## 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

#### 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 are 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. ***Scale water levels:*** The call to the function is:

scale\_waterlevels (X,t,issave,isplot), where X is the water level and t are the times of the water level values. User is prompted to give a scaling factor for values > and <= to 0.

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 are 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.*

### Adding variables to peak and cluster time series

Peak and cluster time series are a subset of the source data set, saved as an independent record. For some workflows, there may be a need to add other variables with the same date-time as the subseries (e.g. to produce a polar plot of the cluster events, direction must be added to the timeseries, or to run a model (such as overtopping) period and direction may need to be added). This can be done using the Derive Output GUI, by selecting the subseries variable (e.g. wave energy clusters) as X (it must be X) and the source data set variable (e.g. wave period or direction) as the Y variable.

Then for the equation string, use a function of the form > *Y.\*(X./X)*

This simply adds the variable defined by Y at the intervals defined by X (assuming X is the subseries). The new variable is then named as required. However, if the new variable is *direction* and it is to be used in the plotting routines (e.g. wave roses) the Variable Name MUST begin with ‘Dir’ but this can be followed by any additional characters.