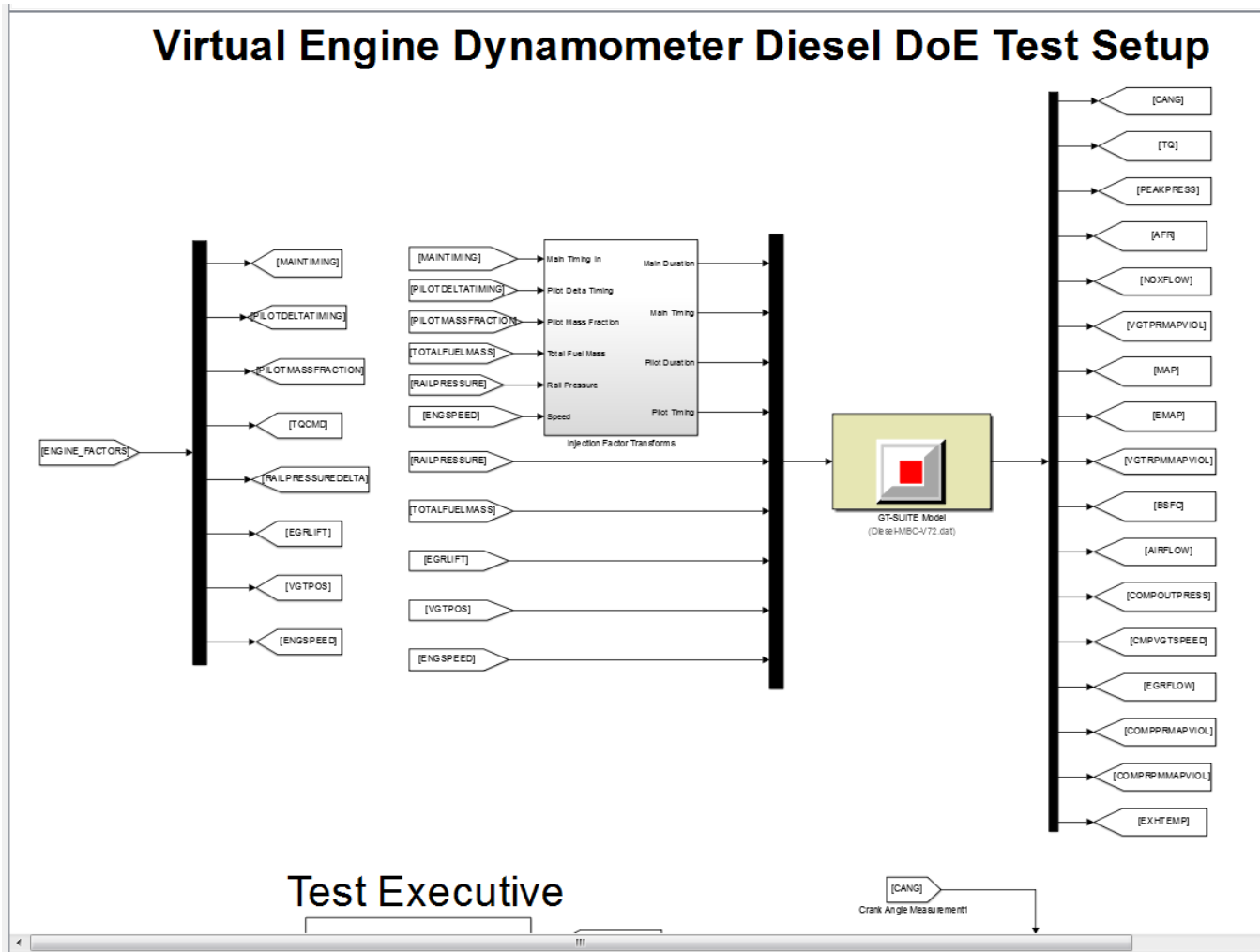
Statistical Modeling

After designing the experiments and collecting the data, you can fit statistical models to the data. You can use the toolbox to generate accurate, fast-running models from the measured engine data.

The following graphic shows the models to define in the toolbox to solve this calibration problem. The graphic shows how the model inputs and output relate to the optimal tables, optimization operating points, objectives and constraints you need to perform the optimization and create the calibration.

I/O of Multi-Inject 3.1L Common Rail Engine Model with Variable Geometry Turbocharger and Cooled EGR
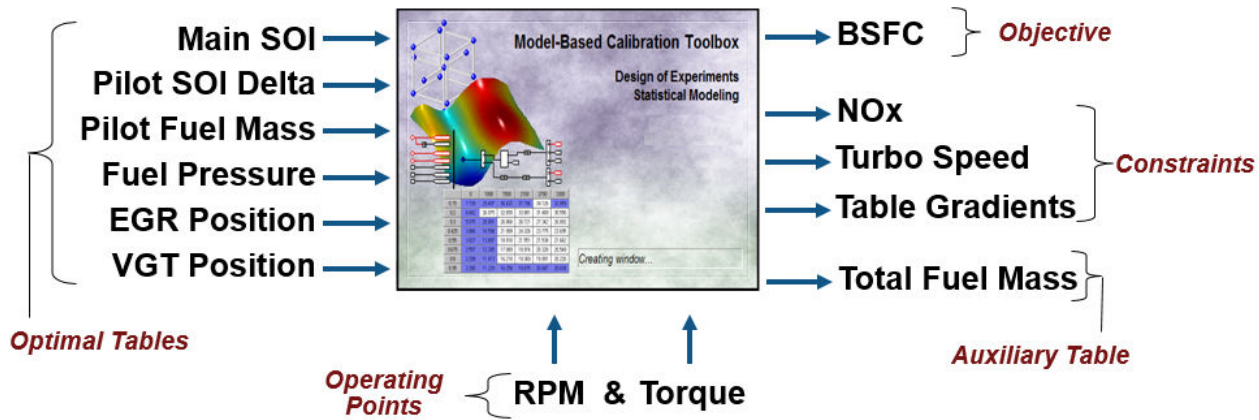
Goal: Minimize BSFC subject to NOx, turbocharger speed, and user-specified table gradient constraints

Optimization Using Statistical Models

After creating statistical models to fit the data, you can use them in optimizations. You can use the accurate statistical engine model to replace the high-fidelity simulation and run much faster, enabling optimizations to generate calibrations.

1   Run an optimization to choose whether to use Pilot Injection at each operating point.

2   Optimize fuel consumption over the drive cycle, while meeting these constraints:

   - Constrain total NOx
   - Constrain turbocharger speed
   - Constrain smoothness of tables

3   Fill lookup tables for all control inputs.

The following plots show a preview of the calibration results.
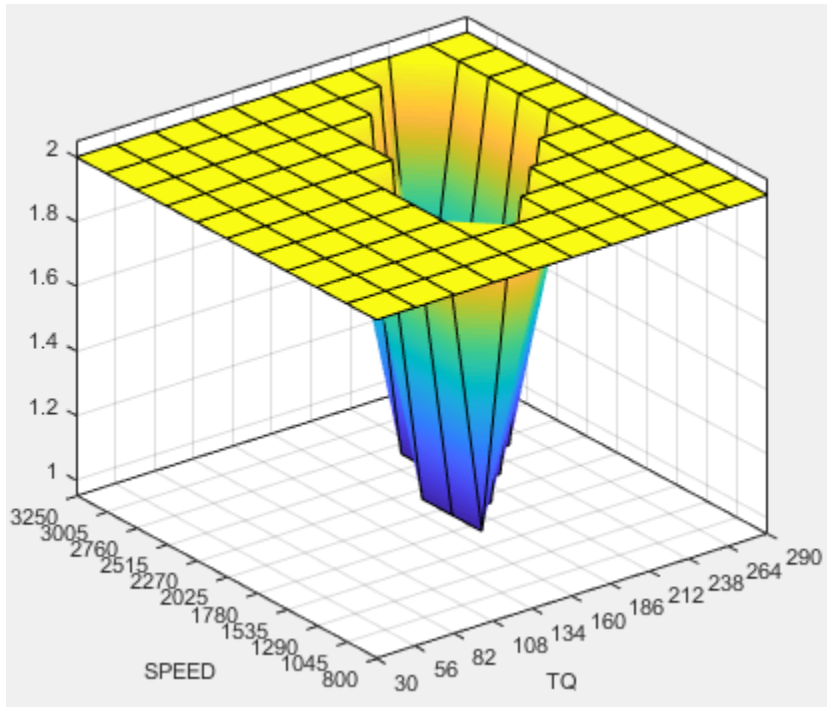
.

Pilot Mode Table

This graphic shows the plot of the table to select the active or inactive pilot mode depending on the speed and commanded torque

You need to fill calibration tables for each control variable described in "Multi-Injection Diesel Problem Definition" on page 4-2, in both pilot modes, active and inactive.

Following are all the pilot active tables.
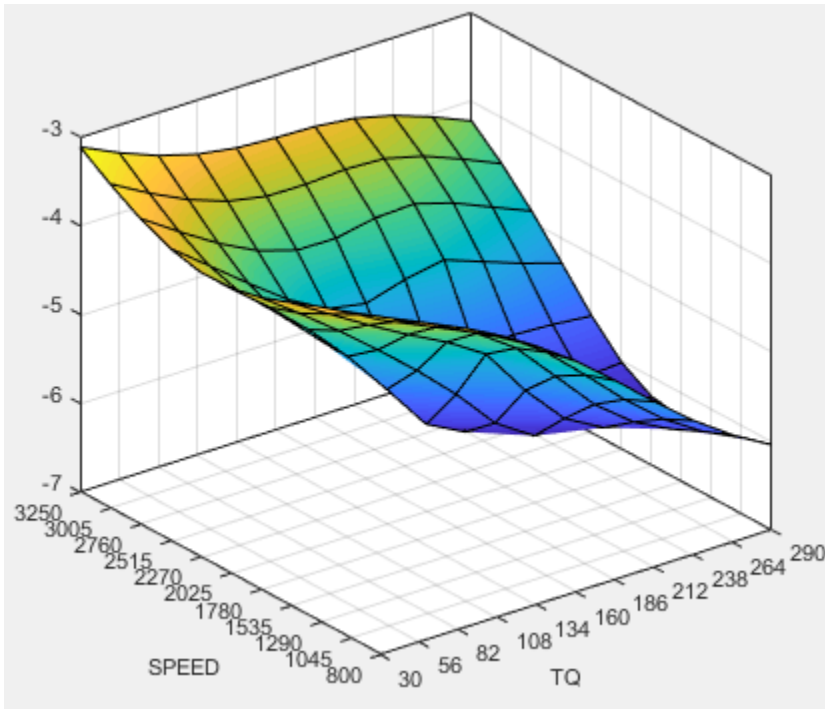
Main Start of Injection (SOI) Timing Table
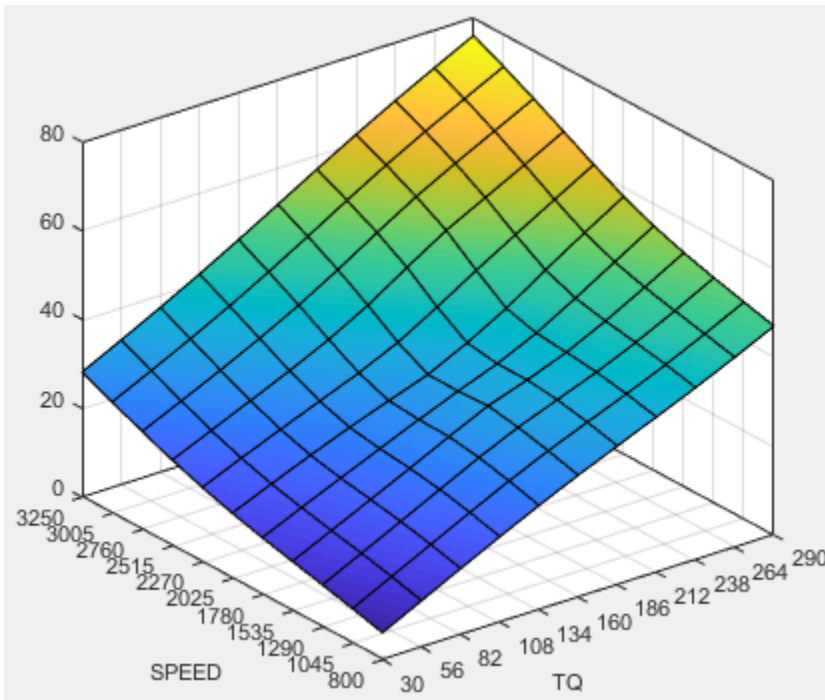
Total Injected Fuel Mass Table
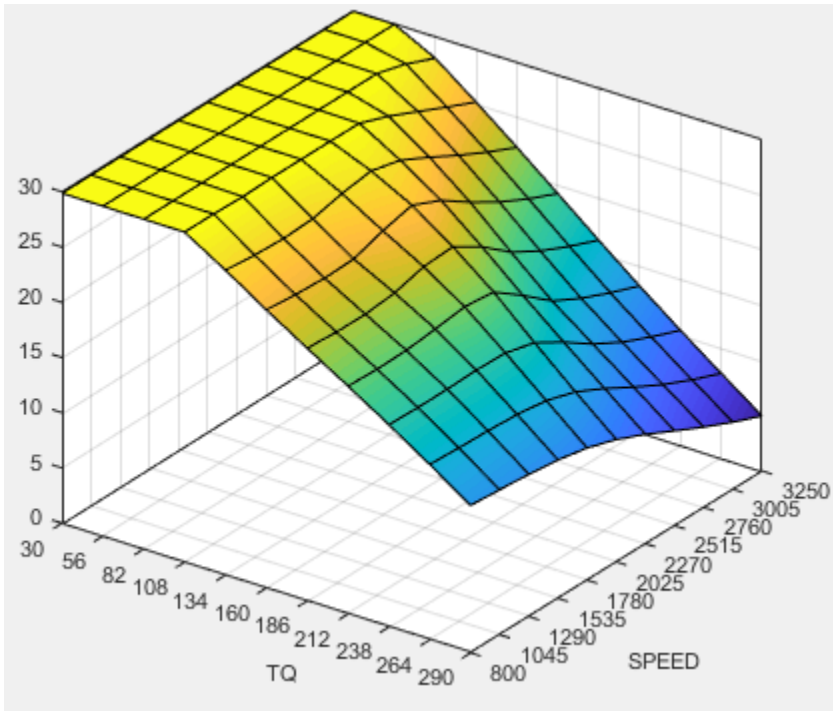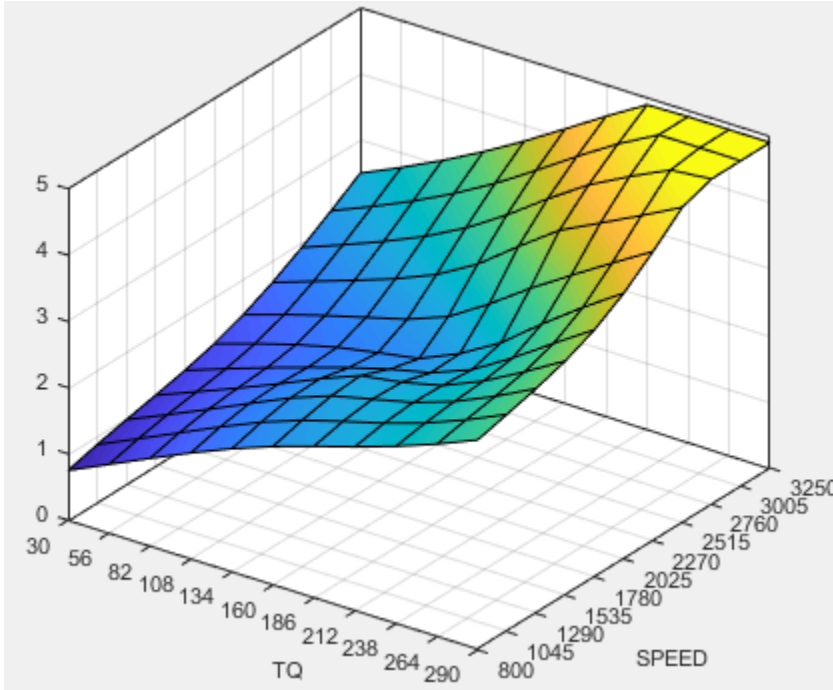
Fuel Pressure Delta Table

Exhaust Gas Recirculation (EGR) Valve Position Table
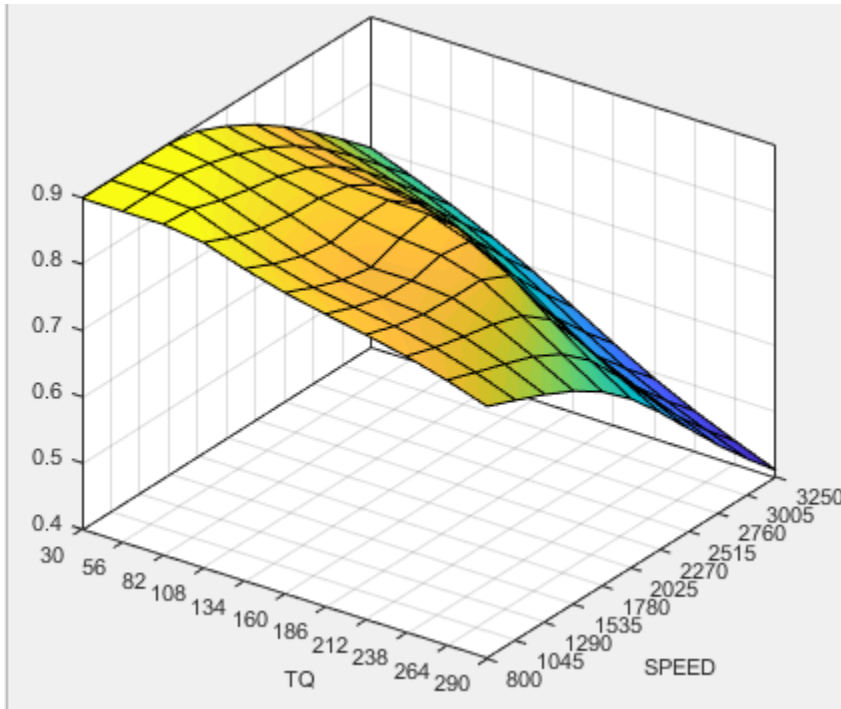
Variable-Geometry Turbo (VGT) Position Table
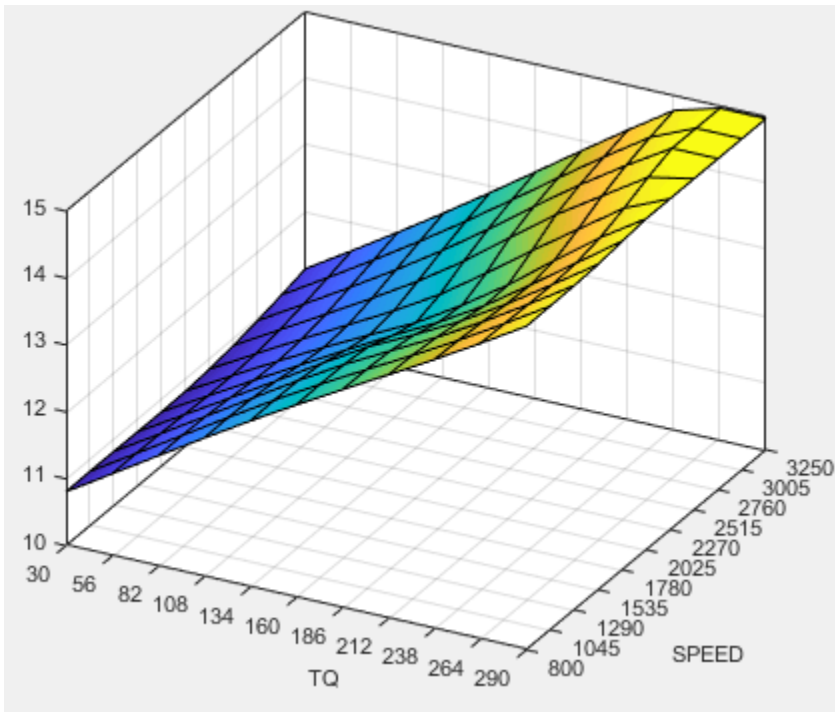
Pilot Injection Timing (Pilot SOI Delta) Table
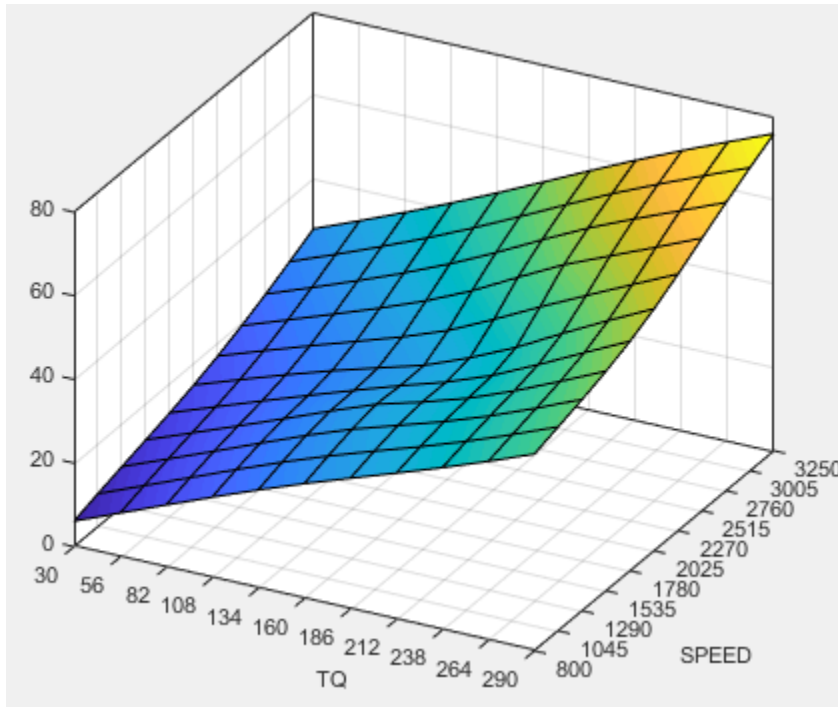
Pilot Fuel Mass Fraction Table

Case Study Example Files

The following sections guide you through opening example files to view each stage of the model-based calibration process. You can examine:

1   Designs, constraints, boundary model, and collected data, in topic "Design of Experiment" on page 4-15.
2   Finished statistical models, in topic "Statistical Modeling" on page 4-25.
3   Optimization setup and results, and filled calibration tables, in topic "Optimization" on page 4-29.

Use these example files to understand how to set up systematic calibrations for similar problems. For next steps, see "Design of Experiment" on page 4-15.

Tip Learn how MathWorks Consulting helps customers develop engine calibrations that optimally balance engine performance, fuel economy, and emissions requirements: see Optimal Engine Calibration.

Design of Experiment

Benefits of Design of Experiment

You use design of experiment to efficiently collect engine data. Testing time (on a dyno cell, or as in this case, using high-fidelity simulation) is expensive, and the savings in time and money can be considerable when a careful experimental design takes only the most useful data. Dramatically reducing test time is increasingly important as the number of controllable variables in more complex engines is growing. With increasing engine complexity, the test time increases exponentially.

Air-System Survey Testing

The first stage to solve this calibration problem is to determine the boundaries of the feasible air-system settings. To do this, create an experimental design and collect data to determine air-system setting boundaries that allow positive brake torque production in a feasible AFR range.
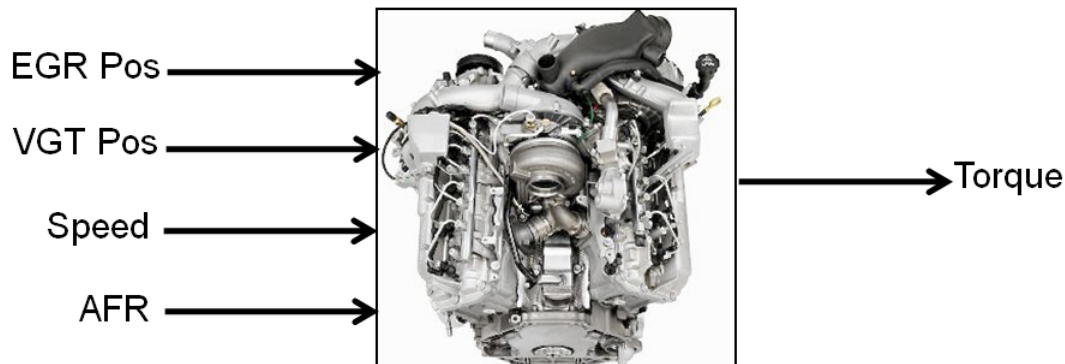
These simplifications were used to conduct the initial study:

- Pilot injection is inactive.
- Main timing is fixed.
- Nominal fuel pressure vs RPM.
- Main fuel mass is moved to match the AFR target.

The design process follows these steps:

1 Set up variable information for the experiment, to define the ranges and names of the variables in the design space.
2 Choose an initial model.
3 Create a base design that contains the correct constraints.
4 Create child designs using varying numbers of points and/or methods of construction.
5 Choose the design to run based on the statistics and how many points you can afford to run.

Create Designs and Collect Data

You can use a space-filling design to maximize coverage of the factors' ranges as quickly as possible, to understand the operating envelope.

To create a design, you need to first specify the model inputs. Open the example file to see how to define your test plan.

1   In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.

2   Select **File > Open Project** and browse to the example file `CI_MultiInject_AirSurvey.mat`, found in `matlab\toolbox\mbc\mbctraining`.

3   The Model Browser displays the top project mode in the **All Models** tree, `CI_Multiinject_AirSurvey`.

4   To see how to define your test plan, click **Design Experiment**. In the new test plan dialog box, observe the inputs pane, where you can change the number of model inputs and specify the input symbols, signals and ranges. This example project already has inputs defined, so click **Cancel**.

5   Click the first test plan node in the **All Models** tree, `AirSurveyDoE`. The test plan view appears.
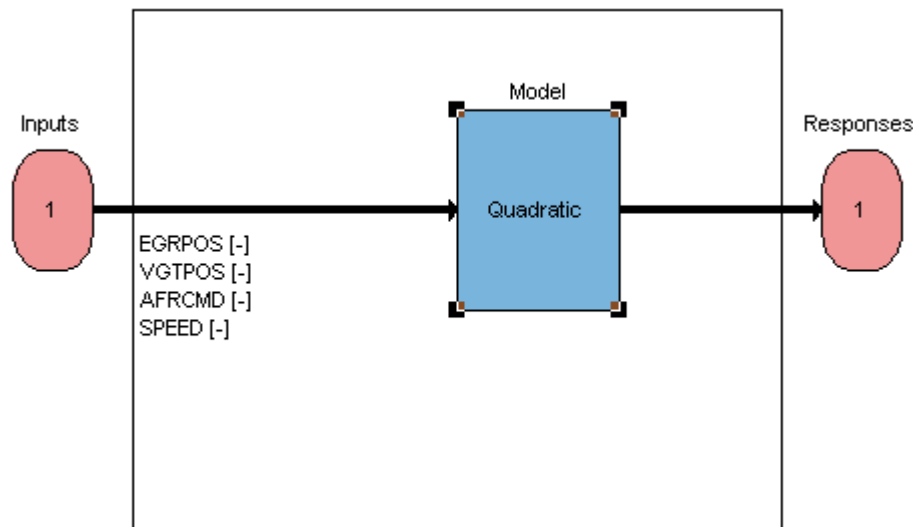
6   Observe the inputs listed on the test plan diagram. Double-click the **Inputs** block to view the ranges and names (symbols) for variables in the Input Factor Set Up dialog box.
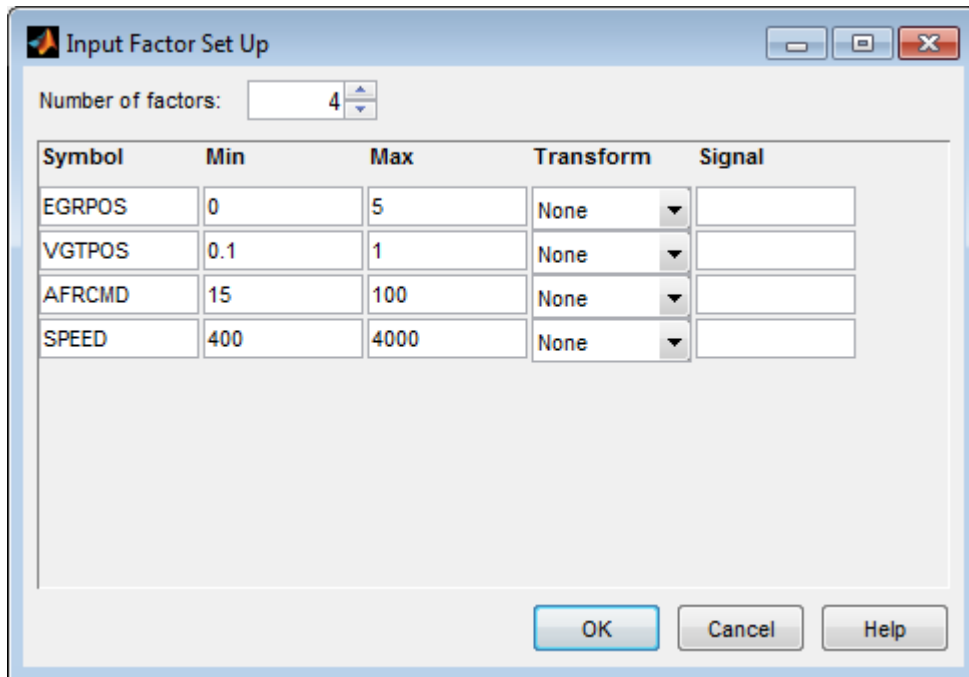
Close the dialog box.

7   After setting up inputs, you can create designs. In the **Common Tasks** pane, click **Design experiment**.

The Design Editor opens. Here, you can see how these designs are built:

- The final Air Survey design is a ~280 point Sobol Sequence design to span the engine operating space. The Sobol Sequence algorithm generated the space-filling design points. The final design is called `AirSurveyMergedDoE` because it contains two merged designs, one with a constraint and one unconstrained:

  - A 232-point design for overall engine operating range called `AirSurveySpaceFill`
  - A 50-point design low speed/load for an idle region called `AirSurveyIdle`

8   Select the `AirSurveySpaceFill` design in the Designs tree, and select **View** > **Current View** > **2D Design Projection**.
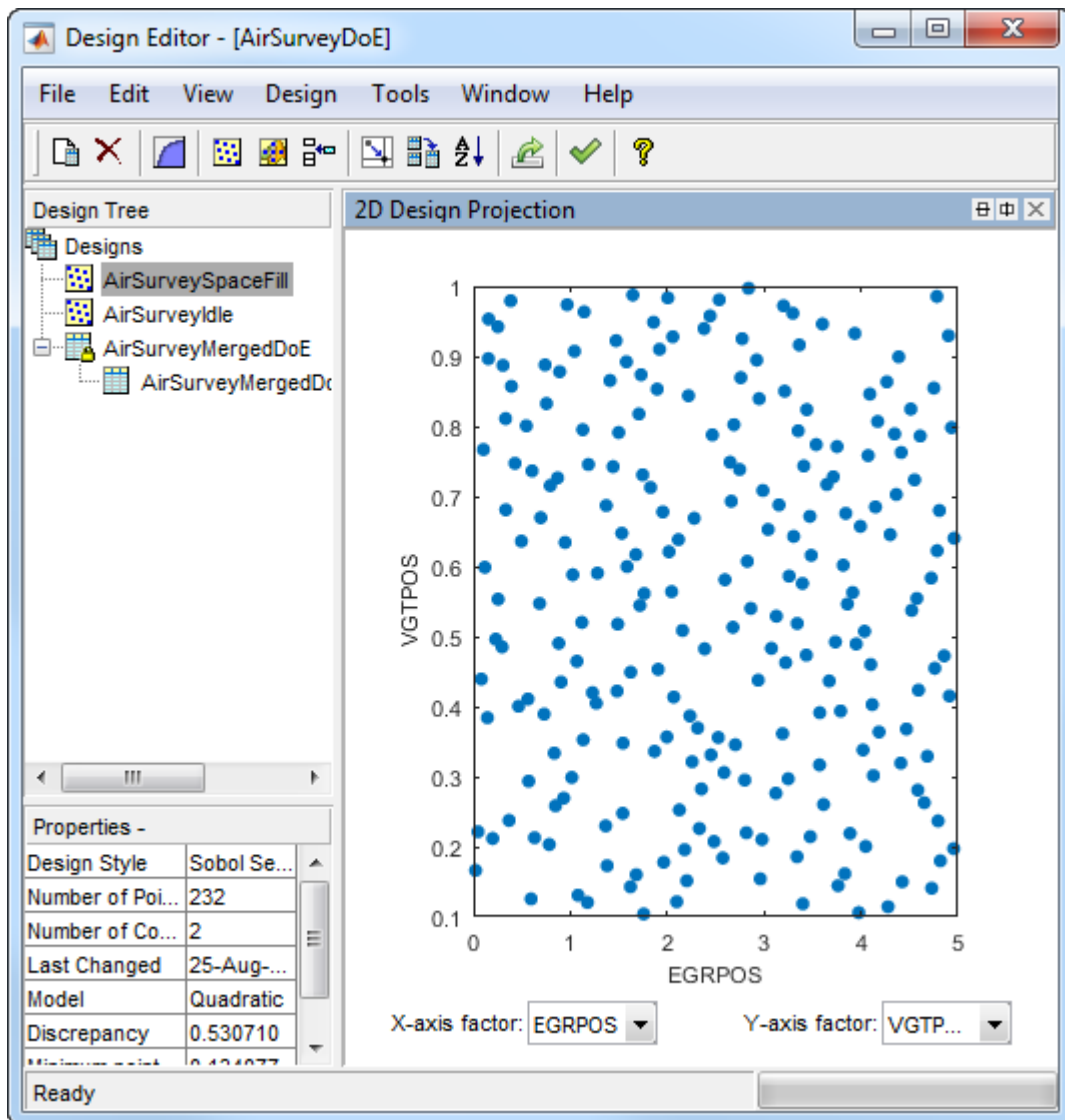
9 Select **Edit > Constraints** to see how the constraints are set up.
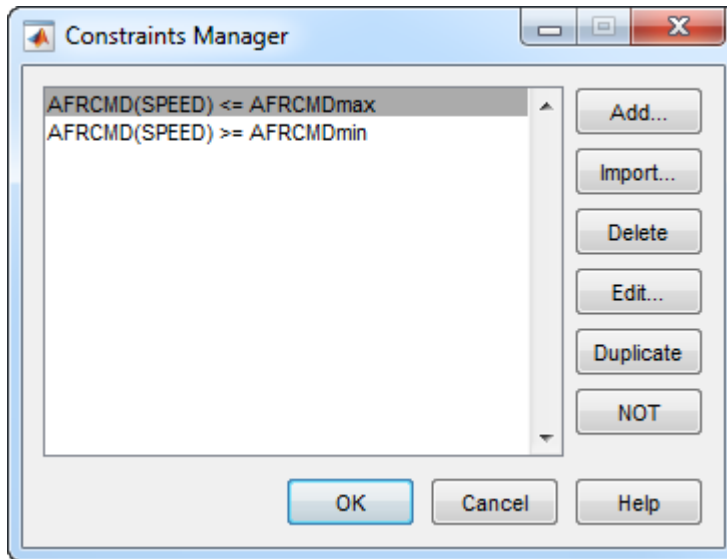
10 In the Constraints Manager dialog box, select a constraint and click **Edit**. Observe that you can define areas to exclude by dragging points or typing in the edit boxes.
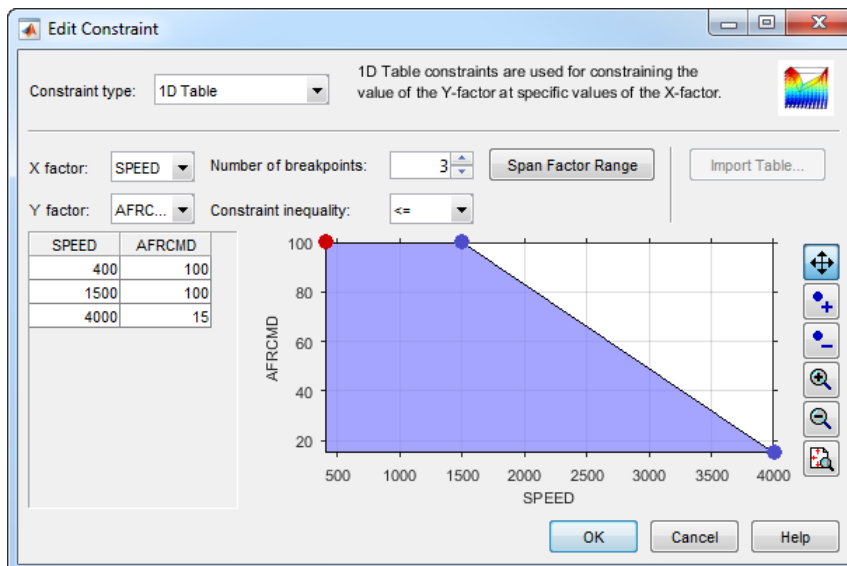
Click **Cancel** to close the Edit Constraint dialog box and leave the constraint unchanged. Close the Constraints Manager dialog box.

11 Observe the `Properties` of the selected `AirSurveySpaceFill` design under the tree, listing 2 constraints, 232 points, and a `Design Style` of `Sobol Sequence`.

12 You can see how to construct a design like this by selecting **Design > Space Filling > Design Browser**, or click the space-filling design toolbar button.

a • A dialog box asks what you want to do with the existing design points. Click **OK** to replace the points with a new design.

• In the Space-Filling Design Browser, click **Generate** several times to try different space-filling points. You can specify the **Number of points**, and view previews of design points. Observe the default design type is `Sobol Sequence`.
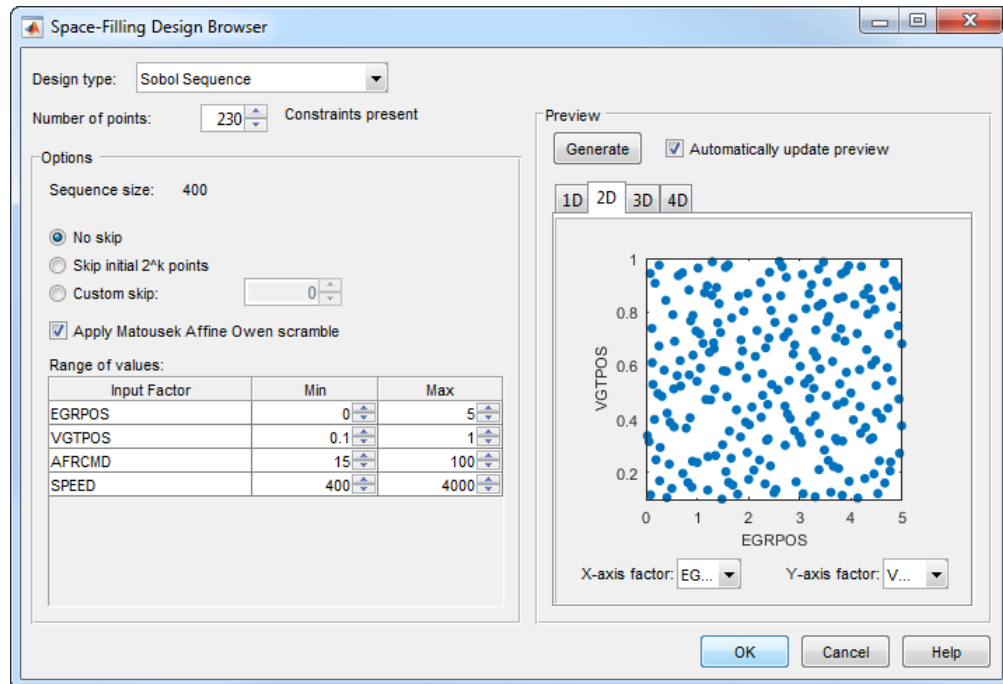
- Click **Cancel** to close the Space-Filling Design Browser and leave the design unchanged.

13 Click the `AirSurveyIdle` design to observe it is also a `Sobol Sequence` design, containing 50 points and no constraints.

14 Click the `AirSurveyMergedDoE` design to observe it is of type `Custom`, and contains 282 points and 2 constraints. This design is formed by merging the previous 2 designs. You can find **Merge Designs** in the **File** menu.

15 Click `AirSurveyMergedDoE_RoundedSorted`, the child design of `AirSurveyMergedDoE`. This design contains the same points but rounded and sorted, using **Edit > Sort Points** and **Edit > Round Factor**. This is the final design used to collect data.

16 Close the Design Editor.

The final air survey design was used to collect data from the GT-Power model with the Simulink and Simscape test harness. The example Model Browser project file `CI_MultiInject_AirSurvey.mat` contains this data, imported to the Model Browser after the air survey. You can also view the data in spreadsheet form in the file `CI_AirSurvey_Data.xlsx` in the `mbctraining` folder.

Fit a Boundary Model to Air Survey Data

You need to fit a boundary model to the data collected at the air survey design points. The test data from the air survey was used to create a boundary model of the engine operating range. The example Model Browser project file `CI_MultiInject_AirSurvey.mat` contains this boundary model.

1 In the Model Browser, click the second test plan node in the **All Models** tree, `AirSurveyBoundary`.

Compare with the `AirSurveyDoE` test plan view. For the boundary modeling test plan,

- Observe an imported data set listed under the **Common Tasks** pane. The second test plan has imported the air survey data in order to fit a boundary model to the data.

- Observe the `Inputs` are different in the test plan diagram. Instead of the `AFRCMD` input in the DoE test plan, there is a `SPEED` input for boundary modeling. `AFRCMD` was used to constrain the design points to collect the air survey data. To model the boundary before creating the final design, you now need the `SPEED` input instead.
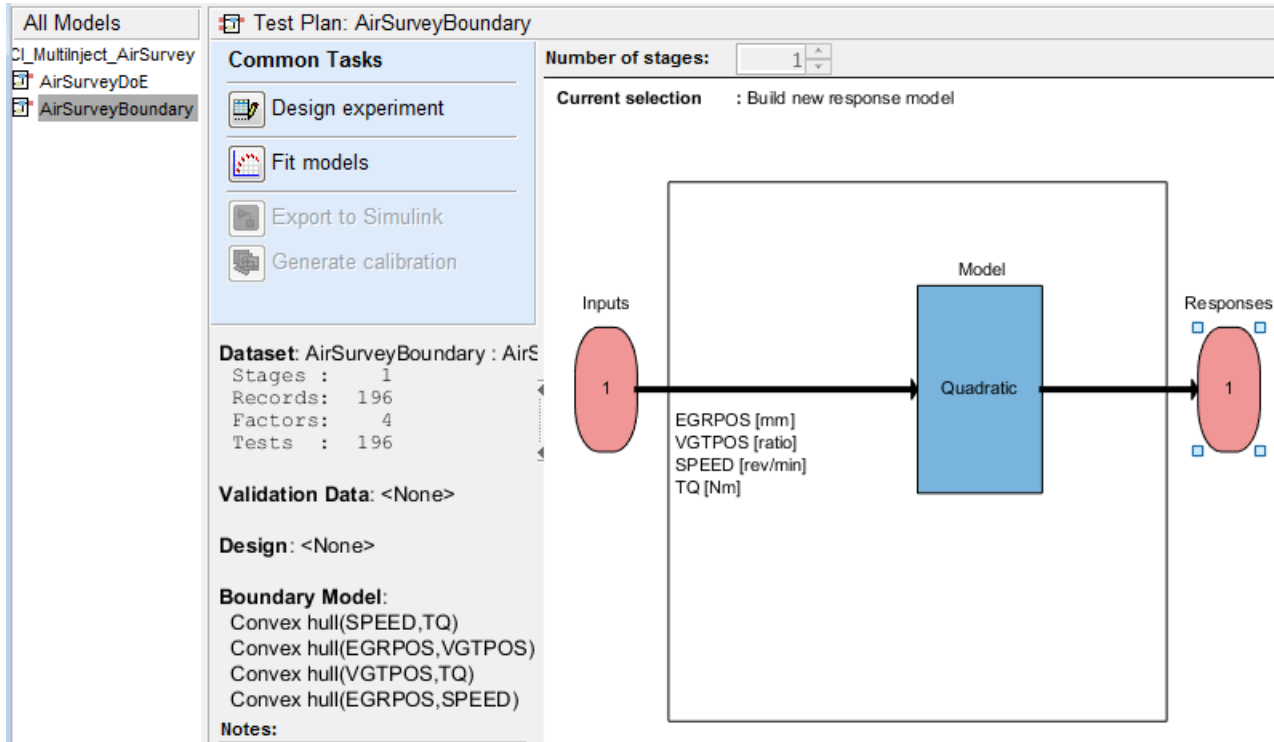
2. In the Model Browser, select **TestPlan > Edit Boundary** to open the Boundary Editor.

3. Examine the boundary model by selecting **View > Current View > Pairwise Projections**. The plots show the shape across the inputs.
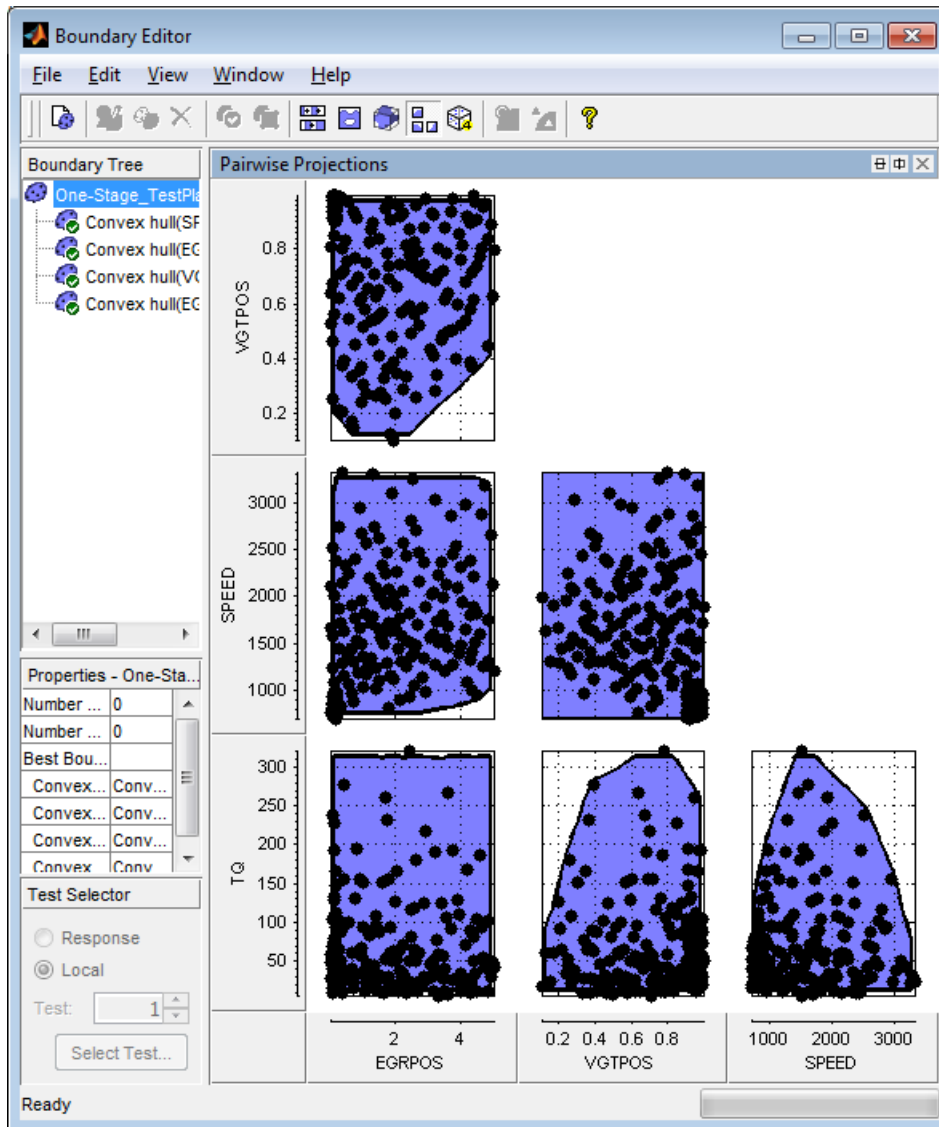
Use the Air Survey and Boundary Model to Create the Final Design

The initial air survey design and test data provide information about the engine operating envelope, where the feasible AFR range can produce positive brake torque. The resulting data lets you create a boundary model. You can use the boundary model to create the final design avoiding infeasible points, and in later steps to guide modeling and constrain optimizations.

Using the boundary model as a constraint, you can generate the final design to collect detailed data about fuel injection effects within those boundaries. You can then use this data to create response models for all the responses you need in order to create an optimal calibration for this engine.

The final Point-by-Point Multi-Injection design (around 7000 points) was generated using a MATLAB script together with the Air Survey boundary model.

1 Open the example files `CI_PointbyPointDoE.m` and `createCIPointbyPointDoEs.m` in the `mbctraining` folder to see the script commands that build the final design.

The script keeps only points that lie within the boundary model, and continues generating design points until it reaches the desired number of points. The Sobol space-filling design type keeps filling in spaces for good coverage of the design space.

After generating design points for each test, the script creates a Model Browser project with a point-by-point test plan, and attaches the point-by-point designs to the test plan.

2   Open the project `CI_MultiInject_PointbyPoint.mat` to view the project created by the script.

3   Select the second test plan node in the tree, and then click the Test Plan tab. In the test plan diagram, right-click the Operating Point Selector block, and select **Design Experiment** to view the designs created by the script.
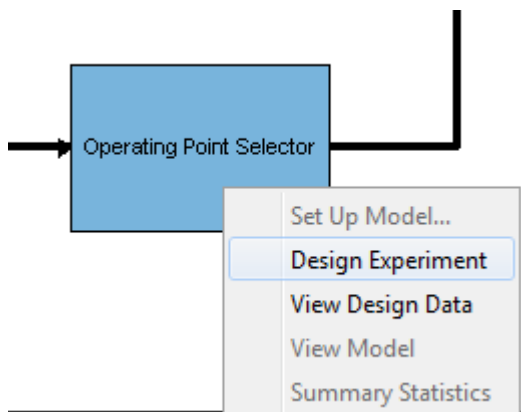
4   In the Design Editor, select **Mode Points** in the Design tree, and view the 2D Design Projection.

5   In the Design tree, select **Actual Design**. Observe that the boundary model constraint removed points in the corners, so that the actual design points collect data only in the feasible region determined by the initial air survey.

6   Return to the Model Browser test plan, right-click the Local Model block, and select **Design Experiment**.
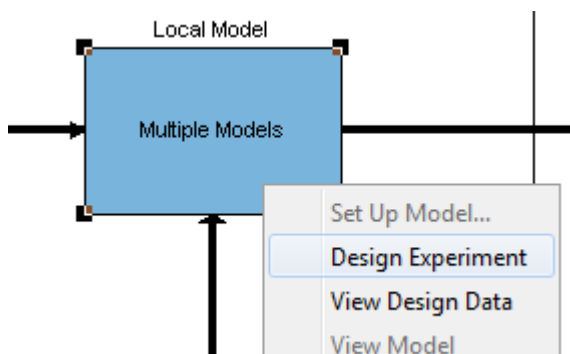
7   Click test numbers to view the local design points at each operating point. At each test, the values of `SPEED` and `TQ` are fixed and the space-filling design selects points across the space of the local inputs.

Multi-Injection Testing

The final design was used to collect data for the following inputs and responses.
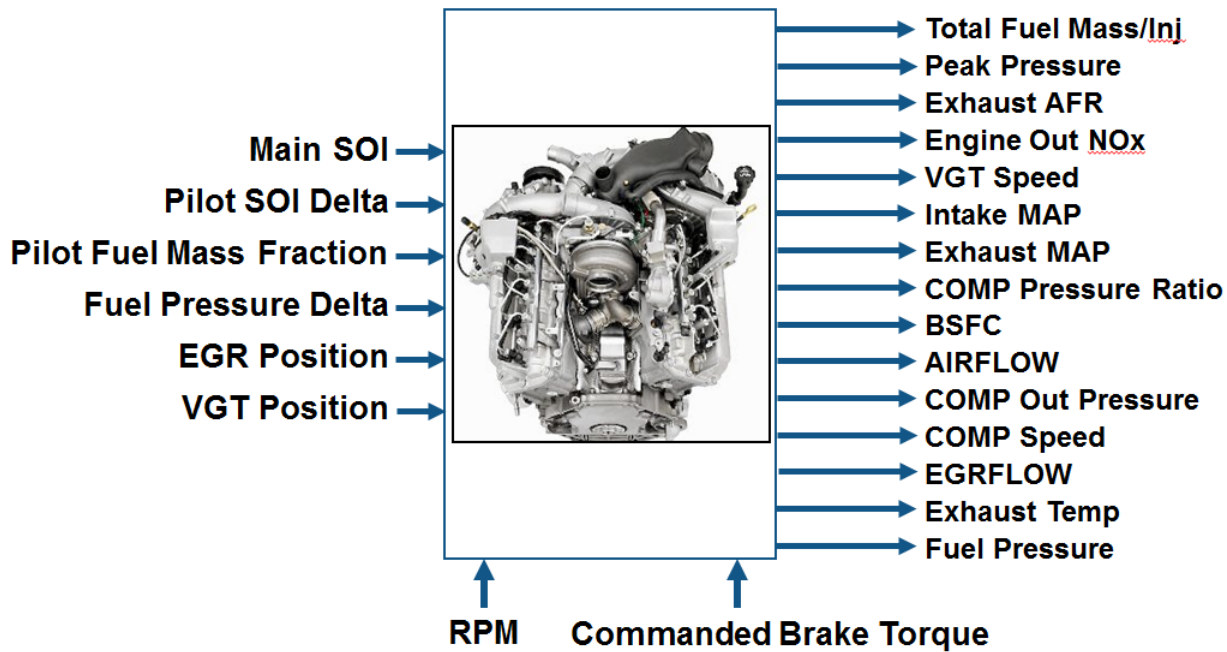
The toolbox provides the data files for you to explore this calibration example. You can view the data in spreadsheet form in the file `CI_MultiInject_PointByPoint_Data.xls`, and the data is imported to the Model Browser project file `CI_MultiInject_PointbyPoint.mat`.

For details on how the data was collected, see "Data Collection and Physical Modeling" on page 4-7.

For next steps, see "Statistical Modeling" on page 4-25.

Tip Learn how MathWorks Consulting helps customers develop engine calibrations that optimally balance engine performance, fuel economy, and emissions requirements: see Optimal Engine Calibration.

Statistical Modeling

Examine the Test Plans for Point-by-Point Models

After designing the experiments and collecting the data, you can fit statistical models to the data. You can use the toolbox to generate accurate, fast-running models from the measured engine data.

The following graphic shows the models to define in the toolbox to solve this calibration problem. The graphic shows how the model inputs and output relate to the optimal tables, optimization operating points, objectives and constraints you need to perform the optimization and create the calibration.
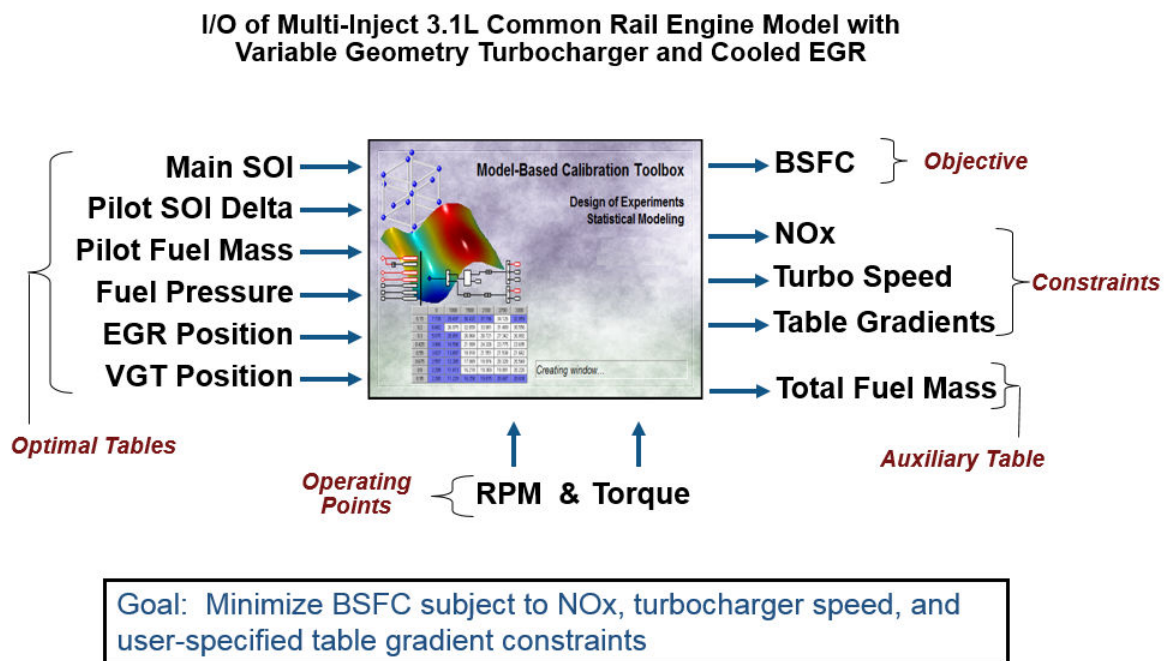
The toolbox provides the data for you to explore this calibration example. For details on how the data was collected, see "Data Collection and Physical Modeling" on page 4-7.

Examine the model setup.

1   In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.

2   In the Model Browser home page, in the **Case Studies** list, select **Multi-injection diesel tested with pilot injection on and off**. Alternatively, select **File > Open Project** and browse to the example file `CI_MultiInject_PointbyPoint.mat`, found in `matlab\toolbox\mbc\mbctraining`.

3   The Model Browser remembers views, so change to the test plan view if necessary, by clicking the `PilotInactivePointbyPoint` test plan node in the **All Models** tree.

The **Response Models** tab shows the cross-section view of the responses. Here you can assess high-level model trends.
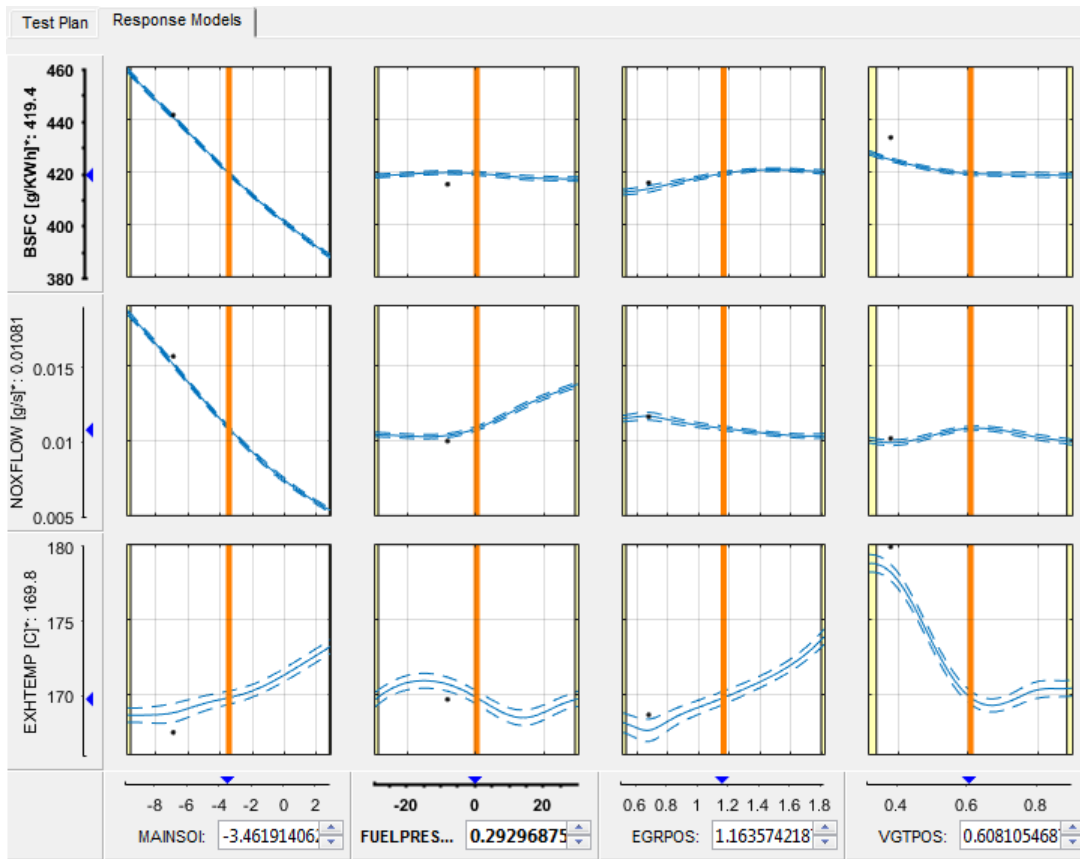
4 Select the **Test Plan** tab.
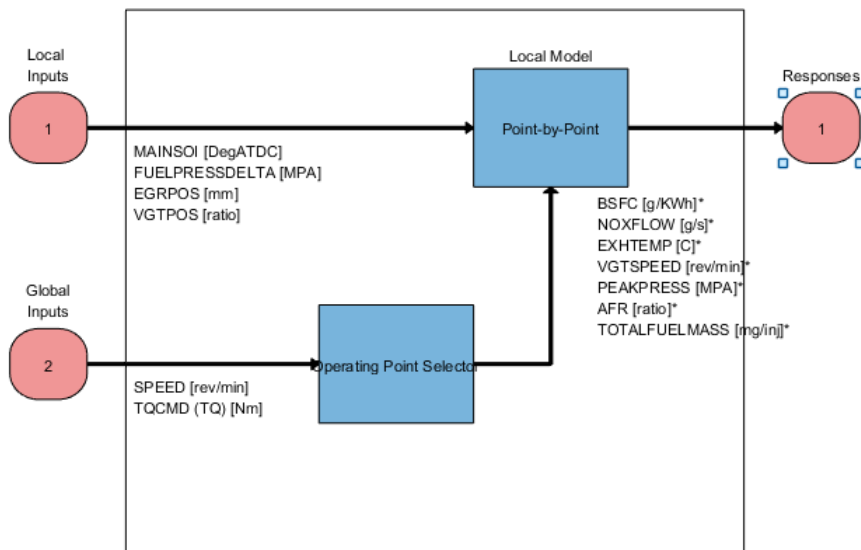
5 Observe the inputs and response model outputs listed on the test plan diagram. This is a Point-by-Point test plan. The Global Inputs SPEED and TQCMD select the operating points for each model.

6   Double-click the **Inputs** blocks to view the ranges and names (symbols) for variables on the Input Factor Set Up dialog box.

7   Double-click the **Local Model** block to view that the Point-by-Point Model Setup dialog. View the list of alternative models to fit at each operating point for a point-by-point test plan. When you use the **Fit models** button in the **Common Tasks** pane, and select a `Point-by-Point` template, the toolbox selects this list of models. This model setup fits four alternative model types at each operating point, and selects the best model based on the **Criteria** setting, in this case `PRESS RMSE`.
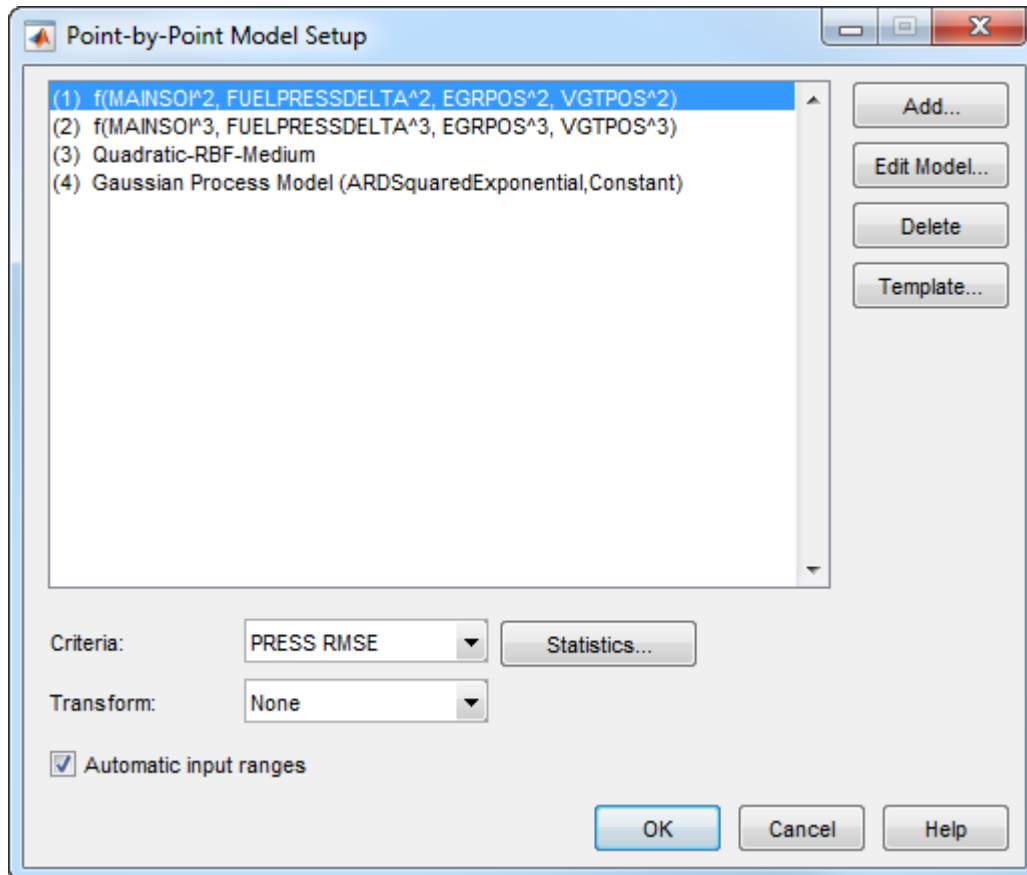
8   Click **Cancel** to close the Model Setup dialog box without altering your example models.

9   Similarly, examine the `PilotActivePointbyPoint` test plan. This test plan has the same response models and model type setup, but two more local inputs for the pilot injection timing and mass, `PILOTDELTASOI` and `PILOTFMF`. The data used to fit the models is also different, with the two additional factors. Observe the **Dataset** information under the **Common Task** pane in the test plan view.

For details on setting up point-by-point models, see "Fit a Point-by-Point Model".

Examine Response Models

1   Expand the `PilotInactivePointByPoint` test plan node in the **All Models** tree and select the nodes for each response name, e.g., `BSFC`.

2   Click through tests to see the model selected as best at each point.

The toolbox tries to select the best model for each operating point based on the selected statistical criteria. However, you should always verify the model choices. To search for the best fit you must examine each model, consider removing outliers, and compare with alternative fits. The example models show the results of this process.

3   Look at the RMSE Plots. These plots can help you identify problem tests to investigate. Double-click to view a test of interest (with larger RMSE) in the other plots.
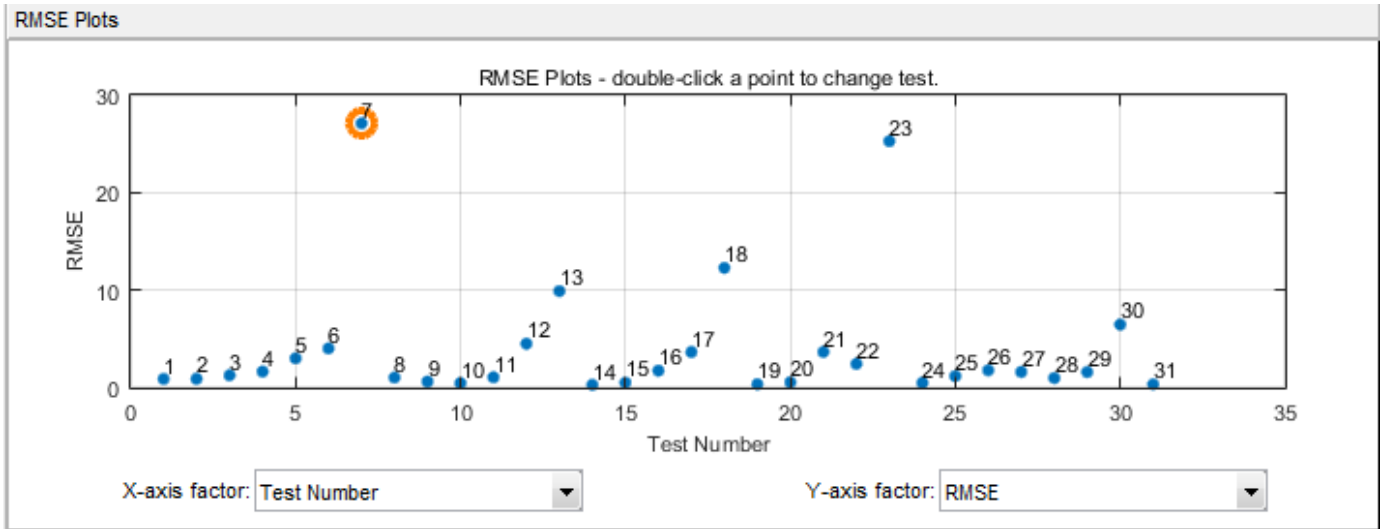
4   Choose a model to use at a particular operating point by selecting the **Best Model** check box in the list of **Alternative Local Models**.

Alternative Local Models						
Name	Observations	Parameters	Box-Cox	PRESS RMSE	RMSE	Best Model
Quadratic	99	9	1	2.87	2.705	☐
Cubic	99	17	1	2.346	2.128	☐
Quadratic-RBF-Medium	99	48	1	1.934	0.891	☑
GPM-ARDSquaredExpo...	99	14.031	1	2.662	2.237	☐

For details on analyzing point-by-point models, see "Analyze Point-by-Point Models and Choose the Best".

For next steps, see "Optimization" on page 4-29.

Tip Learn how MathWorks Consulting helps customers develop engine calibrations that optimally balance engine performance, fuel economy, and emissions requirements: see Optimal Engine Calibration.

Optimization

Optimization Overview

After creating the statistical models to fit the data, you can use them in optimizations. You can use the accurate statistical engine model to replace the high-fidelity simulation and run much faster, enabling optimization to generate calibrations in feasible times.

1   Run an optimization to choose whether to use Pilot Injection at each operating point.

2   Optimize fuel consumption over the drive cycle, and meet these constraints:

- Constrain total NOx
- Constrain turbocharger speed
- Constrain smoothness of tables

3   Fill lookup tables for all control inputs.

Set Up Models and Tables for Optimization

To perform an optimization, you need to import the statistical models created earlier in the Model Browser.

1   In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Optimization**.

2   To see how to import models, on the home page, click **Import From Project**

The CAGE Import Tool opens. Here, you can select models to import from a model browser project file or direct from the Model Browser if it is open. However, the toolbox provides an example file with the models already imported, so click **Close** to close the CAGE Import Tool.

3   On the home page, in the Case Studies list, select the multi-injection example, or select **File > Open Project** and browse to the example file CI_MultiInject.cag, found in `matlab \toolbox\mbc\mbctraining`.

4   Click **Models** in the left **Data Objects** pane to view the models.

5   To see how to set up tables to fill with the results of your optimizations, select **Tools > Create Lookup Tables from Model**. The Create Lookup Tables from Model wizard opens.
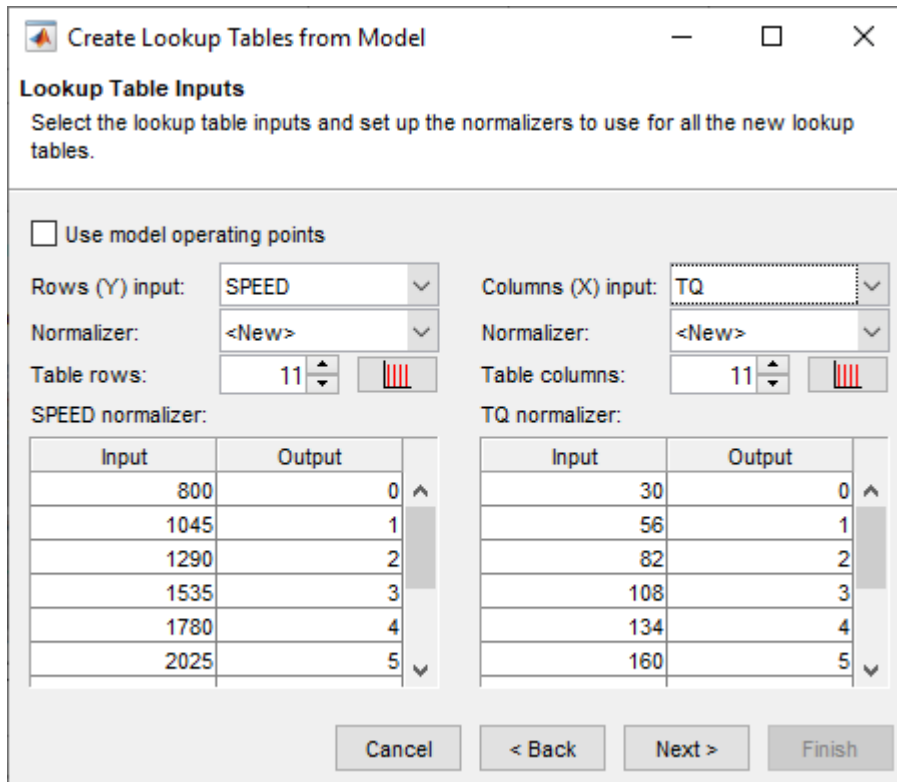
6   Select the model BSFC and the **Create operating point data set** check box, and click **Next**.

7   If you left the defaults, you would use the model operating points for the table breakpoints. However, you need to create different tables to the operating points.

   a   Clear the **Use model operating points** check box.

   b   Select SPEED and TQ for the axis inputs.

   c   Enter 11 for the table rows and columns.

Click **Next**.

8 View the last screen to see how many tables CAGE will create for you if you finish the wizard.

Click **Cancel** to avoid creating new unnecessary tables in your example file. The example file already contains all the tables needed for your calibration results.
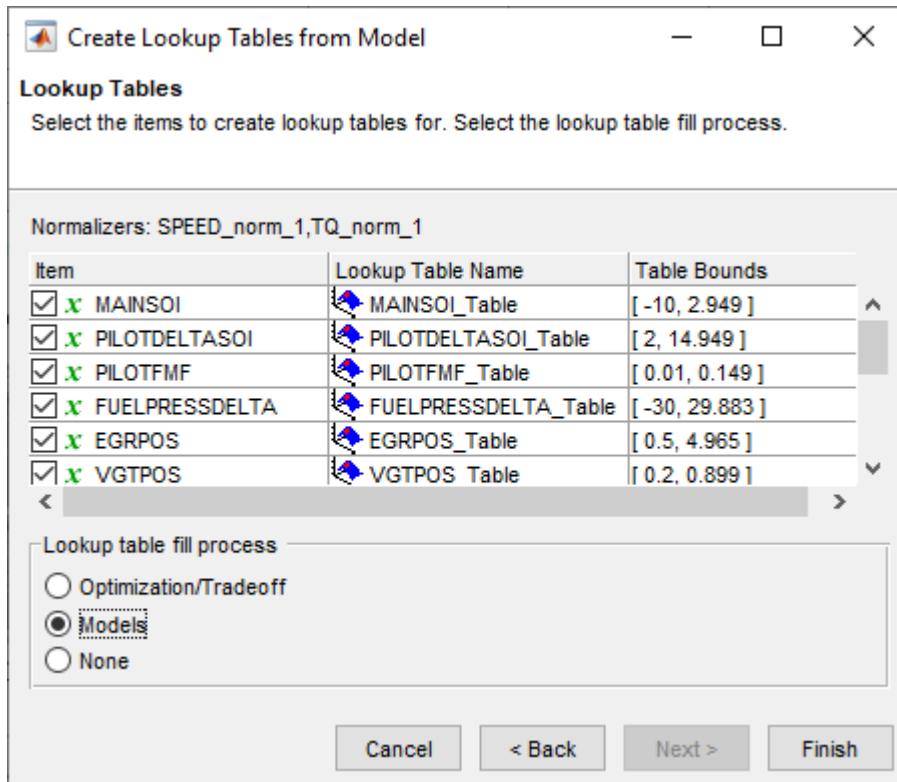
9   Select the **Lookup Tables** view to see all the tables. Observe there are tables named `_wPilot` and `_noPilot` that will be filled by optimization results. The goal is to fill separate tables for each mode, pilot injection active and no pilot injection.

Examine the Point Optimization Setup

The example file `CI_MultiInject.cag` shows the results of the multistage process to finish this calibration.

1   Click **Optimization** in the left **Processes** pane to view the optimizations.
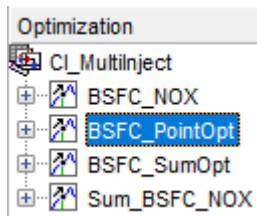
To complete the calibration, you need to start with a point optimization to determine whether to use Pilot Injection at each operating point. You use the results of this point optimization to set up the sum optimization to optimize fuel consumption over the drive cycle, and meet these constraints:

•   Constrain total NOx

•   Constrain turbocharger speed

- Constrain smoothness of tables

2 Select the `BSFC_PointOpt` optimization to view the setup.

3 View the two objectives in the Objectives pane. These are set up to minimize BSFC and PEAKPRESS. Double-click an objective to view its settings.

Objectives		
Name	Description	Type
BSFC	BSFC(MAINSOI, PILOTDELTASOI, PILOTFMF, FUELPRESSDELTA, EGRPOS, VGTPOS, SPEED, TQCMD, PilotMode)	Minimize
PEAKPRESS	PEAKPRESS(MAINSOI, PILOTDELTASOI, PILOTFMF, FUELPRESSDELTA, EGRPOS, VGTPOS, SPEED, TQCMD, PilotMode)	Minimize

4 Observe that the Constraints pane shows a boundary model constraint.

5 Learn the easiest way to set up an optimization like this by selecting **Tools > Create Optimization from Model**.
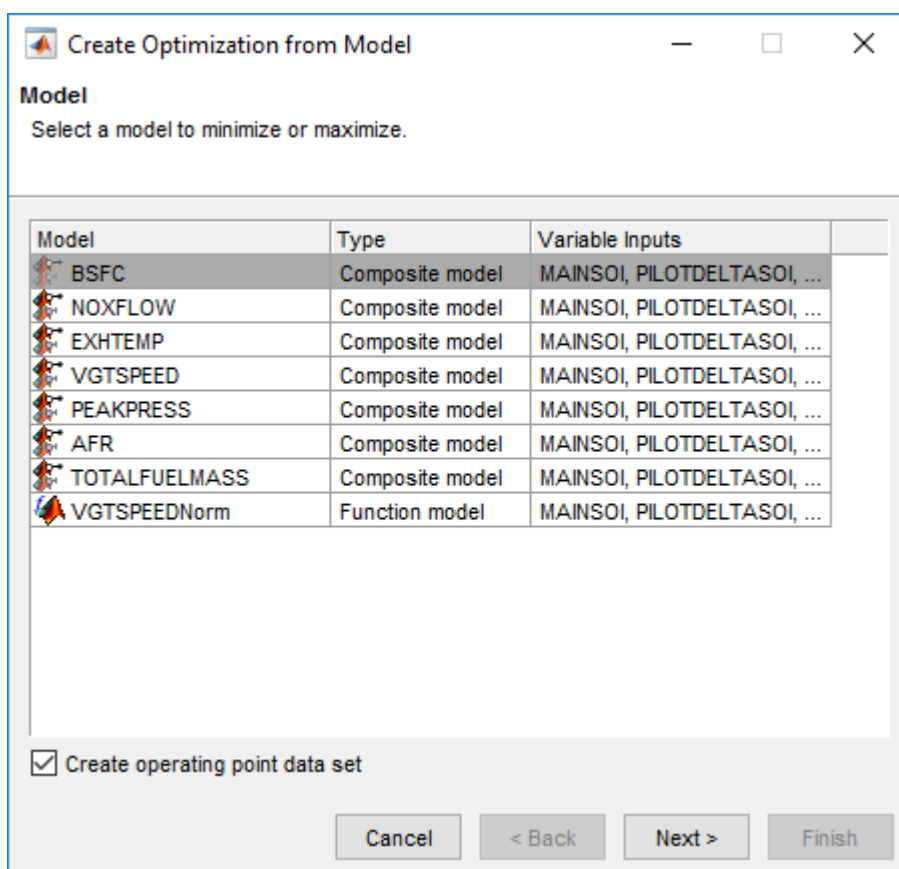
6 Select the model BSFC and the **Create operating point data set** check box, and click **Next**.

7 Observe that the default settings will create a point optimization to minimize BSFC at the model operating points, using six free variables and not SPEED, TQ, or Mode, and constrained by a model boundary constraint.

8   You want the optimization to identify which pilot mode (active pilot injection or no pilot injection) is best to use at each operating point. To do this, you must select `Modal optimization` from the **Algorithm** list.

9   You want to include `Mode` as a free variable, so the optimization can identify which mode is best to use at each operating point. From the **Free variables** list, select the check box to include `Mode`.
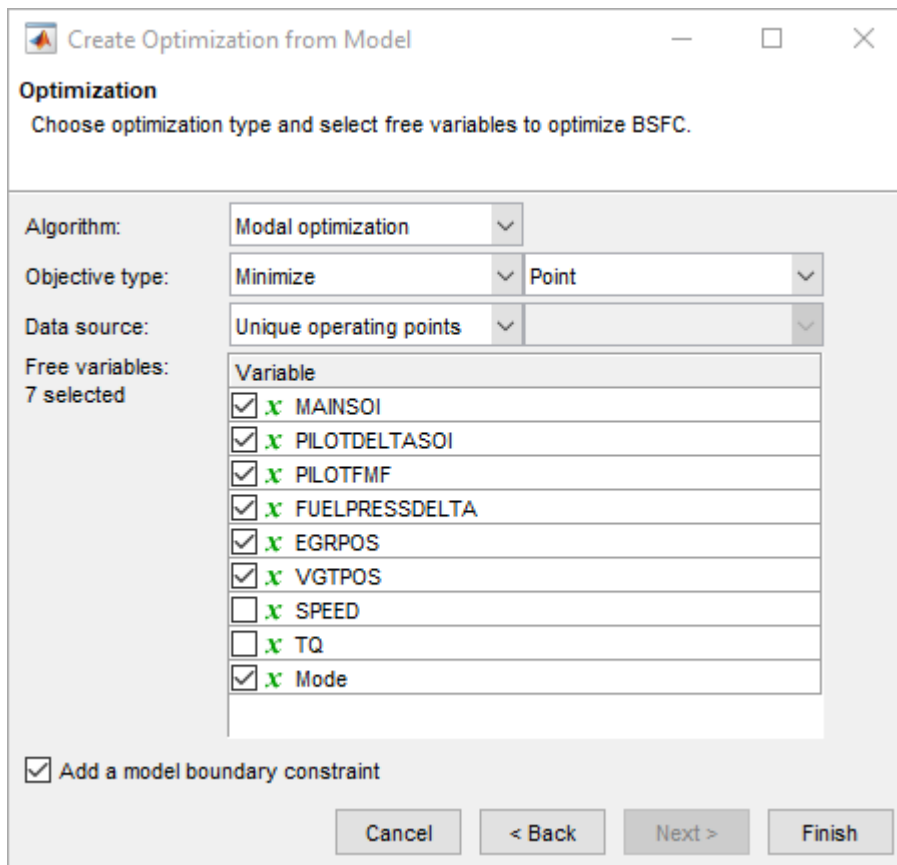
10 Click **Finish** to create the new optimization.

11 Compare your new optimization with the example `BSFC_PointOpt`. Notice the example has a second objective, to minimize PEAKPRESS. To see how to set this up, right-click the Objectives pane and select **Add Objective**.

Examine the Point Optimization Results

The example file `CI_MultiInject.cag` shows the results of the multistage process to finish this calibration.

1 Expand the first optimization node, `BSFC_PointOpt`, and select the output node underneath, `BSFC_PointOpt_Output`.

2 Select **View > Selected Solution** (or the toolbar button) to view which mode the optimization selected as best at each operating point. The left-hand pane provides all optimization results. The right-hand pane provides the results for the current run.

3   Review the Results Contour plot to see which mode has been selected across all operating points. Use this view to verify the distribution of mode selection.

4   Click to select a point in the table or Results Contour, and you can use the **Selected solution** controls at the top to alter which mode is selected at that point. You might want to change selected mode if another mode is also feasible at that point. For example, you can change the mode to make the table more smooth.

5   Use the other objectives to explore the results. For example, you might want to manually change the selected mode based on an extra objective value. It can be useful to view plots of the other objective values at your selected solutions. To display another plot or change the view, right-click the plots to view the options.

For example, right-click the Objective Contours plot. Select **Current View > Objective Graphs**.
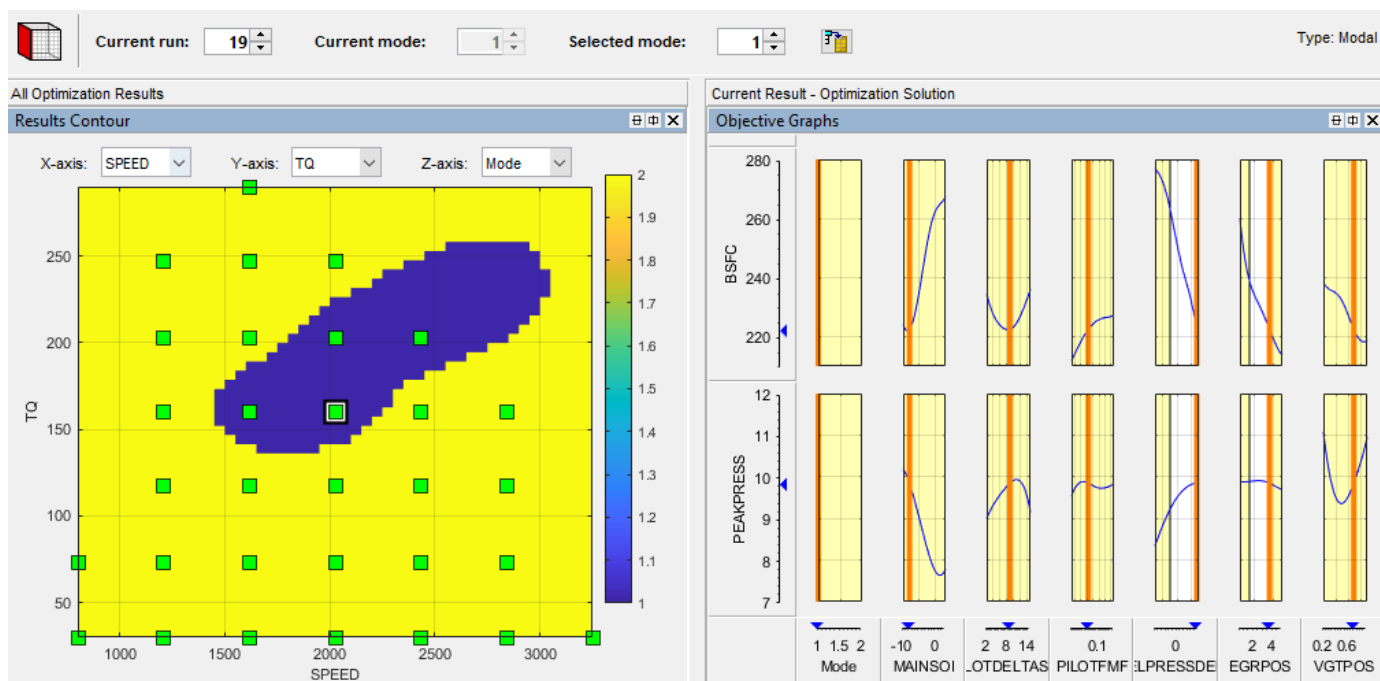
6   To see both solutions for a particular operating point, use the Pareto Slice view. You can inspect the objective value (and any extra objective values) for each solution. If needed, you can manually change the selected mode to meet other criteria, such as the mode in adjacent operating points, or the value of an extra objective.

For details on tools for choosing a mode at each operating point, see "Analyzing Modal Optimization Results".

Create Sum Optimization from Point Optimization

The point optimization results determine whether to use pilot injection at each operating point. When you are satisfied with all selected solutions for your modal optimization, you can make a sum optimization over all operating points. The pilot injection mode must be fixed in the sum optimization to avoid optimizing too many combinations of operating modes.

The results of the point optimization were used to set up the sum optimization to optimize fuel consumption over the drive cycle. To see how to do this:

1   From the point optimization output node, `BSFC_PointOpt_Output`, select **Solution** > **Create Sum Optimization**.

    The toolbox automatically creates a sum optimization for you with your selected best mode for each operating point. The create sum optimization function converts the modal optimization to a standard single objective optimization (`fmincon` algorithm) and changes the `Mode Variable` to a fixed variable.

2   Compare your new optimization with the example sum optimization, `BSFC_SumOpt`. The example shows you need to set up more constraints to complete the calibration.

Constraints			
Name	Description	Type	Application Point Set
BSFC_Boundary	Boundary constraint of BSFC(MAINSOI, PILOTDELTASOI, PI...	Model	
NOxSum	Weighted sum of NOXFLOW(MAINSOI, PILOTDELTASOI, PI...	Sum Constraint	
VGTSpeedMax	VGTSPEEDNorm(MAINSOI, PILOTDELTASOI, PILOTFMF, FU...	Model	
MainSOI_wPilot	Gradient constraint of MAINSOI over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...
MainSOI_noPilot	Gradient constraint of MAINSOI over (SPEED,TQ)	Table Gradient	PilotInactive(SPEED,TQ;...
EGRPOS_wPilot	Gradient constraint of EGRPOS over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...
EGRPOS_noPilot	Gradient constraint of EGRPOS over (SPEED,TQ)	Table Gradient	PilotInactive(SPEED,TQ;...
FuelPressDelta_w...	Gradient constraint of FUELPRESSDELTA over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...
FuelPressDelta_n...	Gradient constraint of FUELPRESSDELTA over (SPEED,TQ)	Table Gradient	PilotInactive(SPEED,TQ;...
VGTPOS_wPilot	Gradient constraint of VGTPOS over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...
VGTPOS_noPilot	Gradient constraint of VGTPOS over (SPEED,TQ)	Table Gradient	PilotInactive(SPEED,TQ;...
PilotDeltaSOI_wPilot	Gradient constraint of PILOTDELTASOI over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...
PilotFMF_wPilot	Gradient constraint of PILOTFMF over (SPEED,TQ)	Table Gradient	PilotActive(SPEED,TQ;M...

The additional constraints make the optimization meet these requirements:

- Constrain total NOx
- Constrain maximum turbocharger speed
- Constrain smoothness of tables with gradient constraints

3   Import all required constraints to your new optimization by selecting **Optimization > Constraints > Import Constraints**. Select the example sum optimization and import all but the first boundary model constraint.

4   Double-click the additional constraints to open the Edit Constraint dialog box and view the setup.

This sum constraint controls the total NOx.
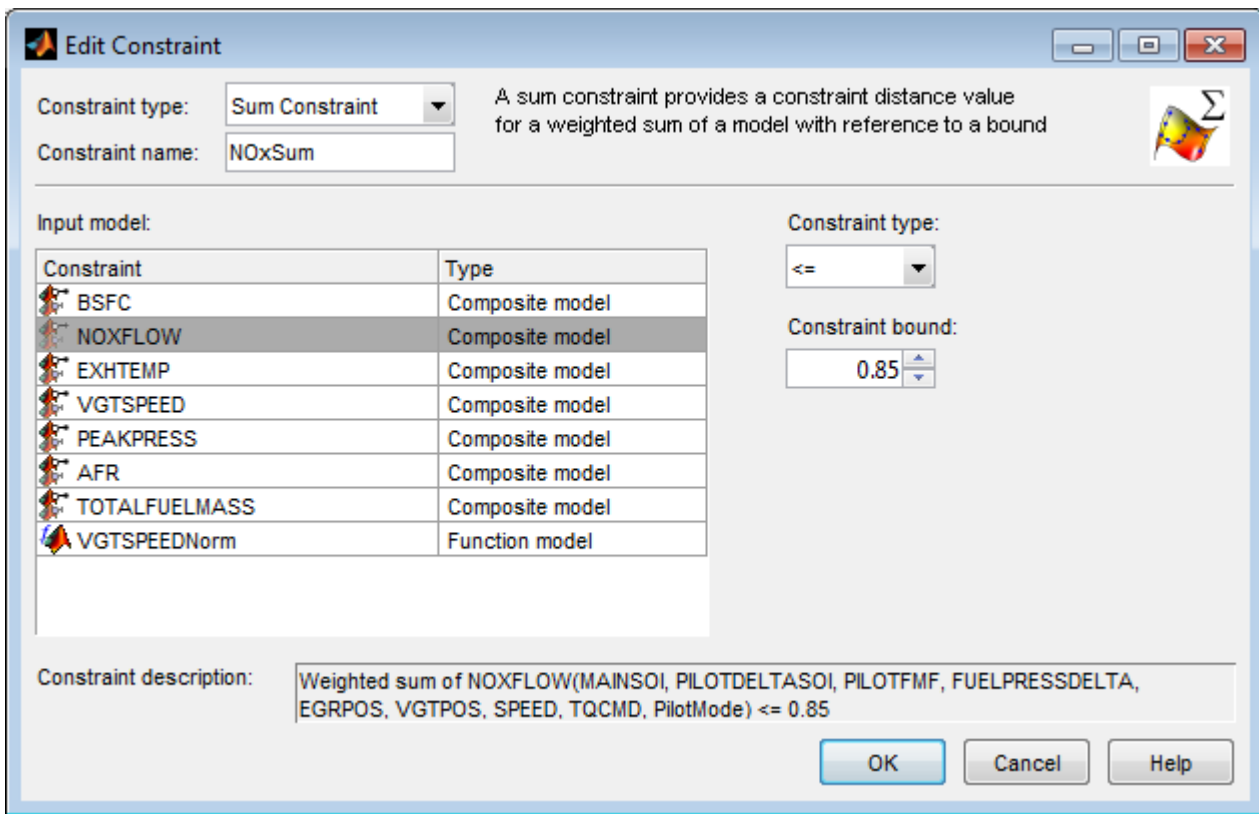
This constraint controls maximum turbocharger speed.
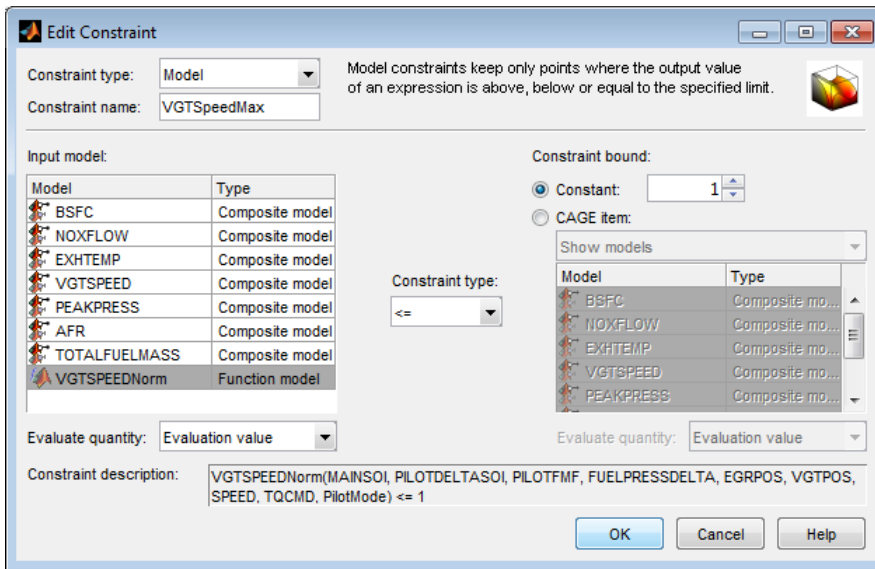
Observe that this constraint does not use the VGTSPEED model directly, but instead uses a function model VGTSPEEDNorm. You can examine this function model in the Models view. The function model scales the constraint to help the optimization routines, which have problems if constraints have very different sizes. VGTSPEED is of order of 165000, while the other constraints are of the order of 1, so the function model VGTSPEEDNorm normalizes VGTSPEED by dividing it by 165000.

The following constraint controls the gradient across the MAINSOI table.
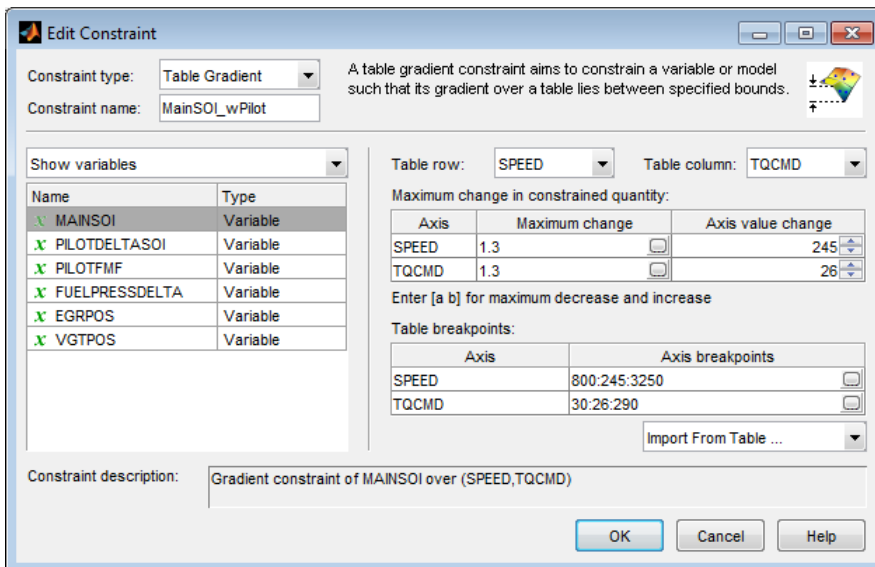
5   In the Optimization view, in the Constraints pane, observe that:

• The gradient constraints are in pairs, one with pilot and one with no pilot. Separate table gradient constraints are required for different modes because the goal is to fill separate tables for each mode.

• All the gradient constraints have an entry in the **Application Point Set** column.

6 Right-click the table gradient constraint `MainSOI_wPilot` and click **Select Application Point Set** to see how these are set up.
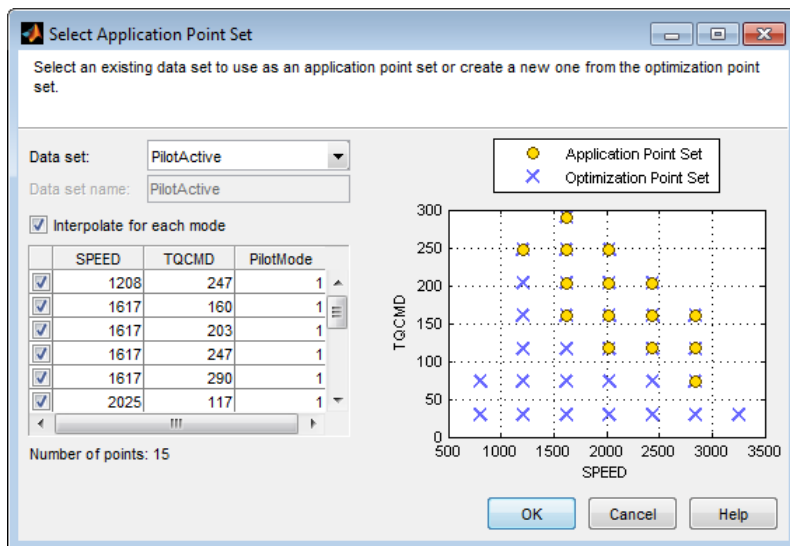
7 Observe the **Interpolate for each mode** option is selected. This application point set restricts the table gradient constraint to `Mode` 1 (active) points only. You can create an application point set like this by selecting `New subset` and then choosing a subset of the optimization points by clicking in the plot or table. The application points here correspond to the operating points where the point optimization determined that the pilot mode should be active (`Mode = 1`). For comparison, here is the results contour plot for the point optimization results.
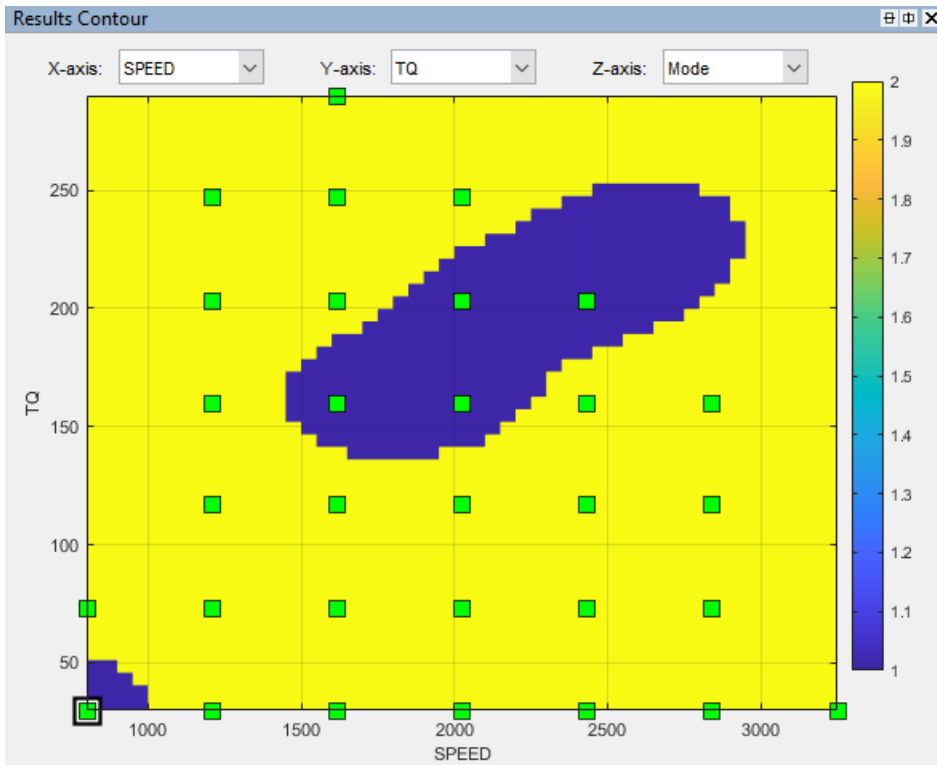
8   Compare the results contour plot with the application points for the next table gradient constraint, `MainSOI_noPilot`. These application points restrict the table gradient constraint to `Mode` 2 points only, where the pilot is inactive.
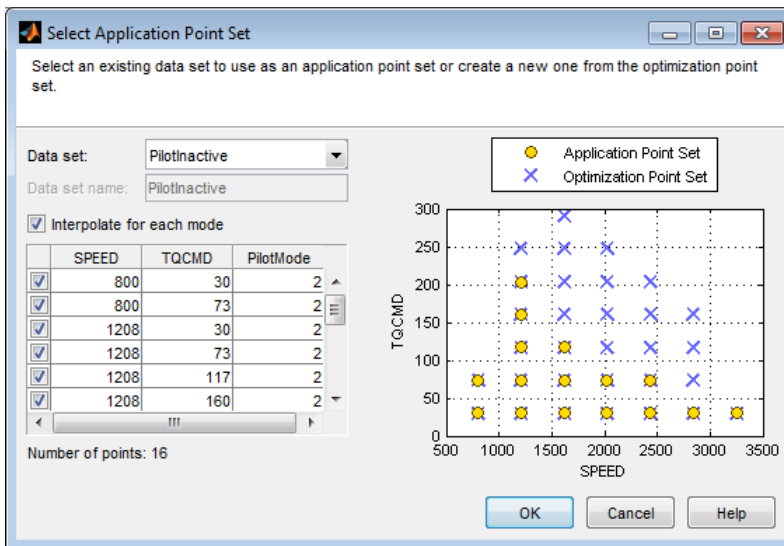
Your sum optimization now contains all the required constraints and is ready to run. Next, view the results in the example sum optimization.

Fill Lookup Tables from Optimization Results

CAGE remembers lookup table filling settings. To view how the example tables are filled:

1   Expand the example sum optimization node `BSFC_SumOpt` and select the `BSFC_SumOpt_Output` node.

2   Select **Solution > Fill Lookup Tables** (or use the toolbar button) to open the Lookup Table Filling from Optimization Results Wizard.

3   On the first screen, observe all the tables in the **CAGE tables to be filled list**. Click **Next**.

4   On the second screen, observe all the tables are matched up with optimization results to fill them with. Click **Next**.
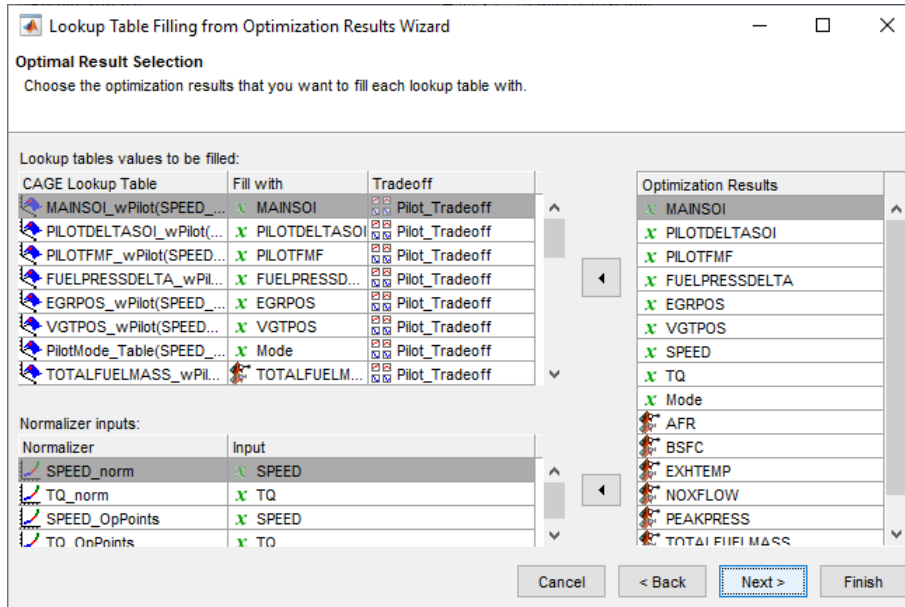
5   On the third screen, observe how some tables have a **Filter Rule** set up so that they are filled only with results where `Mode` is 1 or 2. You can create a filter rule like this by entering `Mode==1` in the **Filter Rule** column.
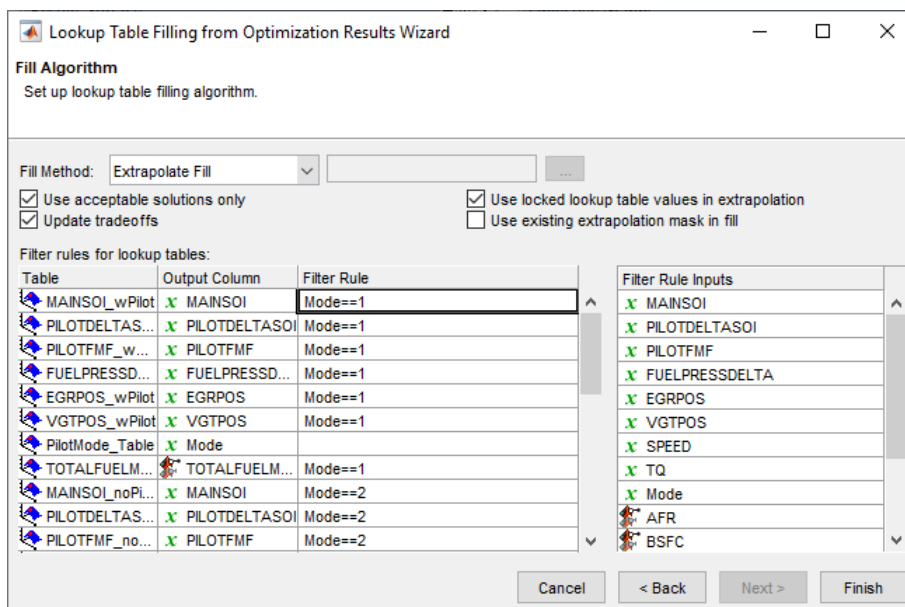
You can either click **Finish** to fill all the tables, or **Cancel** to leave the tables untouched. The example tables are already filled with these settings.

These plots show results similar to the calibration results.
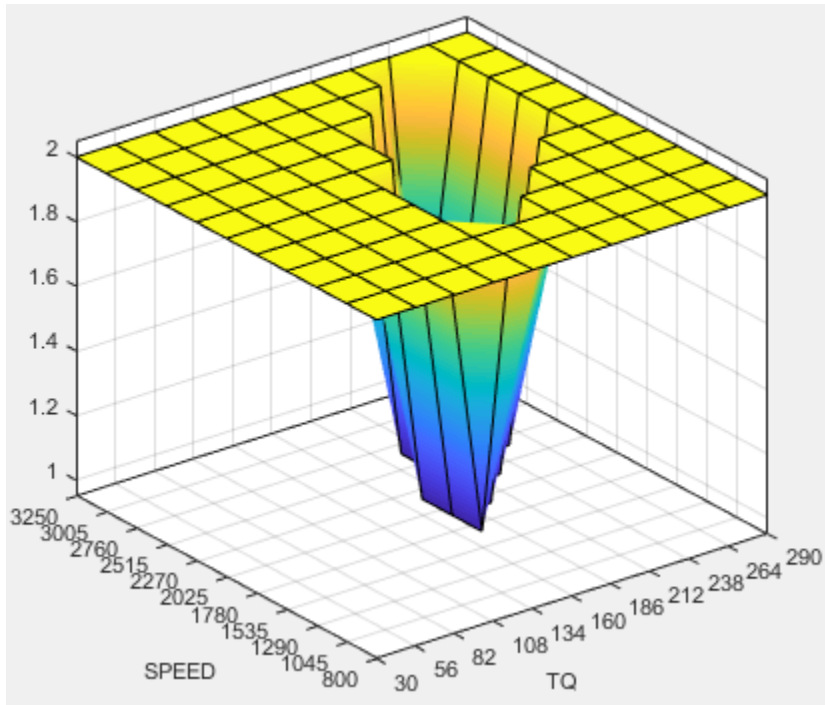
Pilot Mode Table

This graphic shows the plot of the table to select the active or inactive pilot mode depending on the speed and commanded torque

You need to fill calibration tables for each control variable described in "Multi-Injection Diesel Problem Definition" on page 4-2, in both pilot modes, active and inactive.

Following are all the pilot active tables.
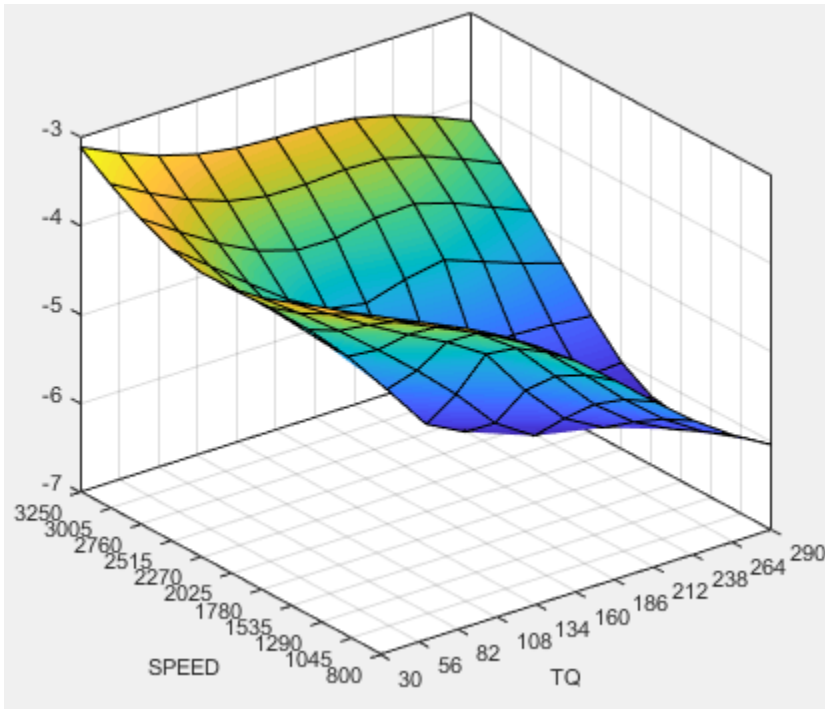
Main Injection Timing (SOI) Table
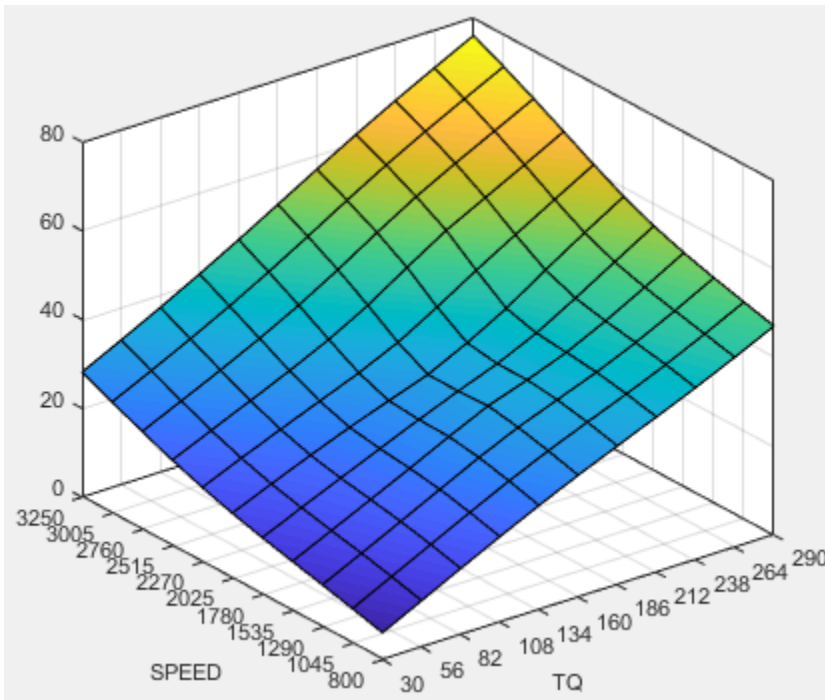
Total Injected Fuel Mass Table
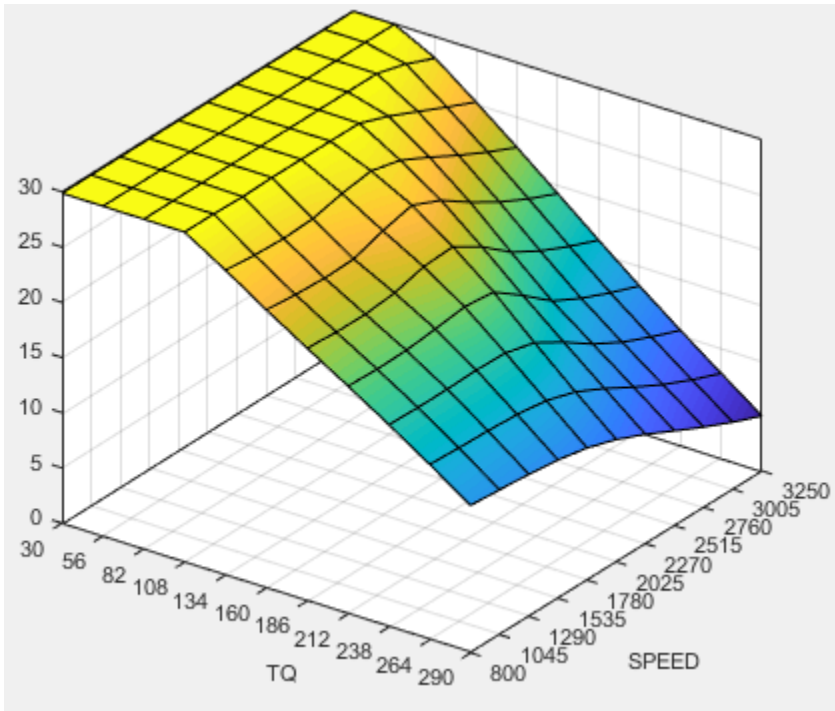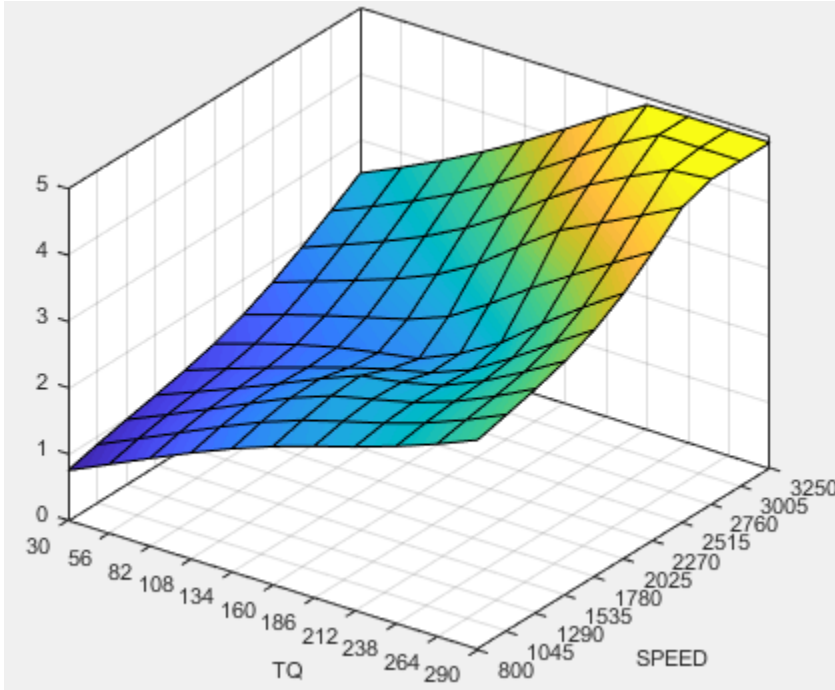
Fuel Pressure Delta Table

Exhaust Gas Recirculation (EGR) Valve Position Table
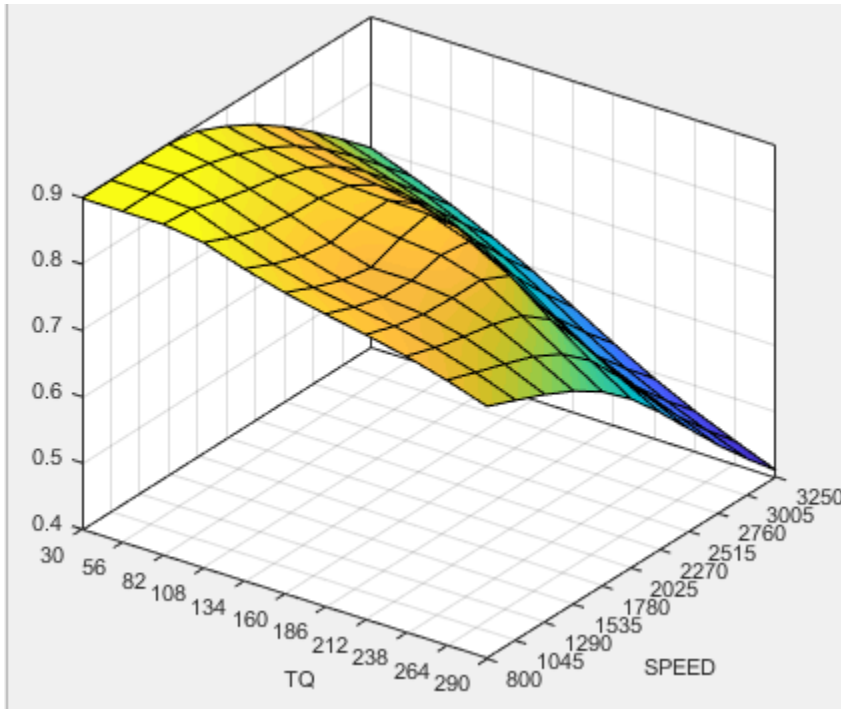
Variable-Geometry Turbo (VGT) Position Table
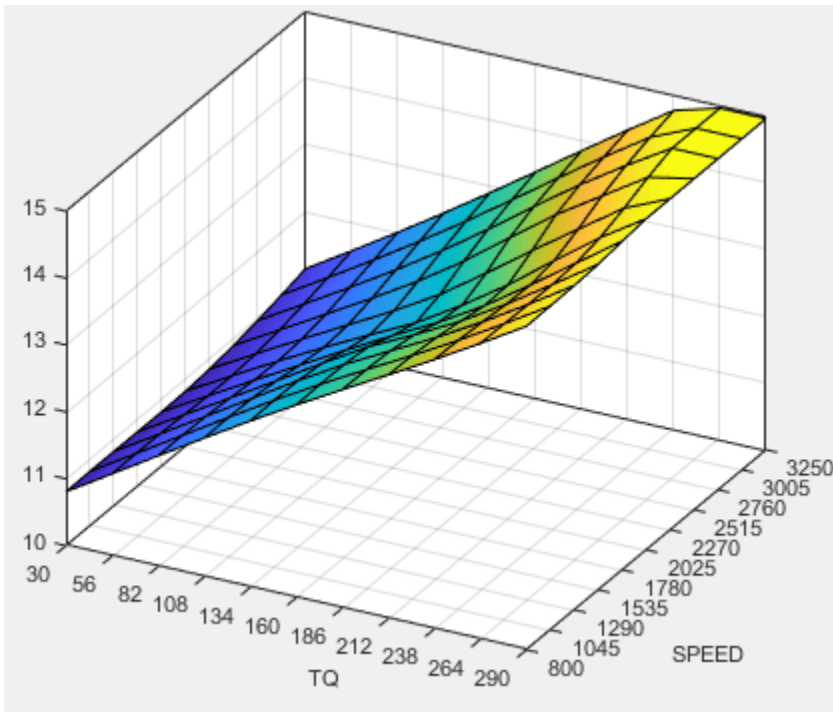
Pilot Injection Timing (Pilot SOI Delta) Table
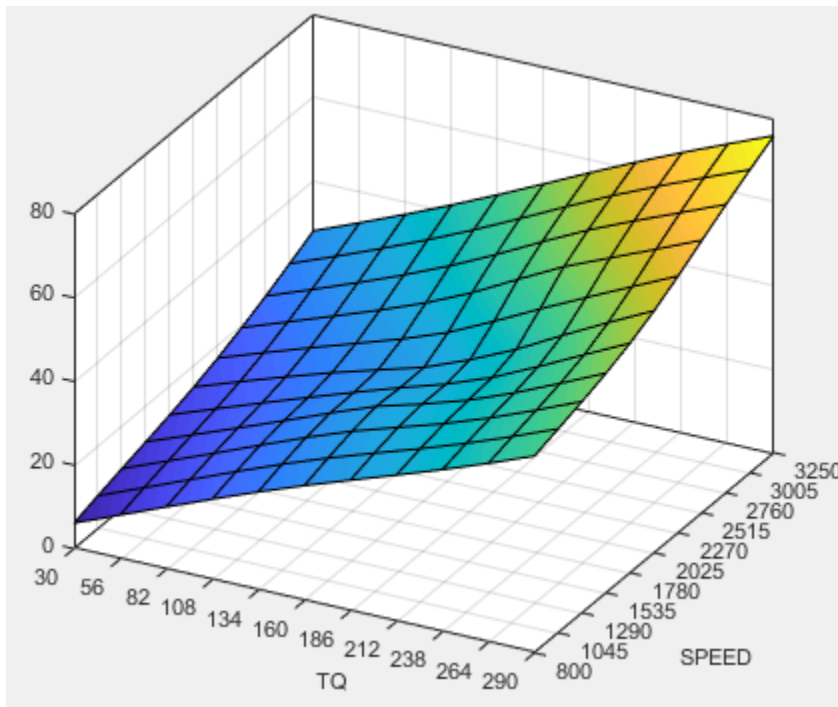
Pilot Fuel Mass Fraction Table

Examine the Multiobjective Optimization

The example file `CI_MultiInject.cag` also shows an example multiobjective optimization. This can useful for calibrations where you want to minimize more than one objective at a time, in this case, `BSFC` and `NOX`. The multiobjective optimization uses the `gamultiobj` algorithm.

1. Select the `BSFC_NOX` node to see how the optimization is set up. Observe the 2 objectives and the optimization information: Multiobjective optimization using a genetic algorithm.

2. Expand the `BSFC_NOX` node and select the `BSFC_NOX_Output` to view the results. To select the view that you want to examine, right-click in the pane and select **Current View**. Select the view.

3. Examine the Pareto graphs. It can be useful to display the Solution Information view at the same time to view information about a selected solution. You might want to select a dominated solution (orange triangle) over a pareto solution (green square) to trade off desired properties.
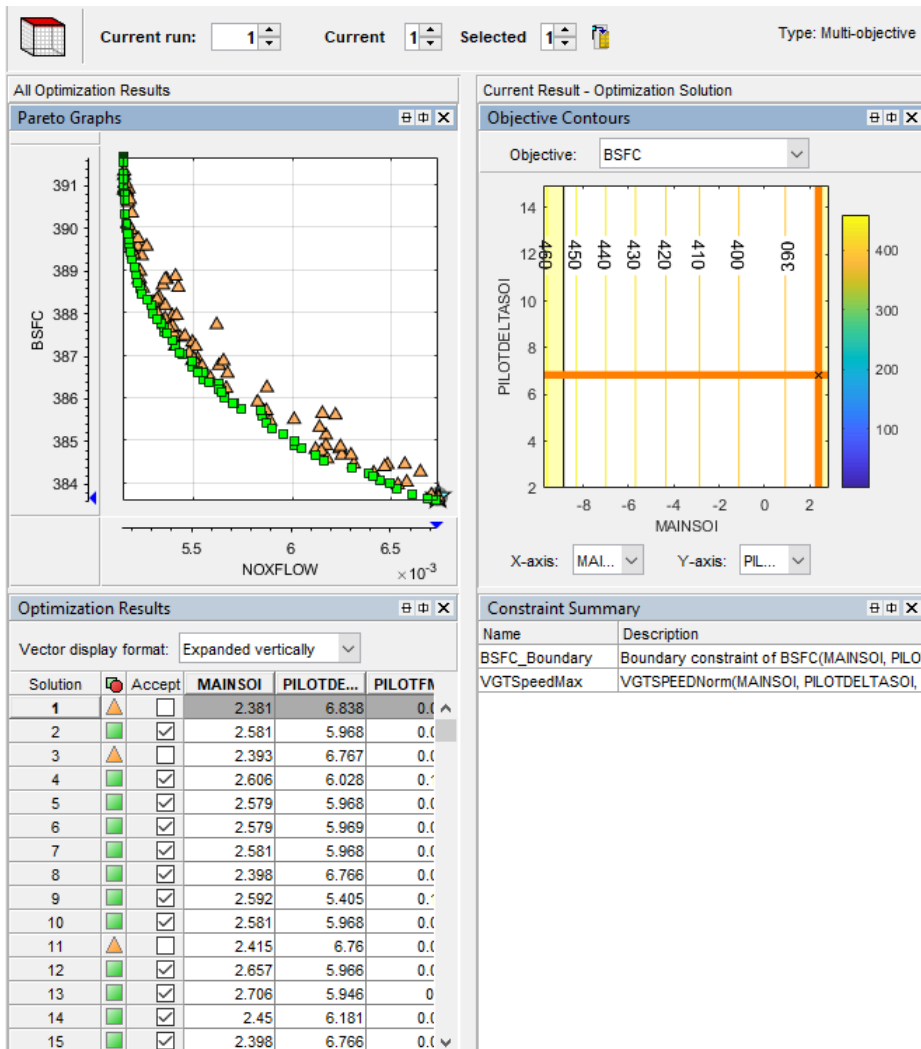
4   Select the `Sum_BSFC_NOX` node to see how the sum optimization is set up. The sum optimization was created from the point optimization results. Observe the 2 objectives and all the constraints.

5   Expand the `Sum_BSFC_NOX` node and select the `Sum_BSFC_NOX_Output` to view the results.

Tip Learn how MathWorks Consulting helps customers develop engine calibrations that optimally balance engine performance, fuel economy, and emissions requirements: see Optimal Engine Calibration.

Model Quickstart

Use a Two-Stage Model To Predict Engine Torque

This two-stage modeling example shows you how to create a statistical model of an engine that predicts the engine brake torque as a function of spark, engine speed, load, and air/fuel ratio. One-stage modeling fits a model to all the data in one process, without accounting for the structure of the data. When data has an obvious hierarchical structure (as here), two-stage modeling is better suited to the task.

The usual way for collecting brake torque data is to fix engine speed, load, and air/fuel ratio within each test and sweep the spark angle across a range of angles. For this experimental setup, there are two sources of variation. The first source is variation within tests when the spark angle is changed. The second source of variation is between tests when the engine speed, load, and air/fuel ratio are changed. The variation within a test is called local, and the variation between tests, global. Two-stage modeling estimates the local and global variation separately by fitting local and global models in two stages. A local model is fitted to each test independently. The results from all the local models are used to fit global models across all the global variables. Once the global models have been estimated, they can be used to estimate the local models' coefficients for any speed, load, and air/fuel ratio. The relationship between the local and global models is shown in this block diagram.
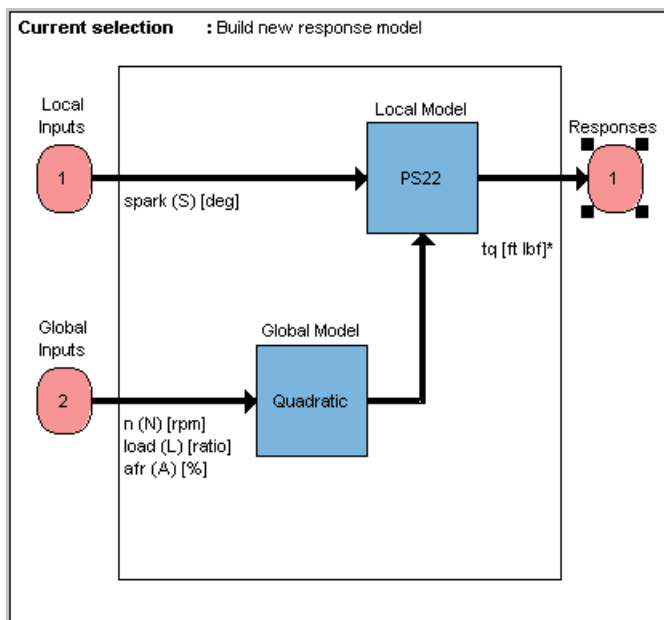
To get started with two-stage modeling, follow these workflow steps.

Workflow Steps	Description
"Open the App and Load Data" on page 5-3	Set up your local and global models, select data for modeling, and specify a response to be modeled.
"Set Up the Model" on page 5-3	Start the toolbox and load and view some data for modeling.
"Verify the Model" on page 5-5	Examine the model fit to the data by looking at the local, global, and two-stage response models.

Workflow Steps	Description
"Export the Model" on page 5-7	Export your completed model, for example, for use in the CAGE part of the toolbox for calibrating.
"Create Multiple Models to Compare" on page 5-8	Useful methods for creating multiple different models to search for the best possible fit to the data.

Open the App and Load Data

1 In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.

2 If you have never used the toolbox before, the User Information dialog box appears. If you want, you can fill in any or all of the fields: your name, company, department, and contact information, or you can click **Cancel**. The user information is used to tag comments and actions so that you can track changes in your files (it does not collect information for MathWorks).

3 When you finish with the User Information dialog box, click **OK**.

The Model Browser window appears.

Load the example data file `holliday.xls`:

1 In the Model Browser, click **Import data**.

2 In the Select data file to import dialog box, open the file `holliday.xls`. The file is in the `<matlabroot>`/toolbox/mbc/mbctraining folder.

The Data Editor opens.

3 View plots of the data in the Data Editor by selecting variables and tests in the lists on the left side. Have a look through the data to get an idea of the shape of curve formed by plotting torque against spark.

Use the editor to prepare your data before model fitting.

4 Close the Data Editor to accept the data and return to the Model Browser.

This data is from Holliday, T., "The Design and Analysis of Engine Mapping Experiments: A Two-Stage Approach," Ph.D. thesis, University of Birmingham, 1995.

Set Up the Model

Specifying Model Inputs

You can use the imported data to create a statistical model of an automobile engine that predicts the torque generated by the engine as a function of spark angle and other variables.

1 In the Model Browser, click **Fit Models**.

2 In the Fit Models dialog box, observe that the `Data Object` you imported is selected in the **Data set** list.

3 Click the **Two-Stage** test plan icon in the **Template** pane.

4 In the **Inputs and Responses** pane, select data channels to use for the responses you want to model.

The model you are building is intended to predict the torque generated by an engine as a function of spark angle at a specified operating point defined by the engine speed, air/fuel ratio, and load. The input to the local model is therefore the spark angle, and the response is torque.

The inputs to the global model are the variables that determine the operating point of the system being modeled. In this example, the operating point of the engine is determined by the engine speed in revolutions per minute (rpm - often called N), load (L), and air/fuel ratio (afr).

a Select `spark` in the **Data channels** list and click the button to add it to the **Local inputs** list.

b Select `n`, `load`, and `afr` in the **Data channels** list and click the button to add them to the **Global inputs** list.
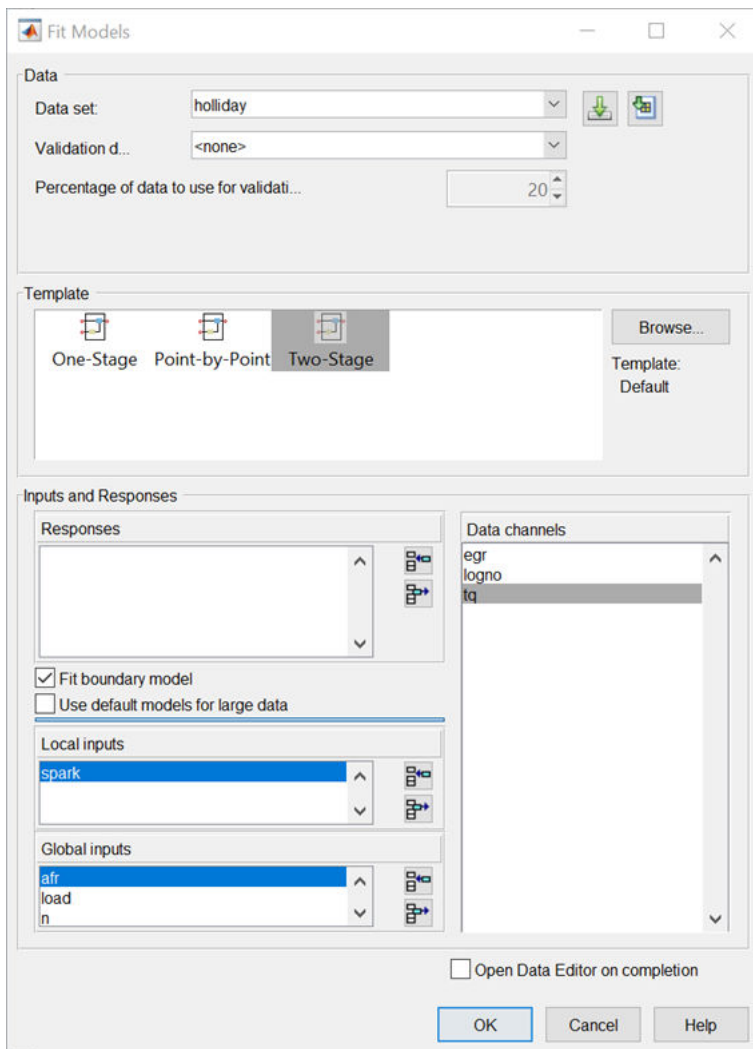
5 Leave the responses empty, and click **OK**.

The default name of the new test plan, `Two-Stage`, appears in the Model Browser tree, in the **All Models** pane.

In this window, the left pane, **All Models**, shows the hierarchy of the models currently built in a tree. At the start, only one node, the project, is in the tree. As you build models, they appear as child nodes

of the project. The right panes change, depending on the tree node selected. You navigate to different views by selecting different nodes in the model tree.

Setting Up the Response Model

To achieve the best fit to torque/spark sweeps, you need to change the local model type from the default. The type of a local model is the shape of curve used to fit the test data, for example, quadratic, cubic, or polynomial spline curves. In this example, you use polynomial spline curves to fit the test data. A spline is a curve made up of pieces of polynomial, joined smoothly together. The points of the joins are called knots. In this case, there is only one knot. These polynomial spline curves are useful for torque/spark models, where different curvature is required above and below the maximum.

To change from the default models and specify polynomial spline as the local model type,

1   In the Model Browser, select the test plan node `Two-Stage`, and in the **Common Tasks** pane, click **Fit Models**. A dialog box asks if you want to change all the test plan models. Click **Yes**.

2   In the Fit Models Wizard, click **Next** to continue using the currently selected data.

3   The next screen shows the model inputs you already selected. Click **Next**

4   To choose the response, on the Response Models screen, select `tq` and click **Add**.

5   Edit the **Local model** type by clicking **Set Up**.

   The Local Model Setup dialog box appears.

   a   Select `Polynomial spline` from the **Local model class** list.

   b   Edit the **Spline Order Below knot** to 2, and leave **Above knot** set to 2.

   c   Click **OK** to close the dialog box.

6   Select `Maximum` under **Datum**. Only certain model types with a clearly defined maximum or minimum can support datum models.

7   Click **Finish**.

   The Model Browser calculates local and global models using the test plan models you just set up.

Notice that the new name of the local model class, `PS` (for polynomial spline) `2,2` (for spline order above and below knot) now appears on a new node in the tree in the **All Models** pane, called `PS22`.

Verify the Model

Verifying the Local Model

The first step is to check that the local models agree well with the data:

1   If necessary, select `PS22` (the local node) on the Model Browser tree.

   The **Local Model** view appears, displaying the local model fitting the torque/spark data for the first test and diagnostic statistics that describe the fit. The display is flexible in that you can drag, open, and close the divider bars separating the regions of the screen to adjust the view.

2   View local model plots and statistics. The Sweep Plot shows the data being fitted by the model (blue dots) and the model itself (line). The red spot shows the position of the polynomial spline knot, at the datum (maximum) point.

3 Look for problem tests with the RMSE Plots. The plot shows the standard errors of all the tests, both overall and by response feature. Navigate to a test of interest by double-clicking a point in the plot to select the test in the other plots in the local model view.

4 In the Diagnostic Statistics plot pane, click the **Y-axis factor** pop-up menu and select `Studentized residuals`.

5 Scroll through local models test by test using the **Test** arrows at the top left, or by using the **Select Test** button.

6 Select Test 588. You see a data point outlined in red. This point has automatically been flagged as an outlier.

7 Right-click the plot and select **Remove Outliers**. Observe that the model is refitted without the outlier.

Both plots have right-click pop-up menus offering various options such as removing and restoring outliers and confidence intervals. Clicking any data point marks it in red as an outlier.

You can use the **Test Notes** pane to record information on particular tests. Each test has its own notes pane. The test numbers of data points with notes recorded against them are colored in the global model plots, and you can choose the color using the **Test Number Color** button in the **Test Notes** pane. Quickly locate tests with notes by clicking **Select Test**.

Verifying the Global Model

The next step is to check through the global models to see how well they fit the data:

1 Expand the `PS22` local node on the Model Browser tree by clicking the plus sign (+) to the left of the icon. Under this node are four response features of the local model. Each of these is a feature of the local model of the response, which is torque.

2 Select the first of the global models, `knot`.

You see a dialog box asking if you want to update fits, because you removed an outlier at the local level. Click **Yes**.

The **Response Feature** view appears, showing the fit of the global model to the data for `knot`. Fitting the local model is the process of finding values for these coefficients or *response features*. The local models produce a value of `knot` for each test. These values are the data for the global model for `knot`. The data for each response feature come from the fit of the local model to each test.

Use the plots to assess model fits.

3 Select the response feature `Bhigh_2`. One outlier is marked. Points with an absolute studentized residual value of more than 3 are automatically suggested as outliers (but included in the model unless you take action). You can use the right-click menu to remove suggested outliers (or any others you select) in the same way as from the Local Model plots. Leave this one. If you zoom in on the plot (**Shift**-click-drag or middle-click-drag) you can see the value of the studentized residual of this point more clearly. Double-click to return to the previous view.

4 Select the other response features in turn: `max` and `Blow 2`. You see that `Blow 2` has a suggested outlier with a very large studentized residual; it is a good distance away from all the other data points for this response feature. All the other points are so clustered that removing this one could greatly improve the fit of the model to the remaining points, so remove it.

Creating the Two-Stage Model

Recall how two-stage models are constructed: two-stage modeling partitions the variation separately between tests and within tests, by fitting local and global models separately. A model is fitted to each test independently (local models). These local models are used to generate global models that are fitted across all tests.

For each sweep (test) of spark against torque, you fit a local model. The local model in this case is a spline curve, which has the fitted response features of `knot`, `max`, `Bhigh_2`, and `Blow_2`. The result of fitting a local model is a value for `knot` (and the other coefficients) for each test. The global model for `knot` is fitted to these values (that is, the `knot` global model fits `knot` as a function of the global variables). The values of knot from the global model (along with the other global models) are then used to construct the two-stage model

The global models are used to reconstruct a model for the local response (in this case, torque) that spans all input factors. This is the two-stage model across the whole global space, derived from the global models.

After you are satisfied with the fit of the local and global models, it is time to construct a two-stage model from them.

1   Return to the Local Model view by clicking the local node `PS22` in the Model Browser tree.

2   To create a two-stage model, click **Create Two-Stage** in the Common Tasks pane.

Comparing the Local Model and the Two-Stage Model

1   Now the plots in the **Local Model** view show two lines fitted to the test data. Scroll though the tests using the left/right arrows or the **Select Test** button at the top left. The plots now show the fit of the two-stage model for each test (green circles and line), compared with the fit of the local model (blue line) and the data (blue dots). Zoom in on points of interest by **Shift**-click-dragging or middle-click-dragging. Double-click to return the plot to the original size.

Compare how close the two-stage model fit is to both the data and the local fit for each test.

2   Notice that the local model icon has changed (from the local icon showing a house, to a two-stage icon showing a house and a globe) to indicate that a two-stage model has been calculated.

Response Node

Click the Response node (`tq`) in the Model Browser tree.

Now at the Response node in the Model Browser tree (`tq`), which was previously blank, you see plots showing you the fit of the two-stage model to the data. You can scroll through the tests, using the arrows at top left, to view the two-stage model against the data for groups of tests.

You have now completed setting up and verifying a two-stage model.

Export the Model

All models created in the Model Browser are exported using the **File** menu. A model can be exported to the MATLAB workspace, to CAGE, or to a Simulink model.

1   Click the `tq` node in the model tree.

2   Choose **File > Export Models**. The Export Model dialog box appears.

3   For the **Export to** parameter, select the `Simulink`, `Workspace`, or `CAGE`.

4   Click **OK** to export the models.

To import models into CAGE to create calibrations, use the CAGE Import Tool instead for more flexibility.

Create Multiple Models to Compare

Methods For Creating More Models

After you have fitted and examined a single model, you normally want to create more models to search for the best fit. You can create individual new models or use the **Create Alternatives** common task to create a selection of models at once, or create a template to save a variety of model settings for reuse.

To use the Model Template dialog box to quickly create a selection of different child nodes to compare, click **Create Alternatives** in the Common Tasks pane. The following exercises show you examples of these processes.

Creating New Local Models

To follow these examples, you need to create the initial models.

1   As an example, select the `tq` response node and click **New Local Model** in the Common Tasks pane.

The Local Model Setup dialog box appears.

2   Select a `Polynomial Spline`, and edit the spline order to 3 below the knot and 2 above. Click **OK**.

A new set of local models (and associated response feature models) is calculated.

3   Return to the parent `tq` response node , and click **New Local Model** again, in the Common Tasks pane.

4   Select a `Polynomial` with an order of 2 in the Local Model Setup dialog box. Click **OK**.

A new set of local models and response feature models is calculated.

Now you have three alternative local models to compare: two polynomial splines (order 3,2 and order 2,2) and a polynomial (order 2), as shown.
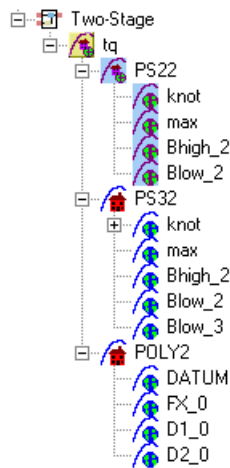
You can select the alternative local models in turn and compare their statistics. For an example, follow these steps:

1 Select the new local model node PS32.

2 Select test 587 in the **Test** edit box.

3 In the **Local statistics** pane, observe the value of RMSE (root mean squared error) for the current (i^th) test.

   The RMSE value is our basic measure of how closely a model fits some data, which measures the average mismatch between each data point and the model. This is why you should look at the RMSE values as your first tool to inspect the quality of the fit — high RMSE values can indicate problems.

4 Now select the local model node POLY2 and see how the value of RMSE changes.

   Observe that the shape of the torque/spark sweep for this test is better suited to a polynomial spline model than a polynomial model. The curve is not symmetrical because curvature differs above and below the maximum (marked by the red cross at the datum). This explains why the value of RMSE is much lower for PS32 (the polynomial spline) than for the POLY2 (polynomial) for this test. The polynomial spline is a better fit for the current test.

5 Look through some other tests and compare the values of RMSE for the different local models. To choose the most suitable local model you must decide which fits the majority of tests better, as there are likely to be differences among best fit for different tests.

6 To help you quickly identify which local models have the highest RMSE, indicating problems with the model fit, check the RMSE Plots.

   a Use the plot to help you identify problem tests. Use the drop-down menus to change the display. For example, select s_knot to investigate the error values for knot (MBT), or RMSE to look at overall error.

   b You can navigate to a test of interest from the RMSE Plots by double-clicking a point in the plot to select the test in the other plots.

7 Look at the value of Local RMSE reported in the **Pooled Statistics** pane on the right (this is pooled between all tests). Now switch between the POLY2 and the PS32 local models again and observe how this value changes.

8 You can compare these values directly by selecting the parent tq response node, when the Local RMSE is reported for each child local model in the list at the bottom.

When all child models have a two-stage model calculated, you can also compare two-stage values of RMSE here. Remember, you can see statistics to compare the list of child models of the response node in this bottom list pane.

When comparing models, look for lower RMSE values to indicate better fits. However, remember that a model that interpolates between all the points can have an RMSE of zero but be useless for predicting between points. Always use the graphical displays to visually examine model fits and beware of "overfitting" — chasing points at the expense of prediction quality. You will return to the problem of overfitting in a later section when you have two-stage models to compare.

Adding New Response Features

Recall that two-stage models are made up of local models and global models. The global models are fitted to the response features of the local models. The response features available are specific to the type of local model. You can add different response features to see which combination of response features makes the best two-stage model as follows:

1   Select the local model node PS32.

2   Select **File > New Response Feature**.

    A dialog box appears with a list of available response features.

3   Select f(x+datum) from the list and enter -10 in the **Value** edit box. Click **OK**.

    A new response feature called FX_less10 is added under the PS32 local model. Recall that the datum marks the maximum, in this case maximum torque. The spark angle at maximum torque is referred to as maximum brake torque (MBT). You have defined this response feature (f(x+datum)) to measure the value of the model (torque) at (-10 + MBT) for each test. It can be useful to use a response feature like this to track a value such as maximum brake torque (MBT) minus 10 degrees of spark angle. This response feature is not an abstract property of a curve, so engineering knowledge can then be applied to increase confidence in the models.

4   Select the local node PS32, and click **Create Two-Stage** in the Common Tasks pane. The Model Selection window opens, because you now need to choose 5 of the 6 response features to form the two-stage model.

    In the Model Selection window, observe four possible two-stage models in the Model List. This is because you added a sixth response feature. Only five (which must include knot) are required for the two-stage model, so you can see the combinations available and compare them. Note that not all combinations of five response features can completely describe the shape of the curve for the two-stage model, so only the possible alternatives are shown.

5   Close the Model Selection window and click **Yes** to accept one of the models as best.

    Notice that the response features chosen to calculate the two-stage model are highlighted in blue, and the unused response feature is not highlighted.

6   Select the tq response node to see a comparison of the statistics of both two-stage models (your original PS22 and the new PS32).

    Remember that the POLY2 local model has no two-stage model yet; no two-stage statistics are reported for POLY2 in the bottom list pane. You cannot fully compare the two-stage models until every local model in the test plan has a two-stage model calculated.

7   To calculate the two-stage model for POLY2, in the Common Tasks pane, click **Create Two-Stage**.

Comparing Models

1   Now you have three two-stage models. Select the `tq` response node and look at the statistics, particularly Local RMSE and Two-Stage RMSE reported in the list of child models at the bottom.

- Look for lower RMSE values to indicate better fits.
- Look for lower PRESS RMSE values to indicate better fits without overfitting. PRESS RMSE is a measure of the predictive power of your models.

  It is useful to compare PRESS RMSE with RMSE as this may indicate problems with overfitting. RMSE is minimized when the model gets close to each data point; "chasing" the data will therefore improve RMSE. However, chasing the data can sometimes lead to strong oscillations in the model between the data points; this behavior can give good values of RMSE but is not representative of the data and will not give reliable prediction values where you do not already have data. The PRESS RMSE statistic guards against this by testing how well the current model would predict each of the points in the data set (in turn) if they were not included in the regression. To get a small PRESS RMSE usually indicates that the model is not overly sensitive to any single data point.

- Look for lower T^2 values. A large T^2 value indicates that there is a problem with the response feature models.
- Look for large negative log likelihood values to indicate better fits.

2   To compare all three two-stage models simultaneously, select **Model > Selection Window**. Here you can see the same statistics to compare the models in the bottom list, but you can also make use of a variety of views to look for the best fit:

- You can plot the models simultaneously on the Tests, Residuals and Cross Section views (**Shift**- or **Ctrl**-click to select models in the list)
- You can view each model in the Response Surface view as a surface; movie, contour or multiline plot, and as a table

3   You can select a model and click **Assign Best** in the Model Selection window, or double-click a model to assign it as best.

4   When you close the Model Selection window and return to the Model Browser, the model you selected as best is copied to the parent response node, `tq`.

Creating New Global Models

In this example, you have not yet searched for the best global model types. You would normally do this before creating and comparing two-stage models. For the purpose of this tutorial, you have already created two-stage models and used RMSE to help you identify better models. The principle is the same at each level in the model tree: add new child models and choose the best. You can create any number of child nodes to search for the best global model fit for each response feature in your tree.

1   Select the local node `POLY2`.

2   To create a selection of alternatives for each response feature node, in the Common Tasks pane, click **Build Global Models**.

3   In the Model Template dialog box, click **New**, then click **OK**.

4   Observe the default list of a variety of model types, then click **OK**. It is worth trying the default model settings for a quick exploration of the trends in the data.

5   In the Model Selection dialog box, leave the default selection criterion for automatically choosing the best child node, and click **OK**.

The toolbox builds the models and selects the best using your selection criteria.

> **Note** The toolbox automatically builds models in parallel if you have Parallel Computing Toolbox.

6 Assess all the fits in the Alternative Models list in case you want to choose an alternative as a better fit.

7 Notice that the child node model assigned as best is highlighted in blue in the Alternative Models list and the model tree. The local node has changed from the two-stage icon back to the local model icon (a red house). This is because you have changed the response feature models, and so you need to recalculate the two-stage model using the new global models for the response features.

When you have chosen best global models for all your response features, you need to recalculate the two-stage model.

8 When you have chosen a best model among alternatives, it can be useful to clean up the rejected models by selecting **Delete Alternatives** in the Common Tasks pane. You can also select **File > Clean Up Tree**. This deletes all rejected child models where best models have been chosen; only the child nodes selected as best remain.

You can use the Model Template dialog box to create and save templates of model types you often want to build. Creating a template containing a list of all the models you want is an efficient way to quickly build a selection of alternative model child nodes for many global models. Use these techniques to find models well suited to the data for each of your global models.

See Also

More About

- "Fit a One-Stage Model"
- "Two-Stage Models for Engines"
- "Data Manipulation for Modeling"
- "Gasoline Engine Calibration"
- "Multi-Injection Diesel Calibration"

Generate Current Controller Calibration Tables for Flux-Based Motor Controllers

Using the Model-Based Calibration Toolbox, you can generate optimized calibration tables for flux-based motor controllers. This example shows how to import data, fit a model, and optimize the data based on objectives and constraints.

Based on nonlinear motor flux data, the calibration tables optimize:

- Motor efficiency
- Maximum torque per ampere (MTPA)
- Flux weakening

The calibration tables are d- and q- axis reference currents as functions of motor torque and motor speed.

To generate optimized current calibration tables, follow these workflow steps.

Workflow Steps	Description
"Collect and Post Process Motor Data" on page 5-14	Collect the nonlinear motor flux data from dynamometer testing or finite element analysis (FEA). For this example, file PMSMEfficiencyData.xlsx contains the data that you need: • Total flux, Ψ_{total}, in Wb • Allowed flux, Ψ_{max}, in Wb • d-axis flux, Ψ_d, in Wb • q-axis flux, Ψ_q, in Wb • d-axis current, I_d, in A • q-axis current, I_q, in A • Current magnitude, I_s, in A • Motor torque, T_e, in N·m • Motor speed, n, in rpm

Workflow Steps	Description
"Model Motor Data" on page 5-16	Use a point-by-point model to fit the `PMSMEfficiencyData.xlsx` data. Specifically: • Import data • Filter and group data • Fit model The PMSM maximum efficiency calibration case study contains the model fit. **Case Studies** Examples • Dual CAM gasoline engine with spark optimiz... • Multi-injection diesel tested with pilot injection ... • CI Mapped Engine – Fuel Input • CI Mapped Engine – Torque Input • SI Mapped Engine – Torque Input • PMSM maximum efficiency calibration
"Generate Calibration" on page 5-20	Calibrate and optimize the data using objectives and constraints. Specifically: • Create functions. • Create tables from model. • Run an optimization. • Generate and fill optimized current controller calibration tables that are functions of motor torque and motor speed. The PMSM maximum efficiency calibration case study contains the calibration results. **Case Studies** Examples • Dual CAM gasoline engine with spark optimiz... • Multi-injection diesel tested with pilot injection ... • CI Mapped Engine – Fuel Input • CI Mapped Engine – Torque Input • SI Mapped Engine – Torque Input • PMSM maximum efficiency calibration

Collect and Post Process Motor Data

Collect this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA):

- *d*- and *q*- axis current
- *d*- and *q*- axis flux linkage
- Electromagnetic motor torque

Use the collected data and motor speed to calculate the total flux, maximum flux, and current magnitude:

$$\psi_{total} = \sqrt{\psi_d{}^2 + \psi_q{}^2}$$
$$i_s = \sqrt{i_d{}^2 + i_q{}^2}$$
$$n = \frac{60\omega_e}{2\pi P}$$
$$\psi_{max} = \frac{V_{dc}}{\sqrt{3}\omega_e}$$

The equations use these variables:

i_d, i_q	*d*- and *q*- axis current, respectively
i_s,	Current magnitude
Ψ_d, Ψ_q	*d*- and *q*- axis flux linkage, respectively
Ψ_{total}, Ψ_{max}	Total and allowed flux, respectively
ω_e	Electrical motor angular speed, rad/s
n	Motor speed, rpm
V_{dc}	Inverter bus voltage
P	Number of pole pairs

Finally, for each data point, create a file containing:

- Total flux, Ψ_{total}, in Wb
- Allowed flux, Ψ_{max}, in Wb
- *d*-axis flux, Ψ_d, in Wb
- *q*-axis flux, Ψ_q, in Wb
- *d*-axis current, I_d, in A
- *q*-axis current, I_q, in A
- Current magnitude, I_s, in A
- Motor torque, T_e, in N·m
- Motor speed, *n*, in rpm

For this example:

- Pole pairs, *P*, is 4
- Inverter bus voltage, V_{dc}, is 500

the data file `matlab\toolbox\mbc\mbctraining\PMSMEfficiencyData.xlsx` contains the motor flux data.

Model Motor Data

To model the motor data, use the **MBC Model Fitting** app to import, filter, and fit the data with a point-by-point model. For this example, the data file PMSMEfficiencyData.xlsx contains a large data set. You could consider using a design of experiment (DOE) to limit the data. However, the data set represents typical FEA analysis results.

Since there is a simple relationship between the *d*- and *q*-axis currents for fixed torque-speed operating points, the point-by-point model provides an accurate fit.

For comparison, the PMSM maximum efficiency calibration case study contains the model fit.
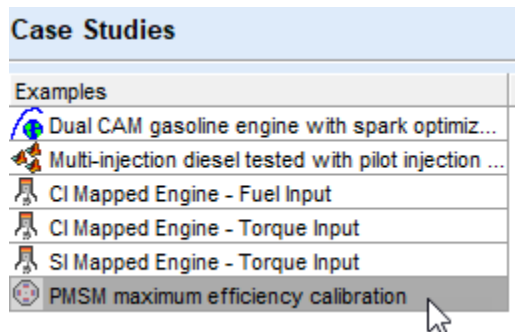
Import Data

For this example, PMSMEfficiencyData.xlsx contains this motor controller data:

- Total flux, Ψ_{total}, in Wb
- Allowed flux, Ψ_{max}, in Wb
- *d*-axis flux, Ψ_d, in Wb
- *q*-axis flux, Ψ_q, in Wb
- *d*-axis current, I_d, in A
- *q*-axis current, I_q, in A
- Current magnitude, I_s, in A
- Motor torque, T_e, in N·m
- Motor speed, *n*, in rpm

1. In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.
2. In the Model Browser home page, click **Import Data**. Click **OK** to open a data source file.
3. Navigate to the matlab\toolbox\mbc\mbctraining folder. Open data file PMSMEfficiencyData.xlsx. The Data Editor opens with your data.
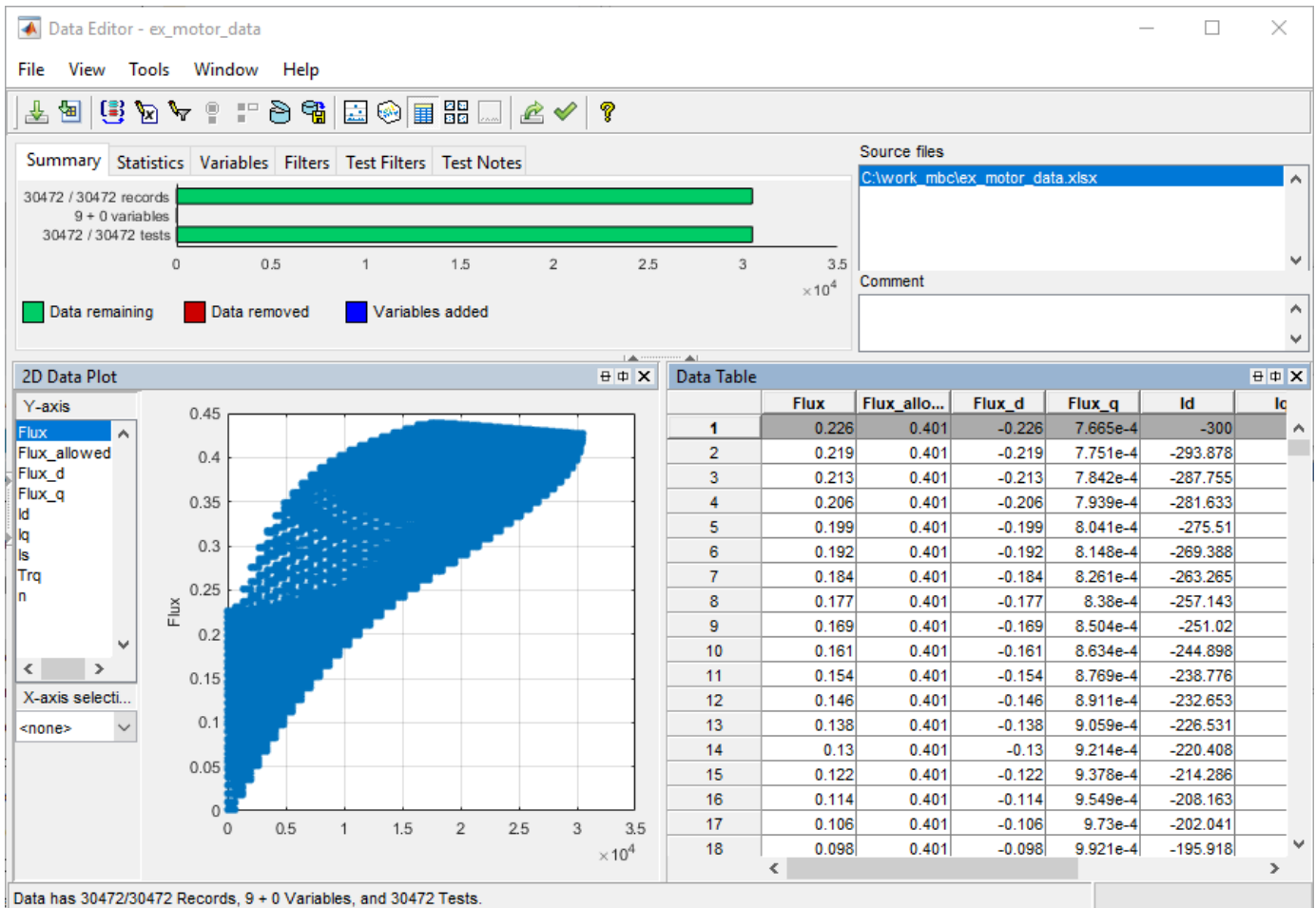
Filter Data

You can filter data to exclude records from the model fit. In this example, set up a filter to include only flux and current magnitudes that are less than a specified threshold. Specifically:

- Current magnitude, I_s, less than or equal to 300 A.
- Total flux, Ψ_{total}, less than or equal to allowed flux Ψ_{max}

1 In the Data Editor, select **Tools** > **Filters** to open the **Filter Editor**. Create these filters:

- Is <= 300
- Flux <= Flux_allowed

Define Test Groupings

For point-by-point models, you need to define test groups. In the example, define groups for motor torque and speed. Set the tolerances to so that Model-Based Calibration Toolbox groups small variations in torque and speed at the same operating point.

1 In the Data Editor, select **Tools > Test Groups** to open the **Define Test Groupings** dialog box. Create groups for the motor torque and speed.

2 Set these tolerances:

- Motor torque, Trq, to `1.000`
- Motor speed, n, to `10.000`
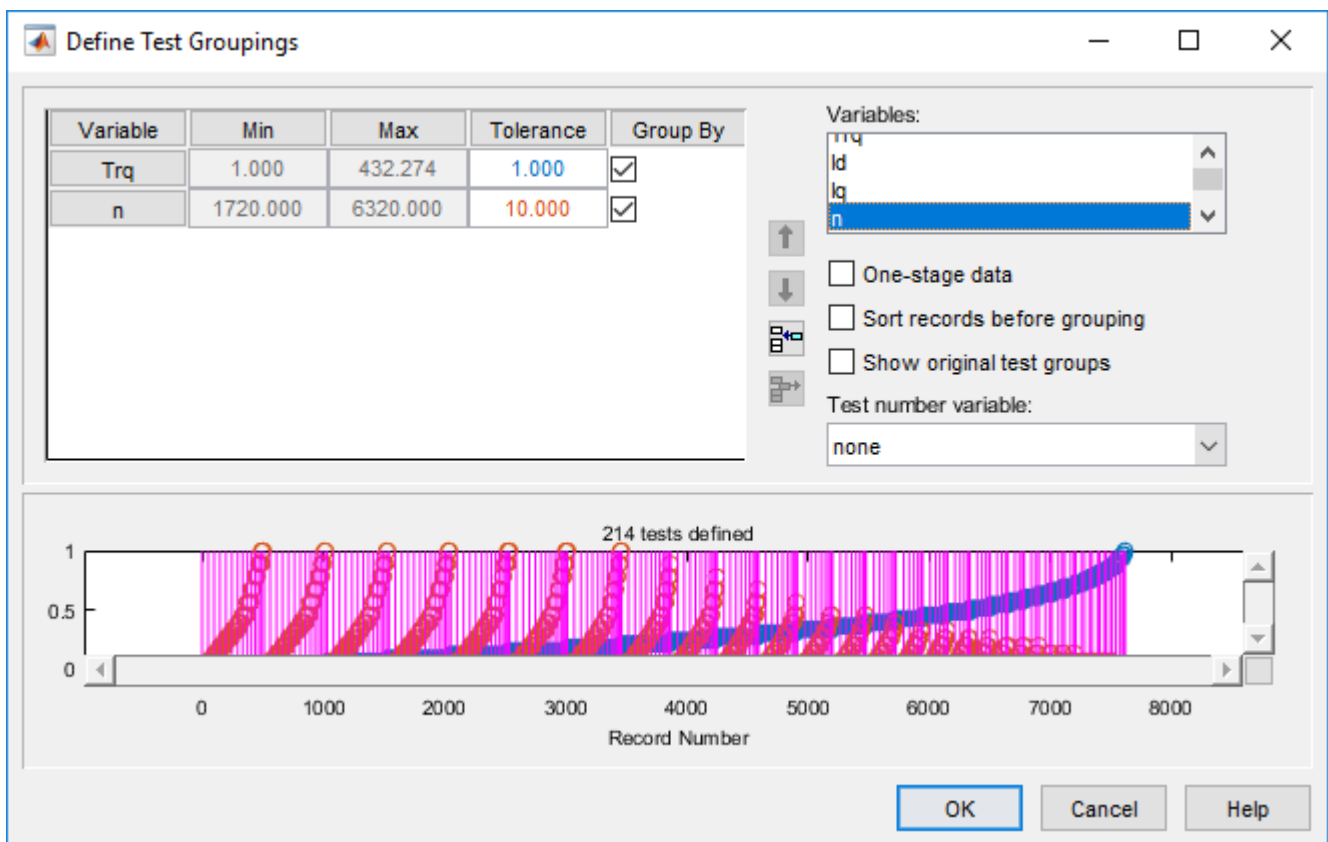
3 In the Data Editor, select **File > Save & Close**. Accept the changes to the data.

Fit Model

Fit the data to a point-by-point model with these responses, local inputs, and operating points:

- Responses

  - q-axis current, I_q, in A
- Local inputs

  - d-axis current, I_d, in A

- Operating points

  - Motor speed, n, in rpm
  - Electromagnetic motor torque, T_e, in N·m

1   In the Model Browser, select **Fit Models**.

2   In **Fit Models**, configure a Point-by-Point model with these responses and inputs.

Responses	Local Inputs	Operating Points
Iq	Id	Trq
		n
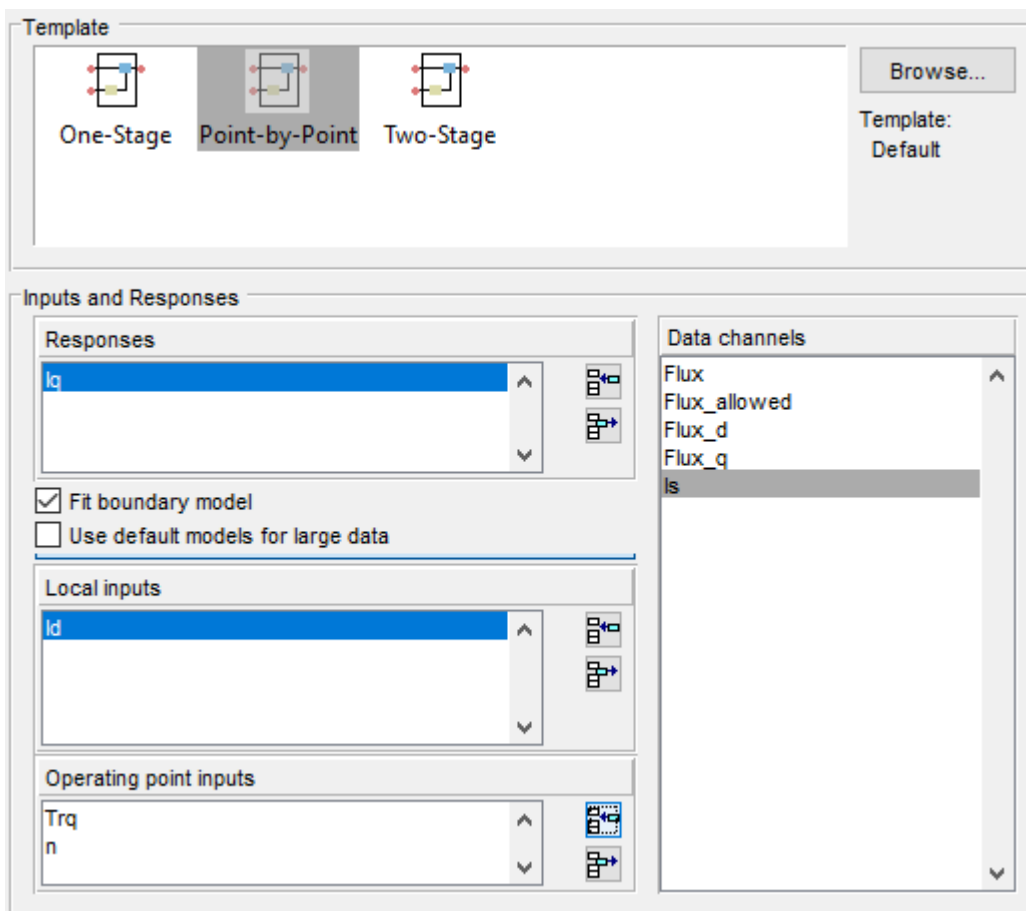
3   To fit the model, select **OK**. If prompted, accept changes to data. By default, the fit uses a Gaussian Process Model (GPM) to fit the data.

4   After the fit completes, examine the response models for I_q. The Model Browser displays information that you can use to determine the accuracy of the model fit.

- In the Model Browser, select Iq. Examine the response surface and diagnostic statistics. These results indicate a reasonably accurate fit. You can browse through each test to examine the response for each torque-speed operating point.
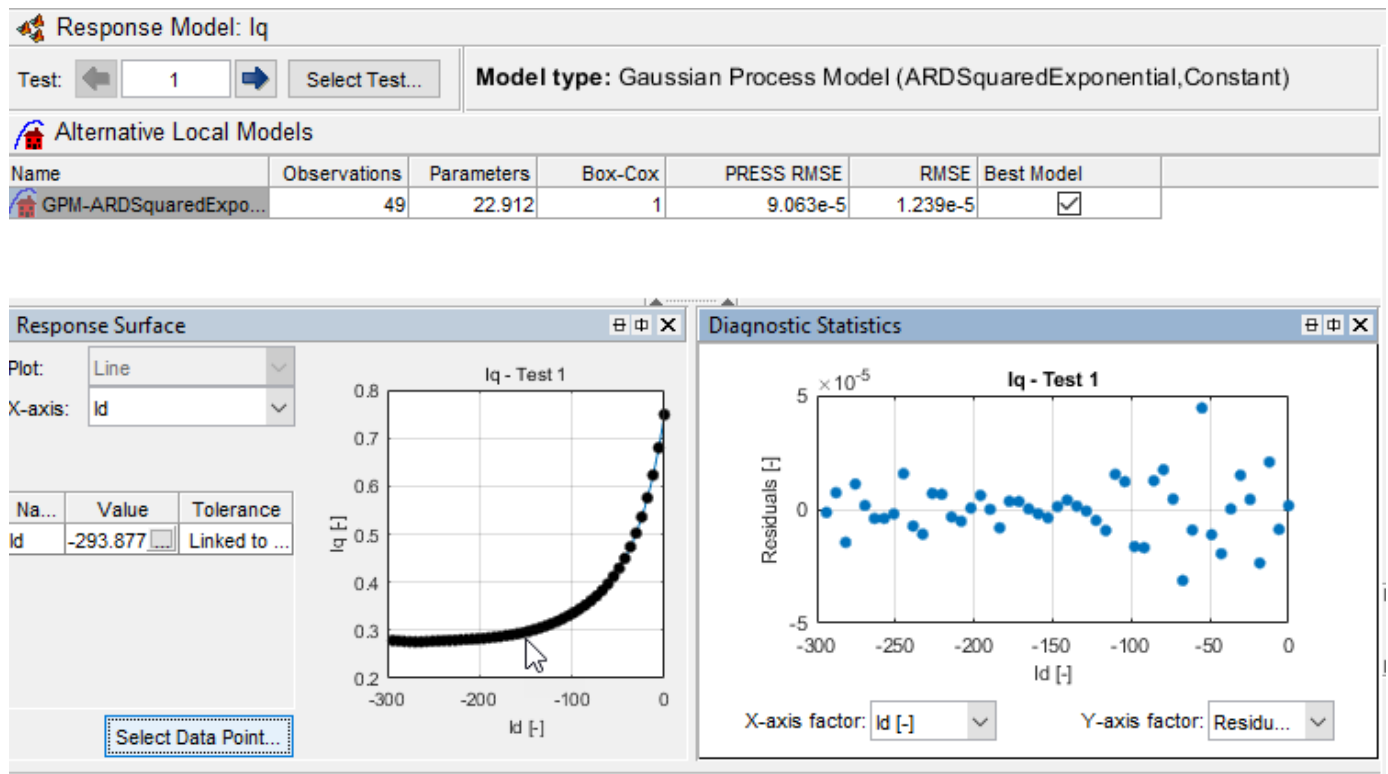
5   Save your project. For example, select **Files > Save Project**. Save `gs_example.mat` to the `work` folder.

Generate Calibration

After you fit the model, create functions and tables, run the optimization, and fill the calibration tables.

For comparison, the PMSM maximum efficiency calibration case study contains the calibration results.
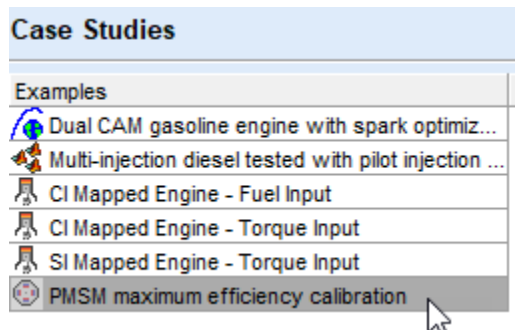
Import Models and Create Functions
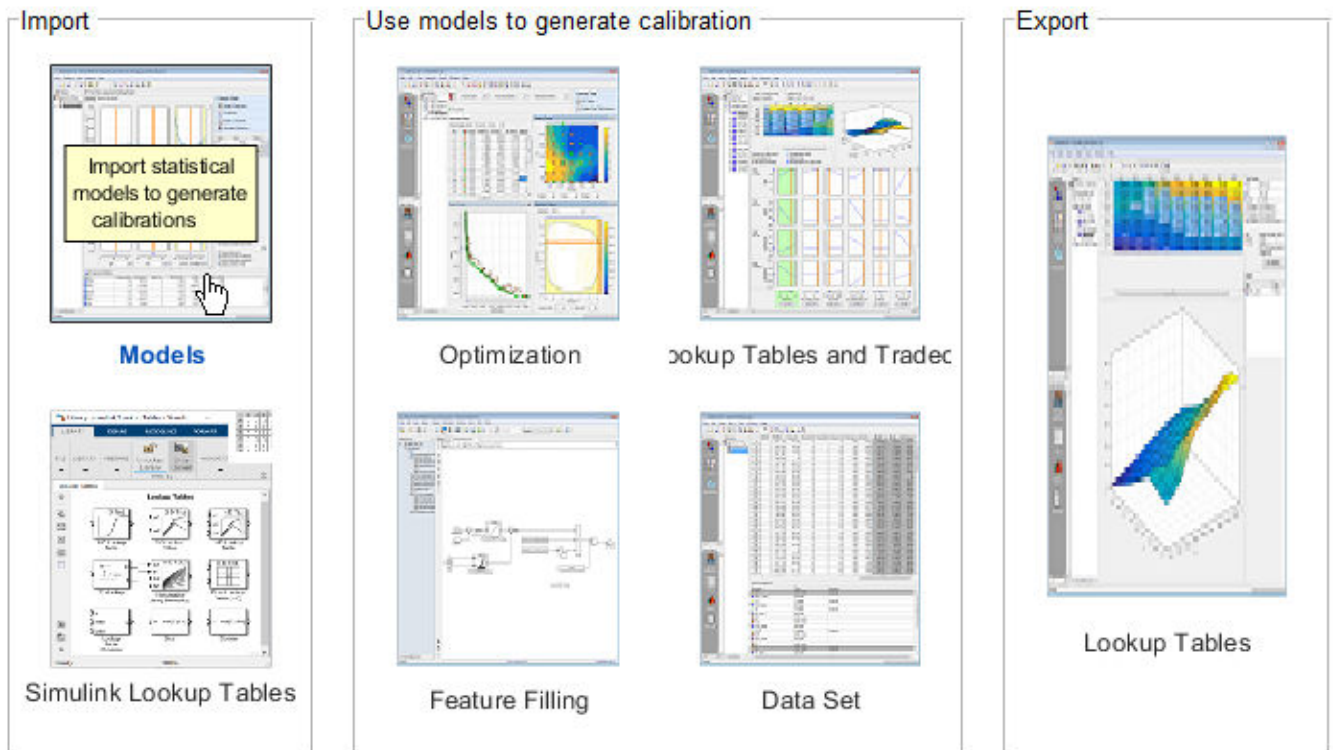
Import models and create the functions to use when you optimize the calibration. In this example, set up functions for:

- Current magnitude, I_s

- Torque per amp, *TPA*

1   In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Optimization**.

2   In the Cage Browser, select **Models**. If it is not already opened, in the MBC Model Fitting browser, open the `gs_example.mat` project.

3   In Import Models, click **OK**. Close the CAGE Import Tool.

4   In the Cage Browser toolbar, use **New Function Model** wizard to create these functions:

- `Is = sqrt(Id^2 + Iq^2)`
- `TPA = Trq/Is`
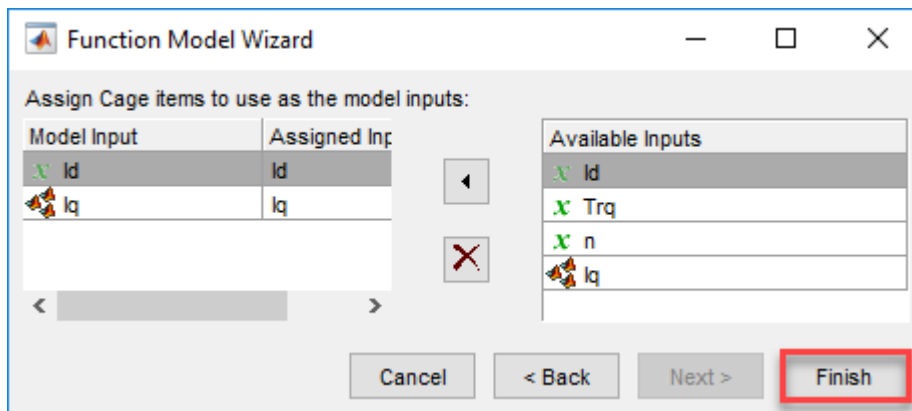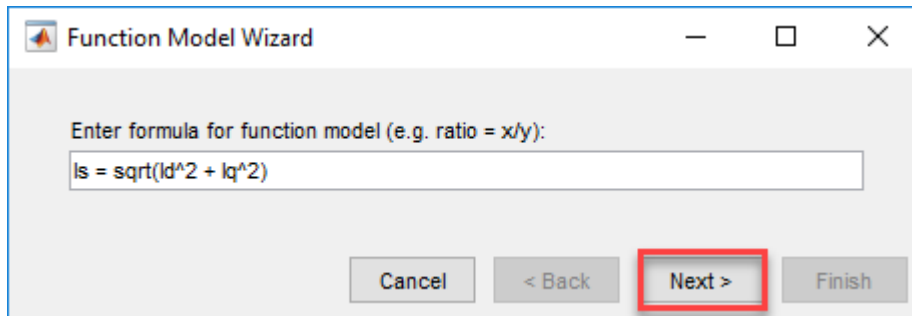
5   In the Cage Browser, verify that the function models for `Is` and `TPA` have these descriptions.

Models					
Name	Type	Inputs	Lower Output Limit	Upper Output Limit	Description
Iq	Point-by-point ...	Id, Trq, n	-Inf	Inf	Created by on 28-Mar-2019.
Is	Function model	Id, Iq	-Inf	Inf	sqrt(Id^2 + Iq^2)
TPA	Function model	Is, Trq	-Inf	Inf	Trq/Is

6   Select **File > Save Project**. Save `gs_example.cag` to the `work` folder.

Create Lookup Tables from Model

Create tables that the Model-Based Calibration Toolboxoptimizers uses to store the optimized parameters. For this example, the tables are:

- *d*-axis current, I_d, as a function of motor torque, `Trq`, and motor speed, n.
- *q*-axis current, I_q, as a function of motor torque, `Trq`, and motor speed, n.

1   In the Cage Browser, select **Lookup Tables and Tradeoff**. In Create Lookup Tables from Model, select `Iq`. Click **Next**.
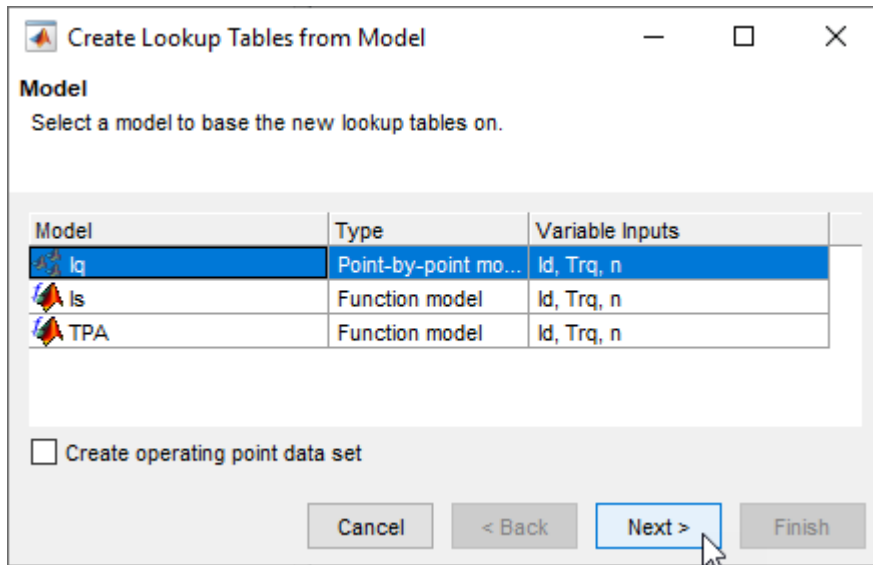
2 In the Create Lookup Tables from Model wizard:

- Clear **Use model operating points**.
- Set **Table rows** to 31.
- Set **Table columns** to 29.
- Click **Next**.
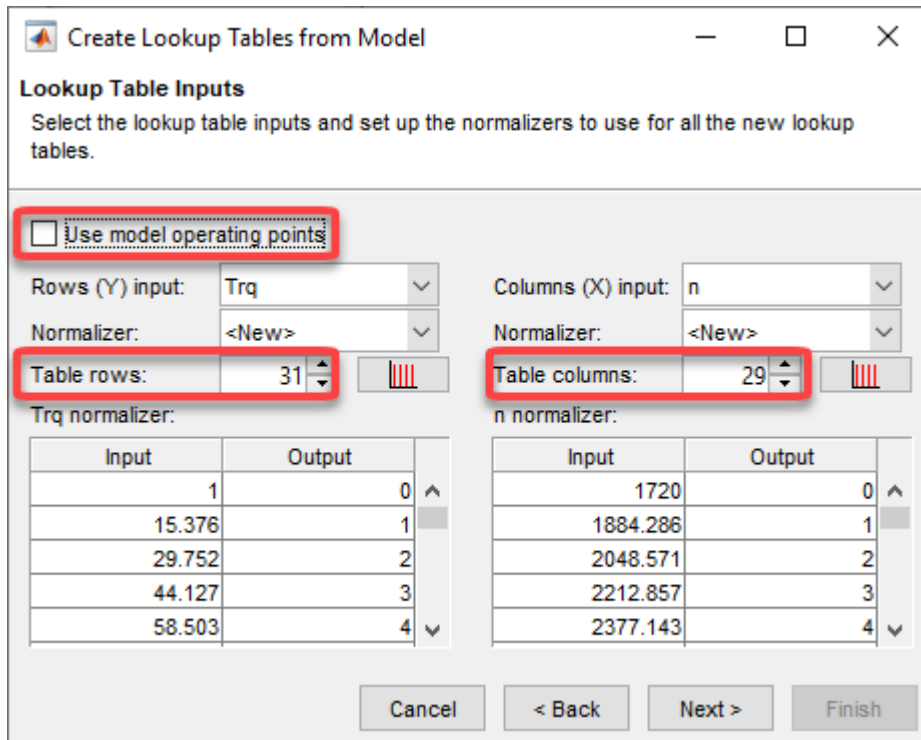
3 In Create Lookup Tables from Model:

- Select `Id` and `Iq`.
- Click **Finish**.
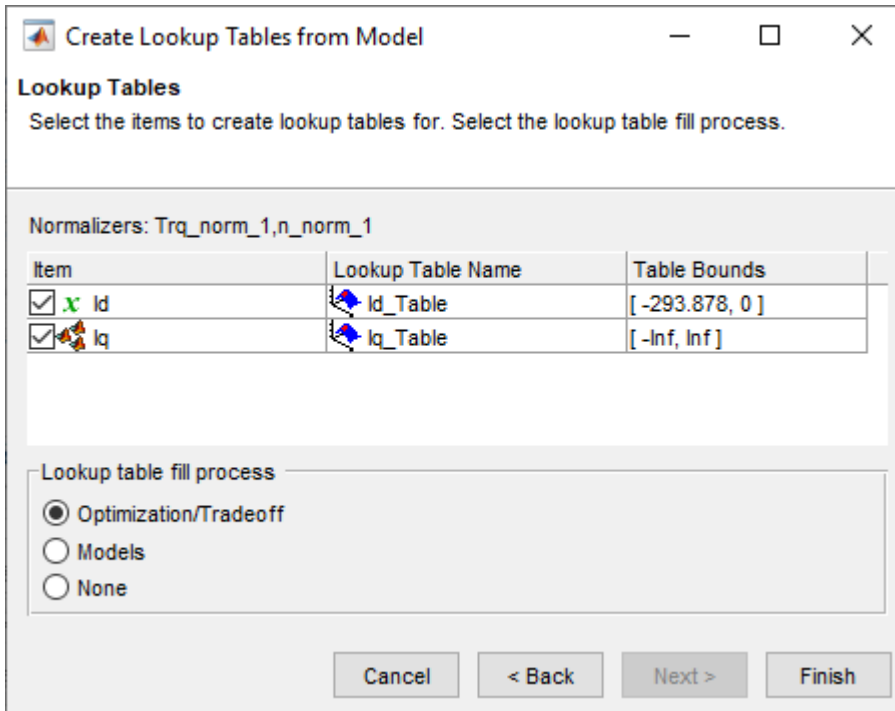
4   In the CAGE Browser, examine the tables.
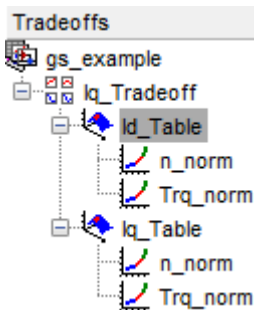
Run Optimization

In this example, run an optimization with these specifications:

- Current magnitude, I_s, less than or equal to 300 A.
- Maximizes torque per ampere, *TPA*.

1   On the Cage Browser home, select **Optimization**.
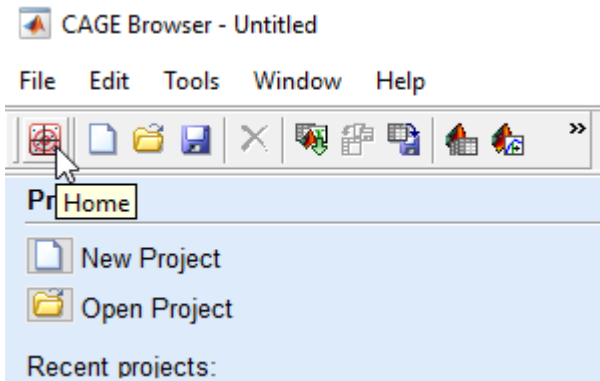
2 In Create Optimization from Model, select TPA and **Next**.
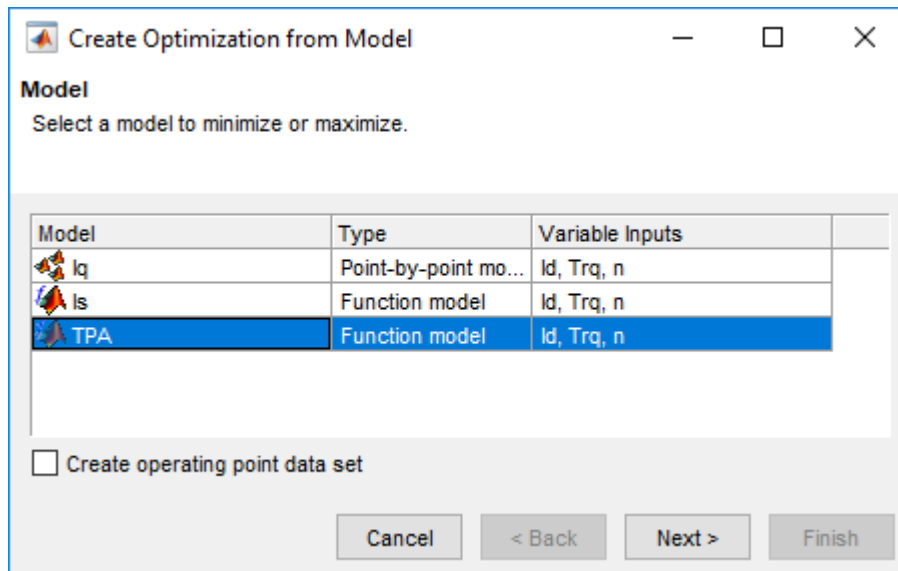
3 In Create Optimization from Model:

- Select `Id`.
- Set **Objective type** to `Maximize`.
- Click **Finish**.
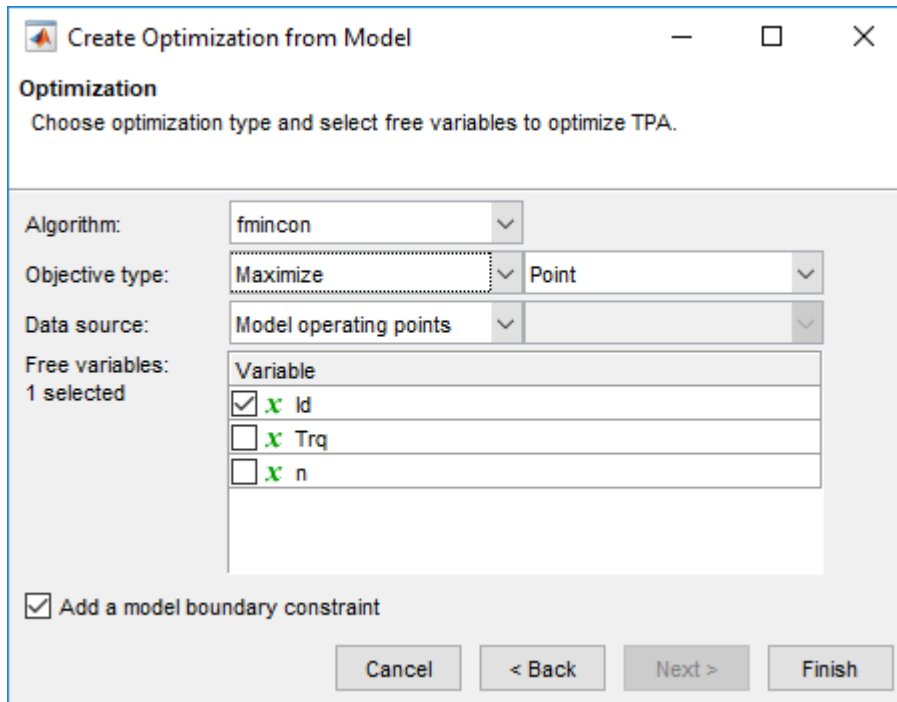
4.   Add the optimization constraint for the current magnitude, I_s. In the CAGE Browser, select **Optimization > Constraints > Add Constraints** to open Edit Constraint. Use the dialog box to create a constraint on the current.

-   `Is <= 300`
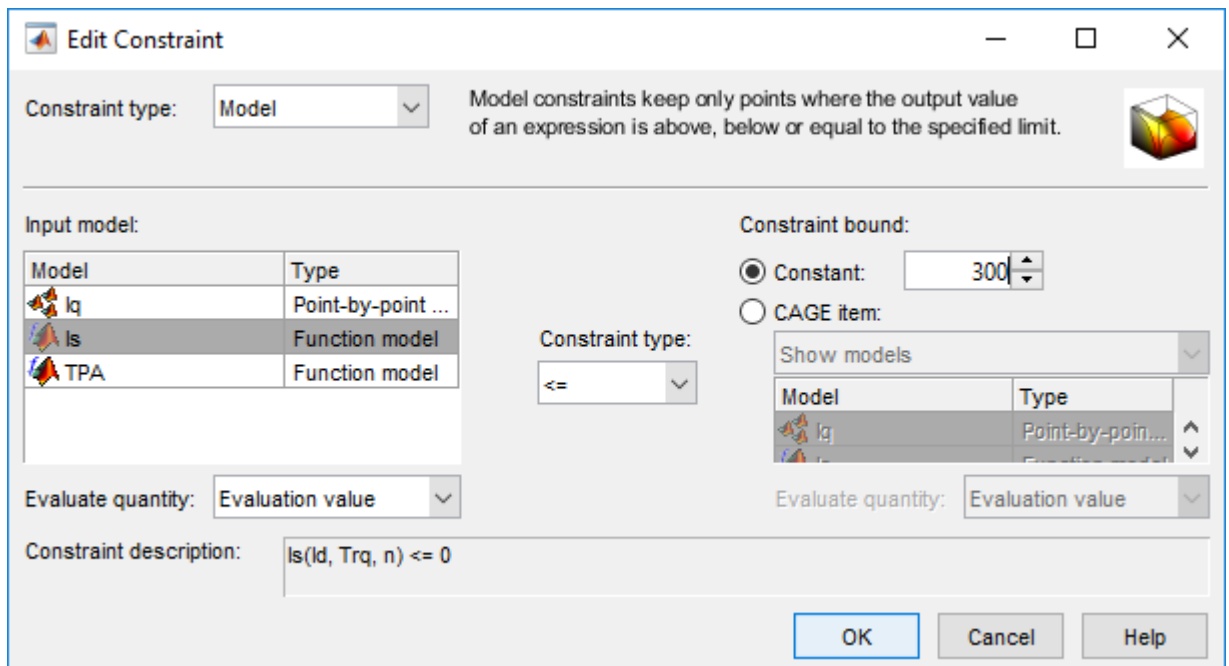
5.   In the Cage Browser, *carefully* verify the Objectives and Constraints.

Objectives		
Name	Description	Type
TPA	TPA(Id, Trq, n)	Maximize

Constraints		
Name	Description	Type
TPA_Boundary	Boundary constraint of TPA(Id, Trq, n)	Model
Is	Is(Id, Trq, n) <= 300	Model

Common Tasks

- Add Constraint...
- Set Up
- Run...
- View Results

Optimization Information

Algorithm name	mbcOSfmincon
Algorithm description	Single objective optimizati
Free variables	Id
Operating point vari...	None
Item scaling	off

Optimization Point Set

Number of operating points: 214

Select Scalar Variables...

Free Variables

Variable:	Id
1	-146.939
2	-146.939
3	-146.939

Fixed Variables

Variable:	Trq	n
1	1	1720
2	1	2020
3	1	2320

6 In the Cage Browser, select **Run**.

The optimization results are similar to these.
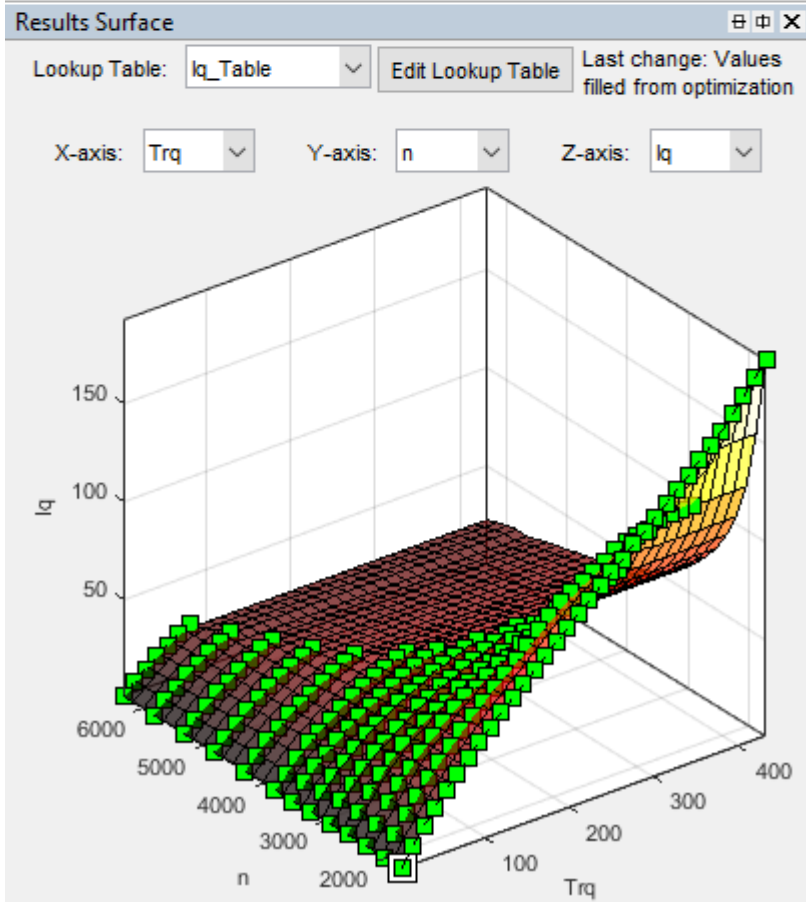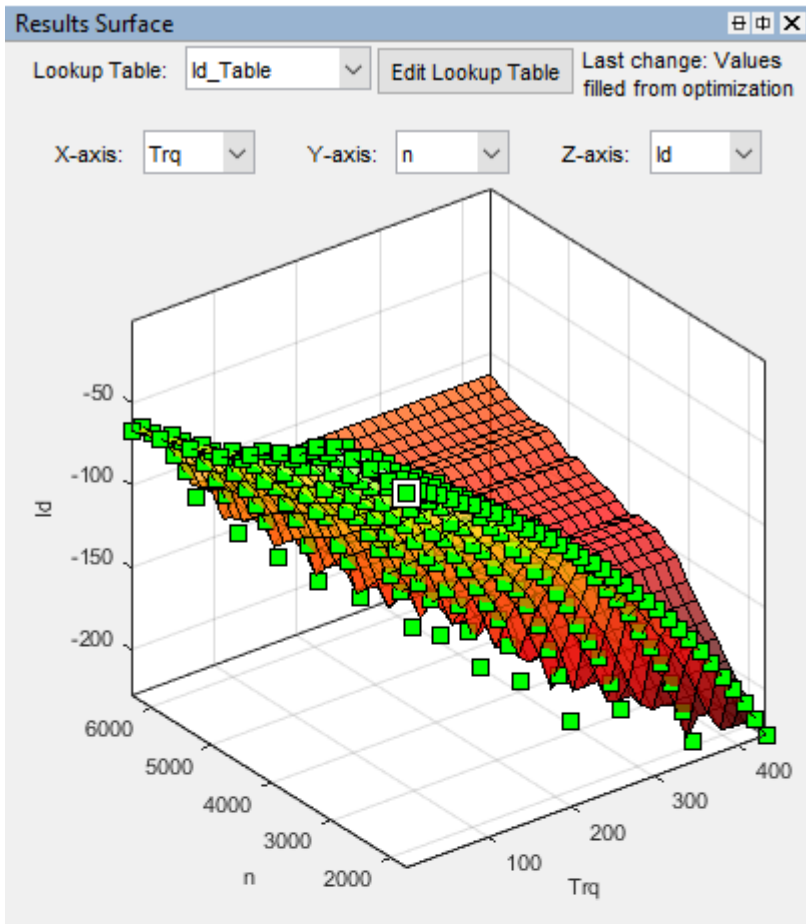
Fill Lookup Tables

1   In the CAGE Browser, select **Fill Lookup Tables**.
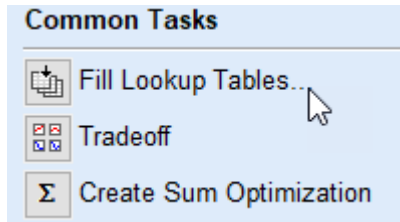
2   Use the Lookup Table Filling from Optimization Results Wizard to fill the `Id_Table` and `Iq_Table` tables.
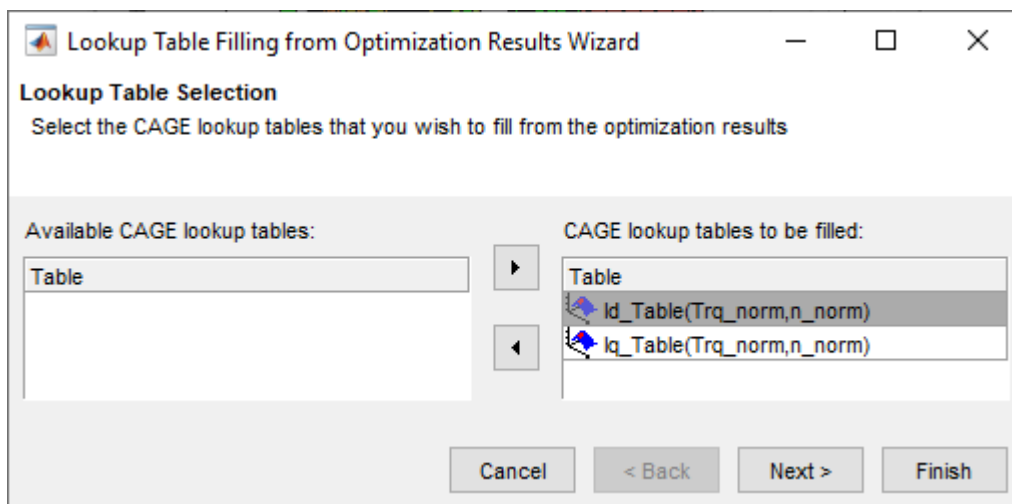
- For the `Id_Table`, fill with `Id`.
- For the `Iq_Table`, fill with `Iq`.
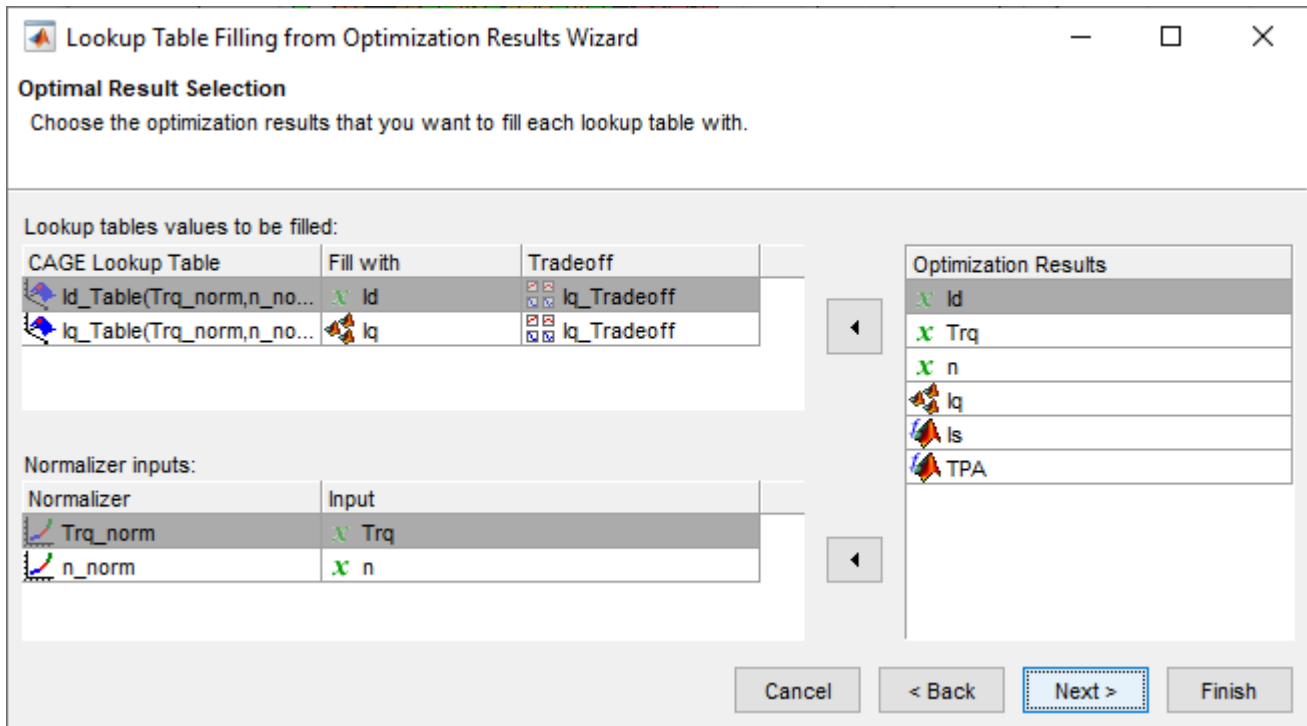
Click **Next**. For the **Fill Method**, select `Clip Fill (column-based)`.
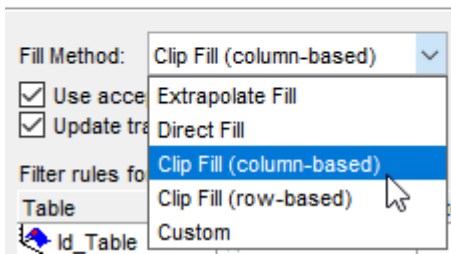
Click **Finish**.

3 Review results for `Iq_Table`. The results are similar to these.
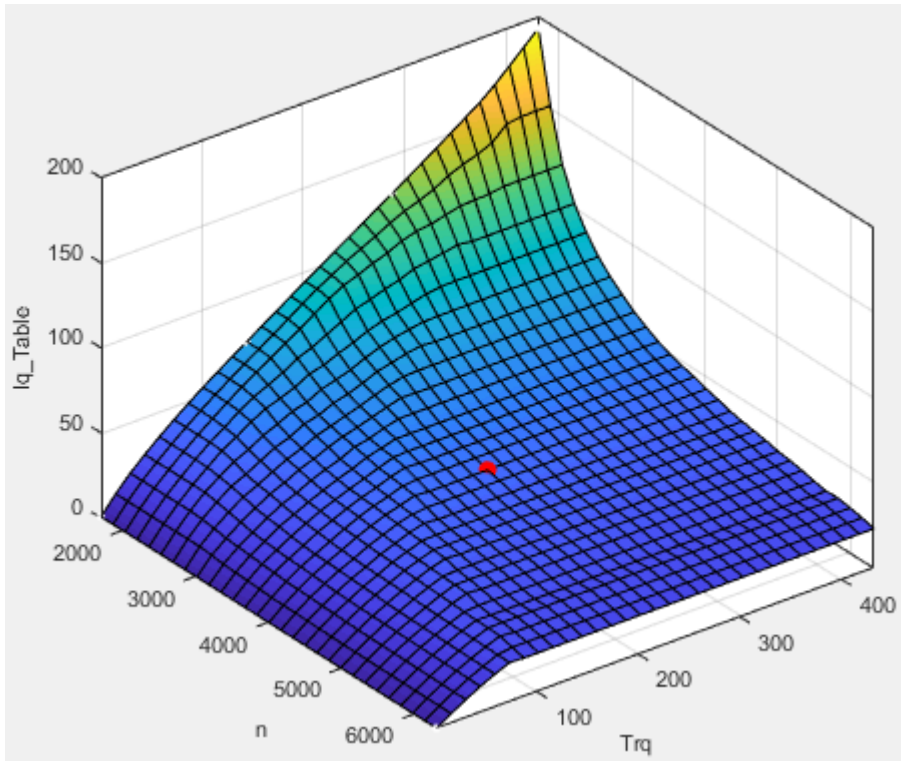
4 Review results for `Id_Table`. The results are similar to these.
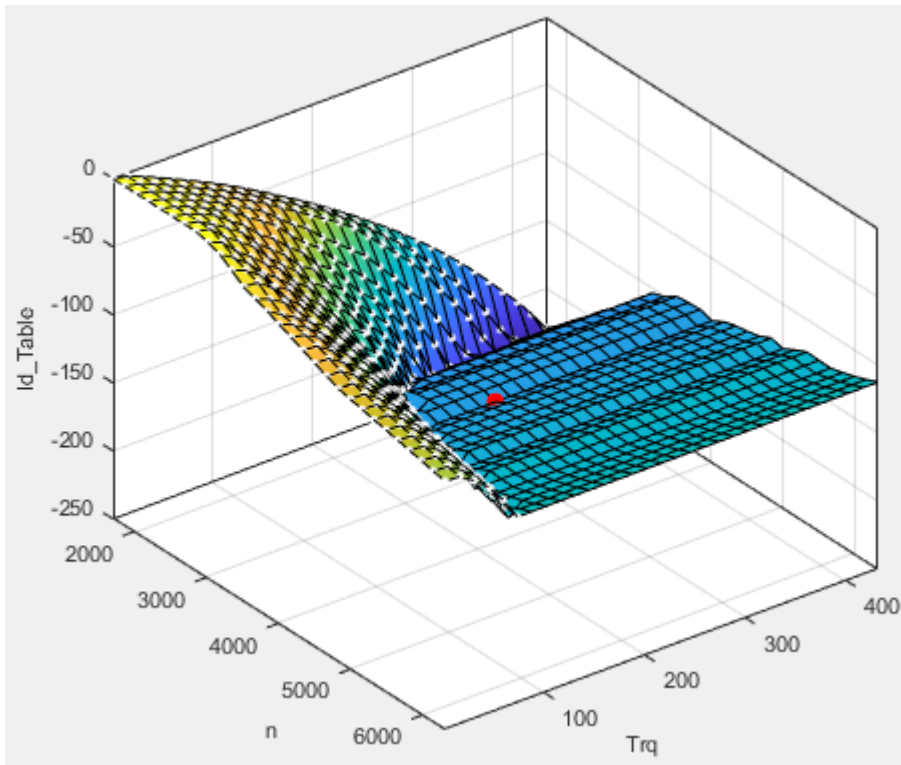
5 Select **File** > **Save Project**. Save `gs_example.cag` to `work` folder.

Export Results

1 Select **File > Export > Calibration**.

2 Use Export Calibration Data to select the items to export and format. For example, export the Id and Iq tables and breakpoints to MATLAB file `gs_example.m`.
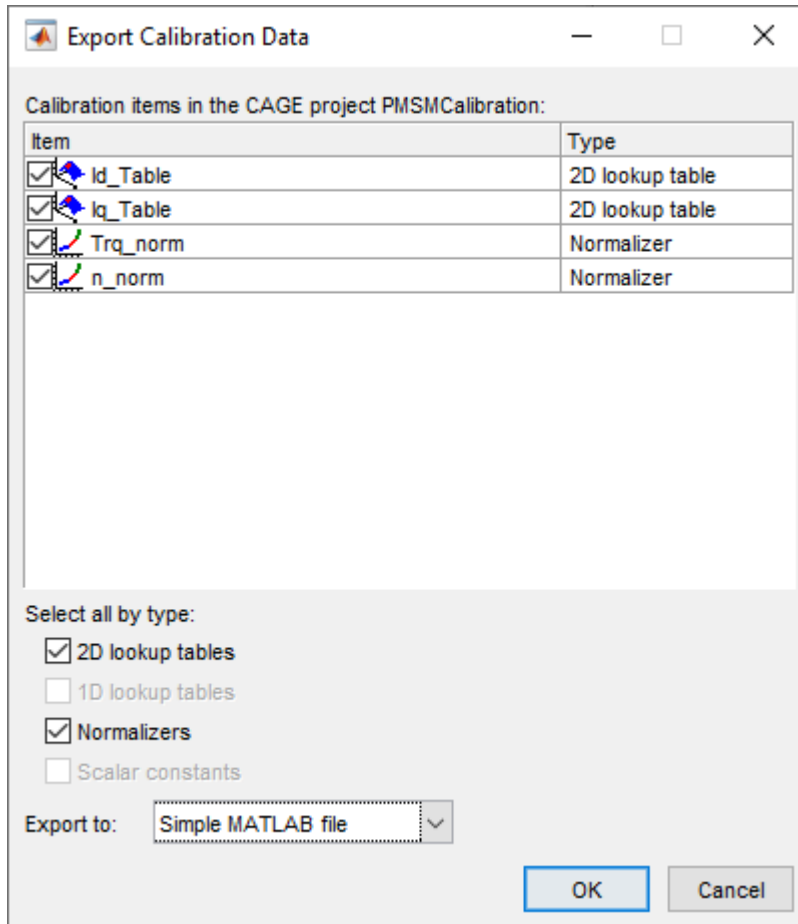
References

[1] Hu, Dakai, Yazan Alsmadi, and Longya Xu. "High fidelity nonlinear IPM modeling based on measured stator winding flux linkage." *IEEE® Transactions on Industry Applications*, Vol. 51, No. 4, July/August 2015.

[2] Chen, Xiao, Jiabin Wang, Bhaskar Sen, Panagiotis Lasari, Tianfu Sun. "A High-Fidelity and Computationally Efficient Model for Interior Permanent-Magnet Machines Considering the Magnetic Saturation, Spatial Harmonics, and Iron Loss Effect." *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 7, July 2015.

[3] Ottosson, J., M. Alakula. "A compact field weakening controller implementation." *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, July, 2006.

Mapped Engine Lookup Tables

The Model-Based Calibration Toolbox includes projects and templates that you can use to calibrate mapped engine lookup tables. Use the lookup tables in these Powertrain Blockset™ engine blocks:

- Mapped CI Engine — Implements a mapped compression-ignition (CI) engine model
- Mapped SI Engine — Implements a mapped spark-ignition (SI) engine model

To help you get started with the calibration, the CAGE browser and Model Explorer include these templates and examples.

Example	Template	Lookup Tables
"Mapped CI Lookup Tables as Functions of Fuel Mass and Engine Speed" on page 5-34	`CI Mapped Engine-Fuel Input`	• Power • Air • Fuel • Temperature • Efficiency • Hydrocarbon (HC) emissions • Carbon monoxide (CO) emissions • Nitric oxide and nitrogen dioxide (NOx) emissions • Carbon dioxide (CO2) emissions • Particulate matter (PM) emissions
"Mapped CI Lookup Tables as Functions of Engine Torque and Speed" on page 5-43	`CI Mapped Engine-Torque Input`	
"Mapped SI Lookup Tables as Functions of Engine Torque and Speed" on page 5-53	`SI Mapped Engine-Torque Input`	

See Also

Mapped CI Engine | Mapped SI Engine

Mapped CI Lookup Tables as Functions of Fuel Mass and Engine Speed

The Model-Based Calibration Toolbox includes projects and templates that you can use to generate calibrated compression-ignition (CI) lookup tables as a function of fuel mass and engine speed. Use the tables in the Powertrain Blockset Mapped CI Engine block.

Use Test Plan Template to Fit Models

1 In the Model Browser, to open the data, select **Import Data**. Navigate to the spreadsheet that contains the data.

For example, open `matlab\toolbox\mbc\mbctraining\CiEngineData.xlsx`.

The spreadsheet contains firing and motor data collected at different engine torques and speeds.

Firing Data	Description
FuelMassCmd	Commanded fuel mass, in mg
Torque	Engine torque, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s
BSFC	Engine brake-specific fuel consumption (BSFC), in g/kWh
CO2MassFlwRate	Carbon dioxide emission mass flow, in kg/s
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s
ExhTemp	Exhaust temperature, in K
FuelMassFlwRate	Fuel mass flow, in kg/s
HCMassFlwRate	Hydrocarbon emission mass flow, in kg/s
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s
PMMassFlwRate	Particulate matter emission mass flow, in kg/s

Nonfiring motor data is collected at different engine speeds, without fuel consumption.

Nonfiring Data	Description
Torque	Engine torque command, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s

2 In the Select Sheet dialog box, select the data that you want to calibrate. For example, select `Firing Data`.

3 Optionally, use the Data Editor filter the data. After you have filtered the data, close the Data Editor.

4 In the Model Browser, select **Fit Models**. In the Fit Models dialog box, in the Template pane, select the template.

For example, to fit the firing data in the spreadsheet, select `MappedEngine-Fuel`. Do not change the default responses and inputs.
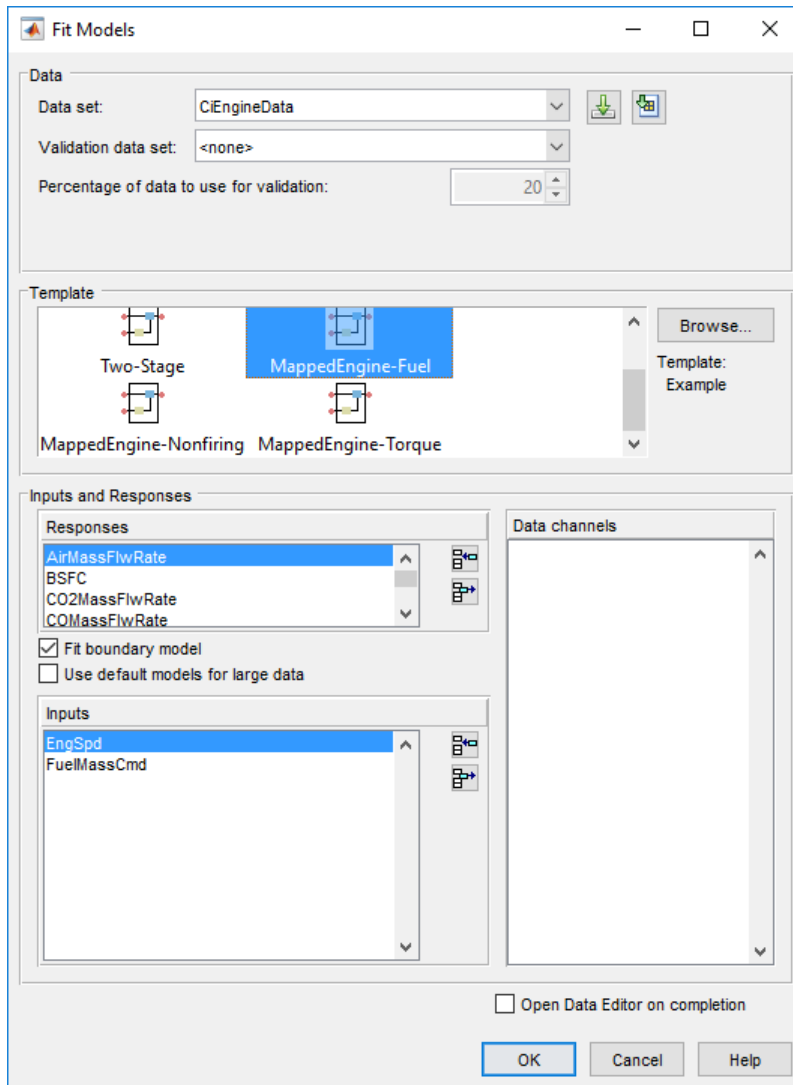
5 Review the model fits.

To review the response models, in the tree, select the top level.
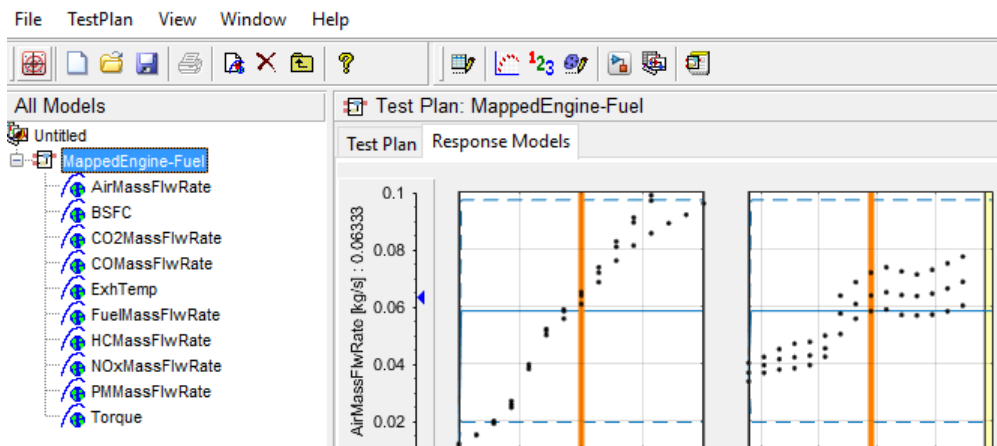
To review the response surfaces, in the tree, select the response.
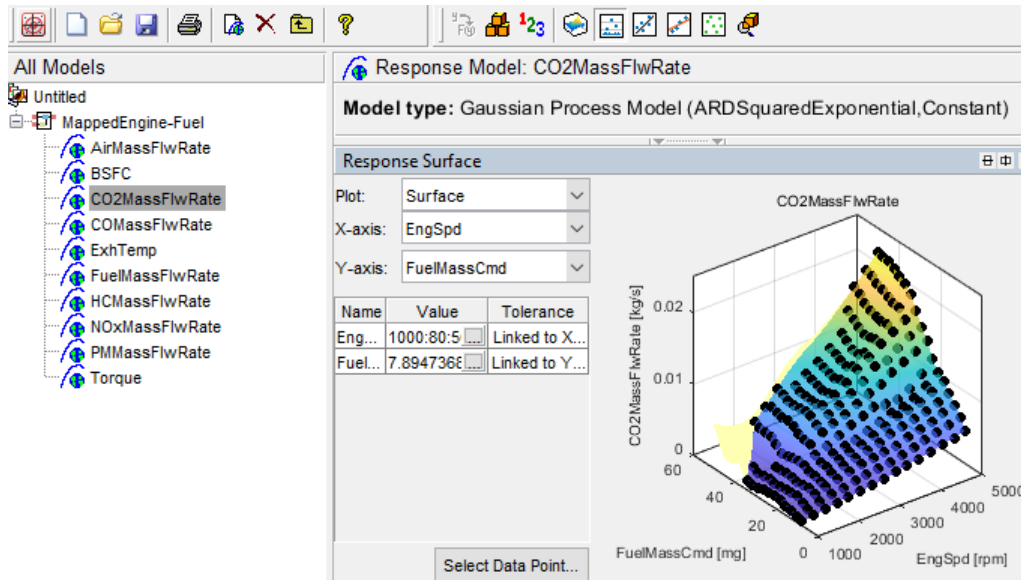
Open CAGE Project

To open the project, in the CAGE **Case Studies** pane:

1 Select `CI Mapped Engine - Fuel Input.`

2 Click **Open Example**.

The project includes these tables.

Name	Description	Table
f_air	Air mass flow, in kg/s	

Name	Description	Table
f_eff	Engine brake-specific fuel consumption (BSFC), in g/kWh	
f_co2	Carbon dioxide emission mass flow, in kg/s	

Name	Description	Table
f_co	Carbon monoxide emission mass flow, in kg/s	
f_texh	Exhaust temperature, in K	

Name	Description	Table
f_fuel	Fuel mass flow, in kg/s	
f_hc	Hydrocarbon emission mass flow, in kg/s	

Name	Description	Table
f_nox	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s	
f_pm	Particulate matter emission mass flow, in kg/s	

Name	Description	Table
`f_brake`	Engine torque command, in Nm	

Use CAGE to Import and Replace Models

Use the project to import and replace existing models with new models.

1 In CAGE, select **File** > **Import** > **Model**. If you do not have a model open, the model browser opens. Select a model.

2 If your current project has two or more test plans, the Import Models dialog box prompts you to merge compatible models. Select **No**.

3 The Import Models dialog box prompts you to `Replace`, `Skip`, or `Create` new models. Select `Replace`.

After you import and replace the existing models, the Import Models wizard opens the Update Tables dialog box. You can use the Update Table dialog box to rerun optimizations and feature fills to update the tables with the new models.

Review and Export Lookup Tables

1 In CAGE, review the calibrated tables.

2 To export the tables, select **File > Export > Calibration > All Items**. Use the **Export to** parameter to specify the format. To export so that you can use the data for the Powertrain Blockset mapped engine blocks, select `Simulink Model Workspace`. The Model-Based Calibration Toolbox saves the mapped engine table and breakpoint data to the model workspace.

See Also
Mapped CI Engine

More About
- "Data Manipulation"
- "Assess High-Level Model Trends"
- "Assess One-Stage Models"
- "Generate Mapped CI Engine from a Spreadsheet" (Powertrain Blockset)

Mapped CI Lookup Tables as Functions of Engine Torque and Speed

The Model-Based Calibration Toolbox includes projects and templates that you can use to generate calibrated compression-ignition (CI) lookup tables as a function of engine torque and speed. Use the tables in the Powertrain Blockset Mapped CI Engine block.

Use Test Plan Template to Fit Models

1. In the Model Browser, to open the data, select **Import Data**. Navigate to the spreadsheet that contains the data.

    For example, open `matlab\toolbox\mbc\mbctraining\CiEngineData.xlsx`.

    The spreadsheet contains firing and motor data collected at different engine torques and speeds.

Firing Data	Description
FuelMassCmd	Commanded fuel mass, in mg
Torque	Engine torque, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s
BSFC	Engine brake-specific fuel consumption (BSFC), in g/kWh
CO2MassFlwRate	Carbon dioxide emission mass flow, in kg/s
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s
ExhTemp	Exhaust temperature, in K
FuelMassFlwRate	Fuel mass flow, in kg/s
HCMassFlwRate	Hydrocarbon emission mass flow, in kg/s
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s
PMMassFlwRate	Particulate matter emission mass flow, in kg/s

Nonfiring motor data is collected at different engine speeds, without fuel consumption.

Nonfiring Data	Description
Torque	Engine torque command, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s

2. In the Select Sheet dialog box, select the data that you want to calibrate. For example, select `Firing Data`.

3. Optionally, use the Data Editor filter the data. After you have filtered the data, close the Data Editor.

4. In the Model Browser, select **Fit Models**. In the Fit Models dialog box, in the Template pane, select the template.

    For example, to fit the firing data in the spreadsheet, select `MappedEngine-Torque`. Do not change the default responses and inputs.

5 Review the model fits.

To review the response models, in the tree, select the top level.

To review the response surfaces, in the tree, select the response.

Open CAGE Project

To open the project, in the CAGE **Case Studies** pane:

1 Select CI Mapped Engine - Torque Input.

2 Click **Open Example**.

The project includes these tables.

Name	Description	Table
f_air	Air mass flow, in kg/s	
f_eff	Engine brake-specific fuel consumption (BSFC), in g/kWh	

Name	Description	Table
f_co2	Carbon dioxide emission mass flow, in kg/s	
f_co	Carbon monoxide emission mass flow, in kg/s	

Name	Description	Table
f_texh	Exhaust temperature, in K	
f_fuel	Fuel mass flow, in kg/s	

Name	Description	Table
f_hc	Hydrocarbon emission mass flow, in kg/s	
f_nox	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s	

Name	Description	Table
f_pm	Particulate matter emission mass flow, in kg/s	
f_brake	Engine torque command, in Nm	

Use CAGE to Import and Replace Models

Use the project to import and replace existing models with new models.

1. In CAGE, select **File > Import > Model**. If you do not have a model open, the model browser opens. Select a model.

2. If your current project has two or more test plans, the Import Models dialog box prompts you to merge compatible models. Select **No**.

3. The Import Models dialog box prompts you to `Replace`, `Skip`, or `Create` new models. Select `Replace`.

After you import and replace the existing models, the Import Models wizard opens the Update Tables dialog box. You can use the Update Table dialog box to rerun optimizations and feature fills to update the tables with the new models.

Review and Export Lookup Tables

1 In CAGE, review the calibrated tables.

2 To export the tables, select **File > Export > Calibration > All Items**. Use the **Export to** parameter to specify the format. To export so that you can use the data for the Powertrain Blockset mapped engine blocks, select `Simulink Model Workspace`. The Model-Based Calibration Toolbox saves the mapped engine table and breakpoint data to the model workspace.

See Also

Mapped CI Engine

More About

- "Data Manipulation"
- "Assess High-Level Model Trends"
- "Assess One-Stage Models"
- "Generate Mapped CI Engine from a Spreadsheet" (Powertrain Blockset)

Mapped SI Lookup Tables as Functions of Engine Torque and Speed

The Model-Based Calibration Toolbox includes a project and template that you can use to generate calibrated spark-ignition (SI) lookup tables as a function of engine torque and speed. Use the tables in the Powertrain Blockset Mapped SI Engine block.

Use Test Plan Template to Fit Models

1 In the Model Browser, to open the data, select **Import Data**. Navigate to the spreadsheet that contains the data.

For example, open `matlab\toolbox\mbc\mbctraining\SiEngineData.xlsx`.

The spreadsheet contains firing and motor data collected at different engine torques and speeds.

Firing Data	Description
Torque	Engine torque, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s.
BSFC	Engine brake-specific fuel consumption (BSFC), in g/kWh
CO2MassFlwRate	Carbon dioxide emission mass flow, in kg/s
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s
ExhTemp	Exhaust temperature, in K
FuelMassFlwRate	Fuel mass flow, in kg/s
HCMassFlwRate	Hydrocarbon emission mass flow, in kg/s
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s
PMMassFlwRate	Particulate matter emission mass flow, in kg/s

Nonfiring motor data is collected at different engine speeds, without fuel consumption.

| Nonfiring Data	Description
Torque	Engine torque command, in Nm
EngSpd	Engine speed, in rpm
AirMassFlwRate	Air mass flow, in kg/s

2 In the Select Sheet dialog box, select the data that you want to calibrate. For example, select `Firing Data`.

3 Optionally, use the Data Editor filter the data. After you have filtered the data, close the Data Editor.

4 In the Model Browser, select **Fit Models**. In the Fit Models dialog box, in the Template pane, select the template.

For example, to fit the firing data in the spreadsheet, select `MappedEngine-Torque`. Do not change the default responses and inputs.

5 Review the model fits.

To review the response models, in the tree, select the top level.

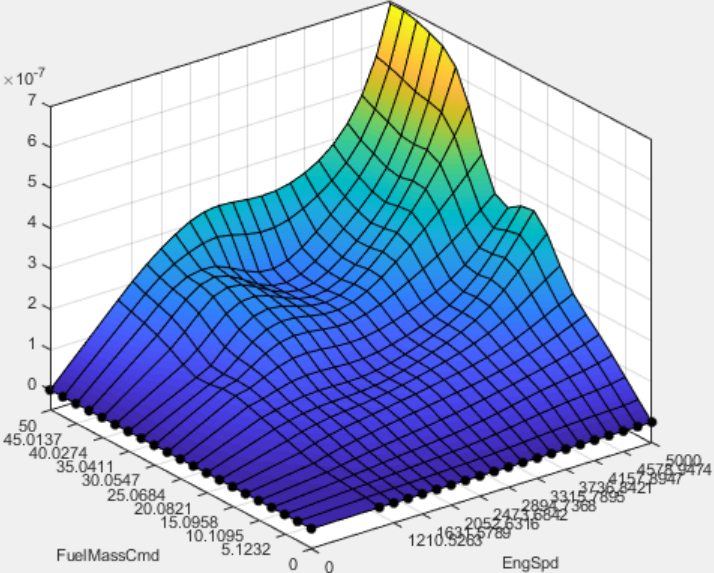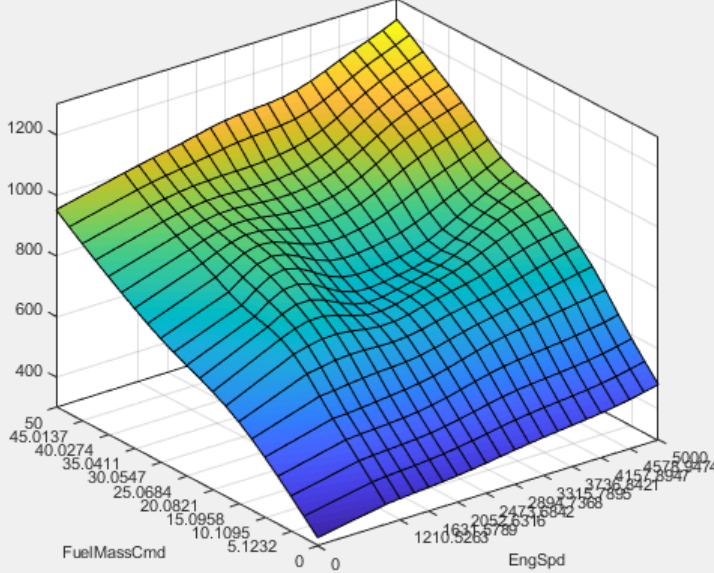To review the response surfaces, in the tree, select the response.

Open CAGE Project

To open the project, in the CAGE **Case Studies** pane:

1   Select `SI Mapped Engine - Torque Input`.

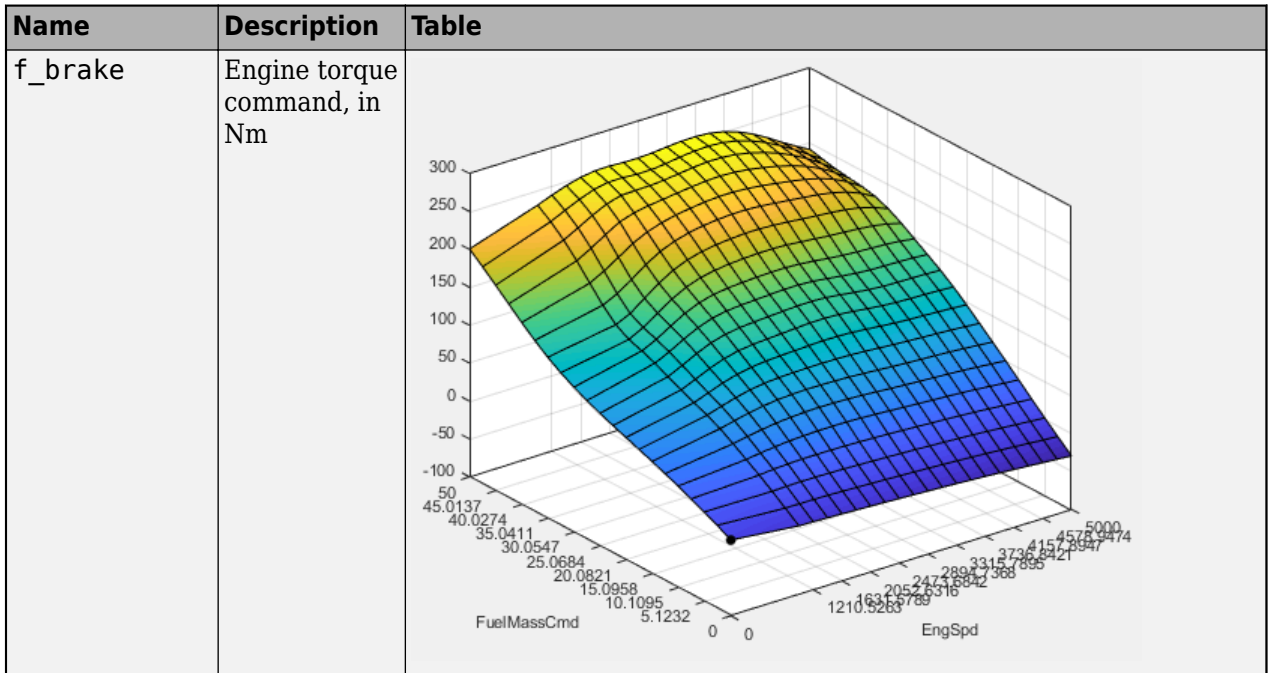**2**   Click **Open Example**.

The project includes these tables.

Name	Description	Table
f_air	Air mass flow, in kg/s	

Name	Description	Table
f_eff	Engine brake-specific fuel consumption (BSFC), in g/kWh	
f_co2	Carbon dioxide emission mass flow, in kg/s	

Name	Description	Table
f_co	Carbon monoxide emission mass flow, in kg/s	
f_texh	Exhaust temperature, in K	

Name	Description	Table
f_fuel	Fuel mass flow, in kg/s	
f_hc	Hydrocarbon emission mass flow, in kg/s	

Name	Description	Table
f_nox	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s	
f_pm	Particulate matter emission mass flow, in kg/s	

Name	Description	Table
f_brake	Engine torque command, in Nm	

Use CAGE to Import and Replace Models

Use the project to import and replace existing models with new models.

1   In CAGE, select **File > Import > Model**. If you do not have a model open, the model browser opens. Select a model.

2   If your current project has two or more test plans, the Import Models dialog box prompts you to merge compatible models. Select **No**.

3   The Import Models dialog box prompts you to `Replace`, `Skip`, or `Create` new models. Select `Replace`.
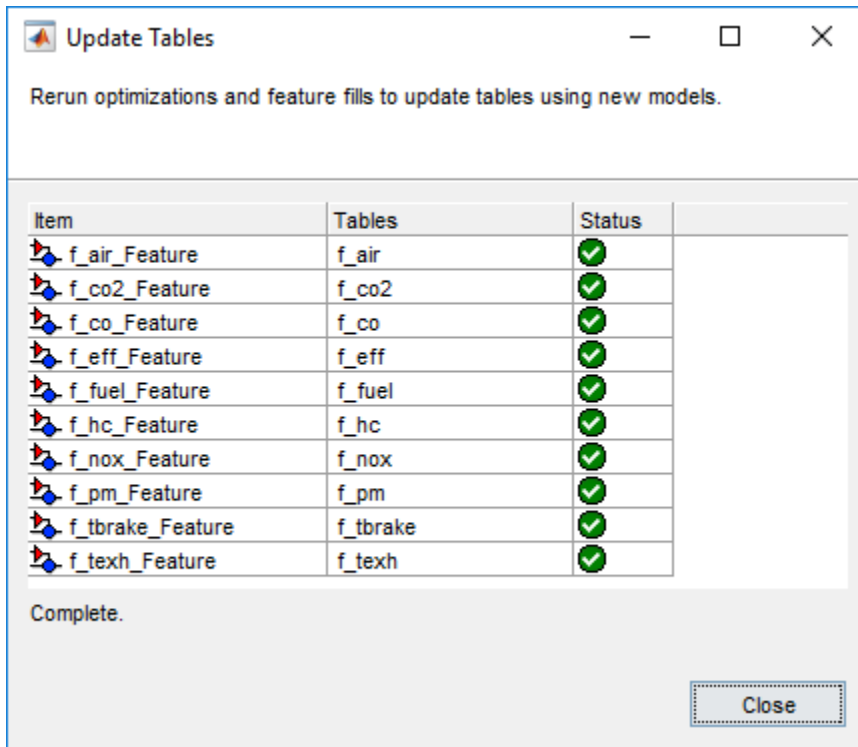

After you import and replace the existing models, the Import Models wizard opens the Update Tables dialog box. You can use the Update Table dialog box to rerun optimizations and feature fills to update the tables with the new models.

Item	Tables	Status
f_air_Feature	f_air	✓
f_co2_Feature	f_co2	✓
f_co_Feature	f_co	✓
f_eff_Feature	f_eff	✓
f_fuel_Feature	f_fuel	✓
f_hc_Feature	f_hc	✓
f_nox_Feature	f_nox	✓
f_pm_Feature	f_pm	✓
f_tbrake_Feature	f_tbrake	✓
f_texh_Feature	f_texh	✓

Review and Export Lookup Tables

1 In CAGE, review the calibrated tables.

2 To export the tables, select **File > Export > Calibration > All Items**. Use the **Export to** parameter to specify the format. To export so that you can use the data for the Powertrain Blockset mapped engine blocks, select `Simulink Model Workspace`. The Model-Based Calibration Toolbox saves the mapped engine table and breakpoint data to the model workspace.

See Also
Mapped SI Engine

More About
- "Data Manipulation"
- "Assess High-Level Model Trends"
- "Assess One-Stage Models"
- "Generate Mapped CI Engine from a Spreadsheet" (Powertrain Blockset)

Design of Experiment

Design of Experiments

Why Use Design of Experiment?

With today's ever-increasing complexity of models, design of experiment has become an essential part of the modeling process. The Design Editor within the Model-Based Calibration Toolbox product is crucial for the efficient collection of engine data. Dyno-cell time is expensive, and the savings in time and money can be considerable when a careful experimental design takes only the most useful data. Dramatically reducing test time is growing more and more important as the number of controllable variables in more complex engines is growing. With increasing engine complexity, the test time increases exponentially.

The traditional method of collecting large quantities of data by holding each factor constant in turn until all possibilities have been tested is an approach that quickly becomes impossible as the number of factors increases. A full factorial design (that is, testing for torque at every combination of speed, load, air/fuel ratio, and exhaust gas recirculation on a direct injection gasoline engine with stratified combustion capability) is not feasible for newer engines. Simple calculation estimates that, for recently developed engines, to calibrate in the traditional way would take 99 years!

With a five-factor experiment including a multiknot spline dimension and 20 levels in each factor, the number of points in a full factorial design quickly becomes thousands, making the experiment prohibitively expensive to run. The Design Editor solves this problem by choosing a set of experimental points that allow estimation of the model with the maximum confidence using just a fraction of the number of experimental runs; for the preceding example just 100 optimally chosen runs is more than enough to fit the model. Obviously, this approach can be advantageous for any complex experimental design, not just engine research.

The Design Editor offers a systematic, rigorous approach to the data collection stage. When you plan a sequence of tests to be run on an example engine, you can base your design on engineering expertise and existing physical and analytical models. During testing, you can compare your design with the latest data and optimize the remaining tests to get maximum benefit.

The Design Editor provides prebuilt standard designs to allow a user with a minimal knowledge of the subject to quickly create experiments. You can apply engineering knowledge to define variable ranges and apply constraints to exclude impractical points. You can increase modeling sophistication by altering optimality criteria, forcing or removing specific design points, and optimally augmenting existing designs with additional points.

Design Styles

The Design Editor provides the interface for building experimental designs. You can make three different styles of design: classical, space-filling, and optimal.

Space-filling designs are better when there is low system knowledge. In cases where you are not sure what type of model is appropriate, and the constraints are uncertain, space-filling designs collect data in such as a way as to maximize coverage of the factors' ranges as quickly as possible.

Optimal designs are best for cases with high system knowledge, where previous studies have given confidence on the best type of model to be fitted, and the constraints of the system are well understood.

Classical designs (including full factorial) are very well researched and are suitable for simple regions (hypercube or sphere). Engines have complex constraints and models (high-order polynomials and splines).

You can augment any design by adding points. Working in this way allows new experiments to enhance the original, rather than simply being a second attempt to gain the necessary knowledge.

Create Examples Using the Design Editor

Follow these steps to construct space-filling, optimal, and classical designs.

1 Choose a model to design an experiment for, enter the Design Editor, and construct an space-filling design. Then construct and apply two different constraints to this design and view the results. Often you would design constraints before constructing a design, but for the purposes of this tutorial you make constraints last so you can view the effects on your design.

2 Create an optimal design.

3 After you create designs, use the displays and tools to examine the properties of the design, save the design, and make changes.

4 Create a classical design, and use the Prediction Error Variance Viewer to compare it with the optimal design. You can also use the Design Evaluation tool to view all details of any design.

See Also

Related Examples

- "Set Up a Model and Create a Design" on page 6-4
- "Create a Constrained Space-Filling Design" on page 6-5
- "View Design Displays" on page 6-8
- "Apply Constraints" on page 6-5
- "Use the Prediction Error Variance Viewer" on page 6-9

More About

- "Design of Experiments"

Set Up a Model and Create a Design

Set Up Model Inputs

You must first specify model inputs for which to design an experiment.

1   Start the Model Browser part of the toolbox by typing `mbcmodel` at the MATLAB command line.

2   From the startup project node view, in the **Common Tasks** pane, click **Design experiment**.

   The **New Test Plan** dialog box appears.

3
   Click the `Two-Stage` test plan icon in the Template pane. A two-stage model fits a model to data with a hierarchical structure.

4   There is only one input to the global model by default. Click the up arrow button to increase the **Number of factors** setting to 3.

5   Change the symbols of the three input factors to `N`, `L`, and `A`.

6   Click **OK** to dismiss the dialog box.

   The Model Browser displays the test plan diagram with your specified model inputs.

Open the Design Editor

In the test plan view, in the **Common Tasks** pane, click **Design experiment**.

The Design Editor window appears.

Alternatively, to open the Design Editor, you can also use either of the following methods:

- Right-click the global model in the diagram and choose **Design Experiment**, as shown.
- You can also access the Design Editor by selecting the menu item **TestPlan > Design Experiment**.

Create a New Design

1
   Click the ![new button] button in the toolbar or select **File > New**. A new node called appears.

2   The new node is automatically selected. An empty Design Table appears (or any view you last used in the Design Editor) because you have not yet chosen a design. For this example you create an optimal design for the default global model.

   You can change the model for which you are designing an experiment from within the Design Editor window by selecting **Edit > Model**.

3   Rename the new node `Space Fill` (you can edit the names by clicking again on a node when it is already selected, or by pressing **F2**, as when selecting to rename in Windows Explorer).

Create a Constrained Space-Filling Design

Space-filling designs should be used when there is little or no information about the underlying effects of factors on responses. For example, they are most useful when you are faced with a new type of engine, with little knowledge of the operating envelope. These designs do not assume a particular model form. The aim is to spread the points as evenly as possible around the operating space. These designs literally fill out the *n*-dimensional space with points that are in some way regularly spaced. These designs can be especially useful with nonparametric models such as radial basis functions (a type of neural network).

1 In the Design Editor, with the new design selected, select **Design > Space Filling > Design Browser**, or click the Space Filling Design button on the toolbar.

2 Leave the default `Sobol Sequence` in the **Design type** list, and the default **Number of points**.

3 Use the Preview tabs to view 2-D and 3-D previews.

4 Click **OK** to calculate the space-filling design and return to the main Design Editor.

Apply Constraints

In many cases, designs might not coincide with the operating region of the system to be tested. For example, a conventional stoichiometric AFR automobile engine normally does not operate with high exhaust gas recirculation (EGR) in a region of low speed (n) and low load (l). You cannot run 15% EGR at 800 RPM idle with a homogeneous combustion process. There is no point selecting design points in impractical regions, so you can constrain the candidate set for test point generation. Only optimal designs have candidate sets of points; classical designs have set points, and space-filling designs distribute points between the coded values of (1, -1).

You would usually set up constraints *before* making designs. Applying constraints to classical and space-filling designs simply removes points outside the constraint. Constraining the candidate set for optimal designs ensures that design points are optimally chosen within the area of interest only.

Designs can have any number of geometric constraints placed upon them. Each constraint can be one of four types: an ellipsoid, a hyperplane, a 1-D lookup table, or a 2-D lookup table.

To add a constraint to your currently selected design:

1 Select **Edit > Constraints** from the Design Editor menus.

2 The Constraints Manager dialog appears. Click **Add**.

The Constraint Editor dialog with available constraints appears. Leave the default `1D Table` in the **Constraint type** list.

3 You can select the appropriate factors to use. For this example, choose speed (N) and air/fuel ratio (A) for the X and Y factors.

4 Move the large dots (click and drag them) to define a boundary. The Constraint Editor should look something like the following.
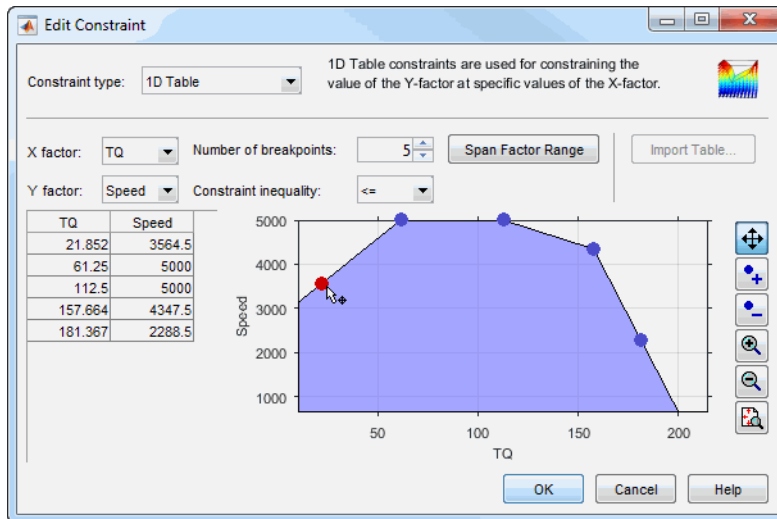
5   Click **OK**.

Your new constraint appears in the Constraint Manager list box. Click **OK** to return to the Design Editor. A dialog appears because there are points in the design that fall outside your newly constrained candidate set.

6   Click **Continue** to remove the points outside the constraint. Note that fixed points are not deleted by this process.
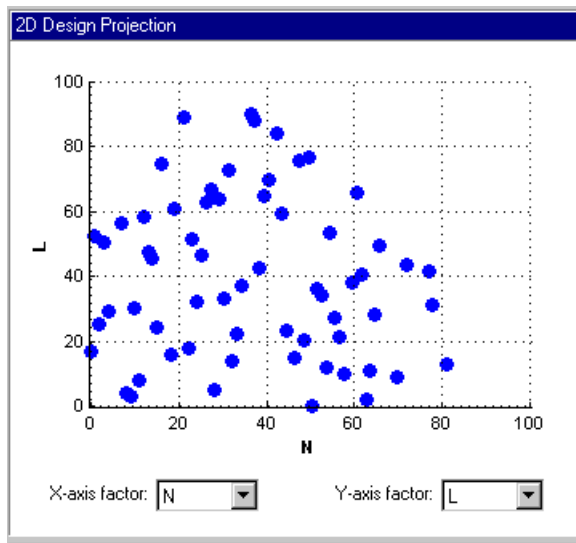
Plot the 2-D projection of the hypercube, and observe the effects of the new constraint on the shape of the design, as shown in the preceding example.

7   Right-click the display pane to reach the context menu, and select **Current View > 3D Constraints**.These views are intended to give some idea of the region of space that is currently available within the constraint boundaries.

8   Return to the Constraint Editor, choose **Edit > Constraint**, and click **Add** in the Constraint Manager.

9   Add an ellipsoid constraint. Choose `Ellipsoid` from the drop-down menu of constraint types.
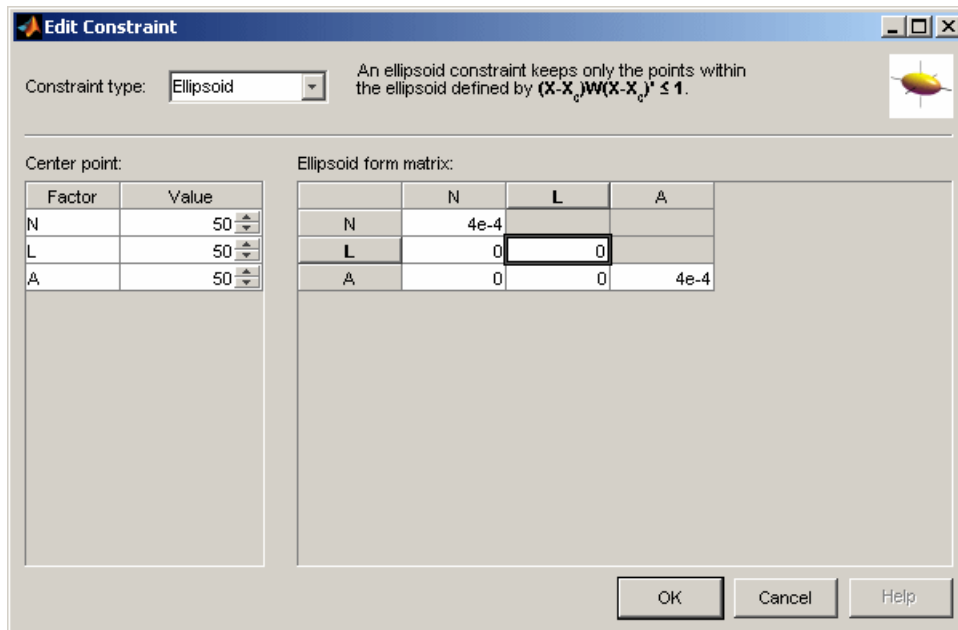
Enter 0 as the value for the **L** diagonal in the table, as shown. This will leave L unconstrained (a cylinder). The default ellipsoid constraint is a sphere. To constrain a factor, if you want a radius of r in a factor, enter 1/(r^2). For this example, leave the other values at the defaults. Click **OK** to apply the constraint.

10 Click **OK**, click **OK** again in the Constraint Manager, and click **Continue** to remove design points outside the new candidate set (or **Replace** if you are constraining an optimal design). Examine the new constraint 3-D constraints plot.

Both constraints are applied to this design.

See Also

Related Examples
- "Design of Experiments" on page 6-2
- "View Design Displays" on page 6-8

More About
- "Design of Experiments"

View Design Displays

In the Design Editor, after you create a design, it shows the **Design Table** view of the design or other views if you previously viewed designs. In the context menu, available by right-clicking on the title bar, you can change the view of the design to 1-D, 2-D, 3-D, 4-D, and pairwise design projections, 2-D and 3-D constraint views, and the table view (also under the **View** menu). This menu also allows you to split the display either horizontally or vertically so that you simultaneously have two different views on the current design. You can also use the toolbar buttons to do this. The split can be merged again. After splitting, each view has the same functionality; that is, you can continue to split views until you have as many as you want. When you click a view, its title bar becomes blue to show it is the active view.

The currently available designs are displayed on the left in a tree structure.

Display Options

The Design Editor can display multiple design views at once, so while working on a design you can keep a table of design points open in one corner of the window, a 3-D projection of the constraints below it and a 2-D or 3-D plot of the current design points as the main plot.

The current view and options for the current view are available either through the context menu or the **View** menu on the Design Editor window.

1 Change the main display to 3-D Projection view.
2 You can rotate the projection with click-drag mouse movement. View your design in several projections (singly, or simultaneously by dividing the pane) by using the right-click context menu in the display pane.

See Also

Related Examples

- "Design of Experiments" on page 6-2
- "Create a Constrained Space-Filling Design" on page 6-5
- "Use the Prediction Error Variance Viewer" on page 6-9

More About

- "Design of Experiments"

Use the Prediction Error Variance Viewer

Introducing the Prediction Error Variance Viewer

A useful measure of the quality of a design is its prediction error variance (PEV). The PEV hypersurface is an indicator of how capable the design is in estimating the response in the underlying model. A bad design is either not able to fit the chosen model or is very poor at predicting the response. The Prediction Error Variance Viewer is only available for linear models. The Prediction Error Variance Viewer is not available when designs are rank deficient; that is, they do not contain enough points to fit the model. Optimal designs attempt to minimize the average PEV over the design region.

With an optimal design selected, select **Tools > Prediction Error Variance Viewer**.
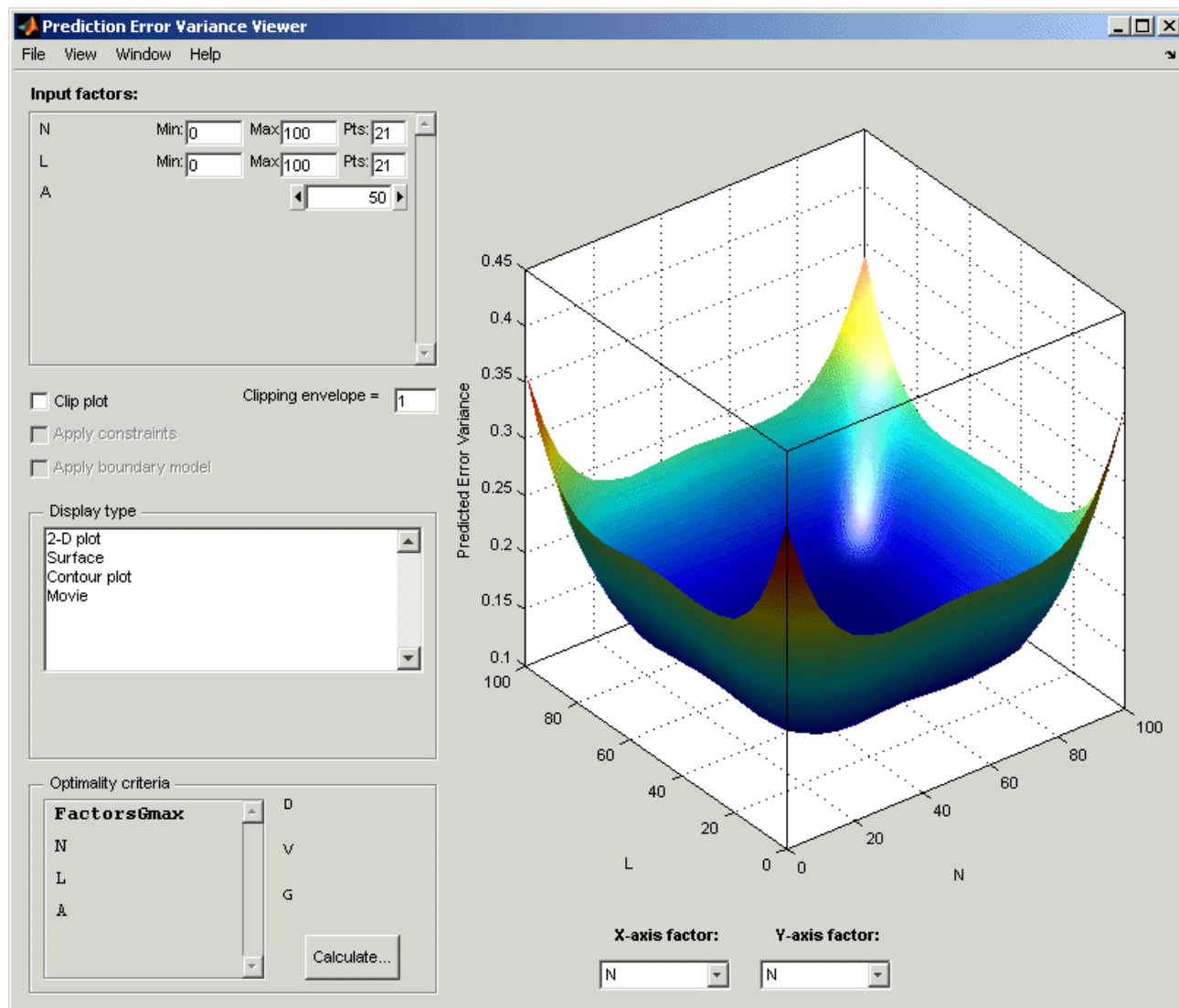
The default view is a 3-D plot of the PEV surface.

This shows where the response predictions are best. This example optimal design predicts well in the center and the middle of the faces (one factor high and the other midrange), but in the corners the design has the highest error. Look at the scale to see how much difference there is between the areas of higher and lower error. For the best predictive power, you want low PEV (close to zero).

You can examine PEV for designs and models. The two are related in this way:

Accuracy of model predictions (model PEV)=Design PEV * MSE (Mean Square Error in measurements).

You can think of the design PEV as multiplying the errors in the data. The smaller the PEV, the greater the accuracy of your final model.

Try the other display options.

- The **View** menu has many options to change the look of the plots.
- You can change the factors displayed in the 2-D and 3-D plots. The pop-up menus below the plot select the factors, while the unselected factors are held constant. You can change the values of the unselected factors using the buttons and edit boxes in the **Input factors** list, top left.
- The `Movie` option shows a sequence of surface plots as a third input factor's value is changed. You can change the factors, replay, and change the frame rate.
- You can change the number, position, and color of the contours on the contour plot with the **Contours** button, as shown.
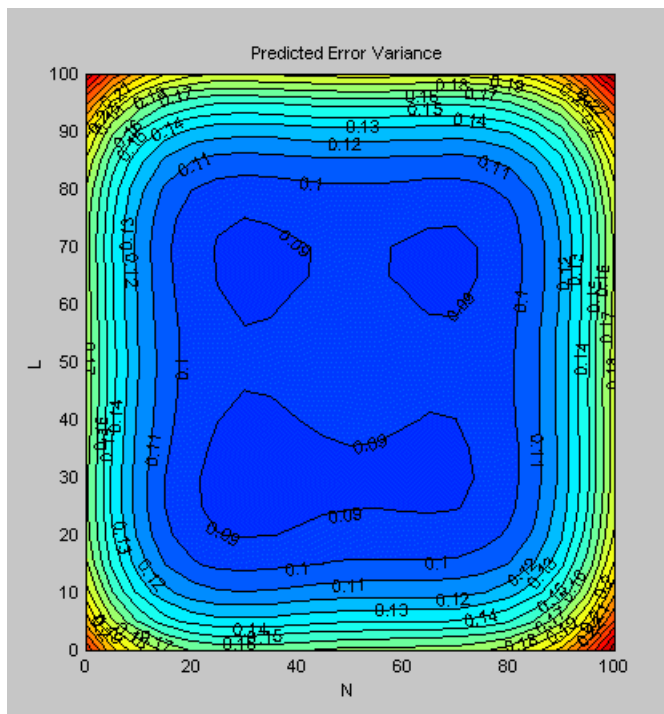
Add Points Optimally

You can further optimize the optimal design by returning to the Optimal Design dialog, where you can delete or add points optimally or at random. The most efficient way is to delete points *optimally* and

add new points *randomly* — these are the default algorithm settings. Only the existing points need to be searched for the most optimal ones to delete (the least useful), but the entire candidate set has to be searched for points to add optimally.

To strengthen the current optimal design:

1 Return to the Design Editor window.

2 Click the Optimal Design button in the toolbar again to reenter the dialog, and add 60 more points. Keep the existing points (which is the default).

3 Click **OK** and watch the optimization progress, then click **Accept** when the number of iterations without improvement starts increasing.

4 View the improvements to the design in the main displays.

5 Once again select **Tools** > **Prediction Error Variance Viewer** and review the plots of prediction error variance and the new values of optimality criteria in the optimality frame (bottom left). The shape of the PEV projection might not change dramatically, but note the changes in the scales as the design improves. The values of D, V, and G optimality criteria will also change (you have to click **Calculate** to see the values).

To see more dramatic changes to the design, return to the Design Editor window (no need to close the Prediction Error Variance Viewer).

1 Split the display so you can see a 3-D projection at the same time as a Table view.

2 You can sort the points to make it easier to select points in one corner. For example, to pick points where N is 100 and L is 0,

    a Select **Edit** > **Sort Points**.

    b Choose to sort by N only (reduce the number of sort variables to one) and click **OK**.

3 Choose **Edit** > **Delete Point**.

4 Using the Table and 3-D views as a guide, in the Delete Points dialog, pick six points to remove along one corner. Add the relevant point numbers to the delete list by clicking the add (>) button.

5 Click **OK** to remove the points. See the changes in the main design displays and look at the new Surface plot in the Prediction Error Variance Viewer (see the example following).
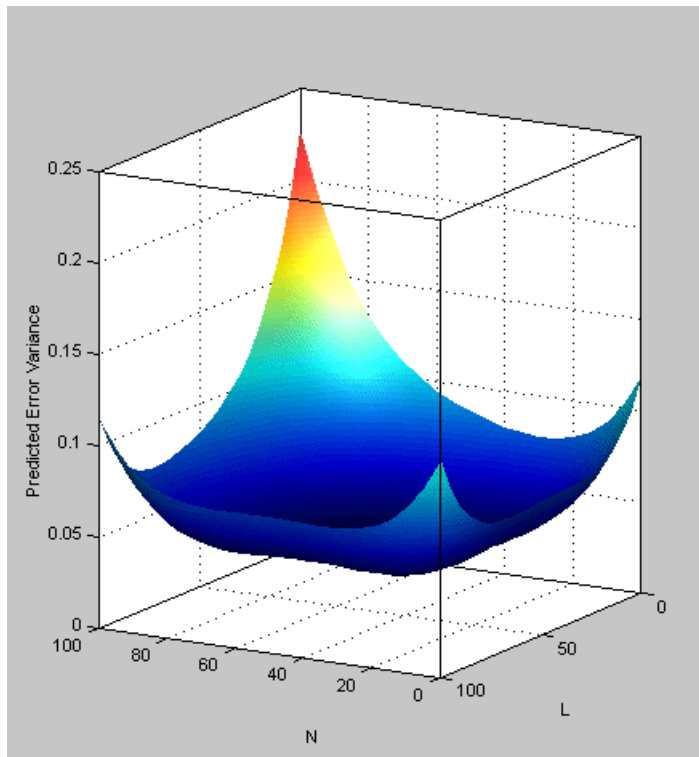
See Also

Related Examples

- "Design of Experiments" on page 6-2
- "Create a Constrained Space-Filling Design" on page 6-5

More About

- "Design of Experiments"

Data Editor for Modeling

Manipulate Data for Modeling

For empirical engine modeling in the Model Browser, first load, process, and select data for modeling. This tutorial shows you how to use the Data Editor for loading data, creating variables, and creating constraints for that data.

You can load data from files (Microsoft® Excel® files, MATLAB files, text files) and from the MATLAB workspace. You can merge data in any of these forms with previously loaded data sets to produce a new data set. Test plans can use only one data set, so the merging function allows you to combine records and variables from different files in one model.

You can define new variables, apply filters to remove unwanted data, and apply test notes to filtered tests. You can store and retrieve these user-defined variables and filters for any data set, and you can store plot settings. You can change and add records and apply test groupings, and you can match data to designs. You can also write your own data loading functions.

To get started, follow these workflow steps.

Workflow Steps	Description
"View and Edit the Data" on page 7-2	Use the Data Editor displays to investigate your data.
"Create New Variables and Filters" on page 7-5	Define your own new variables and filters to remove unwanted data.
"Store and Import Variables, Filters, and Plot Preferences" on page 7-6	Store plot preferences, user-defined variables, filters, and test notes.
"Define Test Groupings" on page 7-6	Use the Define Test Groupings dialog box to group your data.
"Match Data to Experimental Designs" on page 7-8	Use an example project to match experimental data to designs.

View and Edit the Data

Viewing Data

You can split the views to display several plots at once. Use the right-click context menus, the toolbar buttons, or the **View** menu to split views. You can choose 2-D plots, 3-D plots, multiple data plots, data tables, and tabs showing summary, statistics, variables, filters, test filters, and test notes. You can use test notes to investigate problem data and decide whether to remove some points before modeling.

Using Notes to Sort Data for Plotting

1   Right-click a view and select **Current View > Multiple Data Plot**.
2   Right-click the new view and select **Add Plot**.

    The Plot Variables Setup dialog box appears.
3   Select `spark` and click to add to the X Variable box, then select `tq` and click to add to the Y Variable box. Click **OK** to create the plot.
4   Click in the **Tests** list to select a test to plot (or **Shift**-click, **Ctrl**-click, or click and drag to select multiple tests).

5   Select **Tools > Test Notes**.

6   In The Test Note Editor, enter `mean(tq)<10` in the top edit box to define the tests to be noted, and enter `Low torque` in the Test Note edit box. Leave the note color at the default and click **OK**.

7   Click the Test Notes tab to view your note definition.

8   In the Test Selector pane on the left, observe that all the tests that satisfy the condition `mean(tq)<10` show `Low torque` next to them. Click the column header to sort the tests that meet the note condition to the top or bottom of the list.

9   Now create more views.

- Right-click a view and select **Split View > Data Table**.
- Right-click a view and select **Split View > 3D Plot**.

10   In the Test Selector pane, click particular tests with the `Low torque` note.

Notice that when you select a test here, the same test is plotted in the multiple data plots, the 3D data plot, and highlighted in the data table. You can use the notes in this way to easily identify problem tests and decide whether to remove them.

Removing Outliers and Problem Tests

1   Click a point on the **Multiple Data Plots** view.

The point is outlined in red on the plot, and highlighted in the data table. You can remove points you have selected as outliers by selecting **Tools > Remove Data** (or use the keyboard shortcut **Ctrl+A**). Select **Tools > Restore Data** (or use the keyboard shortcut **Ctrl+Z**) to open a dialog box where you can choose to restore any or all removed points.

You can remove individual points as outliers, or you can remove records or entire tests with filters.

2   For example, after examining all the `Low torque` noted tests, you could decide to filter them out.

a   Select **Tools > Test Filters**.

b   In the Test Filter Editor, enter `mean(tq)>10` to keep all tests where the mean torque is greater than 10, and click **OK**.

c   Click the Test Filters tab and observe the new test filter results show it is successfully applied and the number of records removed.

3   To view removed data in the table view, right-click and select **Allow Editing**. Removed records are red. To view removed data in the 2-D and Multiple Data Plots, select **Properties** and select the box **Show removed data**.

Reordering and Editing Data

To change the display, right-click a 2-D plot and select **Properties**. You can alter grid and plot settings including lines to join the data points.

Reorder X Data in the Plot Properties dialog box can be useful when record order does not produce a sensible line joining the data points. For an illustration of this:

1   Ensure that you are displaying a 2-D plot. You can right-click any plot and select **Current Plot > 2-D Plot**, or use the context menu split commands to add new views.

2 Right-click a 2-D plot and select **Properties** and choose `solid` from the **Data Linestyle** drop-down menu. Click **OK**.
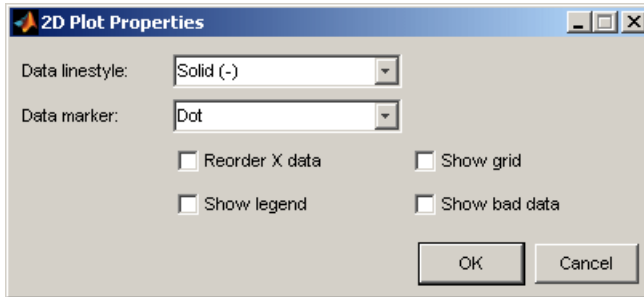
3 Choose `afr` for the *y*-axis.

4 Choose `Load` for the *x*-axis.

5 Select test 590. Use the test controls contained within the 2-D plot. The **Tests** pane on the left applies to other views: tables and 3-D and multiple data plots.
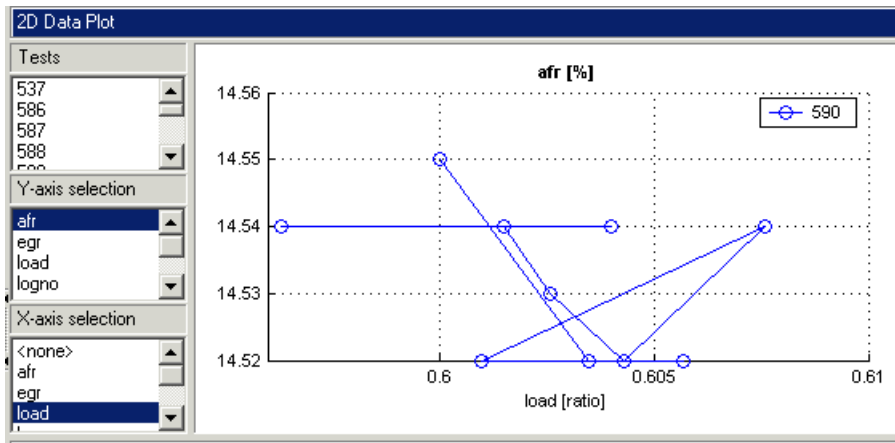
6 Right-click and select **Properties** and choose **Reorder X Data**. Click **OK**.

This command replots the line from left to right instead of in the order of the records, as shown.
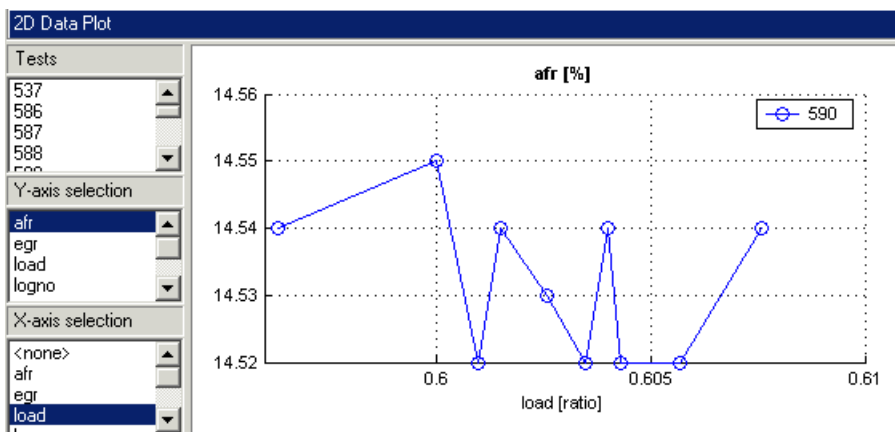
7   Right-click and select **Split Plot** > **Data Table** to split the currently selected view and add a table view. You can select particular test numbers in the **Tests** pane on the left of the Data Editor. You can right-click to select **Allow Editing**, and then you can double-click cells to edit them.

Create New Variables and Filters

Adding New Variables

You can add new variables to the data set.

1   Select **Tools** > **Variables**, or click the toolbar button.

In the Variable Editor, you can define new variables in terms of existing variables. Define the new variable by writing an equation in the edit box at the top.

2   Define a new variable called `POWER` that is defined as the product of two existing variables, `tq` and `n`, by entering `POWER=tq*n`, as seen in the example following. You can also double-click variable names and operators to add them, which can be useful to avoid typing mistakes in variable names, which must be exact including case.

3   Click **OK** to add this variable to the current data set.

4   View the new variable in the Data Editor in the Variables tab at the top. You can also now see `7 + 1 variables` in the summary tab.

Applying a Filter

A filter is a constraint on the data set you can use to exclude some records. You use the Filter Editor to create filters.

1   Choose **Tools** > **Filters**, or click the toolbar button.

In the Filter Editor, define the filter using logical operators on the existing variables.

2   Keep all records with speed (`n`) greater than 1000. Type `n` (or double-click the variable `n`), then type `>1000`.

3   Click **OK** to impose this filter on the current data set.

4   View the new filter in the Data Editor in the Filters tab at the top. Here you can see a list of your user-defined filters and how many records the new filter removes. You can also now see `141/270 records` in the summary tab and a red section in the bar illustrating the records removed by the filter.

Sequence of Variables

You can change the order of user-defined variables in the Variable Editor list using the up and down arrow buttons.

Select **Tools** > **Variables**.

Example:

1   Define two new variables, `New1` and `New2`. You can use the buttons to add or remove a list item to create or delete variables in this view. Click the button to 'Add item' to add a variable, and enter the definitions shown.

Notice that `New2` is defined in terms of `New1`. New variables are added to the data in turn and hence `New1` must appear in the list before `New2`, otherwise `New2` is not well-defined.

2 Change the order by clicking the down arrow in the Variable Editor to produce this erroneous situation. Click **OK** to return to the Data Editor and in the Variables tab you can see the error message that there is a problem with the variable.

3 Use the arrows to order user-defined variables in legitimate sequence.

Deleting and Editing Variables and Filters

You can delete user-defined variables and filters.

Example:

1 To delete the added variable `New1`, select it in the Variables tab and press the **Delete** key.

2 You can also delete variables in the Variable Editor by clicking the Remove Item button.

Similarly, you can delete filters by selecting the unwanted filter in the Filters tab and using the **Delete** key.

Store and Import Variables, Filters, and Plot Preferences

You can store and import plot preferences, user-defined variables, filters, and test notes so they can be applied to other data sets loaded later in the session, and to other sessions.

- Select **Tools > Import Expressions**
- Click the toolbar button

The Data Editor remembers your plot type settings and when reopened displays the same types of views. You can also store your plot layouts to save the details of your Multiple Data Plots.

In the Import Variables, Filters, and Editor Layout dialog box, use the toolbar buttons to import variables, filters, and plot layouts. Import from other data sets in the current project, or from MBC project files, or from files exported from the Data Editor.

To use imported expressions in your current project, select items in the lists and click the toolbar button to apply in the data editor.

To store expressions in a file, in the Data Editor, select **Tools > Export Expressions** and select a file name.

Define Test Groupings

The Define Test Groupings dialog box collects records of the current data object into groups. These groups are referred to as *tests*.

The dialog box is accessed from the Data Editor in either of these ways:

- Using the menu **Tools > Test Groups**
- Using the toolbar button

When you enter the dialog box using the `holliday` data, a plot is displayed as the variable `logno` is automatically selected for grouping tests.

Select another variable to use in defining groups within the data.

1   Select n in the **Variables** list.

2   Click the [icon] button to add the variable (or double-click n).

The variable n appears in the list view on the left. You can now use this variable to define groups in the data. The maximum and minimum values of n are displayed. The **Tolerance** is used to define groups: on reading through the data, when the value of n changes by more than the tolerance, a new group is defined. You change the **Tolerance** by typing directly in the edit box.

You can define additional groups by selecting another variable and choosing a tolerance. Data records are grouped by n or by this additional variable changing outside their tolerances.

3   Clear the box **Group by** for logno. Notice that variables can be plotted without being used to define groups.

4   Add load to the list by selecting it on the right and clicking [icon].

5   Change the load tolerance to 0.01 and watch the test grouping change in the plot.

6   Clear the **Group By** check box for load. Now this variable is plotted without being used to define groups.

The plot shows the scaled values of all variables in the list view (the color of the tolerance text corresponds to the color of data points in the plot). Vertical pink bars show the tests (groups). You can zoom the plot by **Shift**-click-dragging or middle-click-dragging the mouse; zoom out again by double-clicking.

7   Select load in the list view (it becomes highlighted in blue) and remove it from the list by clicking the [icon] button.

8   Double-click to add spark to the list, and clear the **Group By** check box. Select logno as the only grouping variable.

It can be helpful to plot the local model variable (in this case spark) to check that you have the correct test groupings. The plot shows the sweeps of spark values in each test while speed (n) is kept constant. Speed is only changed between tests, so it is a global variable. Try zooming in on the plot to inspect the test groups; double-click to reset.

One-stage data defines one test per record, regardless of any other grouping. This is required if the data is to be used in creating one-stage models.

Sort records before grouping allows you to reorder records in the data set. Otherwise, the groups are defined using the order of records in the original data object.

Show original test groups displays the original test groupings if any were defined.

Test number variable contains a list of all the variables in the current data set. Any of these could be selected to number the tests.

9   make sure that logno is selected for the **Test number variable**.

This changes how the tests are displayed in the rest of the Model Browser. Test number can be a useful variable for identifying individual tests in Model Browser and Data Editor views (instead of 1,2,3...) if the data was taken in numbered tests and you want access to that information during modeling.

If you chose `none` from the **Test number variable** list, the tests would be numbered 1,2,3, and so on, in the order in which the records appear in the data file. With `logno` chosen, you see tests in the Data Editor listed as 586, 587 etc.

Every record in a test must share the same test number to identify it, so when you are using a variable to number tests, the value of that variable is taken in the first record in each test.

Test numbers must be unique, so if any values in the chosen variable are the same, they are assigned new test numbers for the purposes of modeling (this does not change the underlying data, which retains the correct test number or other variable).

10 Click **OK** to accept the test groupings defined and close the dialog box.

In the Data Editor summary tab, view the number of tests.

The number of records shows the number of values left (after filtration) of each variable in this data set, followed by the original number of records. The color coded bars also display the number of records removed as a proportion of the total number. The values are collected into several tests; this number is also displayed. The variables show the original number of variables plus user-defined variables.

Match Data to Experimental Designs

Introducing Matching Data to Designs

You can use an example project to illustrate the process of matching experimental data to designs.

Experimental data is unlikely to be identical to the desired design points. You can use the Design Match view in the Data Editor to compare the actual data collected with your experimental design points. Here you can select data for modeling. If you are interested in collecting more data, you can update your experimental design by matching data to design points to reflect the actual data collected. You can then optimally augment your design (using the Design Editor) to decide which data points it would be most useful to collect, based on the data obtained so far.

You can use an iterative process: make a design, collect some data, match that data with your design points, modify your design accordingly, then collect more data, and so on. You can use this process to optimize your data collection process to obtain the most robust models possible with the minimum amount of data.

1 To see the data matching functions, in the Model Browser, select **File > Open Project** and browse to the file `Data_Matching.mat` in the `matlab\toolbox\mbc\mbctraining` folder.

2 Click the `Spark Sweeps` node in the model tree to change to the test plan view.

Here you can see the two-stage test plan with model types and inputs set up. The global model has an associated experimental design (which you could view in the Design Editor). You are going to use the Data Editor to examine how closely the data collected so far matches to the experimental design.

3 Click the Edit Data button ( ) in the toolbar.

The Data Editor appears.

4 You need a **Design Match** view to examine design and data points. Right-click a view in the Data Editor and select **Current View > Design Match**.

In the Design Match, you can see colored areas containing points. These are "clusters" where the matching algorithm selects closely matching design and data points.

Tolerance values (derived initially from a proportion of the ranges of the variables) are used to determine if any data points lie within tolerance of each design point. Data points that lie within tolerance of any design point are matched to that cluster. Data points that fall inside the tolerance of more than one design point form a single cluster containing all those design and data points. If no data points lie within tolerance of a design point, it remains unmatched and no cluster is plotted.
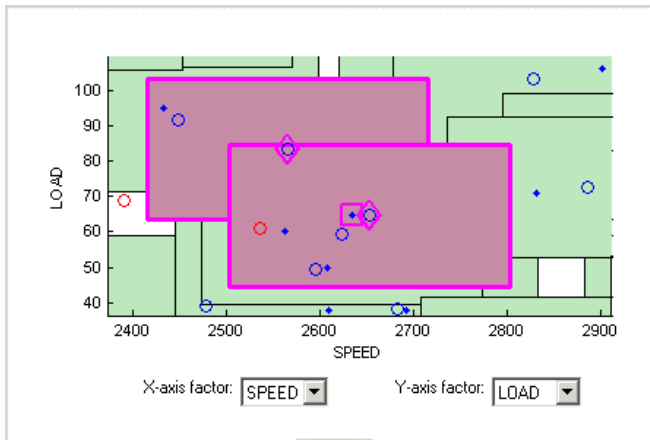
Notice the shape formed by overlapping clusters. The example shown outlined in pink is a single cluster formed where a data point lies within tolerance of two design points.

On this plot, you can see other cleared points that appear to be contained within this cluster. You track points through other factor dimensions using the axis controls to see where points are separated beyond tolerance.

Tolerances and Cluster Information

1   To edit tolerance values, select **Tolerances** in the context menu.

The Tolerance Editor appears. Here you can change the size of clusters in each dimension. Observe that the LOAD tolerance value is 100. This accounts for the elongated shape (in the LOAD dimension) of the clusters in the current plot, because this tolerance value is a high proportion of the total range of this variable.
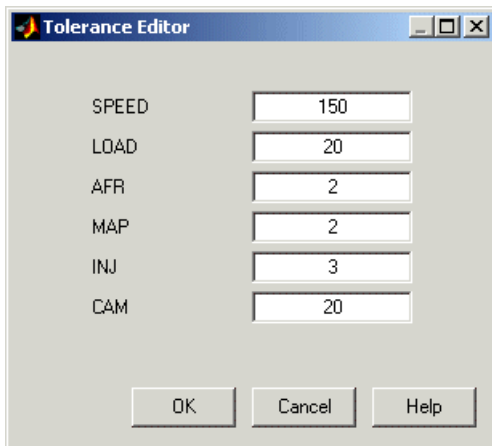
2   Click the **LOAD** edit box and enter 20, as shown. Click **OK**.

Notice the change in shape of the clusters in the **Design Match** view.
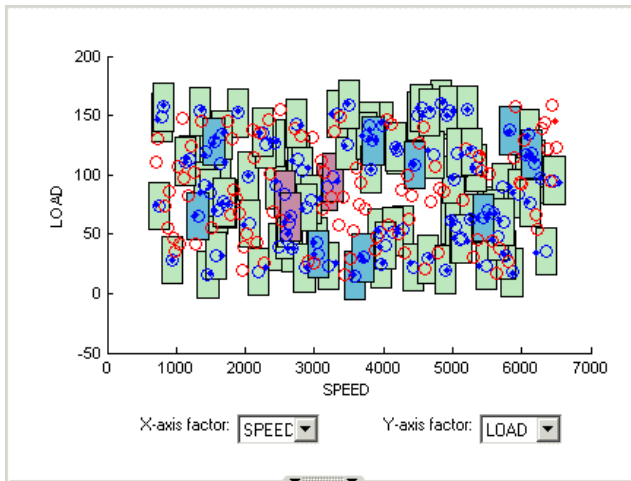
3   Shift click (center-click) and drag to zoom in on an area of the plot, as shown. You can double-click to return to the full-size plot.
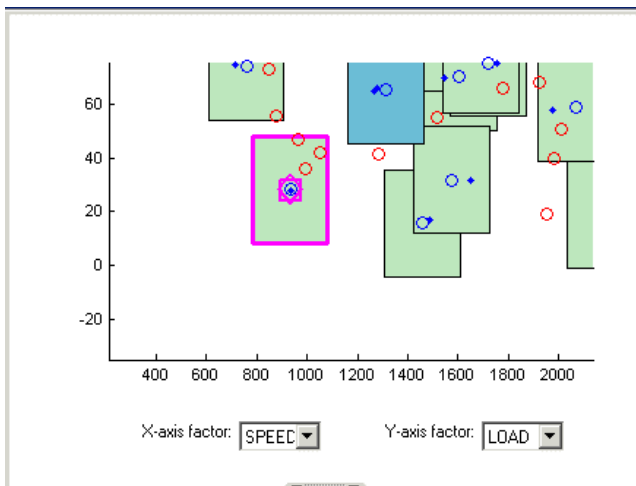
4 Click a cluster to select it. Selected points or clusters are outlined in pink. If you click and hold, you can inspect the values of global variables at the selected points (or for all data and design points if you click a cluster). You can use this information to help you decide on suitable tolerance values if you are trying to match points.

5 Notice that the **Cluster Information** list shows the details of all data and design points contained in the selected cluster. You use the check boxes here to select or exclude data or design points. Click different clusters to see various points. The list shows the values of global variables at each point, and which data and design points are within tolerance of each other. Your selections here determine which data to use for modeling, and which design points are replaced by actual data points.

Understanding Clusters

If you are not interested in collecting more data, then there is no need to make sure that the design is modified to reflect the actual data.

However, if you want your new design (called `Actual Design`) to accurately reflect what data has been obtained so far, for example to collect more data, then the cluster matching is important. All data points with a selected check box are added to the new `Actual Design`, except those in red clusters. The color of clusters indicates what proportion of selected points it contains as follows:

- Green clusters have equal numbers of selected design and selected data points. The data points will replace the design points in the `Actual Design`.

  The proportion of *selected* points determines the color; excluded points (with cleared check boxes) have no effect. Your check box selections can change cluster color.

- Blue clusters have more data points than design points. All the data points will replace the design points in the `Actual Design`.

- Red clusters have more design points than data points. These data points will not be added to your design as the algorithm cannot choose which design points to replace, so you must manually make selections to deal with red clusters if you want to use these data points in your design.

  If you do not care about the `Actual Design` (for example, if you do not intend to collect more data) and you are just selecting data for modeling, then you can ignore red clusters. The data points in red clusters are selected for modeling.

1 Right-click the **Design Match** and select **Select Unmatched Data**. Notice that the remaining unmatched data points appear in the list. Here you can use the check boxes to select or exclude unmatched data in the same way as points within clusters.

2 Select a cluster, then use the drop-down menu to change the **Y-Axis factor** to INJ. Observe the selected cluster now plotted in the new factor dimensions of SPEED and INJ.

  You can use this method to track points and clusters through the dimensions. This can give you a good idea of which tolerances to change to get points matched. Remember that points that do not form a cluster might appear to be perfectly matched when viewed in one pair of dimensions; you must view them in other dimensions to find out where they are separated beyond the tolerance value. You can use this tracking process to decide whether you want particular pairs of points to be matched, and then change the tolerances until they form part of a cluster.

3 Clear the **Equal Data and Design** check box in the **Design Match** view. You control what is plotted using these check boxes.
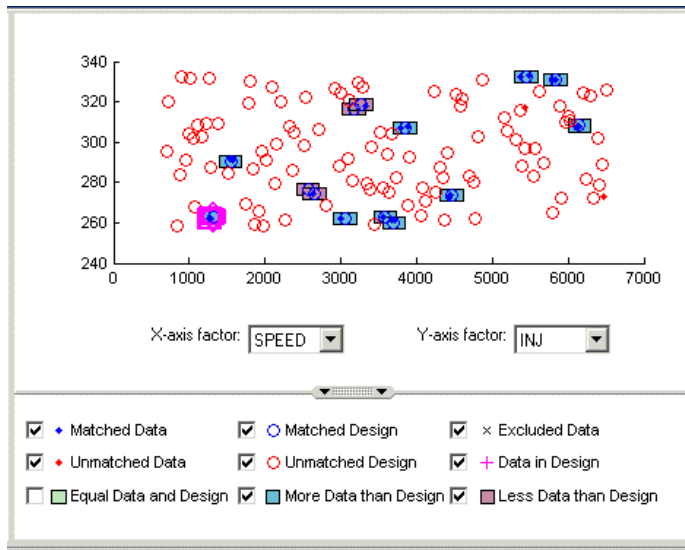
This removes the green clusters from view, as shown. These clusters are matched; you are more likely to be interested in unmatched points and clusters with uneven numbers of data and design points. Removing the green clusters allows you to focus on these points of interest. If you want your new `Actual Design` to accurately reflect your current data, your aim is to get as many data points matched up to design points as possible; that is, as few red clusters as possible.

4   Clear the check box for **More Data than Design**. You might also decide to ignore blue clusters, which contain more data points than design points. These design points are replaced by all data points within the cluster. An excess of data points is unlikely to be a concern.

However, blue clusters might indicate that there was a problem with the data collection at that point, and you might want to investigate why more points than expected were collected.
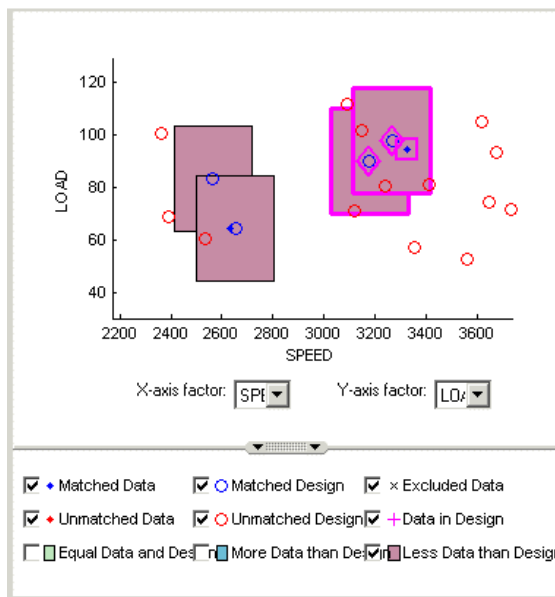
5   Select one of the remaining red clusters. Both of these have two design points within tolerance of a single data point.

6   Choose one of the design points to match to the data point, then clear the check box of the other design point. The cleared design point remains unchanged in the design. The selected design point are replaced by the matched data point.

Notice that the red cluster disappears. This is because your selection results in a cluster with an equal number of selected data and design points (a green cluster) and your current plot does not display green clusters.

7   Repeat for the other red cluster.

Now all clusters are green or blue. There are two remaining unmatched data points.

8   Clear the **Unmatched Design** check box to locate the unmatched data points. Select **Unmatched Design** check box again — to see design points to decide if any are close enough to the data points.

9   Locate and zoom in on an unmatched data point. Select the unmatched data point and a nearby design point by clicking, then use the axis drop-down menus to track the candidate pair through the dimensions. Decide if any design points are close enough to warrant changing the tolerance values to match the point with a design point.

10 Recall that you can right-click the **Design Match** and select **Select Unmatched Data** to display the remaining unmatched data points in the **Cluster Information** list. Here you can use the check boxes to select or exclude these points. If you leave them selected, they are added to the `Actual Design`.

These steps illustrate the process of matching data to designs, to select modeling data and to augment your design based on actual data obtained. Some trial and error is necessary to find useful tolerance values. You can select points and change plot dimensions to help you find suitable values. If you want your new `Actual Design` to accurately reflect your experimental data, you need to make choices to deal with red clusters. Select which design points in red clusters you want to replace with the data points. If you do not, then these data points will not be added to the new design.

When you are satisfied that you have selected all the data you want for modeling, close the Data Editor. At this point, your choices in the Design Match view are applied to the data set and a new design called `Actual Design` is created.

All data points with a selected check box are selected for modeling. Data points with cleared check boxes are excluded from the data set. Changes are made to the existing design to produce the new `Actual Design`. All selected data is added to your new design, except those in red clusters. Selected data points that have been matched to design points (in green and blue clusters) replace those design points.

All these selected data points become fixed design points (red in the Design Editor) and appear as `Data in Design` (pink crosses) when you reopen the Data Editor.

This means that these points will not be included in clusters when matching again. These fixed points will also not be changed in the Design Editor when you add points, though you can unlock fixed points if you want. This can be useful if you want to optimally augment a design, taking into account the data you have already obtained.

See Also

More About

- "Data Manipulation for Modeling"
- "Data Import and Processing"
- "Match Data to Designs"

Tradeoff Calibration

Setup and Perform 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 over the values that produce the maximum value of torque.

This example takes you through the steps required for you to set up this tradeoff, and then to calibrate the lookup table for it.

Set Up a Tradeoff Calibration

Create a Tradeoff

Start CAGE by typing

```
cage
```

at the MATLAB prompt.

Before you can calibrate the lookup tables, you must set up the calibration.

1   Select **File > Open Project** (or the toolbar button) to choose the `tradeoffInit.cag` file, found in the `matlab\toolbox\mbc\mbctraining` folder, then click **OK**.

   The `tradeoffInit.cag` project contains two models and all the variables necessary for this tutorial.

2   To create a tradeoff calibration, select **File > New > Tradeoff**.

   This takes you to the **Tradeoff** view.

Add Tables to a Tradeoff Calibration

The models of torque and NOX are in the current session. Add the lookup table to calibrate.

Both models have five inputs. The inputs for the torque and NOX models are

- Exhaust gas recycling (EGR)
- Air/fuel ratio (AFR)
- Spark angle
- Speed
- Load

For this tutorial, you are interested in the spark angle over the range of speed and load.

To generate a lookup table for the spark angle,

1   Click ![icon](Add New Table) in the toolbar. This opens the Table Setup dialog box.
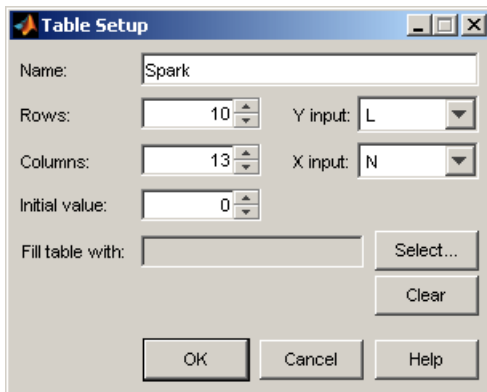
2   Enter `Spark` as the table **Name**.

3   Check that `N` is the **X input** and `L` is the **Y input** (these are selected automatically as the first two variables in the current Variable Dictionary).

4   Enter `10` as the size of the load axis (**Rows**).

5   Enter `13` as the size of the speed axis (**Columns**).

6   Click **Select** to open the dialog box Select Filling Item.
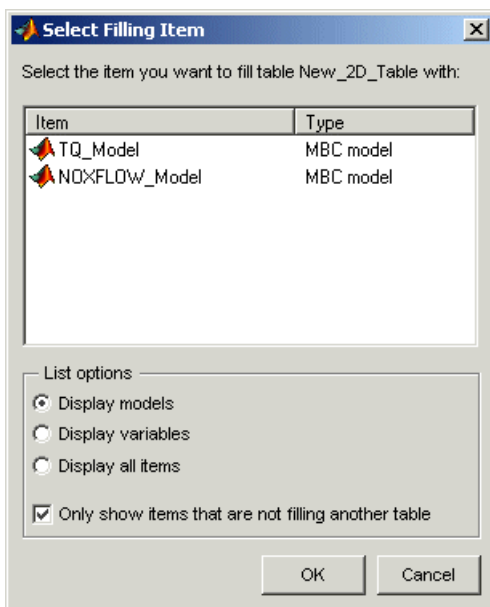
Select the option button **Display variables**, then select `SPK` to fill the table and click **OK**.

7   Click **OK** to close the Table Setup dialog box.

Before you can perform the calibration, you must display the models.

Display the Models

For this tutorial, you are comparing values of the torque and NOX models. Thus, you need to display these models.

To display both models:

- Click ![icon] Add Model to Display List in the toolbar twice. This moves both available models into the Display list.

- Alternatively, **Shift**-click to select both models in the **Available Models** list and click ![>] to include both models in the current display. In this case, you want to include all available models. You can click to select particular models in the list to display.

The **Display Models** pane following shows both models selected for display.

Perform the Tradeoff Calibration

You now fill the lookup table for spark angle by trading off gain in torque for reduction in NOX emissions.

The method that you use to fill the lookup table is

- Obtain the maximum possible torque.
- Restrict NOX to below 250 g/hr at any operating point.

Once you have completed the calibration, you can export the calibration for use in the electronic control unit.

Check the Normalizers

A normalizer is the axis of the lookup table (which is the collection of breakpoints). The breakpoints of the normalizers are automatically spaced over the ranges of speed and load. These define the operating points that form the cells of the tradeoff table.

Expand the Tradeoff tree by clicking the plus sign in the display, so you can see the `Spark` table and its normalizers `Speed` and `Load`. Click to highlight either normalizer to see the normalizer view. A tradeoff calibration does not compare the model and the table directly, so you cannot space the breakpoints by reference to the model.

NNormalizer (N)

Input	Output
500	0
1000	1
1500	2
2000	3
2500	4
3000	5
3500	6
4000	7
4500	8
5000	9
5500	10
6000	11
6500	12

Normalizer Display | Breakpoint Spacing

LNormalizer (L)

Input	Output
0.1	0
0.2	1
0.3	2
0.4	3
0.5	4
0.6	5
0.7	6
0.8	7
0.9	8
1	9

Normalizer Display | Breakpoint Spacing

Set Values for Other Variables

At each operating point, you must fill the values of the spark table. Both of the models depend on spark, AFR (labeled A, in the session), and EGR (labeled E in the session). You could set the values for AFR and EGR individually for each operating point in the table, but for simplicity you set constant values for these model inputs.

To set constant values of AFR and EGR for all operating points,

1 Click **Variable Dictionary** in the **Data Objects** pane.

2 Click A and edit the **Set Point** to `14.3`, the stoichiometric constant, and press **Enter**.

3 Click E and change the **Set Point** to `0` and press **Enter**.

You have set these values for every operating point in your tradeoff table. You can now fill the spark angle lookup table. The process is described next.

4 Click **Tradeoff** in the **Processes** pane to return to the tradeoff view.

5 Highlight the `Spark` table node in the Tradeoff tree display.

6 In the lower pane, check that the value for A is `14.3`, and the value for E is `0`, as shown in the following example. You leave these values unchanged for each operating point.

For each operating point, you change the values of spark to trade off the torque and NOX objectives; that is, you search for the best value of spark that gives acceptable torque within the emissions

constraint. The following example illustrates the controls you use, and there are step-by-step instructions in the following section.
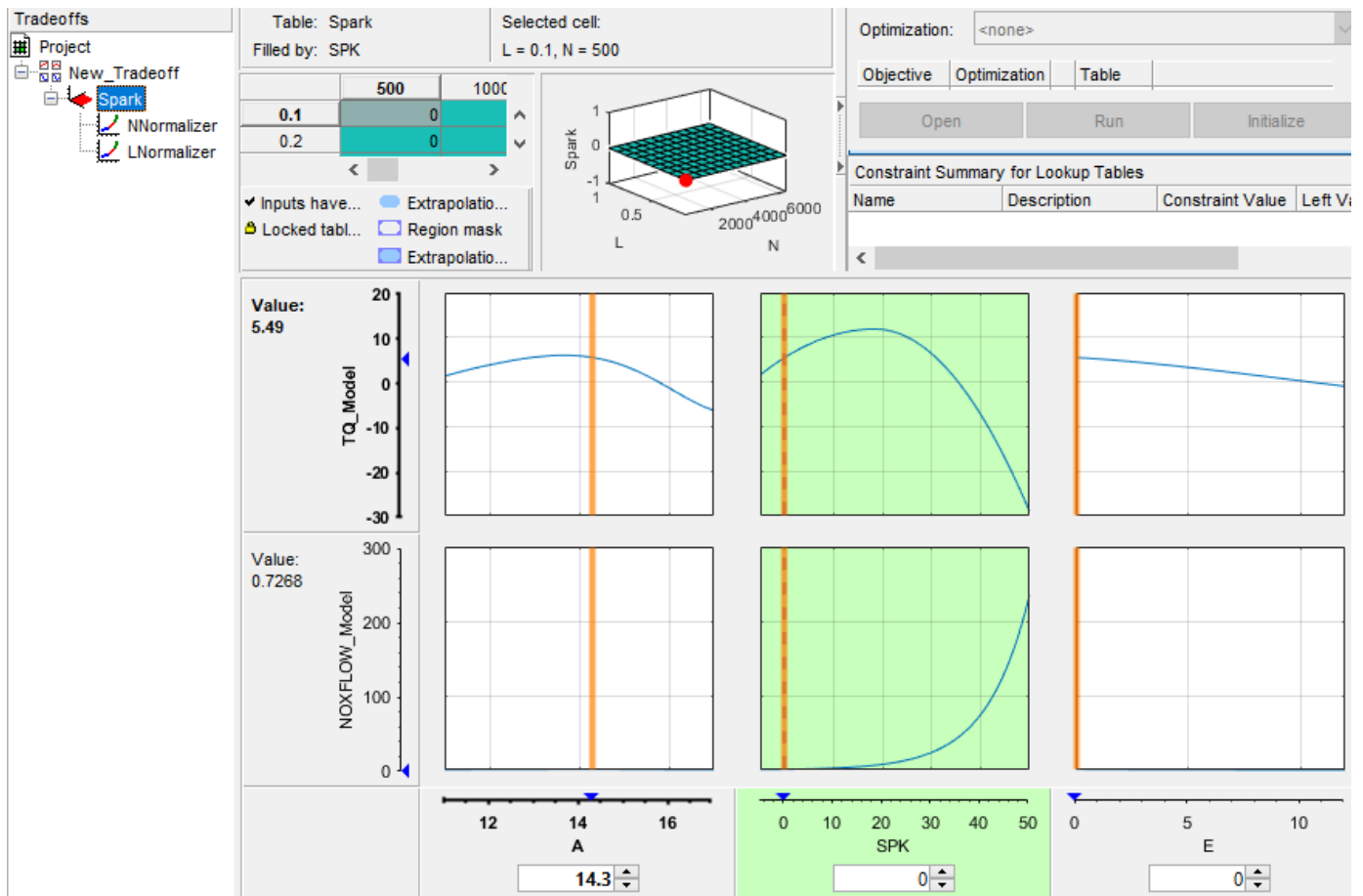
Fill Key Operating Points

You now fill the key operating points in the lookup table for spark angle.

The upper pane displays the lookup table, and the lower pane displays the behavior of the torque and NOX emissions models with each variable.

The object is to maximize the torque and restrict NOX emissions to below 250 g/hr.

At each operating point, the behavior of the model alters. The following display shows the behavior of the models over the range of the input variables at the operating point selected in the table, where speed (N) is 4500 and load (L) is 0.5. You can show confidence intervals by selecting **View > Display Confidence Intervals**.
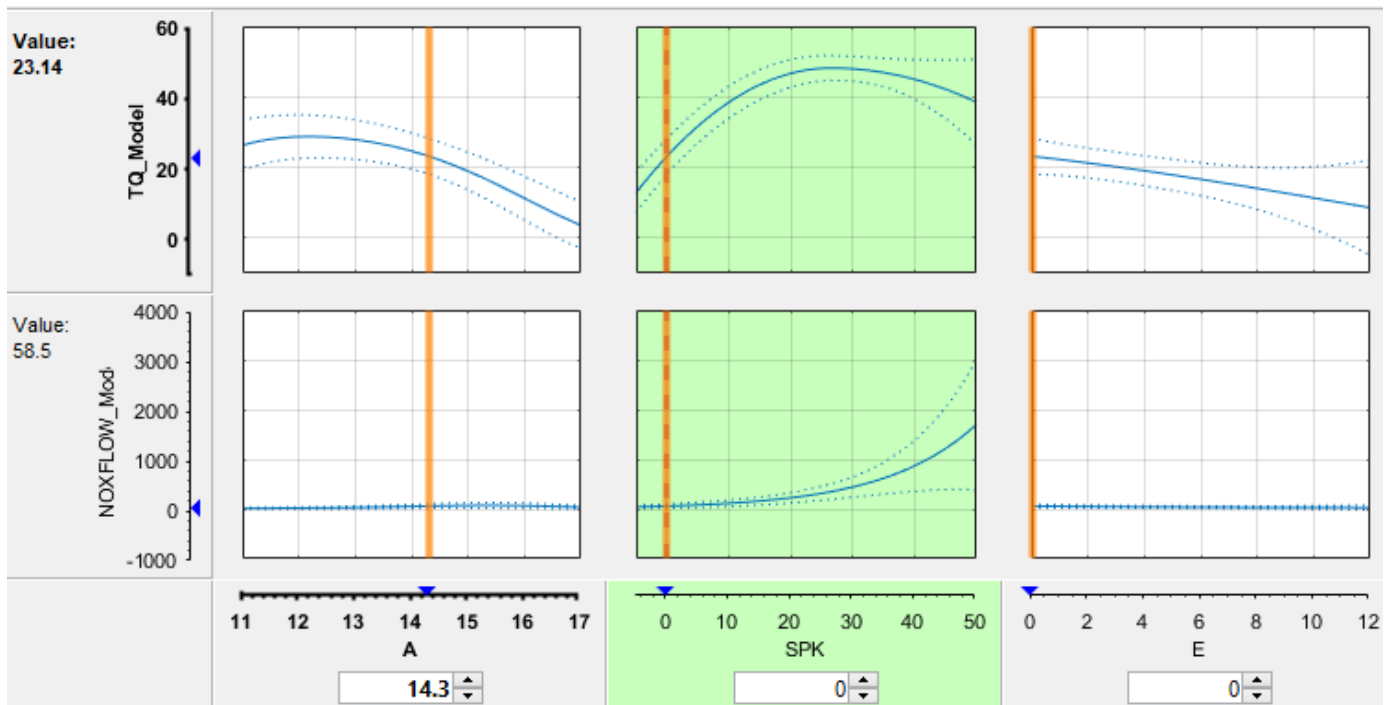
The top three graphs show how the torque model varies with the AFR (labeled A), the spark angle (SPK), and the EGR (E), respectively. The lower panes show how the NOX emissions model varies with these variables.

You are calibrating the Spark table, so the two spark (SPK) graphs are green, indicating that these graphs are directly linked to the currently selected lookup table.

1   Select the operating point N = 4500 and L = 0.5 in the lookup table.

2   Now try to find the spark angle that gives the maximum torque and restricts NOX emissions to below 250 g/hr. You can change the value of spark by clicking and dragging the orange line on the SPK graphs, or by typing values into the SPK edit box. You can change the values of any of the other tradeoff variables in the same way, but as you have already set constant values for A and E you should not change these. Try different values of spark and look at the resulting values of the torque and NOX models.

3   Click to select the top SPK - TQ_Model graph (TQ_Model row, SPK column). When selected the graph is outlined as shown in the following example.

4   Now click 'Find maximum of output' (⚹) in the toolbar. This calculates the value of spark that gives the maximum value of torque. The following display shows the behavior of the two models when the spark angle is 26.4458, which gives maximum torque output.
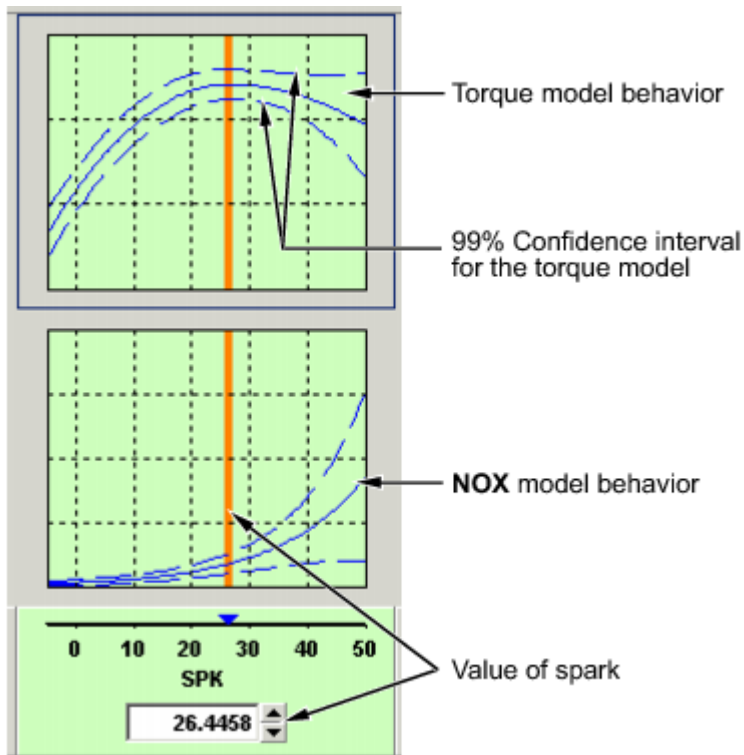
At this operating point, the maximum torque that is generated is 48.136 when the spark angle is 26.4989. However, the value of NOX is 348.968, which is greater than the restriction of 250 g/hr. Clearly you have to look at another value of spark angle.

5   Click and drag the orange bar to change to a lower value of spark. Notice the change in the resulting values of the torque and NOX models.

6   Enter 21.5 as the value of **SPK** in the edit box at the bottom of the SPK column.

     The value of the NOX emissions model is now 249.154. This is within the restriction, and the value of torque is 47.2478.

     At this operating point, this value of 21.5 degrees is acceptable for the spark angle lookup table, so you want to apply this point to your table.

7   Press **Ctrl**+T or click 🗗 (Fill lookup tables with current tradeoff values and save) in the toolbar to apply that value to the spark table.

     This automatically adds the selected value of spark to the table and turns this cell yellow. It is blue when selected, yellow if you click elsewhere. Look at the table legend to see what this means: yellow cells have been added to the extrapolation mask, and the tick mark indicates that you saved this input value by applying it from the tradeoff. You can use the **View** menu to choose whether to display the legend.

8   Now repeat this process of finding acceptable values of spark at four more operating points listed in the table following. In each case,

     • Select the cell in the spark table at the specified values of speed and load.

     • Enter the value of spark given in the table (the spark angles listed all satisfy the requirements).

- Press **Ctrl**+T or click ⧉ (Fill lookup tables with current tradeoff values and save) in the toolbar to apply that value to the spark table.

Speed, N	Load, L	Spark Angle, SPK
2500	0.3	25.75
3000	0.8	10.7
5000	0.7	8.2
6000	0.2	41.3

Fill the Table by Extrapolation

When you have calibrated several key operating points, you can produce a smooth extrapolation of these values across the whole table.

When you apply the value of the spark angle to the lookup table, the selected cell is automatically added to the extrapolation mask. This is why the cell is colored yellow. The extrapolation mask is the set of cells that are used as the basis for filling the table by extrapolation.

Click ⧉ in the toolbar to fill the table by extrapolation.

The lookup table is filled with values of spark angle.

The following figure displays the view after extrapolation over the table.
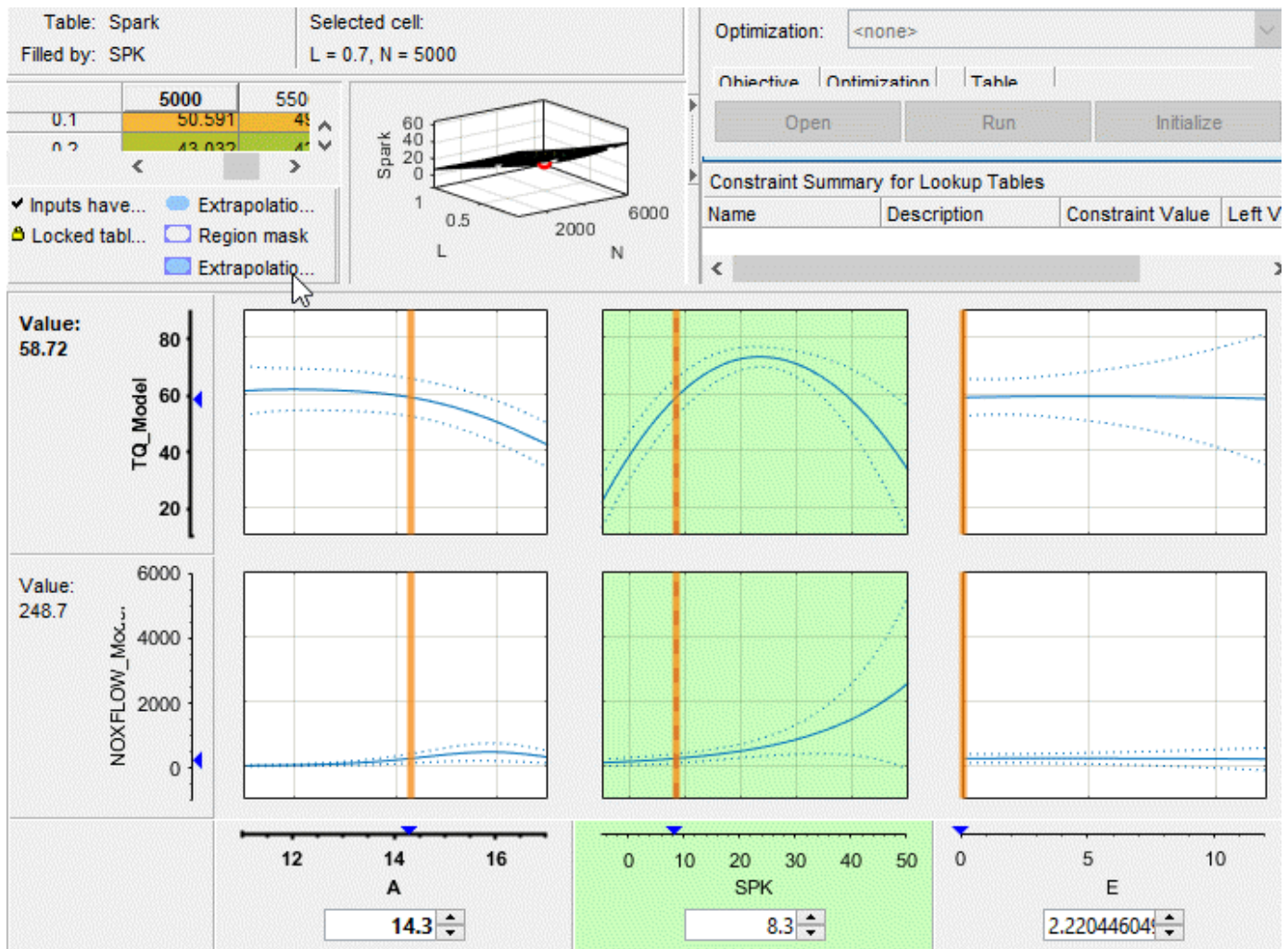
Note Not all the points in the lookup table will necessarily fulfill the requirements of maximizing torque and restricting the NOX emissions.

Export Calibrations

To export your table and its normalizers:

1 Select the `Spark` node in the branch display.

2 Select **File > Export > Calibration**.

3 Choose the file type you want for your calibration. For the purposes of this tutorial, select `Comma Separated Value (.csv)`.

4 Enter `tradeoff.csv` as the file name and click **Save**.

This exports the spark angle table and the normalizers, `Speed`, and `Load`.

See Also

More About

- "Calibration Setup"
- "Tradeoff Calibration"

Data Sets

Compare Calibrations To Data

Setting Up the Data Set

You can use the **Data Sets** view in CAGE to compare features, tables, and models with experimental data. You can use data sets to plot the features, tables, etc., as tabular values or as plots on a graph.

Data sets enable you to view the data at a set of operating points. You can determine the set of operating points yourself, using Build Grid. Alternatively, you can import a set of experimental data taken at a series of operating points. These operating points are not the same as the breakpoints of your tables.

This tutorial takes you through the basic steps required to compare a completed feature calibration to a set of experimental data.

Start CAGE by typing

```
cage
```

To set up the data set tutorial, you need to

1   Open an existing calibration on page 9-2.

2   Import the experimental data on page 9-2.

3   Add on page 9-4 the `Torque` feature to the data set.

Your data set contains all the input factors and output factors required. As the imported data contains various operating points, this information is also included in the data set.

Opening an Existing Calibration

For this tutorial, use the file `datasettut.cag`, found in the `matlab\toolbox\mbc\mbctraining` directory.

To open this file,

1   Select **File > Open Project**.

2   In the file browser, select `datasettut.cag` and click **Open**.

This opens a file that contains a complete calibrated feature with its associated models and variables. This particular feature is a torque calibration, using a torque table (labeled `T1`) and modifiers for spark (labeled `T2`) and air/fuel ratio (labeled `T3`).

3   Select **File > New > Data Set** to add a new data set to your session.

This automatically switches you to the **Factor Information** pane of the data set display.

Importing Experimental Data into a Data Set

To import data into a data set,

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

2   In the file browser, select `meas_tq_data.xls` from the `mbctraining` directory, and click **Open**.

This set of data includes six columns of data, the test cell settings for engine speed (RPM), and the measured values of torque (`tqmeas`), engine speed (`nmeas`), air/fuel ratio (`afrmeas`), spark angle (`spkmeas`), and load (`loadmeas`).

3   The Data Set Import Wizard asks which of the columns of data you would like to import. Click **Next** to import them all.

The following screen asks you to associate variables in your project with data columns in the data.

4   Highlight `afr` in the **Project Assignments** column and `afrmeas` in the **Data Column**, then click the assign button, shown.

5   Repeat this to associate `load` with `loadmeas`, `n` with RPM, and `spk` with `spkmeas`. The dialog box should be the same as shown.
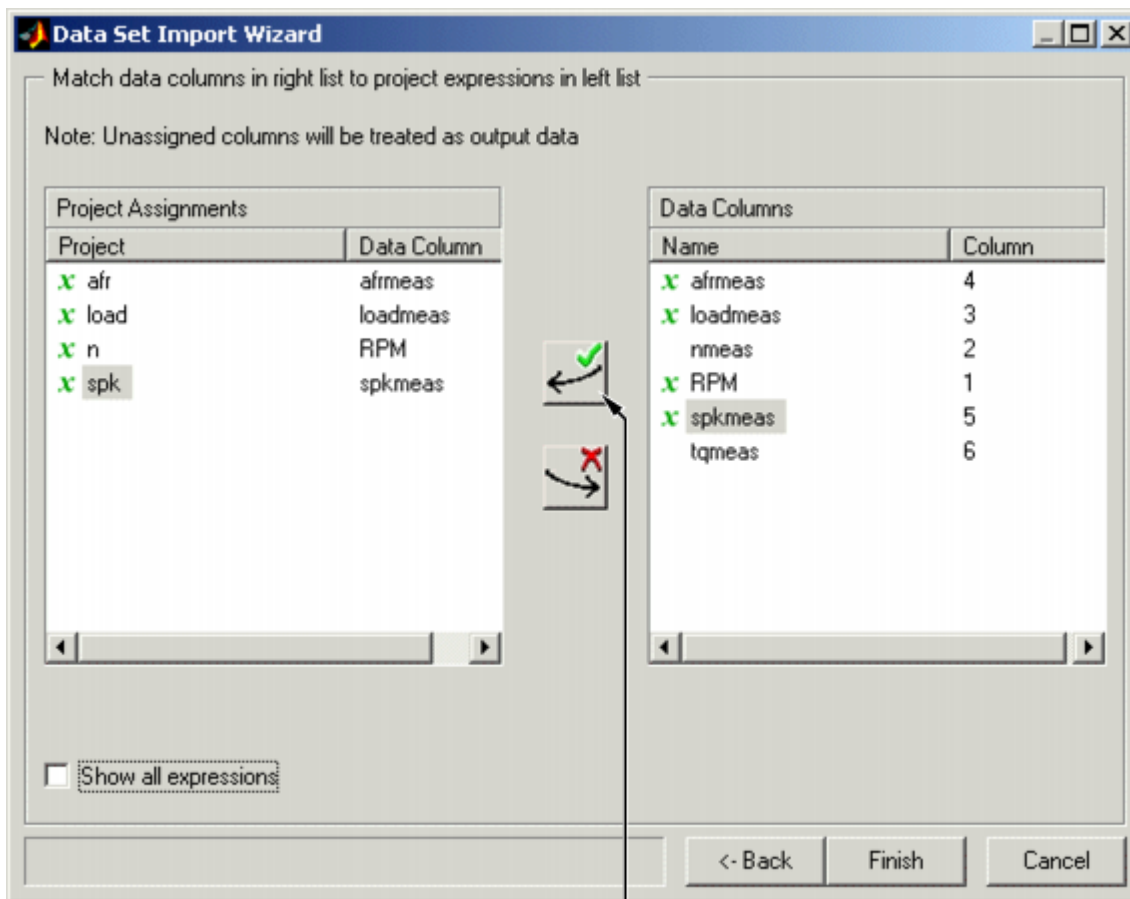
Assign button

6   Click **Finish** to close the dialog box.

Note If you need to reassign any inputs after closing this dialog you can click ▣ in the toolbar or select **Data > Assign**.

Adding an Item to a Data Set

To add the `Torque` feature to the data set,

1   Highlight the `Torque` feature in the lower list of **Project Expressions**.

2   Select **Data** > **Factors** > **Add to Data Set**.

This adds two objects to the data set: `Torque: Model` and `Torque: Strategy`. These two objects make up the `Torque` feature.

- `Torque: Model` is the model used as a reference point to calibrate the feature.
- `Torque: Strategy` is the values of the feature at these operating points.

When these steps are complete, the list of factors includes four input factors and four output factors, as shown.
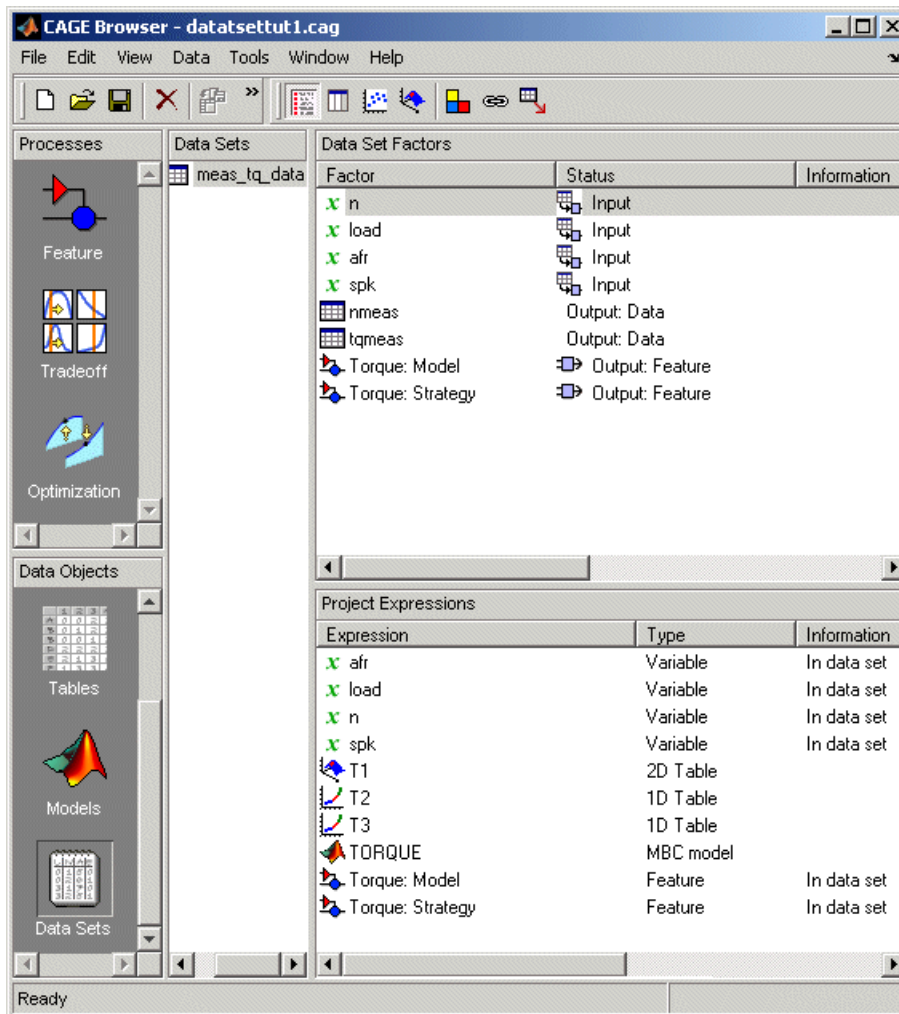
Comparing the Items in a Data Set

Viewing the Data Set as a Table

By viewing the data set, you can compare experimental data with calibrations or models in your project.

Click [icon] in the toolbar to view the data set as a table of values.

	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

In the table, the input cells are white and the output cells are grey. Select the `Torque: Strategy` column header to see the view shown. The selected column turns blue and the column headers of the strategy's inputs (`n`, `load`, `afr` and `spk`) turn cream. Column headers are always highlighted in this way when they are associated with the currently selected column (such as model inputs, strategy inputs or linked columns).

In addition to viewing the columns, you can use data sets to create a column that shows the difference between two columns:

1   Select the `tqmeas` and `Torque: Strategy` columns by using **Ctrl**+click.

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

This creates another column that is the difference between `tqmeas` and `Torque: Strategy`. Note that all the columns that are inputs to this new column have highlighted headers.

	🔧 n	🔧 load	🔧 afr	🔧 spk	nmeas	tqmeas	➡ Torque: Model	➡ Torque: Strategy	➡ tqmeas_minus_Torque
1	2235	0.549	9.5	0.1	2247	66.7	71.666	66.079	-0.621
2	3591	0.454	13.2	0.1	3613	54.1	47.163	46.891	-7.209
3	4946	0.651	12	0.1	4974	73.7	47.573	79.256	5.556
4	881	0.648	11.9	5.7	881	75.8	99.23	80.211	4.411
5	2234	0.441	13.3	0.1	2247	55.9	51.256	45.152	-10.748
6	3591	0.747	10.9	0.1	3612	90	92.837	105.586	15.586
7	4947	0.541	9.7	0.1	4973	62.8	57.76	57.587	-5.213
8	881	0.622	9.9	0.1	884	72.1	76.198	60.926	-11.174
9	1219	0.333	14	0.1	1224	41.8	33.226	21.318	-20.482
10	1558	0.382	12	0.1	1567	49.4	40.487	31.957	-17.443
11	1896	0.209	10.7	3.3	1906	28.5	3.492	4.197	-24.303
12	2234	0.284	9.8	3.2	2245	36	23.063	19.891	-16.109
13	2574	0.407	13.4	3	2588	49.9	49.629	44.794	-5.106
14	2914	0.595	11.5	3.1	2929	70.5	84.68	82.229	11.729
15	3251	0.781	12.3	3.1	3268	90.5	117.424	117.259	26.759
16	3589	0.668	13.5	3	3608	77.1	87.987	96.408	19.308
17	3930	0.452	11.9	3.1	3952	52.7	46.511	51.722	-0.978
18	4268	0.235	10.9	3	4293	27.7	5.253	3.085	-24.615
19	4606	0.194	12	3.2	4633	21.3	-2.088	-5.771	-27.071

The error column is simply the difference between `tqmeas` and `Torque: Strategy`. This provides a simple way of comparing the feature and the measured data.

Viewing the Data Set as a Plot

1 Click ⬚ or select **View > Plot** to view the data set as a plot.

The lower pane lists all the output expressions in the data set and in the project.

2 Use **Ctrl**+click to select `tqmeas` and `Torque: Strategy` from the lower list.
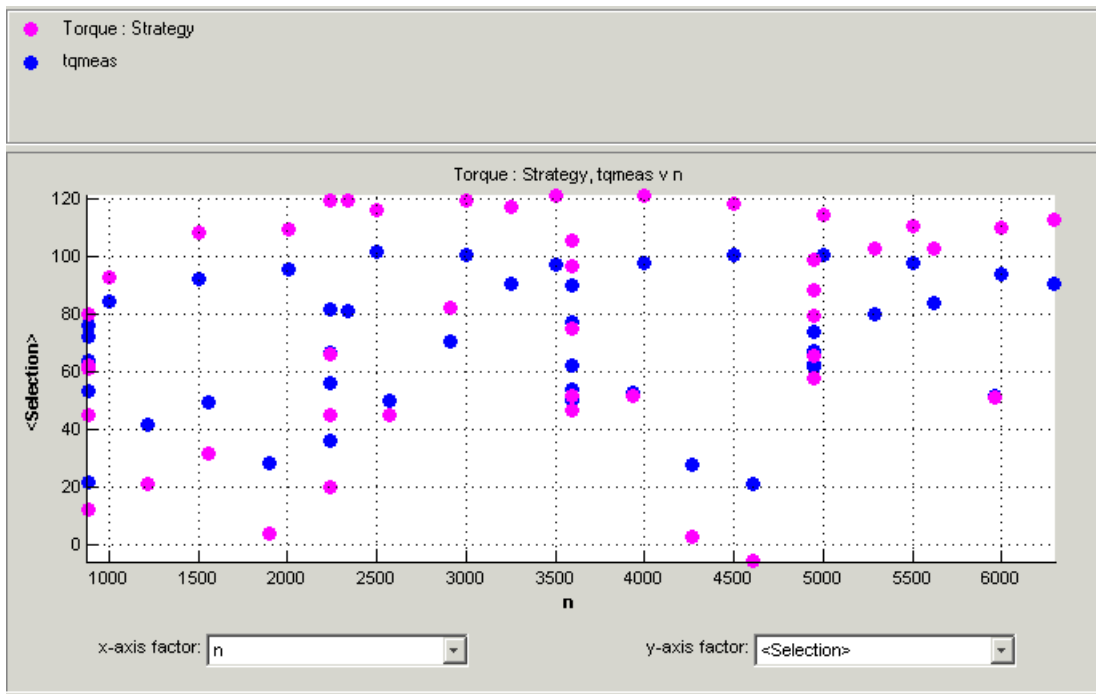
3 Change the **x-axis factor** to n from the drop-down menu.

This displays the calibrated values of torque from the feature, and the measured values of torque from the experimental data, against the test cell settings for engine speed.

Clearly there is some discrepancy between the two.

Displaying the Error

View the error between the calibrated and measured values of torque.
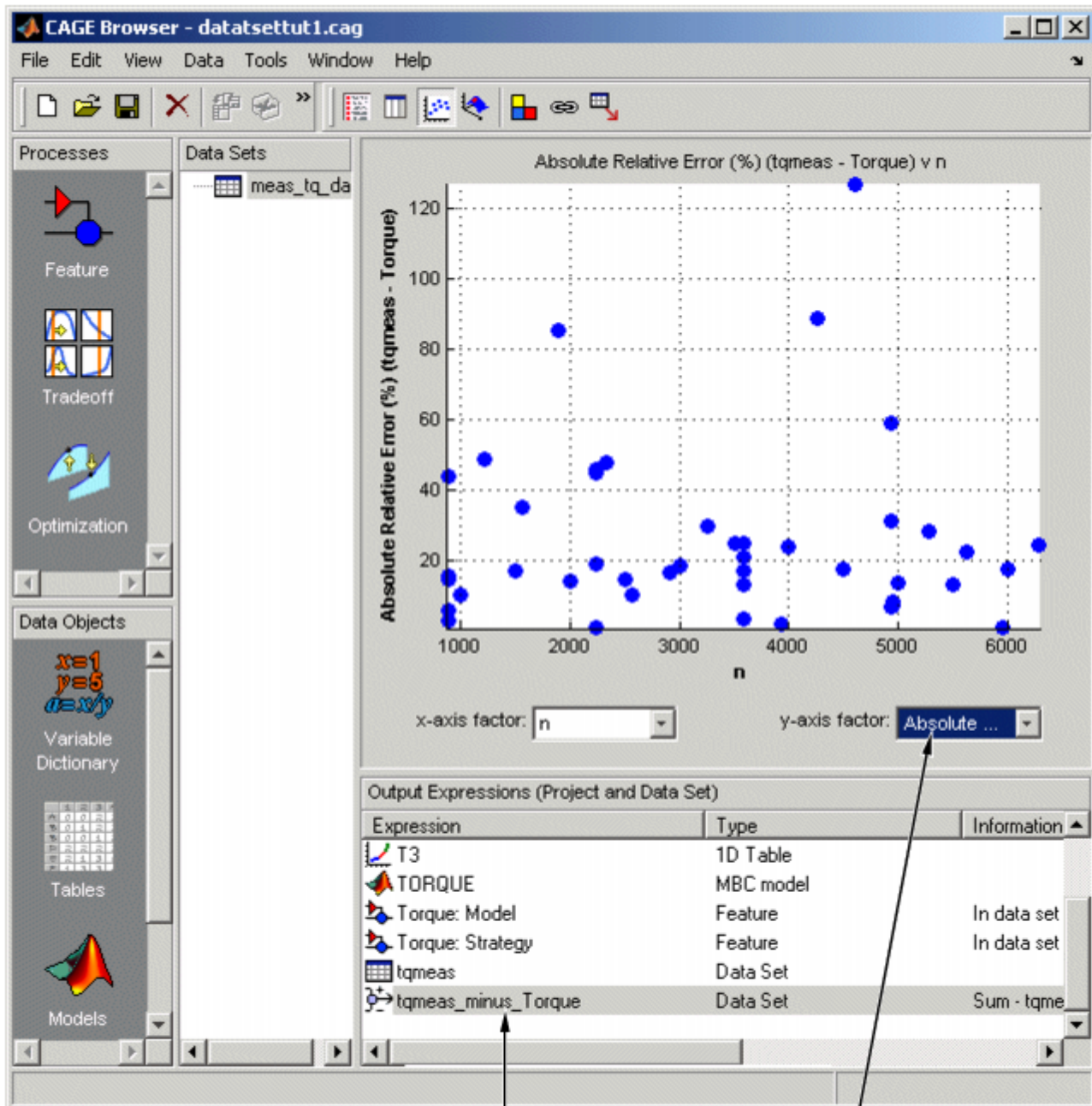
1. Select tqmeas_minus_Torque.

2. Select Absolute Relative Error (tqmeas - Torque).

1 Select tqmeas_minus_Torque from the lower list (**Output Expressions**).

2 For the **y-axis factor**, select Absolute Relative Error (tqmeas - Torque) from the drop-down menu.

As you can see, there seems to be no particular correlation between engine speed and the error in the calibration.

Coloring the Display

1 Select **Color by Value** from the right-click menu on the graph.

2 From the **Color by** drop-down menu, select load.

In this display, you can see that some of the low values of load display a high error.

Limiting the Range of the Colors
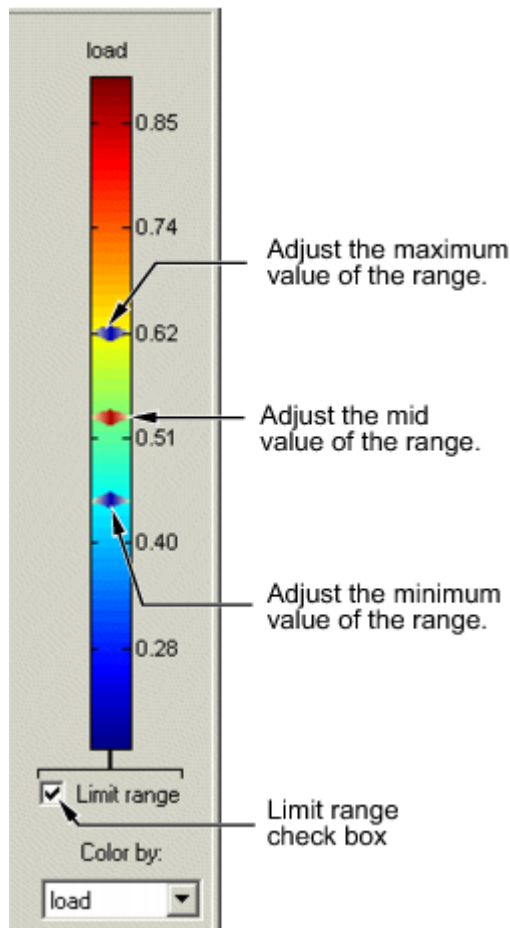
To view the colors in more detail, you can limit the range of the colors:

1 Select the **Limit range** box (or you could right-click the graph and select **Restrict Color to Limits**).

2 Set the minimum value of the color range to be as low as possible by dragging the minimum value down.

3   Set the maximum value of the color range to be around 0.4.

As the low values of load are causing large errors, it would be wise to reexamine the calibration, particularly at small values of `load`.

Reassigning Variables

You can alter the data set by changing which variables are used for project expressions.

Instead of using the test cell settings for the engine speed (RPM), you might want to use the measured values of engine speed (`nmeas`). So you have to reassign the variable `n` to `nmeas`.

To reassign `n`,

1   Click ⊞ or select **Data > Assign**.

2   In the dialog that appears, select `n` from the **Project Assignments** pane and `nmeas` from the **Data Columns** pane.

3   Click the assign button.
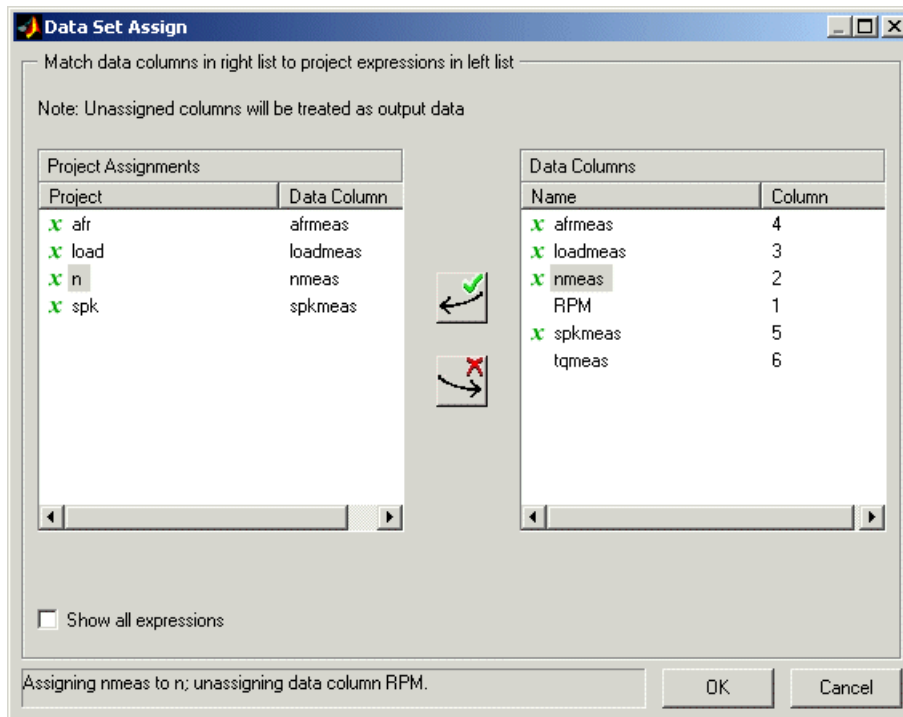
See Also

More About

- "Data Sets in CAGE"
- "Feature Calibration"

Filling Lookup Tables from Data

Fill Lookup Tables from Data

Setting Up a Lookup Table and Experimental Data

If you are considering a straightforward strategy, you might want to fill lookup tables directly from experimental data. For example, a simple torque strategy fills a lookup table with values of torque over a range of speed and relative air charge, or load. You can use CAGE to fill this strategy (which is a set of lookup tables) by referring to a set of experimental data. You can also fill lookup tables with the output of optimizations.

This tutorial takes you through the steps of calibrating a lookup table for torque, based on experimental data.

Start CAGE by typing

`cage`

First, set up a blank lookup table ready for filling using experimental data or optimization output.

1 Add the variables on page 10-2 for speed and load by importing a variable dictionary.
2 Add a new table on page 10-2 to your session.
3 Import your experimental data on page 10-4.

Add Variables

Before you can add lookup tables to your session, you must add variables to associate with the normalizers or axes.

To add a variable dictionary,

1 Select **File** > **Import** > **Variable Dictionary**.
2 Select `table_filling_tutorial.xml` from the `matlab\toolbox\mbc\mbctraining` folder.

This loads a variable dictionary into your session. The variable dictionary includes:

- `N`, the engine speed
- `L`, the relative air charge
- `A`, the air/fuel ratio (AFR)
- `stoich`, the stoichiometric constant

You can now add a lookup table to your session.

Add a New Lookup Table

First, add a lookup table to fill.

1 Select **File** > **New** > **2D Lookup Table**.

This opens a dialog box that asks you to specify the variable names for the normalizers. As you can see in the dialog box controls, accepting the defaults creates a lookup table with 10 rows and 10 columns with an initial value of `0` in each cell.

2 Change the number of columns to 7.

3   Select L as the variable for normalizer **Y** and N as the variable for normalizer **X**, then click **OK**.

CAGE takes you to the **Lookup Tables** view.
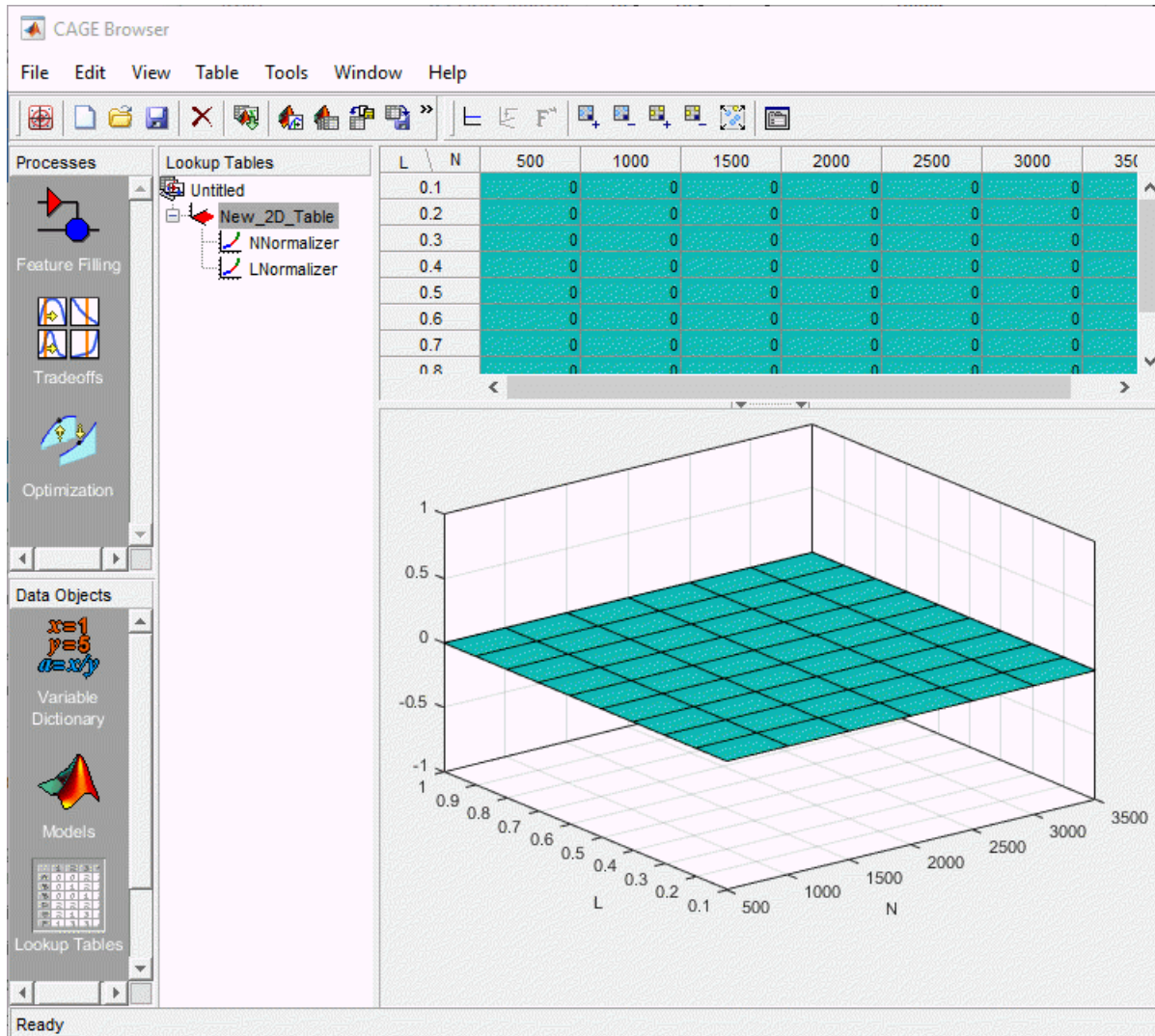
CAGE has automatically initialized the normalizers by spacing the breakpoints evenly across the range of values for the engine speed (N) and load (L). The variable ranges are found in the variable dictionary. Switch to the Normalizer view to inspect the normalizers.

Expand the lookup table branch by t `NNormalizer` as shown.
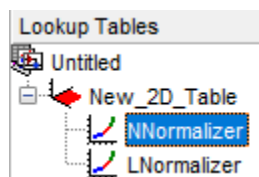
This displays the two normalizers for the lookup table.

You have an empty lookup table with breakpoints over the ranges of the engine speed and load, which you can fill with values based on experimental data.

Import Experimental Data

To fill a lookup table with values based on experimental data, you must add the data to your session. If you want to fill a lookup table with the output of an optimization, the output appears automatically in the Data Sets view as a new data set called `Exported_Optimization_Data` when you select the Export to Data Set toolbar button. For this tutorial, you need some experimental data.

CAGE uses the **Data Sets** view to store grids of data. Thus, add a data set to your session.

Select **File > New > Data Set** to add a data set to your session. This changes the view to the **Data Set** view.

You can now import experimental data into the data set:

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

2   In the file browser, select `meas_tq_data.csv` from the `matlab\toolbox\mbc\mbctraining` folder and click **Open**.

   This set of data includes six columns of data: the test cell settings for engine speed (RPM), and the measured values of torque (`tqmeas`), engine speed (`nmeas`), air/fuel ratio (`afrmeas`), spark angle (`spkmeas`), and load (`loadmeas`).

3   This opens the Data Set Import Wizard. The first screen asks which of the columns of data you want to import. Click **Next** to import them all.

   The following screen asks you to associate variables in your project with data columns in the data.

4   Highlight N in the **Project Assignments** column and `nmeas` in the **Data Column**, then click the assign button, shown.

5   Repeat this to associate L with `loadmeas`. The dialog box should be the same as the following.
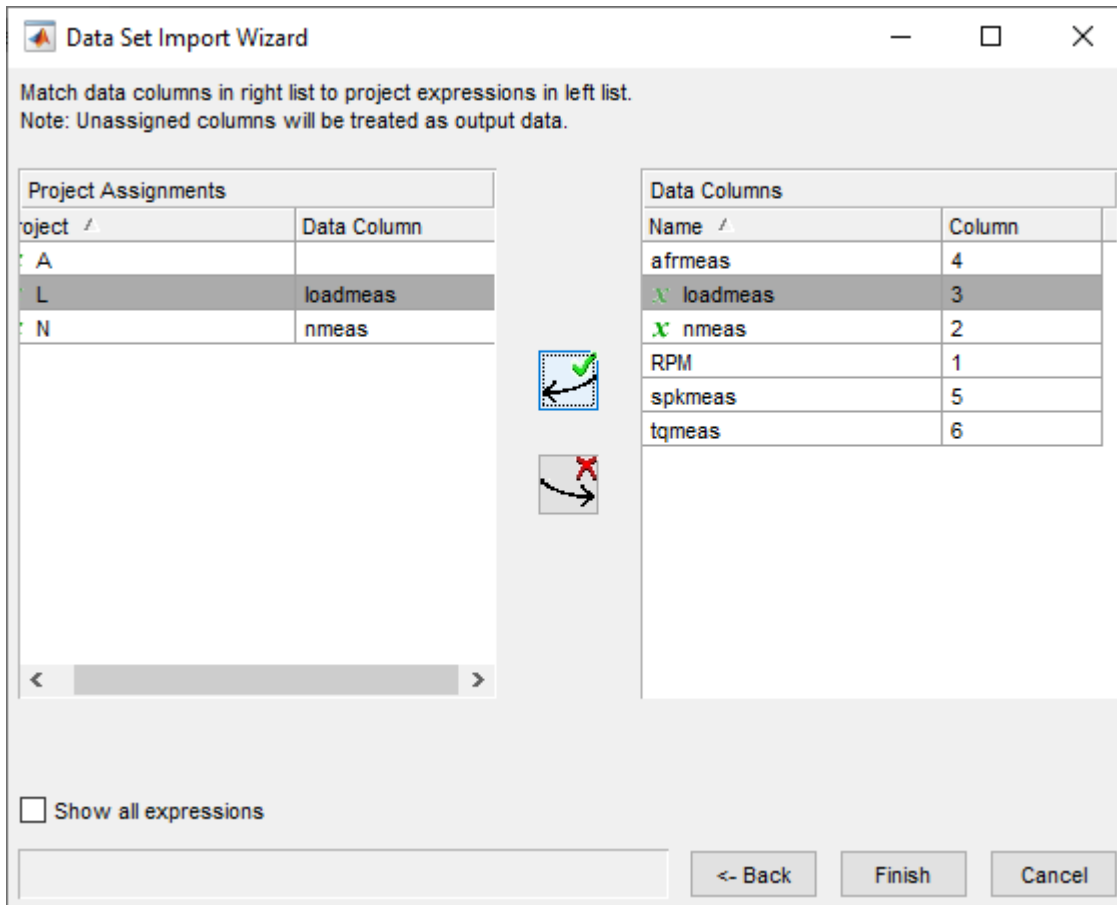
6 Click **Finish** to close the dialog box.

You now have an empty lookup table and some experimental data in your session. You are ready to fill the lookup table with values based on this data.

Fill Lookup Table fromExperimental Data

You have an empty lookup table and the experimental data in your session. You can now fill the lookup table with values based on your data.

The data that you have imported is a series of measured values of torque at a selection of different operating points. These operating points do not correspond to the values of the breakpoints that you have specified. The lookup table has a range of engine speed from 500 revolutions per minute (rpm) to 3500 rpm. The range of the experimental data is far greater.

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

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

1 To view the **Lookup Table Filler** display, click in the toolbar in the **Data Sets** view; or select **View > Lookup Table Filler**.

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

2   In the lower pane, select `New_2D_Table` from the **Lookup table to fill** list.

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

4   Select `N` from the **X-axis factor** list and `L` from the **Y-axis factor** list. Your session should be similar to this.

The upper pane displays the breakpoints of your lookup 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 lookup table before you fill the lookup table.
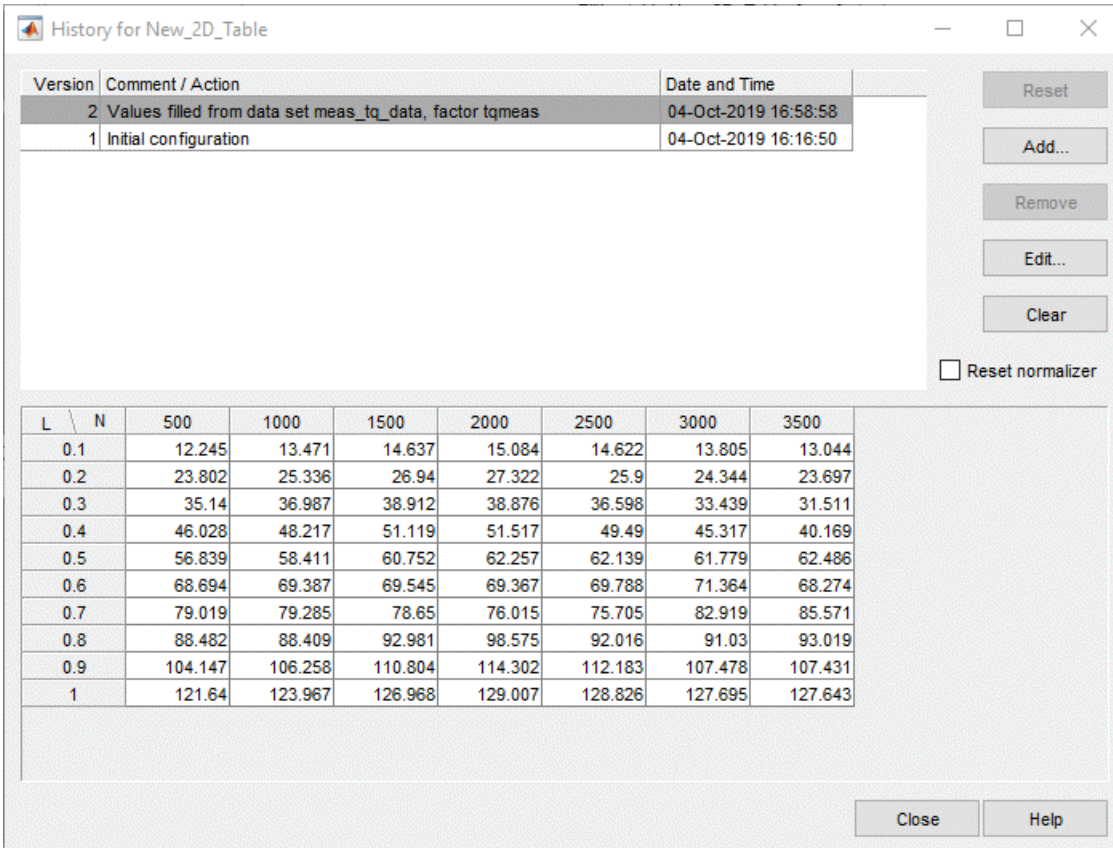
5 To view the lookup table after it is filled, ensure that the **Show lookup table history after fill** box, at the bottom left, is selected.

6 To fill the lookup table with values of `tqmeas` extrapolated over the range of the normalizers, click **Fill**.

This opens the History dialog box

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

7 Click **Close** to close the History dialog box and return to the **Lookup Table Filler** display.

8 To view the graph of your lookup table, as shown, select **Data > Plot > Surface**.
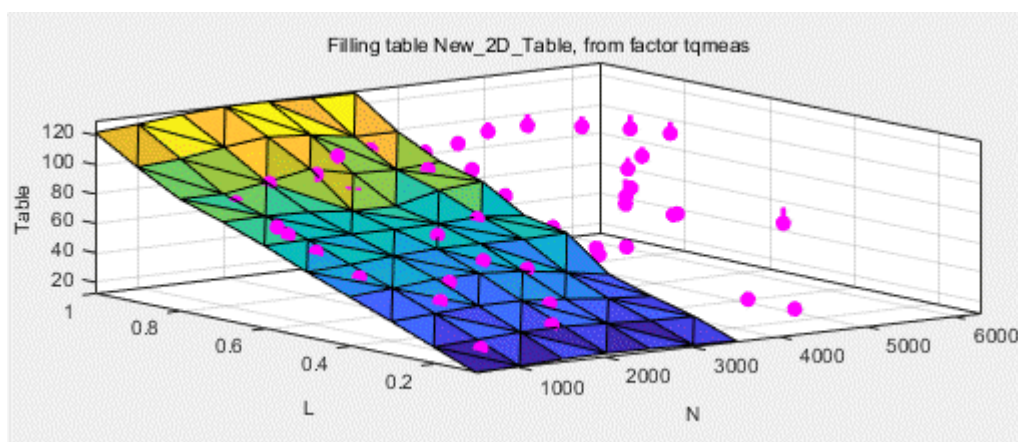
This display shows the lookup table filled with the experimental points overlaid as purple dots.

The lookup table has been calibrated by extrapolating over the values of your data and filling the values that the data predicts at your breakpoints.

Notice that the range of the lookup table is smaller than the range of the data, as the lookup table only has a range from 500 rpm to 3500 rpm.

The data outside the range of the lookup table affects the values that the lookup table is filled with. You can exclude the points outside the range of the lookup table so that only points in the range that you are interested in affect the values in the lookup table.

Select Data Regions

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

For example, in this tutorial the experimental data ranges over values that are not included in the lookup table. You want to ignore the values of engine speed that are greater than the range of the lookup table.

To ignore points in the data set,

1 Select **Data > Plot > Data Set**. This returns you to 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. Highlight all the points that are included in your lookup table range, as shown.
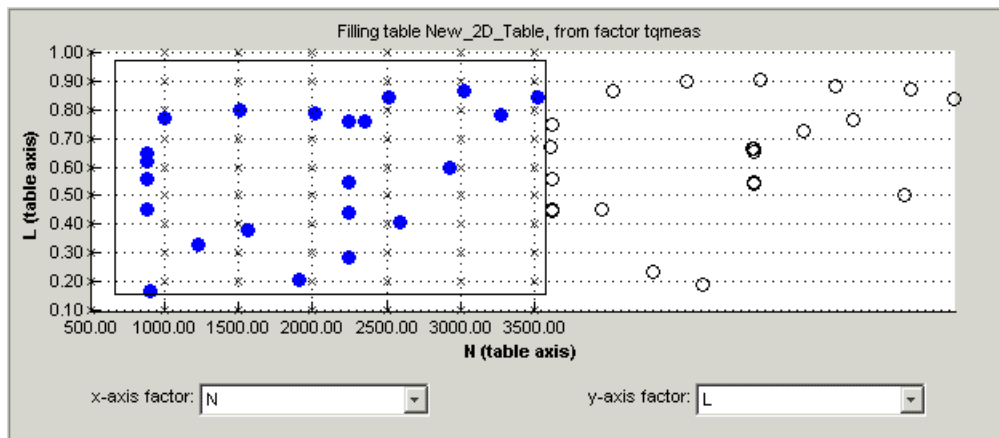
3 To fill the lookup table based on an extrapolation over these data points only, click **Fill**. This opens the History display again.

4 In the History display, select version 2 and 3, using **Ctrl**+click. The following display shows a comparison between the lookup table filled with two different extrapolations.

History for New_2D_Table

Version	Comment / Action	Date and Time
3	Values filled from data set meas_tq_data, factor tqmeas	04-Oct-2019 17:03:37
2	Values filled from data set meas_tq_data, factor tqmeas	04-Oct-2019 16:58:58
1	Initial configuration	04-Oct-2019 16:16:50

Reset / Add... / Remove / Edit... / Clear / ☐ Reset normalizer

L \ N	500	1000	1500	2000	2500	3000	3500
0.1	-0.546	-0.469	-0.819	-2.981	-8.146	-15.176	-22.286
0.2	-0.11	0.097	0.38	-0.627	-4.705	-11.148	-19.033
0.3	-0.024	-0.023	0.734	0.93	-0.93	-4.744	-11.835
0.4	0.126	-0.302	-0.126	0.341	0.083	0.076	-1.831
0.5	0.307	0.01	0.401	0.203	-0.651	-1.355	-0.751
0.6	-0.138	0.162	1.424	1.255	-0.513	-0.358	-0.55
0.7	0.013	0.356	2.564	3.815	0.021	-2.374	-3.074
0.8	-0.185	-1.112	-0.22	-0.597	0.347	1.168	0.749
0.9	-4.612	-5.869	-4.211	-1.742	-0.536	-1.946	-4.051
1	-9.123	-9.353	-6.858	-3.88	-4.057	-7.952	-11.229

Close / Help

5 Click **Close** to close the History viewer.

6 Select **Data** > **Plot** > **Surface** to view the surface again.

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

You have filled a lookup table with values taken from experimental data.

Export the Calibration

To export the calibration:

1 To highlight the lookup table that you want to export, click **Lookup Tables**.

Lookup Tables

2 Highlight the `New_2D_Table`.

3 Select **File** > **Export** > **Calibration** > **Selected Item**.

4 Choose the type of file you want to save your calibrations as. For the purposes of this tutorial, select `Comma Separated Value (.csv)`.

5 Enter `table_filling_tutorial.csv` as the file name and click **Save**.

See Also

More About

- "Manipulate Models in Data Set View"
- "Export Data Sets"

Optimization and Automated Tradeoff

Optimization and Automated Tradeoff

Import Models to Optimize

To open the CAGE browser and set up your optimization:

1   Start the CAGE Browser part of the Model-Based Calibration Toolbox product by typing

    cage

2   If you have not imported models, click **Import Models**. After you have imported models, you can optimize them.

    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:

    a   Import a model or models.

    b   Set up a new optimization.

3   If you have imported models, click **Optimization** in the **Processes** pane.

You can use the **Optimization** view to set up, run, view, and export optimizations. You must also set up optimizations here in order to use them for automated tradeoff.

When you first open the **Optimization** view both panes are blank until you create an optimization. After you set up your optimizations, the left **Optimization** pane shows a tree hierarchy of your optimizations, and the right hand panes display details of the optimization selected in the tree, as with other CAGE processes.

The following tutorial guides you through this process to evaluate this optimization problem: MaxTQ (SPK, N, L).

That is, find the maximum of the torque model (TQ) as a function of spark (SPK), engine speed (N), and load (L). You will use the NOXFLOW model to constrain these optimization problems.

For any optimization, you need to load one or more models. You can use the CAGE Import tool to import models from Model Browser projects. For this tutorial you can load a CAGE project from the mbctraining directory that contains two models for the optimization problems. Load the project containing models to optimize as follows:

1   Select **File** > **Open Project** to choose the tradeoffInit.cag file, found in the matlab \toolbox\mbc\mbctraining directory, then click **OK**.

    The tradeoffInit.cag project contains two models and all the variables necessary for this tutorial.

2   CAGE displays the Models view. You can view your models at any time by clicking the **Models** button in the **Data Objects** pane.

Observe that the project you have opened contains two models: `TQ_Model` and `NOXFLOW_Model`. In this tutorial you use these models to optimize torque values subject to emissions constraints.

3   To view the items in the Variable Dictionary, click the **Variable Dictionary** button in the **Data Objects** pane.
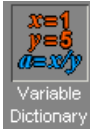
The **Variable Dictionary** view appears, displaying the variables, constants, and formulas in the current project. The project already has the relevant variables defined, so you do not need to import a variable dictionary. Note that the variables have ranges and set points defined.

See Also

More About

- "Calibration Setup"
- "Import Models and Calibration Items Using CAGE Import Tool"

Model-Based Calibration Toolbox Examples

Loading and Modifying Data

This example shows how to load and modify data using Model-Based Calibration Toolbox™ command-line interface. Data can be loaded from files (Excel® files, MATLAB® files, text files) and from the MATLAB® workspace. You can define new variables, apply filters to remove unwanted data, and apply test notes to filtered tests.

Load Data from Excel®

Load data from holliday.xlsx.

```
dataFile = fullfile( matlabroot, 'toolbox',...
    'mbc', 'mbctraining', 'holliday_data.mat' );
data = mbcmodel.CreateData( dataFile );
get( data )
data.SignalNames
```

```
            Name: 'holliday_data'
      NumRecords: 270
      NumSignals: 7
        NumTests: 27
   RecordsPerTest: [1x27 double]
      IsEditable: 1
    IsBeingEdited: 0
           Owner: []
      SignalNames: [7x1 string]
      SignalUnits: [7x1 string]
          Filters: [0x0 struct]
      TestFilters: [0x0 struct]
    UserVariables: [0x0 struct]


ans =

  7x1 string array

    "afr"
    "egr"
    "load"
    "n"
    "spark"
    "logno"
    "tq"
```

Plot Data

You can use the SignalName as an input to the Value method. Plot the first 5 tests.

```
x = zeros(10,5);
y = zeros(10,5);
name = cell(1,5);
% Collect the data as columns to pass to plot.
for tn = 1:5
    x(:,tn) = data.Value( 'spark', tn );
    y(:,tn) = data.Value( 'tq', tn );
    name{tn} = sprintf( 'Test %d', tn );
```

```
end
plot( x, y, 'x-' );
legend( name );
grid on
xlabel( sprintf( '%s [%s]', 'spark', data.SignalUnits{5} ) );
ylabel( sprintf( '%s [%s]', 'tq', data.SignalUnits{5} ) );
title( 'tq vs. spark' );
```
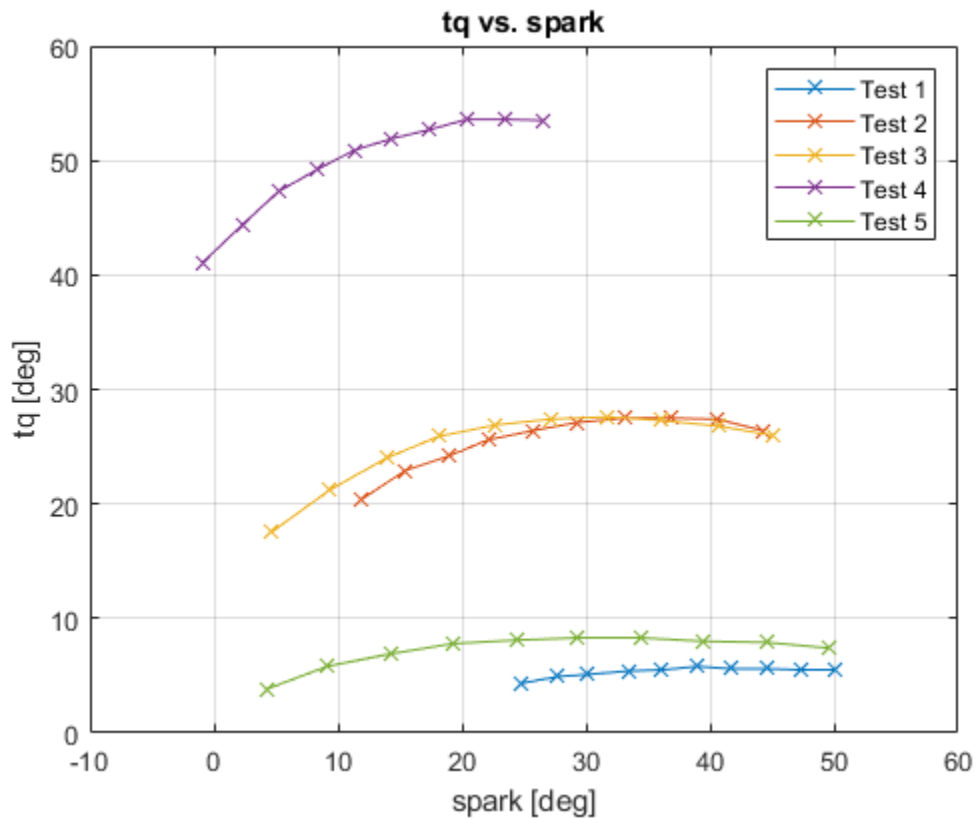

Remove Outliers and Problem Tests

Add a filter to keep tests where the mean torque is greater than 10. A filter is a constraint on the data set you can use to exclude some tests (test filter) or records (filter). You must call BeginEdit before making changes. The data is only updated when you call CommitEdit.

```
numberOfTestsBefore = data.NumberOfTests;
data.BeginEdit;
data.AddTestFilter( 'mean(tq)>10' );
data.CommitEdit;
numberOfTestsAfter = data.NumberOfTests;
fprintf( 'Removed %d tests.\n', numberOfTestsBefore-numberOfTestsAfter );
```

```
Removed 9 tests.
```

Add New Variable

You can add new variables to the data set.

```
data.BeginEdit;
data.AddVariable( 'POWER=tq*n' );
```

```
data.CommitEdit;
signalNamesBefore = data.SignalNames
% POWER is now in the list of SignalNames, and can be used to define other
% new variables.
data.BeginEdit;
data.AddVariable( 'POWER_SQUARED=POWER^2' );
data.CommitEdit;
signalNamesAfter = data.SignalNames
```

```
signalNamesBefore =

  8x1 string array

    "afr"
    "egr"
    "load"
    "n"
    "spark"
    "logno"
    "tq"
    "POWER"


signalNamesAfter =

  9x1 string array

    "afr"
    "egr"
    "load"
    "n"
    "spark"
    "logno"
    "tq"
    "POWER"
    "POWER_SQUARED"
```

Apply a Filter

Add a filter to keep only records where speed is greater than 1000.

```
numberOfRecordsBefore = data.NumberOfRecords;
data.BeginEdit;
data.AddFilter( 'n>1000' );
data.CommitEdit;
numberOfRecordsAfter = data.NumberOfRecords;
fprintf( 'Removed %d records.\n', numberOfRecordsBefore-numberOfRecordsAfter );
```

```
Removed 38 records.
```

Gasoline Case Study Design of Experiment

This example shows how to design an experiment for the gasoline case study problem using the command-line interface to Model-Based Calibration Toolbox™. The gasoline case study describes how to systematically develop a set of optimal steady-state engine calibration tables using Model-Based Calibration Toolbox™ software.

Create Project and Test Plan

```
project = mbcmodel.CreateProject('GasolineCaseStudy');
% Define Inputs for test plan
localInputs = mbcmodel.modelinput(...
    'Symbol', 'S',...
    'Name', 'SPARK',...
    'Range', [0 50]);
globalInputs = mbcmodel.modelinput(...
    'Symbol', {'N','L','ICP','ECP'},...
    'Name',   {'SPEED','LOAD','INT_ADV','EXH_RET'},...
    'Range',  {[500 6000],[0.0679 0.9502],[-5 50],[-5 50]});
% Create test plan
TP = CreateTestplan( project, {localInputs,globalInputs} );
```

Create Space-Filling Design

`CreateDesign` defaults to creating a design for the outer (global) level.

```
sfDesign = CreateDesign(TP, ...
    'Type', 'Latin Hypercube Sampling',...
    'Name', 'Space Filling');
```

Add Boundary Constraints

Load boundary constraints from another project file and add to design.

```
projectFile = [matlabroot,'\toolbox\mbc\mbctraining\Gasoline_project.mat'];
otherProject = mbcmodel.LoadProject( projectFile );
boundaryConstraints = otherProject.Testplans(1).BoundaryModel('global');
% Design constraints are specified as an array of
% mbcdoe.designconstraint objects.
sfDesign.Constraints = boundaryConstraints;
```

Change Selection Criteria of LHS Design

Get the design properties and change the SelectionCriteria to 'minimax'. Putting the changed properties back into the design causes the design to update.

```
designGenerator = sfDesign.Generator;
% Use "minimax"
designGenerator.SelectionCriteria = 'minimax'
sfDesign.Generator = designGenerator;


designGenerator =

Latin Hypercube Sampling design generator
```

Get Required Number of Points for a Constrained Design.

As in the Design Editor, when a design has constraints it is hard to achieve the number of points you require. In the design editor you try different numbers of points and regenerate. At the command line you can use the `ConstrainedGenerate` method to do this.

Generate Design

Use `ConstrainedGenerate` to make a 200 point design.

```
sfDesign = ConstrainedGenerate( sfDesign, 200, ...
    'UnconstrainedSize', 800, ...
    'MaxIter',10  );

% How did we do?
finalNumberOfPoints = sfDesign.NumberOfPoints
% How many points did we need in total?
totalNumberOfPoints = sfDesign.Generator.NumberOfPoints


finalNumberOfPoints =

    203


totalNumberOfPoints =

    852
```

Generate Design for Parked Cam Phasers

Make another design for some points with parked cam phasers. These points are important because you need an accurate model when cams are parked.

```
parkedCamsDesign = sfDesign.CreateDesign( 'Name', 'Parked' );
parkedCamsDesign = ConstrainedGenerate( parkedCamsDesign, 10, ...
    'UnconstrainedSize', 40, ...
    'MaxIter',10  );

% Explicitly set ECP and ICP to 0 - this changes the 'Type' to 'Custom'
designTypeBefore = parkedCamsDesign.Type
parkedCamsDesign.Points(:,3:4) = 0;
designTypeAfter = parkedCamsDesign.Type
% Merge with first design to create a new design
mainDesign = Merge( sfDesign, parkedCamsDesign );
mainDesign.Name = 'Main Design';


designTypeBefore =

    'Latin Hypercube Sampling'


designTypeAfter =

    'Custom'
```

Plot Design Points

Scatter2D is a method of mbcdoe.design.

```
designPoints = mainDesign.Points;
inputs = mainDesign.Inputs;
subplot(2,1,1)
Scatter2D( mainDesign, 1, 2 );
subplot(2,1,2)
Scatter2D( mainDesign, 3, 4 );
```
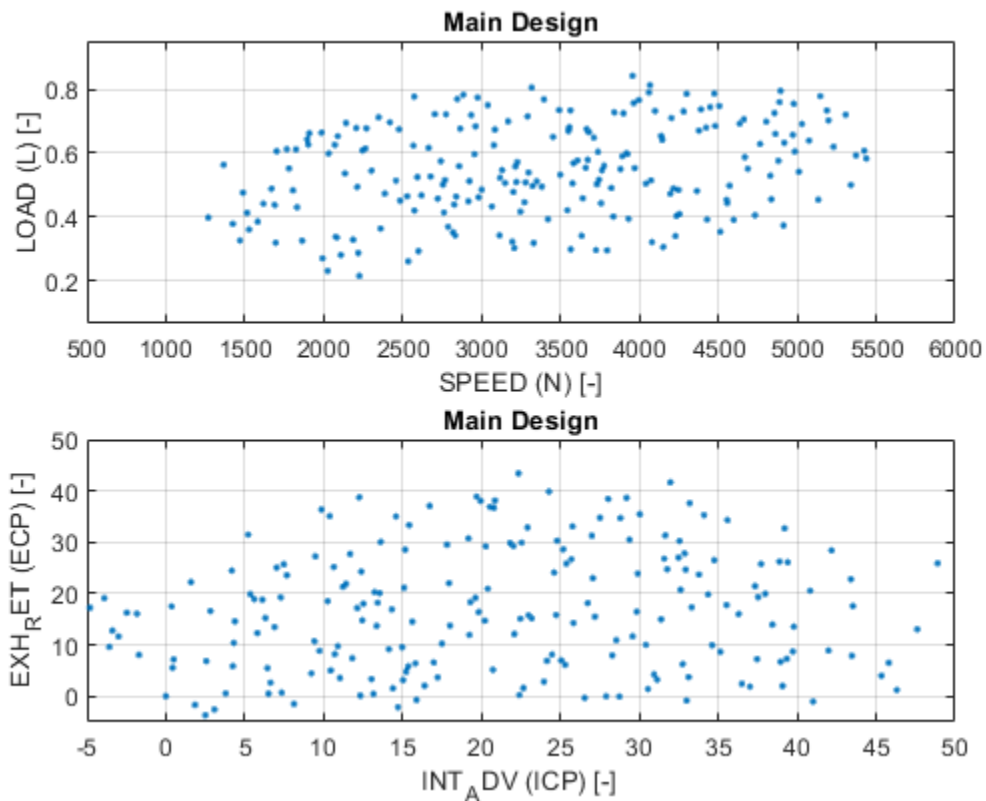

Add the Designs to the Test Plan

Testplans have a Design property that is a cell array (one cell for each stage). Here you add the designs to the 2nd stage.

```
TP.Design{2} = [sfDesign parkedCamsDesign mainDesign];
get(TP)

                 Data: []
               Levels: 2
      InputSignalNames: [5x1 string]
        InputsPerLevel: [1 4]
             Responses: [0x0 mbcmodel.abstractnode]
              Boundary: []
     SummaryStatistics: [0x0 table]
                Inputs: {[1x1 mbcmodel.modelinput]  [4x1 mbcmodel.modelinput]}
               Designs: {[1x0 mbcdoe.design]  [1x3 mbcdoe.design]}
```

```
              BestDesign: {[]  []}
           DefaultModels: {[1x1 mbcmodel.localmodel]  [1x1 mbcmodel.linearmodel]}
          ValidationData: [0x0 mbcmodel.data]
           ResponseNames: {1x0 cell}
                    Name: 'Two-Stage'
                 Project: [1x1 mbcmodel.project]
```

Set the BestDesign

Testplans also have a BestDesign property (also a cell array). Set the Best Design for the 2nd stage.

```
TP.BestDesign{2} = mainDesign


TP =

  testplan with properties:

                  Data: []
                Levels: 2
       InputSignalNames: [5x1 string]
          InputsPerLevel: [1 4]
              Responses: [0x0 mbcmodel.abstractnode]
               Boundary: []
       SummaryStatistics: [0x0 table]
                 Inputs: {[1x1 mbcmodel.modelinput]  [4x1 mbcmodel.modelinput]}
                Designs: {[1x0 mbcdoe.design]  [1x3 mbcdoe.design]}
             BestDesign: {[]  [1x1 mbcdoe.design]}
          DefaultModels: {[1x1 mbcmodel.localmodel]  [1x1 mbcmodel.linearmodel]}
         ValidationData: [0x0 mbcmodel.data]
          ResponseNames: {1x0 cell}
                   Name: 'Two-Stage'
                Project: [1x1 mbcmodel.project]
```

Create Validation Design

Make another space-filling design for collecting validation data.

```
validationDesign = sfDesign.CreateDesign( 'Name', 'Validation' );
validationDesign = ConstrainedGenerate( validationDesign, 25,...
    'UnconstrainedSize', 100, ...
    'MaxIter',10  );

% Add the parked cams point.
validationDesign.Points(end+1,:) = [3500 0.5 0 0];
% Add this to testplan as well - when you add one design only, using
% AddDesign is more convenient. By default this adds the design to 2nd
% stage.
% Note: alternatively, you could add the design to TP.Design{2} directly.
TP.AddDesign(validationDesign);
% The list of designs for the 2nd stage
allDesigns = TP.Design{2}


allDesigns =

  1x4 design array with properties:
```

```
Style
Type
NumPoints
NumInputs
Name
Points
PointTypes
Inputs
Generator
Constraints
Model
```

Using Constraints

This example shows how to create and apply constraints to a design using Model-Based Calibration Toolbox™ command-line interface.

Create a Design

Create a Full Factorial design, because this will show the constraint boundaries as clearly as possible. Create the design from inputs. For simplicity, only 2 inputs (speed and load) are used.

```
inputs = mbcmodel.modelinput(...
    'Symbol', {'N','L'},...
    'Name',   {'SPEED','LOAD'},...
    'Range',  {[500 6000],[0.0679 0.9502]});

design = CreateDesign( inputs, 'Type', 'Full Factorial' );
design = Generate( design, 'NumberOfLevels', [50 50] );
% design has a Constraints property, initially this is empty.
constraints = design.Constraints


constraints =
```

Create a Linear Constraint

```
cLinear = CreateConstraint( design, 'Type', 'Linear' );
cLinear.A = [-2.5e-4, 1];
cLinear.b = 0.25;
cLinear
design.Constraints = cLinear;
design = Generate(design);


cLinear =

Linear design constraint: -0.00025*N + 1*L <= 0.25
```

Show Constraint

Plot the points to show the linear constraint.

```
Scatter2D(design, 1, 2);
title( 'Linear Constraint' );
```

Linear Constraint
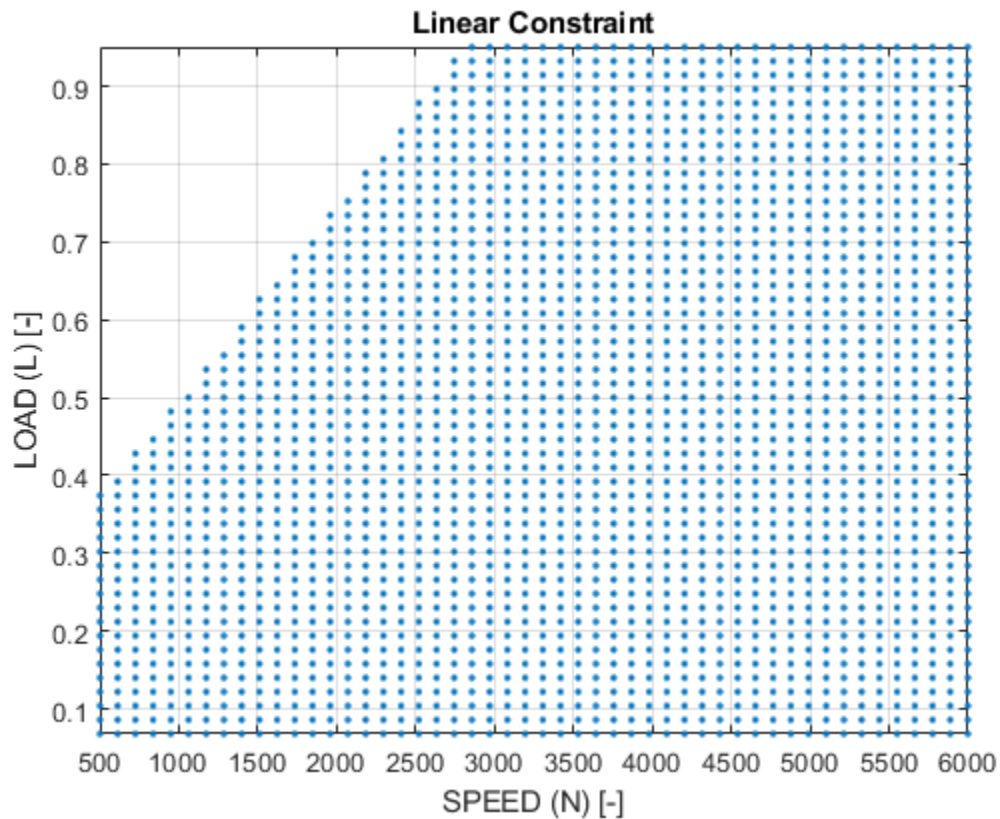
Create a 1D Table Constraint

```
cTable1d = CreateConstraint( design, 'Type', '1D Table' );
cTable1d.Table = [0.9 0.5];
cTable1d.Breakpoints = [500 6000];
cTable1d
design.Constraints = cTable1d;
design = Generate(design);
```

```
cTable1d =

1D Table design constraint: L(N) <= Lmax
```

Show Constraint

Plot the points to show the 1D Table constraint.

```
Scatter2D( design, 1, 2 );
title( '1D Table Constraint ' );
```

1D Table Constraint
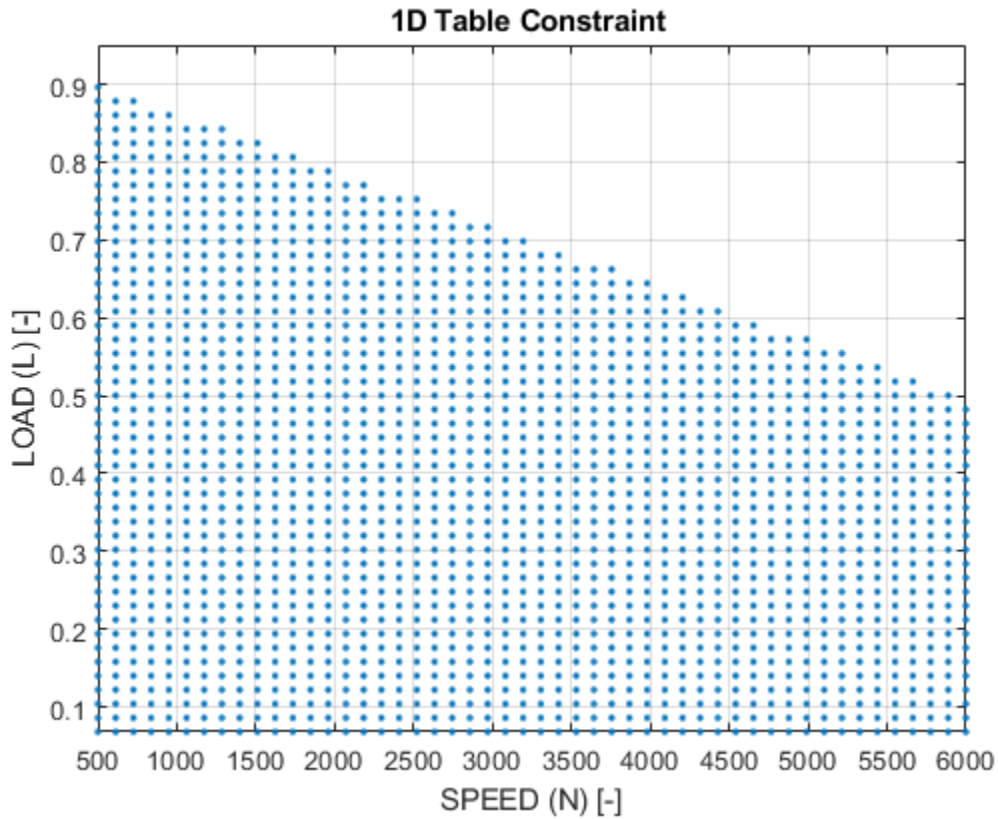
Combining Constraints

Constraints is an array of the constraints to apply.

```
design.Constraints = [cLinear, cTable1d];
constraints = design.Constraints
design = Generate(design);
```

```
constraints =
```

```
Linear design constraint: -0.00025*N + 1*L <= 0.25
1D Table design constraint: L(N) <= Lmax
```

Show Constraint

Plot the points to show both constraints.

```
Scatter2D( design, 1, 2 );
title( 'Linear and 1D Table Constraint ' );
```

Linear and 1D Table Constraint
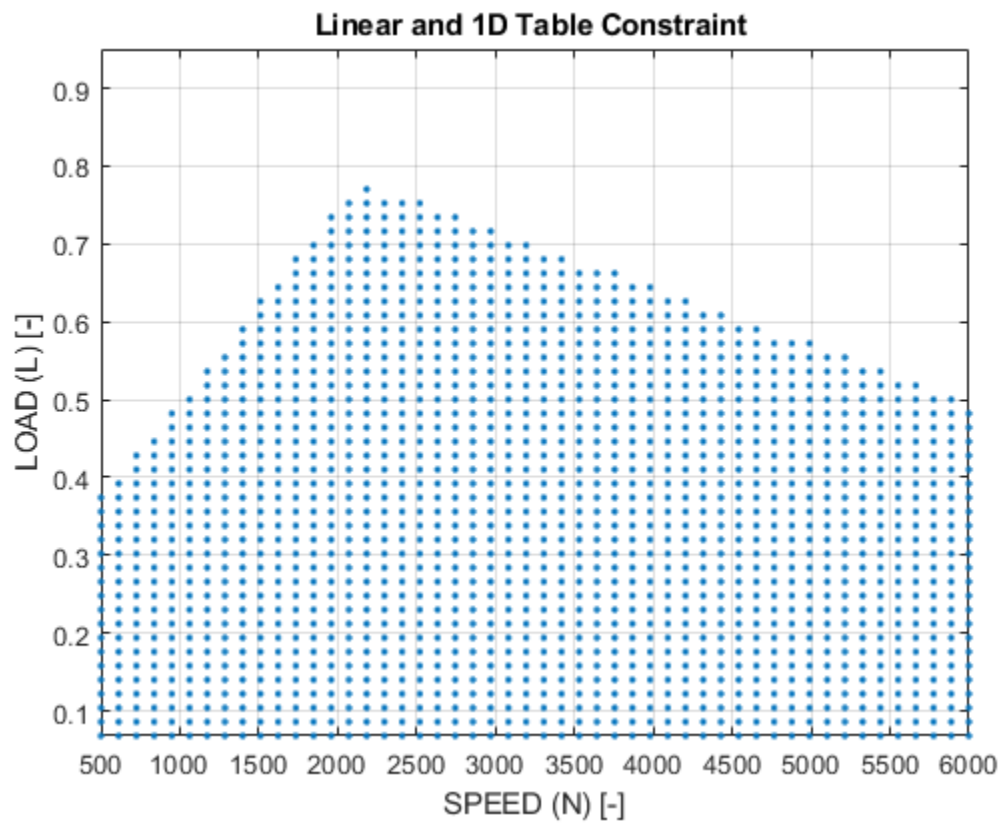
Local Designs

This example shows how to use the command-line functionality to generate local designs at each global operating point. This particular example shows how you can produce local maps for a diesel engine calibration.

Create Project and Test Plan

Speed (N) and fuel (F) are global inputs. Injection (soi), fuel pressure (fuelpress), variable geometry turbo rack position (grackmea) and exhaust gas recirculation (EGR) are local inputs.

```
project = mbcmodel.CreateProject('DieselMulti');

% Define Inputs for test plan
LocalInputs = mbcmodel.modelinput('Symbol',{'S','P','G','E'},...
    'Name',{'soi','fuelpress','grackmea','egrlift'},...
    'Units',{'deg','MPa','ratio','mm'},...
    'Range',{[-9 3],[60 160],[0.2 0.9],[0.5 5]});
GlobalInputs = mbcmodel.modelinput('Symbol',{'N','F'},...
    'Name',{'measrpm','basefuelmass'},...
    'Units',{'rpm','mg/stroke'},...
    'Range',{[1600 2200],[20 200]});
% create test plan
TP = CreateTestplan( project, {LocalInputs,GlobalInputs} );
```

Global Design

Generate a 15 point Latin Hypercube Sampling (LHS) design for the global inputs.

```
globalDesign = TP.CreateDesign(2, 'Type', 'Latin Hypercube Sampling');
% Fuel constraint:  Maximum 200 at 1600 rpm, 175 at 2200 rpm
C = globalDesign.CreateConstraint('Type','1D Table');
% set up the 1D Table constraint
C.InputFactor = 'N';
C.Breakpoints = [1600 2000];
C.TableFactor = 'F';
C.Table = [200 175];
% assign constraint to design
globalDesign.Constraints = C;
% generate a 15 point design
globalDesign = Generate(globalDesign, 15);
% set as best design in test plan
TP.BestDesign{2} = globalDesign;
```

Create a Local Design for Each Global Point

For each global point, adjust limits for fuel pressure and grackmea, and generate a 30 point LHS design.

```
% create a local design
localDesign = TP.CreateDesign(1,'Type','Latin Hypercube Sampling');
localDesignGenerator = localDesign.Generator;
localDesignGenerator.NumberOfPoints = 30;
DList = mbcdoe.design.empty( 0, 1);
for i = 1:globalDesign.NumberOfPoints;
    GlobalPoint = globalDesign.Points(i,:);
    speed = GlobalPoint(1);
```

```
    % fuel pressure limits dependent on speed
    f = (speed-1600)/(2200-1600);
    % note because you use the Limits property to specify the input range
    % you get LHS designs with exactly 30 points.
    localDesignGenerator.Limits(2,:) = (1-f)*[90 120] + f*[110 160];
    % grackmea limits dependent on speed
    localDesignGenerator.Limits(3,:) = (1-f)*[0.2 0.4] + f*[0.6 0.9];

    % set design properties and generate local design
    localDesign.Generator = localDesignGenerator;

    % Make design name which reflects the global point
    localDesign.Name = sprintf('Test %2d (%s=%4.0f,%s=%3.0f)', i,...
        GlobalInputs(1).Symbol,GlobalPoint(1),....
        GlobalInputs(2).Symbol,GlobalPoint(2));

    % Plot Design
    Scatter2D(localDesign);

    DList(i) = localDesign;
end

% assign list of local designs to test plan
TP.Designs{1} = DList;
% List of local designs
localDesigns = TP.Designs{1}


localDesigns =

  1x14 design array with properties:

    Style
    Type
    NumPoints
    NumInputs
    Name
    Points
    PointTypes
    Inputs
    Generator
    Constraints
    Model
```
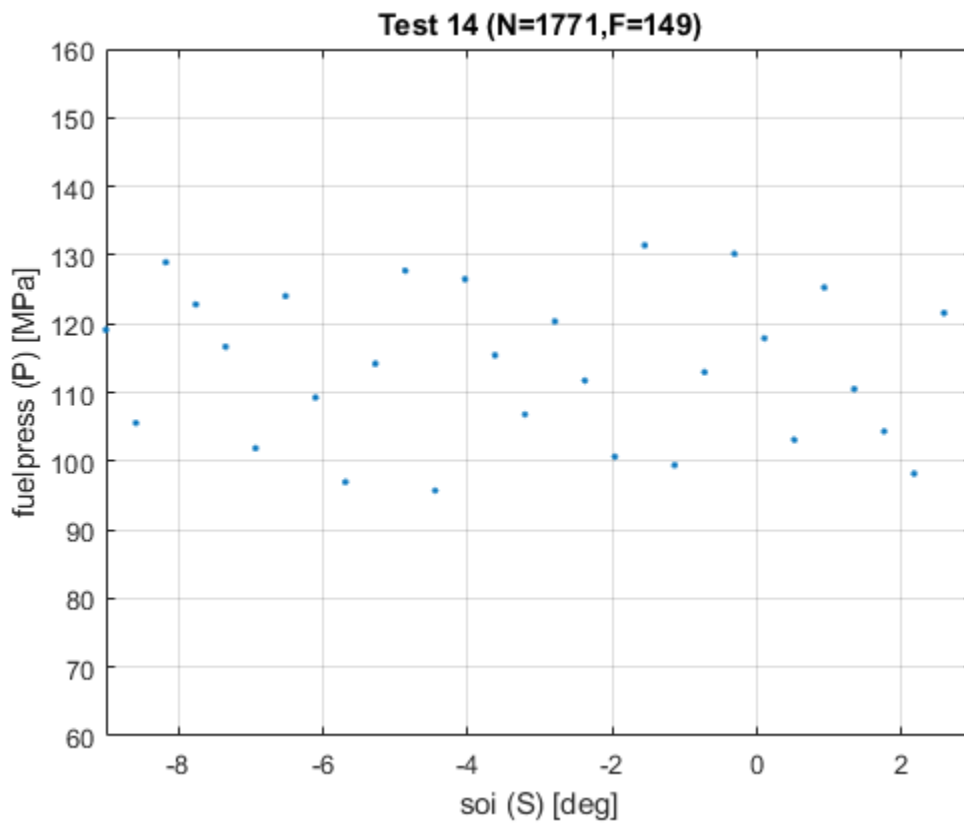
Optimal Designs

This example shows how to create an optimal design for a polynomial model using Model-Based Calibration Toolbox™ command-line interface.

Create the Model

You need a model to create an optimal design.

```
inputs = mbcmodel.modelinput(...
    'Symbol', {'N','L','A'},...
    'Name',   {'n','load','afr'},...
    'Range',  {[1000 5000],[0.2 0.65],[10.9 14.65]});

model = mbcmodel.CreateModel( 'Polynomial', inputs );
model.Properties.Order = [2 2 2];
```

Create a V-optimal Design

Create a design for the polynomial model.

```
optimalDesign = CreateDesign( model,...
    'Type', 'V-optimal',...
    'Name', 'Optimal Design' );
```

Generate Design

Create a CandidateSet to use.

```
CandidateSet = optimalDesign.CreateCandidateSet( 'Type', 'Grid' );
CandidateSet.NumberOfLevels = [21 21 21];
% Pass in Generator properties to Generate
optimalDesign = Generate( optimalDesign,...
    'NumberOfPoints', 30,...
    'CandidateSet', CandidateSet,...
    'MaxIterations', 200,...
    'NoImprovement', 50 );
```

Alternative Way to Generate Design

Instead of passing generator settings to `Generate`, you can modify the design's generator. The two methods are equivalent, because assigning the generator back to the design causes a call to `Generate`.

```
anotherOptimalDesign = CreateDesign( model, 'Type', 'V-optimal', ...
    'Name', 'Another Optimal Design' );
optimalGenerator = anotherOptimalDesign.Generator;
optimalGenerator.NumberOfPoints = 30;
optimalGenerator.CandidateSet.Type = 'Grid';
optimalGenerator.CandidateSet.NumberOfLevels = [21 21 21];
optimalGenerator.MaxIterations = 200;
optimalGenerator.NoImprovement = 50;
```

Setting the Generator causes `Generate` to be called.

```
anotherOptimalDesign.Generator = optimalGenerator;
```

Optimal Criteria

```
criteria = OptimalCriteria( optimalDesign );
fprintf( ['Optimality Criteria for "%s":\n',...
    'V = %.3f\nD = %.3f\nA = %.3f\nG = %.3f\n'],...
    optimalDesign.Name, criteria );
```

```
Optimality Criteria for "Optimal Design":
V = 0.181
D = 2.492
A = 1.050
G = 0.552
```

Pairwise Plots Of Design

Plot each input against the others.

```
subplot(2,2,1);
Scatter2D( optimalDesign, 1, 2 );

subplot(2,2,3);
Scatter2D( optimalDesign, 1, 3 );

subplot(2,2,4);
Scatter2D( optimalDesign, 2, 3 );
```
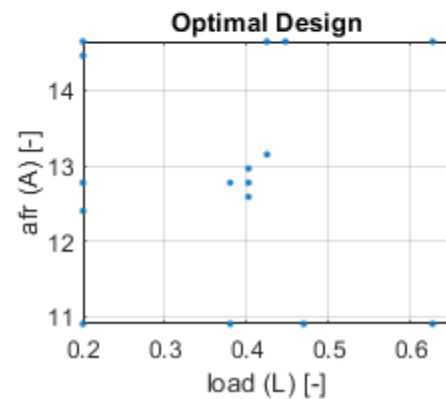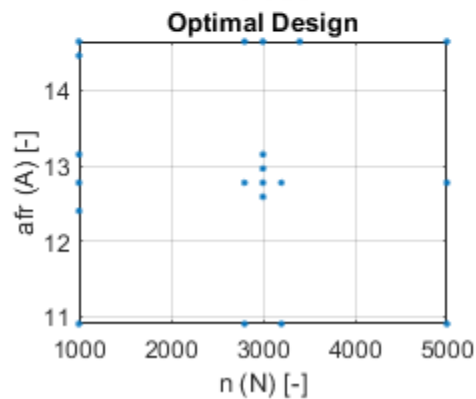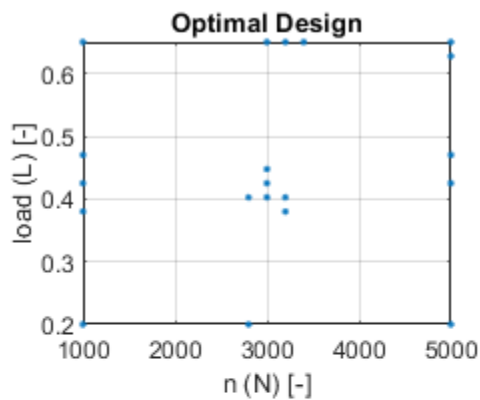
Gasoline Case Study

This example shows how to automatically generate an mbcmodel project for the gasoline case study using the command-line tools in Model-Based Calibration Toolbox™ .

Requires DIVCP_Main_DoE_Data.xls from mbctraining folder.

Create a New mbcmodel Project

```
project = mbcmodel.CreateProject;
```

Load Data into Project

- Group data into tests and add some filters.
- Add filters to remove bad data.
- Remove tests which do not have sufficient data to fit local models.

```
datafile = fullfile( mbcpath, 'mbctraining', 'DIVCP_Main_DoE_Data.mat' );
data = CreateData( project, datafile );

BeginEdit( data );

% Group data by test number GDOECT.
DefineTestGroups( data, 'GDOECT', 0.5, 'GDOECT', false );
% Get rid of data which is probably unstable.
AddFilter( data, 'RESIDFRAC < 35' );
AddFilter( data, 'AFR > 14.25' );
% Get rid of the tests that are too small.
AddTestFilter( data, 'length(BTQ) > 4' );

% Commit the changes to the project.
CommitEdit( data );
```

Build Test Plan

Define Inputs for test plan.

```
LocalInputs = mbcmodel.modelinput('Symbol','S',...
    'Name','SPARK',...
    'Range',[0 50]);
GlobalInputs = mbcmodel.modelinput('Symbol',{'N','L','ICP','ECP'},...
    'Name',{'SPEED','LOAD','INT_ADV','EXH_RET'},...
    'Range',{[500 6000],[0.0679    0.9502],[-5 50],[-5 50]});
% Create test plan.
testplan = CreateTestplan( project, {LocalInputs,GlobalInputs} );
% Attach data to the test plan.
AttachData( testplan, data );
```

Build Boundary Models

Create a global boundary model. CreateBoundary does not add the boundary model to the tree.

```
B = CreateBoundary(testplan.Boundary.Global,'Star-shaped');
% Add boundary model to the test plan. The boundary model is fitted when it
% is added to the boundary model tree. The boundary model is included in
% the best boundary model for the tree by default.
% All inputs are used in the boundary model by default.
```

```
B = Add(testplan.Boundary.Global,B);

% Now make a boundary model using only speed and load and add to the
% boundary tree.
B.ActiveInputs = [true true false false];
B = Add(testplan.Boundary.Global,B);
% Look at the global boundary tree.
testplan.Boundary.Global


ans =

  Tree with properties:

         Data: [189x4 double]
       Models: {[1x1 mbcboundary.Model]  [1x1 mbcboundary.Model]}
    BestModel: [1x1 mbcboundary.Boolean]
       InBest: [1 1]
     TestPlan: [1x1 mbcmodel.testplan]
```

Build Responses

Build response models for torque, exhaust temperature and residual fraction * Use a local polynomial spline model for torque. * Use a local polynomial model with datum for exhaust temperature and residual fraction

```
LocalTorque  = mbcmodel.CreateModel('Local Polynomial Spline',1);
LocalTorque.Properties.LowOrder = 2;
% Use the default global model.
GlobalModel = testplan.DefaultModels{2};
CreateResponse(testplan,'BTQ',LocalTorque,GlobalModel,'Maximum');
% make exhaust temperature and residual fraction models
LocalPoly  = mbcmodel.CreateModel('Local Polynomial with Datum',1);
CreateResponse(testplan,'EXTEMP',LocalPoly,GlobalModel,'Linked');
CreateResponse(testplan,'RESIDFRAC',LocalPoly,GlobalModel,'Linked');
```

Remove Local Outliers

Remove data if abs(studentized residuals) > 3. Note that a different process was used in the project Gasoline_project to decide which outliers to remove.

```
TQ_response = testplan.Responses(1);
numTests = TQ_response.NumberOfTests;
LocalBTQ = TQ_response.LocalResponses;
for tn = 1:numTests
    % Find observations with studentized residuals greater than 3
    studentRes = DiagnosticStatistics( LocalBTQ, tn, 'Studentized residuals' );
    potentialOut  = abs( studentRes )> 3;
    if any(potentialOut)
        % Don't update response feature models until end of loop
        RemoveOutliersForTest( LocalBTQ, tn, potentialOut , false);
    end
    % get local model for test and look at summary statistics
    mdl = ModelForTest(LocalBTQ,tn);
    if ~strcmp(mdl.Status,'Not fitted')
        LocalStats = SummaryStatistics(mdl);
    end
end
```

Update response features.

```
UpdateResponseFeatures(LocalBTQ);
```

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 12).

Remove Points Where MBT<0 or MBT>60

```
knot = LocalBTQ.ResponseFeatures(1);
PointsToRemove = knot.DoubleResponseData<0 | knot.DoubleResponseData>60;
knot.RemoveOutliers(PointsToRemove);
```

Create Alternative Response Feature Models

Make a list of candidate models and select the best based on AICc.

- Quadratic
- Cubic
- RBF with a range of centers
- Polynomial-RBF with a range of centers

Get the base model. You use this to create the other models.

```
rf = LocalBTQ.ResponseFeatures(1);
BaseModel = rf(1).Model;
```

Make a quadratic model that uses Minimize PRESS to fit, and add it to the list.

```
m = BaseModel.CreateModel('Polynomial');
m.Properties.Order = [2 2 2 2];
m.FitAlgorithm = 'Minimize PRESS';
mlist = {m};
```

Make a cubic model and add it to the list.

```
m.Properties.Order = [3 3 3 3];
m.Properties.InteractionOrder = 2;
mlist{2} = m;
```

Make RBF models with a range of centers. The maximum number of centers is set in the center selection algorithm.

```
m = BaseModel.CreateModel('RBF');
Centers = [50 80];
Start = length(mlist);
mlist = [mlist cell(size(Centers))];
for i = 1:length(Centers)
    fitAlgorithm = m.FitAlgorithm.WidthAlgorithm.NestedFitAlgorithm;
    fitAlgorithm.CenterSelectionAlg.MaxCenters = Centers(i);
    m.FitAlgorithm.WidthAlgorithm.NestedFitAlgorithm = fitAlgorithm;
    mlist{Start+i} = m;
end
```

Make Polynomial-RBF models with a range of centers.

```
m = BaseModel.CreateModel('Polynomial-RBF');
m.Properties.Order = [2 2 2 2];
```

```
Start = length(mlist);
mlist = [mlist cell(size(Centers))];
for i = 1:length(Centers)
    % Maximum number of centers is set in the nested fit algorithm
    m.FitAlgorithm.WidthAlgorithm.NestedFitAlgorithm.MaxCenters = Centers(i);
    mlist{Start+i} = m;
end
```

Make alternative models for each response feature and select the best model based on AICc.

```
criteria = 'AICc';
CreateAlternativeModels( LocalBTQ, mlist, criteria );
```

Alter Response Feature Models

Get the alternative responses for knot and alter models using stepwise regression.

```
knot = LocalBTQ.ResponseFeatures(1);
AltResponse = knot.AlternativeResponses(1);
```

Get the stepwise statistics.

```
knot_model = AltResponse.Model;
[stepwise_stats,knot_model] = StepwiseRegression(knot_model);
```

Use PRESS to find the best term to change, and toggle the stepwise status of the term with this index.

```
[bestPRESS, ind] = min(stepwise_stats(:,4));
[stepwise_stats,knot_model] = StepwiseRegression(knot_model, ind);
```

Get a VIF statistic.

```
VIF = MultipleVIF(knot_model)
```

```
VIF =

    3.1290
    1.2918
    1.6841
    1.1832
    1.3230
    2.6617
    1.6603
    1.3306
    1.2856
    1.4317
    2.6550
```

Get the RMSE.

```
RMSE = SummaryStatistics(knot_model, 'RMSE')
```

```
RMSE =

    5.1578
```

Change the model to a Polynomial-RBF with a maximum of 10 centers.

```
new_knot_model = knot_model.CreateModel('Polynomial-RBF');
new_knot_model.Properties.Order = [1 1 1 1];
new_knot_model.FitAlgorithm.WidthAlgorithm.NestedFitAlgorithm.MaxCenters = 10;
% Fit the model with current data.
[new_knot_model,S] = fit(new_knot_model);
```

If there were no problems with the changes then update the response, otherwise you will continue to use the original model.

```
if strcmp(new_knot_model.Status,'Fitted')
    new_RMSE = SummaryStatistics(new_knot_model,'RMSE')
    % Update the response with the new model.
    UpdateResponse(new_knot_model);
end
```

```
new_RMSE =

    3.5086
```

Make the Two-stage Model for Torque

```
doMLE = true;
MakeHierarchicalResponse( LocalBTQ, doMLE );
% Look at the Local and Two-Stage RMSE.
BTQ_RMSE = SummaryStatistics( LocalBTQ, {'Local RMSE', 'Two-Stage RMSE'} )
```

```
BTQ_RMSE =

    0.8319    4.6116
```

Plot the Two-stage Model of Torque Against Spark

Plot the 5th test

```
testToPlot = 5;
BTQInputData = TQ_response.DoubleInputData(testToPlot);
BTQResponseData = TQ_response.DoubleResponseData(testToPlot);
BTQPredictedValue = TQ_response.PredictedValue( BTQInputData );
fig = figure;
plot( BTQInputData(:,1), BTQResponseData, 'o',...
    BTQInputData(:,1), BTQPredictedValue, '-' );
xlabel( 'spark' );
ylabel( 'torque' );
title( 'Test 5' );
grid on
```
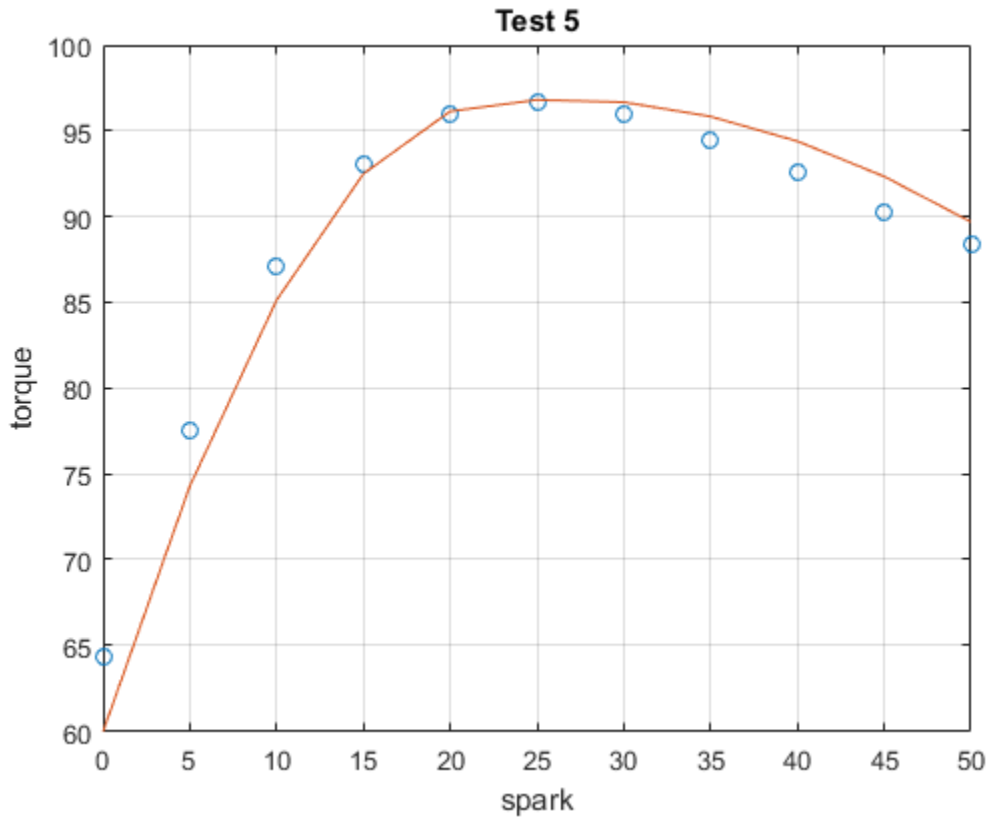
Build Other Responses

Build models for exhaust temperature and residual fraction.

- Copy outliers from torque model
- Make alternative models for each response feature
- Make two-stage model without MLE

EXTEMP Response

```
EXTEMP = testplan.Responses(2).LocalResponses(1);
EXTEMP.RemoveOutliers(OutlierIndices(LocalBTQ));
CreateAlternativeModels( EXTEMP,mlist, criteria );
MakeHierarchicalResponse( EXTEMP, false );
EXTEMP_RMSE = SummaryStatistics( EXTEMP, {'Local RMSE', 'Two-Stage RMSE'} )
```

```
EXTEMP_RMSE =

   10.5648   27.9931
```

RESIDFRAC Response

```
RESIDFRAC = testplan.Responses(3).LocalResponses(1);
RESIDFRAC.RemoveOutliers(OutlierIndices(LocalBTQ));
CreateAlternativeModels( RESIDFRAC,mlist, criteria );
```

```
ok = MakeHierarchicalResponse( RESIDFRAC, false );
RESIDFRAC_RMSE = SummaryStatistics( RESIDFRAC, {'Local RMSE', 'Two-Stage RMSE'} )

if isgraphics(fig)
    % delete figure made during example
    delete(fig)
end
```

```
RESIDFRAC_RMSE =

    0.0824    0.5595
```

Point-by-point Modeling for a Diesel Engine

This example shows how to use the Model-Based Calibration Toolbox™ command-line functionality for point-by-point engine modeling projects.

Multiple injection diesel engines and gasoline direct-injection engines can often only be modeled with point-by-point models. You can use point-by-point models to build a model at each operating point of an engine with the necessary accuracy to produce an optimal calibration. Point-by-point command-line functionality is necessary to handle the complexity of developing designs for each operating point.

Why are point-by-point models necessary? Engine actuators and sensors are continually being added to Engine Management Systems (EMS) in response to ever-increasing engine emissions, fuel economy, and performance requirements. In some cases, optimal engine calibration development processes that rely on two-stage modeling may no longer be able to model engine performance responses with sufficient accuracy across the engine operating range. Point-by-point models can provide the necessary model accuracy at measured operating points. However, point-by-point models do not provide estimated responses at other operating points.

This example uses the two-stage models generated in the diesel case study as a surrogate for an engine dynamometer or CAE engine model, in order to generate the point-by-point data for this example. The example shows you how to:

- Generate local designs at each operating point. If insufficient design points can be collected, you can augment the local design using Sobol sequences.
- Create local **multiple models** to model each of the responses at each operating point.
- Build a point-by-point boundary model to define the data boundary at each operating point for later use in calibration optimization.

Finally, you must visually inspect the fitted models to verify that the model quality is acceptable. You usually need to identify and remove outliers. You can use the command-line to plot diagnostics and remove outliers, but it is easier to use the graphical and statistical tools in the Model Browser (mbcmodel) provided in the Model-Based Calibration Toolbox.

Load Models from Diesel Project

This example uses engine data generated from the models in the diesel case study.

The inputs are MAINSOI, SPEED, BASEFUELMASS, FUELPRESS, VGTPOS, EGRPOS

```
DieselProject = mbcmodel.LoadProject(...
    fullfile(mbcpath,'mbctraining','Diesel_project.mat'));
% Store the models in a structure for convenience
DieselResponses = DieselProject.Testplans.Responses;
Models.BTQ = DieselResponses(1);
Models.VGTSPEED = DieselResponses(2);
Models.EQREXH = DieselResponses(3);
Models.PEAKPRESS = DieselResponses(4);
Models.NOX = DieselResponses(6);
Models.EGRMF = DieselResponses(7);
```

Define Inputs for Point-by-Point Models and Create Local Model

- The operating point variables are speed (SPEED) and brake torque (BTQ).

- The local inputs are main start of injection (MAINSOI), fuel pressure (FUELPRESS), variable gate turbo position (VGTPOS), and exhaust gas recirculation position (EGRPOS).

```
OperatingPointInputs = mbcmodel.modelinput('Symbol',{'SPEED','BTQ'},...
    'Name',{'SPEED','BTQ'},...
    'Units',{'rpm','Nm'},...
    'Range',{[1600 2200],[0 1600]});
LocalInputs = mbcmodel.modelinput(...
    'Symbol',{'MAINSOI','FUELPRESS','VGTPOS','EGRPOS'},...
    'Name',{'MAINSOI','FUELPRESS','VGTPOS','EGRPOS'},...
    'Units',{'deg','MPa','ratio','mm'},...
    'Range',{[-9 3],[90 160],[0.2 0.9],[0.5 5]});
% Create a local multiple model
L = mbcmodel.CreateModel('Point-by-Point',LocalInputs);
% Select the best model using the PRESS RMSE statistic.
L.Properties.SelectionStatistic = 'PRESS RMSE';
```

Define Engine Operating Points

The design for the operating points is a 7 point drive cycle.

```
Xg = [2200.0 1263.0
    2200.0    947.0
    2200.0    632.0
    2200.0    126.0
    1600.0   1550.0
    1600.0   1163.0
    1600.0    775.0];

OperatingPointDesign = CreateDesign(OperatingPointInputs);
OperatingPointDesign.Points = Xg;
OperatingPointDesign.Name = 'Drive cycle';
```

Create a Local Design for Each Operating Point

For each operating point, adjust limits for main injection, fuel pressure and VGTPOS, and generate a 128 point Sobol design.

```
localDesign = CreateDesign(LocalInputs,'Type','Sobol Sequence');
localDesignGenerator = localDesign.Generator;
NumLocalPoints = 128;
localDesignGenerator.NumberOfPoints = NumLocalPoints;
DList = mbcdoe.design.empty( 0, 1);

data = [];
for i = 1:OperatingPointDesign.NumberOfPoints

    OperatingPoint = OperatingPointDesign.Points(i,:);
    speed = OperatingPoint(1);
    TQDemand = OperatingPoint(2);
```

Name the Local Design by Operating Point

```
    localDesign.Name = sprintf('Test %2d (%s=%4.0f,%s=%3.0f)', i,...
        OperatingPointInputs(1).Symbol,OperatingPoint(1),....
        OperatingPointInputs(2).Symbol,OperatingPoint(2));
```

Set Up Limits Depending on Speed for Local Inputs

When you use the Limits property to specify the input range, you generate a Sobol design with exactly NumLocalPoints points.

```
f = (speed-1600)/(2200-1600);
% The main soi range varies from [-3,3] at 1600 rpm to
% [-9,-3] at 2200 rpm.
localDesignGenerator.Limits(1,:) = (1-f)*[-3,3] + f*[-9,-3];
% The fuel pressure range varies from [90,130] at 1600 rpm to
% [120,160] at 2200 rpm.
localDesignGenerator.Limits(2,:) = (1-f)*[90 120] + f*[110,160];
% The VGTPOS range varies from [0.2,0.4] at 1600 rpm to
% [0.6,0.9] at 2200 rpm.
localDesignGenerator.Limits(3,:) = (1-f)*[0.2 0.4] + f*[0.6 0.9];

% set design properties and generate local design
localDesign.Generator = localDesignGenerator;
```

Collect Engine Data for Local Design

Find the fuel required to obtain the demanded torque for each point in the local design

```
[Xlocal,XTS] = mbcSolveTQ(Models,localDesign.Points,speed,TQDemand);
```

Augment Designs if Necessary

Check that enough points have been collected after running the 128 point DOE (design of experiment). Collect extra points by augmenting the Sobol Sequence with the next N points in the sequence if necessary.

```
N =  NumLocalPoints;
count = 1;
while size(Xlocal,1) < NumLocalPoints*0.75 && count<10
    localDesign = Generate(localDesign,...
        'Skip',N,...
        'NumberOfPoints',N);
    N = N*2;
    % Find the fuel required to obtain the demanded torque for each point in
    % the augmented local design
    [Xlocalaug,XTSaug] = mbcSolveTQ(Models,localDesign.Points,speed,TQDemand);
    Xlocal = [Xlocal; Xlocalaug]; %#ok<AGROW>
    XTS = [XTS; XTSaug]; %#ok<AGROW>
end

% Update points in the local design
localDesign.Points = Xlocal;
fprintf('%s: %d points\n',localDesign.Name,localDesign.NumberOfPoints);
```

```
Test  1 (SPEED=2200,BTQ=1263): 127 points

Test  2 (SPEED=2200,BTQ=947): 128 points

Test  3 (SPEED=2200,BTQ=632): 128 points

Test  4 (SPEED=2200,BTQ=126): 128 points

Test  5 (SPEED=1600,BTQ=1550): 116 points

Test  6 (SPEED=1600,BTQ=1163): 119 points
```

```
Test  7 (SPEED=1600,BTQ=775): 128 points
```

Collect Response Data

Calculate response data from the diesel case study models

```
    BTQ = TQDemand*ones(size(Xlocal,1),1);
    AFR = 14.46./Models.EQREXH.PredictedValue(XTS);
    EGRMF = Models.EGRMF.PredictedValue(XTS);
    BSNOX = 3600*Models.NOX.PredictedValue(XTS)/159.5573;
    PEAKPRESS = Models.PEAKPRESS.PredictedValue(XTS)/1e6;
    VGTSPEED = Models.VGTSPEED.PredictedValue(XTS);
    BSFC = 5400.*XTS(:,3)./(BTQ*pi);
```

Check Fit for BSFC

Check RMSE < 2

```
    [stats,Lbsfc] = Fit(L,Xlocal,BSFC);
    if stats(5)>2
        fprintf('Poor fit for test %d.\n',i)
    end
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 12).
```

Accumulate Data and Local Designs

You can ensure tests are automatically defined by defining a variable 'logno'.

```
    data = [data ;
        Xlocal XTS(:,2:3) ...
        BTQ BSFC AFR EGRMF BSNOX PEAKPRESS VGTSPEED i*ones(size(Xlocal,1),1)]; %#ok<AGROW>
    DList(i) = localDesign;
```

```
end
```

Create Project and Test Plan

Create an mbcmodel project and build models

```
project = mbcmodel.CreateProject('DieselPointByPoint');
% Create test plan
TP = CreateTestplan( project, {LocalInputs,OperatingPointInputs} );
TP.Name = 'Point-by-point';

% Assign list of local designs to test plan
TP.Designs{1} = DList;
% Set as best design in test plan
TP.BestDesign{2} = OperatingPointDesign;
```

Make and Import Data Structure

```
D = project.CreateData();
s = D.ExportToMBCDataStructure;
s.varNames = {LocalInputs.Name ...
    'SPEED', 'MAINFUEL','BTQ' 'BSFC' 'AFR' 'EGRMF' 'BSNOX' 'PEAKPRESS' 'VGTSPEED','logno'};
s.varUnits = {'deg','MPa','mm','ratio',...
    'rpm','mg/stroke','Nm','g/kWhr','ratio','ratio','g/kWhr','MPa','rpm',''};
s.data = data;
```

```
D.BeginEdit;
D.ImportFromMBCDataStructure(s);
D.CommitEdit;
TP.AttachData(D);
```

Build Point-by-Point Boundary Model

Use a convex hull for local boundaries.

```
B = TP.Boundary.Local.CreateBoundary('Point-by-Point');
B.LocalModel = CreateBoundary(B.LocalModel,'Convex hull');
% Add point-by-point boundary model to project.
TP.Boundary.Local.Add(B);
```

Build Response Models

Use a local multiple model and no global model.

```
Responses = {'BSFC','BSNOX','AFR','EGRMF','PEAKPRESS','VGTSPEED','MAINFUEL'};
for i = 1:length(Responses)
    TP.CreateResponse(Responses{i},L,[]);
end
```

Inspect and Refine Models

Finally, you must visually inspect the fitted models to verify that the model quality is acceptable. You usually need to identify and remove outliers. You can use the command-line to plot diagnostics and remove outliers, but it is easier to use the graphical and statistical tools in the Model Browser (mbcmodel) provided in the Model-Based Calibration Toolbox.

You must save the project to a file before loading it into the Model Browser.

```
project.Save('DieselPointByPoint.mat');
```

```
mbcmodel('DieselPointByPoint.mat')
```