



Preface

MRBreach is a Matlab™ App to support the design of the sort of breach in a sea wall that is commonly required for managed realignment schemes. These schemes aim to increase the area of intertidal by allowing the sea back into areas of land that have been protected by sea defences in the past. It is usually impractical to remove the entire sea wall. There is therefore a need to determine the size of breach required to allow tidal exchange into and out of the site. In such design there is also the practical consideration of being able to construct the breach safely, in the time available over a tidal cycle. This piece of software implements a method proposed for the design of such breaches based on creating one, or more, stable channels into the site.

Requirements

The model is written in Matlab™ and provided as Open Source code (issued under a GNU General Public License) and runs under v2016b or later. MRBreach uses the `muitoolbox` and `dstoolbox`.

Resources

The MRBreach App and two toolboxes (`muitoolbox` and `dstoolbox`) can be downloaded from www.coastalsea.uk.

Bibliography

Townend, I.H., 2008a. Breach design for managed realignment sites. *Proc.Instn Civ.Engrs., Maritime Engineering*, 161(MA1), 9-21.

Townend, I.H., 2008b. Hypsometry of estuaries, creeks and breached sea wall sites. *Proc.Instn Civ.Engrs., Maritime Engineering*, 161(MA1), 23-32.

Revision history

Version	Date	Changes
3.1	Oct 2022	Added additional properties to the output tables
3.0	Oct 2021	Adapted to run as an App using <code>muitoolbox</code>
2.0	July 2019	Updated to be compatible with ModelUI v2
1.0	Jan.2018	First release via www.coastalsea.uk



Contents

1	Introduction	1
2	Getting started	2
2.1	Configuration.....	2
2.1.1	Installing the toolboxes	2
2.2	Installing the App	2
2.3	Model Set-up	2
3	Application Menus	4
3.1	File.....	4
3.2	Tools.....	4
3.3	Project.....	4
3.4	Setup.....	5
3.5	Run	5
3.6	Analysis.....	7
3.6.1	Plotting	7
3.6.2	Statistics.....	8
3.7	Help	11
3.8	Tabs	11
3.9	UI Data Selection	11
3.10	Accessing data from the Command Window	12
4	Supporting Information	14
4.1	Model Outputs.....	14
4.2	Derive Output.....	14
4.2.1	Calling an external function	15
4.2.2	Input and output format for external functions.....	16
4.2.3	Pre-defined functions	18
5	Program Structure.....	19
6	Bibliography.....	21
	Appendix A – Hypsometry file format.....	22



1 Introduction

When creating an opening in a seawall to allow the tide to flow onto the land behind the seawall, as is often done in projects to realign the sea defences (so called managed realignment), there is a need to design the size of the breach required. This depends on the land levels of the site behind the seawall, as defined by the site hypsometry (variation in plan area with elevation), the tidal conditions and the soil properties of the newly formed breach channel. With this information it is possible to estimate the regime or equilibrium channel (Townend, 2008a). Some analysis is also possible based on idealised descriptions of the site hypsometry (Townend, 2008b). The MRBreach model implements this design method, with the option to interactively compare theoretical and measured site hypsometry. Site properties can be varied, and the resultant regime channel sections compared.

For a review of managed realignment schemes, see Scott *et al.* (2011). A framework for scheme design is provided in Townend *et al.* (2010). Further information about individual managed realignment schemes across Europe can be found at: www.omreg.net.

2 Getting started

2.1 Configuration

MRBreach is installed as an App and requires `muitoolbox` and `dstoolbox` to be installed. The download for each of these includes the code, documentation and example files. The files required are:

`dstoolbox`: `dstoolbox.mltbx`

`muitoolbox`: `muitoolbox.mltbx`

The App file: `MRBreach.mlappinstall`

2.1.1 Installing the toolboxes

The two toolboxes can be installed using the *Add-Ons* > *Manage Add-Ons* option on the Home tab of Matlab™. Alternatively, right-click the mouse on the ‘`mltbx`’ files and select install. All the folder paths are initialised upon installation and the location of the code is also handled by Matlab™. The location of the code can be accessed using the options in the *Manage Add-Ons* UI.

2.2 Installing the App

The App is installed using the Install Apps button on the APPS tab in Matlab™. Alternatively, right-click the mouse on the ‘`mlappinstall`’ file and select install. Again all the folder paths are initialised upon installation and the location of the code is handled by Matlab™.

Once installed, the App can be run from the APPS tab. This sets the App environment paths, after which the App can be run from the Command Window using:

```
>> MRBreach;
```

The App environment paths can be saved using the Set Path option on the Matlab™ Home tab.

Documentation can be viewed from the App Help menu, or the Supplemental Software in the Matlab™ documentation. The location of the code can be accessed by hovering over the App icon and then finding the link in the pop-up window.

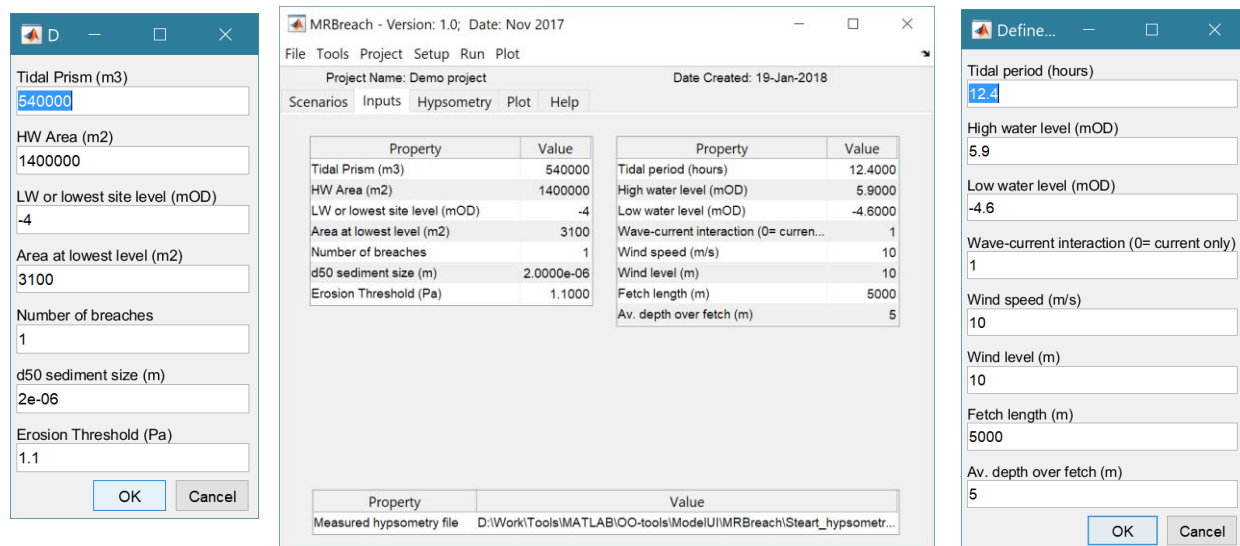
2.3 Model Set-up

File > *New* to create a new project space.

Setup > *Input Data* > *Site Data*

Setup > *Input Data* > *Hydraulic Data*

The UIs request data for the model variables. Once added the current set of variables can be viewed using the *Inputs* tab.



Property	Value	Property	Value
Tidal Prism (m3)	540000	Tidal period (hours)	12.4000
HW Area (m2)	1400000	High water level (mOD)	5.9000
LW or lowest site level (mOD)	-4	Low water level (mOD)	-4.6000
Area at lowest level (m2)	3100	Wave-current interaction (0= curren...	1
Number of breaches	1	Wind speed (m/s)	10
d50 sediment size (m)	2.0000e-06	Wind level (m)	10
Erosion Threshold (Pa)	1.1000	Fetch length (m)	5000
		Av. depth over fetch (m)	5

If some measured values of the site hypsometry are available, the data file can be loaded using:

Setup>Input Data>Hypsometry

The file format for the hypsometry data has two header lines and then two columns of data that define the elevations (mOD) and plan areas (m²):

```
Steart hypsometry
Elevation    Cumulative area
0.03        40
0.06        81
0.09        133
0.12        189
0.15        232
0.18        275
```

Run> Set hypsometry

Generates a figure of the Empirical site hypsometry derived using the site tidal prism and plan areas at the highest and lowest tidal elevations in the site. If some observed data have been loaded this is also plotted. The sliders on the figure allow the fit parameters to be adjusted. The buttons allow the user to select whether to use the empirical fit, or the observed hypsometry in the model run.

The current empirical fit and the imported data can be viewed on the *Hypsometry* tab. The thicker line is the hypsometry currently selected for use in the model.

Run> Run model

When the run has completed the user is prompted to provide a description of the model run (scenario).

The run is listed on the *Scenarios* tab and the design regime section for the most recent run can be viewed on the *Q-Plot* tab. This figure includes the minimum rectangular section as well as the regime section (see Townend, 2008a for details).

Plot>Plot menu

The results from a run can be selected and plotted (regime section only). By using the Add button additional model runs can be included on the plot, allowing different scenarios to be compared.

3 Application Menus

The UI comprises a series of drop down menus that provide access to a number of commonly used functions such as file handling, management of run scenarios, model setup, running and plotting of the results. In addition, Tabs are used to display set-up information of the Cases that have been run. In this manual text in *Red italic* refers to drop down menus and text in *Green italic* refers to Tab titles.

3.1 File

File>New: clears any existing model (prompting to save if not already saved) and a popup dialog box prompts for Project name and Date (default is current date).

File>Open: existing models are saved as *.mat files. User selects a model from dialog box.

File>Save: save a file that has already been saved.

File>Save as: save a file with a new or different name.

File>Exit: exit the program. The close window button has the same effect.

3.2 Tools

Tools>Refresh: updates *Cases* tab.

Tools>Clear all>Project: deletes the current project, including setup parameters and all Cases.

Tools>Clear all>Figures: deletes all results plot figures (useful if a large number of plots have been produced).

Tools>Clear all>Cases: deletes all cases listed on the *Cases* tab but does not affect the model setup.

3.3 Project

Project>Project Info: edit the Project name and Date.

Project>Cases>Edit Description: select a scenario description to edit.

Project>Cases>Edit Data Set: edit a data set. Initialises a data selection UI to define the record to be edited and then lists the variable in a table so that values can be edited. The user can also limit the data set retrieved based on the variable range and the independent variable (X) or time. This can be useful in making specific edits (eg all values over a threshold or values within a date range). Using the Copy to Clipboard button also provides a quick way of exporting selected data.

Project>Cases>Save: select the Case to be saved from the list of Cases, select whether to save the Case as a *dstable* or a *table* and name the file. The dataset *dstable* or *table* are saved to a mat file.

Project>Cases>Delete: select the Case(s) to be deleted from the list of Cases and these are deleted (model setup is not changed).

Project>Cases>Reload: select a previous model run and reload the input values as the current input settings.

Project>Cases>View settings: display a table of the model input parameters used for a selected Case.

Project>Import/Export>Import: load a Case class instance from a Matlab binary 'mat' file. Only works for data sets saved using Export.

Project>Import/Export>Export: save a Case class instance to a Matlab binary 'mat' file.

These last two functions can be used to move Cases between projects or models.

NB: to export the data from a Case for use in another application (eg text file, Excel, etc), use the *Project>Cases>Edit Data Set* option to make a selection and then use the ‘Copy to Clipboard’ button to paste the selection to the clipboard.

3.4 Setup

The setup menu provides a series of menus to enable different components of the model to be defined.

Setup>Hydraulic Parameters: prompts parameters for tide and wave conditions.

Setup>Site Parameters: prompts for parameters that define the site (tidal prism, levels, etc).

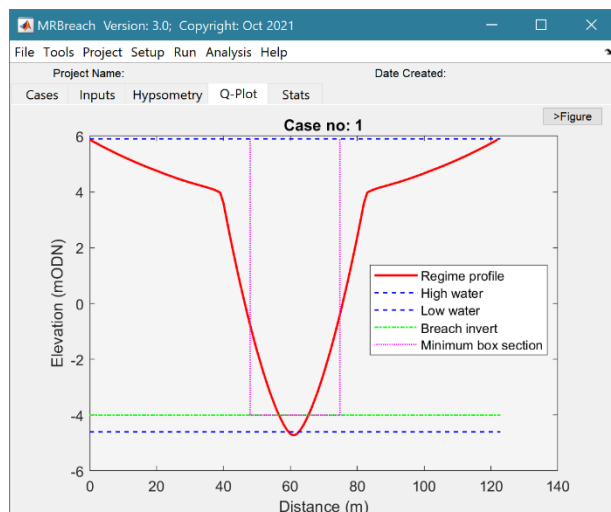
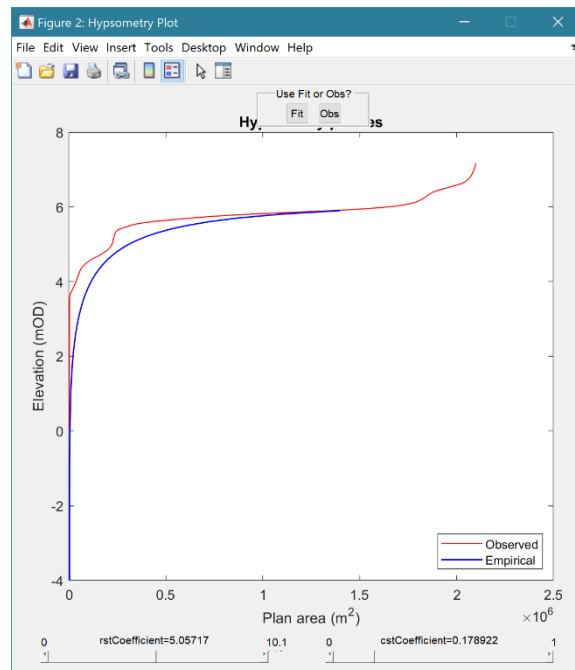
Setup>Site Hypsometry: allows the user to load a file defining the hypsometry of the existing site (see Appendix A – Hypsometry file format for details).

Setup> Model Constants: various constants are defined for use in models, such as the acceleration due to gravity, viscosity and density of sea water, and density of sediment. Generally, the default values are appropriate (9.81, 1.36e-6, 1025 , 2650 respectively) but these can be adjusted and saved with the project if required.

3.5 Run

Run> Set Hypsometry: generates figure of Empirical fit for hypsometry. Adds Observed hypsometry if loaded. Allows user to adjust the fit parameters and choose which hypsometry to use in the model.

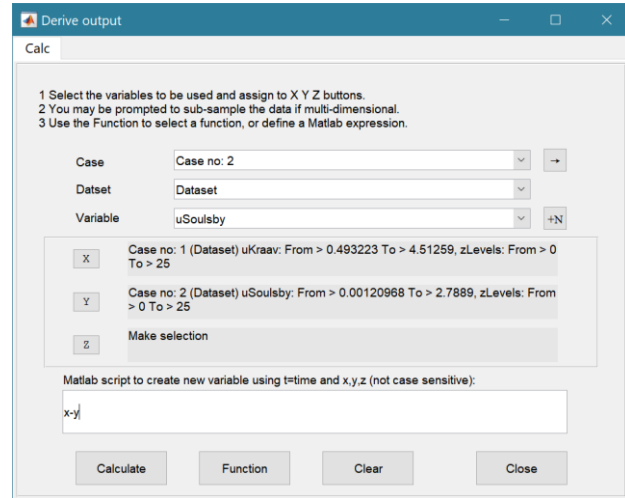
Run> Run Model: runs model, prompts for Case description which is added to the listing on the *Cases* tab.



Run > Derive Output: data that has been added (either as data or modelled values) can be used to derive new variables. The UI allows the user to select data and use a chosen selection of data/variable/range to define either a Variable, XYZ dimension, or Time. Each data set is sampled for the defined data range. If the data set being sampled includes NaNs the default is for these to be included (button to right of Var-limits is set to '+N'). To exclude NaNs press the button so that it displays '-N'.

The selection is assigned by clicking one of the X, Y or Z buttons. The user is prompted to assign a Variable, XYZ dimension, or Time (the options available varies with the type of variable selected) – see Section for details of how this works.

An equation is then defined in the text box below using the x, y, z or t variables¹. Based on the user selection the routine applies the defined variable ranges to derive a new variable. In addition text inputs required by the call and the model object (mobj) can also be passed.



Adding the comment `%time` or `%rows`, allows the the row dimension to be added to the new dataset. For example if x and y data sets are timeseries, then a MatlabTM expression, or function call, call can be used to create a new time series as follows:

```
x^2+y %time
```

The output from function calls can be figures or tables, a single numeric value, or a dataset to be saved (character vectors, arrays or dstables). External functions should return the table RowNames (e.g., time or location) as the first variable (or an empty first variable), followed by the other variables to be saved.

If there is no output to be passed back the function should return a string variable.

If `varout = 'no output'`; this suppresses the message box, which is used for single value outputs. For expressions that return a result that is the same length as one of the variables used in the call, there is the option to add the variable to the input dataset as a new variable. In all there are three ways in which results can be saved:

1. As a new dataset;
2. As an additional variable(s) to one of the input datasets;
3. As an additional variable(s) to some other existing dataset.

For options 2 and 3, the length of the new variables must be the same length (numero of rows) as the existing dataset.

An alternative when calling external functions is to pass the selected variables as dstables, thereby also passing all the associated metadata and RowNames for each dataset selected. For this option up to 3 variables (plus time if defined for a selected variable) can be selected but they are defined in the call using `dst`, for example:

```
[time,varout] = myfunction(dst, 'usertext', mobj);
```

¹ Various pre-defined function templates can be accessed using the 'Function' button. Alternatively, text can be pasted into the equation box from the clipboard by right clicking in the text box with the mouse.


```
dst = myfunction(dst, 'usertext', mobj);
```

This passes the selected variables as a struct array of dstables to the function. Using this syntax the function can return a dstable, or struct of dstables, or a number of variables. When a dstable, or struct of dstables is returned, it is assumed that the dsproperties have been defined in the function called and dstables are saved without the need to define the meta-data manually.

Some further details on using this option and the **'Function'** library available are provided in Section **XX**

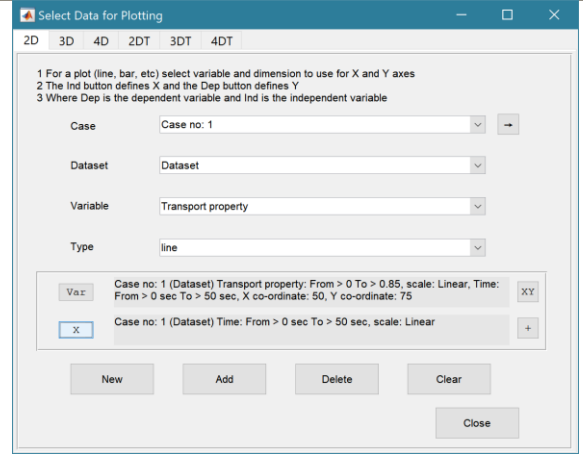
4.2.

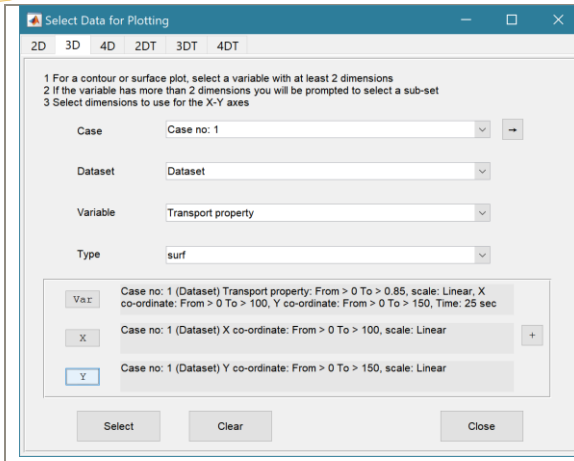
3.6 Analysis

Plotting and Statistical Analysis both use the standard Data selection UI. These both require Case, Dataset and Variables to be selected from drop-down lists and assigned to a button. Further details of how this works are given in Section 3.9.

3.6.1 Plotting

Analysis>Plot menu: initialises the Plot UI to select variables and produce several types of plot. The user selects the Case, Dataset, and Variable to be used and the plot Type from a series of drop-down lists. There are then buttons to create a New figure, or Add, or Delete variables from an existing figure for 2D plots, or simply a Select button for 3D and 4D plots. The following figures illustrate the options available.

	<p>2D plot</p> <p>For each selection choose the Case, Dataset and Variable to be used.</p> <ul style="list-style-type: none"> > Assign a variable, or a dimension, to the Var and X buttons to set the Y and X axes, respectively Each selection can be scaled (log, normalised, etc) and the range to be plotted can be adjusted when assigning the selection to a button. > Select plot type (line, bar, scatter, stem, etc) <p><u>Control Buttons:</u></p> <ul style="list-style-type: none"> → : updates the list of Cases XY : swaps the X and Y axes + : switches between cartesian and polar plot type <p><i>If polar selected then Ind assumed to be in degrees.</i></p>
---	---



3D plot

For each selection choose the Case, Dataset and Variable to be used.

> Assign selections to the Var, X and Y buttons

Take care to ensure that the assignments to X and Y correctly match the dimensions selected for the variable (including any adjustment of the dimension ranges to be used).

> Select plot type.

Control Buttons: see 2D plot above.

For all plot types, when the data has more dimensions than the plot or animation the user is prompted to sub-select from the data (by selecting sampling values for the dimensions that are not being used).

Selection of User plot type

Calls the user_plot.m function, where the user can define a workflow, accessing data and functions already provided by the particular App or the mui toolbox. The sample code can be found in the pfunctions folder and illustrates the workflow to a simple line plot using x-y data from the 2D tab and a surface plot using x-y-z data from the 3D tab.

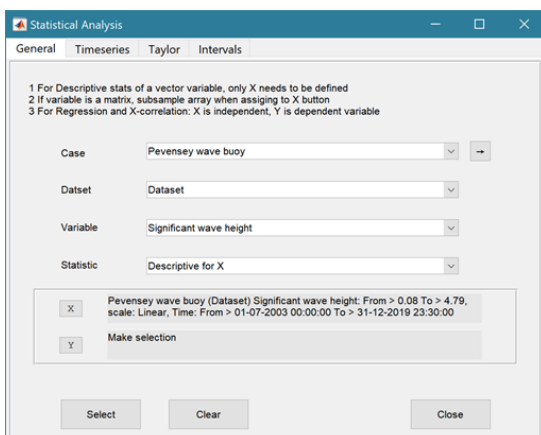
3.6.2 Statistics

Analysis > Statistics: several statistical analysis options have been included within the Statistical Analysis GUI. The tabs are for *General* statistics, *Timeseries* statistics, model comparisons using a *Taylor* Plot, and the generation of a new record based on the statistics over the *Intervals* defined by another timeseries.

General tab

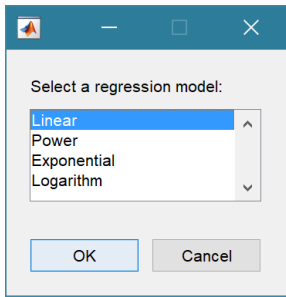
The General tab allows the user to apply the following statistics to data loaded in ModelUI:

1) **Descriptive for X**: general statistics of a variable (mean, standard deviation, minimum, maximum,



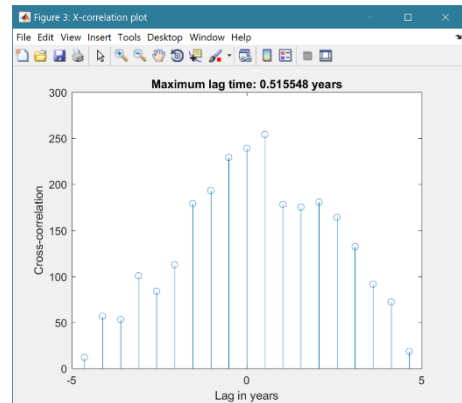
sum and linear regression fit parameters). Only X needs to be defined. The range of the variable can be adjusted when it is assigned to the X button (see Section 3.9). If the variable being used is a multi-dimensional matrix (>2D), the user is prompted to define the range or each additional dimension, or select a value at which to sample. The function can return statistics for a vector or a 2D array.

The results are tabulated on the *Stats > General* tab and can be copied to the clipboard for use in other applications.



2) **Regression:** generates a regression plot of the dependent variable, Y, against the independent variable, X. For time series data, the default data range is the maximum period of overlap of the two records. For other data types the two variables must have the same number of data points. After pressing the Select button, the user is prompted to select the type of model to be used for the regression. The results are output as a plot with details of the regression fit in the plot title.

3) **Cross-correlation:** generates a cross-correlation plot of the reference variable, X, and the lagged variable, X (uses the Matlab 'xcorr' function). For time series data, the default data range is the maximum period of overlap of the two records. For other data types the two variables must have the same number of data points. This produces a plot of the cross-correlation as a function of the lag in units selected by the user.



4) **User:** calls the function user_stats.m, in which the user can implement their own analysis methods and display results in the UI or add output to the project Catalogue. Currently implements an analysis of clusters as detailed for Timeseries data below.

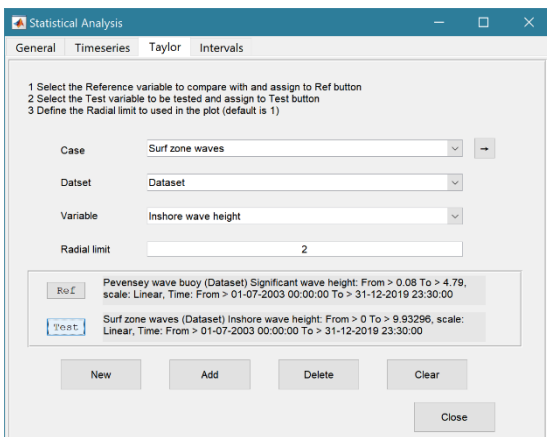
Taylor tab

The Taylor tab allows the user to create a Taylor Plot using 1D or 2D data (e.g timeseries or grids):

A Reference dataset and a Test dataset are selected. Datasets need to be the same length if 1D, or same size if 2D. If the data are timeseries they are clipped to a time-period that is common to both, or any user defined interval that lies within this clipped period. The statistics (mean, standard deviation, correlation coefficient and centred root mean square error) are computed, normalized using the reference standard deviation and plotted on a polar Taylor diagram (Taylor, 2001).

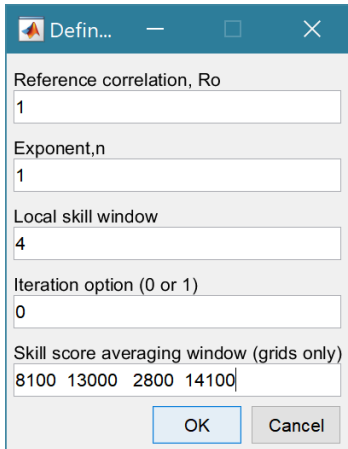
[The ModelSkill App provides additional tools to test data and the ModelSkill App manual provides further details of the methods used.]

Selecting New generates a new Taylor Plot. Selecting the Add button adds the current selection to an existing plot and the Delete button deletes the current selection. The Clear button resets the UI to a blank selection.



Once New or Add are selected, the user is asked whether they want to plot the skill score (Yes/No). If Yes, then the user is prompted to set the skill score parameters. As further points are added to the plot, this selection remains unchanged (i.e. the skill score is or is not included). To reset the option it is necessary to close and reopen the Statistics UI.

If the number of points in the Reference and Test datasets are not the same the user is prompted to select which of the two to use for interpolation.



This is the maximum achievable correlation (see Taylor (2001) for discussion of how this is used).

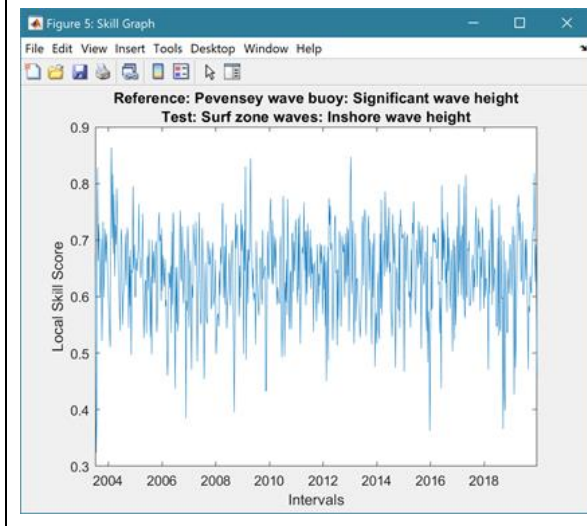
Exponent used in computing the skill score (see ModelSkill Manual for further details).

Number of points (+/-W) used to define a local window around the ith point. If W=0 (default) the local skill score is not computed.

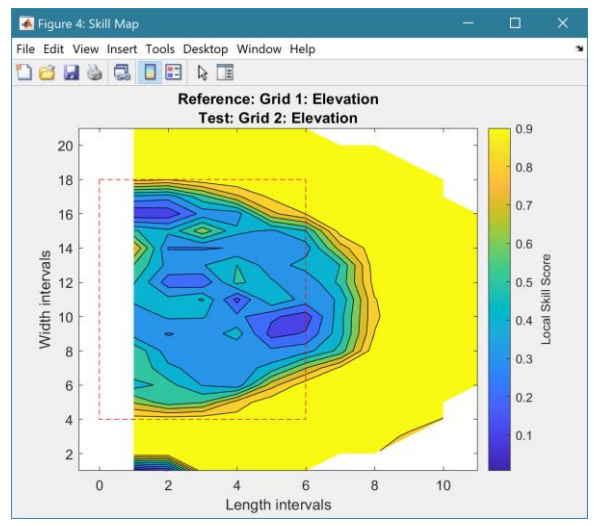
Local skill score is computed for window around every grid cell (=1), or computes score for all non-overlapping windows (=0)

Window definition to sub-sample grid for the computation of the average **local** skill score. Format is [xMin, xMax, yMin, yMax].

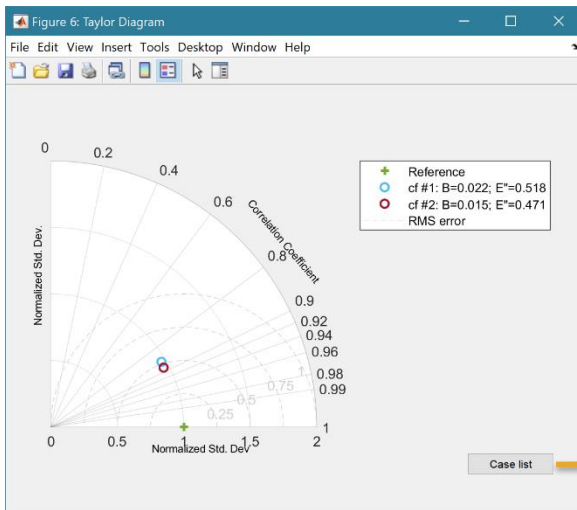
(a) time series skill score plot



(b) grid skill score plot

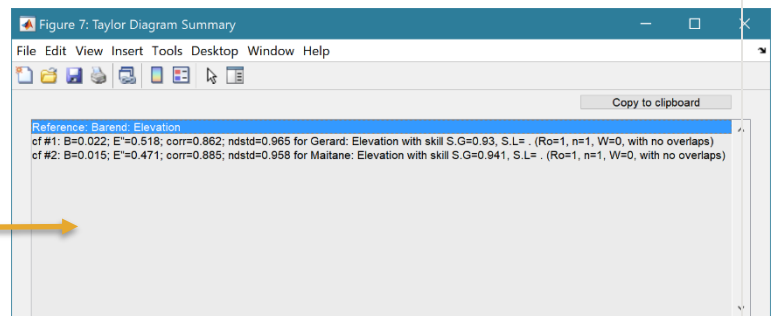


The Taylor Plot shows the Reference point as a green cross and the Test points as coloured circles. The legend details the summary statistics and the Case List button generate a table figure listing all the results. These can be copied to the clipboard.



Taylor diagram legend includes: B – bias; E' – normalised RMS difference

The normalised standard deviation and correlation coefficient are also given in the Case List table, along with the global skill score, Sg, and the average local skill score, Sl.



3.7 Help

The help menu provides options to access the App documentation in the Matlab™ Supplemental Software documentation, or the App manual.

3.8 Tabs

To examine what has been set-up the Tabs provide a summary of what is currently defined. Note: the tabs update when clicked on using a mouse but values displayed cannot be edited from the Tabs.

Cases: lists the cases that have been run with a case id and description. Clicking on the first column of a row generates a table figure with details of the variables for the case and any associated metadata. Buttons on the figure provide access the class definition metadata and any source information (files input or models used).

Inputs: tabulates the system properties that have been set (display only).

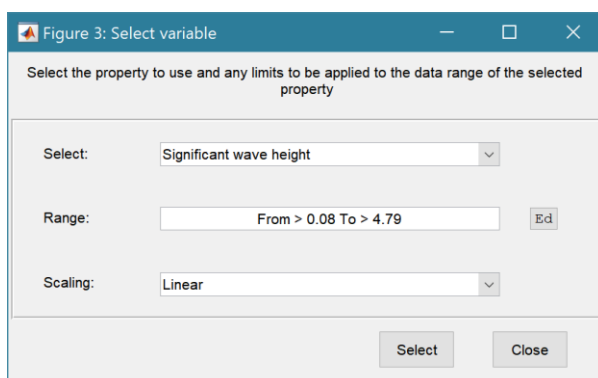
Hypsometry: displays the

Q-Plot: displays a quick-plot defined for the class of the selected case (display only).

Stats: displays a table of results for any analyses that have been run (can be copied to clip board).

3.9 UI Data Selection

Functions such as Derive Output (3.5), Plotting (3.6.1) and Statistics (3.6.2) use a standardised UI for data selection. The Case, Dataset and Variable inputs allow a specific dataset to be selected from drop down lists. One each of these has been set to the desired selection the choice is assigned to a button. The button varies with application and may be X, Y, Z, or Dependent and Independent, or Reference and Sample, etc. Assigning to the button enables further sub-sampling to be defined if required. Where an application requires a specific number of dimensions (e.g., a 2D plot), then selections that are not already vectors will need to be subsampled. At the same time, the range of a selected variable can be adjusted so that a contiguous window within the full record can be extracted. In most applications, any scaling that can be applied to the variable (e.g., linear, log, relative, scaled, normalised, differences) is also selected on this UI. The selection is defined in two steps:



Step 1.

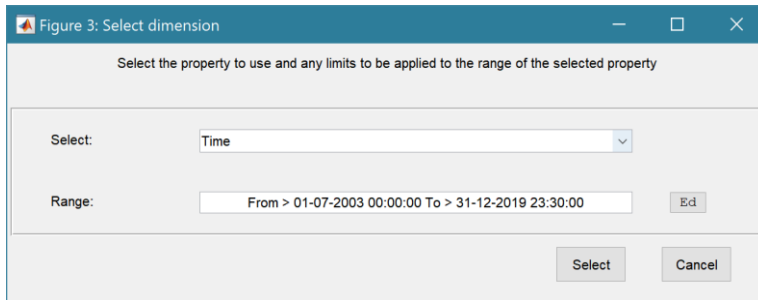
Select the attribute to use. This can be the variable or any of its associated dimensions, which are listed in the drop-down list.

The range for the selection can be adjusted by editing the text box or using the Edit (Ed) button.

Any scaling to be applied is selected from the drop-down list.

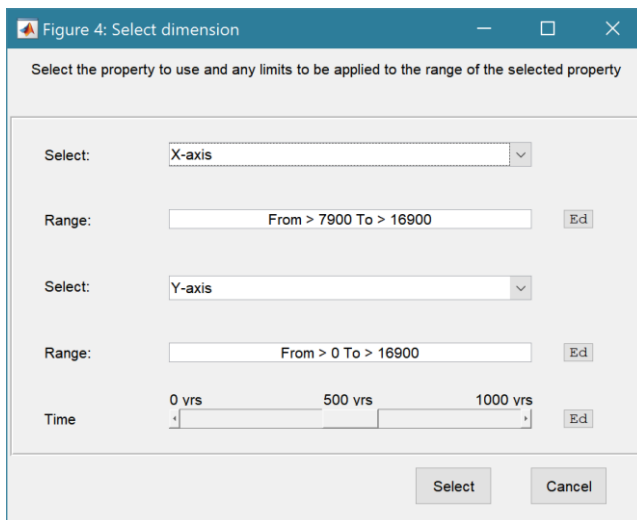
Press Select to go to the next step or Close to quit.

The number of variables listed on the UI depends on the dimensions of the selected variable. For each one Select the attribute to use and the range to be applied.



Step 2 - Variable only has dimension of time.

No selection to be made.
Edit range if required.



Step 2 - Variable has 3 dimensions but only 2 are needed for the intended use.

Select the 1st variable to use as a dimension.
Edit range if required.

Select the 2nd variable to use as a dimension.
Edit range if required.

Use the slider or the Edit (Ed) button to set the value of the dimension to use. (A value of t=500 is selected in the example shown).

Press Select to accept the selection made.

[NB: Only unused dimensions can be selected from the Select drop-down lists. To adjust from the default list this can sometimes require that the second Select list-box is set first to allow the first Select list-box to be set to the desired value.]

The resulting selection is then detailed in full (including the ranges or values to be applied to all dimensions) in the text box alongside the button being defined.

Note where a variable is being selected as one property and a dimension as a second property, any sub-selection of range must be consistent in the two selections. This is done to allow variables and dimensions to be used as flexibly as possible.

3.10 Accessing data from the Command Window

In addition to the options to save or export data provided by the *Project>Cases>Save* and *Project>Import/Export* options, data can also be accessed directly for use in Matlab™, or to copy to other software packages. This requires use of the Command Window in Matlab™, and a handle to the App being used. To get a handle, open the App from the Command Window as follows:

```
>> myapp = <AppName>;           e.g., >> as = Asmita;
```

Simply typing:

```
>> myapp
```

Which displays the results shown in the left column below with an explanation of each data type in the right hand column.

<pre>myapp = <AppName> with properties:</pre>	<p>Purpose</p>
---	-----------------------



Inputs: [1×1 struct]	A struct with field names that match all the model parameter input fields currently
Cases: [1×1 muiCatalogue]	muiCatalogue class with properties DataSets and Catalogue. The former holds the data the latter the details of the currently held records.
Info: [1×1 muiProject]	muiProject class with current project information such as file and path name.
Constants: [1×1 muiConstants]	muiConstants class with generic model properties (e.g. gravity, etc).

To access current model settings, use the following:

```
>> myapp.Inputs.<InputClassName>
```

To access the listing of current data sets, use:

```
>> cs.Cases.Catalogue
```

To access imported or model data sets, use:

```
>> cs.Cases.DataSets.<DataClassName>
```

If there are more than one instance of the model output, it is necessary to specify an index. This then provides access to all the properties held by that data set. Two of these may be of particular interest, RunParam and Data. The former holds the input parameters used for that specific model run.

RunParam is a struct with fields that are the class names required to run the model (similar to Inputs above). The Data property is a model specific struct with field names defined in the code for the model class. If there is only a single table assigned this will be given the field name of 'Dataset'. To access the *dstable* created by the model, use:

```
>> cs.Cases.DataSets.<DataClassName>(idx).Data.Dataset
```

```
>> cs.Cases.DataSets.<DataClassName>(idx).Data.<ModelSpecificName>
```

To access the underlying *table*, use:

```
>> cs.Cases.DataSets.<DataClassName>(idx).Data.Dataset.DataTable
```

The result can be assigned to new variables as required. Note that when assigning *dtables* it may be necessary to explicitly use the copy command to avoid creating a handle to the existing instance and potentially corrupting the existing data.

4 Supporting Information

4.1 Model Outputs

The output from MRBreach is stored in two ‘dstables’. These are held in the catalogue as the *Section* and *Properties* Datasets. They can be accessed for plotting and statistics by selecting the required Case and Dataset in the appropriate UI. The variables held in each dstable are as follows:

Dataset	Variable	Description
Section	zCoords	Elevation held as a function of the Width which is stored as the X dimension.
Properties	Water level	The tide level for the rising limb of the tide from LW to HW as a function of time.
	Horizontal velocity	The velocity in the breach as a function of time
	Vertical velocity	The rate of vertical rise of the water surface as a function of time.
	Total plan area	The change in surface area of the site as a function of time.

The Q-Plot tab provides a bespoke plot the Section output, together with the minimum rectangular breach section, the invert level used and details of the peak velocity in the breach over the flooding tide.

4.2 Derive Output

The *Run> Derive Output* option allows the user to make use of the data held within App to derive other outputs or, pass selected data to an external function (see Section 3.5). The equation box can accept t, x, y, z in upper or lower case. Time can be assigned to X, Y, or Z buttons, or simply included in the equation as t (as long as the data being used in one of the variables includes a time dimension). Each data set is sampled for the defined data range. If the data set being sampled includes NaNs, the default is for these to be included (button to right of Variable is set to ‘+N’). To exclude NaNs press the button so that it displays ‘-N’. The selection is based on the variable limits defined whenever a variable is assigned to X, Y or Z using the X, Y, Z buttons.

The equation string entered in the UI is used to construct an anonymous function as follows:

```
heq = str2func(['@(t,x,y,z,mobj) ',inp.eqn]); %handle to anonymous function
```

```
[varout{:}] = heq(t,x,y,z,mobj);
```

or when using dstables:

```
heq = str2func(['@(dst,mobj) ',inp.eqn]); %handle to anonymous function
```

```
[varout{:}] = heq(dst,mobj);
```

This function is then evaluated with the defined variables for t, x, y, and z and optionally mobj, where mobj passes the handle for the main UI to the function. Some functions may alter the length of the input variables (x, y, z, t), or return more than one variable. In addition, the variables selected can be sub-sampled when each variable is assigned to the X, Y, or Z buttons. The dimensions of the vector or array with these adjustments applied need to be dimensionally correct for the function being called. This may influence how the output can be saved (see Section 4.2.2).

If the function returns a single valued answer, this is displayed in a message box, otherwise it is saved, either by adding to an existing dataset, or creating a new one (see Section 4.2.2 and 3.5).

NB1: functions are forced to lower case (to be consistent with all Matlab functions), so any external user defined function call must be named in lower case.

Equations can use functions such as `diff(x)` - difference between adjacent values - but the result is `n-1` in length and may need to be padded, if it is to be added to an existing data set. This can be done by adding a NaN at the beginning or the end:

e.g.: `[NaN;diff(x)]`

NB: the separator needs to be a semi-colon to ensure the correct vector concatenation. Putting the NaN before the equation means that the difference over the first interval is assigned to a record at the end of the interval. If the NaN is put after the function, then the assignment would be to the records at the start of each interval.

Another useful built-in function allows arrays to be sub-sampled. This requires the array, `z`, to be multiplied by an array of the same size. By including the dimensions in a unitary matrix, the range of each variable can be defined. For a 2D array that varies in time one way of doing this is:

```
>> [z.*repmat(1, length(t), length(x), length(y))]
```

NB2: the order of the dimensions `t, x, y` must match the dimensions of the array, `z`.

NB3: When using Matlab compound expressions, such as the above sub-sampling expression, the expression must be enclosed in square brackets to distinguish it from a function call.

Adding the comment `%time` or `%rows`, allows the the row dimension to be added to the new dataset. For example if `x` and `y` data sets are timeseries, then a Matlab™ expresion, or function call, can be used to create a new time series as follows:

```
x^2+y %time
```

4.2.1 Calling an external function

The Derive Output UI can also be used as an interface to user functions that are available on the Matlab search path. Simply type the function call with the appropriate variable assignment and the new variable is created. (NB: the UI adopts the Matlab convention that all functions are lower case). Some examples of functions provided in MRBreach are detailed in Section 4.2.3.

The input variables for the function must match the syntax used for the call from the Derive Output UI, as explained above. In addition, functions can return a single value, one or more vectors or arrays, or a `dstable` (see Section 4.2.2). If the variables have a dimension (e.g., *time*) then this should be the first variable, with other variables following. If there is a need to handle additional dimensions then use the option to return a `dstable`.

If there is no output to be passed back, the function should return a variable containing the string 'no output' to suppress the message box, which is used for single value outputs (numerical or text).

An alternative when calling external functions is to pass the selected variables as `dstable`s, thereby also passing all the associated metadata and `RowNames` for each dataset selected. For this option up to 3 variables can be selected and assigned to the X, Y, Z buttons but they are defined in the call using `dst`, for example:

```
[time,varout] = myfunction(dst, 'usertext', mobj);
```

```
dst = myfunction(dst, 'usertext', mobj);
```

where *'usertext'* and *mobj* are call strings and a handle to the model, respectively.

This passes the selected variables as a struct array of dstables to the function. Using this syntax, the function can return a dstable or struct of dstables, or as variables, containing one or more data sets.

4.2.2 Input and output format for external functions

There are several possible use cases:

4.2.2.1 Null return

When using a function that generates a table, plots a figure, or some other stand alone operation, where the function does not return data to the main UI, the function should have a single output variable. The output variable can be assigned a text string, or 'no output', if no user message is required, e.g.:

```
function res = phaseplot(x,y,t,labels)
...
    res = {'Plot completed'}; %or res = {'no output'}; for silent mode
...
end
```

4.2.2.2 Single value output

For a function that may in some instances return a single value this should be the first variable being returned and can be numeric or text, e.g.:

```
function [qtime,qdrift] = littoraldriftstats(qs,tdt,varargin)
...
    %Case 1 - return time and drift
    qdtime = array1;
    qdrift = array2;
    %Case 2 - return summary value
    qtime = mean(array2); %return single value
    %Case 3 - return summary text
    qtime = sprintf('Mean drift = %.1f',mean(array2)); %return test string
...
end
```

4.2.2.3 Using variables

If only one variable is returned (length>1), or the first variable is empty and is followed by one or more variables, the user is prompted add the variables to:

- i) Input Cases – one of the datasets used in the function call;
- ii) New Case – use output to define a new dataset;
- iii) Existing Case – add the output to an existing dataset (data sets for the selected existing case and the data being added must have the same number of rows.

In each case the user is prompted to define the properties for each of the variables.

Note that variable names and descriptions must be unique within any one dataset.

```
function y = moving(x,m,fun)
    %a single variable is returned with no rows
    y is a vector or array
```

```
...
end

or

function [x,y,z] = afunction(x,m,fun)
    %multiple variables returned but the first variable is empty
    x = [];
    y and z are a vectors or arrays
    ...
end
```

When the first variable defines the rows of a table and subsequent variables the table entries, all variables must be the same length for the first dimension. This is treated as a new Case and the user is prompted to define the properties for each of the variables.

```
function [trange,range,hwl,lwl] = tidalrange(wl,t,issave,isplot)
    %first variable is row dimension followed by additional variables
    trange,range,hwl,lwl are vectors or arrays
    ...
end
```

4.2.2.4 Using dstables

When the output has multiple variables of a defined type it can be more convenient to define the dsproperties within the function and return the data in a dstable. This avoids the need for the user to manually input the meta-data properties. In addition, if the function generates multiple dstables, these can be returned as a struct, where the struct fieldnames define the Dataset name.

```
function dst = tidalrange(wl,t,issave,isplot)
    %dst is a dstable with variables, dimensions and dsproprties assigned
    %as required, or a struct of dstables with the struct fieldnames defining
    %each Dataset.
    dst = ...
    ...
end
```

Similarly if the input is also using dstables, the syntax is as follows:

```
function dst_out = myfunction3(dst_in,'usertext',mobj)
    %dst_in is one or more input dstables, 'usertext' is some additional
    %instruction to the function and mobj is a handle to the model
    %allowing access to other datasets. dst_out is either a dstable, or a
    %struct of dstables with the struct fieldnames defining each Dataset.
    dst = ...
    ...
end
```

Adding functions to the Function library

To simplify accessing and using a range of functions that are commonly used in an application, the function syntax can be predefined in the file functionlibrarylist.m which can be found in the utils folder of the mutoolbox. This defines a struct for library entries that contain:

- fname - cell array of function call syntax;

- fvars - cell array describing the input variables for each function;
- fdesc - cell array with a short description of each function.

New functions can be added by simply editing the struct in functionlibrarylist.m, noting that the cell array of each field in the struct must contain an entry for the function being added. In addition, a sub-selection of the list can be associated with a given App based on the class name of the main UI. To amend the selection included with an App or to add a selection for a new App edit the ‘switch classname’ statement towards the end of the function.

The Function button on the Derive Output UI is used to access the list, select a function and add the syntax to the function input box, where it can be edited to suit the variable assignment to the XYZ buttons.

4.2.3 Pre-defined functions

The following examples are provided within MRBreach, where the entry in the UI text box is given in Courier font and X, Y, Z, refer to the button assignments.

Some useful examples include:

1. **Recursive plot.** Generates a plot of a variable plotted against itself with an offset (e.g. $x(i)$ versus $x(i+1)$). This is called from the Derive Output GUI using:

`recursive_plot(x, 'varname', nint)`, where x is the variable, ‘varname’ is a text string in single quotes and $nint$ is an integer value that defines the size of the offset.

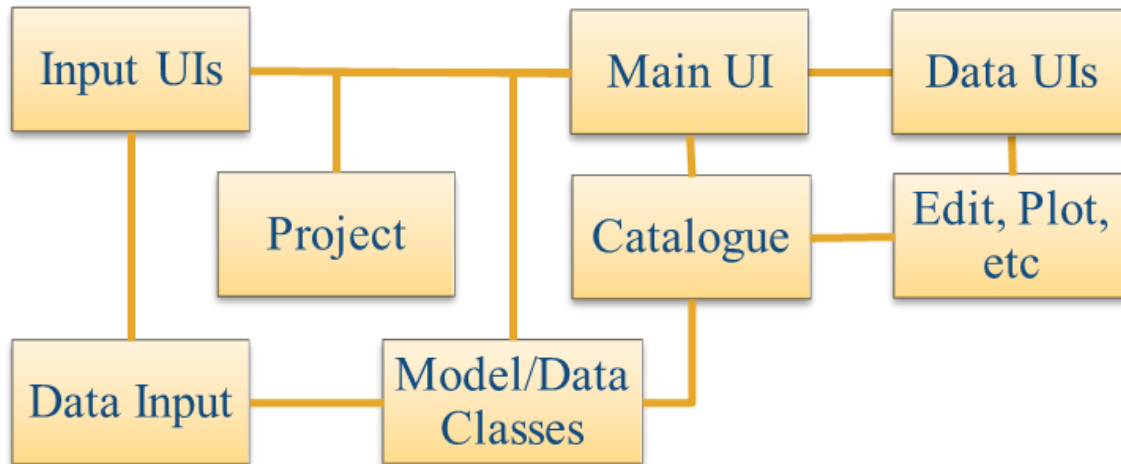
2. **Phase plot.** This function is similar to the recursive plot function but generates a plot based on two variables that can, optionally, be functions of time. The call to this function is:

`phaseplot(X, Y, t)`, where X and Y are the variables assigned to the respective buttons and t is time (this does not need to be assigned to a button and t can be omitted if a time stamp for the datapoints is not required).

5 Program Structure

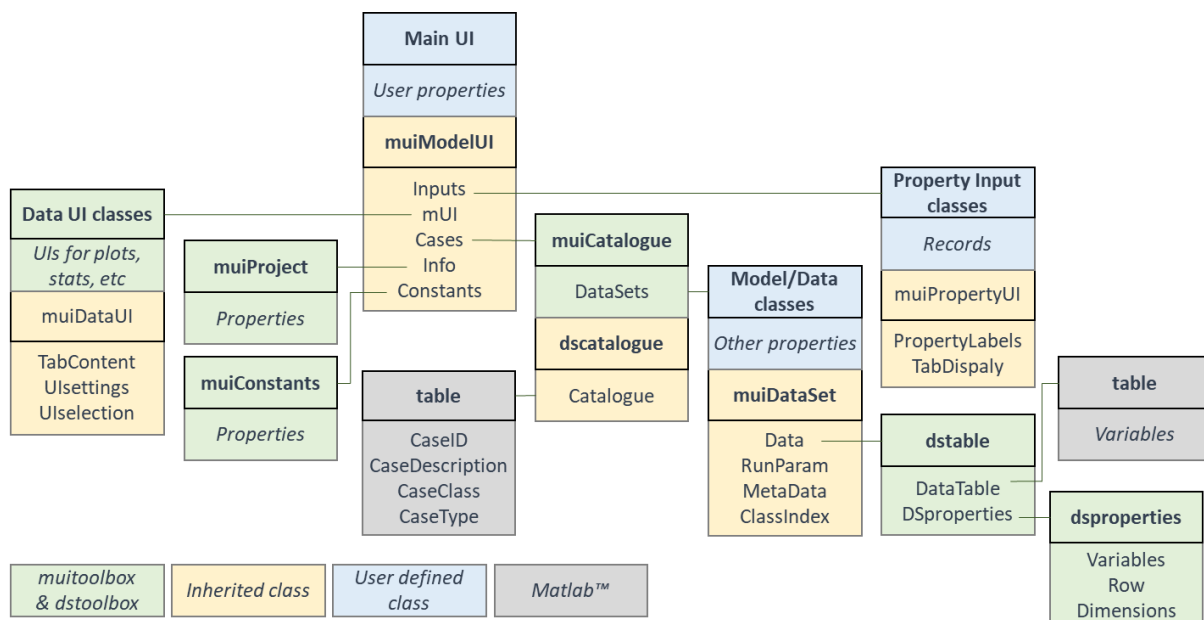
The overall structure of the code is illustrated schematically in Figure 1. This is implemented through several classes that handle the graphical user interface and program workflows (Main UI) and several classes that handle the data manipulation and plotting (Input UIs and Data UIs).

Figure 1 -- High level schematic of program structure



The interfaces and default functionality are implemented in the MRBreach App using the following mui toolbox classes depicted in Figure 2, which shows a more detailed schematic of the program structure. See the mui toolbox and dstoolbox documentation for more details.

Figure 2 – schematic of program structure showing how the main classes from mui toolbox and dstoolbox are used



In addition, the MRBreach App uses the following classes and functions:

MRBreach – modifies the UI to redirect menu dropdown options and provide additional tab functions.

mrBreachModel – provides the methods to calculate and plot the regime breach section

mrBreachData – handles the input of parameters for tide and wave data required by the model.

mrSiteData – handles the input of parameters that define the site (tidal prism, levels, etc)



mrHypsometry – calculates the empirical hypsometry and provides the methods to allow the user to interactively adjust the fit coefficients. Also provides methods to load observed hypsometry from a text file.

The following coastal tool functions are used in **mrBreachModel**:

ucrit.m, tma_spectrum.m and celerity.m.

Further details of how the model is implemented within the ModelUI framework are given in the ModelUI manual.

6 Bibliography

- Scott, C.R., Armstrong, S., Townend, I.H., Dixon, M., Everard, M., 2011. Lessons learned from 20 years of managed realignment and regulated tidal exchange in the UK. ICE Coastal Management 2011. ICE, pp. 1-10.
- Taylor, K.E., 2001. Summarizing multiple aspects of model performance in a single diagram. Journal of Geophysical Research - Atmospheres, 106(D7), 7183-7192. [10.1029/2000JD900719]
- Townend, I.H., 2008a. Breach design for managed realignment sites. Proc.Instn Civ.Engrs., Maritime Engineering, 161(MA1), 9-21.
- Townend, I.H., 2008b. Hypsometry of estuaries, creeks and breached sea wall sites. Proc.Instn Civ.Engrs., Maritime Engineering, 161(MA1), 23-32.
- Townend, I.H., Scott, C.R., Dixon, M., 2010. Managed Realignment: A Coastal Flood Management Strategy. In: G. Pender, C.R. Thorne, I. Cluckie, H. Faulkner (Eds.), Flood Risk Science and Management. Blackwell Publishing Ltd, Oxford, pp. 60-86.



Appendix A – Hypsometry file format

Hypsometry loads text files with the following format

```
Start hypsometry
Elevation    Cumulative area
0.03        40
0.06        81
0.09        133
0.12        189
0.15        232
0.18        275
0.21        314
0.24        356
0.27        401
```

Elevation is relative to Ordnance Datum (mAOD) and area is in m².