Preface

EstuaryDB is a MatlabTM App enables tables of estuary data (multi-estuary) to be loaded, plotted and analysed. Several be-spoke plotting functions are included to examine the variation amongst estuaries, including cross-plots to examine variations about central values (e.g., HW, MT, LW).

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. EstuaryDB uses the muitoolbox and dstoolbox.

Resources

The EstuaryDB App and two toolboxes (muitoolbox and dstoolbox) can be downloaded from [www.coastalsea.uk](http://www.coastalsea.uk).

*Cite as*:

Townend, I.H., 2021, EstuaryDB manual, CoastalSEA, UK, pp, [www.coastalsea.uk](http://www.coastalsea.uk).

Bibliography

Manning A J, 2012, Enhanced UK estuaries database: explanatory notes and metadata, Report No: TR167,pp. 1-20, Wallingford.

Whitehouse R J S, 2006, Review and formalisation of geomorphological concepts and approaches for estuaries, Joint Defra/EA Flood and Coastal Erosion Risk Management R&D Programme, Report No: FD2116/TR2 prepared by HR Wallingford, ABPmer and Prof Pethick,pp. 1-335, www.estuary-guide.net/pdfs/FD2116\_TR2.pdf, www.estuary-guide.net/pdfs/FD2116\_TR2.pdf

Townend I H, 2005, An examination of empirical stability relationships for UK estuaries. Journal of Coastal Research, 21 (5), 1042-1053.

Acknowledgements

Revision history

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 1.1 | May 2024 | Ported to use muitoolbox |
| 1.0 | Aug 2019 | First release via www.coastalsea.uk |

Contents

[1 Introduction 1](#_Toc166794529)

[2 Getting started 2](#_Toc166794530)

[2.1 Configuration 2](#_Toc166794531)

[2.1.1 Installing the toolboxes 2](#_Toc166794532)

[2.1.2 Installing the App 2](#_Toc166794533)

[2.2 Model Set-up 2](#_Toc166794534)

[3 Application Menus 4](#_Toc166794535)

[3.1 File 4](#_Toc166794536)

[3.2 Tools 4](#_Toc166794537)

[3.3 Project 4](#_Toc166794538)

[3.4 Setup 5](#_Toc166794539)

[3.5 Run 5](#_Toc166794540)

[3.6 Analysis 6](#_Toc166794541)

[3.6.1 Plotting 6](#_Toc166794542)

[3.6.2 Statistics 9](#_Toc166794543)

[3.7 Help 14](#_Toc166794544)

[3.8 Tabs 14](#_Toc166794545)

[3.9 UI Data Selection 14](#_Toc166794546)

[3.10 Accessing data from the Command Window 15](#_Toc166794547)

[4 Implementation/Supporting Information/Demonstration models 17](#_Toc166794548)

[4.1 Derive Output 17](#_Toc166794549)

[4.1.1 Calling an external function 18](#_Toc166794550)

[4.1.2 Input and output format for external functions 18](#_Toc166794551)

[4.1.3 Pre-defined functions 21](#_Toc166794552)

[5 Program Structure 26](#_Toc166794553)

[6 Bibliography 27](#_Toc166794554)

[Appendix - Data set properties (DSproperties) 28](#_Toc166794555)

# Introduction

EstuaryDB App …

# Getting started

## Configuration

XXX 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: XXX.mlappinstall

### Installing the toolboxes

The two toolboxes can be installed using the *Add-Ons>Manage Add-Ons* option on the Home tab of MatlabTM. 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.

### 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:

>> XXX;

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.

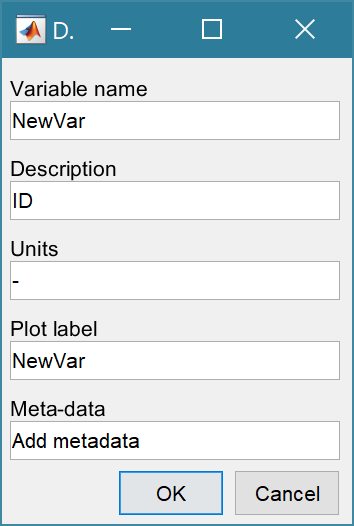
## Model Set-up

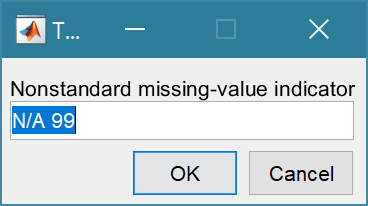
*File>New* to create a new project space.

*Setup>Input Data>Load*

The user is prompted for a file. Accepted formats are:

1. a table created in Matlab (e.g. save('JNCC\_var.mat','jnccdb'), where the table was assigned to the variable ‘jnccdb’) and saved as a mat file, or
2. 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 fist 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.

In both cases, 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 any additional meta-data that helps understand or identify the source of the variable.

To load the data the user is prompted to define any likely non-standard missing value indicators (e.g. 99, -999, na, Inf, etc). These are reassigned to the default value for the data type of the particular variable (e.g NaN for numeric data and ‘’ for character data).

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

*Setup>Add Data*: The user is first 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. 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.

The names of the files associated with each Case on the *Scenarios* tab are listed on the *Inputs* tab.

*Run> Run model*

The Run model option generates a Geyer-McCready plot using discharge and tidal range data.

*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.

# 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.

## 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.

## 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.

## Project

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

*Project>Cases>Edit Description*: select a scenario 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.

## 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, Delete, Quality Control. The availability of these options may vary depending on what is defined in the data specific format file.

Select one or more files 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 (scenario) and is listed on the *Cases* tab. The source file(s) area also listed on the *Inputs* tab.

*Setup>Import data> Load data*: prompts for file format to be loaded. The options available vary with Data type. The data is then loaded and the user is prompted for a description (working title) for the data set.

*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). Only files with the format used to create the data set can be used to Add data to a data record and this is selected when the first file is loaded using the Load menu option.

*Setup>Import data > Delete data*: prompts for Case from which some part of the data is to be deleted.

*Setup>Import data > Data QC*: runs a series of checks on the data. This is only available if defined for the specific data format.

*Setup> Input Parameters*:enter and edit the specified model parameters.

*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.

## Run

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

Graphical user interface, text, application, email

Description automatically generated*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 1.9 for details of how this works.

An equation is then defined in the text box below using the x, y, z or t variables[[1]](#footnote-1). 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 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 XX

## 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 1.9.

### 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.

|  |  |
| --- | --- |
|  | **2D plot**  For each selection choose theCase, 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 theCase, 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. |
|  | **4D**  For each selection choose theCase, Dataset and Variable to be used.  > Assign selections to the Var, X, Y and Z buttons  Take care to ensure that the assignments to X, Y and Z correctly match the dimensions selected for the variable (including any adjustment of the dimension ranges to be used).  > Select plot type.  To produce a new plot, use the Clear button to remove the previous selection.  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).

Animations follow a similar workflow. There are buttons at the bottom of each tab to:

**Run** the selection and create an animation,

**Save** the animation to a file (the animation needs to have been run first) . There is also an option to save on the bottom left of the animation figure.

**Clear** the current selection.

|  |  |
| --- | --- |
|  | **2DT animation**  For each selection choose theCase, Dataset and Variable to be used.  > Assign a variable, or a dimension, to the Var, Time and X buttons.  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  + : switches between cartesian and polar plot type  *If polar selected, then X assumed to be in degrees and when prompted select Polar and NOT Rose.* |
|  | **3DT animation**  For each selection choose theCase, Dataset and Variable to be used.  > Assign selections to the Var, Time, X and Y buttons  Take care to ensure that the assignments to Time, 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 2DT plot above. |
|  | **4DT animation**  For each selection choose theCase, Dataset and Variable to be used.  > Assign selections to the Var, Time, X, Y and Z buttons  Take care to ensure that the assignments to Time, X, Y and Z correctly match the dimensions selected for the variable (including any adjustment of the dimension ranges to be used).  > Select plot type.  Control Buttons: see 2DT plot above. |

***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 psfunctions 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.

### 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 1.9). 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.

1. Graphical user interface, application

   Description automatically generated**Regression**: generates a regresion plot of the dependent variable, Y, against the independed 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.

Chart

Description automatically generated

1. **Cross**-**correlation**: generates a cross-corrleation 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.
2. **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.

Graphical user interface, application

Description automatically generatedVarious ‘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.

1. **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.

1. **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.
2. **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.
3. **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 value for the fit.
4. **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 that 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; Antoniades *et al.*, 2021; Brandi and Di Matteo, 2021).

1. **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.

Graphical user interface, text, application, email

Description automatically generatedOnce 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.

Graphical user interface, text, application, email

Description automatically generated

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.

Chart, radar chart

Description automatically generated Taylor diagram legend includes: B – bias; E’’ – normalised RMS difference

Graphical user interface, text, application, email

Description automatically generatedThe 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.

**Intervals tab**

The Intervals tab allows the user to compute selected simple statistics of a timeseries variable for the intervals between the times recorded in another timeseries. For example the mean wave energy between beach profile surveys.

The Reference dataset defines the time intervals to be used. As only the times are used it does not matter which variable is selected. The Sample dataset is the timeseries variable to be used. The analysis is run using the Select button and the user is prompted to define the statistical function.

Graphical user interface, application

Description automatically generated

The UI provides a list of the functions that can be used. [*Any function that simply requires a single variable as an input could be used. To modify the list, edit the variable ‘statoptions’ in muiStats.getIntervalStats*.]

Default properties are derived from the source variable and the selected statistic. The user is prompted to confirm, or edit these default properties. The results are saved as a new timeseries dataset in which the variable describes the statistical value for the interval preceding each time.

## Help

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

## 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).

***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).

## UI Data Selection

Functions such as Derive Output (1.5), Plotting (1.6.1) and Statistics (1.6.2) 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:

**Graphical user interface, text, application, email

Description automatically generatedStep 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.

+ 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.

**Graphical user interface, text, application, email

Description automatically generatedStep 2 - Variable only has dimension of time**.

No selection to be made.

Edit range if required.

**Graphical user interface

Description automatically generatedStep 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.

## 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 MatlabTM, or to copy to other software packages. This requires use of the Command Window in MatlabTM, 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] | muuiCatalogue 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 stuct 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 *dstables* 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.

# Implementation/Supporting Information/Demonstration models

<*If required* *insert ‘Sec4.X\_DeriveOutput.docx’ here using copy and paste (keep source formatting). The heading numbers and bookmarks should update*>

## 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 1.1.2).

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 1.1.2 and YYY).

*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 MatlabTM expresion, or function call, can be used to create a new time series as follows:

x^2+y %time

### 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 XXX are detailed in Section 1.1.3.

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 1.1.2). 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.

### Input and output format for external functions

There are several possible use cases:

#### 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

* + - 1. *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

#### 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:

1. Input Cases – one of the datasets used in the function call;
2. New Case – use output to define a new dataset;
3. 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

#### 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

#### 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.

### Pre-defined functions

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

[*Select from the following according to the App*

This is illustrated in the Diffusion2D model. The function *userderivedoutput* can be called to generate either the integral under the surface at each time step, or the surface gradients at each time step.

For the integral option enter > userderivedoutput(t,x,y,z,’integral’)

For the gradient option enter > userderivedoutput(t,x,y,z,’gradient’)

|  |  |
| --- | --- |
| Graphical user interface, text, application, email  Description automatically generated | Graphical user interface, chart  Description automatically generated |

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.

1. ***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 ca 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.

1. ***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.

1. ***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.
2. ***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.

1. ***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.

1. ***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).

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.

1. ***Add sea level rise*** ***to tidal water levels*** (ie typically predictions rather than measured water levels). Based on exponential growth from 1900 and zeroed to a defined year using:

addslrtotides(X, t, delta, exprate, pivotyear), where X and t relate to the water level variable to be adjusted, delta is a rate for the year 1900 (e.g. 0.001 m/yr), *exprate* is the rate of exponential growth (e.g 0.011 for a fit to observations to-date) *pivotyear* is the year to use for zero sea level rise (e.g. 1900 adds slr based on change since 1900, whereas 2000 assumes that the tidal predictions are correct to the datum for the year 2000 and adjusts the record based on the slr function relative to that year).

1. ***Tidal range time series from a water level series***. The call to the function is:

tidalrange(X, t, issave, isplot) where X is the water level and t is the times of the water level values. Assumes that there are multiple water level values per tide. Also, outputs mean water level and tidal range values as a table. issave = true to save the results as a dataset and isplot = true to generate a plot of the results.

1. ***Fit trends and cycles to tidal range or high/low water time series***: The call to the function is:

tidalrange\_nltc(X, t, issave, isplot, ‘titletxt’) where X is the water level and t is the times of the water level values. Assumes that there are multiple water level values per tide. When issave = true the fit parameters of the selected are also saved. If isplot = true the fit plots are generated by the function in addition to the summary table of fit parameters. The ‘titletxt’ variable is optional and is used to define the output label used for the dataset.

1. ***Selection of plots for water level frequency and duration*** using the function:

waterlevelfreqplots (X, t) where X is the variable and t is time. Plot options include Water level elevation frequency, Water level spectrum, Elevations above a threshold, Duration of threshold exceedance, Elevation frequency above threshold. Designed to analyse water levels but could easily be adapted for other variables.

|  |  |
| --- | --- |
| Water level elevation frequency  1.5  a 0.5  -0.5  -1.5  Probability of occurrence (%) | 0.6  0.5  0.4  0.3  0.2  1.5  Duration frequency above 0.5 mOD  Duration (hours)  4.5  hr |
|  |  |

1. **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.

1. **Depth dependent wave steepness using the function:**

wave\_steepness(X, Y , Z, t) where X is the wave height, Y the wave period and Z the water depth, with t passing the time associated with the timeseries variables. Note that the water depth data should be a times series of the same length and at the same time intervals as the wave data, or specified as a single constant value (i.e.: wavesteepness(X, Y , 3.5, t)).

1. ***Wave height-period scatter plots*** using the function:

wave\_scatter(dst) where *dst* invokes the option to pass the data as dstables. This requires that the wave height, wave period and water depth time series to be used are assigned to the x, y and x buttons respectively.

1. ***Ratio of alongshore to cross-shore transport*** using the function:

beachtransportratio(x,theta) where X is the wave direction and theta is the beach or shoreline angle (both in degTN).

1. ***Chart

   Description automatically generated with medium confidenceLittoral drift statistics***. Plots the annual and monthly volumes of drift along with details of gaps and calms. The call to this function is:

littoraldriftstats(X, t, ’period’), where X is the rate of drift, time is the associated time for that variable and *period* can be ‘year’, or ’month’.

If no *period* is specified, the default is month. The *period* selection does not alter the plot (which shows both) but if the results are saved as a timeseries, *period* determines the timeseries interval. In the lower plot, the diamonds denote the start and end of the timeseries.

1. ***Ratio of alongshore to cross-shore transport***. The CERC formula for littoral transport is based on the energy flux (P) in the direction of wave advance per unit length of beach. ie: F = P.cos(), where  is the angle between wave crest and bed contour. The longshore component of energy flux is P.cos().sin(), which leads to the main terms in the CERC formula. It follows that the cross-shore component is P.cos2(). The ratio of longshore/cross-shore energy flux (or transport potential) = tan(). The call to the function is:

beachtransportratio(X,theta,isvector) where X is a timeseries of inshore wave directions, ‘*theta*’ is the angle of the shoreline to True North and ‘isvector’ is optional and is set to 1 or true if the direction is be retained in the output (default if 0/false). The sign convention is the same as for littoral drift: positive is left to right when looking at the shore from offshore.

1. **Examining the rate of beach profile change (accretion and erosion).** The function computes the change in the variable over each time interval and the rate of change (assuming a linear rate of change between surveys), subdivides the population into positive and negative change values (typically this represents accretion and erosion when using volumes, or shoreline position, data) and presents the histogram and exponential fit for each data set.

Warning: the results are entirely dependent on the adequacy of the source data to represent change (e.g. volumes or shoreline position). If the survey frequency is not regular the results are unlikely to be reliable.

Select a variable X. The variable can be any metric such as beach volume or shoreline position. Use the function:

posneg\_v\_stats(x, t, ’VariableName’), where x is a time series variable, such as beach volume of shoreline position, with associated time, t (defined by variable selection). The *VariableName* is optional but if used should be between single quote marks. The *VariableName* is used to label the plot axes.

The output comprises the following plots and dialogue box:

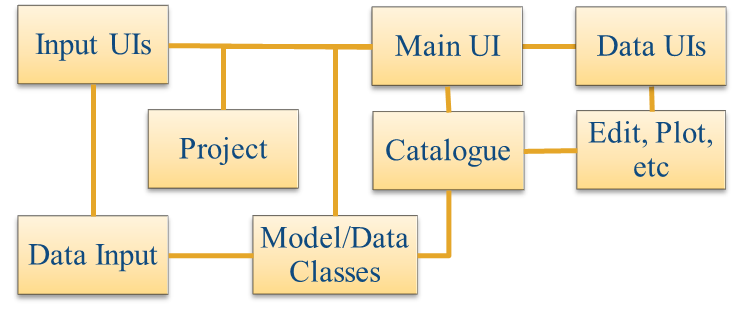
|  |  |
| --- | --- |
| Graphical user interface  Description automatically generated | Chart, histogram  Description automatically generated |
| Graphical user interface, text, application  Description automatically generated | Where dV is the change in the variable (e.g. volume) and dVdt is the rate of change. In the plot above, an exponential pdf has been fitted to the data and the coefficient is reported in the title for each subplot as the ‘mu’ value and listed in the dialogue box (as shown on the left). The probability density function (pdf) is:  A picture containing text, gauge  Description automatically generated |

*N.B. posneg\_dv\_stats.m requires the statistics and machine learning toolbox.*

# 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 EstuaryDB App using the following muitoolbox classes depicted in Figure 2, which shows a more detailed schematic of the program structure. See the muitoolbox and dstoolbox documentation for more details.

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



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

# 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 - 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.

|  |
| --- |
|  |

DSproperties for a 2D variable with 2 spatial dimensions.

|  |
| --- |
|  |

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. [↑](#footnote-ref-1)