Model-Based Calibration Toolbox

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Programming
- Simulation

Reference

Version 3

The MathWorks

How to Contact The MathWorks

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

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Model-Based Calibration Toolbox Reference

© COPYRIGHT 2005–2006 by The MathWorks, Inc.

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

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

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks, and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc.

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

Patents

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

Revision History

November 2005	Online only	New for Version 3.0 (Release 14SP3+)
September 2006	Online only	Version 3.1 (Release 2006b)

Contents

Commands – By Category

1

Commands — Alphabetical List

2

1

Commands – By Category

Handling Data

mbcmodel.data — Properties (p. 1-2)	Examine data objects
mbcmodel.data — Methods (p. 1-3)	Work with data objects

mbcmodel.data — Properties

`Filters`	Structure array holding user-defined filters
`IsBeingEdited`	Boolean signaling if data is being edited
`IsEditable`	Boolean signaling whether data is editable
`Name`	Name of project, data, test plan, or model
`NumberOfRecords`	Total number of records in data object
`NumberOfTests`	Total number of tests being used in model
`Owner`	Object from which data was received
`RecordsPerTest`	Number of records in each test
`SignalNames`	Names of signals held by data
`SignalUnits`	Names of units in data
`TestFilters`	Structure array holding user-defined test filters
`UserVariables`	Structure array holding user-defined variables

mbcmodel.data — Methods

`AddFilter`	Add user-defined filter to data set
`AddTestFilter`	Add user-defined test filter to data set
`AddVariable`	Add user-defined variable to data set
`Append`	Append data to data set
`BeginEdit`	Begin editing session on data object
`CommitEdit`	Update temporary changes in data
`DefineNumberOfRecordsPerTest`	Define exact number of records per test
`DefineTestGroups`	Define rule-based test groupings
`ExportToMBCDataStructure`	Export data to MBC data structure
`ImportFromFile`	Load data from file
`ImportFromMBCDataStructure`	Load data from MBC data structure
`ModifyFilter`	Modify user-defined filter in data set
`ModifyTestFilter`	Modify user-defined test filter in data set
`ModifyVariable`	Modify user-defined variable in data set
`RemoveFilter`	Remove user-defined filter from data set
`RemoveTestFilter`	Remove user-defined test filter from data set
`RemoveVariable`	Remove user-defined variable from data set
`RollbackEdit`	Undo most recent changes to data
`Value`	Double data from data object

Handling Projects

mbcmodel.project — Properties (p. 1-4)	Examine project objects
mbcmodel.project — Methods (p. 1-4)	Work with project objects

mbcmodel.project — Properties

`Data`	Array of data objects in project or test plan
`Filename`	Full path to project file
`Modified`	Boolean signaling whether project has been modified
`Name`	Name of project, data, test plan, or model
`TestPlans`	Array of test plan objects in project

mbcmodel.project — Methods

`CopyData`	Create data object from copy of existing object
`CreateData`	Create data object
`CreateTestplan`	Create new test plan
`Load`	Load existing project file
`New`	Create new project file
`Remove`	Remove project, test plan, or model
`RemoveData`	Remove data from project
`Save`	Save project to currently selected filename
`SaveAs`	Save project to new file

Handling Test Plans

mbcmodel.testplan — Properties

`Data`	Array of data objects in project or test plan
`InputSignalNames`	Names of signals in data that are being modeled
`InputsPerLevel`	Number of inputs at each level in model
`Levels`	Number of levels in hierarchical model
`Name`	Name of project, data, test plan, or model
`Responses`	Array of available responses for test plan

mbcmodel.testplan — Methods

`AttachData`	Attach data from project to test plan
`CreateResponse`	Create new response model for test plan
`DetachData`	Detach data from test plan
`GetDesignMatrix`	Design points from test plan
`Remove`	Remove project, test plan, or model

Handling Models

Hierarchical Models (p. 1-6)	Working with hierarchical models
Local Models (p. 1-7)	Working with local models
Response Models (p. 1-9)	Working with response models
Models (p. 1-10)	Working with model objects
Model Parameters (p. 1-11)	Examine model parameter objects

Hierarchical Models

mbcmodel.hierarchicalresponse — Properties

`InputSignalNames`	Names of signals in data that are being modeled
`Level`	Level in test plan of response
`LocalResponses`	Array of local responses for response
`Name`	Name of project, data, test plan, or model
`NumberOfTests`	Total number of tests being used in model
`ResponseSignalName`	Name of signal or response feature being modeled

mbcmodel.hierarchicalresponse — Methods

`AlternativeModelStatistics`	Summary statistics for alternative models
`CreateAlternativeModels`	Create alternative models from model template
`DoubleInputData`	Data being used as input to model

`DoubleResponseData`	Data being used as output to model for fitting
`Export`	Make command-line or Simulink export model
`OutlierIndices`	Indices of `DoubleInputData` marked as outliers
`PEV`	Predicted error variance of model at specified inputs
`PredictedValue`	Predicted value of model at specified inputs
`Remove`	Remove project, test plan, or model
`SummaryStatistics`	Summary statistics for response

Local Models

mbcmodel.localresponse — Properties

`InputSignalNames`	Names of signals in data that are being modeled
`Level`	Level in test plan of response
`Name`	Name of project, data, test plan, or model
`NumberOfTests`	Total number of tests being used in model
`ResponseFeatures`	Array of response features for response
`ResponseSignalName`	Name of signal or response feature being modeled

mbcmodel.localresponse — Methods

`AlternativeModelStatistics`	Summary statistics for alternative models
`CreateAlternativeModels`	Create alternative models from model template
`DiagnosticStatistics`	Diagnostic statistics for response
`DoubleInputData`	Data being used as input to model
`DoubleResponseData`	Data being used as output to model for fitting
`Export`	Make command-line or Simulink export model
`MakeHierarchicalResponse`	Build two-stage model from response feature models
`OutlierIndices`	Indices of `DoubleInputData` marked as outliers
`OutlierIndicesForTest`	Indices marked as outliers for test
`PEVForTest`	Local model predicted error variance for test
`PredictedValueForTest`	Predicted local model response for test
`Remove`	Remove project, test plan, or model
`RemoveOutliers`	Remove outliers in input data by index or rule, and refit models
`RemoveOutliersForTest`	Remove outliers on test by index or rule and refit models
`SummaryStatistics`	Summary statistics for response

Response Models

mbcmodel.response — Properties

`AlternativeResponses`	Array of alternative responses for this response
`InputSignalNames`	Names of signals in data that are being modeled
`Level`	Level in test plan of response
`Model`	Model object within response object
`Name`	Name of project, data, test plan, or model
`NumberOfTests`	Total number of tests being used in model
`ResponseSignalName`	Name of signal or response feature being modeled

mbcmodel.response — Methods

`AlternativeModelStatistics`	Summary statistics for alternative models
`ChooseAsBest`	Choose best model from alternative responses
`CreateAlternativeModels`	Create alternative models from model template
`DiagnosticStatistics`	Diagnostic statistics for response
`DoubleInputData`	Data being used as input to model
`DoubleResponseData`	Data being used as output to model for fitting
`Export`	Make command-line or Simulink export model

`OutlierIndices`	Indices of `DoubleInputData` marked as outliers
`PEV`	Predicted error variance of model at specified inputs
`PredictedValue`	Predicted value of model at specified inputs
`Remove`	Remove project, test plan, or model
`RemoveOutliers`	Remove outliers in input data by index or rule, and refit models
`SummaryStatistics`	Summary statistics for response

Models

Response objects contain an `mbcmodel.model` object with the following properties and methods.

mbcmodel.model — Properties

`NumberOfInputs`	Number of model inputs
`Parameters`	Model parameters
`Response`	Response for model object
`Status`	Model status: fitted, not fitted or best
`XData`	X (or input) data for model
`XDataNames`	X data (or input) variable names for model
`YData`	Y (or response) data for model

mbcmodel.linearmodel — Linear Model Methods

`ParameterStatistics`	Calculate parameter statistics for linear model
`StepwiseRegression`	Change stepwise selection status for specified terms

mbcmodel.model — Methods

`Fit`	Fit model to new or existing data, and provide summary statistics
`Jacobian`	Calculate Jacobian matrix for model at existing or new X points
`ModelSetup`	Opens Model Setup dialog box where you can alter model type
`PEV`	Predicted error variance of model at specified inputs
`PredictedValue`	Predicted value of model at specified inputs
`SummaryStatistics`	Summary statistics for response
`UpdateResponse`	Replace model in response

Model Parameters

These properties of the `mbcmodel.modelparameters` object are all read-only. An `mbcmodel.modelparameters` object is contained within the `Parameters` property of an `mbcmodel.model` object.

mbcmodel.modelparameters — Properties

`Names`	Model parameter names
`NumberOfParameters`	Number of included model parameters
`Values`	Values of model parameters

mbcmodel.linearmodelparameters — Linear Model Properties

A mbcmodel.linearmodelparameters object is a mbcmodel.modelparameters object plus the following properties.

`SizeOfParameterSet`	Number of model parameters
`StepwiseSelection`	Model parameters currently included and excluded
`StepwiseStatus`	Stepwise status of parameters in model

mbcmodel.rbfmodelparameters — RBF Model Properties

A mbcmodel.rbfmodelparameters object is a mbcmodel.linearmodelparameters object plus the following properties.

`Centers`	Centers of RBF model
`Widths`	Width data from RBF model

Commands — Alphabetical List

AddFilter

Purpose

Add user-defined filter to data set

Syntax

```
D = AddFilter(D, expr)
```

Description

This is a method of `mbcmodel.data`.

A filter is a constraint on the data set used to exclude some records. You define the filter using logical operators or a logical function on the existing variables.

`D` is the `mbcmodel.data` object you want to filter.

`expr` is an input string holding the expression that defines the filter.

Examples

```
  AddFilter(D, 'AFR < AFR_CALC + 10');
```

The effect of this filter is to keep all records where AFR < AFR_CALC +10.

```
  AddFilter(D, 'MyFilterFunction(AFR, RPM, TQ, SPK)');
```

The effect of this filter is to apply the function MyFilterFunction using the variables AFR, RPM, TQ, SPK.

All filter functions receive an `nx1` vector for each variable and must return an `nx1` logical array out. In that array, true (or `1`) indicates a record to keep, and false (or `0`) indicates a record to discard.

See Also

`ModifyFilter`; `RemoveFilter`; `Filters`; `AddTestFilter`
`ModifyTestFilter`

Purpose

Add user-defined test filter to data set

Syntax

```
D = AddTestFilter(D, expr)
```

Description

This is a method of `mbcmodel.data`.

A test filter is a constraint on the data set used to exclude some entire tests. You define the test filter using logical operators or functions on the existing variables.

`D` is your data object

`expr` is the input string holding the definition of the new test filter.

Examples

```
AddTestFilter(d1, 'any(n>1000)');
```

The effect of this filter is to include all tests in which all records have speed (`n`) greater than 1000.

Similar to filters, test filter functions are iteratively evaluated on each test, receiving an `nx1` vector for each variable input in a test, and must return an `1x1` logical array out. In that array, true (or `1`) indicates a record to keep, and false (or `0`) indicates a test to discard.

```
AddTestFilter(data, 'length(LOGNO) > 6');
```

The effect of this filter is to include all tests with more than 6 records.

See Also

`ModifyTestFilter`; `RemoveTestFilter`; `TestFilters`; `AddFilter`

AddVariable

Purpose

Add user-defined variable to data set

Syntax

```
D = AddVariable(D, expr, units)
```

Description

This is a method of `mbcmodel.data`.

You can define new variables in terms of existing variables. Note that variable names are case sensitive.

`D` is your data object

`expr` is the input string holding the definition of the new variable

`units` is an optional input string holding the units of the variable

Examples

```
AddVariable(D, 'MY_NEW_VARIABLE = TQ*AFR/2');
AddVariable(D, 'funcVar = MyVariableFunction(TQ, AFR, RPM)',
'lb');
AddVariable(D, 'TQ=tq');
```

The last example could be useful if the signal names in the data do not match the model input factor names in the test plan template file.

See Also

`ModifyVariable`; `RemoveVariable`; `UserVariables`

Purpose

Summary statistics for alternative models

Syntax

```
S = AlternativeModelStatistics(R)
S = AlternativeModelStatistics(R, Name)
```

Description

This is a method of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

This returns an array (`S`) of summary statistics of all the alternative model fits, to be used to select the best model. These are the summary statistics seen in the list view at the bottom of the Model Browser GUI in any model view.

You must use CreateAlternativeModels before you can compare the alternative responses using AlternativeModelStatistics. Then use ChooseAsBest.

`R` is the model object whose alternative response models you want to compare. `R` could be a local (`L`), response feature (`R`) or hierarchical response (`HR`) model.

`S` is a structure containing `Statistics` and `Names` fields.

- `S.Statistics` is a matrix of size (number alternative responses x number of statistics)
- `S.Names` is a cell array containing the names of all the statistics

The available statistics vary according to what kind of parent model (two-stage, local, response feature or response) produced the alternative models, and include PRESS RMSE, RMSE, and Two-Stage RMSE.

All the available statistics are calculated unless you specify which you want. You can specify only the statistics you require using the following form:

```
S = AlternativeModelStatistics(R, Name)
```

This returns a double matrix containing only the statistics specified in Name.

Note that you use SummaryStatistics to examine the fit of the current model, and AlternativeModelStatistics to examine the fit of several alternative child models.

Examples

```
S = AlternativeModelStatistics(R);
```

See Also

CreateAlternativeModels; SummaryStatistics; ChooseAsBest

Purpose

Array of alternative responses for this response

Syntax

```
altR = get(R, 'AlternativeResponses')
```

Description

This is a property of the response model object, `mbcmodel.response` (`R`).

It returns a list of alternative responses used for one-stage or response feature models.

Examples

```
R = get(testplan, 'Responses');
TQ = R(1);
AR = get(TQ, 'AlternativeResponses');
```

See Also

`LocalResponses`; `ResponseFeatures`

Append

Purpose

Append data to data set

Syntax

```
D = Append(D, otherData)
```

Description

This is a method of mbcmodel.data.

You can use this to add new data to your existing data set, D.

otherData is the input argument holding the extra data to add below the existing data. This argument can either be an mbcmodel.data object or a double array. The behavior is different depending on the type.

If otherData is an mbcmodel.data object then Append will look for common SignalNames between the two sets of data. If no common SignalNames are found then a error will be thrown. Any common signals will be Appended to the existing data and other signals will be filled with NaN.

If otherData is a double array then it must have exactly the same number of columns as there are SignalNames in the data, and a simple vertcat (vertical concatenation) is applied between the existing data and otherData.

Examples

```
Append(D, CreateData('aDataFile.xls'));
Append(D, rand(10,100));
```

See Also

CreateData

Purpose

Attach data from project to test plan

Syntax

```
newD = AttachData(T, D, Property1, Value, Property2, Value...)
```

Description

This is a method of `mbcmodel.testplan`.

Use it to attach the data you want to model to the test plan.

`T` is the test plan object, `D` is the data object.

The following table shows the valid properties and their corresponding possible values. These are the settings shown in the last page of the Data Wizard (if there is a design) in the Model Browser. For more information on the meaning of these settings, refer to the Data Wizard section (under Data) in the Model Browser User's Guide. Note that if the testplan has responses set up the models will be fitted when you attach data.

Property	Value	Default
`unmatcheddata`	{'`all`', '`none`'}	`'all'`
`moredata`	{'`all`', '`closest`'}	`'all'`
`moredesign`	{'`none`', '`closest`'}	`'none'`
`tolerances`	`[1xNumInputs double]`	ModelRange/20

Examples

```
newD = AttachData(T1, D1, `more data', `all');

tol = [0.075, 100, 1, 2];
unmatch = 'all';
moredata = 'all';
moredes = 'none';
AttachData(testplan, data ,...
    'tolerances',    tol,...
    'unmatcheddata', unmatch,...
    'moredata',      moredata,...
```

```
                   'moredesign',    moredes);
```

See Also

`Data`; `DetachData`

Purpose

Begin editing session on data object

Syntax

```
D = BeginEdit(D)
```

Description

This is a method of `mbcmodel.data`.

You must call this method before you can make any changes to a data object.

There are no input arguments. You must call `BeginEdit` before attempting to modify your data object (`D` in the example below) in any way. An error will be thrown if this condition is not satisfied. Data which cannot be edited (see `IsEditable`) will throw an error if `BeginEdit` is called.

Examples

```
  BeginEdit(D);
```

See Also

`CommitEdit`; `RollbackEdit`; `IsEditable`; `IsBeingEdited`

Purpose

Centers of RBF model

Syntax

```
centers = get(params, 'Centers')
```

Description

This is a property of `mbcmodel.rbfmodelparameters`, for Radial Basis Function (RBF) models only. This returns an array of size number_of_centers by number_of_variables.

Examples

```
  centers = get(params, 'Centers');
```

See Also

`Widths`

Purpose

Choose best model from alternative responses

Syntax

```
ChooseAsBest(R, Index)
```

Description

This is a method of the response model object, `mbcmodel.response`. This is the same function as selecting the best model in the Model Selection window of the Model Browser GUI. For a local model `MakeHierarchicalResponse` performs a similar function.

`R` is the object containing the response model

`Index` is the number of the response model you want to choose as best. Use `AlternativeResponses` to find the index for each response model, and use AlternativeModelStatistics to choose the best fit.

Examples

```
ChooseAsBest(R, AlternativeModel)
RMSE = AlternativeModelStatistics(R, 'RMSE');
[mr, Best] = min(RMSE);
ChooseAsBest(R, Best);
```

See Also

`AlternativeResponses`; `AlternativeModelStatistics`; `DiagnosticStatistics`; `MakeHierarchicalResponse`

CommitEdit

Purpose

Update temporary changes in data

Syntax

```
D = CommitEdit(D)
```

Description

This is a method of mbcmodel.data.

Use this to apply changes you have made to the data, such as creating new variables and applying filters to remove unwanted records.

There are no input arguments. Once you have finished editing your data object D you must commit your changes back to the project. Data can only be committed if both IsEditable and IsBeingEdited are true. CommitEdit will throw an error if these conditions are not met.

Examples

```
D = get(P, 'Data');
BeginEdit(D);
AddVariable(D, 'TQ = tq', 'lbft');
AddFilter(D, 'TQ < 200');
DefineTestGroups(D, {'RPM' 'AFR'}, [50 10], 'MyLogNo');
CommitEdit(D);
```

For an example situation which results in CommitEdit failing:

```
D  = get(p, 'Data');
D1 = get(p, 'Data');
BeginEdit(D1);
tp = get(p, 'Testplan');
Attach(tp, D);
```

Where p is an mbcmodel.project object, and D and D1 are mbcmodel.data objects.

At this point IsEditable(D1) becomes false because it is now Attached to the test plan and hence can only be modified from the test plan. If you now enter:

```
OK = get(D1, 'IsEditable')
```

the answer is `false`.

If you now enter:

```
CommitEdit(D1);
```

An error is thrown because the data is no longer editable. The error message informs you that the data may have been attached to a test plan and can only be edited from there.

See Also

`BeginEdit`; `RollbackEdit`; `IsEditable`; `IsBeingEdited`

Purpose

Create data object from copy of existing object

Syntax

```
newD = CopyData(P, D)
newD = CopyData(P, Index)
```

Description

This is a method of `mbcmodel.project`.

Use this to duplicate data, for example if you want to make changes for further modeling but want to retain the existing data set. You can refer to the data object either by name or index.

`P` is the project object.

`D` is the data object you want to copy.

`Index` is the index of the data object you want to copy.

Examples

```
D2 = CopyData(P1, D1);
```

See Also

`Data`; `CreateData`; `RemoveData`

Purpose

Create alternative models from model template

Syntax

```
R = CreateAlternativeModels(R, models, criteria)
R = CreateAlternativeModels(R,
LocalModels,LocalCriteria,GlobalModels,GlobalCriteria)
```

Description

This is a method of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

This is the same as the Build Models function in the Model Browser GUI. A selection of child node models are built. The results depend on where you call this method from. Note that the hierarchical model is automatically constructed when `CreateAlternativeModels` is called for a local model.

- This option makes alternative response feature models for each response feature.

  ```
  R = CreateAlternativeModels(R, models, criteria)
  ```

  - `Models` is the list of models (from the model template)
  - `Criteria` is the selection criteria for best model (from the statistics available from `AlternativeModelStatistics`).

- This option makes alternative local models as well as alternative response feature models

  ```
  R = CreateAlternativeModels(R,
  LocalModels,LocalCriteria,GlobalModels,GlobalCriteria)
  ```

  - `LocalModels` is the list of local models - you must pass in an empty matrix)
  - `LocalCriteria` is `'Two-Stage RMSE'`
  - `GlobalModels` is the list of global models (from the model template)
  - `GlobalCriteria` is the selection criteria for best model

You construct a model template (such as ‘mymodels.mbm’) in the Model Browser. From any response (global or one-stage model) with alternative responses (child nodes), select **Model > Make Template**. You can save the child node model types of your currently selected modeling node as a model template. Alternatively from any response click **Build Models** in the toolbar and create a series of alternative response models in the dialog.

Examples

```
mymodels = 'mymodels.mbm';
mlist = {};
load('-mat', mymodels);
critera = 'PRESS RMSE';
CreateAlternativeModels(R, [], 'Two-Stage RMSE', mlist,
criteria);
```

Note that the model template contains the variable `mlist`.

See Also

`AlternativeModelStatistics`

Purpose

Create data object

Syntax

```
D = mbcmodel.CreateData(filename, filetype)
```

D = mbcmodel.CreateData(P, filename, filetype)

Description

The first syntax is a function, the second (using `P`) is a method of `mbcmodel.project`. They both create a data object for use with command-line MBC. You can use the first syntax to manipulate data independently of any project, and the second method attaches the data to a particular project object.

Use this to create a new data set for modeling.

`P` is the project object.

`filename` and `filetype` are both optional arguments that are passed to `ImportFromFile` to ensure that there is a quick mechanism for creating data from a file. You must call `BeginEdit` before you can make other changes to the data if you want, such as adding filters.

If you do not specify a `filename` to use the shortcut for loading data, you must call `BeginEdit` and then fill the empty data set by calling `ImportFromFile`. You can then add filters or user variables as before, then call `CommitEdit` to save your changes.

`filename` is a string holding the full path to the file to load.

`filetype` is an optional file type to load. See `xregcheckindataloadingfunction` for the specification of the allowed filetypes. This defaults to 'auto' which will attempt to guess the filetype based on the extension of the file being loaded. i.e. if the file extension is .xls then MBC will try the Excel File Loader.

Examples

```
data = mbcmodel.CreateData(P, 'D:\MBCWork\data1.xls');
D = mbcmodel.CreateData(P);
```

Where P is an `mbcmodel.project` object.

See Also

`BeginEdit`; `CopyData`; `RemoveData`; `Data`; `ImportFromFile`; `CommitEdit`

Purpose

Create project object

Syntax

```
P = mbcmodel.CreateProject
```

Description

This is a function that creates an `mbcmodel.project` object.

`P` is the project object.

Examples

```
   P = mbcmodel.CreateProject;
```

CreateResponse

Purpose

Create new response model for test plan

Syntax

```
R = CreateResponse(T, name)
```

Description

This is a method of `mbcmodel.testplan`.

`T` is the test plan object, `R` is the new response model.

`name` is the variable name for the new response.

Examples

```
R = CREATERESPONSE(T, 'torque');
TQ_response = CreateResponse(testplan, 'TQ');
```

See Also

`Responses`

Purpose

Create new test plan

Syntax

```
T = CreateTestplan(P, templateFilename, name)
```

Description

This is a method of the `mbcmodel.project` object.

You need a test plan template to use this method from the command line. You set these up in the Model Browser GUI. This set up includes number of stages, inputs, base models, and designs. If the test plan is used as part of a previous project it is also possible to save response models in the test plan.

Once you have created a new test plan (using a template) you can add data to model, and new responses. Note that the model input signal names specified in the template *must* match the signal names in the data.

`P` is the project object.

`templateFilename` is the full name and path to the template file.

`name` is the optional name for the new test plan object.

Examples

```
T = CreateTestplan(P1, 'd:\MBCwork\TQtemplate1', 'newtestplan')
testplan = CreateTestplan(P, 'example_testplan')
```

See Also

`AttachData`; `CreateResponse`; `Responses`; `Data`; `Levels`; `InputSignalNames`; `InputsPerLevel`

Data

Purpose

Array of data objects in project or test plan

Syntax

```
allD = get(p, 'Data')
allD = get(T, 'Data')
```

Description

This is a property of mbcmodel.project and mbcmodel.testplan.

It returns an array of mbcmodel.data objects. There may be many data objects in a project, but a test plan can only have one or none.

Examples

```
allD = get(p, 'Data');
```

For a project object p, this example returns an nx1 array of all the data objects.

```
allD = get(T, 'Data');
```

For the test plan object T, this example returns a 1x1 array if the test plan has a data object attached, and 0x1 otherwise.

See Also

CreateData; RemoveData; CopyData

Purpose

Define exact number of records per test

Syntax

```
D = DefineNumberOfRecordsPerTest(D, number, testnumAlias)
```

Description

This is a method of `mbcmodel.data`.

You can use this to set one test per record for one-stage modeling.

`number` is the input specifying the number of records to include in each test. Most usually this will be used to specify one test per record.

`testnumAlias` is an optional string input to define the `SignalName` that should be used as the testnumber within MBC. Defaults to the index of the test.

Note `testnumaAias` uses the first record in the test as the testnumber, and testnumbers *are* unique so any duplicates will be modified.

Examples

```
DefineNumberOfRecordsPerTest(D, 1);
DefineNumberOfRecordsPerTest(D, 10, 'MYLOGNO');
```

See Also

`DefineTestGroups`

Purpose

Define rule-based test groupings

Syntax

```
D = DefineTestGroups(D, variables, tolerances, testnumAlias,
reorder)
```

Description

This is a method of `mbcmodel.data`.

You can impose rules to collect records of the current data set (`D`) into groups; these groups are referred to as **tests**. Test groupings are used to define hierarchical structure in the data for two-stage modeling.

Select a variable or variables to group by and set `tolerances`. The tolerance is used to define groups: on reading through the data, when the value of any specified variable changes by more than the tolerance, a new group is defined.

`variables` is the input cell array of strings holding the `SignalNames` on which to define the test groupings

`tolerances` is the input double array of the same length as variables holding the required tolerances for the test grouping definition

`testnumAlias` is an optional string input to define the `SignalName` that should be used as the testnumber within MBC. Defaults to the index of the test.

Note `testnumAlias` uses the first record in the test as the testnumber, and testnumbers *are* unique so any duplicates will be modified.

`reorder` is an optional Boolean indicating that the data should be reordered within the data set. Defaults to `false`.

See the section on Test Groupings (under Data) in the Model Browser User's Guide for more information on these inputs.

Examples

```
DefineTestGroups(D, {'AFR' 'RPM'}, [0.1 30], 'MYLOGNO', false);
```

See Also

`DefineNumberOfRecordsPerTest`; `NumberOfTests`

Purpose

Detach data from test plan

Syntax

```
T = DetachData(T)
```

Description

This is a method of `mbcmodel.testplan`.

`T` is the test plan object. A test plan can only use a single data set, so you do not need to specify the data object.

Examples

```
  DetachData(T1);
```

See Also

`AttachData`

Purpose

Diagnostic statistics for response

Syntax

```
S = DiagnosticStatistics(R, TestNumbers, Stats)
```

Description

This is a method of the local and response model objects, `mbcmodel.localresponse` and `mbcmodel.response`.

The options available are model-specific and are the same options shown in the drop-down menus of the scatter plots (the top plots) in the local and global (response feature) model views of the toolbox GUI.

`S` is a structural array containing `Statistics` and `Names` fields.

`R` is the response model object.

`Testnumbers` specifies the index into tests for local or hierarchical models.

`Stats` is an optional input that defines which diagnostic statistics you want from the available list. If you don't specify `Stats`, you get all available statistics.

A row is set to `NaN` if that point is removed.

Examples

```
studentRes = DiagnosticStatistics(local, tn, 'Studentized
residuals');
```

See Also

`SummaryStatistics`; `AlternativeModelStatistics`

DoubleInputData

Purpose

Data being used as input to model

Syntax

```
X = DoubleInputData(R, TestNumber)
```

Description

This is a method of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`. It returns an array (`X`) containing the input data used for fitting the model.

`R` is the response model object

`TestNumber` is an optional input to specify the tests you want.

Examples

```
  X = DoubleInputData(R);
  x = DoubleInputData(local, tn);
```

See Also

`DoubleResponseData`

Purpose

Data being used as output to model for fitting

Syntax

```
Y = DoubleResponseData(R, TestNumber)
```

Description

This is a method of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`. It returns an array (`Y`) containing the response data used for fitting the model.

`R` is the response model object.

`TestNumber` is an optional input to specify the tests you want.

Examples

```
Y = DoubleResponseData(R);
y = DoubleResponseData(local, tn);
```

See Also

`DoubleInputData`

Export

Purpose

Make command-line or Simulink export model

Syntax

```
M = Export(R, Format)
```

Description

This is a method of these model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

`Format` must be `'MATLAB'` or `'Simulink'`; an error will be thrown if this is incorrect.

You can evaluate models exported to the MATLAB workspace in the same way as when exported from the Model Browser. You can save these models as a `*.mat` file and load them into CAGE.

`R` is the object containing the response models from the node you are exporting from.

Examples

```
M = Export(R2, 'MATLAB');
mbt_model = Export(maxTQ, 'MATLAB');
```

Purpose

Export data to MBC data structure

Syntax

```
mbcStruct = ExportToMBCDataStructure (D)
```

Description

This is a method of `mbcmodel.data`.

It converts the specified data object (`D`) to the MBC Data Structure format.

An MBC Data Structure is a structure array that contains the following fields:

- `varNames` is a cell array of strings that hold the names of the variables in the data (`1xn` or `nx1`)
- `varUnits` is a cell array of strings that hold the units associated with the variables in varNames (`1xn` or `nx1`). This array can be empty, in which case no units are defined
- `data` is an array that holds the values of the variables (`m x n`)
- `comment` is an optional string holding comment information about the data.

For more information see the Data Loading Function section (under Technical Documents) in the Model Browser User's Guide. See also `xregcheckindataloadingfunction` for the specification.

Examples

```
   X = ExportToMBCDataStructure(D1);
```

See Also

`ImportFromMBCDataStructure`

Filename

Purpose

Full path to project file

Syntax

```
Name = get(P, 'Filename')
```

Description

This is a property of `mbcmodel.project`.

Examples

```
Name = get(P, 'Filename');
```

Purpose

Structure array holding user-defined filters

Syntax

```
filt = get(D, `Filters')
```

Description

This is a property of `mbcmodel.data`.

It returns a structure array holding information about the currently defined filters. The array will be the same length as the number of currently defined filters, with the following fields for each filter:

- `Expression` — The string expression as defined in `AddFilter` or `ModifyFilter`
- `AppliedOK` — Boolean indicating that the filter was successfully applied
- `RemovedRecords` — Boolean vector indicating which records the filter removed. Note that many filters could remove the same record
- `Message` — String holding information on the success or otherwise of the filter

Examples

```
filters = get(D1, 'Filters');
```

See Also

`AddFilter`; `ModifyFilter`; `RemoveFilter`

Purpose

Fit model to new or existing data, and provide summary statistics

Syntax

```
statistics = Fit(model, optional X, optional Y)
```

Description

This is a method of `mbcmodel.model`.

This fits the model to new data or its existing data. If X and Y are not specified then the existing model XData and YData are used, otherwise X and Y are placed in XData and YData and the model fitted.

The statistics returned are defined by the summary statistics for the response object the model came from. To see these call SummaryStatistics. These are the statistics that appear in the Summary Statistics pane of the Model Browser GUI. The statistics returned depend on the model type.

For a linear model, the statistics are:

’Observations’,’Parameters’,’Box-Cox’,’PRESS RMSE’,’RMSE’.

For a neural net model:

’Observations’,’Parameters’, ’Box-Cox’,’RMSE’, ’R^2’.

Examples

```
statistics = Fit(knot)
statistics =
   27.0000    7.0000    1.0000    3.0184    2.6584
```

See Also

`SummaryStatistics`; `UpdateResponse`

GetDesignMatrix

Purpose

Design points from test plan

Syntax

```
design = GetDesignMatrix(T)
```

Description

This is a method of `mbcmodel.testplan`.

It returns a double array holding the values of the design points.

Examples

```
design = GetDesignMatrix(T);
```

ImportFromFile

Purpose

Load data from file

Syntax

```
D = ImportFromFile(D, filename, filetype)
```

Description

This is a method of the `mbcmodel.data` object.

First you must use `CreateData`, than `BeginEdit` before you can call ImportFromFile to bring data into your new data object, `D`.

Note that you can specify `filename` and `filetype` when you call `CreateData` as a shortcut for loading data from a file. You still need to call `BeginEdit` before you can make changes to the data.

`filename` is a string holding the full path to the file to load.

`filetype` is an optional file type to load. See `xregcheckindataloadingfunction` for the specification of the allowed filetypes. This defaults to '`auto`' which will attempt to guess the filetype based on the extension of the file being loaded. i.e. if the file extension is .xls then MBC will try the Excel File Loader.

Examples

```
ImportFromFile(D, 'D:\MBCData\Raw Data\testdata.xls');
```

See Also

`CreateData`; `BeginEdit`; `ImportFromMBCDataStructure`; `RemoveData`; `Append`

Purpose

Load data from MBC data structure

Syntax

```
D = ImportFromMBCDataStructure(D, mbcStruct)
```

Description

This is a method of `mbcmodel.data`.

First you must use `CreateData`, than `BeginEdit` before you can bring data into your new data object.

An MBC Data Structure is a structure array that contains the following fields:

- `varNames` is a cell array of strings that hold the names of the variables in the data (`1xn` or `nx1`)
- `varUnits` is a cell array of strings that hold the units associated with the variables in varNames (`1xn` or `nx1`). This array can be empty, in which case no units are defined
- `data` is an array that holds the values of the variables (`m x n`)
- `comment` is an optional string holding comment information about the data.

For more information see the Data Loading Function section (under Technical Documents) in the Model Browser User's Guide. See also `xregcheckindataloadingfunction` for the specification.

Examples

```
ImportFromMBCDataStructure(D, mbcStruct);
```

See Also

`ImportFromFile`; `CreateData`; `BeginEdit`; `RemoveData`; `Append`; `ExportToMBCDataStructure`

InputSignalNames

Purpose

Names of signals in data that are being modeled

Syntax

```
inputs = get(A, 'InputSignalNames')
```

Description

This is a property of `mbcmodel.testplan` and the modeling objects `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

`A` can be a test plan (`T`) or model (`L`, `R`, `HR`) object.

Examples

```
inputs =  get(T, 'InputSignalNames');
InputFactors = get(thisRF, 'InputSignalNames');
```

See Also

`SignalNames`

Purpose

Number of inputs at each level in model

Syntax

```
L = get(T, 'InputsPerLevel')
```

Description

This is a property of `mbcmodel.testplan`.

This is a vector of length `Levels`. Each element defines the number of inputs at that level. See “Understanding Model Structure” for an explanation of the levels in a test plan.

Examples

```
L = get(T, 'InputsPerLevel')
L =
  2  4
```

This answer means the test plan `T` has 2 local inputs and 4 global inputs.

See Also

`Levels`; `Level`

IsBeingEdited

Purpose

Boolean signaling if data is being edited

Syntax

```
OK = get(D, 'IsBeingEdited')
```

Description

This is a property of `mbcmodel.data`.

This Boolean property indicates that the data is currently being edited. It also indicates that previously there was a successful call to `BeginEdit` and hence that whatever changes have been applied can be undone by calling `RollbackEdit`. It does not indicate that a call to `CommitEdit` will necessarily succeed. See `CommitEdit` for an example of this case.

Examples

```
OK = get(D, 'IsBeingEdited');
```

See Also

`BeginEdit`; `IsEditable`; `CommitEdit`; `RollbackEdit`

IsEditable

Purpose

Boolean signaling whether data is editable

Syntax

```
OK = get(d, 'IsEditable')
```

Description

This is a property of `mbcmodel.data`.

This Boolean property indicates if a particular piece of data is editable. The following rules apply

- If the data was created using `mbcmodel.CreateData` and was not `Attached` to a test plan it is editable.
- If the data was created or retrieved from the project and was not `Attached` to a test plan it is editable.
- If the data was `Attached` to a test plan and was subsequently retrieved from that test plan it is editable.

Examples

```
D  = get(p, `Data');
D1 = get(p, `Data');
BeginEdit(D1);
tp = get(p, `Testplan');
Attach(tp, D);
```

Where `p` is an `mbcmodel.project` object, and `D` and `D1` are `mbcmodel.data` objects.

At this point `get(D1, 'IsEditable')` becomes false because `D1` is now `Attached` to the test plan and hence can only be modified from the test plan. If you now enter:

```
OK = get(D1, 'IsEditable')
```

the answer is `false`.

See Also

`BeginEdit`; `IsBeingEdited`; `CommitEdit`; `RollbackEdit`

Jacobian

Purpose

Calculate Jacobian matrix for model at existing or new X points

Syntax

```
J = Jacobian(model, optional X)
```

Description

This is a method of `mbcmodel.model`.

This calculates the Jacobian matrix for the model at existing or new X points. If X is not specified then the existing XData is used. The Jacobian is the regression matrix for linear models and RBF models.

The Jacobian matrix (for linear and RBF models) is the same as the Regression Matix in the Design Evaluation Tool GUI. These matrices only include the terms currently selected in the model.

If all terms are included (none removed by Stepwise) then the Jacobian (for linear and RBF models) is the same as the Full FX matrix found in the Design Evaluation Tool GUI. The Jacobian matrix only includes the currently selected model terms.

To determine the condition number, use the MATLAB command `cond(J)`.

Examples

```
J = Jacobian(knot);
```

See Also

`XData`

Purpose

Level in test plan of response

Syntax

```
level = get(R, 'Level')
```

Description

This is a property for all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

`R` is the response for which you want the level.

The level is usually 0 for hierarchical models, usually 1 for local models, and usually 2 or 1 for response models. See “Understanding Model Structure” for an explanation of what `Level` indicates about a response.

Examples

```
  level = get(R, 'Level');
```

See Also

`Levels`

Levels

Purpose

Number of levels in hierarchical model

Syntax

```
levels = get(T, 'Levels')
```

Description

This is a property of `mbcmodel.testplan`.

See "Understanding Model Structure" for an explanation of what `Levels` mean.

Examples

```
  levels = get(T, 'Levels');
```

See Also

`Level`

Purpose

Load existing project file

Syntax

```
P = Load(P, Filename)
```

Description

This is a method of `mbcmodel.project`.

`P` is a project object, and `Filename` is the full path to the project you want to load.

Examples

```
   P2 = Load(P2, 'D:/MBCwork/TQproject2.mat');
```

See Also

`New`

LocalResponses

Purpose

Array of local responses for response

Syntax

```
local = get(R, 'LocalResponses')
```

Description

This is a property of the `mbcmodel.hierarchicalresponse` object.

It returns the local model response objects that belong to the hierarchical response `R`.

See "Understanding Model Structure" for an explanation of the relationship between the different response types.

Examples

```
  local = get(TQ_response, 'LocalResponses');
```

Purpose

Build two-stage model from response feature models

Syntax

```
OK = MakeHierarchicalResponse(L,MLE)
```

Description

This is a method of `mbcmodel.localresponse`, that builds a two-stage model from the response feature models and optionally runs MLE (Maximum Likelihood Estimation).

This performs a similar function to `ChooseAsBest` for response models. You can call `MakeHierarchicalResponse` directly, or indirectly by calling `CreateAlternativeModels` for a local model. If you call `CreateAlternativeModels` for a local model, `MakeHierarchicalResponse` will be called automatically.

An error will be thrown if the local and response models are not ready to calculate a two-stage model. This can be the case if you have created alternative models and not chosen the best. A sufficient number of response features models to calculate the two-stage model must be selected.

`L` is the local model object

`MLE` can be `true` or `false`. If `true`, MLE will be calculated.

Examples

```
OK = MakeHierarchicalResponse(L, true)
```

See Also

`ChooseAsBest`

Model

Purpose

Model object within response object

Syntax

```
M = get(R, 'Model')
```

Description

This is a property of all mbcmodel.response objects.

Each response contains a model object (`mbcmodel.model`) that can be extracted and manipulated independently of the project.

Extract a model object from any response object (see `Response`), and then:

- Fit to new data (`Fit`).
- Change model type and settings (`ModelSetup`).
- Include and exclude terms to improve the model (`StepwiseRegression`).
- Examine coefficient values, predicted values, and regression matrices (`ParameterStatistics`; `PredictedValue`; `Jacobian`).
- If you change the model you need to use `UpdateResponse` to replace the new model back into the response object in the project. When you use `UpdateResponse` the new model is fitted to the response data.

Examples

```
knot = get(AlternativeResponses(1), 'Model')
```

Purpose

Opens Model Setup dialog box where you can alter model type

Syntax

```
[newModel, OK] = ModelSetup(oldModel)
```

Description

This is a method of `mbcmodel.model` objects.

This method opens the **Model Setup** dialog box where you can choose new model types and settings. If you click **Cancel** to dismiss the dialog, `OK = false` and `newModel` = `oldModel`. If you click **OK** to close the dialog box, then `OK = true` and `newModel` is your new chosen model setup. Data and response remain the same as `oldModel`. The new model is refitted when you click OK.

Call UpdateResponse to put the new model type back into the response.

Examples

```
  [RBF, OK] = ModelSetup(Cubic);
```

See Also

`UpdateResponse`; `Fit`

Modified

Purpose

Boolean signaling whether project has been modified

Syntax

```
Name = get(P, 'Modified')
```

Description

This is a property of `mbcmodel.project`.

Examples

```
  Name = get(Project, 'Modified');
```

Purpose

Modify user-defined filter in data set

Syntax

```
D = ModifyFilter(D, Index, expr)
```

Description

This is a method of `mbcmodel.data`.

You call this method to modify the expression that defines existing filters.

`D` is a data object.

`Index` is the input index to indicate which of the available filters you wish to modify. Use the property `Filters` to find the index for each filter.

`expr` is the input string holding the expression that defines the filter, as for `AddFilter`.

Examples

```
ModifyFilter(D, 3, 'AFR < AFR_CALC + 20');
```

The effect of this filter is to modify filter number `3` to keep all records where AFR < AFR_CALC + 20.

```
ModifyFilter(D, 2, 'MyNewFilterFunction(AFR, RPM, TQ, SPK)');
```

This modifies filter number `2` to apply the function MyNewFilterFunction.

See Also

`AddFilter`; `RemoveFilter`; `Filters`

ModifyTestFilter

Purpose

Modify user-defined test filter in data set

Syntax

```
D = ModifyTestFilter(D, Index, expr)
```

Description

This is a method of `mbcmodel.data`.

You call this method to modify the expression that defines existing filters.

`D` is a data object.

`Index` is the input index to indicate which of the available test filters you wish to modify. Use the property `TestFilters` to find the index for each test filter.

`expr` is the input string holding the expression that defines the test filter, as for `AddTestFilter`.

Examples

```
  ModifyTestFilter(d1, 2, 'any(n>2000)');
```

The effect of this is to modify test filter number `2` to include all tests in which any records have speed (`n`) greater than 1000.

See Also

`AddTestFilter`; `RemoveTestFilter`; `TestFilters`

Purpose

Modify user-defined variable in data set

Syntax

```
D = ModifyVariable(D, Index, expr, units)
```

Description

This is a method of `mbcmodel.data`.

You call this method to modify the expression that defines existing variables.

`D` is a data object.

`Index` is the input index to indicate which of the available variables you wish to modify. Use the property `UserVariables` to find the index for each variable.

`expr` is the input string holding the expression that defines the variable, as for `AddVariable`

`units` is an optional input string holding the units of the variable

Examples

```
  ModifyVariable(D, 2, 'MY_NEW_VARIABLE = TQ*AFR/2');
```

See Also

`AddVariable`; `RemoveVariable`; `UserVariables`

Name

Purpose

Name of project, data, test plan, or model

Syntax

```
name = get(A, 'Name')
```

Description

This is a property of project, data, test plan, and response objects.

'A' can be any test plan (T), data (D), project (P) or model (L, R, HR) object.

You can change the names of these objects as follows:

```
set(A, `Name', newName)
```

For response (output or Y data) signal names, see ResponseSignalName.

For model parameter names, see Names, and for model object input names, see XDataNames.

For testplan and response object input names, see InputSignalNames, and for data objects, see SignalNames.

Examples

```
ResponseFeatureName = get(thisRF, 'Name');
```

See Also

Names; InputSignalNames; SignalNames; XDataNames; ResponseSignalName

Purpose

Model parameter names

Syntax

```
N = get (params, 'Names')
```

Description

This is a property of `mbcmodel.modelparameters`. It returns the names of all the parameters in the model. These are read-only.

Examples

```
N = get (paramsknot, 'Names')
N =
'1'
'N'
'N^2'
'N*L'
'N*A'
'L'
'L^2'
'L*A'
'A'
'A^2';
```

See Also

`NumberOfParameters`; `Values`; `Name`

New

Purpose

Create new project file

Syntax

```
P = New(P)
```

Description

This is a method of `mbcmodel.project`. Use this to modify a project object to make a new project from scratch. Note the current project gets removed from memory when you open a new one.

`P` is the new project object.

Examples

```
   New(P);
```

See Also

`Load`

NumberOfInputs

Purpose Number of model inputs

Syntax

```
N = get(model, 'NumberOfInputs')
```

Description This is a property of `mbcmodel.model`. It returns the number of inputs to the model.

Examples

```
  N = get(knot, 'NumberOfInputs');
```

See Also `XData`

NumberOfParameters

Purpose

Number of included model parameters

Syntax

```
N = get (knotparams, 'NumberOfParameters')
```

Description

This is a read-only property of `mbcmodel.linearmodelparameters`, for linear models only.

The number returned is the number of parameters currently in the model (you can remove some parameters by using StepwiseRegression). To see which parameters are currently in the model, use StepwiseSelection. Only parameters listed as ‘`in`’ are currently included.

To see the the total possible number of parameters in a linear model, use SizeOfParameterSet.

Use Names and Values to get the parameter names and values.

Examples

```
  N = get (knotparams, 'NumberOfParameters');
```

See Also

`SizeOfParameterSet`; `StepwiseSelection`; `StepwiseRegression`; `Names`; `Values`

NumberOfRecords

Purpose

Total number of records in data object

Syntax

```
get(D, 'NumberOfRecords')
```

Description

This is a property of data objects: `mbcmodel.data`.

Examples

```
numRecords = get(Data, 'NumberOfRecords');
```

NumberOfTests

Purpose

Total number of tests being used in model

Syntax

```
numtests = get(A,'NumberOfTests')
```

Description

This is a property of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`, and data objects `mbcmodel.data`. 'A' can be any model or data object.

Examples

```
  numTests = get(TQ_response, 'NumberOfTests');
```

See Also

`DefineTestGroups`

Purpose

Indices of DoubleInputData marked as outliers

Syntax

```
indices = OutlierIndices(R)
```

Description

This is a method of all model objects: mbcmodel.hierarchicalresponse, mbcmodel.localresponse and mbcmodel.response.

Examples

```
ind = OutlierIndices(R);
bad = OutlierIndices(thisRF);
```

See Also

DoubleInputData

OutlierIndicesForTest

Purpose

Indices marked as outliers for test

Syntax

```
indices = OutlierIndicesForTest(R, TestNumber)
```

Description

This is a method of the local model object, `mbcmodel.localresponse`.

This shows the current records discarded as outliers.

You can use ’:’ to use all tests.

Examples

```
   ind = OutlierIndicesForTest(R, ':');
   bad = OutlierIndicesForTest(local, tn);
```

See Also

`OutlierIndices`

Purpose

Object from which data was received

Syntax

```
O = get(D1, 'Owner')
```

Description

This is a property of `mbcmodel.data`.

- This is empty if the data was created using mbcmodel.`CreateData`
- This is an `mbcmodel.project` object if the data was extracted from a project
- This is an `mbcmodel.testplan` object if the data was extracted from a test plan

Examples

```
O = get(D1, 'Owner');
```

Parameters

Purpose

Model parameters

Syntax

```
P = get(model, 'Parameters')
```

Description

This is a property of `mbcmodel.model`., that contains an object `mbcmodel.modelparameters`. This object contains a number of read-only parameters that describe the model.

All models have these properties:

- `SizeOfParameterSet`
- `Names`
- `Values`

Linear models also have these properties:

- `StepwiseStatus`
- `NumberOfParameters`
- `StepwiseSelection`

Radial Basis Function (RBF) models have all the above properties and these additional properties:

- `Centers`
- `Widths`

Examples

```
  P = get(knot, 'Parameters');
```

See Also

`SizeOfParameterSet`; `Names`; `Values`; `StepwiseStatus`; `NumberOfParameters`; `StepwiseSelection`; `Centers`; `Widths`

Purpose

Calculate parameter statistics for linear model

Syntax

```
values = ParameterStatistics(linearmodel, optional statType)
```

Description

This is a method of `mbcmodel.model`, for linear models only. This calculates parameter statistics for the linear model. If you don't specify `statType`, then a structure with all valid types is output. `statType` may be a string specifying a particular statistic or a cell array of string specifying a number of statistics to output. If `statType` is a string, then `values` is an array of doubles. If `statType` is a cell array of strings, then `values` is a cell array of array of doubles.

The valid types are:

`'Alias'`

`'Covariance'`

`'Correlation'`

`'VIFsingle'`

`'VIFmultiple'`

`'VIFpartial'`

`'Stepwise'`

These types (except Stepwise) appear in the Design Evaluation tool; see the documentation for this tool for details of these matrices.

The `Stepwise` field contains the values found in the Stepwise table. In this array (and in the Stepwise GUI) you can see for each parameter in the model: the value of the coefficient, the standard error of the coefficient, the t value and Next PRESS (the value of PRESS if the status of this term is changed at the next iteration). See the documentation for the Stepwise table. You can also see these Stepwise values when you use `StepwiseRegression`.

Examples

```
values = ParameterStatistics(knot)
values =
         Alias: [7x3 double]
    Covariance: [7x7 double]
   Correlation: [7x7 double]
     VIFsingle: [5x5 double]
   VIFmultiple: [7x1 double]
    VIFpartial: [5x5 double]
      Stepwise: [10x4 double]

values.Stepwise
ans =
  1.0e+003 *
    0.0190    0.0079    0.0210       NaN
    0.0000    0.0000    0.0210    1.9801
    0.0000    0.0000    0.0200    0.2984
   -0.0000    0.0000    0.0200    0.2768
    0.0000    0.0000    0.0200    0.2890
   -0.0526    0.0367    0.0210    0.2679
    0.0911    0.0279    0.0210    0.3837
   -0.0041    0.0024    0.0210    0.2728
   -0.0178    0.0095    0.0200    0.2460
    0.0001    0.0000    0.0210    0.3246
```

See Also

`StepwiseRegression`

Purpose

Predicted error variance of model at specified inputs

Syntax

```
pev = PEV(R, X)
```

Description

This is a method of the hierarchical, response and model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.response`, and `mbcmodel.model`.

`R` is the model object, and `X` is the array of input values where you want to evaluate the PEV of the model.

Note that for an `mbcmodel.model` and `mbcmodel.response` objects only, the the X is optional. That is, the syntax is:

```
PEV = PEV(model, optional X)
```

This calculates the Predicated Error Variance at X. If X is not specified, then X is the existing input values. An array is returned of PEV values evaluated at each data point.

Examples

```
pev = PEV(R, X);
```

See Also

`PEVForTest`

PEVForTest

Purpose

Local model predicted error variance for test

Syntax

```
pev = PEVforTest(L, TestNumber, X)
```

Description

This is a method of the local model object, `mbcmodel.localresponse`.

`L` is the local model object.

`TestNumber` is the test for which you want to evaluate the model PEV.

`X` is the array of inputs where you want to evaluate the PEV of the model.

Examples

```
  pev = PEVforTest(L, TestNumber, X);
```

See Also

`PEV`

Purpose

Predicted value of model at specified inputs

Syntax

```
y = PredictedValue(R,X)
```

Description

This is a method of the hierarchical, response and model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.response`, and `mbcmodel.model`.

`R` is the model object, and `X` is the array of inputs where you want to evaluate the output of the model.

Note that for an `mbcmodel.model` and `mbcmodel.response` objects only, the X is optional. That is, the syntax is:

```
y = PredictedValue(model, optional X)
```

This calculates the predicted value at X. If X is not specified then the X is the existing input values. An array is returned of predicted values evaluated at each data point.

Note that you cannot evaluate model output for a hierarchical model until you have constructed it using `MakeHierarchicalResponse` (or `CreateAlternativeModels`). If you have created alternative response feature models then a best model must be selected. If you have made changes such as removing outliers since choosing a model as best, you may need to choose a new best model.

Examples

```
y = PredictedValue(R, X);
modelPred = PredictedValue(thisRF, x);
```

See Also

`PredictedValueForTest`; `ChooseAsBest`

PredictedValueForTest

Purpose

Predicted local model response for test

Syntax

```
y = PredictedValueForTest(L, TestNumber, X)
```

Description

This is a method of the local model object, `mbcmodel.localresponse`.

`L` is a local model object.

`TestNumber` is the test for which you want to evaluate the model

`X` is the array of inputs where you want to evaluate the output of the model.

Examples

```
  y = PredictedValueForTest(L, TestNumber, X);
```

See Also

`PredictedValue`

Purpose

Number of records in each test

Syntax

```
get(D, 'RecordsPerTest')
```

Description

This is a property of data objects: `mbcmodel.data`. It returns an array, of length `NumberOfTests`, containing the number of records in each test.

Examples

```
numRecords = get(Data, 'RecordsPerTest');
```

Remove

Purpose

Remove project, test plan, or model

Syntax

```
OK = Remove(A)
```

Description

This is a method of all the non-data objects: projects, test plans and all models.

`A` can be any project, test plan or model object.

Datum models cannot be removed if they are in use by other models.

Examples

```
OK = Remove(R3);
```

Purpose

Remove data from project

Syntax

```
P = RemoveData(P, D)
P = RemoveData(P, Index)
```

Description

This is a method of `mbcmodel.project`.

You can refer to the data object either by name or index.

`P` is the project object.

`D` is the data object you want to remove.

`Index` is the index of the data object you want to remove.

Examples

```
  RemoveData(P, D);
```

See Also

`CreateData`; `Data`; `CopyData`

RemoveFilter

Purpose

Remove user-defined filter from data set

Syntax

```
D = RemoveFilter(D, Index)
```

Description

This is a method of the `mbcmodel.data` object.

`Index` is the input index indicating the filter to remove. Use the property `Filters` to find out which filters are present.

Examples

```
  RemoveFilter(D1, 3);
```

See Also

`AddFilter`; `Filters`

Purpose

Remove outliers in input data by index or rule, and refit models

Syntax

```
R = RemoveOutliers(R, Selection);
R = RemoveOutliers(L, LocalSelection, GlobalSelection)
```

Description

This is a method of the local model object, `mbcmodel.localresponse` and the response feature model object `mbcmodel.response`.

All the response feature models are refitted after the local models are refitted. Outlier selection is applied to all tests.

For a response model:

- `R` is a response object
- `Selection` specifies either a set of indices or the name of an outlier selection function, of the following form:

  ```
  Indices = myMfile(model, data, factorName)
  ```

  The factors are the same as defined in `DiagnosticStatistics`
- `data` contains the factors as columns of a matrix
- `factorNames` is a cell array of the names for each factor

For a local model:

- `LocalSelection` is the local outlier selection indices or function
- `GlobalSelection` is the global outlier selection indices or function

  Outlier selection functions must conform to this prototype:

  ```
  Indices = myMfile(model, data, factorName)
  ```

  The factors are the same as appear in the scatter plot in the Model Browser.
- `data` contains the factors as columns of a matrix

- `factorNames` is a cell array of the names for each factor

Examples

```
outlierind = [1 4 6 7];
RemoveOutliers(thisRF, outlierind);
```

See Also

`RemoveOutliersForTest`

Purpose

Remove outliers on test by index or rule and refit models

Syntax

```
R = RemoveOutliers(L, TestNumber,LocalSelection,GlobalSelection);
```

Description

This is a method of the local model object, `mbcmodel.localresponse`.

All the response feature models are refitted after the local models are refitted.

`L` is the local model object.

`TestNumber` is the single test number to refit.

`LocalSelection` is either a set of indices or the name of a local outlier selection function.

`GlobalSelection` is either a set of indices or the name of a global outlier selection function.

Outlier selection functions must take the following form:

```
Indices= myMfile(model, data, factorName);
```

The factors are the same as defined in `DiagnosticStatistics`.

`data` contains the factors as columns of a matrix.

`factorNames` is a cell array of the names for each factor.

Examples

```
RemoveOutliersForTest(local, tn, indices);
```

See Also

`RemoveOutliers`

RemoveTestFilter

Purpose

Remove user-defined test filter from data set

Syntax

```
D = RemoveTestFilter(D, Index)
```

Description

This is a method of `mbcmodel.data`.

`D` is the data object.

`Index` is the input index indicating the filter to remove.

Use the property `TestFilters` to find the index of the test filter you want to remove.

Examples

```
  RemoveTestFilter(D1, 2);
```

See Also

`AddTestFilter`; `TestFilters`

Purpose

Remove user-defined variable from data set

Syntax

```
D = RemoveVariable(D, Index)
```

Description

This is a method of `mbcmodel.data`.

`D` is the data object.

`Index` is the input index indicating the variable to remove.

Use `UserVariables` to find the index of the variable you want to remove.

Examples

```
RemoveVariable(D1, 2);
```

See Also

`AddVariable`; `UserVariables`

Response

Purpose

Response for model object

Syntax

```
R = get(model, 'Response')
```

Description

This is a property of `mbcmodel.model`. It returns the response the model object came from (e.g. a response object).

If you make changes to the model object (for example by changing the model type using ModelSetup, or using StepwiseRegression) you must use UpdateResponse to return the new model object to the response in the project.

Examples

```
   R = get(knot, 'Response');
```

See Also

`UpdateResponse`; `ModelSetup`

Purpose

Array of response features for response

Syntax

```
RFs = get(L, 'ResponseFeatures')
```

Description

This is a property of the local model object, `mbcmodel.localresponse`.

`L` is the local response.

See “Understanding Model Structure” in the Getting Started documentation for an explanation of the relationships between local responses and other responses.

Examples

```
   RFs = get(local, 'ResponseFeatures');
```

ResponseSignalName

Purpose Name of signal or response feature being modeled

Syntax

```
ysignal = get(R, 'ResponseSignalName')
```

Description This is a property of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse` and `mbcmodel.response`.

`R` can be a hierarchical response, local response or response.

Examples

```
yName = get(local, 'ResponseSignalName');
```

See Also `InputSignalNames`

Purpose

Array of available responses for test plan

Syntax

```
R = get(T, 'Responses')
```

Description

This is a property of `mbcmodel.testplan`.

`T` is the test plan object.

See “Understanding Model Structure” for an explanation of the relationship between test plans and responses.

Examples

```
R = get(T, 'Responses');
```

RollbackEdit

Purpose

Undo most recent changes to data

Syntax

```
D = RollbackEdit(D)
```

Description

This is a method of mbcmodel.data. Use this if you change your mind about changes you have made to the data since you called BeginEdit, such as importing or appending data, applying filters or creating new user variables.

There are no input arguments. If for your data object D, IsBeingEdited is true, then RollbackEdit will return it to the same state as it was when BeginEdit was called. If IsEditable(D) is true then you can still modify it, if not it will revert to being read-only. See the example below.

Examples

```
D = get(P, 'Data');
BeginEdit(D);
AddVariable(D, 'TQ = tq', 'lbft');
AddFilter(D, 'TQ < 200');
DefineTestGroups(D, {'RPM' 'AFR'}, [50 10], 'MyLogNo');
RollbackEdit(D);
```

This returns the data object D to the same state as when BeginEdit was called. If the data object IsEditable then the returned object will still return true for IsBeingEdited, else it will not be editable.

For an example case where IsEditable is false and IsBeingEdited is true:

```
D  = get(p, 'Data');
D1 = get(p, 'Data');
BeginEdit(D1);
tp = get(p, 'Testplan');
Attach(tp, D);
```

Where p is an mbcmodel.project object, and D and D1 are mbcmodel.data objects.

At this point IsEditable for D1 becomes false because it is now Attached to the test plan and hence can only be modified from the test plan. However

```
OK = get (D1, 'IsBeingEdited')
```

will still be true at this point, and trying to call CommitEdit will fail.

See Also

BeginEdit; CommitEdit; IsBeingEdited

Save

Purpose

Save project to currently selected filename

Syntax

```
OK = Save(P, Name)
```

Description

This is a method of `mbcmodel.project`.

Examples

```
   OK = Save(proj, 'Example.mat');
```

See Also

`SaveAs`

Purpose

Save project to new file

Syntax

```
OK = SaveAs(P, Name)
```

Description

This is a method of `mbcmodel.project`.

Examples

```
OK = SaveAs(proj, 'Example.mat');
```

See Also

`Save`

SignalNames

Purpose

Names of signals held by data

Syntax

```
names = get (D, 'SignalNames')
```

Description

This is a property of `mbcmodel.data`.

This is a cell array of strings that hold the names of the signals within the data. These names can be used to reference the appropriate signals in the `Value` method. The subset of these names that are being used for modeling may also be found in the test plan and responses `InputSignalNames` properties.

Examples

```
  names = get (D, 'SignalNames');
```

See Also

`SignalUnits`; `InputSignalNames`; `Value`

Purpose

Names of units in data

Syntax

```
units = get(D, 'SignalUnits')
```

Description

This is a property of `mbcmodel.data`.

D is the data object.

It returns a cell array of strings holding the units of the signals.

Examples

```
  units = get(D, 'SignalUnits');
```

See Also

`SignalNames`

SizeOfParameterSet

Purpose

Number of model parameters

Syntax

```
N = get (params, 'SizeOfParameterSet')
```

Description

This is a property of `mbcmodel.linearmodelparameters`, for linear models only. It returns the total possible number of parameters in the model. Note that not all of these terms are necessarily currently included in the model, as you may remove some using StepwiseRegression.

Call NumberOfParameters to see how many terms are currently included in the model. Call StepwiseSelection to see which terms are included and excluded.

Use Names and Values to get the parameter names and values.

Examples

```
N = get (knotparams, 'SizeOfParameterSet')
```

See Also

`NumberOfParameters`; `StepwiseSelection`; `Names`; `Values`

Purpose

Model status: fitted, not fitted or best

Syntax

```
S = get(model, 'Status')
```

Description

This is a property of `mbcmodel.model`. It returns a string: ‘`Fitted'` if the model is fitted, ‘`Not fitted'` if the model is not fitted (for example there is not enough data to fit the model), or ‘`Best'` if the model has been selected as best from some alternative models. A model must be `Fitted` before it can be selected as `Best`.

Examples

```
S = get(knot, 'Status')
S =
     `Fitted'
```

See Also

`ChooseAsBest`;

Purpose

Change stepwise selection status for specified terms

Syntax

```
S = StepwiseRegression(model, optional toggleTerms)
```

Description

This is a method of `mbcmodel.model`, for linear models only. This method returns the Stepwise table (as in the Stepwise values for `ParameterStatistics`). Leave out `toggleTerms` to get the current Stepwise values. You can choose to remove or include parameters using StepwiseRegression, as long as their StepwiseStatus is `Step`.

The Stepwise values returned are the same as those found in the table in the Stepwise GUI. For each parameter, the columns are: the value of the coefficient, the standard error of the coefficient, the `t` value and Next PRESS (the value of PRESS if the status of this term is changed at the next iteration). Look for the lowest Next PRESS to indicate which terms to toggle in order to improve the predictive power of the model.

Call StepwiseRegression to toggle between `in` and `out` for particular parameters. `toggleTerms` can be either an index that specifies which parameters to toggle, or an array or logical where a true value indicates that a toggle should occur. The example shown toggles parameter 4, after inspection of the Next PRESS column indicates changing the status of this term will result in the lowest PRESS. StepwiseRegression returns the new Stepwise values after toggling a parameter.

After making changes to the model using StepwiseRegression you must call UpdateResponse.

Use StepwiseStatus (on the child modelparameters object) to see which parameters have a status of `Step`; these can be toggled between `in` and `out` using StepwiseRegression (on the parent model object).

Use StepwiseSelection (on the child modelparameters object) to view which terms are `in` and `out`, as shown in the example.

Examples

```
S = StepwiseRegression(knot)
S =
  1.0e+003 *

    0.1316    0.0606    0.0200       NaN
    0.0000    0.0000    0.0200    2.0919
    0.0000    0.0000    0.0190    0.2828
   -0.0000    0.0000    0.0190    0.2531
    0.0000    0.0000    0.0190    0.2680
   -0.0551    0.0347    0.0200    0.2566
    0.0919    0.0264    0.0200    0.3672
   -0.0040    0.0023    0.0200    0.2564
   -0.0178    0.0095    0.0200    0.2644
    0.0008    0.0004    0.0200    0.2787

S = StepwiseRegression(knot, 4)

S =

  129.8406   60.1899   19.0000       NaN
    0.0048    0.0008   19.0000  662.3830
    0.0000    0.0000   18.0000  290.8862
   -0.0021    0.0019   19.0000  245.9833
    0.0001    0.0002   18.0000  281.4104
  -50.4091   34.7401   19.0000  262.8346
   94.9675   26.3690   19.0000  400.6572
   -4.0887    2.2488   19.0000  262.6588
  -17.9412    9.4611   19.0000  276.7535
    0.8229    0.3734   19.0000  292.0827

params = get(knot, 'Parameters');
N = get (params, 'StepwiseSelection')

N =
    'in'
    'in'
    'out'
    'in'
    'out'
```

```
        'in'
        'in'
        'in'
        'in'
        'in'

>> StepwiseRegression(knot, 4);
params = get(knot, 'Parameters');
N = get (params, 'StepwiseSelection')

N =
        'in'
        'in'
        'out'
        'out'
        'out'
        'in'
        'in'
        'in'
        'in'
        'in'
```

See Also

StepwiseSelection; StepwiseStatus; UpdateResponse

Purpose

Model parameters currently included and excluded

Syntax

```
N = get (paramsknot, 'StepwiseSelection')
```

Description

This is a read-only property of `mbcmodel.linearmodelparameters`, for linear models only. It returns a status for each parameter in the model, `in` or `out`, depending on whether the term is included or excluded. You can choose to remove or include parameters using StepwiseRegression, as long as their StepwiseStatus is `Step`. Call StepwiseRegression (on the parent model object) to toggle between `in` and `out` for particular parameters. You must then call UpdateResponse before calling StepwiseSelection.

Examples

```
   N = get (paramsknot, 'StepwiseSelection')
  N =
      'in'
      'in'
      'out'
      'out'
      'out'
      'in'
      'in'
      'in'
      'in'
      'in'
```

See Also

`StepwiseRegression`; `StepwiseStatus`; `NumberOfParameters`; `UpdateResponse`

StepwiseStatus

Purpose

Stepwise status of parameters in model

Syntax

```
N = get (paramsknot, 'StepwiseStatus')
```

Description

This is a method of `mbcmodel.linearmodelparameters`, for linear models only. It returns the stepwise status of each parameter in the model.

The stepwise status for each term can be Always, Never or Step. The status determines whether you can use the StepwiseRegression function to throw away terms in order to try to improve the predictive power of the model.

- `Always` - Always included in the model
- `Never` - Never included in the model
- `Step` - You can choose whether to include or exclude this term. Do this by using StepwiseRegression to toggle between `in` and `out` for particular parameters.

Use StepwiseSelection to find out which terms are currently included and excluded.

Examples

```
N = get (paramsknot, 'StepwiseStatus')
N =
    'Always'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
```

See Also

`StepwiseRegression`; `StepwiseSelection`

Purpose

Summary statistics for response

Syntax

```
S = SummaryStatistics(R, Name)
```

Description

This is a method of all model objects: `mbcmodel.hierarchicalresponse`, `mbcmodel.localresponse`, `mbcmodel.response`, and `mbcmodel.model`.

These are the statistics that appear in the Summary Statistics pane of the Model Browser GUI.

R is the response object.

`S` is a structure array containing `Statistics` and `Names` fields for the response `R`.

`Name` is an optional input where you can specify which statistics you want. If you do not use `Name` all statistics are calculated.

Examples

```
   S = SummaryStatistics(R2);
```

See Also

`DiagnosticStatistics`; `AlternativeModelStatistics`

Purpose

Structure array holding user-defined test filters

Syntax

```
testf = get (D, 'TestFilters')
```

Description

This is a property of `mbcmodel.data`.

It returns a structure array holding information about the currently defined test filters for the data object `D`. The array will be the same length as the number of currently defined test filters, with the following fields for each filter:

- `Expression` — The string expression as defined in `AddTestFilter` or `ModifyTestFilter`.
- `AppliedOK` — Boolean indicating that the filter was successfully applied.
- `RemovedTests` — Boolean vector indicating which tests the filter removed. Note that many filters could remove the same test.
- `Message` — String holding information on the success or otherwise of the filter.

Examples

```
testf = get (D, 'TestFilters');
```

See Also

`AddTestFilter`; `ModifyTestFilter`; `RemoveTestFilter`

TestPlans

Purpose Array of test plan objects in project

Syntax

```
tps = get (P, 'TestPlans')
```

Description This is a property of `mbcmodel.project`.

`P` is the project object.

Examples

```
tps = get (P, 'TestPlans');
```

Purpose

Replace model in response

Syntax

```
UpdateResponse(model)
```

Description

This is a method of `mbcmodel.model`. This takes the model and places it back into the response it came from. Appropriate action is taken if a refit is necessary because you have modified either the model, response data or model data in the interim. For example, if you have changed the model type, the new model is fitted to the response data. If you have changed the response data (e.g. removed an outlier), the model is fitted to the new response data.

Note that when changing the model type or settings (using the ModelSetup command) the response is not refitted until you call UpdateResponse. If you have changed the model by using StepwiseRegression you must call UpdateResponse.

Examples

```
  UpdateResponse(knot);
```

See Also

`ModelSetup`

UserVariables

Purpose

Structure array holding user-defined variables

Syntax

```
userV = get(D, 'UserVariables')
```

Description

This is a property of `mbcmodel.data`.

This returns a structure array holding information about the currently defined filters. The array will be the same length as the number of currently defined variables, with fields

- `Variable` — variable name
  - `Expression` — The string expression as defined in `AddVariable` or `ModifyVariable`
  - `Units` — The string defining the units
  - `AppliedOK` — Boolean indicating that the variable expression was successfully applied
  - `Message` — String holding information on the success or otherwise of the variable

Examples

```
myvars = get(D1, 'UserVariables')
```

This returns the following information about the user-defined variable in the example data object `D1`:

```
  Variable: 'BSFC'
Expression: 'BSFC = FUELFLO./(BTQ.*(ENGSPEED*2*pi/60))'
     Units: 'kg/Nm'
 AppliedOK: 1
   Message: 'Variable successfully added'
```

`Variable` is the parsed name of the variable being added. Note that this might differ from the string used in `AddVariable` because the `SignalName` must be a valid MATLAB variable name, and hence MBC will parse and modify the input string appropriately.

See Also

`AddVariable`; `ModifyVariable`; `RemoveVariable`

Value

Purpose

Double data from data object

Syntax

```
val = Value(D, varNames, testNumbers)
```

Description

This is a method of `mbcmodel.data`.

Use this to extract particular data values.

`varNames` is an optional input that specifies either the name of the signal that you want to extract (such as `'SPK'`) or an array of names (`{'SPK' 'AFR' 'TQ'}`) the indices of the signals (`[1 4 5]`). Defaults to ':' meaning all.

`testNumbers` is an optional input that specifies which test indices you want. Defaults to ':' meaning all.

`val` outputs the double values held in the data.

Examples

```
  dblValues = Value(D, 'SPK', 1);
  dblValues = Value(D, {'SPK' 'AFR'}, ':');
  dblValues = Value(D, [1 3 4 5]);
  dblValues = Value(D, ':', [1 4 6 8]);
```

See Also

`SignalNames`

Purpose

Values of model parameters

Syntax

```
vals = get (paramsknot, 'Values')
```

Description

This is a read-only property of `mbcmodel.modelparameters`. It returns the value of each parameter in the model. Use Names to find out the names of these terms.

Examples

```
  vals = get (paramsknot, 'Values');
```

See Also

`Names`

Widths

Purpose

Width data from RBF model

Syntax

```
Width = get(params, 'Widths')
```

Description

This is a property of `mbcmodel.rbfmodelparameters`, for Radial Basis Function (RBF) models only.

Width is usually a single value, but can also be of size 1 by number of variables in the case of the width per dimension algorithm, or number of centers by number of variables in the case of tree regression.

Examples

```
  Width = get(params, 'Widths');
```

See Also

`Centers`

Purpose

X (or input) data for model

Syntax

```
D = get (model, 'XData')
```

Description

This is a property of `mbcmodel.model`. It returns an array of the input variable data currently in the model.

Examples

```
  D = get (knot, 'XData');
```

See Also

`XDataNames`; `YData`

XDataNames

Purpose X data (or input) variable names for model

Syntax

```
D = get (model, 'XDataNames')
```

Description This is a property of `mbcmodel.model`. It returns the names of the input variables in the data.

Examples

```
D = get (knot, 'XDataNames');
```

See Also `XData`

Purpose

Y (or response) data for model

Syntax

```
D = get (model, 'YData')
```

Description

This is a property of `mbcmodel.model`.

It returns an array of the response data currently in the model.

Examples

```
  D = get (knot, 'YData');
```

See Also

`XData`