



Preface

TableViewer is a MatlabTM App that enables tables of data to be loaded from text files, spreadsheets, or .mat files containing tables. Some standard tools to plot and analyse the data are included and the two function provide the ability for the user to add their own plotting and analysis functions.

Requirements

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

Resources

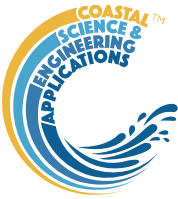
The TableViewer App and two toolboxes (muitoolbox and dstoolbox) can be downloaded from <https://github.com/CoastalSEA>.

Cite as:

Townend, I.H., 2024, TableViewer manual, CoastalSEA, UK, pp30, www.coastalsea.uk.

Revision history

Version	Date	Changes
1.0	Oct 2024	First release via https://github.com/CoastalSEA



Contents

1	Introduction	1
2	Getting started	2
2.1	Configuration.....	2
2.1.1	Installing the toolboxes	2
2.1.2	Installing the App	2
2.2	Table Set-up	2
3	Application Menus	3
3.1	File.....	3
3.2	Tools.....	3
3.3	Project.....	3
3.4	Setup.....	4
3.5	Run	6
3.6	Analysis	7
3.6.1	Plotting	7
3.6.2	Statistics.....	8
3.6.3	User Plots	13
3.7	Help	13
3.8	Tabs	13
3.9	UI Data Selection	13
3.10	Accessing data from the Command Window	15
4	Implementation/Supporting Information/Demonstration models	17
4.1	Derive Output	17
4.1.1	Calling an external function	18
4.1.2	Input and output format for external functions.....	18
4.1.3	Pre-defined functions	21
5	Program Structure.....	23
6	Bibliography.....	25
	Appendix A - Import file formats.....	26
	Appendix B - Data set properties (DSproperties).....	27
	Appendix C – Sample DSProperties for data import	28



1 Introduction

TableViewer App is a MatlabTM App to load tabular data from text files, spreadsheets and table or dstable data held in a .mat file. This allows the rapid viewing of a range of default plots, interactive plotting and statistics tools and the ability to add bespoke analysis and plotting. The latter has the advantage that the functions and outputs can be more clearly documented than is often the case in a spreadsheet, thereby maintaining the history (especially if the functions are maintained in a version control repository, such as git or svn). If the latter is the intended use, you can clone the TableViewer from github (<https://github.com/CoastalSEA>) if you want to build and maintain your own version of the App. Alternatively, you can create a branch and contribute your additions to the development.

2 Getting started

2.1 Configuration

TableViewer 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: `TableViewer.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.

2.1.2 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:

```
>> TableViewer;
```

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.

2.2 Table Set-up

File>New to create a new project space.

Setup>Input Data>Load

Load some data from an array in a text file, a worksheet of a spreadsheet, or an existing table or dstable from a .mat file (see Section 3.4 and Appendix A - Import file formats). The user is prompted for a file and then for a description of the data source (text string) and a name for the dataset (must be a valid Matlab™ variable name or code will try to convert the text entered to a valid name).

When loaded into a dstable the data are assigned dsproperties (i.e., name, description, unit, label and qcflag or format – see Appendix B - Data set properties (DSproperties)). By default, these are assigned the variable name and qcflag is set to none. The user is then prompted to load a file of definitions (.m file, or text file), use a UI to input the definitions, or skip this step and use the defaults. For details of how to set the DSproperties and import file formats see Section 3.4 and Appendix C – Sample DSProperties for data import).

Setup>Import data > Add data: prompts for file to be added (only one file at a time can be added) and the Case to use (if more than one Case).

Individual tables can be views on the *Table* tab and simple default plots are accessed using the *Q-Plot* tab.

Analysis>Plots

The imported data can be selected and plotted. By using the Add button additional model runs can be included on the plot, allowing different Cases 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 cases, 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 case description to edit.

Project>Cases>Edit DS properties: edit the properties that define the meta-data for a dataset.

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>Import Data: dialog with sub-menu options to Load, Add, and Delete.

Select a file to load. Once added the current set of variables can be viewed using the *Inputs* tab. When the data has been loaded, the user is prompted to provide a description of the data set (a case) and is listed on the *Cases* tab.

Setup>Import data> Load data: the user is prompted for a file. Accepted formats are:

1. A table or dstable created in Matlab and saved as a mat file
e.g., `>> save('my_table.mat', mytable')`, where the table was assigned to the variable ‘mytable’;
2. As option 1, containing a struct of tables or dstables, where the fieldnames of the struct define the dataset name assigned to each table.
3. The first worksheet in an Excel spreadsheet. This uses the `readtable` function in `DSDataset.loadDSdata` with the option to `ReadRowNames` set to true. This assumes that the first column of the spreadsheet contains unique names for the data. The `readtable` function allows other options including reading variable names and handling date-time input. The variable names must be valid variable names (a valid variable name starts with a letter, followed by letters, digits, or underscores).
4. An ASCII text file using the MatlabTM `readtable` function. The header contains the variable names with tab or comma separation (a valid variable name starts with a letter, followed by letters, digits, or underscores). The first column contains the row name (if used) and subsequent columns contain the data with the same separator as the header.

Examples of the spreadsheet and text file formats are provided in Appendix A - Import file formats.

The user is then prompted for a description of the data source (text string) and a name for the dataset (must be a valid MatlabTM variable name or code will try to convert the text entered to a valid name).

When data are loaded into a dstable the data are assigned dsproperties (i.e., name, description, unit, label and qcflag or format – see Appendix B - Data set properties (DSproperties)). By default, these are assigned the variable name and qcflag is set to none.

The user is then prompted to select options for File, UI and Skip:

File - loads a file of definitions (.m or text file), or

UI - uses a UI to input the definitions,

Skip - skips this step and use the defaults.

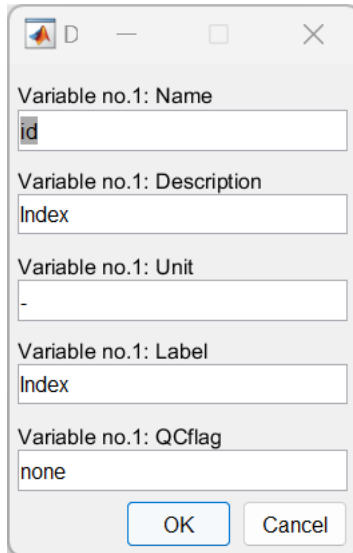
[NB: when loading a struct of tables from a .mat file it is assumed that these already have DSproperties and the UI is used to step through the definitions to provide the opportunity to edit, or modify the definitions.]

Load DSproperties from file:

A MatlabTM function file that defines a variable that holds a struct of the DSproperties, as explained in Appendix B - Data set properties (DSproperties) and illustrated in Appendix C – Sample DSProperties for data import.

A text file lists the definitions of the variables only as illustrated in Appendix C – Sample DSProperties for data import. These are then scrolled through using the UI (see below) and allows the definition of the rownames to be edited to suit the dataset.

Load DSproperties using UI:



Data in the imported table is used, where possible, to define the meta-data for each variable. The user is then prompted to either accept the definition to be assigned for each variable in turn. As well as the variable name, this includes a description, units, the label to use when plotting this and similar variables, and the quality control flag for variables, or format for the rowname. The format property defines the date format when used using the Matlab identifiers (e.g., 'dd-MM-yyy HH:mm:ss' for 26-Oct_2024 13:22:00 date style).

NB : the UI prompts for Dimensions with a value of 0. For scalar table data, only rows are used to hold a unique identifier (i.e., a dimension). Accept the default value to continue.

When the data has been loaded, the user is prompted to provide a description of the data set (case) and this is listed on the *Cases* tab. All input data sets can be viewed (just one variable at a time) on the *Q-Plot* tab.

Individual tables can be viewed on the *Table* tab and default plots are accessed using the *Q-Plot* tab.

Setup>Import Data>Add: There are three options for adding data:

Setup> Import Data>Add>Rows: add rows of data to an existing dataset. Data can be loaded from a table or spreadsheet but the number of variables should be the same as the existing dataset.

Setup> Import Data>Add>Variables: add variables of data to an existing dataset. Data can be loaded from a table or spreadsheet but the number of rows should be the same as the existing dataset.

Setup> Import Data>Add>Dataset: this links another dataset to a selected Case record. The dataset is loaded using the same process as used to *Load data* (see above).

The user is prompted to choose the existing data set that the data is to be added to.

The next prompt is to define any likely non-standard missing value indicators. The incoming table/spreadsheet is then concatenated to the existing table. The data added can be as additional rows, in which case the first column of the incoming table is used to define row names and these must be unique for the combined table and the number of variables must match the existing table. Or the incoming table/spreadsheet can contain additional variables, in which case the incoming table must have the same number of rows as the existing table. When variables are added, the user is prompted to accept or update the meta-data for each new variable.

Setup>Import data > Add data: prompts for file to be added (only one file at a time can be added) and the Case to use (if more than one Case)..

Setup>Import data > Delete: There are three options for adding data:

Setup> Import Data> Delete >Rows: select rows to be deleted.

Setup> Import Data> Delete >Variables: select variables to be deleted.

Setup> Import Data> Delete >Dataset: select dataset to be deleted. If there is only one dataset linked to the case you will be prompted to use *Project>Cases>Delete*.

Setup>Input Data>Input Parameters: This provides the option to enter additional parameters that may be needed for a particular analysis. The class provided is a template and simply requires the parameter descriptions and property names to be edited in the class file (TVparameters.m). The input parameters can be viewed on the *Inputs* tab.

Accessing the Input Parameters:

The data are saved to mobj.Inputs.TVparameters, where mobj is an instance of the TableViewer class. The parameters are set using: setClassObj(mobj, 'Inputs', 'TVparameters', obj), where obj is an instance of TVparameters; and can be accessed using getClassObj(mobj, 'Inputs', 'TVparameters'), or directly using mobj.Inputs.TVparameters. Both return the saved instance of the TVparameters object. Alternatively, using getProperties for a cell array, getPropertiesStruct to return a struct and getPropertiesTable to return a table; e.g., getProperties(mobj.Inputs.TVparameters).

Setup>Input Data>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.

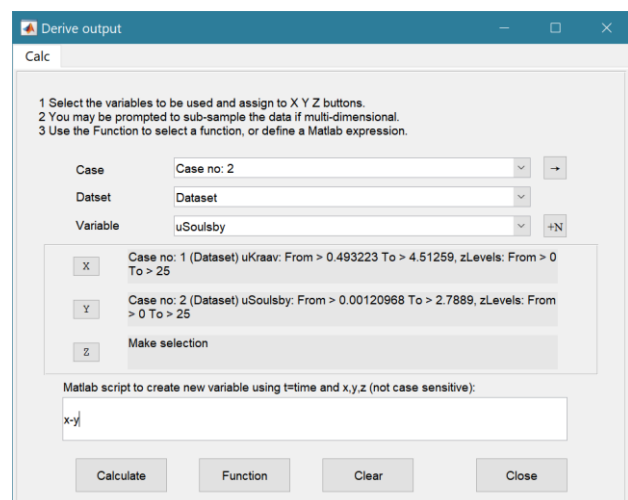
3.5 Run

Run> User Tools: calls function 'tableviewer_user_tools.m', which includes a function to create a figure tabulating a dataset and the option for the user to add functions as required.

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 3.9 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.



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.

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, 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 (numbere 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);
```

```
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.

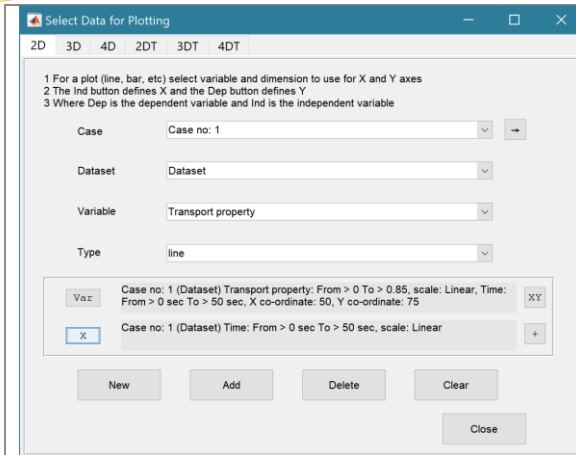
Some further details on using this option and the '**Function**' library available are provided in Section 4.1.

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 plots. The following figures illustrate the options available.



2D plot

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

> 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)

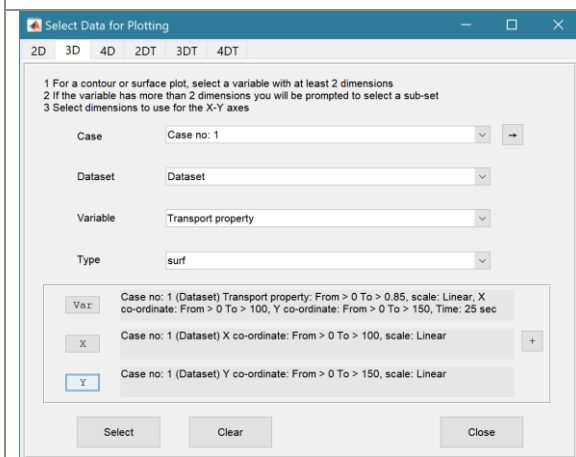
Control Buttons:

→ : updates the list of Cases

XY : swaps the X and Y axes

+ : switches between cartesian and polar plot type

If polar selected then Ind assumed to be in degrees.



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 muitoolbox. 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.

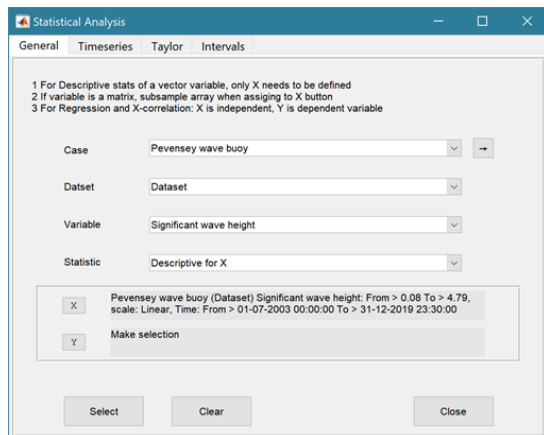
3.6.2 Statistics

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**.

General tab

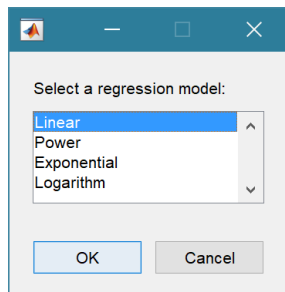
The General tab allows the user to apply the following statistics to data loaded in ModelUI:

- 1) **Descriptive for X:** general statistics of a variable (mean, standard deviation, minimum, maximum, sum and linear regression fit parameters). Only X



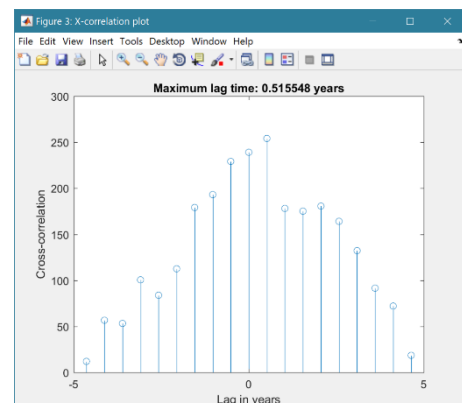
needs to be defined. The range of the variable can be adjusted when it is assigned to the X button (see Section 3.9). If the variable being used is a multi-dimensional matrix (>2D), the user is prompted to define the range or each additional dimension, or select a value at which to sample. The function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

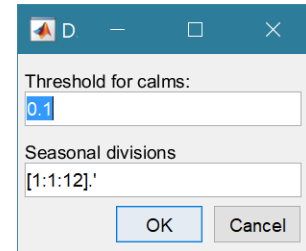
Timeseries tab

The Timeseries tab allows the user to select a single Timeseries variable and apply any of the following statistics:

- 1) **Descriptive:** general statistics of a variable (mean, standard deviation, minimum, maximum, sum and linear regression fit parameters). The results are tabulated in a new window and can be copied to the clipboard for use in other applications.

Various ‘seasonal’ sub-divisions can be defined. The required option is selected from the table in the UI, by selecting a Syntax cell and then closing the UI.

The next UI prompts for a threshold for calms (values below threshold are deemed to be “calm” conditions) and allows the selected ‘seasonal’ divisions to be changed (if the desired option is not in the default list), or edited. The divisions can be expressed in several ways, as detailed below:



Script	Result
1	Descriptive statistics for the full-time series
[1:1:12].'	Descriptive statistics for the full-time series and monthly values (the .' creates a column vector).
[12,1,2; 3,4,5; 6,7,8; 9,10,11]	Descriptive statistics for the full-time series and seasons based on groupings – Dec-Feb, Mar-May, Jun-Aug, Sep-Nov shown.

When seasonal statistics are produced with more than 2 seasons a plot is generated. This can be a cartesian or polar plot of the mean values with error bars used to depict +/- one standard deviation. The polar plot maps the year as one revolution.

- 2) **Peaks:** generates a new timeseries of peaks over a defined threshold. There are three methods that can be selected:

- 1 = all peaks above the threshold;
- 2 = the peak value within each up-down crossing of threshold; and
- 3 = peaks that have a separation of at least ‘*tint*’ hours.

For option 3, the separation between peaks (‘*tint*’) is also be defined in the pop-up gui. This can be used to try and ensure that peaks are independent. The peaks are marked on a plot with the defined threshold. If rejected, new values can be defined. If accepted a new timeseries is added. This has the class of the Data Type that was used as the source timeseries but is not appended to that timeseries because the date/times are a subset of the source.

- 3) **Clusters:** The selection process is similar to peaks, where the user defines a threshold, selection method and time between peaks (for method 3). In addition, the cluster interval is defined in days. This is the period of time separating two peaks for them to be no longer considered part of a cluster (e.g. if a sequence of storms occurs every few days they will form a cluster. If there is then a gap of, say, 31 days to the next storm, with a cluster time interval of 30 days this would be considered as part of the next cluster). Once a selection has been made, a plot is generated that shows the peaks for each cluster with a different symbol. The user can either choose a different definition, or accept the definition. Once accepted, the results are added as a new timeseries, with the class of the Data Type that was used as the source timeseries. Two values are stored at the time of each peak in the clusters: the magnitude of the peak; and the number of the cluster to which it belongs (numbered sequentially from the start). This allows the data for individual clusters to be retrieved, if required.
- 4) **Extremes:** The selection process is similar to peaks, where the user defines a threshold, selection method and time between peaks (for method 3). A figure is generated with two plots. The left-hand plot shows the peaks for the defined threshold and the right hand plots shows the mean excess above the threshold (circles), the 95% confidence interval (dotted red lines) and the number of peaks (vertical bars + right hand axis) as a function of threshold. This plot can be used to help identify a suitable threshold for the peak-over-threshold extremes analysis method. The user can either choose a different definition, or accept the definition. Once

accepted, the user is prompted to select a plot type. Options are: None; Type 1 – a single return period plot; Type 2 – a composite plot showing the probability, quantile, return period and density plots. See Coles (2001) for further details of the method used and the background to these plots. The results are tabulated on the *Stats/Extremes* tab and can be copied to the clipboard for use in other applications.

- 5) **Poisson Stats:** user is prompted to select a threshold, method and peak separation (see Peaks above) and the function generates a plot of the peak magnitude, time between peaks (interarrival time) and the duration above the threshold for each peak. The plot shows a histogram of each variable and the exponential pdf derived from the data, along with the μ value for the fit.
- 6) **Hurst Exponent:** user is prompted to select from one of 3 methods, which are based on different computation routines taken from the Matlab Forum, as follows:
 - 1 = Chiarello matrix method,
 - 2 = Abramov loop code,
 - 3 = Aalok-Ihlen code and
 - 4 = Aste using unweighted option.

Methods 1 and 2 are similar, whereas method 3 explores the effect of scale and method 4 derives the unweighted generalized Hurst exponent. The main difference between the first two methods is that Abramov uses a different form for S , rather than the Matlab standard deviation function (std).

The Hurst parameter H is a measure of the extent of long-range dependence in a time series (while it has another meaning in the context of self-similar processes). H takes on values from 0 to 1. A value of 0.5 indicates the absence of long-range dependence. The closer H is to 1, the greater the degree of persistence or long-range dependence. H less than 0.5 corresponds to a lack of persistence, which as the opposite of LRD indicates strong negative correlation so that the process fluctuates violently. H is also directly related to fractal dimension, D , where $1 < D < 2$, such that $D = 2 - H$.

This is experimental code (for code see `.../muitoolbox/psfunctions/hurst_exponent.m`, `hurst_aalok_ihlen.m` and `genhurstw.m`) and the user should refer to the background literature for further details. (Di Matteo *et al.*, 2003; Pacheco *et al.*, 2008; Ihlen, 2012; Morales *et al.*, 2012; Sutcliffe *et al.*, 2016; Abramov and Khan, 2017; Antoniadis *et al.*, 2021; Brandi and Di Matteo, 2021).

- 7) **User:** calls `user_stats.m` function, where the user can define a workflow, accessing data and functions already provided by the particular App, or the `muitoolbox`. The sample code can be found in the `psfunctions` folder and illustrates the workflow to produce a clusters plot. Some code in the header (commented out) shows how to get a time series using the handles passed to the function (`obj` and `mobj`). This code would get the same timeseries as the one passed to the function. However, by modifying the 'options' variable it is possible to access other timeseries variables.

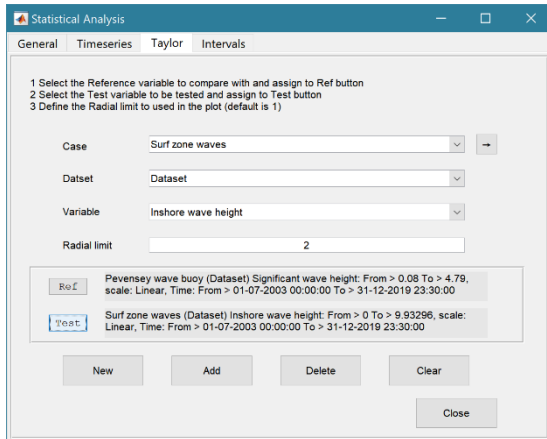
Taylor tab

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).

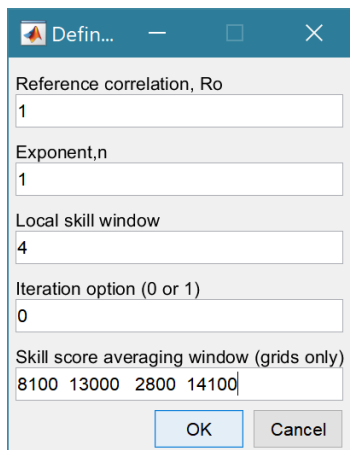
[The ModelSkill App provides additional tools to test data and the ModelSkill App manual provides further details of the methods used.]

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).

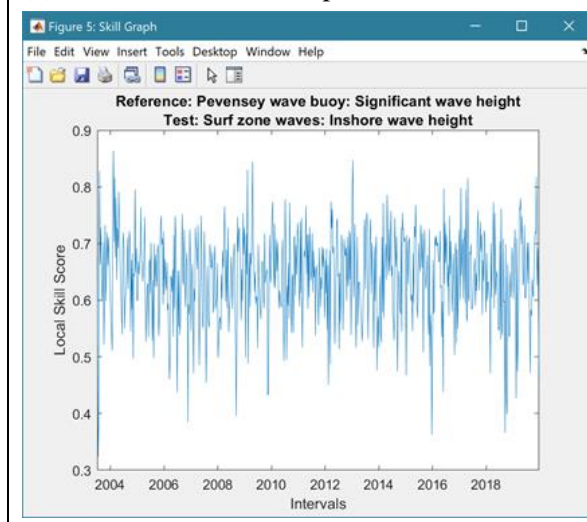
Exponent used in computing the skill score (see ModelSkill manual for details).

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.

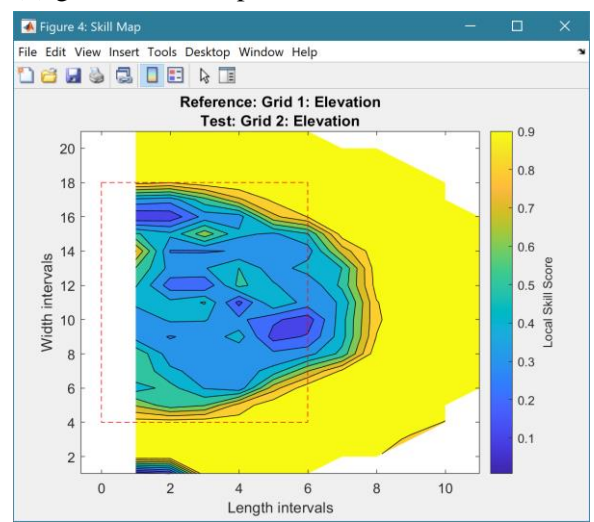
Local skill score is computed for window around every grid cell (=1), or computes score for all non-overlapping windows (=0)

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

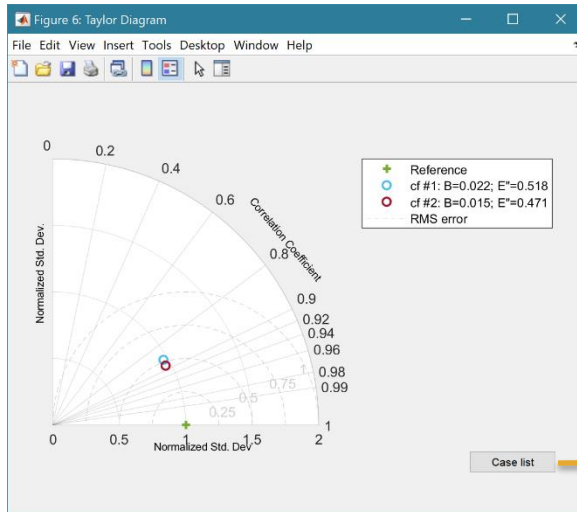
(a) time series skill score plot



(b) grid skill score plot

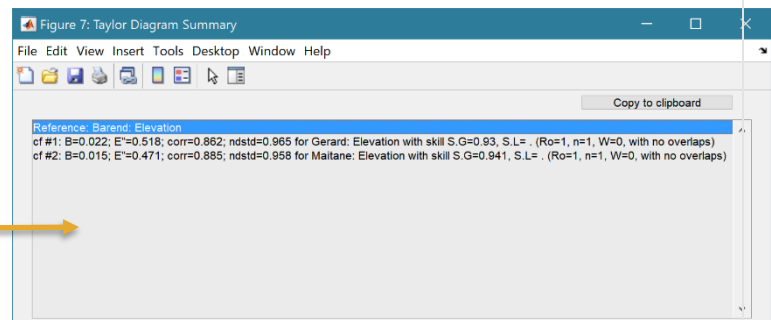


The Taylor Plot shows the Reference point as a green cross and the Test points as coloured circles. The legend details the summary statistics and the Case List button generate a table figure listing all the results. These can be copied to the clipboard.



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.



3.6.3 User Plots

Analysis> User Plots: calls function ‘tableviewer_user_plots.m’, which includes functions to plot a scatter diagram of two or three variables from any case (selected variables must be the same length) and a bar chart of a variable with the bars coloured based on a selected classification variable (from the same dataset as the main variable).

3.7 Help

The help menu provides options to access the App documentation in the MatlabTM 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, source information (files input or models used) and any user data (e.g., tables of derived parameters) that is saved with the data set.

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

Table: tabulates a selected dataset (display only).

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:

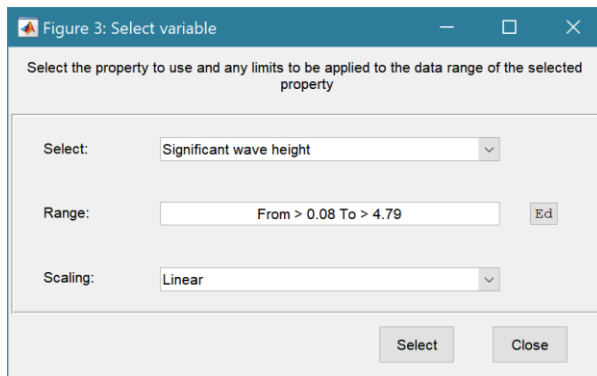


Figure 3: Select variable

Select the property to use and any limits to be applied to the data range of the selected property

Select: Significant wave height

Range: From > 0.08 To > 4.79 Ed

Scaling: Linear

Select Close

+ scaling options include Linear; Log; Relative ($V - V(x=0)$); Scaled ($V/V(x=0)$); Normalised; Normalised ($=ve$); Differences; Rolling mean.

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.

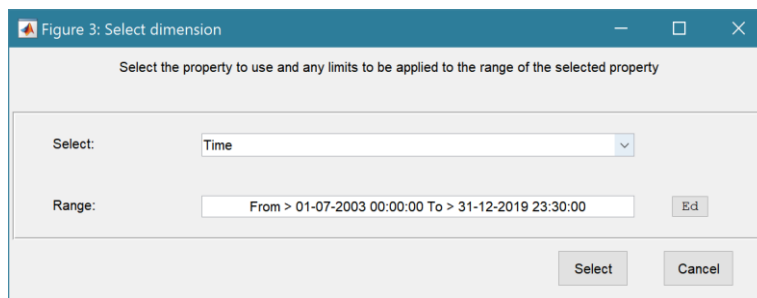


Figure 3: Select dimension

Select the property to use and any limits to be applied to the range of the selected property

Select: Time

Range: From > 01-07-2003 00:00:00 To > 31-12-2019 23:30:00 Ed

Select Cancel

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.

Step 2 - Variable only has dimension of time.

No selection to be made.

Edit range if required.

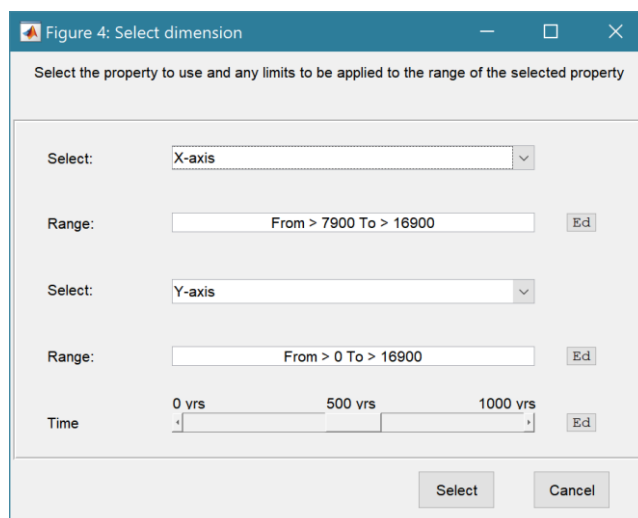


Figure 4: Select dimension

Select the property to use and any limits to be applied to the range of the selected property

Select: X-axis

Range: From > 7900 To > 16900 Ed

Select: Y-axis

Range: From > 0 To > 16900 Ed

Time 0 yrs 500 yrs 1000 yrs

Select Cancel

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.

myapp = <AppName> with properties:	Purpose
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:

```
>> myapp.Cases.Catalogue
```

To access imported or model data sets, use:

```
>> myapp.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:

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



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

To access the underlying *table*, use:

```
>> myapp.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 Implementation/Supporting Information/Demonstration models

4.1 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 4.1.2).

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 4.1.2).

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 $\text{diff}(x)$ - 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
```

4.1.1 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 TableViewer are detailed in Section 4.1.3.

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 4.1.2). 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 dstables, 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.

4.1.2 Input and output format for external functions

There are several possible use cases:

4.1.2.1 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
```

4.1.2.2 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
qdrift = array1;
qtime = 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
```

4.1.2.3 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
```

4.1.2.4 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
```

4.1.2.5 Saving additional model parameters

When saving function results as dstable, it is also possible to save additional parameters as part of the table. The following example puts a table of summary statistics in the dstable UserData property.

```
function dst = tidalrange(wl,t,issave,isplot)
    ....
    dsp = setDSproperties();
    results = {R,hwl,lwl};
    dst = dstable(results{:},'RowNames',rt,'DSproperties',dsp);
    %Put fit parameters in UserData
    dst.UserData = summary_stats_table;

    ...

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 muitoolbox. 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.

4.1.3 Pre-defined functions

The following examples are provided within TableViewer, 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 primarily for timeseries data include: :

1. **Moving Average.** There are several moving average functions available from the Matlab Exchange Forum, such as moving.m. The call to this function is:

`moving(X, n, 'func')` , where x is the variable to be used, n specifies the number of points to average over and ‘func’ is the statistical function to use (e.g. mean, std, etc). If omitted the *mean* is used. Add %time to the call, to include time in the output dataset.

2. **Moving average (or similar) of timeseries**, over defined duration, advancing at defined interval

`movingtime(x, t, tdur, tstep, 'func')`, where x is the variable to be used and t the associated datetimes (defined by variable selection), *tdur* is the duration over which to apply the statistic, *tstep* is the interval to advance the start time for the averaging period and ‘func’ is the statistical function to use (e.g. mean, std, etc). If omitted the *mean* is used. *tdur* and *tstep* are both duration character strings of form ‘2.5 d’. Any of the following duration intervals can be used: y, d, h, m, or s. Returns a time series based on the defined *tstep*, where the time used is for the beginning of each stepping interval, i.e. every *tstep* from the start of the record to the nearest interval that is less than *tdur* from the end of the record.

3. **Down-sampling a time series.** This allows a timeseries to be resampled at a different interval (that must be less than the source timeseries). The call to this function is:

`downsample(x, t, 'period', 'method')`, where x is the variable to be resampled, time is the associated time for that variable, period can be ‘year’, ‘month’, ‘day’, ‘hour’, ‘minute’, ‘second’, and method can be any valid function call such as ‘mean’, ‘std’, etc. The ‘period’ is required but the ‘method’ is optional and if omitted the mean is used.

For timeseries with gaps the ‘nanmean’ function is particularly useful but requires the Statistics toolbox.

4. **Interpolate and add noise.** To infill a record with additional points and, if required, add some random noise to the interpolated values. This is called using:

`interpwithnoise(x, t, npad, scale, method, ispos)` , where X is the variable, t is time, npad is the number of points to add between the existing data points, scale determines the magnitude of the random noise (a value of 0 results in an interpolated record with no noise), method is the Matlab algorithm used for the interpolation (the default is linear) and ispos is a true/false flag which sets negative values to zero if true.

5. **Subsample one record at the time intervals of another record** (e.g. subsample water levels to be at the same intervals as the wave data). Function is:

`subsample_ts(X, t, mobj)`, where X and t are the variable to be subsampled and *mobj* is the UI handle (must be *mobj*). The user is prompted to select the dataset to be used to define the time intervals. A time series is returned and added as a Derived data set. The user is prompted to define the metadata for the new data set.

6. **Subsample one record based on a threshold defined for another record** (e.g. subsample waves based on a threshold water level). Function is:

`subsample(X, t, thr, mobj)`, where `X` and `t` are the variable to be subsampled, `thr` is the threshold value and `mobj` is the UI handle (must be `mobj`). The user is prompted to select the dataset and variable to be used to define the condition and a condition operator (`<=`, `==`, etc). A time series is returned and added as a Derived data set. The user is prompted to define the metadata for the new data set.

7. **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).

8. **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.

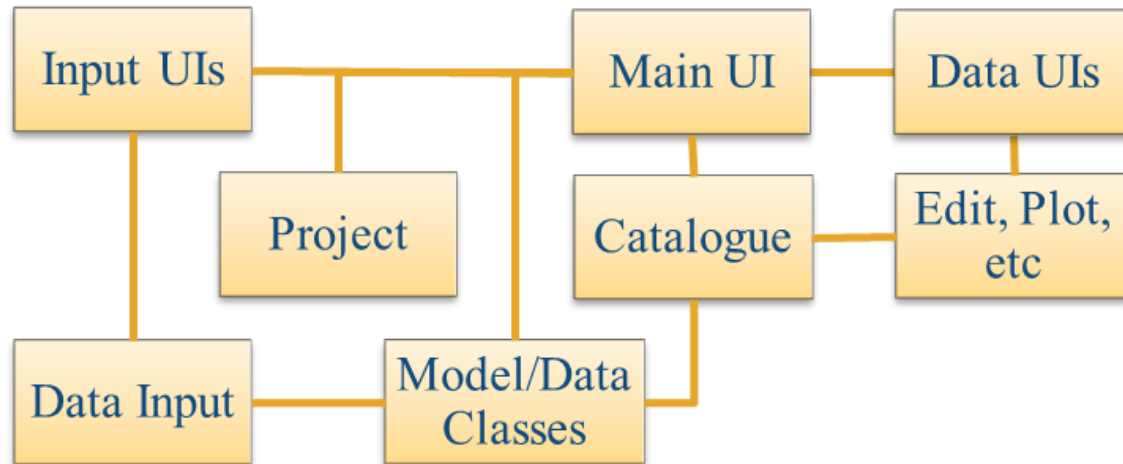
9. **Selection of frequency analysis plots of timeseries data** using the function:

`frequencyanalysis(X, t, 'vardesc')` where `X` is the variable, `t` is time and `vardesc` is the description of the variable to be used in the plots (optional – defaults to 'Variable'). Plot options include Time series plot of variable, Time series plot of variable above threshold, Plot variable frequency, Plot variable frequency above threshold, Spectral analysis plot, Duration of threshold exceedance, Rolling mean duration above a threshold.

5 Program Structure

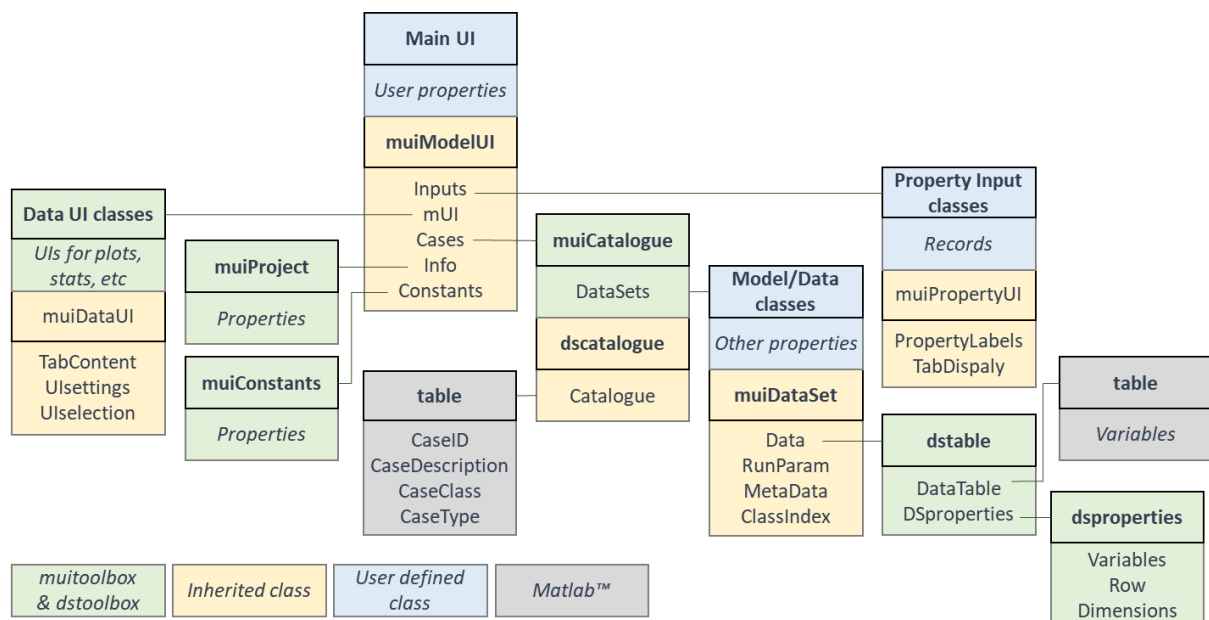
The overall structure of the code is illustrated schematically in Figure 1. This is implemented through several classes that handle the graphical user interface and program workflows (Main UI) and several classes that handle the data manipulation and plotting (Input UIs and Data UIs).

Figure 1 – High level schematic of program structure



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

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



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

TableViewer – class to define the behaviour of the main UI.

TVparameters – class to input parameters. This is a template for the user to define any inputs required for bespoke data analysis functions. The class file has to be edited to define the required inputs.



tableviewer_user_tools – function which includes a function to create a figure tabulating a dataset and the option for the user to add functions as required.

tableviewer_user_plots – function which includes functions to plot a scatter diagram of two variables from any case (selected variables must be the same length) and a bar chart of a variable with the bars coloured based on a selected classification variable (from the same dataset as the main variable)

6 Bibliography

Abramov V and Khan M K, 2017, A Practical Guide to Market Risk Model Validations (Part II - VaR Estimation). p. 70, <http://dx.doi.org/10.2139/ssrn.3080557>.

Antoniades I P, Brandi G, Magafas L and Di Matteo T, 2021, The use of scaling properties to detect relevant changes in financial time series: A new visual warning tool. *Physica A: Statistical Mechanics and its Applications*, 565, 10.1016/j.physa.2020.125561.

Brandi G and Di Matteo T, 2021, On the statistics of scaling exponents and the multiscaling value at risk. *The European Journal of Finance*, pp. 1-22, 10.1080/1351847x.2021.1908391.

Coles S, 2001, *An Introduction to Statistical Modeling of Extreme Values*, Springer-Verlag, London.

Di Matteo T, Aste T and Dacorogna M M, 2003, Scaling behaviors in differently developed markets. *Physica A: Statistical Mechanics and its Applications*, 324 (1-2), pp. 183-188, 10.1016/s0378-4371(02)01996-9.

Ihlen E A, 2012, Introduction to multifractal detrended fluctuation analysis in matlab. *Front Physiol*, 3, p. 141, 10.3389/fphys.2012.00141.

Morales R, Di Matteo T, Gramatica R and Aste T, 2012, Dynamical generalized Hurst exponent as a tool to monitor unstable periods in financial time series. *Physica A: Statistical Mechanics and its Applications*, 391 (11), pp. 3180-3189, <https://doi.org/10.1016/j.physa.2012.01.004>.

Pacheco J C R, Román D T and Vargas L E, 2008, R/S Statistic: Accuracy and Implementations, (ed)^(eds), 18th International Conference on Electronics, Communications and Computers (conielecomp 2008), IEEE, 10.1109/conielecomp.2008.14.

Sutcliffe J, Hurst S, Awadallah A G, Brown E and Hamed K, 2016, Harold Edwin Hurst: the Nile and Egypt, past and future. *Hydrological Sciences Journal*, 61 (9), pp. 1557-1570, 10.1080/02626667.2015.1019508.

Taylor K E, 2001, Summarizing multiple aspects of model performance in a single diagram. *Journal of Geophysical Research - Atmospheres*, 106 (D7), pp. 7183-7192, 10.1029/2000JD900719.



Appendix A - Import file formats

File formats for Load and Add tabular data

ASCII text file:

1	Name	id	JNCC	ERP2_Type	Type_id	Range	Vmhw	Vmtl	Vmlw	Smhw	Smtl	Smlw	Amhw	Amtl	Amhw
2	Thames Estuary	117	Coastal Plain	Funnel	5	6.5 9.67E+08	6.11E+08	4.17E+08	2.00E+08	8.19E+07	6.49E+07	5.81E+04	3.85E+04	2.31E+04	
3	Medway Estuary	119	Coastal Plain	Spit Enclosed	4	4.1 2.44E+08	1.36E+08	7.96E+07	7.56E+07	2.75E+07	2.17E+07	2.25E+04	3.84E+04	1.52E+04	
4	Pegwell Bay	121	Embayment	Spit Enclosed	4	4.5 1.61E+07	2.92E+06	6.03E+05	9.53E+06	2.50E+06	5.26E+05	2.35E+03	9.00E+02	2.04E+02	
5	Chichester Harbour	128	Bar Built Estuary	Tidal inlet	7	4.2 9.50E+07	5.14E+07	2.35E+07	3.41E+07	1.66E+07	1.21E+07	8.00E+03	5.45E+03	3.46E+03	
6	Langstone Harbour	129	Bar Built Estuary	Tidal inlet	7	4.2 6.19E+07	3.25E+07	1.36E+07	2.68E+07	1.10E+07	8.36E+06	4.70E+03	3.74E+03	2.93E+03	
7	Portsmouth Harbour	130	Bar Built Estuary	Tidal inlet	7	4.1 7.21E+07	4.79E+07	2.63E+07	1.95E+07	1.15E+07	9.07E+06	6.23E+03	4.79E+03	3.29E+03	
8	Southampton Water	131	Coastal Plain	Spit Enclosed	4	4 1.89E+08	1.33E+08	7.93E+07	6.28E+07	2.67E+07	2.00E+07	1.97E+04	1.55E+04	1.02E+04	
9	Beaulieu River	132	Bar Built Estuary	Spit Enclosed	4	3.2 1.38E+07	7.16E+06	3.47E+06	8.41E+06	2.60E+06	1.84E+06	6.43E+03	4.34E+03	1.90E+03	
10	Lymington Estuary	133	Coastal Plain	Spit Enclosed	4	2.5 2.45E+06	1.41E+06	6.00E+05	2.38E+06	7.44E+05	4.86E+05	2.88E+03	1.79E+03	6.99E+02	
11	Bembridge Harbour	134	Coastal Plain	Spit Enclosed	4	3.1 5.55E+06	1.87E+06	5.69E+05	2.19E+06	1.66E+06	3.83E+05	5.27E+03	3.26E+03	1.42E+03	
12	Wootton Creek & Ryde Sands	135	Coastal Plain	Spit Enclosed	4	3.8 6.27E+05	2.14E+05	5.09E+04	9.56E+04	1.12E+05	6.33E+04	1.01E+03	4.88E+02	1.17E+02	
13	Medina Estuary	136	Coastal Plain	Funnel	5	4.2 4.72E+06	2.94E+06	1.42E+06	2.43E+06	9.45E+05	7.29E+05	2.62E+03	1.81E+03	9.18E+02	
14	Newtown Estuary	137	Bar Built Estuary	Spit Enclosed	4	2.9 6.01E+06	4.32E+06	2.79E+06	1.57E+06	1.11E+06	8.01E+05	1.88E+03	1.14E+03	4.37E+02	
15	Yar Estuary	138	Coastal Plain	Spit Enclosed	4	2.5 5.35E+05	2.81E+05	9.53E+04	3.66E+05	1.78E+05	1.11E+05	1.41E+03	8.38E+02	2.47E+02	
16	Christchurch Harbour	139	Bar Built Estuary	Spit Enclosed	4	1.2 1.95E+06	1.04E+06	5.43E+05	4.62E+06	9.58E+05	7.90E+05	1.19E+02	4.17E+01	1.70E+01	
17	Poole Harbour	140	Bar Built Estuary	Spit Enclosed	4	1.4 5.65E+07	4.32E+07	2.20E+07	2.94E+07	2.44E+07	1.82E+07	3.27E+03	3.05E+03	2.64E+03	
18	The Fleet & Portland Harbour	141	Bar Built Estuary	Tidal inlet	7	1.9 3.85E+07	2.22E+07	1.66E+07	1.84E+07	1.06E+07	4.84E+06	3.32E+04	2.95E+04	2.66E+04	
19	Axe Estuary	142	Bar Built Estuary	Spit Enclosed	4	3.7 1.11E+05	9.53E+03	8.72E+01	1.79E+05	2.19E+04	5.21E+02	2.58E+02	7.70E+01	4.40E+00	
20	Otter Estuary	143	Bar Built Estuary	Spit Enclosed	4	4.1 7.47E+04	2.82E+04	5.56E+03	4.20E+04	2.24E+04	9.62E+03	1.78E+02	3.95E+01	2.00E+01	
21	Eve Estuary	144	Bar Built Estuary	Spit Enclosed	4	4.1 4.69E+07	2.25E+07	5.23E+06	1.81E+07	1.06E+07	7.46E+06	2.81E+03	1.67E+03	1.06E+03	
22	Teign Estuary	145	Ria	Spit Enclosed	4	4.2 5.13E+06	2.48E+06	4.69E+06	1.91E+06	1.03E+06	1.46E+06	5.34E+02	3.31E+02	4.52E+02	
23	Dart Estuary	146	Ria	Spit Enclosed	4	4.90E+07	3.47E+07	2.00E+07	1.02E+07	6.70E+06	5.74E+06	9.99E+03	8.24E+03	6.17E+03	
24	Salcombe & Kingsbridge Estuary	147	Ria	Ria	3	4.6 1.68E+07	6.85E+06	1.91E+06	1.20E+07	2.77E+06	1.45E+06	1.10E+04	8.21E+03	5.31E+03	

Excell spreadsheet:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	id	JNCC	ERP2_Type	Type_id	Range	V mhw	V mtl	V mlw	S mhw	S mtl	S mlw	A mhw	A mtl	A mlw
2	Hayle Estuary	1	Bar Built Estuary	Spit Enclosed	4	5	3.87E+06	2.73E+06	2.32E+06	1.49E+06	1.50E+05	1.44E+05	6.60E+00	0.00E+00	0.00E+00
3	Gannell Estuary	2	Ria	Ria	3	6.4	3.09E+06	1.59E+06	7.20E+05	1.19E+06	3.16E+05	2.55E+05	7.75E+03	5.62E+03	3.67E+03
4	Cannel Estuary	3	Ria	Ria	3	5.9	3.61E+07	1.87E+07	8.23E+06	7.77E+06	4.10E+06	1.81E+06	1.08E+04	6.34E+03	3.02E+03
5	Bridgwater Bay	6	Embayment	Spit Enclosed	4	11.1	3.21E+07	1.01E+07	2.64E+05	7.44E+06	2.94E+06	7.48E+05	7.31E+03	3.48E+03	3.44E+02
6	Severn Estuary	7	Coastal Plain	Funnel	5	12.3	7.14E+09	4.19E+09	2.00E+09	6.17E+08	4.42E+08	3.63E+08	2.81E+05	2.06E+05	1.36E+05
7															
8															
9															

Appendix B - Data set properties (DSproperties)

Data are stored in a *dstable*, which extends a Matlab *table* to hold more comprehensive metadata for multi-dimensional data sets. This makes use of a *dsproperties* class object to hold the metadata. The *dstable* and *dsproperties* classes are part of the *dstoolbox*. When loading data or saving model results the DSproperties can be defined, loaded and saved when creating a new Case within the application. These data are used in the application to provide descriptions of variables and dimensions in UIs, define units and formats and generic labels which are used for plots and analysis outputs. Further details of the *dstable* and *dsproperties* classes can be found in the Matlab Supplemental Software Documentation for the *dstoolbox*. An example of the code to load DSproperties for a time series and a variable with 2 spatial dimensions are shown below.

DSproperties for a set of timeseries variables.

```
dsp = struct('Variables', [], 'Row', [], 'Dimensions', []);

dsp.Variables = struct(...
    'Name', {'AvSpeed', 'MaxSpeed', 'Dir'}, ...
    'Description', {'Mean wind speed', 'Maximum wind speed', 'Mean wind direction'}, ...
    'Unit', {'m/s', 'm/s', 'deg'}, ...
    'Label', {'Wind speed (m/s)', 'Wind speed (m/s)', 'Wind direction (deg)'}, ...
    'QCflag', repmat({'raw'}, 1, 3));

dsp.Row = struct(...
    'Name', {'Time'}, ...
    'Description', {'Time'}, ...
    'Unit', {'h'}, ...
    'Label', {'Time'}, ...
    'Format', {'dd-MM-yyyy HH:mm:ss'});

dsp.Dimensions = struct(...
    'Name', {''}, ...
    'Description', {''}, ...
    'Unit', {''}, ...
    'Label', {''}, ...
    'Format', {''});
```

DSproperties for a 2D variable with 2 spatial dimensions.

```
dsp = struct('Variables', [], 'Row', [], 'Dimensions', []);
|
dsp.Variables = struct(...    %cell arrays can be column or row vectors
    'Name', {'u'}, ...
    'Description', {'Transport property'}, ...
    'Unit', {'m/s'}, ...
    'Label', {'Transport property'}, ...
    'QCflag', {'model'});

dsp.Row = struct(...
    'Name', {'Time'}, ...
    'Description', {'Time'}, ...
    'Unit', {'s'}, ...
    'Label', {'Time (s)'}, ...
    'Format', {'s'});

dsp.Dimensions = struct(...
    'Name', {'X', 'Y', 'Z'}, ...
    'Description', {'X co-ordinate', 'Y co-ordinate', 'Z co-ordinate'}, ...
    'Unit', {'m', 'm', 'm'}, ...
    'Label', {'X co-ordinate (m)', 'Y co-ordinate (m)', 'Z co-ordinate (m)'}, ...
    'Format', {'-', '-', '-'});
```

Appendix C – Sample DSProperties for data import

Text file of variable definitions

```
1 name→description→unit→label→qcflag/format
2 id→Index→-→Index→none
3 JNCC_Type→JNCC_Type→-→Estuary_type→none
4 ERP2_Type→ERP2_Type→-→Estuary_type→none
5 Type_id→Type.index→-→Estuary_type→none
6 Range→Tidal.range→m→Range→raw
7 Vmhw→Volume.at.Mean.High.Water→m^3→Volume→model
8 Vmtl→Volume.at.Mean.Tidel.Level→m^3→Volume→model
9 Vmlw→Volume.at.Mean.Low.Water→m^3→Volume→model
10 Smhw→Surface.area.at.Mean.High.Water→m^2→Surface.area→model
11 Smtl→Surface.area.at.Mean.Tide.Level→m^2→Surface.area→model
12 Smlw→Surface.area.at.Mean.Low.Water→m^2→Surface.area→model
13 Amhw→Cross-section.area.at.Mean.High.Water→m^2→Cross-section.area→model
14 Amtl→Cross-section.area.at.Mean.Tide.Level→m^2→Cross-section.area→model
15 Amlw→Cross-section.area.at.Mean.Low.Water→m^2→Cross-section.area→model
```

Matlab function with definition of all DSproperties:

```
function dsp = sample_dsproperties()
%sample function to define the dsproperties for a data set
%define a dsproperties struct and add the model metadata
dsp = struct('Variables',[],'Row',[],'Dimensions',[]);
%define each variable to be included in the data table and any
%information about the dimensions. dstable Row and Dimensions can
%accept most data types but the values in each vector must be unique

%struct entries are cell arrays and can be column or row vectors
dsp.Variables = struct(...
    'Name',{'id','JNCC_Type','ERP2_Type','Type_id','Range','Vmhw',...
        'Vmtl','Vmlw','Smhw','Smtl','Smlw','Amhw','Amtl','Amlw'},...
    'Description',{'Index','JNCC Type','ERP2 Type',...
        'Type index','Tidal range',...
        'Volume at Mean High Water',...
        'Volume at Mean Tidel Level',...
        'Volume at Mean Low Water',...
        'Surface area at Mean High Water',...
        'Surface area at Mean Tide Level',...
        'Surface area at Mean Low Water',...
        'Cross-sectional area at Mean High Water',...
        'Cross-sectional area at Mean Tide Level',...
        'Cross-sectional area at Mean Low Water'},...
    'Unit',{'-','-','-','-','m',...
        'm^3','m^3','m^3','m^2','m^2','m^2','m^2','m^2'},...
    'Label',{'Index','Estuary type','Estuary type',...
        'Estuary type','Range','Volume','Volume','Volume',...
        'Surface area','Surface area','Surface area',...
        'Cross-sectional area','Cross-sectional area','Cross-sectional area'},...
    'QCflag',repmat({'data'},1,14));
dsp.Row = struct(...
    'Name',{'Location'},...
    'Description',{'Location'},...
    'Unit',{'-'},...
    'Label',{'Location'},...
    'Format',{''});
dsp.Dimensions = struct(...
    'Name',{''},...
    'Description',{''},...
    'Unit',{'-'},...
    'Label',{''},...
    'Format',{''});
end
```