Model-Based Calibration Toolbox™ 3
Reference

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
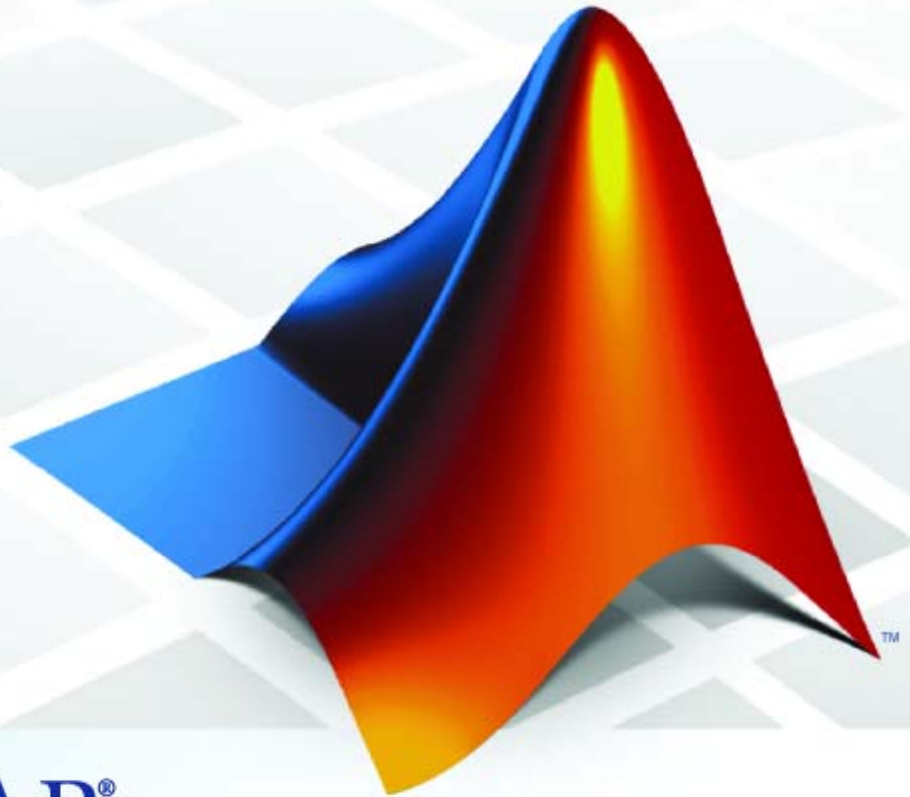
How to Contact The MathWorks

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

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

Model-Based Calibration Toolbox™ Reference

Trademarks

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

Patents

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

Revision History

Contents

Function Reference

1

2

1

Function Reference

Functions

CreateData	Create data object
CreateModel	Create new model
CreateProject	Create project object
DataFileTypes	Data file types
LoadProject	Load mbcmodel.project
modelinput	Create modelinput object

Handling Data

mbcmodel.data — Properties

Filters	Structure array holding user-defined filters
IsBeingEdited	Boolean signaling if data or model is being edited
IsEditable	Boolean signaling whether data is editable
Name	Name of object
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

`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 object
`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

BestDesign	Best design in test plan
Data	Array of data objects in project or test plan
DefaultModels	Default models for test plan
Designs	Designs in test plan
Inputs	Inputs for test plan, model, design or constraint
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 object
Responses	Array of available responses for test plan

mbcmodel.testplan — Methods

AddDesign	Add design to test plan
AttachData	Attach data from project to test plan
BoundaryModel	Get boundary model from test plan

`CreateDesign`	Create design object for test plan or model
`CreateResponse`	Create new response model for test plan
`DetachData`	Detach data from test plan
`FindDesign`	Find design by name
`InputSetupDialog`	Open Input Setup dialog box to edit inputs
`Remove`	Remove project, test plan, or model
`RemoveDesign`	Remove design from test plan
`UpdateDesign`	Update design in test plan

Handling Designs

mbcdoe.design — Properties

Constraints	Constraints in design
Generator	Design generation options
Inputs	Inputs for test plan, model, design or constraint
Model (for designs)	Model for design
Name	Name of object
NumberOfInputs	Number of model or design object inputs
NumberOfPoints	Number of design points
Points	Matrix of design points
PointTypes	Fixed and free point status

`Style`	Style of design type
`Type (for designs and generators)`	Design type

mbcdoe.design — Methods

`AddConstraint`	Add design contraint
`Augment`	Add design points
`ConstrainedGenerate`	Generate constrained space-filling design of specified size
`CreateCandidateSet`	Create candidate set for optimal designs
`CreateConstraint`	Create design contraint
`Discrepancy`	Discrepancy value
`FixPoints`	Fix design points
`Generate`	Generate new design points
`getAlternativeTypes`	Alternative model or design types
`Maximin`	Maximum of minimum of distance between design points
`Merge`	Merge designs
`Minimax`	Minimum of maximum distance between design points
`OptimalCriteria`	Optimal design criteria (V, D, A, G)
`RemovePoints`	Remove all nonfixed design points
`Scatter2D`	Plot design points

mbcdoe.generator — Properties

`NumberOfInputs`	Number of model or design object inputs
`Type (for designs and generators)`	Design type

mbcdoe.generator — Methods

`getAlternativeTypes`	Alternative model or design types
`Properties (for design generators)`	View and edit design generator properties

mbcdoe.candidateset — Properties

`NumberOfInputs`	Number of model or design object inputs
`Type (for candidate sets)`	Candidate set type

mbcdoe.candidateset — Methods

`getAlternativeTypes`	Alternative model or design types
`Properties (for candidate sets)`	View and edit candidate set properties

mbcdoe.designconstraint — Properties

`Inputs`	Inputs for test plan, model, design or constraint
`Name`	Name of object

NumberOfInputs	Number of model or design object inputs
Type (for design constraints)	Design constraint type

mbcdoe.designconstraint — Methods

Evaluate	Evaluate model or design constraint
getAlternativeTypes	Alternative model or design types
MatchInputs	Match design constraint inputs
Properties (for design constraints)	View and edit design constraint properties

Handling Models

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 object
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 object
NumberOfTests	Total number of tests being used in model
ResponseFeatures(Local Response)	Array of response features for local 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
`CreateResponseFeature`	Create new response feature for local model
`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
`ModelForTest`	Model for specified test
`OutlierIndices`	Indices of `DoubleInputData` marked as outliers
`OutlierIndicesForTest`	Indices marked as outliers for test
`PEV`	Predicted error variance of model at specified inputs
`PEVForTest`	Local model predicted error variance for test
`PredictedValue`	Predicted value of model at specified inputs
`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
RestoreData	Restore removed outliers
RestoreDataForTest	Restore removed outliers for test
SummaryStatistics	Summary statistics for response
SummaryStatisticsForTest	Statistics for specified test
UpdateResponseFeatures	Refit response feature models

mbcmodel.localmodel — Properties

LocalModel Properties	Edit local model properties
ResponseFeatures(Local Model)	Set of response features for local model

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 Object	Model object within response object
Name	Name of object
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
`RestoreData`	Restore removed outliers
`SummaryStatistics`	Summary statistics for response

Models

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

mbcmodel.model — Properties

`FitAlgorithm`	Fit algorithm for model
`InputData`	Input data for model
`Inputs`	Inputs for test plan, model, design or constraint
`IsBeingEdited`	Boolean signaling if data or model is being edited
`NumberOfInputs`	Number of model or design object inputs
`OutputData`	Output (or response) data for model
`Parameters`	Model parameters
`Properties (for models)`	View and edit model properties
`Response`	Response for model object
`Status`	Model status: fitted, not fitted or best
`Type (for models)`	Valid model types
`Units`	Model output units

mbcmodel.linearmodel — Linear Model Methods

`AliasMatrix`	Alias matrix for linear model parameters
`BoxCoxSSE`	SSE and confidence interval for Box-Cox transformations
`Correlation`	Correlation matrix for linear model parameters
`Covariance`	Covariance matrix for linear model parameters
`MultipleVIF`	Multiple VIF matrix for linear model parameters

`ParameterStatistics`	Calculate parameter statistics for linear model
`PartialVIF`	Partial VIF matrix for linear model parameters
`SingleVIF`	Single VIF matrix for linear model parameters
`StepwiseRegression`	Change stepwise selection status for specified terms

mbcmodel.model — Methods

`CreateDesign`	Create design object for test plan or model
`Evaluate`	Evaluate model or design constraint
`Export`	Make command-line or Simulink export model
`Fit`	Fit model to new or existing data, and provide summary statistics
`getAlternativeTypes`	Alternative model or design types
`InputSetupDialog`	Open Input Setup dialog box to edit inputs
`Jacobian`	Calculate Jacobian matrix for model at existing or new X points
`ModelSetup`	Open 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
`StatisticsDialog`	Open summary statistics dialog box

SummaryStatistics	Summary statistics for response
UpdateResponse	Replace model in response

mbcmodel.fitalgorithm — Methods

An `mbcmodel.fitalgorithm` object is contained within the `Properties` property of an `mbcmodel.model` object.

CreateAlgorithm	Create algorithm
getAlternativeNames	List alternative algorithm names
IsAlternative	Test alternative fit algorithm
SetupDialog	Open fit algorithm setup dialog box

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

Model Properties

mbcmodel.linearmodelproperties — Methods

`GetAllTerms`	List all model terms
`GetIncludedTerms`	List included model terms
`SetTermStatus`	Set status of model terms

Commands — Alphabetical List

AddConstraint

Purpose	Add design contraint
Syntax	D = AddConstraint(D,c)
Description	AddConstraint is a method of mbcdoe.design. D = AddConstraint(D,c) adds constraint c to the design. You must call AddConstraint to apply the constraint and remove points outside the constraint.
See Also	CreateConstraint

Purpose	Add design to test plan
Syntax	D = AddDesign(T,D) D = AddDesign(T,Level,D) D = AddDesign(T,Level,D,Parent)
Description	AddDesign is a method of mbcmodel.testplan.

D = AddDesign(T,D)

D = AddDesign(T,Level,D)

D = AddDesign(T,Level,D,Parent)

D is the array of designs to be added to the test plan, T.

Level is the test plan level. By default the level is the outer level (i.e., Level 1 for One-stage, Level 2 (global) for Two-stage).

Parent is the parent design in the design tree. By default designs are added to the top level of the design tree. See Designs for more information on the design tree.

In order to ensure that the design names are unique in the test plan, the design name will be changed when adding a design to a test plan if a design of the same name already exists. The array of designs with modified names is an output.

Examples	To add three designs to the test plan global (2) level:

```
D = AddDesign(TP, [sfDesign, parkedCamsDesign, mainDesign])
```

See Also	UpdateDesign; RemoveDesign; FindDesign

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   Alias matrix for linear model parameters

Syntax   `A = M.AliasMatrix`

Description   This is a method of `mbcmodel.linearmodel`.

A = `M.AliasMatrix` calculates the alias matrix for the linear model parameters (where `M` is a linear model).

Examples   `A = AliasMatrix(knot_model)`

See Also   `ParameterStatistics`

AlternativeModelStatistics

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

AlternativeResponses

Purpose   Array of alternative responses for this response

Syntax   altR = 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 = testplan.Responses;
TQ = R(1);
AR = TQ.AlternativeResponses;
```

See Also   LocalResponses, ResponseFeatures(Local Response)

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

AttachData

| **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 If the testplan has responses set up the models are 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

When you attach data to a test plan the Name property of the test plan inputs is used to select data channels. If the Name is empty then the Symbol is used as the Name. If the Name does not exist in the data set, an error is generated.

When a test plan has data attached, it is only possible to change the symbols, ranges or nonlinear transforms of the test plan inputs.

You can use AttachData to use data from one project in another project, as follows:

```
p1 = mbcmodel.LoadProject( filename );
p2 = mbcmodel.LoadProject( filename2 );
p1.Testplan.AttachData( p2.Data );
```

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, CreateData, DetachData

Augment

Purpose   Add design points

Syntax   ```
                 D = Augment(D,Numpoints)
                 D = Augment(D,'Prop1',value1,...)
                 ```

**Description**  Augment is a method of mbcdoe.design. Use it to add points to a design using a specified design generator. After augmenting a design, the design Style is set to Custom unless an optimal design is used for augmentation, as in the Design Editor.

D = Augment(D,Numpoints) augments the design with the number of points specified by Numpoints using the current generator settings.

D = Augment(D,'Prop1',value1,...) augments the design with the generator specified by the generator property value pairs.

You can use the Augment method to add points to an existing type using a different design type.

```
OptDesign = Augment(OptDesign,...
    'Type','V-optimal',...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',20);
```

To set all designs points to fixed and then augment an existing design optimally, use the FixPoints method to fix all the points as follows:

```
OptDesign = FixPoints(OptDesign);
OptDesign = Augment(OptDesign,...
    'Type','V-optimal',...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',20);
```

When augmenting with an optimal design generator existing points which are not fixed may be changed. To add points optimally and keep only fixed points, use RemovePoints before augmenting, e.g.,

```
OptDesign = RemovePoints(OptDesign,'free');
OptDesign = Augment(OptDesign,...
    'Type','V-optimal',...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',20);
```

To get a candidate set object for use with an optimal design:

```
C = CreateCandidateSet(OptDesign,'Type', 'Grid',...
        'NumberOfLevels',[21 21 21]);
```

You see an error if you try to call Augment when the design Style is User-defined or Experimental data.

Examples   To create a candidate set and then optimally augment a design with 10 points:

```
CandidateSet = augmentedDesign.CreateCandidateSet( 'Type', 'Grid' )
CandidateSet.NumberOfLevels = [21 21 21 21];
augmentedDesign = Augment( augmentedDesign,...
    'Type', 'V-optimal',...
    'NumberOfPoints', 10,...
    'CandidateSet', CandidateSet,...
    'MaxIterations', 200,...
    'NoImprovement', 50 );
```

See Also   Generate; CreateCandidateSet

BeginEdit

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   Best design in test plan

Syntax   `T.BestDesign{Level} = d;`

Description   `BestDesign` is a property of `mbcdmodel.testplan`.

`T.BestDesign{Level} = d;` sets d as the best design, where `Level` is the test plan level. There can be one best design for each level, but the best global (2) level design is used for matching to data when you call `AttachData`.

`BestDesign` is a cell array with a cell per level. `TP.BestDesign{1}` is the best design for the first level and `TP.BestDesign{2}` is best design for the second level.

Examples   To set the design `globalDesign` as the best design at the global (2) level:

```
T.BestDesign{2} = globalDesign
```

See Also   `CreateDesign`

BoundaryModel

Purpose	Get boundary model from test plan
Syntax	DC = BoundaryModel (T) DC = BoundaryModel (T, Type)
Description	BoundaryModel is a method of mbcmodel.testplan.

DC = BoundaryModel (T) returns the best boundary model for the test plan, T. DC is a mbcdoe.designconstraint object.

DC = BoundaryModel (T, Type) is the best boundary model for the specified type associated with the test plan.

Type can be any of the following values:

- 'all' : Best boundary model for all inputs (default)
- 'local' : Best local boundary model
- 'global' : Best global boundary model

Examples   To load boundary constraints from another project file and add to design:

```
otherProject = mbcmodel.LoadProject( [matlabroot,'\toolbox\...
mbc\mbctraining\Gasoline_project.mat']);
boundaryConstraints = otherProject.Testplans(1).BoundaryModel('global'
Design.Constraints = boundaryConstraints;
```

Purpose   SSE and confidence interval for Box-Cox transformations

Syntax
```
[sse, ci, lambda] = BoxCoxSSE(Model, lambda)
[sse, ci, lambda] = BoxCoxSSE(Model)
BoxCoxSSE(Model, ...)
```

Description   This is a method of mbcmodel.linearmodel.

[sse, ci, lambda] = BoxCoxSSE(Model, lambda) computes the sum of squares error (sse) and confidence interval (ci) for values of the model under different Box-Cox transforms (as given by the parameter lambda). The data used is that which was used to fit the model. sse is a vector the same size as lambda and ci is a scalar. There is no statistical difference between the Box-Cox transforms where sse less than ci.

[sse, ci, lambda] = BoxCoxSSE(Model) If lambda is not specified, then default values for are used and these are returned in third output argument.

BoxCoxSSE(Model, ...) If no output arguments are requested then a plot of SSE versus lambda is displayed. The confidence intervals are also displayed on this plot.

Examples   To try several different values, of the Box-Cox parameter and plot the results:

```
lambda = -3:0.5:3;
[sse, ci] = BoxCoxSSE( M, lambda);
semilogy( lambda, sse, 'bo-', lambda([1,end]), [ci, ci], 'r--' );
xlabel( 'Box-Cox parameter, \lambda' );
ylabel( 'SSE' );
```

Note that BoxCoxSSE does not set a Box-Cox transform in the model. To do this use:

```
M.Properties.BoxCox = O;
[S,M] = M.Fit;
```

See Also   ParameterStatistics

Centers

Purpose   Centers of RBF model

Syntax   centers = 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 = 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 = 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  = p.Data;
D1 = p.Data;
BeginEdit(D1);
tp = 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 = 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

ConstrainedGenerate

Purpose   Generate constrained space-filling design of specified size

Syntax   ```
design = ConstrainedGenerate(design, NumPoints,
    'UnconstrainedSize', Size, 'MaxIter', NumIterations)
design = ConstrainedGenerate(design, NumPoints, OPTIONS)
```

Description ConstrainedGenerate is a method of mbcdoe.design. Use it to
generate a space-filling design of specified size within the constrained
region. This method only works for space-filling designs. It may not
be possible to achieve a specified number of points, depending on the
generator settings and constraints.

design = ConstrainedGenerate(design, NumPoints,
'UnconstrainedSize', Size, 'MaxIter', NumIterations) tries to
generate a design with the number of constrained points specified by
NumPoints. You can supply parameter value pairs for the options or
you can use a structure:design = ConstrainedGenerate(design,
NumPoints, OPTIONS).

- MaxIter — Maximum iterations. Default: 10
- UnconstrainedSize — Total number of points in unconstrained
  design. Default: NumPoints

The algorithm ConstrainedGenerate produces a sequence of calls to
Generate, and updates the UnconstrainedSize using the following
formula:

```
UnconstrainedSize = ceil(UnconstrainedSize * NumPoints/D.NumberOfPoints
```

Examples   With ConstrainedGenerate, make a 200 point design, using an existing
space-filling design sfDesign, and inspect the constrained and total
number of points:

```
sfDesign = ConstrainedGenerate( sfDesign, 200, 'UnconstrainedSize', 80(

% How did we do?
finalNumberOfPoints = sfDesign.NumberOfPoints
```

```
% How many points did we need in total?
totalNumberOfPoints = sfDesign.Generator.NumberOfPoints

finalNumberOfPoints =
   200
totalNumberOfPoints =
   839
```

See Also   CreateConstraint; Generate

Constraints

Purpose   Constraints in design

Syntax   Constraints = D.Constraints

Description   Constraints is a property of mbcdoe.design.

Constraints = D.Constraints Designs have a Constraints property,
initially this is empty:

```
constraints = design.Constraints

constraints =
OxO array of mbcdoe.designconstraint
```

Use CreateConstraint to form constraints.

See Also   CreateConstraint; AddConstraint

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

Correlation

Purpose   Correlation matrix for linear model parameters

Syntax   STATS = Correlation(LINEARMODEL)

Description   This is a method of mbcmodel.linearmodel.

STATS = Correlation(LINEARMODEL) calculates the correlation matrix
for the linear model parameters.

Examples   Stats = Correlation(knot_model)

See Also   ParameterStatistics

Purpose   Covariance matrix for linear model parameters

Syntax   `STATS = Covariance(LINEARMODEL)`

Description   This is a method of `mbcmodel.linearmodel`.

STATS = `Covariance(LINEARMODEL)` calculates the covariance matrix for the linear model parameters.

Examples   `Stats = Covariance(knot_model)`

See Also   `ParameterStatistics`

CreateAlgorithm

Purpose   Create algorithm

Syntax   `newalg = alg.CreateAlgorithm( AlgorithmName)`

Description   This is a method of `mbcmodel.fitalgorithm`.

`newalg = alg.CreateAlgorithm( AlgorithmName)` creates an algorithm of the specified type. `alg` is a `mbcmodel.fitalgorithm` object. `AlgorithmName` must be in the list of alternative algorithms given by `alg.getAlternativeNames`.

To change the fit algorithm for a model:

```
>> mdl = mbcmodel.CreateModel('Polynomial', 2);
>> minpress = mdl.FitAlgorithm.CreateAlgorithm('Minimize PRESS');
>> mdl.FitAlgorithm = minpress;
```

The `AlgorithmName` determines what properties you can set. You can display the properties for an algorithm as follows:

```
>> mdl.FitAlgorithm.properties

Algorithm: Minimize PRESS
Alternatives: 'Least Squares','Forward Selection','Backward
Selection','Prune'
     MaxIter: Maximum Iterations (int: [1,1000])
```

The following sections list the properties available for each algorithm type.

Linear Model Algorithm Properties

Linear Models Algorithms

Used by polynomials, hybrid splines and as the StepAlgorithm for RBF algorithms.

Algorithm: Least Squares

Alternatives: `'Minimize PRESS','Forward Selection','Backward Selection','Prune'`

Algorithm: Minimize PRESS

Alternatives: `'Least Squares','Forward Selection','Backward Selection','Prune'`

- MaxIter: Maximum Iterations (int: [1,1000])

Algorithm: Forward Selection

Alternatives: `'Least Squares','Minimize PRESS','Backward Selection','Prune'`

- ConfidenceLevel: Confidence level (%) (numeric: [70,100])
- MaxIter: Maximum Iterations (int: [1,1000])
- RemoveAll: Remove all terms first (Boolean)

Algorithm: Backward Selection

Alternatives: `'Least Squares','Minimize PRESS','Forward Selection','Prune'`

- ConfidenceLevel: Alpha (%) (numeric: [70,100])
- MaxIter: Maximum Iterations (int: [1,1000])
- IncludeAll: Include all terms first (Boolean)

Algorithm: Prune

Alternatives: `'Least Squares','Minimize PRESS','Forward Selection','Backward Selection'`

- Criteria (PRESS RMSE|RMSE|GCV|Weighted PRESS|-2logL|AIC|AICc|BIC|R^2|R^2 adj|PRESS R^2|DW|Cp|cond(J))
- MinTerms: Minimum number of terms (int: [0,Inf])
- Tolerance (numeric: [0,1000])

- IncludeAll: Include all terms before prune (Boolean)

- Display (Boolean)

RBF Algorithm Properties

For information about any of the RBF and Hybrid RBF algorithm properties, see "Radial Basis Functions", and especially "Fitting Routines" in the Model Browser User's Guide.

Algorithm: RBF Fit

- WidthAlgorithm: Width selection algorithm (mbcmodel.fitalgorithm)

- StepAlgorithm: Stepwise (mbcmodel.fitalgorithm)

Width Selection Algorithms

Alternatives: `'WidPerDim'`,`'Tree Regression'`

Algorithm: TrialWidths

- NestedFitAlgorithm: Lambda selection algorithm (mbcmodel.fitalgorithm)

- Trials: Number of trial widths in each zoom (int: [2,100])

- Zooms: Number of zooms (int: [1,100])

- MinWidth: Initial lower bound on width (numeric: [2.22045e-016,1000])

- MaxWidth: Initial upper bound on width (numeric: [2.22045e-016,100])

- PlotFlag: Display plots (Boolean)

- PlotProgress: Display fit progress (Boolean)

Algorithm: WidPerDim

Alternatives: `'TrialWidths'`,`'Tree Regression'`

- NestedFitAlgorithm: Lambda selection algorithm (mbcmodel.fitalgorithm)

- DisplayFlag: Display (Boolean)

- MaxFunEvals: Maximum number of test widths (int: [1,1e+006])

- PlotProgress: Display fit progress (Boolean)

Algorithm: Tree Regression

Alternatives: `'TrialWidths','WidPerDim'`

- MaxNumRectangles: Maximum number of panels (int: [1,Inf])

- MinPerRectangle: Minimum data points per panel (int: [2,Inf])

- RectangleSize: Shrink panel to data (Boolean)

- AlphaSelectAlg: Alpha selection algorithm (mbcmodel.fitalgorithm)

Lambda Selection Algorithms
Algorithm: IterateRidge

Alternatives: `'IterateRols','StepItRols'`

- CenterSelectionAlg: Center selection algorithm (mbcmodel.fitalgorithm)

- MaxNumIter: Maximum number of updates (int: [1,100])

- Tolerance: Minimum change in log10(GCV) (numeric: [2.22045e-016,1])

- NumberOfLambdaValues: Number of initial test values for lambda (int: [0,100])

- CheapMode: Do not reselect centers for new width (Boolean)

- PlotFlag: Display (Boolean)

Algorithm: IterateRols

Alternatives: `'IterateRidge','StepItRols'`

CreateAlgorithm

- CenterSelectionAlg: Center selection algorithm (mbcmodel.fitalgorithm)

- MaxNumIter: Maximum number of iterations (int: [1,100])

- Tolerance: Minimum change in log10(GCV) (numeric: [2.22045e-016,1])

- NumberOfLambdaValues: Number of initial test values for lambda (int: [0,100])

- CheapMode: Do not reselect centers for new width (Boolean)

- PlotFlag: Display (Boolean)

Algorithm: StepItRols

Alternatives: `'IterateRidge','IterateRols'`

- MaxCenters: Maximum number of centers (evalstr)

- PercentCandidates: Percentage of data to be candidate centers (evalstr)

- StartLambdaUpdate: Number of centers to add before updating (int: [1,Inf])

- Tolerance: Minimum change in log10(GCV) (numeric: [2.22045e-016,1])

- MaxRep: Maximum number of times log10(GCV) change is minimal (int: [1,100])

Center Selection Algorithms
Algorithm: Rols

Alternatives: `'RedErr','WiggleCenters','CenterExchange'`

- MaxCenters: Maximum number of centers (evalstr)

- PercentCandidates: Percentage of data to be candidate centers (evalstr)

- Tolerance: Regularized error tolerance (numeric: [2.22045e-016,1])

Algorithm: RedErr

Alternatives: `'Rols'`,`'WiggleCenters'`,`'CenterExchange'`

- MaxCenters: Number of centers (evalstr)

Algorithm: WiggleCenters

Alternatives: `'Rols'`,`'RedErr'`,`'CenterExchange'`

- MaxCenters: Number of centers (evalstr)
- PercentCandidates: Percentage of data to be candidate centers (evalstr)

Algorithm: CenterExchange

Alternatives: `'Rols'`,`'RedErr'`,`'WiggleCenters'`

- MaxCenters: Number of centers (evalstr)
- NumLoops: Number of augment/reduce cycles (int: [1,Inf])
- NumAugment: Number of centers to augment by (int: [1,Inf])

Tree Regression Algorithms
Algorithm: Trial Alpha

Alternatives: `'Specify Alpha'`

- AlphaLowerBound: Initial lower bound on alpha (numeric: [2.22045e-016,Inf])
- AlphaUpperBound: Initial upper bound on alpha (numeric: [2.22045e-016,Inf])
- Zooms: Number of zooms (int: [1,Inf])
- Trials: Trial alphas per zoom (int: [2,Inf])

- Spacing: Spacing (Linear|Logarimthic)
- CenterSelectAlg: Center selection algorithm (mbcmodel.fitalgorithm)

Algorithm: Specify Alpha

Alternatives: `'Trial Alpha'`

- Alpha: Width scale parameter, alpha (numeric: [2.22045e-016,Inf])
- NestedFitAlgorithm: Center selection algorithm (mbcmodel.fitalgorithm)

Algorithm: Tree-based Center Selection

Alternatives: `'Generic Center Selection'`

- ModelSelectionCriteria: Model selection criteria (BIC|GCV)
- MaxNumberCenters: Maximum number of centers (evalstr)

Algorithm: Generic Center Selection

Alternatives: `'Tree-based Center Selection'`

- CenterSelectAlg: Center selection algorithm (mbcmodel.fitalgorithm)

Hybrid RBF Algorithms
Algorithm: RBF Fit

- WidthAlgorithm: Width selection algorithm (mbcmodel.fitalgorithm)
- StepAlgorithm: Stepwise (mbcmodel.fitalgorithm)

Width Selection Algorithms
Algorithm: TrialWidths

- NestedFitAlgorithm: Lambda and term selection algorithm (mbcmodel.fitalgorithm)
- Trials: Number of trial widths in each zoom (int: [2,100])

- Zooms: Number of zooms (int: [1,100])

- MinWidth: Initial lower bound on width (numeric: [2.22045e-016,1000])

- MaxWidth: Initial upper bound on width (numeric: [2.22045e-016,100])

- PlotFlag: Display plots (Boolean)

- PlotProgress: Display fit progress (Boolean)

Nested Fit Algorithms
Algorithm: Twostep

Alternatives: `'Interlace'`

- MaxCenters: Maximum number of centers (evalstr)

- PercentCandidates: Percentage of data to be candidate centers (evalstr)

- StartLambdaUpdate: Number of terms to add before updating (int: [1,Inf])

- Tolerance: Minimum change in log10(GCV) (numeric: [2.22045e-016,1])

- MaxRep: Maximum number of times log10(GCV) change is minimal (int: [1,100])

- PlotFlag: Display (Boolean)

Algorithm: Interlace

Alternatives: `'Twostep'`

- MaxParameters: Maximum number of terms (evalstr)

- MaxCenters: Maximum number of centers (evalstr)

- PercentCandidates: Percentage of data to be candidate centers (evalstr)

- StartLambdaUpdate: Number of terms to add before updating (int: [1,Inf])

- Tolerance: Minimum change in log10(GCV) (numeric: [2.22045e-016,1])

- MaxRep: Maximum number of times log10(GCV) change is minimal (int: [1,100])

Examples   First get a fitalgorithm object, F, from a model:

```
M = mbcmodel.CreateModel('Polynomial', 4);
F = M.FitAlgorithm

F =
Algorithm: Least Squares
Alternatives: 'Minimize PRESS','Forward Selection','Backward
Selection','Prune'
1x1 struct array with no fields.
```

Then, to create a new algorithm type:

```
Alg = CreateAlgorithm(F, 'Minimize PRESS')

Alg =
Algorithm: Minimize PRESS
Alternatives: 'Least Squares','Forward Selection','Backward
Selection','Prune'
    MaxIter: 50
```

See Also   getAlternativeNames, SetupDialog

Purpose   Create alternative models from model template

Syntax   R = CreateAlternativeModels(R, modeltemplate, criteria)
R = CreateAlternativeModels(R, modellist, 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. You can use a model template
    file (.mbm) created in the Model Browser, or a cell array of
    mbcmodel.model objects.

  - 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'.

CreateAlternativeModels

- GlobalModels is the list of global models (from the model template).

- GlobalCriteria is the selection criteria for best model.

You construct a model template file (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.

```
CreateAlternativeModels( RESPONSE, 'alternative_models.mbm', 'Weighted PRESS' )
```

creates alternative response feature models based upon the model template file alternative_models.mbt, and chooses the best model based upon each model's Weighted PRESS statistic.

See Also   AlternativeModelStatistics

Purpose	Create candidate set for optimal designs
Syntax	D = CreateCandidateSet(D) D = CreateCandidateSet(D,prop1,value1,...)
Description	CreateCandidateSet is a method of mbcdoe.design. Candidate sets are very similar to design generators. They are not used directly in specifying a design but are used to specify the set of all possible points to be considered as part of an optimal design. You obtain the candidate set from an optimal design generator or by using mbcdoe.design.CreateCandidateSet. D = CreateCandidateSet(D) creates a candidate set (mbcdoe.candidateset object) for the design. D = CreateCandidateSet(D,prop1,value1,...) creates a candidate set with the specified properties for the design. To see the properties you can set, see the table of candidate set properties, Candidate Set Properties (for Optimal Designs) on page 2-158.
Examples	CandidateSet = augmentedDesign.CreateCandidateSet('Type',... 'Grid'); CandidateSet.NumberOfLevels = [21 21 21 21];
See Also	Properties (for candidate sets); Augment

CreateConstraint

Purpose	Create design contraint
Syntax	`c = CreateConstraint(D)` `c = CreateConstraint(D,prop1,val1,...)`
Description	`CreateConstraint` is a method of `mbcdoe.design`.

Designs have a Constraints property, initially this is empty:

```
constraints = design.Constraints

constraints =
0x0 array of mbcdoe.designconstraint
```

Use CreateConstraint to form constraints.

`c = CreateConstraint(D)` creates a default constraint for the design.

`c = CreateConstraint(D,prop1,val1,...)` creates a constraint with the specified properties. See Constraint Properties on page 2-161.

By default a 1D table constraint is created for designs with two or more inputs.

For a design with one input a linear constraint is created by default.

You can specify the constraint type during creation by using the `Type` property, e.g.,

```
c = D.CreateConstraint('Type','Linear')
```

Other available properties depend on the design type. See the table Constraint Properties on page 2-161.

This method does not add the constraint to the design. You must explicitly add the constraint to the design using the Constraints property of the design e.g.,

```
D= AddConstraint(D,c)
```

or

```
D.Constraints(end+1) = c;
```

You must call `AddConstraint` to apply the constraint and remove design points outside the constraint.

Examples   To create a `Linear` constraint, add it to a design, and regenerate the design points:

```
cLinear = design.CreateConstraint( 'Type', 'Linear' );
cLinear.A = [-2.5e-4, 1];
cLinear.b = 0.25;
cLinear
design.Constraints = cLinear;
design = Generate(design);
```

To create and apply a `1D Table` constraint:

```
cTable1d = design.CreateConstraint( 'Type', '1D Table' );
cTable1d.Table = [0.9 0.5];
cTable1d.Breakpoints = [500 6000];
cTable1d
design.Constraints = cTable1d;
design = Generate(design);
```

To combine constraints, use an array of the constraints you want to apply:

```
design.Constraints = [cLinear, cTable1d];
constraints = design.Constraints
design = Generate(design);

constraints =
1x2 array of mbcdoe.designconstraint
Linear design constraint: -0.00025*N + 1*L <= 0.25
1D Table design constraint: L(N) <= Lmax
```

To load boundary constraints from another project file and add to design:

CreateConstraint

```
otherProject = mbcmodel.LoadProject( [matlabroot,'\toolbox\...
mbc\mbctraining\Gasoline_project.mat']);
boundaryConstraints = otherProject.Testplans(1).BoundaryModel...
('global');
Design.Constraints = boundaryConstraints;
```

See Also   Properties (for design constraints); AddConstraint

Purpose Create data object

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

Description The first syntax is a method of `mbcmodel.project`. Use this to create a new data object in an existing project. `P` is the project object.

filename and filetype are optional arguments that are used to load data from a file into the new data object at creation time.

`filename` is a string specifying the full path to the file.

`filetype` is a string specifying the file type. See `DataFileTypes` for the specification of allowed file types (and `mbccheckindataloadingfcn` to specify your own data loading function). If `filetype` is not provided, then MBC will attempt to infer the file type from the file extension, i.e. if the file extension is .xls then MBC will try the Excel File Loader.

If `filename` is not provided then no data will be loaded into the new data object. Data can be loaded subsequently using `ImportFromFile`, provided that editing of the data object has been enabled via a call to `BeginEdit`. Call `CommitEdit` to apply edits.

If you create the data object specifying a `filename`, then the Name property is set to the filename. However, if you use ImportFromFile after creation to load data from a file, the name of the data object does not change.

The second syntax is a function. Use this to create a new data object independent of any project. You can use AttachData to use the data object in another test plan, e.g.,

```
d = mbcmodel.CreateData( filename );
testplan.AttachData( d );
```

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

CreateData

Where P is an mbcmodel.project object.

See Also DataFileTypes, BeginEdit, CopyData, RemoveData, Data, ImportFromFile, CommitEdit, AttachData

Purpose   Create design object for test plan or model

Syntax   ```
D = CreateDesign(Testplan)
D = CreateDesign(Testplan,Level)
D = CreateDesign(Testplan,Level,prop1,value1,...)
D = CreateDesign(Model)
D = CreateDesign(Model,prop1,value1,...)
D = CreateDesign(Inputs)
D = CreateDesign(Inputs,prop1,value1,...)
D = CreateDesign(Design)
```

**Description**    CreateDesign is a method of mbcmodel.testplan, mbcmodel.model, and mbcmodel.modelinput. Property value pairs can be specified at creation time. The property value pairs are properties of mbcdoe.design.

### Properties of mbcdoe.design

| mbcdoe.design Property | Description |
| --- | --- |
| Constraints | Constraints in design. |
| Generator | Design generation options. |
| Inputs | Inputs for design. |
| Model | Model for design. |
| Points | Matrix of design points. |
| PointTypes | Fixed and free point status. |
| Style | Style of design type. |
| NumberOfInputs | Read-only — Number of model inputs. |

**Properties of** `mbcdoe.design` **(Continued)**

| mbcdoe.design Property | Description |
|---|---|
| NumberOfPoints | Read-only — Number of design points. |
| Type | Design type. The design property Type can *only* be specified with CreateDesign and is subsequently read-only for design objects. |

`D = CreateDesign(Testplan)` creates a design for the test plan, where `Testplan` is an `mbcmodel.testplan` object.

`D = CreateDesign(Testplan,Level)` creates a design for the specified level of the test plan. By default the level is the outer level (i.e., Level 1 for one-stage, Level 2 (global) for two-stage).

If you do not specify any properties, the method creates a default design type. The default design types are a Sobol Sequence for two or more inputs, and a Full Factorial for a single input.

`D = CreateDesign(Testplan,Level,prop1,value1,...)` creates a design with the specified properties.

`D = CreateDesign(Model)` creates a design based on the inputs of the `mbcmodel.model` object, `Model`.

`D = CreateDesign(Model,prop1,value1,...)` creates a design with the specified properties based on the inputs of the model.

`D = CreateDesign(Inputs)` creates a design based on the inputs of the `mbcmodel.modelinput` object, `Inputs`.

`D = CreateDesign(Inputs,prop1,value1,...)` creates a design with the specified properties based on the inputs.

`D = CreateDesign(Design)` creates a copy of an existing design.

**Examples**    To create a space-filling design for a test plan TP:

```
sfDesign = CreateDesign(TP, ...
    'Type', 'Latin Hypercube Sampling',...
    'Name', 'Space Filling');
```

Create an optimal design based on the inputs of a model:

```
optimalDesign = CreateDesign( model,...
    'Type', 'V-optimal',...
    'Name', 'Optimal Design' );
```

Create a classical full factorial design based on the inputs defined by a mbcmodel.modelinput object:

```
design = CreateDesign( inputs, 'Type', 'Full Factorial' );
```

Create a new design based on an existing design (ActualDesign) in order to augment it:

```
augmentedDesign = ActualDesign.CreateDesign('Name',...
 'Augmented Design');
```

Create a local level design for the two-stage test plan TP:

```
localDesign = TP.CreateDesign(1,'Type',...
 'Latin Hypercube Sampling');
```

Create a global level design for the two-stage test plan TP:

```
globalDesign = TP.CreateDesign(2, 'Type',...
 'Latin Hypercube Sampling');
```

See Also   Generate; modelinput

CreateModel

Purpose	Create new model
Syntax	`M = mbcmodel.CreateModel(Type, INPUTS)` `NewModel = CreateModel(model,Type)`

Description   `M = mbcmodel.CreateModel(Type, INPUTS)` This syntax is a function that creates an `mbcmodel.model` object of the specified `Type`.

`mbcmodel.linearmodel` and `mbcmodel.localmodel` are subclasses of `mbcmodel.model`. Model types that begin with the word "local" specify an `mbcmodel.localmodel` object.

`NewModel = CreateModel(model,Type)` This syntax is a function that creates a new model (of the specified Type) with the same inputs as an existing `model`. `model` is an `mbcmodel.model` object. You can use getAlternativeTypes to generate a list of valid model types. See `Type` (for models) for a list of valid model types. Spaces and case in `Type` are ignored.

`INPUTS` can be a `mbcmodel.modelinput` object, or any valid input to the `mbcmodel.modelinput` constructor. See `modelinput`.

Examples   To create a hybrid spline with four input factors, enter:

```
M = mbcmodel.CreateModel('Hybrid Spline', 4)
```

To create an RBF with four input factors, enter:

```
Inputs = mbcmodel.modelinput('Symbol',{'N','L','EXH','INT'}',...
    'Name',{'ENGSPEED','LOAD','EXHCAM','INTCAM'}',...
    'Range',{[800 5000],[0.1 1],[-5 50],[-5 50]}');

RBFModel = mbcmodel.CreateModel( 'RBF', Inputs);
```

To create a polynomial with the same input factors as the previously created RBF, enter:

```
PolyModel = CreateModel(RBFModel,'Polynomial')
```

See Also   getAlternativeTypes, modelinput, CreateProject, CreateData, Type
(for models)

CreateProject

Purpose	Create project object
Syntax	`P = mbcmodel.CreateProject`
Description	This is a function that creates an `mbcmodel.project` object.
	P is the project object.
	`P = mbcmodel.CreateProject` creates an `mbcmodel.project` called Untitled. `P = mbcmodel.CreateProject( NAME )` creates an `mbcmodel.project` called NAME.
Examples	`P = mbcmodel.CreateProject;`
	Create a project called MBT_Project:
	`P = mbcmodel.CreateProject( 'MBT_Project' );`

Purpose   Create new response model for test plan

Syntax   R = CreateResponse(T, Varname)
                  R = CreateResponse(T, Varname, Model)
                  R = CreateResponse(T, Varname, LocalModel, GlobalModel)
                  R = CreateResponse(T, Varname, LocalModel, GlobalModel,
                     DatumType)

Description   This is a method of mbcmodel.testplan.

                  R = CreateResponse(T, Varname) creates a model of the variable
                  Varname using the test plan's one- or two-stage default models. T is the
                  test plan object, R is the new response object.

                  R = CreateResponse(T, Varname, Model) creates a one-stage model
                  of Varname, where T must be a one-stage test plan object.

                  R = CreateResponse(T, Varname, LocalModel, GlobalModel) or
                  R = CreateResponse(T, Varname, LocalModel, GlobalModel,
                  DatumType) creates a two-stage model of Varname. T must be a
                  two-stage test plan object. DatumType can only be specified if the local
                  model type permits a datum model. Only the model types "Polynomial
                  Spline" and "Polynomial with Datum" permit datum models.

                  Varname is the variable name for the new response.

                  Model is the One-stage model object (if you leave this field empty, the
                  default is used).

                  LocalModel is the Local Model object (if you leave this field empty,
                  the default is used).

                  GlobalModel is the Response Feature model object (if you leave this
                  field empty, the default is used).

                  DatumType can be 'None' 'Maximum' 'Minimum' or 'Linked'.

Examples   To create a response using the default models, enter:

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

CreateResponse

To create a response and specify the local and global model types, enter:

```
mdls = T.DefaultModels
LocalModel = CreateModel(mdl{1}, 'Local Polynomial Spline');
GlobalModel = CreateModel(mdl{2}, 'RBF');
R = CreateResponse(T, 'TQ', LocalModel, GlobalModel, 'Maximum')
```

See Also   Responses

Purpose   Create new response feature for local model

Syntax   RF = CreateResponseFeature(RF,RFType)
                     RF = CreateResponseFeature(RF,RFType,EvaluationPoint)

Description   This is a method of mbcmodel.localresponse.

                    RF = CreateResponseFeature(RF,RFType)

                    RF = CreateResponseFeature(RF,RFType,EvaluationPoint)

                    RFType is a description string belonging to the set of alternative
                    response features for the current local model.

                    EvaluationPoint is a row vector with an element for each model input
                    and is used for response features that require an input value to evaluate
                    the response feature (e.g., function evaluation, derivatives). It is an
                    error to specify an evaluation point for a response feature type that does
                    not require an evaluation point.

                    You should use this method to add response features without refitting
                    all local and global models.

Examples   RF = CreateResponseFeature(RF,'Beta_1')

See Also   ResponseFeatures(Local Model)

CreateTestplan

Purpose   Create new test plan

Syntax
```
T = CreateTestplan(P, TestPlanTemplate)
T = CreateTestplan(P, TestPlanTemplate, newtestplanname)
T = CreateTestplan(P, InputsPerLevel)
T = CreateTestplan(P, InputsPerLevel, newtestplanname)
T = CreateTestplan(P, Inputs)
T = CreateTestplan(P, Inputs, newtestplanname)
```

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

You can use this method with a test plan template or input information.

You set templates up in the Model Browser GUI. This setup 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. It is not possible to change the number of stages after creation of the test plan.

After you create a new test plan, 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.

Use CreateTestplan in the following ways:

`T = CreateTestplan(P, TestPlanTemplate)`

`T = CreateTestplan(P, TestPlanTemplate, newtestplanname)`

P is the project object.

TestPlanTemplate is the full name and path to the test plan template file created in the Model Browser.

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

`T = CreateTestplan(P, InputsPerLevel)`

`T = CreateTestplan(P, InputsPerLevel, newtestplanname)`

InputsPerLevel is a row vector with number of inputs for each stage.

`T = CreateTestplan(P, Inputs)`

```
T = CreateTestplan(P, Inputs, newtestplanname)
```

Inputs is a cell array with input information for each level. The input information can be specified as a cell array of mbcmodel.modelinput objects (one for each level), or as a cell array of cell arrays (one for each level).

Examples   To create a test plan using a test plan template, enter:

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

testplan = CreateTestplan(P, 'example_testplan')
```

To create a test plan using inputs per level, enter:

```
T = P.CreateTestplan([1,2])
```

To specify the input information in a cell array of mbcmodel.modelinput objects, enter:

```
% Define Inputs for test plan
LocalInputs = mbcmodel.modelinput('Symbol','S',...
    'Name','SPARK',...
    'Range',[O 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} );
```

Or

```
T = P.CreateTestplan({LocalInputs,GlobalInputs})
```

To specify the input information in a cell array, enter:

```
localInputs = {'S',0,50,'','SPARK'};
globalInputs = {'N',   800, 5000, '', 'ENGSPEED'
```

```
                      'L',   0.1,    1, '', 'LOAD'
                      'EXH', -5,    50, '', 'EXHCAM'
                      'INT', -5,    50, '', 'INTCAM'};

         T = CreateTestplan(P,{localInputs,globalInputs});
```

See Also   AttachData, CreateResponse, Responses, Data, Levels,
                InputSignalNames, InputsPerLevel, Inputs, modelinput

Purpose   Array of data objects in project or test plan

Syntax   allD = project.Data
                  allD = testplan.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 = P.Data;

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

                      allD = 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

DataFileTypes

Purpose	Data file types
Syntax	`f = mbcmodel.DataFileTypes`
Description	This is a function to return a list of data file types for `mbcmodel`.

Examples

```
f = mbcmodel.DataFileTypes

f =

    Columns 1 through 4
      'Excel file'    'FT/DB data files'    'Delimited Text File'
    [1x25 char]
    Column 5
      'MATLAB Data File'
```

See Also `ImportFromFile`, `CreateData`

Purpose Default models for test plan

Syntax `testplan.DefaultModels`

Description This is a read-only property of `mbcmodel.testplan`. It returns a cell array of `mbcmodel.model` objects (one array for each stage).

Examples To get the default model objects for use in creating a response, enter:

```
mdls = T.DefaultModels
LocalModel = CreateModel(mdl{1}, 'Local Polynomial Spline');
GlobalModel = CreateModel(mdl{2}, 'RBF');
R = CreateResponse(T, 'TQ', LocalModel, GlobalModel, 'Maximum')
```

See Also `CreateResponse` `modelinput`

DefineNumberOfRecordsPerTest

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);

DefineTestGroups

See Also   `DefineNumberOfRecordsPerTest, NumberOfTests`

Purpose   Designs in test plan

Syntax   `D = T.Designs`

Description   `Designs` is a property of `mbcmodel.testplan`.

`D = T.Designs` returns a cell array of designs in the test plan, `T`, one element for each level.

When using designs at the command line, designs are treated as an array. In the Design Editor you can build a design tree, where child designs inherit characteristics such as constraints from the parent design. At the command line you can copy and modify designs. By default, designs are added to the top level of the design tree. To build tree structures at the command line, you can use the `Parent` argument of the `AddDesign` method to specify the parent design in the design tree. The tree structure cannot be used at the command line any further, but you can use the design tree in the Design Editor after you load the project into the Model Browser.

Examples   To get local designs only:

```
LocalDesigns = T.Designs{1}
```

To get global designs only:

```
GlobalDesigns = T.Designs{2}
```

To get the fifth global design:

```
D = T.Design {2}(5)
```

After modifying the design, you must call `UpdateDesign`, or reassign to the test plan as follows:

T.Design {2}(5) = D

See Also   `UpdateDesign`

DetachData

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

Discrepancy

Purpose   Discrepancy value

Syntax   `s = Discrepancy(D)`

Description   `Discrepancy` is a method of `mbcdoe.design`.

`s = Discrepancy(D)` returns the discrepancy, which is a measure of the deviation from the average point density. `Discrepancy` is defined over the unconstrained design and is only available for space-filling designs.

See Also   `Maximin`; `Minimax`

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

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

Purpose   Evaluate model or design constraint

Syntax   `Y = Evaluate(M, X)`
                   `Y = Evaluate(C, X)`

Description   This is a method of `mbcmodel.model` and `mbcdoe.designconstraint`.

                   `Y = Evaluate(M, X)` evaluates the model M at X.

                   `Y = Evaluate(C, X)` evaluates the design constraint C at X (negative results are within the constraint).

                   X is a (*numpoints*-by-*nfactors*) array.

                   Y is a (*numpoints*-by-1) array.

See Also   `PredictedValue`, `PEV`

Export

Purpose	Make command-line or Simulink® export model
Syntax	ExportedModel = Export(MODEL) ExportedModel = Export(MODEL, Format)
Description	This is a method of these model objects: mbcmodel.hierarchicalresponse, mbcmodel.localresponse, mbcmodel.response and mbcmodel.model.

Description (continued)

ExportedModel = Export(MODEL) exports the model to MATLAB® software.

ExportedModel = Export(MODEL, Format) exports the model in the specified format, which can be 'MATLAB' or 'Simulink'.

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 you export them from the Model Browser. You can save these models as a *.mat file and load them into CAGE.

Model 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 Data) in the Model Browser User's Guide ("Data Loading Application Programming Interface"). See also `mbccheckindataloadingfcn` to specify your own data loading function.

Examples   `X = ExportToMBCDataStructure(D1);`

See Also   `ImportFromMBCDataStructure`

Filename

Purpose   Full path to project file

Syntax   Name = P.Filename

Description This is a property of mbcmodel.project.

Examples   Name = P.Filename;

Purpose   Structure array holding user-defined filters

Syntax   `filt = 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   `filt = D.Filters;`

See Also   `AddFilter`, `ModifyFilter`, `RemoveFilter`

FindDesign

Purpose	Find design by name
Syntax	`D = FindDesign(T,Name)` `D = FindDesign(T,Level,Name)`
Description	FindDesign is a method of `mbcmodel.testplan`.

<div></div>

`D = FindDesign(T,Name)` finds a design with a matching name from the test plan `T`.

`Name` is a string or a cell array of strings specifying a design name.

`Level` is the test plan level. By default the level is the outer level (i.e., Level 1 for one-stage, Level 2 (global) for two-stage).

`D = FindDesign(T,Level,Name)` finds a design with a matching name from the specified level of the test plan.

FitAlgorithm

Purpose	Fit algorithm for model
Syntax	F = M.FitAlgorithm

Description

This is a property of mbcmodel.model.

An mbcmodel.model.FitAlgorithm object is contained within the FitAlgorithm property of an mbcmodel.model object. This object has a Name property, and the following methods: CreateAlgorithm, getAlternativeNames, IsAlternative, SetupDialog, properties.

Examples

To get a fitalgorithm object, F, from a model:

```
M = mbcmodel.CreateModel('Polynomial', 4);
F = M.FitAlgorithm

F =
Algorithm: Least Squares
Alternatives: 'Minimize PRESS','Forward Selection','Backward
Selection','Prune'
1x1 struct array with no fields.
```

See Also

CreateAlgorithm, getAlternativeNames, IsAlternative, SetupDialog.

Fit

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

Syntax   ```
[statistics, model] = Fit(model, X, Y)
[statistics, model] = Fit(model)
```

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

[statistics, model] = Fit(model, X, Y) This fits the model to the specified data. After you have called Fit specifying the data to use, then you can refit the model by calling [statistics, model] = Fit(model)

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 network 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

**Purpose**          Fix design points

**Syntax**           D = FixPoints(D)
                     D = FixPoints(D,indices)

**Description**      FixPoints is a method of mbcdoe.design.

                     D = FixPoints(D) fixes all points in the design.

                     D = FixPoints(D,indices) fixes all points specified by indices.

**See Also**         PointTypes; RemovePoints

# Generate

**Purpose**     Generate new design points

**Syntax**      ```
D = Generate(D)
D = Generate(D,NumPoints)
D = Generate(D,'Prop1',value1,...)
```

Description Generate is a method of mbcdoe.design. The Generate method always generates a new design and replaces the existing points (fixed or free).

D = Generate(D) regenerates the design with the current generator settings (the current design properties and current number of points). It is possible that a different design will result (e.g., for Latin Hypercube Sampling designs).

D = Generate(D,NumPoints) generates the number of points specified by NumPoints using the current generator settings. You cannot specify the number of points for all design types (e.g., Central Composite, Box Behnken) and therefore the NumPoints second input is not supported for all design types.

D = Generate(D,'Prop1',value1,...) generates a new design with the generator specified by the generator property value pairs.

You can use the property value pairs to specify design generator properties (such as the design Type) as part of the Generate command, e.g.,

```
C = OptDesign.CreateCandidateSet(OptDesign,...
        'Type', 'Grid',...
        'NumberOfLevels',[21 21 21]);

OptDesign = Generate(OptDesign,...
    'Type','V-optimal',...
    'CandidateSet',C,...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',200);
```

This is equivalent to the following code setting the properties individually and then assigning the updated generator object to the design:

```
P = OptDesign.Generator;
P.Type = `V-optimal';
P.CandidateSet.NumberOfLevels(:)=21;
P.MaxIterations = 200;
P.NumberOfPoints = 200;
P.NoImprovement = 50;
OptDesign.Generator = P;
```

You see an error if you try to call Generate when the design Style is User-defined or Experimental data.

For space-filling designs, see also ConstrainedGenerate.

Examples   To generate a design with 10 points:

```
d = Generate( d, 10 );
```

Note The design Type must have a writeable property 'NumberOfPoints' to use this syntax D = Generate(D,NumPoints). See Type (for designs and generators).

To create and generate a 15 point latin hypercube sampling design:

```
globalDesign = TP.CreateDesign(2, 'Type',...
 'Latin Hypercube Sampling');
globalDesign = Generate(globalDesign, 15)
```

To regenerate the design and get a different 15 point latin hypercube sampling design:

```
globalDesign = Generate(globalDesign);
```

Generate

To create and generate a halton design with 50 points:

```
haltonDesign = CreateDesign( inputs, 'Type',...
 'Halton Sequence', 'Name', 'Halton' );
haltonDesign = Generate( haltonDesign, 'NumberOfPoints', 50 );
```

To create and generate a halton design with specified scrambling and
other properties:

```
haltonDesignWithScrambling = haltonDesign.CreateDesign...
( 'Name', 'Scrambled Halton' );
haltonDesignWithScrambling = Generate...
(haltonDesignWithScrambling,
 'Scramble', 'RR2', 'PrimeLeap', true );
```

To create a full factorial design and specify the number of levels when
generating the design:

```
design = CreateDesign( inputs, 'Type', 'Full Factorial' );
design = Generate( design, 'NumberOfLevels', [50 50] );
```

See Also   Augment; CreateDesign; ConstrainedGenerate

Purpose   Design generation options

Syntax   D.Generator
                 D.Generator = NewGenerator

Description Generator is a property of mbcdoe.design.

D.Generator returns an mbcdoe.generator object.

D.Generator = NewGenerator generates a new design based on the new design generator. Design generators provide the properties for all the design types.

The properties you can set depend on the design Type. To view the properties for generating designs, see Properties (for design generators).

Use getAlternativeTypes to get a list of alternative generators.

See Also   Generate; Properties (for design generators); Type (for designs and generators); getAlternativeTypes.

GetAllTerms

Purpose   List all model terms

Syntax   `Terms = M.Properties.GetAllTerms`

Description   This is a method of `mbcmodel.linearmodelproperties`.

`Terms = M.Properties.GetAllTerms` returns a list of all terms in this model. M is an `mbcmodel.linearmodel` object.

`Terms` is a (*numterms*-by-*nfactors*) array. The $(m,n)^{th}$ element is the power of the nth factor in the mth term.

Examples   The following example creates a model, and finds which terms are quadratic in the first input factor (X1):

```
mdl = mbcmodel.CreateModel('Polynomial', 2)

mdl =

    1 + 2*X1 + 8*X2 + 3*X1^2 + 6*X1*X2 + 9*X2^2 + 4*X1^3
      + 5*X1^2*X2 + 7*X1*X2^2 + 10*X2^3
    InputData: [0x2 double]
    OutputData: [0x1 double]
    Status: Not fitted
    Linked to Response: <not linked>

>>terms = mdl.Properties.GetAllTerms;
>>x1quadraticterms = find(terms(:,1)==2)

x1quadraticterms =

     4
     8
```

See Also   `GetIncludedTerms`

Purpose   List alternative algorithm names

Syntax   F.getAlternativeNames
                AltList = getAlternativeNames(F)

Description This is a method of mbcmodel.fitalgorithm.

                F.getAlternativeNames or AltList = getAlternativeNames(F)
                return a cell array of alternative algorithm names. F is a
                mbcmodel.fitalgorithm object.

Examples   mdl = mbcmodel.CreateModel('Polynomial', 2);
                   F = mdl.FitAlgorithm;
                   altAlgs = F.getAlternativeNames

                   altAlgs =

                       'Least Squares'   'Minimize PRESS'   'Forward Selection'
                   'Backward Selection'   'Prune'

See Also   CreateAlgorithm, IsAlternative

getAlternativeTypes

Purpose	Alternative model or design types
Syntax	`list = getAlternativeTypes(Model)`
	`list = getAlternativeTypes(Design)`
	`list = getAlternativeTypes(Design,Style)`
	`list = getAlternativeTypes(DesignGenerator)`
	`list = getAlternativeTypes(DesignGenerator,Style)`
	`list = getAlternativeTypes(CandidateSet)`
	`list = getAlternativeTypes(DesignConstraint)`

Description This is a method of `mbcmodel.model`, and all the design objects: `mbcdoe.design`, `mbcdoe.generator`, `mbcdoe.candidateset`, and `mbcdoe.designconstraint`.

Models

`list = getAlternativeTypes(Model)` returns a cell array of alternative model types with the same number of inputs as `Model`.

Designs

`list = getAlternativeTypes(Design)` returns a list of design types which could be used as alternative designs for current design.

`list = getAlternativeTypes(Design,Style)` returns a list of design types of the specified style. The design style must be one of `'Space-Filling'`, `'Classical'` or `'Optimal'`.

Design Generators

`list = getAlternativeTypes(DesignGenerator)` returns a list of design generator types which could be used as alternative designs for current design generator.

`list = getAlternativeTypes(DesignGenerator,Style)` returns a list of design generator types of the specified style. The design generator style must be one of `'Candidate Set'`,`'Space-Filling'`, `'Classical'` or `'Optimal'`.

Design Candidate Sets

`list = getAlternativeTypes(CandidateSet)` is a list of candidate set types which could be used as alternative candidate sets for the current candidate set. The candidate set can be obtained from an optimal design generator or using `mbcdoe.design.CreateCandidateSet`.

Design Constraints

`list = getAlternativeTypes(DesignConstraint)` returns a list of design constraint types.

Examples

```
mdl = mbcmodel.CreateModel('RBF', 2);
altmodels = getAlternativeTypes(mdl)
```

This produces the output:

```
altmodels =

  Columns 1 through 6

    'Polynomial'    'Hybrid Spline'    'RBF'    'Polynomial-RBF'
   'Multiple Linear'

  Columns 7 through 8

    'Neural Network'    'Transient'
```

See Also   `Type (for models)`, `CreateModel`

GetIncludedTerms

Purpose   List included model terms

Syntax   `Terms = M.Properties.GetIncludedTerms`

Description   This is a method of `mbcmodel.linearmodelproperties`.

`Terms = M.Properties.GetIncludedTerms` returns a list of those terms that will be used to fit the model. `M` is an `mbcmodel.linearmodel` object.

`Terms` is a (*numincludedterms*-by-*nfactors*) array. The $(m,n)^{\text{th}}$ element is the power of the nth factor in the mth included term.

Examples
```
>>mdl = mbcmodel.CreateModel('Polynomial', 2);


>>includedterms = mdl.Properties.GetIncludedTerms;
>>x1quadraticterms = find(includedterms(:,1)==2)

x1quadraticterms =

    4
    8
```

See Also   `GetAllTerms`, `SetTermStatus`

Purpose   List labels for model terms

Syntax   `Labels = M.Properties.GetTermLabel`
`Labels = M.Properties.GetTermLabel( Terms )`
`Labels = M.Properties.GetTermLabel( Terms, 'Format',`
 `OutputFormat )`

Description This is a method of `mbcmodel.linearmodelproperties`, which returns a user-friendly label for one or more specified terms.

`Labels = M.Properties.GetTermLabel`

`Labels = M.Properties.GetTermLabel( Terms )`

`Labels = M.Properties.GetTermLabel( Terms, 'Format',`
`OutputFormat )`

`M` is an `mbcmodel.linearmodel` object.

The specified terms form a row where each value gives the power of that parameter. `OutputFormat`

can be `'List'` or `'Formula'`.

Examples
```
mdl = mbcmodel.CreateModel('Polynomial', 2);
mdl.Properties.GetTermLabel([1 2; 1 O] )
```

produces `{'X1*X2^2';'X1'}` and

```
mdl.Properties.GetTermLabel([1 2; 1 O], 'Format', 'Formula' )
```

produces `'X1*X2^2 + X1'`.

See Also   `GetAllTerms, GetIncludedTerms`

GetTermStatus

Purpose   List status of some or all model terms

Syntax   ```
Status = M.Properties.GetTermStatus
Status = M.Properties.GetTermStatus(Terms)
```

**Description**  This is a method of `mbcmodel.linearmodelproperties`.

`Status = M.Properties.GetTermStatus` returns the status of all of
the terms in this model. `Status` is a cell array of status strings. `M` is
an `mbcmodel.linearmodel` object.

`Status = M.Properties.GetTermStatus(Terms)` returns the status of
the specified terms in this 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.

**Examples**
```
mdl = mbcmodel.CreateModel('Polynomial', 2);
```

Get status of $X2^3$ term:

```
status = mdl.Properties.GetTermStatus([O 3])

status =

    'Step'
```

Get status of all terms linear in X1:

```
status = mdl.Properties.GetTermStatus([1 0; 1 1; 1 2])

status =

    'Step'
    'Step'
    'Step'
```

See Also   SetTermStatus, StepwiseStatus

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 `DataFileTypes` for the specification of the allowed file types (and `mbccheckindataloadingfcn` to specify your own data loading function).

Filetype 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, DataFileTypes, 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 Data) in the *Model Browser User's Guide* ("Data Loading Application Programming Interface"), and see also `mbccheckindataloadingfcn` to specify your own data loading function.

Examples   `ImportFromMBCDataStructure(D, mbcStruct);`

See Also   `ImportFromFile`, `CreateData`, `BeginEdit`, `RemoveData`, `Append`, `ExportToMBCDataStructure`

InputData

Purpose　　　Input data for model

Syntax　　　`D = M.InputData`

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

Examples　　　`D = knot.InputData;`

See Also　　`OutputData`

Purpose   Inputs for test plan, model, design or constraint

Syntax   ```
testplan.Inputs
model.Inputs
design.Inputs
```

Description This is a property of `mbcmodel.testplan`, `mbcmodel.model`, `mbcdoe.design` and `mbcdoe.designconstraint`.

For `mbcmodel.testplan`, this property returns a cell array of `mbcmodel.modelinput` objects (one array for each stage). You cannot change the number of stages after creation of the test plan.

For `mbcmodel.model`, this property returns an `mbcmodel.modelinput` object. It is not editable when attached to a response. You cannot change number of inputs after creation.

In both cases, verification of valid variable names and symbols occurs before assigning inputs to model at the command line. Names and Symbols must be unique.

For `mbcdoe.design`, `D.Inputs = NewInputs` updates the inputs.

The number of design inputs cannot be changed. Many designs have `Limits` properties in addition to model input ranges. These properties allow you to restrict the range of the design without changing the model or losing points via a constraint.

See Also   `CreateTestplan`, `modelinput`

InputSetupDialog

Purpose   Open Input Setup dialog box to edit inputs

Syntax   `[NEWMODEL, OK] = InputSetupDialog(OLDMODEL)`
                  `[NEWTESTPLAN, OK] = InputSetupDialog(OLDTESTPLAN)`

Description   This is a method of `mbcmodel.model` and `mbcmodel.testplan`.

`[NEWMODEL, OK] = InputSetupDialog(OLDMODEL)` opens the Input Setup dialog box, where you can edit the model inputs (names, symbols, and ranges).

`[NEWTESTPLAN, OK] = InputSetupDialog(OLDTESTPLAN)` opens the Input Setup dialog box, where you can edit the test plan inputs (names, symbols, and ranges).

If you click **Cancel** to dismiss the dialog box, `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. The new model is refitted when you click OK.

Purpose Names of signals in data that are being modeled

Syntax inputs = 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 = T.'InputSignalNames;

InputFactors = thisRF.InputSignalNames';

See Also SignalNames

InputsPerLevel

Purpose Number of inputs at each level in model

Syntax `L = 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 = T.InputsPerLevel
L =
  2  4
```

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

See Also `Levels`, `Level`

Purpose   Test alternative fit algorithm

Syntax   OK = IsAlternative(F1, F2)

Description   This is a method of mbcmodel.fitalgorithm.

OK = IsAlternative(F1, F2) tests whether F is an alternative
mbcmodel.fitalgorithm for F1.

See Also   CreateAlgorithm, getAlternativeNames

IsBeingEdited

Purpose	Boolean signaling if data or model is being edited
Syntax	OK = D.IsBeingEdited
Description	This is a property of mbcmodel.data and mbcmodel.model.
	This Boolean property indicates that the data or model is currently being edited.
	For data, 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 = D.IsBeingEdited; OK = knot.IsBeingEdited;
See Also	BeginEdit, IsEditable, CommitEdit, RollbackEdit

Purpose   Boolean signaling whether data is editable

Syntax   OK = 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  = p.Data;
D1 = p.Data;
BeginEdit(D1);
tp = p.Testplan;
Attach(tp, D);
```

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

At this point `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 = 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 data 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),

Purpose   Level in test plan of response

Syntax   `level = 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 = R.Level;`

See Also   `Levels`

Levels

Purpose   Number of levels in hierarchical model

Syntax   levels = T.'Levels

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

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

Examples   levels = 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

LoadProject

Purpose	Load mbcmodel.project
Syntax	P = MBCMODEL.LOADPROJECT(FILENAME)
Description	P = mbcmodel.LoadProject(FILENAME) loads a mbcmodel.project from the file FILENAME.
See Also	CreateProject, Load

Purpose   Edit local model properties

Syntax   Props = localmodel.Properties

Description   This is a property of the mbcmodel.localmodel object, which is a
                   subclass of mbcmodel.model.

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

                   Every local model object has an mbcmodel.modelproperties object
                   (within the Properties property). In this object, each local model type
                   has specific properties, as described in the following tables.

Local Polynomial Properties

Property	Description
Order	Polynomial order (vector int: {[0,Inf],2})
InteractionOrder	Maximum order of interaction terms (int: [0,Inf])
TransformInputRange	Transform inputs (Boolean)
ParameterNames	List of parameter names (read-only)
StepwiseStatus	Stepwise status {'Always','Never','Step'} (cell)
Transform	Transform function (char) or empty ('')

LocalModel Properties

Local Polynomial Properties (Continued)

Property	Description
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)
CorrelationModel	Correlation Model (enum: `{'None','MA(1)','AR(1)', 'AR(2)'}`)

Local Hybrid Spline Properties

Property	Description
Order	Spline and polynomial order (vector int: `{[0,3],2}`)
SplineVariable	Spline variable
SplineInteraction	Order of interaction between spline and polynomial (int: `[0,3]`)
Knots: Position of knots (vector real)	ParameterNames: List of parameter names (read-only)
StepwiseStatus	Stepwise status `{'Always','Never','Step'}` (cell)
Transform	Transform function (char) or empty (`''`)
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)
CorrelationModel	Correlation Model (enum: `{'None','MA(1)','AR(1)', 'AR(2)'}`)

Local Polynomial Spline Properties

Property	Description
HighOrder	Polynomial order above knot (int: `[2,Inf]`)
LowOrder	Polynomial order below knot (int: `[2,Inf]`)
Transform	Transform function (char) or empty (`''`)
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)
CorrelationModel	Correlation Model (enum: `{'None','MA(1)','AR(1)', 'AR(2)'}`)
DatumType	Datum Type (enum: `{'None','Maximum','Minimum', 'Linked'}`)

Local Polynomial With Datum Properties

Property	Description
Order	Polynomial order (int: `[0,Inf]`)
Transform	Transform function (char) or empty (`''`)
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)

LocalModel Properties

Local Polynomial With Datum Properties (Continued)

Property	Description
CorrelationModel	Correlation Model (enum: {'None','MA(1)','AR(1)', 'AR(2)'})
DatumType	Datum Type (enum: {'None','Maximum','Minimum', 'Linked'})

Local Free Knot Spline Properties

Property	Description
Order	Spline Order (int: [0,Inf])
NumKnots	Number of knots (int: 'Positive')
Transform	Transform function (char) or empty ('')
CovarianceModel	Covariance Model (enum: {'None','Power', 'Exponential','Mixed'})
CorrelationModel	Correlation Model (enum: {'None','MA(1)','AR(1)', 'AR(2)'})

Local Truncated Power Series Properties

Property	Description
Order	Polynomial order (int: 'Positive')

Local Truncated Power Series Properties (Continued)

Property	Description
NumKnots	Number of knots (int: `'Positive'`)
Transform	Transform function (char) or empty (`''`)
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)
CorrelationModel	Correlation Model (enum: `{'None','MA(1)','AR(1)', 'AR(2)'}`)

Local Growth Properties

Property	Description
Model	Growth model (enum: `{'expgrowth','gomp', 'logistic','logistic4', 'mmf','richards', 'weibul'}`)
AlternativeModels	List of growth models (read-only)
Transform	Transform function (char) or empty (`''`)
TransformBothSides	Transform both sides (Boolean)
CovarianceModel	Covariance Model (enum: `{'None','Power', 'Exponential','Mixed'}`)
CorrelationModel	Correlation Model (enum: `{'None','MA(1)','AR(1)', 'AR(2)'}`)

LocalModel Properties

Local User-Defined Properties

Property	Description
Model	Name of user-defined model (enum: {`'exponential'`})
AlternativeModels	List of registered user-defined models (read-only)
Transform	Transform function (char) or empty (`' '`)
TransformBothSides	Transform both sides (Boolean)
CovarianceModel	Covariance Model (enum: {`'None'`,`'Power'`, `'Exponential'`,`'Mixed'`})
CorrelationModel	Correlation Model (enum: {`'None'`,`'MA(1)'`,`'AR(1)'`, `'AR(2)'`})

Local Transient Properties

Property	Description
Model	Name of transient model (enum: {`'fuelPuddle'`})
AlternativeModels	List of registered transient models (read-only)
Transform	Transform function (char) or empty (`' '`)
TransformBothSides	Transform both sides (Boolean)

Local Transient Properties (Continued)

Property	Description
CovarianceModel	Covariance Model (enum: {'None','Power', 'Exponential','Mixed'})
CorrelationModel	Correlation Model (enum: {'None','MA(1)','AR(1)', 'AR(2)'})

Local Multiple Models Properties

Property	Description
ModelCandidates	List of candidate models (cell)
SelectionStatistic	Selection statistic for automatic model selection (char)
AutomaticInputRanges	Use data range as model input ranges (Boolean)
Transform	Transform function (char) or empty ('')

Local Average Fit Properties

Property	Description
Model	[1x1 mbcmodel.linearmodel]
Transform	Transform function (char) or empty ('')

Examples   To create a local model object, create a model specifying any model Type that begins with the word "local", e.g.,

```
L = mbcmodel.CreateModel('Local Polynomial',2);
```

LocalModel Properties

To show properties, at the command line enter:

```
P = L.Properties

P =
Local Polynomial Properties
                    Order: [3 3]
          InteractionOrder: 3
      TransformInputRange: 1
            ParameterNames: {10x1 cell}
            StepwiseStatus: {10x1 cell}
                 Transform: ''
           CovarianceModel: 'None'
          CorrelationModel: 'None'
```

To set the Order property to a quadratic, enter:

```
>> P.Order = [2,2]

P =
Local Polynomial Properties
                    Order: [2 2]
          InteractionOrder: 2
      TransformInputRange: 1
            ParameterNames: {6x1 cell}
            StepwiseStatus: {6x1 cell}
                 Transform: ''
           CovarianceModel: 'None'
          CorrelationModel: 'None'
```

To update the local model, the properties object must be reassigned to the model as follows:

```
>> L.Properties = P

L =
```

```
                1 + 2*X1 + 5*X2 + 3*X1^2 + 4*X1*X2 + 6*X2^2
                InputData: [0x2 double]
                OutputData: [0x1 double]
                Status: Being Edited
                Linked to Response: not linked
```

See Also   CreateModel, Type (for models), ResponseFeatures(Local Model)

LocalResponses

Purpose   Array of local responses for response

Syntax   `local = response.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 = response.LocalResponses;`

Purpose   Build two-stage model from response feature models

Syntax   OK = MakeHierarchicalResponse(L,MLE)

Description   This method of mbcmodel.localresponse builds a two-stage model from the response feature models and optionally runs MLE (Maximum Likelihood Estimation). If there are more response features than the number of parameters in the local model, the subset of response features that leads to the best hierarchical response is chosen. The best hierarchical response is chosen using PRESS RMSE (root mean square prediction error — see "PRESS statistic") if all the response feature models are linear. Otherwise, the best hierarchical response is chosen using Two-stage RMSE.

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 is called automatically.

If the local and response models are not ready to calculate a two-stage model, an error is generated. This situation can occur 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

MatchInputs

Purpose   Match design constraint inputs

Syntax   C = MatchInputs(C,DesignInputs)
                   C = MatchInputs(C,DesignInputs,mapping)

Description   MatchInputs is a method of mbcdoe.designconstraint. Use it to
                   match inputs for constraints from different sources.

                   C = MatchInputs(C,DesignInputs)

                   C = MatchInputs(C,DesignInputs,mapping) matches inputs where
                   mapping defines the relationship between the inputs in C, and
                   DesignInputs.

Examples   A design constraint does not have required inputs EXH_RET and
                   INT_ADV. Use MatchInputs to match the constraint inputs to the
                   design inputs as follows:

```
c = p.Testplans.BoundaryModel('all')
c =
Star(N-3.5e+003,L-0.54)

originalInputs=c.Inputs
originalInputs =
   SPEED (N) [rpm] [500,6000]
   LOAD (L) [%] [0.06,0.95]

designInputs = Design.Inputs
designInputs =
   SPEED (N) [rpm] [500,6000]
   LOAD (L) [%] [0.06,0.95]
   EXH_RET (ECP) [DegCrank] [-5,50]
   INT_ADV (ICP) [DegCrank] [-5,50]

c2=MatchInputs(c,designInputs,[1 2]);
newInputs=c2.Inputs
newInputs =
   SPEED (N) [rpm] [500,6000]
```

```
LOAD (L) [%] [0.06,0.95]
EXH_RET (ECP) [DegCrank] [-5,50]
INT_ADV (ICP) [DegCrank] [-5,50]
```

See Also   CreateConstraint

Maximin

Purpose   Maximum of minimum of distance between design points

Syntax   s = Maximin(D)

Description   Maximin is a method of mbcdoe.design.

s = Maximin(D) returns the maximum of the minimum distance
between design points. Maximin is defined over the unconstrained
design and is only available for space-filling design types.

See Also   Minimax

Purpose	Merge designs
Syntax	D = Merge(D1,D2,...)
Description	Merge is a method of mbcdoe.design.
	D = Merge(D1,D2,...) merges the specified designs D1, D2, etc. into a single design D. The resulting design is a custom design Style.
See Also	Style; Augment

Minimax

Purpose Minimum of maximum distance between design points

Syntax s = Minimax(D)

Description Minimax is a method of mbcdoe.design.

s = Minimax(D) returns the minimum of the maximum distance between design points. Minimax is defined over the unconstrained design and is only available for space-filling designs.

See Also Maximin

Purpose	Model for design
Syntax	`D.Model = NewModel`
Description	`Model` is a property of `mbcdoe.design`.
	`D.Model = NewModel` changes the model for the design to `NewModel`.
	The number of inputs cannot be changed. Many designs have `Limits` properties in addition to model input ranges.
	Setting this property changes optimal designs to `custom` if the new model does not support optimal designs.
See Also	`Inputs`

Model Object

Purpose   Model object within response object

Syntax   M = response.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, properties, and fit algorithm settings (ModelSetup, Type (for models); Properties (for models), CreateAlgorithm).

- Create a copy of the model with the same inputs (CreateModel).

- 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   M = response.Model;

Purpose   Model for specified test

Syntax   `mdl = ModelForTest(L,TestNo);`

Description   This is a method of `mbcmodel.localresponse`.

                   `mdl = ModelForTest(L,TestNo);`

Examples   To get the model for test 22, enter:

                       `mdl = ModelForTest(L,22);`

modelinput

Purpose   Create modelinput object

Syntax
```
Inputs = mbcmodel.modelinput('PropertyName1',value1,
    'PropertyName2',value2,...);
Inputs = mbcmodel.modelinput(NUMINPUTS);
Inputs = mbcmodel.modelinput(INPUTCELLARRAY);
```

Description   This is the constructor for the mbcmodel.modelinput object.

```
Inputs =
mbcmodel.modelinput('PropertyName1',value1,'PropertyName2',value2,...);
```

You can set the properties shown in the following table.

Property	Description
Range	[min,max]
NonlinearTransform	{'','1./x','sqrt(x)', 'log10(x)','x.^2', 'log(x)'}
Name	String. Signal name from data set. Inputs for a test plan must be set before selecting data.
Symbol	String. Short name for plot labels and for use in CAGE.
Units	String. Units are overwritten from the data set units when a data is selected.

Specify "property, value" pairs as follows:

```
Inputs = mbcmodel.modelinput('Symbol',{'A','B'},...
  'Range',{[0 100],[-20 20]});
```

Scalar expansion of properties is supported, e.g.,

```
Inputs = mbcmodel.modelinput('Symbol',{'A','B'},...
```

```
     'Range',[0 100]);
```

```
Inputs = mbcmodel.modelinput(NUMINPUTS);
```

NUMINPUTS is the number of inputs. Symbols are automatically set to `'X1'`, `'X2'`,...,`'Xn'`. The default range is [-1,1]. For example:

```
  Inputs = mbcmodel.modelinput(2);
```

```
Inputs = mbcmodel.modelinput(INPUTCELLARRAY);
```

INPUTCELLARRAY is a cell array with one row per input and 5 columns to specify factor names, symbols, ranges and nonlinear transforms as follows.

The columns of INPUTCELLARRAY must be:

1 Factor symbol (string)

2 Minimum (double)

3 Maximum (double)

4 Transform (string) — empty for none

5 Signal name

These columns are the same as the columns in the Model Factor Setup dialog box, which can be launched from the test plan in the model browser.

Examples   To create a modelinput object with 2 inputs, enter:

```
  Inputs = mbcmodel.modelinput(2);
```

To create a modelinput object and define symbols and ranges, enter:

```
  Inputs = mbcmodel.modelinput('Symbol',{'A','B'},...
   'Range',{[0 100],[-20 20]});
```

modelinput

```
Inputs = mbcmodel.modelinput('Symbol',{'A','B'},...
 'Range',[0 100]);
```

To create a modelinput object and define inputs with a cell array, enter:

```
Inputs = mbcmodel.modelinput( {...
 'N',   800, 5000, '', 'ENGSPEED'
 'L',   0.1,    1, '', 'LOAD'
'EXH',  -5,    50, '', 'EXHCAM'
'INT',  -5,    50, '', 'INTCAM'} );
```

See Also   CreateModel, CreateTestplan

Purpose   Open 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 = P.Modified`
Description	This is a property of `mbcmodel.project`.
Examples	`Name = 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`

MultipleVIF

Purpose   Multiple VIF matrix for linear model parameters

Syntax   `VIF = MultipleVIF(LINEARMODEL)`

Description This is a method of `mbcmodel.linearmodel`.

`VIF = MultipleVIF(LINEARMODEL)` calculates the multiple Variance Inflation Factor (VIF) matrix for the linear model parameters.

Examples   `VIF = MultipleVIF(knot_model)`

See Also   `ParameterStatistics`

Purpose	Name of object
Syntax	`name = A.Name`
Description	This is a property of project, data, test plan, input, model, fitalgorithm, design, and design constraint objects.
	A can be any test plan (`T`), data (`D`), project (`P`) model (`L`, `R`, `HR`), fitalgorithm (`F`), design (`D`), or design constraint (`C`) object.
	You can change the names of these objects as follows:
	`A.Name = newName`
	For response (output or Y data) signal names, see `ResponseSignalName`.
	For `mbcmodel.model.Name`, the Name property refers to the model output name. This property is set to the data signal name when the response is created or if a model is assigned to a response. This property cannot be set when a response is attached to the model.
	For model parameter names, see `Names`.
	For testplan and response object input names, see `InputSignalNames`, and for data objects, see `SignalNames`.
Examples	`ResponseFeatureName = thisRF.Name;`
See Also	`Names`, `InputSignalNames`, `SignalNames`, `ResponseSignalName`

Names

Purpose	Model parameter names
Syntax	`N = 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 = paramsknot.Names
N =
'1'
'N'
'N^2'
'N*L'
'N*A'
'L'
'L^2'
'L*A'
'A'
'A^2';
```

See Also   `NumberOfParameters`, `Values`, `Name`

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 or design object inputs

Syntax   N = model.NumberOfInputs

Description   This is a property of `mbcmodel.model`, `mbcmodel.modelproperties`, and the design objects `mbcdoe.design`, `mbcdoe.generator`, `mbcdoe.candidateset`, and `mbcdoe.designconstraint`. It returns the number of inputs to the model or design object.

Examples   N = knot.'NumberOfInputs;

Purpose   Number of included model parameters

Syntax   N = 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 = knotparams.NumberOfParameters;

See Also   SizeOfParameterSet, StepwiseSelection, StepwiseRegression,
                     Names, Values

NumberOfPoints

Purpose   Number of design points

Syntax   `D.NumberOfPoints`

Description   `NumberOfPoints` is a read only property of `mbcdoe.design` (constrained number of points).

`D.NumberOfPoints` is the number of points in the design after applying the constraints.

You specify the number of points for a design using the generator object. The `NumberOfPoints` property of `mbcdoe.generator` is the number of points before any constraints are applied. You cannot specify the number of points for all design types (e.g., it is not allowed for Central Composite, Box Behnken). To see which design types have an editable `NumberOfPoints` property, see the tables in `Type (for designs and generators)`.

See Also   `Type (for designs and generators)`

Purpose   Total number of records in data object

Syntax   `numRecords = D.NumberOfRecords`

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

Examples   `numRecords = Data.NumberOfRecords;`

NumberOfTests

Purpose   Total number of tests being used in model

Syntax   numtests = 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 = TQ_response.NumberOfTests;

See Also   DefineTestGroups

Purpose   Optimal design criteria (V, D, A, G)

Syntax
```
s = OptimalCriteria(D)
s = OptimalCriteria(D,Criteria)
```

Description   `OptimalCriteria` is a method of `mbcdoe.design`. `OptimalCriteria` can only be used for optimal designs.

`s = OptimalCriteria(D)` returns an array with the values of optimal criteria [V,D,A,G].

`s = OptimalCriteria(D,Criteria)` returns the specified optimal criteria. `Criteria` must be one of V,D, A, or G.

OutlierIndices

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`

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

OutputData

Purpose	Output (or response) data for model
Syntax	D = M.OutputData
Description	This is a property of mbcmodel.model.
	It returns an array of the response data currently in the model.
Examples	D = knot.OutputData;
See Also	InputData

Purpose   Object from which data was received

Syntax   O = D1.Owner

Description This property of mbcmodel.data is:

- Empty if the data was created using mbcmodel.CreateData
- An mbcmodel.project object if the data was extracted from a project
- An mbcmodel.testplan object if the data was extracted from a test plan

Examples   O = D1.Owner;

Parameters

Purpose	Model parameters
Syntax	`P = 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 = model.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 =
```

ParameterStatistics

```
              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   Partial VIF matrix for linear model parameters

Syntax   STATS = PartialVIF(LINEARMODEL)

Description   This is a method of `mbcmodel.linearmodel`.

STATS = PartialVIF(LINEARMODEL) calculates the partial Variance
Inflation Factor (VIF) matrix for the linear model parameters.

Examples   VIF = PartialVIF(knot_model)

See Also   ParameterStatistics

PEV

Purpose	Predicted error variance of model at specified inputs
Syntax	`pev = PEV(R, X)`
Description	This is a method of the hierarchical, local response, 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. For a local response, the predicted value uses the hierarchical model.

Note that for an mbcmodel.model and mbcmodel.response objects only, 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

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

Points

Purpose   Matrix of design points

Syntax   designPoints = D.Points

Description   Points is a property of mbcdoe.design.

designPoints = D.Points returns the matrix of design points.

You can perform any valid MATLAB® operation on this matrix. The number of columns of the points matrix must be the same as the number of inputs when setting Points. If you make an assignment to the Points, the design type changes to Custom. Points are only updated in the underlying design if they have changed.

See Also   FixPoints; PointTypes; RemovePoints; NumberOfPoints

Purpose   Fixed and free point status

Syntax   `D.PointTypes`

Description   `PointType` is a property of `mbcdoe.design`. Each point has a type of `free`, `fixed` or `data`.

You can specify `fixed` points. `free` is the default. If a point has been matched to data then it is of type `data`.

`D.PointTypes` returns a cell array of PointTypes, one for each design point. You cannot change a PointType of `data` to something else as the `data` is set by the test plan when matching the design to `data`.

You can use the method `FixPoints` to fix all the points in a design.

See Also   `FixPoints`; `Points`; `RemovePoints`

PredictedValue

Purpose   Predicted value of model at specified inputs

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

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

`y = PredictedValue(R,X)` evaluates the model at the specified inputs, where 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`, `mbcmodel.localresponse` and `mbcmodel.response` objects, the X is optional. If X is not specified then the X is the existing input values. That is, the syntax is:

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

`y = PredictedValue(R)` calculates the predicted value at the fit data. An array is returned of predicted values evaluated at each data point. For local models, this is equivalent to `y= PredictedValue(L, L.InputData)`.

Note that you cannot evaluate model output for a local response or hierarchical response 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. For a local response, the predicted value uses the hierarchical model. If no data is specified then the data from all tests is used.

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

See Also   `PredictedValueForTest`, `ChooseAsBest`, `PEV`, `Evaluate`

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

Properties (for candidate sets)

Purpose   View and edit candidate set properties

Syntax
```
properties(CS)
CS.PropertyName = NewValue
```

Description   "Properties" is a method of mbcdoe.candidateset, which returns a list of properties.

properties(CS) lists the candidate set properties.

CS.PropertyName = NewValue sets the candidate set property.

The candidate set Type determines which properties you can set.

The following table lists the properties available for each candidate set type.

Candidate Set Properties (for Optimal Designs)

Candidate Set Type	Property	Description
All built-in: Grid/Lattice, Grid, Lattice, Stratified Lattice, Sobol, Halton	NumberOfPoints (read-only for Grid and Grid/Lattice)	Number of points (int: [0,Inf])
	Limits	Design Limits
Grid	Levels	Selection criteria for best LHS design (cell)
	NumberPerLevel	Symmetric design (vector int: {[-Inf,Inf], NumberOfInputs})
Lattice	Generators	Prime number generators for lattice (vector int: {[0,Inf], NumberOfInputs})

Candidate Set Properties (for Optimal Designs) (Continued)

Candidate Set Type	Property	Description
Stratified Lattice	StratifyLevels	Number of levels for each factors (vector int: {[0,Inf], NumberOfInputs})
Sobol Sequence	Scramble	Scramble method (enum: {'none', 'MatousekAffineOwen'}
	SkipMode	Skip mode options (enum: {'None','2^k','Custom'})
	Skip	Skip size (int: [0,Inf])
Halton Sequence	Scramble	Scrambling method for sequence (enum: {'None','RR2'})
	PrimeLeap	Leap sequence points using prime number (boolean)
	SkipZero	Skip zero point (boolean)
User-defined	NumberOfPoints	User-defined points (read-only)
	Points	User-defined points

Examples   You can use property value pairs to specify candidate set properties as part of the CreateCandidateSet command, or you can set properties individually.

To create a candidate set with type grid and specified grid levels:

Properties (for candidate sets)

```
CandidateSet = augmentedDesign.CreateCandidateSet...
( 'Type', 'Grid' );
CandidateSet.NumberOfLevels = [21 21 21 21];
```

See Also   CreateCandidateSet

Purpose   View and edit design constraint properties

Syntax   properties(C)
                  C.PropertyName = NewValue

Description   "Properties" is a method of mbcdoe.designconstraint, which returns
                  a list of properties.

                  properties(C) lists the constraint properties.

                  C.PropertyName = NewValue sets the constraint property.

                  The constraint Type determines which properties you can set. For
                  more information, see the following table or Type (for design
                  constraints).

                  The following table lists the properties available for each constraint
                  type.

Constraint Properties

Constraint Type	Property	Description
Linear design constraint: 1*Input1 + 1* Input2 + 1* Input3 <= 0	A	Matrix for linear constraint (matrix: [1,NumberOfInputs])
	b	Bound for linear constraint (double)
Ellipsoid design constraint: Ellipsoid at (Input1=0, Input2=0, Input3=0)	CenterPoint	Center of ellipse (vector: NumberOfInputs)
	Matrix	Ellipsoid form matrix (positive semi-definite) (matrix: [NumberOfInputs, NumberOfInputs])

Properties (for design constraints)

Constraint Properties (Continued)

Constraint Type	Property	Description
1D Table design constraint: InputY(InputX) <= InputY max	Table	Table constraint (vector)
	Breakpoints	Breakpoints for rows (vector)
	Inequality	Relational Operator (enum: {'<=','>='})
	InputFactor	Column input symbol (enum: {'InputX','InputY'})
	TableFactor	Table input symbol (enum: {'InputX','InputY'})

Constraint Properties (Continued)

Constraint Type	Property	Description
2D Table design constraint: InputZ(InputX,InputY) <=InputZmax	Table	: Table constraint (matrix))
	RowBreakpoints	Breakpoints for rows (vector)
	ColumnBreakpoints	Breakpoints for columns (vector)
	Inequality	Relational operator (enum: {'<=','>='})
	RowFactor	Row input symbol (enum: {'InputX','InputY, 'InputZ'})
	ColumnFactor	Column input symbol (enum: {'InputX','InputY, 'InputZ'})
	TableFactor	Table input symbol (enum: {'InputX','InputY', 'InputZ'}

Examples You can use property value pairs to specify constraint properties as part of the CreateConstraint command, or you can set properties individually.

For examples, see CreateConstraint.

See Also CreateConstraint

Properties (for design generators)

Purpose   View and edit design generator properties

Syntax   properties(Generator)
        Generator.PropertyName = NewValue

Description "properties" (lower case p) is a method of mbcdoe.generator, which returns a list of properties.

properties(Generator) lists the generator properties.

Generator.PropertyName = NewValue sets the generator property.

The design generator object Type determines which properties you can set. For more information, see Type (for designs and generators).

The settings are applied immediately, you do not need to call generate on the design object.

The following tables list the properties available for each design type.

Optimal Design Properties (D-, V- and A-Optimal)

Property	Description
NumberOfPoints	Number of points (int: [0,Inf])
InitialPoints	Initial design points (Matrix)
CandidateSet	Candidate set (mbcdoe.candidateset)
AllowReplicates	Allow replicate points (boolean)
AugmentMethod	Methods to add points (enum: {'random','optimal'})
Tolerance	Tolerance (numeric: 'positive')
MaxIterations	Maximum Iterations (int: 'positive')

Optimal Design Properties (D-, V- and A-Optimal) (Continued)

Property	Description
NumberOfPointsToAlter	Number of points to alter per iteration using the random augment method (p) (int: 'positive')
NoImprovement	Number of iterations with no improvement using the random augment method (p) (int: 'positive')

Note Optimal designs have dependencies between NumberOfPoints, InitialPoints and CandidateSets. When you change NumberOfPoints, an initial point is drawn from the existing candidate set. Setting NumberOfPoints updates InitialPoints. Likewise setting InitialPoints updates NumberOfPoints. When changing the candidate set a new initial design is drawn from the new candidate set.

Space-Filling Design Properties

Design Type	Property	Description
All space-filling design types (Lattice, Latin Hypercube Sampling, Stratified Latin Hypercube, Sobol, Halton)	NumberOfPoints	Number of points (int: [0,Inf])
	Limits	Design Limits (matrix: [NumberOfInputs,2])
Lattice	PrimeGenerators	Prime number generators for lattice for each input (vector int: [0,Inf])

Properties (for design generators)

Space-Filling Design Properties (Continued)

Design Type	Property	Description
Latin Hypercube Sampling and Stratified Latin Hypercube	SelectionCriteria	Selection criteria for best LHS design (enum: {'discrepancy', 'minimax', 'maximin', 'cdfvariance', 'cdfmaximum'})
	Symmetry	Symmetric design (boolean)
Stratified Latin Hypercube	StratifyLevels	Number of levels for each factors (vector int: {[0,Inf], NumberOfInputs})
	StratifyValues	Stratify levels (cell)
Sobol Sequence	Scramble	Scramble method (enum: {'none', 'MatousekAffineOwen'}
	SkipMode	Skip mode options (enum: {'None','2^k', 'Custom'})
	Skip	Skip size (int: [0,Inf])

Space-Filling Design Properties (Continued)

Design Type	Property	Description
Halton Sequence	Scramble	Scrambling method for sequence (enum: {'None','RR2'})
	PrimeLeap	Leap sequence points using prime number (boolean)
	SkipZero	Skip zero point (boolean)

Classical Design Properties

Design Type	Property	Description
All (Box-Behnken, Central Composite, Full Factorial, Plackett-Burman, Regular Simplex)	NumberOfPoints (read-only)	Number of points (int: [0,Inf])
	Limits	Design limits
All except Plackett-Burman	NumberOfCenterPoints	Number of center points (int: [0,Inf])

Properties (for design generators)

Classical Design Properties (Continued)

Design Type	Property	Description
Central Composite	StarPoints	Star point position (enum: {`'FaceCenteredCube'`, `'Spherical'`, `'Rotatable'`, `'Custom'`})
	Inscribe	Inscribe points (boolean)
	Alpha	Star point location (vector: {`'positive'`, NumberOfInputs})
Full Factorial	Levels	Cell array of levels for each input (cell)
	NumberOfLevels	Number of levels for each input (vector int: {`'positive'`, NumberOfInputs })

Examples   You can use property value pairs to specify design generator properties as part of the Generate and Augment commands. You can also set properties individually. Some examples:

To create a full factorial design and specify the number of levels when generating the design:

```
design = CreateDesign( inputs, 'Type', 'Full Factorial' );
design = Generate( design, 'NumberOfLevels', [50 50] );
```

To create a latin hypercube sampling design:

```
globalDesign = TP.CreateDesign(2,...
'Type', 'Latin Hypercube Sampling');
```

To create and generate a halton design with 50 points:

```
haltonDesign = CreateDesign( inputs, 'Type',...
  'Halton Sequence', 'Name', 'Halton' );
haltonDesign = Generate( haltonDesign, 50 );
```

To explicitly specify the NumberOfPoints property:

```
haltonDesign = Generate( haltonDesign, 'NumberOfPoints', 50 );
```

To create and generate a halton design with specified scrambling and other properties:

```
haltonDesignWithScrambling = haltonDesign.CreateDesign...
( 'Name', 'Scrambled Halton' );
haltonDesignWithScrambling = Generate...
( haltonDesignWithScrambling,...
  'Scramble', 'RR2', 'PrimeLeap', true );
```

To generate an optimal design with specified properties:

```
OptDesign = Generate(OptDesign,...
    'Type','V-optimal',...
    'CandidateSet',C,...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',200);
```

The previous code is equivalent to setting the properties individually and then calling Generate as follows:

```
P = OptDesign.Generator;
P.Type = `V-optimal';
P.CandidateSet.NumberOfLevels(:)=21;
P.MaxIterations = 200;
P.NumberOfPoints = 200;
P.NoImprovement = 50;
OptDesign.Generator = P;
```

Properties (for design generators)

To augment a design optimally with 20 points:

```
OptDesign = Augment(OptDesign,...
    'Type','V-optimal',...
    'MaxIterations',200,...
    'NoImprovement', 50,...
    'NumberOfPoints',20);
```

See Also   CreateDesign; Generate; Augment;Properties (for candidate
sets); Properties (for design constraints)

Purpose   View and edit model properties

Syntax
```
properties=M.Properties
M.Properties.PropertyName = NewValue
properties(M.Properties)
f=M.Properties.properties
```

Description   "`Properties`" is a property of `mbcmodel.model`.

`properties=M.Properties` returns a `mbcmodel.modelproperties` object.

To edit a property, use the syntax `M.Properties.PropertyName = NewValue`

"`properties`" is a method of `mbcmodel.fitalgorithm` and `mbcmodel.modelproperties` which returns a list of properties.

`properties(M.Properties)` lists the property names, types and allowed values.

`f=M.Properties.properties` returns the property names as a cell array.

The model `Type` determines which properties you can set. For more information, see `Type (for models)`.

To get a `mbcmodel.modelproperties` object from a model:

```
>> M = mbcmodel.CreateModel('Polynomial', 4);
>> disp(M)
mbcmodel.linearmodel:Polynomial

>>modelproperties=M.Properties

modelproperties =
Polynomial Properties
               Order: [3 3 3 3]
      InteractionOrder: 3
    TransformInputRange: 1
```

Properties (for models)

```
            ParameterNames: {35x1 cell}
            StepwiseStatus: {35x1 cell}
                    BoxCox: 1
```

To create a model and list the properties:

```
>> M = mbcmodel.CreateModel('RBF',2)

M =

   A radial basis function network using a multiquadric kernel
     with 0 centers
   and a global width of 2.
   The regularization parameter, lambda, is 0.0001.
   InputData: [0x2 double]
   OutputData: [0x1 double]
   Status: Not fitted
   Linked to Response: <not linked>

>> properties(M.Properties)
RBF Properties
     Kernel: RBF kernel (enum: {'multiquadric',...
'recmultiquadric','gaussian','thinplate','logisticrbf',...
'wendland', 'linearrbf','cubicrbf'})
     Continuity: Continuity for Wendland kernel...
 (0,2,4,6) (int: [0,6])
     ParameterNames: List of parameter names (read-only)
     StepwiseStatus: Stepwise status {'Always','Never',...
'Step'} (cell)
     BoxCox: Box-Cox transform (power) (numeric: [-3,3])
```

The following syntax returns the properties as a cell array:

```
>> f=M.Properties.properties

f =

     'Kernel'
```

```
'Continuity'
'ParameterNames'
'StepwiseStatus'
'BoxCox'
```

Change a property as follows:

```
>>M.Properties.Kernel = 'thinplate';
```

The model changes state to 'Being Edited'. The settings are not applied until you call Fit on the model object.

The following sections list the properties available for each model type.

Linear Models — Polynomial Properties

mbcmodel.linearmodel:Polynomial

Order: Polynomial order (vector int: {[0,Inf],NumberOfInputs})

InteractionOrder: Maximum order of interaction terms (int: [0,Inf])

TransformInputRange: Transform inputs (Boolean)

ParameterNames: List of parameter names (read-only)

StepwiseStatus: Stepwise status {'Always','Never','Step'} (cell)

BoxCox: Box-Cox transform (power) (numeric: [-3,3])

Linear Models — Hybrid Spline Properties

mbcmodel.linearmodel:Hybrid Spline

Order: Spline and polynomial order (vector int: {[0,3],NumberOfInputs})

SplineVariable: Spline variable

SplineInteraction: Order of interaction between spline and polynomial (int: [0,3])

Knots: Position of knots (vector real)

ParameterNames: List of parameter names (read-only)

StepwiseStatus: Stepwise status {'Always','Never','Step'} (cell)

BoxCox: Box-Cox transform (power) (numeric: [-3,3])

Linear Models — RBF Properties

mbcmodel.linearmodel:RBF

Kernel: RBF kernel (enum: {'multiquadric','recmultiquadric','gaussian','thinplate','logisticrbf','wendland', 'linearrbf','cubicrbf'})

Continuity: Continuity for Wendland kernel (0,2,4,6) (int: [0,6])

ParameterNames: List of parameter names (read-only)

StepwiseStatus: Stepwise status {'Always','Never','Step'} (cell)

BoxCox: Box-Cox transform (power) (numeric: [-3,3])

Linear Models — Polynomial-RBF Properties

mbcmodel.linearmodel:Polynomial-RBF

Order: Polynomial order (vector int: {[0,Inf],NumberOfInputs})

InteractionOrder: Maximum order of interaction terms (int: [0,Inf])

Kernel: RBF kernel (enum:

{'multiquadric','recmultiquadric','gaussian','thinplate','logisticrbf','wendland', 'linearrbf','cubicrbf'})

Continuity: Continuity for Wendland kernel (0,2,4,6) (int: [0,6])

ParameterNames: List of parameter names (read-only)

StepwiseStatus: Stepwise status {'Always','Never','Step'} (cell)

BoxCox: Box-Cox transform (power) (numeric: [-3,3])

Linear Models — Hybrid Spline-RBF Properties

mbcmodel.linearmodel:Hybrid Spline-RBF

Order: Spline and polynomial order (vector int: {[0,3],NumberOfInputs})

SplineVariable: Spline variable

SplineInteraction: Order of interaction between spline and polynomial (int: [0,3])

Knots: Position of knots (vector real)

Kernel: RBF kernel (enum: {'multiquadric','recmultiquadric','gaussian','thinplate','logisticrbf','wendland', 'linearrbf','cubicrbf'})

Continuity: Continuity for Wendland kernel (0,2,4,6) (int: [0,6])

ParameterNames: List of parameter names (read-only)

StepwiseStatus: Stepwise status {'Always','Never','Step'} (cell)

BoxCox: Box-Cox transform (power) (numeric: [-3,3])

Nonlinear Models — Free Knot Spline Properties

mbcmodel.model:Free Knot Spline

Order: Spline order (int: [0,3])

NumKnots: Number of knots (int: 'Positive')

Nonlinear Models — Neural Network Properties

mbcmodel.model:Neural Network

HiddenLayers: Number of hidden layers (int: [1,2])

Neurons: Number of Neurons in each hidden layer (vector int: 'Positive')

Examples

```
>> properties=M.Properties

properties =
Polynomial Properties
                   Order: [3 3 3 3]
        InteractionOrder: 3
      TransformInputRange: 1
           ParameterNames: {35x1 cell}
           StepwiseStatus: {35x1 cell}
```

Properties (for models)

```
                            BoxCox: 1


         >> M.Properties.Order = [3 2 2 3]

         M =

            1 + 2*X1 + 10*X4 + 15*X2 + 18*X3 + 3*X1^2 + 6*X1*X4
        ...+ 8*X1*X2 + 9*X1*X3 +
            11*X4^2 + 13*X4*X2 + 14*X4*X3 + 16*X2^2 + 17*X2*X3
        ...+ 19*X3^2 + 4*X1^3 +
            5*X1^2*X4 + 7*X1*X4^2 + 12*X4^3
            InputData: [0x4 double]
            OutputData: [0x1 double]
            Status: Being Edited
            Linked to Response: <not linked>
```

See Also   Type (for models), LocalModel Properties

Purpose   Number of records in each test

Syntax   numRecords = 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 = D.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`

RemoveDesign

Purpose   Remove design from test plan

Syntax
```
RemoveDesign(T,Name)
RemoveDesign(T,Level,Name)
RemoveDesign(T,D)
RemoveDesign(T,Level,D)
```

Description   `RemoveDesign` is a method of `mbcmodel.testplan`.

RemoveDesign(T,Name) removes a design with a matching name from the test plan T.

Name can be a string, or a cell array of strings.

RemoveDesign(T,Level,Name) removes a design with a matching name from the specified level of the test plan. By default the level is the outer level (i.e., Level 1 for one-stage, Level 2 (global) for two-stage).

RemoveDesign(T,D) removes D, an array of designs to be deleted. All designs with matching names are deleted.

RemoveDesign(T,Level,D) removes D from the specified level.

See Also   AddDesign; UpdateDesign; FindDesign

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

RemoveOutliers

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

RemoveOutliersForTest

Purpose   Remove outliers on test by index or rule and refit models

Syntax   L = RemoveOutliersForTest(LOCALRESPONSE, TESTNUMBER,
                     LOCALSELECTION)
                   L = RemoveOutliersForTest(LOCALRESPONSE, TESTNUMBER,
                     LOCALSELECTION, doUpdate)

Description   This is a method of mbcmodel.localresponse.

                  L = RemoveOutliersForTest(LOCALRESPONSE, TESTNUMBER,
                  LOCALSELECTION) removes outliers, refits the local model, and refits the
                  response feature models.

                  L = RemoveOutliersForTest(LOCALRESPONSE, TESTNUMBER,
                  LOCALSELECTION, doUpdate) removes outliers and if doUpdate is true,
                  refits all response features after the local model is refitted.

                  TESTNUMBER is the single test number to refit.

                  LOCALSELECTION can either be a set of indices or a function name.

                  An outlier selection function 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, and FACTORNAME is a
                  cell array of the names for each factor.

Examples   For a local response LOCALRESPONSE, to remove first two data points and
                  do not update response features:

                      RemoveOutliersForTest(LOCALRESPONSE,1,1:2,false);

                  To find list of indices of removed data points:

                      indices = OutliersForTest(LOCALRESPONSE,1);

                  To restore first data point:

```
RestoreDataForTest(LOCALRESPONSE,1,1,false);
```

To restore all data:

```
RestoreDataForTest(LOCALRESPONSE,1,':',false);
```

To update response features:

```
UpdateResponseFeatures(LOCALRESPONSE);
```

See Also   UpdateResponseFeatures, RestoreDataForTest,
OutlierIndicesForTest, RemoveOutliers

RemovePoints

Purpose	Remove all nonfixed design points
Syntax	D = RemovePoints(D) D = RemovePoints(D,PointType) D = RemovePoints(D,indices)
Description	RemovePoints is a method of mbcdoe.design. D = RemovePoints(D) removes all nonfixed points from the design. D = RemovePoints(D,PointType) removes the specified type of points, where PointType is one of 'free','fixed' or 'data'. D = RemovePoints(D,indices) removes the points specified by indices.
Examples	To remove all free points: Design = RemovePoints(Design,'free');
See Also	FixPoints

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

RemoveVariable

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`

Purpose   Response for model object

Syntax   R = 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 = model.Response;

See Also   UpdateResponse, ModelSetup

ResponseFeatures(Local Model)

Purpose   Set of response features for local model

Syntax   RFs = L.ResponseFeatures

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

RFs = L.ResponseFeatures returns a `mbcmodel.responsefeatures` object. `L` is the local model.

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

Available properties and methods are described in the following tables.

Property	Description
EvaluationPoints	Cell array of evaluation points for the response feature set (read-only). An element of EvaluationPoints is empty if the response feature does not use the Evaluation point. This property is set up when the response feature is created (see the Add method).
Types	Cell array of types for response feature set (read-only). This property is set up when the response feature is created (see the Add method).
NumberOfResponseFeatures	Number of response features in set (read-only).
IsFitted	The local model has been fitted.

Method	Description

Add	Add new response feature to response feature set
	
	`RF = Add(RF,RFtype)`
	
	`RFtype` is a description string belonging to the set of alternative response features. See getAlternativeTypes.
	
	`RF = Add(RF,RFtype,EvaluationPoint)`
	
	`EvaluationPoint` is a row vector with an element for each model input and is used for response features that require an input value to evaluate the response feature (e.g., function evaluation, derivatives). It is an error to specify an evaluation point for a response feature type that does not require an evaluation point.

Remove	Remove a response feature from the response feature set `RF = Remove(RF,index)`
Select	Select a subset of response features from the response feature set `RF = Select(RF,indices)`
getDefaultSet	List of default response features `RF = getDefaultSet(RF)` Returns an `mbcmodel.responsefeatures` object with the default set of response features for the local model.

ResponseFeatures(Local Model)

Method	Description
getAlternativeTypes	List of all alternative response feature types for local model ```RFtypes = getAlternativeTypes(RF)``` Returns a cell array of response feature type strings for the local model.

Evaluate	Evaluate response features
	
	```rfvals = Evaluate(RF);```
	
	Returns the values for the response features for the current local model.
	
	```[rfvals,stderr] = Evaluate(RF)```
	
	Also returns the standard errors for the response features for the current local model. The local model must be fitted before evaluating response features.

Jacobian	Jacobian matrix of response features with respect to parameters ```J = Jacobian(RF)``` The local model must be fitted before calculating the Jacobian matrix.
Covariance	Covariance matrix for response features ```rfvals = Covariance(RF);``` The local model must be fitted before calculating the covariance matrix.

Method	Description
Correlation	Correlation matrix for response features `rfvals = Correlation(RF)` Errors occur if model is not fitted.
ReconstructSets	List of subsets of response features which can be used to reconstruct the local model `RFlist = ReconstructSets(RF)` `RFlist` is a cell array of `mbcmodel.responsefeatures`. Each element of `RFlist` can be used to reconstruct the local model from response feature values.

Examples First, create a local model object:

```
L = mbcmodel.CreateModel('Local Polynomial',2)


L =

   1 + 2*X1 + 8*X2 + 3*X1^2 + 6*X1*X2 + 9*X2^2 + 4*X1^3...
 + 5*X1^2*X2 + 7*X1*X2^2 +
   10*X2^3
   InputData: [0x2 double]
   OutputData: [0x1 double]
   Status: Not fitted
   Linked to Response: not linked
```

The properties of the local model object are the same as the properties of an `mbcmodel.model` object with the additional property "ResponseFeatures". Look at the response features property as follows:

ResponseFeatures(Local Model)

```
>> RFs = L.ResponseFeatures

RFs =

Response features for Polynomial
    'Beta_1'
    'Beta_X1'
    'Beta_X1^2'
    'Beta_X1^3'
    'Beta_X1^2*X2'
    'Beta_X1*X2'
    'Beta_X1*X2^2'
    'Beta_X2'
    'Beta_X2^2'
    'Beta_X2^3'

% Set up response features
RFtypes = getAlternativeTypes(RFs);
RF = Add(RF, RFtypes{end},-10);

% assign to local model
L.ResponseFeatures = RFs;
```

Purpose   Array of response features for local response

Syntax   RFs = 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 = Local.ResponseFeatures;

ResponseSignalName

Purpose	Name of signal or response feature being modeled
Syntax	ysignal = R.ResponseSignalName
Description	This is a property of all response objects: mbcmodel.hierarchicalresponse, mbcmodel.localresponse and mbcmodel.response.
	R can be a hierarchical response, local response or response.
Examples	yName = local.ResponseSignalName;
See Also	InputSignalNames

Purpose   Array of available responses for test plan

Syntax   R = 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 = T.Responses;

RestoreData

Purpose	Restore removed outliers
Syntax	R = RestoreData(RESPONSE) R = RestoreData(RESPONSE, OUTLIERINDICES)
Description	This is a method of mbcmodel.localresponse and mbcmodel.response. R = RestoreData(RESPONSE) restores all data previously removed as outliers. R = RestoreData(RESPONSE, OUTLIERINDICES) restores all removed data specified in OutlierIndices. For a local response, the indices refer to record numbers for all tests.
Examples	RemoveOutliers(R, 1:5) RestoreData(R, 1:2)
See Also	RemoveOutliersForTest, RemoveOutliers, OutlierIndices

Purpose   Restore removed outliers for test

Syntax   L = RestoreDataForTest(LOCALRESPONSE, TESTNUMBER, Indices)
                 L = RestoreDataForTest(LOCALRESPONSE, TESTNUMBER, Indices,
                   doUpdate)

Description This is a method of mbcmodel.localresponse.

L = RestoreDataForTest(LOCALRESPONSE, TESTNUMBER, Indices) restores all removed data for TESTNUMBER specified in Indices.

L = RestoreDataForTest(LOCALRESPONSE, TESTNUMBER, Indices, doUpdate) restores all specified removed data and if doUpdate is true, refits all response features. By default, all response feature models will be updated. If a number of tests are being screened it is more efficient to set doUpdate to false and call UpdateResponseFeatures when all the tests have been screened.

Indices must be numbers and must belong to the set of outliers in OutliersForTest.

Examples   For a local response LOCALRESPONSE, to remove first two data points without updating response features:

    RemoveOutliersForTest(LOCALRESPONSE,1,1:2,false);

To find list of indices of removed data points:

    indices = OutliersForTest(LOCALRESPONSE,1);

To restore first data point:

    RestoreDataForTest(LOCALRESPONSE,1,1,false);

To restore all data:

    RestoreDataForTest(LOCALRESPONSE,1,':',false);

To update response features:

RestoreDataForTest

```
UpdateResponseFeatures(LOCALRESPONSE);
```

See Also UpdateResponseFeatures, RemoveOutliersForTest,
OutlierIndicesForTest

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

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

RollbackEdit

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 = D1.IsBeingEdited
```

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

See Also   BeginEdit, CommitEdit, IsBeingEdited

Purpose	Save project to currently selected filename
Syntax	OK = Save(P, Name)
Description	This is a method of `mbcmodel.project`.
	The project name is used as the filename if none is specified.
Examples	`OK = Save(proj, 'Example.mat');`
See Also	SaveAs

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

Purpose   Plot design points

Syntax   Scatter2D(D,Xindex,Yindex)
                   Scatter2D(D,xindex,yindex,plotArguments)

Description   Scatter2D is a method of mbcdoe.design.

Scatter2D(D,Xindex,Yindex) creates a scatter plot of the design points in design D, where X and Y are the indices or symbols for the input factors to plot on the X and Y axis.

Scatter2D(D,xindex,yindex,plotArguments) creates a scatter plot with additional arguments.plotArguments specifies additional arguments to the MATLAB® plot command. The plot command used in Scatter2D is

```
plot(D.Points(:,v1),D.Points(:,v2),varargin{:})
```

The default for varargin is '.'.

Examples   Scatter2D(mainDesign, 1, 2);

SetTermStatus

Purpose	Set status of model terms
Syntax	M.Properties = M.Properties.SetTermStatus(Terms, Status)
Description	This is a method of mbcmodel.linearmodelproperties.

M.Properties = M.Properties.SetTermStatus(Terms, Status) sets the status of the specified terms in this model. Status must be a cell array of status strings.

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.

M is an mbcmodel.linearmodel object.

Examples
```
M = mbcmodel.CreateModel('Polynomial', 2);
M.Properties = M.Properties.SetTermStatus([1 2; 1 0],
 {'Never', 'Always'});
```

This example sets the status of the X1*X2^2 term to Never and the X1 term to Always.

See Also GetTermStatus, StepwiseStatus

Purpose   Open fit algorithm setup dialog box

Syntax   `[OPT,OK]= SetupDialog(F)`

Description   This is a method of `mbcmodel.fitalgorithm`.

`[OPT,OK]= SetupDialog(F)` opens the fit algorithm setup dialog box, where you can edit the algorithm parameters. `F` is a `mbcmodel.fitalgorithm` object.

If you click **Cancel** to dismiss the dialog, `OK = false` and no changes are made. If you click **OK** to close the dialog box, then `OK = true` and your new chosen algorithm parameters are set up.

Examples   `[OPT,OK]= SetupDialog(F)`

See Also   `CreateAlgorithm`, `getAlternativeNames`

SignalNames

Purpose	Names of signals held by data
Syntax	names = 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 = D.SignalNames;
See Also	`SignalUnits`, `InputSignalNames`, `Value`

Purpose   Names of units in data

Syntax   units = 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 = D.SignalUnits;

See Also   SignalNames

SingleVIF

Purpose　　　Single VIF matrix for linear model parameters

Syntax　　　`VIF = SingleVIF(LINEARMODEL)`

Description　This is a method of `mbcmodel.linearmodel`.

　　　　　　`VIF = SingleVIF(LINEARMODEL)` calculates the single Variance Inflation Factor (VIF) matrix for the linear model parameters.

Examples　　　`VIF = SingleVIF(knot_model)`

See Also　　`ParameterStatistics`

SizeOfParameterSet

Purpose   Number of model parameters

Syntax   N = 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 = knotparams.SizeOfParameterSet

See Also   NumberOfParameters, StepwiseSelection, Names, Values

StatisticsDialog

Purpose   Open summary statistics dialog box

Syntax   [mdl,OK]= StatisticsDialog(mdl)

Description   This is a method of mbcmodel.model.

[mdl,OK]= StatisticsDialog(mdl) opens the Summary Statistics
dialog box, where you can select the summary statistics you want to use.

If you click **Cancel** to dismiss the dialog, OK = false and no changes
are made. If you click **OK** to close the dialog box, then OK = true and
your new chosen summary statistics are set up.

See Also   SummaryStatistics

Purpose   Model status: fitted, not fitted or best

Syntax   S = 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 = knot.Status
S =
    `Fitted'
```

See Also   ChooseAsBest

StepwiseRegression

Purpose   Change stepwise selection status for specified terms

Syntax   [S, model] = 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, knot] = 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, knot] = 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 = knot.Parameters;
N = params.StepwiseSelection

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

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

>> StepwiseRegression(knot, 4);
params = knot.Parameters;
N = 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 = 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 = 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 = 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 = paramsknot.StepwiseStatus
N =
    'Always'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
    'Step'
```

See Also   StepwiseRegression, StepwiseSelection

Purpose   Style of design type

Syntax   D.Style

Description   Style is a read-only property of mbcdoe.design.

D.Style

The style of the design is one of :

- 'User-defined'
- 'Optimal'
- 'Space-filling'
- 'Classical'
- 'Experimental data'

The read-only Style property is derived from the design Type.

See Also   Type (for designs and generators)

SummaryStatistics

Purpose   Summary statistics for response

Syntax   S = SummaryStatistics(M)
                 S = SummaryStatistics(M, Names)

Description This is a method of all model objects (mbcmodel.model
                 and mbcmodel.localmodel) and response objects
                 (mbcmodel.hierarchicalresponse, mbcmodel.localresponse, and
                 mbcmodel.response).

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

                 S = SummaryStatistics(M) returns summary statistics for the model
                 or response in a structure array containing Statistics and Names
                 fields.

                 S = SummaryStatistics(M, Names) returns summary statistics
                 specified by Names for the model or response in an array. Names can be a
                 char array, or a cell array of strings.

Examples   S = SummaryStatistics(R2);

See Also   DiagnosticStatistics, AlternativeModelStatistics

Purpose   Statistics for specified test

Syntax   SS = SummaryStatisticsForTest(LocalResponse, TestNumber)
                SS = SummaryStatisticsForTest(LocalResponse,TestNumber,Names)

Description This is a method of `mbcmodel.localresponse`.

SS = SummaryStatisticsForTest(LocalResponse, TestNumber)
returns a structure array containing Statistics and Names fields values
for the local model for test TestNumber.

SS =
SummaryStatisticsForTest(LocalResponse,TestNumber,Names)
returns an array of the statistics specified by Names. Names can be a
char array, or a cell array of strings.

Examples   SS = SummaryStatisticsForTest(L, 22)

See Also   SummaryStatistics

TestFilters

Purpose   Structure array holding user-defined test filters

Syntax   testf = data.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 = data.TestFilters;

See Also   AddTestFilter, ModifyTestFilter, RemoveTestFilter

Purpose   Array of test plan objects in project

Syntax   `tps = project.TestPlans`

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

P is the project object.

Examples   `tps = project.TestPlans;`

Type (for candidate sets)

Purpose   Candidate set type

Syntax   `C.Type`

Description   This is a property of `mbcdoe.candidateset`.

`C.Type` returns the candidate set type. You can only choose a type when you create the candidate set, when calling `CreateCandidateset`.

You can specify the candidate set type during creation by using the `Type` property, e.g.,

```
CandidateSet = augmentedDesign.CreateCandidateSet...
( 'Type', 'Grid' );
```

Other available properties depend on the candidate set type. To see the properties you can set, see the table of candidate set properties, Candidate Set Properties (for Optimal Designs) on page 2-158.

See Also   `CreateCandidateSet`

Purpose   Design type

Syntax   ```
D.Type
G.Type = NewType
```

**Description**      This is a read-only property of `mbcdoe.design`, and a settable property of `mbcdoe.generator`.

D.Type returns the design type. You can only choose a type when you create designs. After design creation, you can only set the Type of a `mbcdoe.generator` object, or when calling Generate or Augment.

G.Type = NewType changes the Type, where G is a `mbcdoe.generator` object.

The design Type determines which properties you can set. To set properties, see Properties (for design generators).

Get a list of types which could be used as alternative designs for current design, using `getAlternativeTypes`, by entering the following syntax:

```
Dlist = getAlternativeTypes(D)
```

where D is an `mbcdoe.design` object.

The design Type must be one shown in the following table. The read-only Style property is derived from the Type.

Style	Type
Optimal	D-Optimal
	V-Optimal
	A-Optimal

Type (for designs and generators)

Style	Type
Classical	Box-Behnken
	Central Composite
	Full Factorial
	Plackett-Burman
	Regular Simplex
Space-filling	Lattice
	Latin Hypercube Sampling
	Stratified Latin Hypercube
	Sobol Sequence
	Halton Sequence
Experimental data	Design points replaced by data points
Custom	Any design with a mix of Types (eg an optimally augmented space-filling design)

Examples To specify the Type while creating and then generating a design of a given size:

```
D = CreateDesign(mdl,'Type','Sobol Sequence')
D = Generate(D,128);
```

See Also Properties (for design generators); Generate; Augment

Type (for design constraints)

Purpose   Design constraint type

Syntax   `C.Type`

Description   This is a property of `mbcdoe.constraint`.

C.Type returns the design constraint type. You can only choose a type when you create the constraint, when calling CreateConstraint.

You can specify the constraint type during creation by using the Type property, e.g.,

```
c = D.CreateConstraint('Type','Linear')
```

Other available properties depend on the constraint type. See the table Constraint Properties on page 2-161.

The constraint Type must be one shown in the following table.

Constraint Type	**Description**
'Linear'	Linear design constraint: 1*Input1 + 1* Input2 + 1* Input3 <= 0
'Ellipsoid'	Ellipsoid design constraint: Ellipsoid at (Input1=0, Input2=0, Input3=0)
'1D Table'	1D Table design constraint: InputY(InputX) <= InputY max
'2D Table'	2D Table design constraint: InputZ(InputX,InputY) <=InputZmax

See Also   CreateConstraint; Constraint Properties on page 2-161

Type (for models)

Purpose   Valid model types

Syntax
```
model.Type
M = mbcmodel.CreateModel(Type, NUMINPUTS)
M2 = CreateModel(M, Type)
```

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

`model.Type` returns the model type. This property is set at creation time. See `CreateModel`.

The model Type determines which properties you can set. To set properties, see `Properties (for models)`, and `LocalModel Properties`.

> **Note** Spaces and case in model `Type` are ignored.

The model type must be one shown in the following table.

Type	Model Object
Polynomial	mbcmodel.linearmodel
Hybrid Spline	mbcmodel.linearmodel
RBF	mbcmodel.linearmodel
Hybrid RBF	mbcmodel.linearmodel
Polynomial-RBF	mbcmodel.linearmodel
Hybrid Spline-RBF	mbcmodel.linearmodel
Multiple Linear	mbcmodel.linearmodel
Free Knot Spline	mbcmodel.model
Transient	mbcmodel.model
User-Defined	mbcmodel.model

Type	Model Object
Neural Network	mbcmodel.model
Local Polynomial Spline	mbcmodel.localmodel
Local Polynomial with Datum	mbcmodel.localmodel
Local Polynomial	mbcmodel.localmodel
Local Hybrid Spline	mbcmodel.localmodel
Local Truncated Power Series	mbcmodel.localmodel
Local Free Knot Spline	mbcmodel.localmodel
Local Multiple Models	mbcmodel.localmodel
Local Growth Model	mbcmodel.localmodel
Local User-defined Model	mbcmodel.localmodel
Local Transient Model	mbcmodel.localmodel
Local Average Fit	mbcmodel.localmodel

Get a list of types, using `getAlternativeTypes`, by entering the following syntax:

```
Mlist = getAlternativeTypes(M)
```

where `M` is an `mbcmodel.model` object.

Create an alternative model as follows: `M = mbcmodel.CreateModel(Type, NUMINPUTS)` or `M2 = CreateModel(M, Type)`.

See Also   Properties (for models), getAlternativeTypes, CreateModel

Units

Purpose   Model output units

Syntax
```
model.Units
modelinput.Units
```

Description   This is a property of `mbcmodel.model` and `mbcmodel.modelinput` objects.

`model.Units` or `modelinput.Units` return the units of the model or modelinput object.

This property is set to the data signal units when the response is created or if a model is assigned to a response. This property cannot be set when a response is attached to the model.

Purpose	Update design in test plan
Syntax	D = UpdateDesign(T,D) D = UpdateDesign(T,Level,D)
Description	UpdateDesign is a method of mbcmodel.testplan. You must call UpdateDesign to replace an edited design in the test plan.
	D = UpdateDesign(T,D)
	D = UpdateDesign(T,Level,D)
	D is the array of designs to be updated in the test plan, T.
	Level is the test plan level. By default the level is the outer level (i.e., Level 1 for One-stage, Level 2 (global) for Two-stage).
	The design Name is used to decide what to update. If no name match is found in the test plan, the design is added.
	Design names must be unique so any repeated names will be changed. The array of designs is an output.
See Also	AddDesign; RemoveDesign; FindDesign

UpdateResponse

Purpose	Replace model in response
Syntax	UpdateResponse(model) M = UpdateResponse(M , R); updates the response specified by R
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.

```
UpdateResponse( M )
```

updates the model in the response associated with the model.

```
M = UpdateResponse( M , R);
```

updates the response specified by R.

Examples	UpdateResponse(knot);
See Also	ModelSetup

Purpose   Refit response feature models

Syntax   UpdateResponseFeatures(L)

Description   This is a method of mbcmodel.localresponse.

UpdateResponseFeatures(L) refits all response feature models. You
need to call this if you used RemoveOutliersForTest without specifying
refitting the response features (doUpdate set to false).

Examples   For a local response LOCALRESPONSE, to remove first two data points
without updating response features:

    RemoveOutliersForTest(LOCALRESPONSE,1,1:2,false);

To update response features:

    UpdateResponseFeatures(LOCALRESPONSE);

See Also   RemoveOutliersForTest, RestoreDataForTest

UserVariables

Purpose   Structure array holding user-defined variables

Syntax   userV = 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 = 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 = 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 = paramsknot.Values;`

See Also   `Names`

Widths

Purpose	Width data from RBF model
Syntax	`Width = 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 = params.Widths;`
See Also	`Centers`