



Preface

ModelUI is a Matlab™ App to demonstrate how to use the *multoolbox* to create bespoke interfaces for data analysis and modelling applications that produce some combination of graphical and/or time series outputs. The *multoolbox* is designed to enable the rapid prototyping of models by allowing the model developer to focus on the model, rather than the functional or operational needs of the software package itself. To this end, the UI provides a standard interface with drop-down menus, tools to open and close files, keep track of model runs, provide a rapid means to implement model set-up and data import and export, derivation of new variables, and some basic plotting and statistical tools. The ModelUI package includes three example models to illustrate how to create and modify the UI for different applications. These include:

VerticalProfile to illustrate the basic UI and a simple model producing spatial data

SimpleTide to illustrate handling timeseries data in a similar UI.

Diffusion2D to illustrate how to handle time plus 2 or 3 space dimensions.

Requirements

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

Resources

The ModelUI App and two toolboxes (*multoolbox* and *dstoolbox*) can be downloaded from www.coastalsea.uk.

Cite as:

Townend, I.H., 2021, ModelUI manual, CoastalSEA, UK, pp26, www.coastalsea.uk.

Bibliography

The models used to illustrate the UI are based on standard published methods, including:

Prandle, D., 1982. The vertical structure of tidal currents and other oscillatory flows. *Continental Shelf Research*, 1(2), 191-207.

Pugh D T, 1987, *Tides, Surges and Mean Sea-level*, John Wiley & Sons.

Acknowledgements

The Diffusion model is based on the code to solve the 2-D diffusion equation developed by Suraj Shanka, Copyright (c) 2012 and made available via the Matlab™ Exchange Forum.

Testing undertaken by Tian Qi.

Revision history

Version	Date	Changes
3.0	May 2021	ModelUI packaged as a Matlab App and migrated to use mui toolbox and dstoolbox.
2.1	Dec. 2020	Help text added to function. Various bug fixes. Package core suite as a toolbox.
2.0	July 2019	<p>Restructured UI, Data and Model classes to make Data and Models independent of class handles defined for a specific UI. This allows Data and Model classes to easily be used in any UI, or for multiple models to be included within a single UI. Unfortunately, this means that Classes developed for version 1.1 need to be updated (a relatively minor task) because they are no longer compatible.</p> <p>All imported data sets and model outputs now inherit either TSDataset for timeseries data and DSDataset for table data. These super classes in turn inherit the generic functions in DataSet. This allows data sets with different formats to be included in the same class (e.g. imported wave data with different variables and formats).</p> <p>Added descriptive and timeseries statistics class (DataStats) and range of functions (in MUIfunctions) so that Statistical analysis can be included in the UI.</p> <p>Added panel to tabs in DataGUIinterface to improve behaviour</p> <p>Improved behaviour when running multiple UIs/Models</p>
1.1	June 2018	<p>Minor corrections to improve cross-platform compatibility</p> <p>Fixed bugs when setting data ranges in Data GUIs and selecting data from a pop-up list.</p> <p>Improved plotting option selection for xyz data with no time</p> <p>Scenario selection added to control of Plot tab</p>
1.0	Mar 2018	Full release with some updates to code but no changes to model structure or requirements.
0.1	Jan 2018	Preliminary release via www.coastalsea.uk



Contents

1	Introduction	4
2	Getting started	5
2.1	Configuration.....	5
2.1.1	Installing the toolboxes	5
2.1.2	Installing the App	5
2.2	Model Set-up	5
3	Application Menus	6
3.1	File.....	6
3.2	Tools.....	6
3.3	Project.....	6
3.4	Setup.....	7
3.5	Run	7
3.6	Analysis.....	8
3.6.1	Plotting	9
3.6.2	Statistics.....	11
3.7	Help	12
3.8	Tabs	12
3.9	UI Data Selection	12
3.10	Accessing data from the Command Window	13
4	Demonstration models.....	15
4.1	Vertical Tidal Current Profile.....	15
4.1.1	Workflow to Run Model	15
4.1.2	Plotting results.....	15
4.2	Simple tide.....	16
4.3	Diffusion model.....	16
4.3.1	Functions to derive additional outputs	17
4.4	Derive Output.....	17
4.4.1	Calling an external function	18
4.4.2	Input and output format for external functions.....	19
4.4.3	Pre-defined functions	21
5	Program Structure.....	23
6	Bibliography	25

1 Introduction

Whether prototyping a new method or writing a new application, dealing with the coding infrastructure needed to handle the user interface, data input/output and plotting the results can be time consuming. ModelUI aims to speed up the process and provide users with a standard UI that can be rapidly adapted for new applications. This manual explains the functionality of the standard UI and how it can be used as the UI for other models. There are three sample models to illustrate different ways of using ModelUI. Within ModelUI there is a simple model for vertical velocity profiles. The SimpleTide model illustrates handling of timeseries data within an interface that uses ModelUI. The Diffusion2D model illustrates how to handle time plus 2 or 3 space dimensions with its own bespoke interface. These models are summarised in Section 4.

The models supplied with ModelUI are used to illustrate how to implement some basic uses of ModelUI and the muitoolbox. There are two ways of doing this:

- (i) using the ModelUI interface ‘as is’ and adding a new model to the interface.
- (ii) implementing a bespoke interface to provide the required functionality.

The requirements and options available are explained in the muitoolbox documentation and manual.

A simple model may only require the addition of one new class to implement using default ModelUI interface. However much more sophisticated applications are also possible. The ASMITA and CoastalTools applications use muitoolbox to implement bespoke interfaces to meet the needs of the application. They also provide a resource for many data handling methods that may be of use for other applications. These include some variations of the default statistical methods, as well as a variety of alternative plotting options.



2 Getting started

2.1 Configuration

ModelUI 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: `ModelUI.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:

```
>> ModelUI;
```

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

Documentation can be viewed from 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 Model Set-up

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

Setup>*Input Data*>*Model Data*

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

Run> *Run model*

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

The run is listed on the *Cases* tab and the tidal elevations for the most recent run can be viewed on the *Plot* tab.

Plot>*Plot menu*

The results from a run 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 scenarios, model setup, running and plotting of the results. In addition, Tabs are used to display set-up information of the Cases that have been run. In this manual text in *Red italic* refers to drop down menus and text in *Green italic* refers to Tab titles.

3.1 File

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

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

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

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

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

3.2 Tools

Tools>Refresh: updates *Cases* tab.

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

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

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

3.3 Project

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

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

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

Project>Cases>Save: select the Case to be saved from the list of Cases and is prompted to save the Case as a *dstable* or a *table* and then to 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 model run (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, 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 and then loads the data and prompts 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.

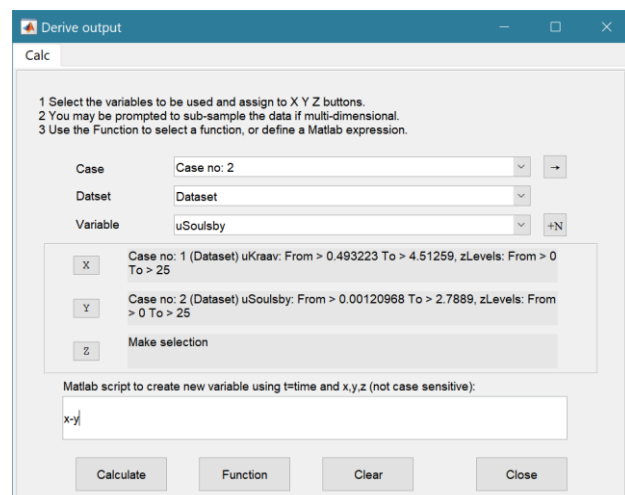
Setup>Input Data>Model Constants: various constants are defined for use in models, such as the acceleration due to gravity, viscosity and density of sea water, and density of sediment. Generally, the default values are appropriate (9.81, 1.36e-6, 1025 , 2650 respectively) but these can be adjusted and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

3.5 Run

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

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

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

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 (numeroe 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 4.4.

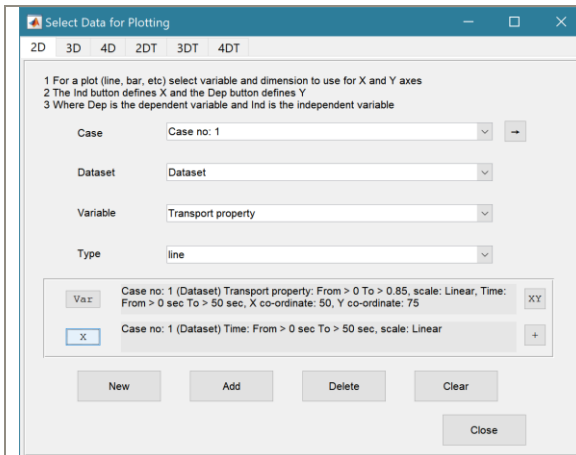
3.6 Analysis

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. The sections below present all the options available. However, the options enabled vary with each application. This can be changed by editing the DataUITabs property for Plot and Stats in the setMUI function of the model class (e.g. ModelUI, SimpleTide or Diffusion2D).

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.

3.6.1 Plotting

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



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)

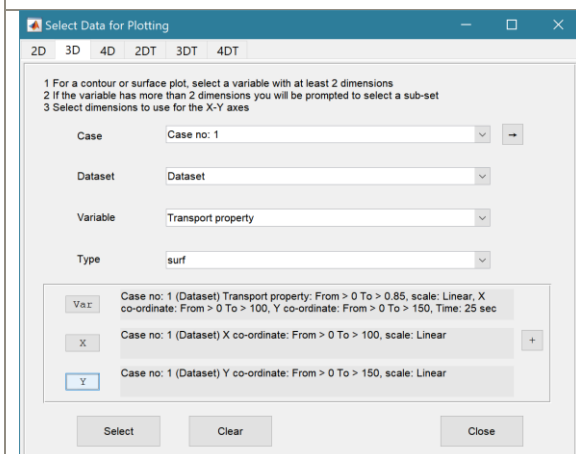
Control Buttons:

→ : 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

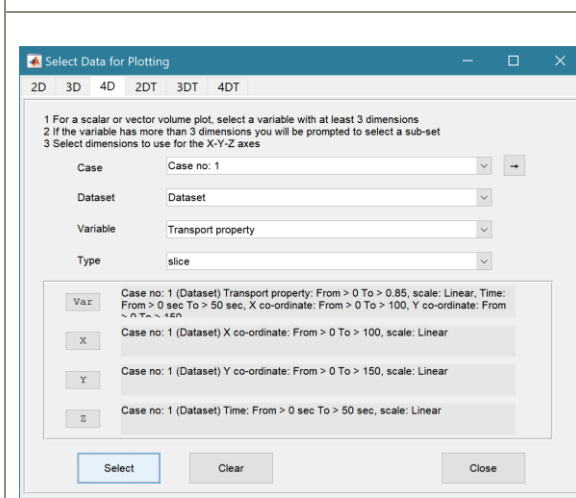
For each selection choose the Case, Dataset and Variable to be used.

> 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.



4D

For each selection choose the Case, 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.

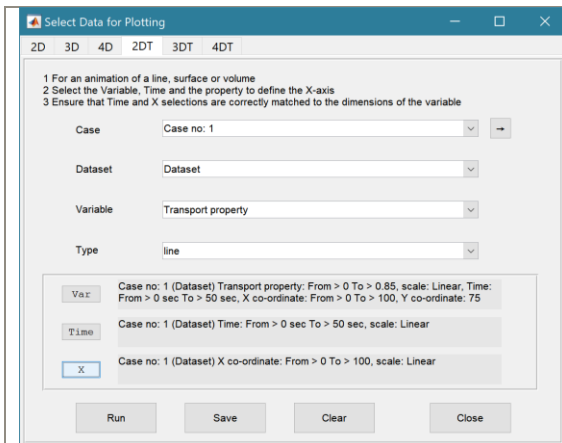
For all plot types, when the data has more dimensions than the plot or animation the user is prompted to sub-select from the data (by selecting sampling values for the dimensions that are not being used).

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 the Case, 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.

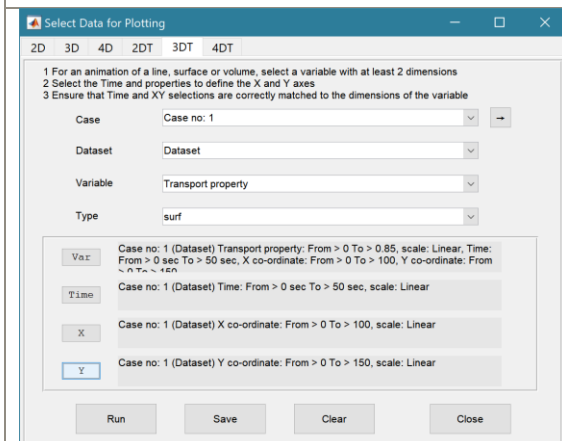
> Select plot type (line, bar, scatter, stem, etc)

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

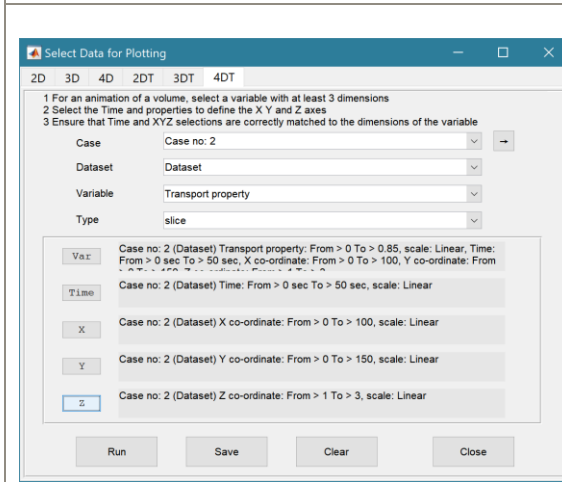
For each selection choose the Case, Dataset and Variable to be used.

> 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 the Case, 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 pfunctions folder and illustrates the workflow to a simple line plot using x-y data from the 2D tab and a surface plot using x-y-z data from the 3D tab.

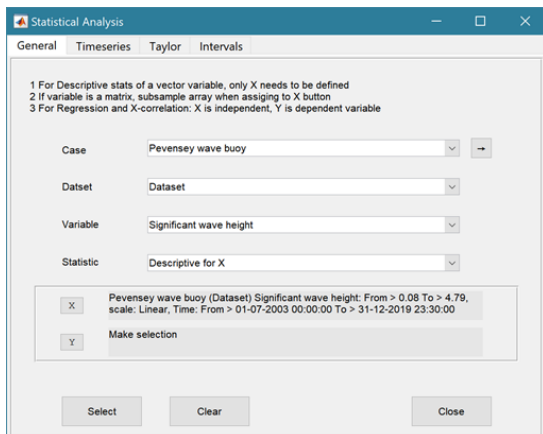
3.6.2 Statistics

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

General tab

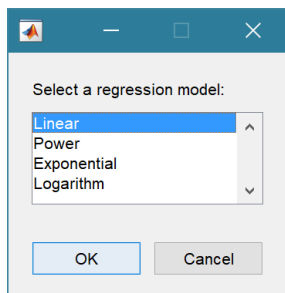
The General tab allows the user to apply the following statistics to data loaded in ModelUI:

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



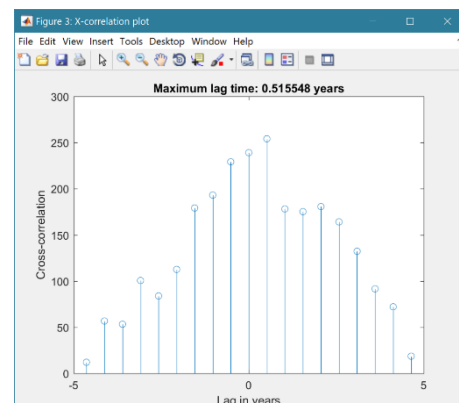
sum and linear regression fit parameters). Only X needs to be defined. The range of the variable can be adjusted when it is assigned to the X button (see Section 3.9). If the variable being used is a multi-dimensional matrix (>2D), the user is prompted to define the range or each additional dimension,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

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



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

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



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

3.7 Help

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

3.8 Tabs

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

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

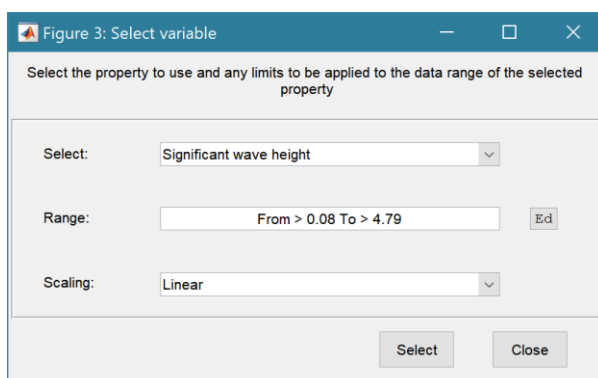
Inputs: tabulates the system properties that have been set (display only).

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

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

3.9 UI Data Selection

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



Step 1.

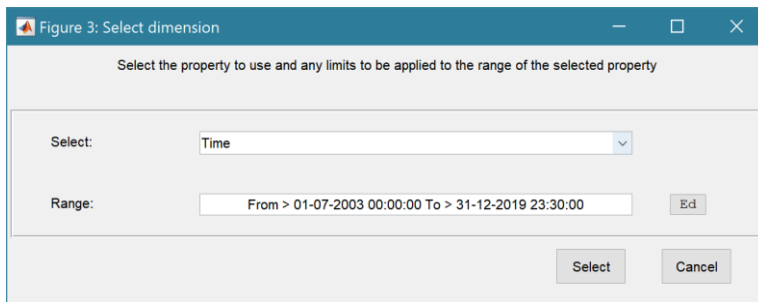
Select the attribute to use. This can be the variable or any of its associated dimensions, which are listed in the drop-down list.

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

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

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

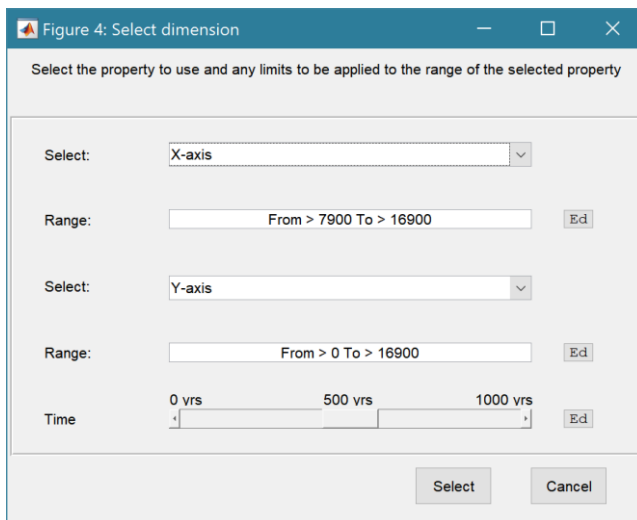
The number of variables listed on the UI depends on the dimensions of the selected variable. For each one Select the attribute to use and the range to be applied.



Step 2 - Variable only has dimension of time.

No selection to be made.

Edit range if required.



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

Select the 1st variable to use as a dimension.

Edit range if required.

Select the 2nd variable to use as a dimension.

Edit range if required.

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

Press Select to accept the selection made.

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

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

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

3.10 Accessing data from the Command Window

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

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

Simply typing:

```
>> myapp
```

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

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



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

To access current model settings, use the following:

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

To access the listing of current data sets, use:

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

To access imported or model data sets, use:

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

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

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

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

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

To access the underlying *table*, use:

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

The result can be assigned to new variables as required. Note that when assigning *dstable*s 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 Demonstration models

To illustrate the use of the interface for different types of output (graphical and time series), four sample models are provided. The first model is provided as the demonstrator model within ModelUI and computes the vertical tidal current profile. The second model generates a simple tidal curve, and the third generates an animation of 2D diffusion. These models also illustrate the following types of change to the core functionality of ModelUI:

SimpleTide – emulates tidal elevation and velocity as timeseries over a specified period.

Diffusion2D – The main purpose of this model is to illustrate the use of tables to hold a multi-dimensional array that defines a variable at each time-step (eg 1, 2 or 3D arrays to represent a variable in x, y and z).

The models themselves are briefly summarised in the following sections. How to modify ModelUI, or develop a new application, is explained in the mui-toolbox manual.

4.1 Vertical Tidal Current Profile

The model used in the ModelUI App implements the approach outlined by Prandle (1982) to compute the variation of the vertical profile given measurements at one elevation. This implementation has the option to use several different eddy viscosity formulations when computing the profile.

4.1.1 Workflow to Run Model

The following outlines the steps in a typical workflow to setup and run the vertical tidal current profile mode.:

File>New – create a new project (name and date)

Setup>Input Data>Model Data – define model parameters (see Prandle (1998) for details of the parameters required. To see the parameters that are currently set for the model use the *Inputs* tab.

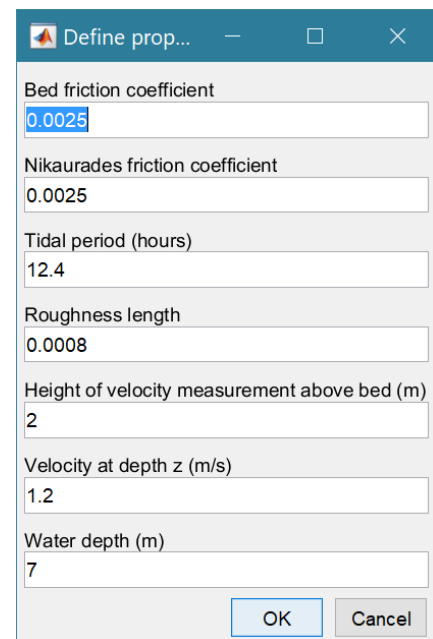
File>Save as – save model setup to a *.mat file.

Run>Run model – runs the model and prompts user for a description of the scenario. The results can be viewed on the *Q-Plot* tab.

Completed ‘Cases’ are listed based on the user descriptions on the *Cases* tab.

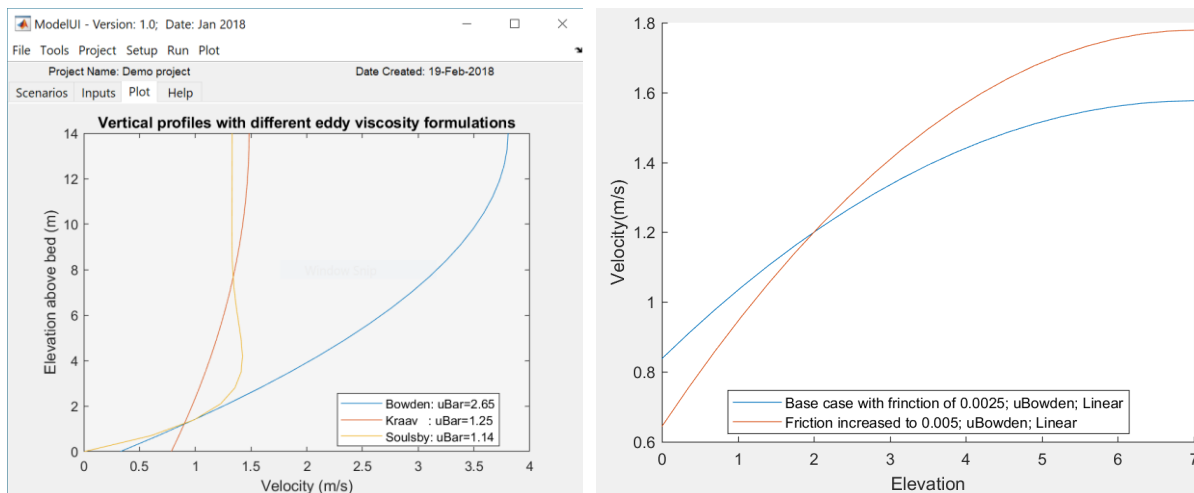
4.1.2 Plotting results

Results can be plotted using the plotting UI (*Plot>Plot Menu*). Eddy viscosity cases can be selected individually and plotted against profiles from other Cases.



Parameter	Value
Bed friction coefficient	0.0025
Nikaurades friction coefficient	0.0025
Tidal period (hours)	12.4
Roughness length	0.0008
Height of velocity measurement above bed (m)	2
Velocity at depth z (m/s)	1.2
Water depth (m)	7

Figure 1 – Vertical profiles from a model run plotted in UI Plot tab and comparing several Cases using the Plot Menu



4.2 Simple tide

SimpleTide provides a simple representation of the diurnal-semidiurnal and spring-neap variations in the tide based on simple summation of M2, S2 and O1 contributions, scaled to the defined tidal range. Tidal currents are derived in a similar way and scaled to the defined tidal velocity amplitude. The model generates a time series of tidal amplitude, vertical and horizontal velocity based on defined amplitude and phases. The model is implemented in two ways, to illustrate different ways of adding models

- (i) By defining a new class, STData for data input and a model function file simpletidemodel. This runs within ModelUI and is similar to the implementation of the Vertical Profile model.
- (ii) by defining three new classes: SimpleTide, STData and STModel. The SimpleTide class inherits from ModelUI and redirects menu options to STData and STModel. This illustrates a minimal adjustment to add a new model. The STData and STModel classes simply define a bespoke data input and replace Model in ModelUI.

4.3 Diffusion model

Diffusion2D implements the 2-D Diffusion equation using a Finite Difference Method. The Numerical scheme used is first order upwind in time and second order central difference in space, with Implicit and Explicit options and either Dirichlet or Neumann boundary conditions. The model operates on a rectangular domain and is perturbed over a defined area at $t=0$ and then allowed to diffuse at a rate determined by the diffusion coefficient. This application implements the solution of the 2-D diffusion equation based on the code developed by Suraj Shanka, Copyright (c) 2012 and made available via the Matlab™ Exchange Forum. The main purpose of this application is to provide time varying 2-D data (and 3-D by replicating the 2-D matrix) to demonstrate the use of variables that have dimension of time and xyz. A *dstable* can hold multiple variables for each time step, each with a consistent xyz definition (i.e. the size of the variable array at each time step is constant). The positions that define XYZ and Time are stored as properties of the *dstable* and each (multi-dimensional) variable is a column vector in the table. If the variable array remains constant but the positions (xyz) change with time, then xyz need to be added as variables rather than as an xyz definition. The model is implemented by defining four new classes: Diffusion2D, DFmodel, DFparams and DFrundata. In addition, the function diffusion2Dmodel does the diffusion computations. DFparams and DFrundata classes are for data entry, illustrating how to separate out different aspects of the model input

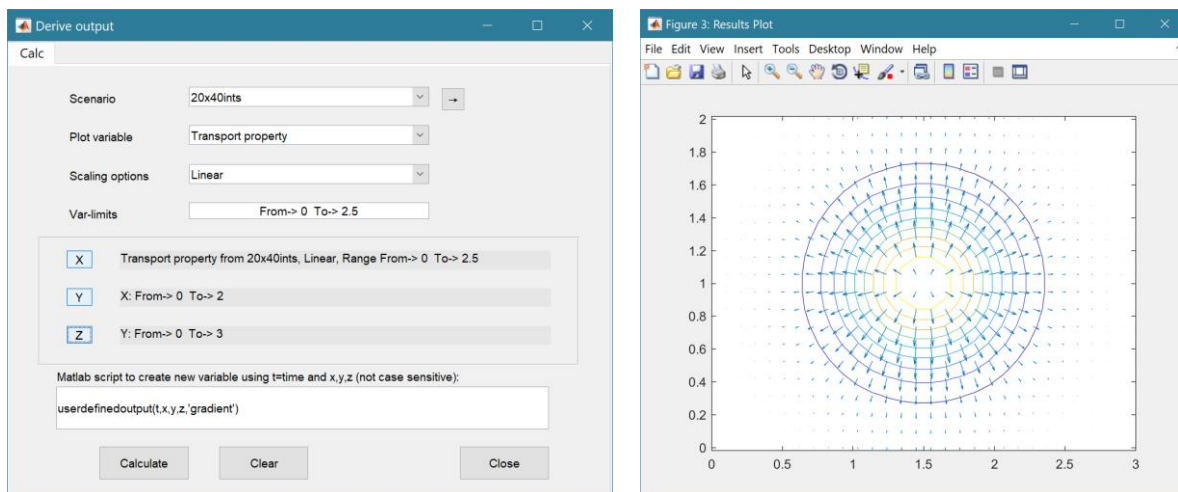
information. The Diffusion2D class inherits ModelUI to provide model specific functionality and the DFmodel class implements the control of model runs, tab plot display and holds the model results.

4.3.1 Functions to derive additional outputs

The function `userderivedoutput` can be called using the Derive Output UI, 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')`



4.4 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:

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 4.4.2).

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 4.4.2 and 3.5).

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

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

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

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

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

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

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

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

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

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

4.4.1 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 ModelUI are detailed in Section 4.4.3.

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 4.4.2). If the variables have a dimension (e.g., *time*) then this should be the first variable, with other variables following. If there is a need to handle additional dimensions then use the option to return a `dstable`.

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

An alternative when calling external functions is to pass the selected variables as `dstable`s, thereby also passing all the associated metadata and `RowNames` for each dataset selected. For this option up to 3

variables can be selected and assigned to the X, Y, Z buttons but they are defined in the call using *dst*, for example:

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

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

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

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

4.4.2 Input and output format for external functions

There are several possible use cases:

4.4.2.1 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
```

4.4.2.2 Single value output

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

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

4.4.2.3 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
```

4.4.2.4 Using dstables

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

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

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

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

Adding functions to the Function library

To simplify accessing and using a range of functions that are commonly used in an application, the function syntax can be predefined in the file `functionlibrarylist.m` which can be found in the `utils` folder of the `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.

4.4.3 Pre-defined functions

The following examples are provided within ModelUI, 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) >` where `n` specifies the number of points to average over.
2. 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, `t` 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.
3. 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) %time>`, 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.
4. 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) %time>`, 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.

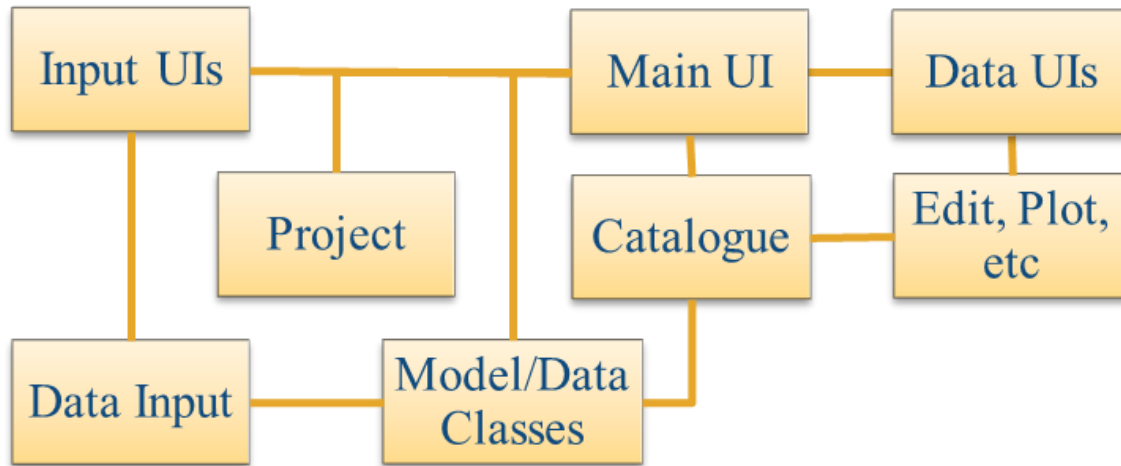
5. Recursive plot. Generates a plot of a variable plotted against itself with an offset (e.g. $x(i)$ versus $x(i+1)$). This is called from the Derive Output GUI using: `<recursive_plot(x, 'varname', nint)>`, where x is the variable, 'varname' is a text string in single quotes and $nint$ is an integer value that defines the size of the offset.
6. 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).

The Function button on the Derive Output UI provides access to these predefined functions and allows the user to select one and load a function call template into the UI equation text box. The list is defined in the function `functionlibrarylist.m` which is in the `muifunctions` folder.

5 Program Structure

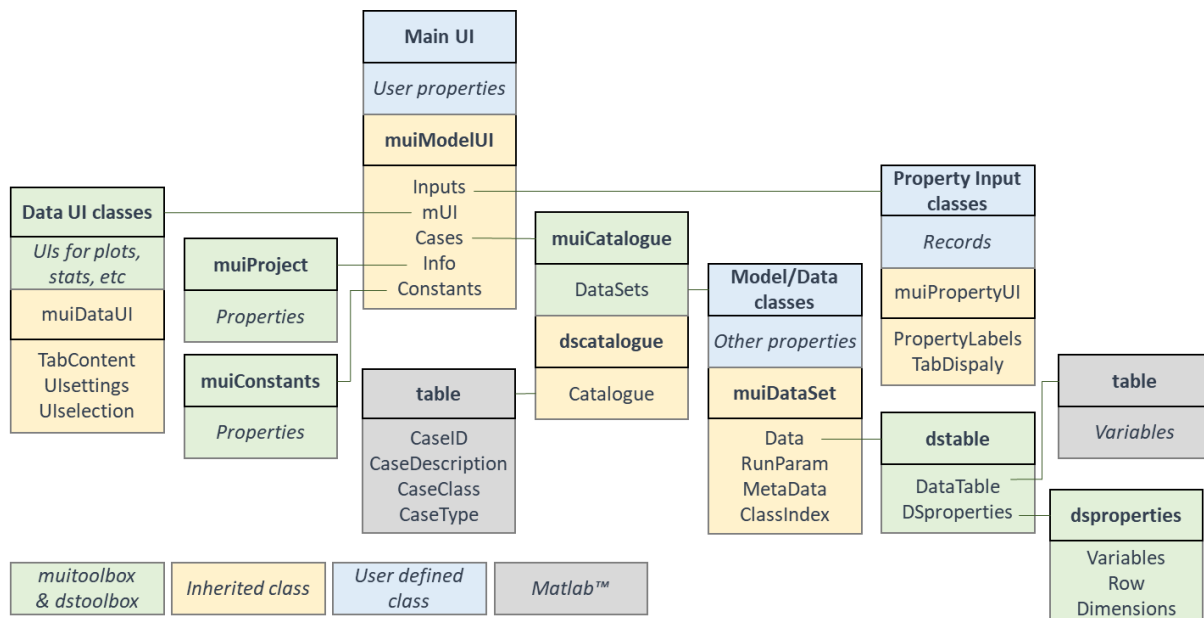
The overall structure of the code is illustrated schematically in Figure 2. This is implemented through several classes that handle the graphical user interface and program workflows (Main UI) and several classes that handle the data manipulation and plotting (Input UIs and Data UIs).

Figure 2 -- High level schematic of program structure



The interfaces and default functionality are implemented in the ModelUI App using the following mui toolbox classes depicted in Figure 3, which shows a more detailed schematic of the program structure. See the mui toolbox and dstoolbox documentation for more details.

Figure 3 – schematic of program structure showing how the main classes from mui toolbox and dstoolbox are used



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

VPparam - handles the input of parameters required by the model.

VPdata – allows data to be imported for comparison with the model results.

VPdata_ff – is an alternative to **VPdata** (edit call in ModelUI to use this option). The class inherits **muiDataSet** but uses the same format file (VPformat.m).

VPmodel – defines model output properties and implements model functions

VPformat is used by **VPdata** and defines the read format and metadata for data being imported.

The SimpleTide example application uses the following classes and functions:

SimpleTide - inherits from ModelUI and redirects menu options to STData and STModel

STparam - handles the input of parameters required by the model.

STdata – allows data to be imported for comparison with the model results.

STmodel – defines model output properties and implements model functions.

STformat is used by **STdata** and defines the read format and metadata for data being imported.

simple_tide is used to compute the variation of tidal elevation and velocity for given constituents.

The Diffusion2D example application uses the following classes and functions:

Diffusion2D – inherits from muiModelUI and defines additional options in the user interface.

DFparams - handles the input of parameters required by the model.

DFrunprops – handles the run time properties.

DFmodel – defines model output properties and implements model functions.

diffusion2Dmodel is a function modified from the code of Suraj Shanka, Copyright (c) 2012, downloaded from the Matlab™ Exchange Forum.

The function *userderivedoutput* can be called from the Derive Output menu option. Select the x, y and z variables of the diffusion dataset as X, Y, Z. Then call the function *userderivedoutput(t,x,y,z,<option>)*, where <option> is either 'integral' or 'gradient'. The user is then prompted to name the new data set and define the display variable (ResDef). The new data set is saved as ModelData in the Derived class.

6 Bibliography

- Bartlett, R., Mortimer, R.J.G., Morris, K., 2008. Anoxic nitrification: Evidence from Humber Estuary sediments (UK). *Chemical Geology*, 250(1-4), 29-39.
- Coles, S., 2001. *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer-Verlag, London.
- Prandle, D., 1982. The vertical structure of tidal currents and other oscillatory flows. *Continental Shelf Research*, 1(2), 191-207.
- Taylor, K.E., 2001. Summarizing multiple aspects of model performance in a single diagram. *Journal of Geophysical Research - Atmospheres*, 106(D7), 7183-7192.
- Townend, I.H., 2008a. Breach design for managed realignment sites. *Proc.Instn Civ.Engrs., Maritime Engineering*, 161(MA1), 9-21.
- Townend, I.H., 2008b. Hypsometry of estuaries, creeks and breached sea wall sites. *Proc.Instn Civ.Engrs., Maritime Engineering*, 161(MA1), 23-32.