# Pdap Manual

**Pedersen, Mads Mølgaard; Larsen, Torben J.**

*Publication date:*
2014

*Document Version*
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

*Citation (APA):*
Pedersen, M. M., & Larsen, T. J. (2014). *Pdap Manual*. DTU Wind Energy. DTU Wind Energy E No. 0032

# Pdap Manual

Department of Wind Energy E Report 2014

Mads M. Pedersen, Torben J. Larsen

DTU Wind Energy E-0032

December 2014

DTU Vindenergi
Institut for Vindenergi

**Authors:** Mads M. Pedersen, Torben J. Larsen
**Title:** Pdap Manual
**Department:** Aeroelastic design (AED)

**DTU Wind Energy E-0032**
**December 2014**

**Summary (max 2000 characters):**
Pdap, Python Data Analysis Program, is a program for post processing, analysis, visualization and presentation of data *e.g. simulation results and measurements*. It is intended but not limited to the domain of wind turbines.
It combines an intuitive graphical user interface with python scripting that allows automation and implementation of custom functions.
This manual gives a short introduction to the graphical user interface, describes the mathematical background for some of the functions, describes the scripting API and finally a few examples on how automate analysis via scripting is presented.
The newest version, and more documentation and help on how to used, extend and automate Pdap can be found at the webpage www.hawc2.dk

**ISBN: 978-87-92896-46-9**

**Danmarks Tekniske Universitet**
DTU Vindenergi
Nils Koppels Allé
Bygning 403
2800 Kgs. Lyngby
Telephone  40816644

mmpe@dtu.dk
www.vindenergi.dtu.dk

# Content

# 1. Summary

Pdap, Python Data Analysis Program, is a program for post processing, analysis, visualization and presentation of data *e.g. simulation results and measurements*. It is intended but not limited to the domain of wind turbines. It combines an intuitive graphical user interface with python scripting that allows automation and implementation of custom functions. This manual gives a short introduction to the graphical user interface, describes the mathematical background for some of the functions, describes the scripting API and finally a few examples on how automate analysis via scripting is presented.

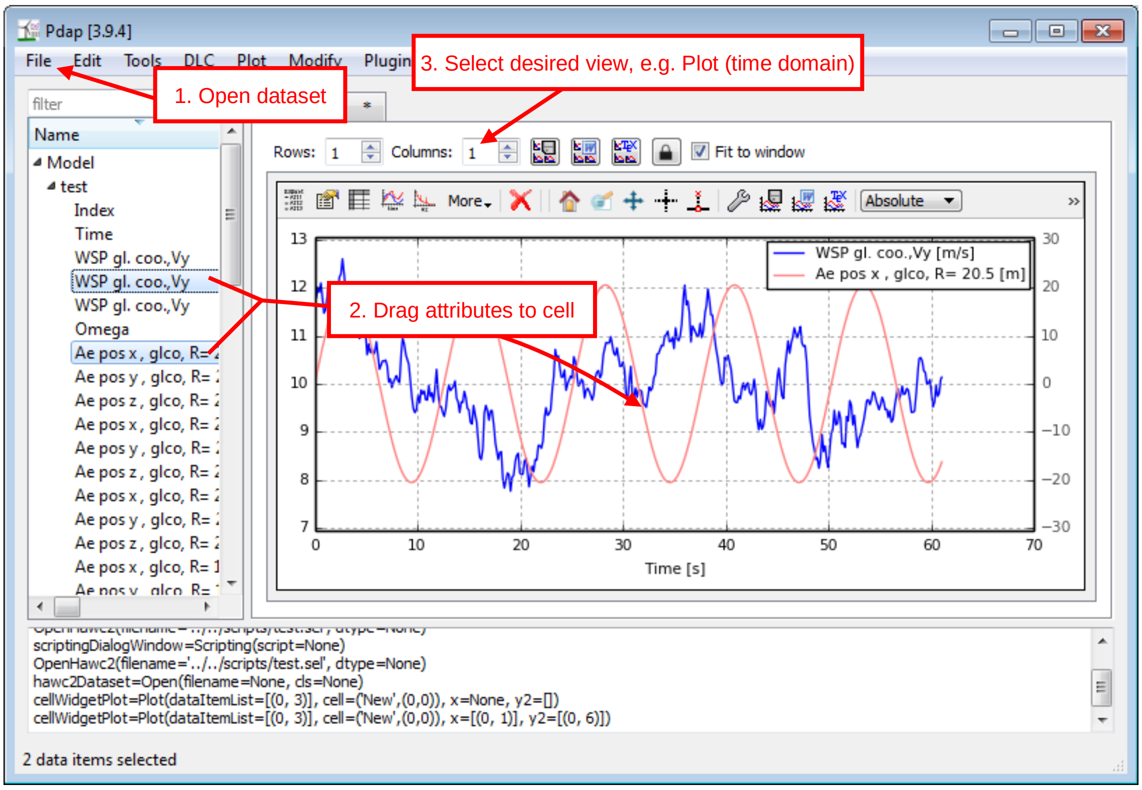
The newest version, and more documentation and help on how to used, extend and automate Pdap can be found at the webpage www.hawc2.dk

# 2. Terminology

| | |
|---|---|
| Attribute | Item in dataset (often a sensor). |
| Cell | Graphical sub element of a tab. Displays a cell widget |
| Cell widget | Graphical element presenting data items, e.g. data info, data view, plot (time domain) |
| Data item | Dataset, attribute or attribute group |
| Data model | Root of data item hierarchy |
| Dataset | Item in data model having one or more attributes. Can be opened from and saved to files |
| Sensor | See Attribute |
| Tab | Tab in tab pane. Contains one or more cells |
| Widget | See cell widget |

# 3. Quick guide: Plot time series

Attributes, e.g. time series of data from sensors in a measurement setup, are easily plotted, see Figure 1.


Pdap [3.9.4]
File Edit Tools DLC Plot Modify Plugin
1. Open dataset
3. Select desired view, e.g. Plot (time domain)
2. Drag attributes to cell
Rows: 1 Columns: 1 Fit to window
More
Absolute
Name
Model
test
Index
Time
WSP gl. coo.,Vy
Omega
Ae pos x , glco, R=
WSP gl. coo.,Vy [m/s]
Ae pos x , glco, R= 20.5 [m]
Time [s]

```
scriptingDialogWindow=Scripting(script=None)
OpenHawc2(filename='../../scripts/test.sel', dtype=None)
hawc2Dataset=Open(filename=None, cls=None)
cellWidgetPlot=Plot(dataItemList=[(0, 3)], cell=('New',(0,0)), x=None, y2=[])
cellWidgetPlot=Plot(dataItemList=[(0, 3)], cell=('New',(0,0)), x=[(0, 1)], y2=[(0, 6)])
```

2 data items selected

**Figure 1 – Plotting attributes**

# 4. Installation

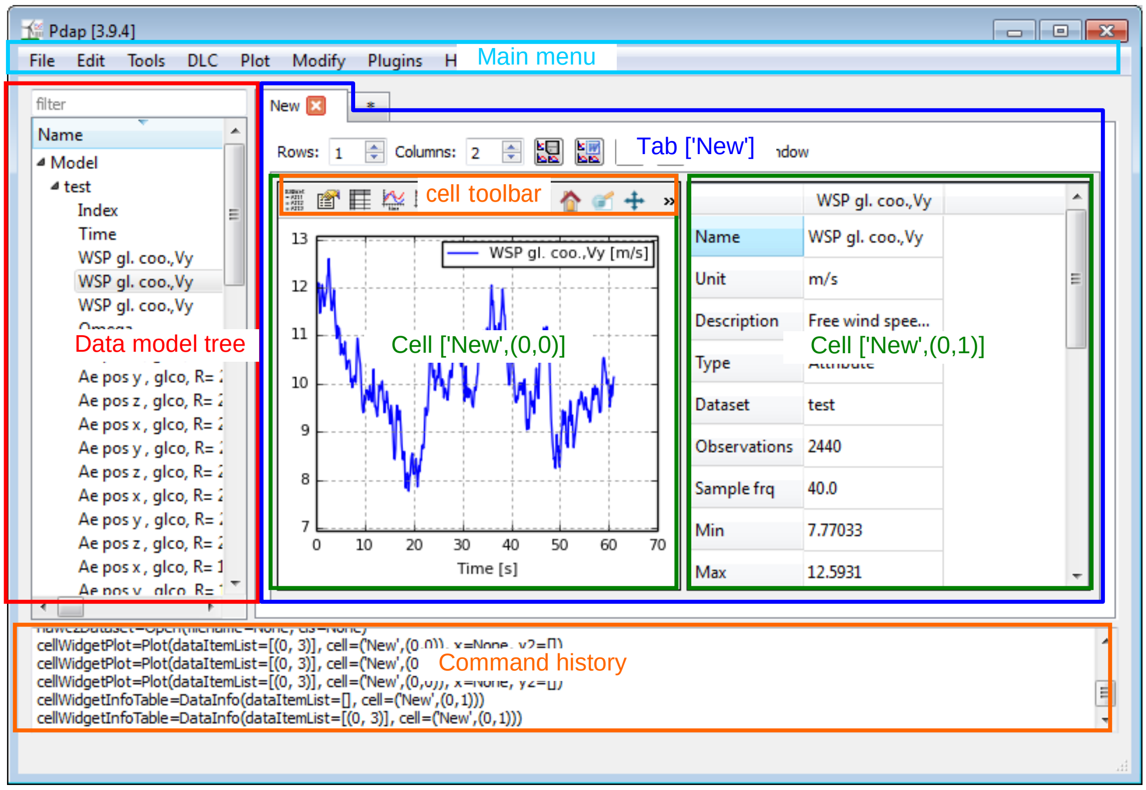
## 4.1 Standalone (windows only)

A standalone version of Pdap for Windows can be downloaded via www.hawc2.dk. It includes all necessary packages and dependencies.

# 5. The graphical user interface

## 5.1 Overview

The graphical user interface consist of a main menu, a data model tree showing opened datasets and their attributes, a tab pane where each tab contains one or more cells and finally a command history in the bottom, see Figure 2.



| | WSP gl. coo.,Vy |
|---|---|
| Name | WSP gl. coo.,Vy |
| Unit | m/s |
| Description | Free wind spee... |
| Type | Attribute |
| Dataset | test |
| Observations | 2440 |
| Sample frq | 40.0 |
| Min | 7.77033 |
| Max | 12.5931 |

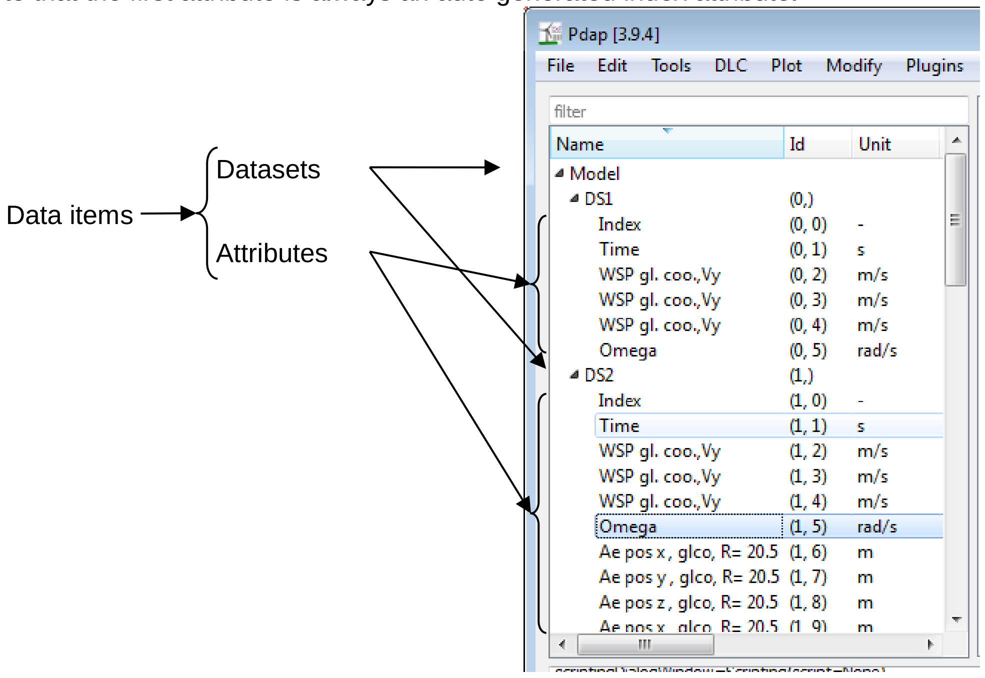
```
hawc2Dataset=Open(filename=None, cls=None)
cellWidgetPlot=Plot(dataItemList=[(0, 3)], cell=('New',(0,0)), x=None, y2=[])
cellWidgetPlot=Plot(dataItemList=[(0, 3)], cell=('New',(0
cellWidgetPlot=Plot(dataItemList=[(0, 3)], cell=('New',(0,0)), x=None, y2=[])
cellWidgetInfoTable=DataInfo(dataItemList=[], cell=('New',(0,1)))
cellWidgetInfoTable=DataInfo(dataItemList=[(0, 3)], cell=('New',(0,1)))
```

**Figure 2 - The graphical user interface**

## 5.2 Data model tree

The data model tree show the name, id, unit and description of all open datasets, their attributes, and attribute groups, see Figure 3.

Note that the first attribute is always an auto generated index attribute.

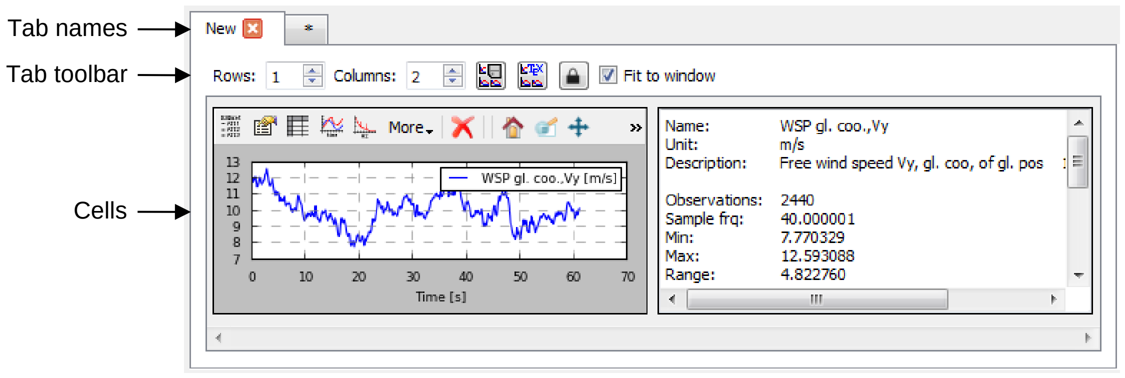


| Name | Id | Unit |
|---|---|---|
| Model | | |
| DS1 | (0,) | |
| Index | (0, 0) | - |
| Time | (0, 1) | s |
| WSP gl. coo.,Vy | (0, 2) | m/s |
| WSP gl. coo.,Vy | (0, 3) | m/s |
| WSP gl. coo.,Vy | (0, 4) | m/s |
| Omega | (0, 5) | rad/s |
| DS2 | (1,) | |
| Index | (1, 0) | - |
| Time | (1, 1) | s |
| WSP gl. coo.,Vy | (1, 2) | m/s |
| WSP gl. coo.,Vy | (1, 3) | m/s |
| WSP gl. coo.,Vy | (1, 4) | m/s |
| Omega | (1, 5) | rad/s |
| Ae pos x , glco, R= 20.5 | (1, 6) | m |
| Ae pos y , glco, R= 20.5 | (1, 7) | m |
| Ae pos z , glco, R= 20.5 | (1, 8) | m |
| Ae pos x , glco, R= 20.5 | (1, 9) | m |

**Figure 3 - The data model tree**

## 5.3 Tabs

The tab pane allows the user to work with multiple tabs, see Figure 4. Tabs can be renamed via the popup menu that appears by right-clicking on the tab name.
The order of the tabs can be changed by dragging the tab name.



Name: WSP gl. coo.,Vy
Unit: m/s
Description: Free wind speed Vy, gl. coo, of gl. pos

Observations: 2440
Sample frq: 40.000001
Min: 7.770329
Max: 12.593088
Range: 4.822760

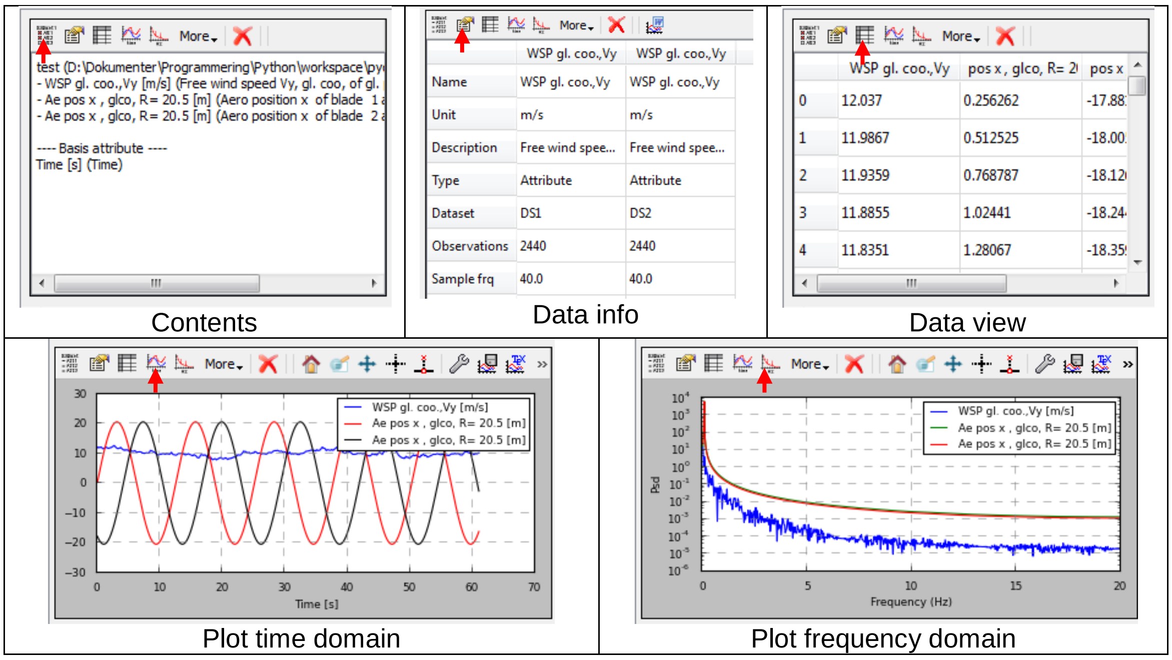
**Figure 4 - The tab pane**

The numbers of rows and columns of cells of a tab can be specified in the tab toolbar.
As default the position of the cells are locked, but after unlocking (click on the padlock icon) the cells can be interchanged by drag and drop.
Finally the size of the cells can be set manually by a width and a height slider that appears when the 'Fit to window' checkbox is unchecked.

## 5.4 Cells

Cells are visual boxes that present their data items by one of several views.
Data items, i.e. datasets, attributes or attribute groups, are added to a cell from the data model tree by drag and drop or double click. In order to append data items instead of replacing, the <ctrl>-key must be pressed during drop or double click.
The cell can be emptied by clicking the ✗-icon on the cell toolbar
There are five standard widgets, see Figure 5:

| | |
|---|---|
| Contents: | Lists the data items of the cell |
| Data info: | Shows info about the data items |
| Data view: | Table view showing the raw data of the attributes |
| Plot time domain | Plot the attributes in time domain |
| Plot frequency domain | Plot the attributes in time domain |



```
test (D:\Dokumenter\Programmering\Python\workspace\py
- WSP gl. coo.,Vy [m/s] (Free wind speed Vy, gl. coo, of gl.
- Ae pos x , glco, R= 20.5 [m] (Aero position x  of blade  1 a
- Ae pos x , glco, R= 20.5 [m] (Aero position x  of blade  2 a

---- Basis attribute ----
Time [s] (Time)
```

| | WSP gl. coo.,Vy | WSP gl. coo.,Vy |
|---|---|---|
| Name | WSP gl. coo.,Vy | WSP gl. coo.,Vy |
| Unit | m/s | m/s |
| Description | Free wind spee... | Free wind spee... |
| Type | Attribute | Attribute |
| Dataset | DS1 | DS2 |
| Observations | 2440 | 2440 |
| Sample frq | 40.0 | 40.0 |

| | WSP gl. coo.,Vy | pos x , glco, R= 2 | pos x |
|---|---|---|---|
| 0 | 12.037 | 0.256262 | -17.88 |
| 1 | 11.9867 | 0.512525 | -18.00 |
| 2 | 11.9359 | 0.768787 | -18.12 |
| 3 | 11.8855 | 1.02441 | -18.24 |
| 4 | 11.8351 | 1.28067 | -18.35 |

**Figure 5 – The five standard views: Contents, Data info, Data view, Plot time domain, Plot frequency domain**

In the 'more' menu some additional views/tools are found, e.g. scatter plot, kaimal fit and histogram.

# 6. Functions

## 6.1 Open dataset

Datasets can be opened in several ways:

- Main menu - File - Open - [HAWC2, CSV, FLEX4, DAQWin, ...]
- Main menu - File - Open... (ctrl+O)
  (If the format cannot be determined from the file name, the format must be chosen from a list)
- Right clicking in data model tree opens a popup menu that provides options similar to the main menu
- Drag a file from for instance Windows Explorer and drop it in the data model tree
- Copy a file from for instance Windows Explorer and paste (Ctrl+V or right-click menu) it into the data model three

The following file formats are supported:

| Name | Description | Associated extension | Data format |
|---|---|---|---|
| HAWC2 ASCII | ASCII output from HAWC2 | *.sel | ASCII |
| HAWC2 BINARY | Binary output from HAWC2 | *.sel | INT16 with scalefactors |
| HAWC2(Subset) | Similar to 'HAWC2', but this option allows the user to select a subset of the sensors and/or a subset of the observations to be opened. Practical when dealing with huge datasets | *.sel | INT16 with scale factors |
| CSV | Comma, semicolon or whitespace separated files. Support for both comma and dot as decimal separator. | *.csv | ASCII |
| FLEX4 | FLEX4 file | *.int | INT16 with scale factors |
| DAQWin | DAQWin file stored in folder or zip file | *.tim / *.zip | FLOAT32 |

## 6.2 The edit menu

The edit menu provides functions to work with data items:

| | |
|---|---|
| Rename | Edit name, unit, and description of data item |
| Add dataset | Add dataset to model |
| Add attribute | Add attribute to dataset |
| Delete | Delete data item |
| Reload | Reload dataset from file |
| Reset | Close all tabs and remove all data items from data model |
| Copy/paste | Copy and paste attributes into existing or a new dataset |
| Concatenate | Concatenate two or more datasets to one time continuous dataset |
| Group | Groups two or more attributes. Useful when working with many attributes |

## 6.3 Plotting

As seen in Figure 6 the canvas of the 'Plot' view are divided into sections, that allows the user change the default x-attribute (time or index) to a custom attribute by dragging the desired attribute to the green section.

Dropping attributes in the red field, adds the attributes to a secondary axis, which scales independently of the main axis. In this way two attributes with different mean values can be compared in the same plot.

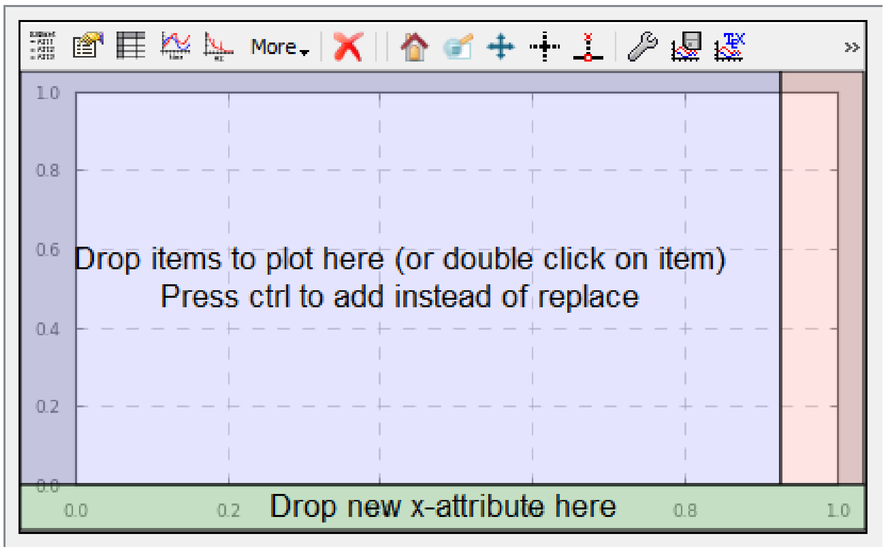

More
Drop items to plot here (or double click on item)
Press ctrl to add instead of replace
Drop new x-attribute here
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.8
1.0

**Figure 6 – Plot canvas**

The plotting view adds some extra icons to the cell toolbar

| | | |
|---|---|---|
| | Home | Resets to the original view |
| | Zoom | Zoom to rectangle |
| | Move | Pan with left mouse, zoom with right |
| | Crosshair | Show crosshair cursor |
| | Snap | Let mouse cursor snap to curve |
| | Customize | Edit curve lines and axes parameters |
| | Save | Save plot as image, see section 6.5.2 |
| | Export latex | Save plot as vector graphic and generates a latex file that includes the figure, see section 6.5.3 |

To include the dataset name in the legend text,(usefull when plotting two attributes with the same name from different datasets), see Figure 7, right click on the plot and select "Include dataset name" from the popup menu

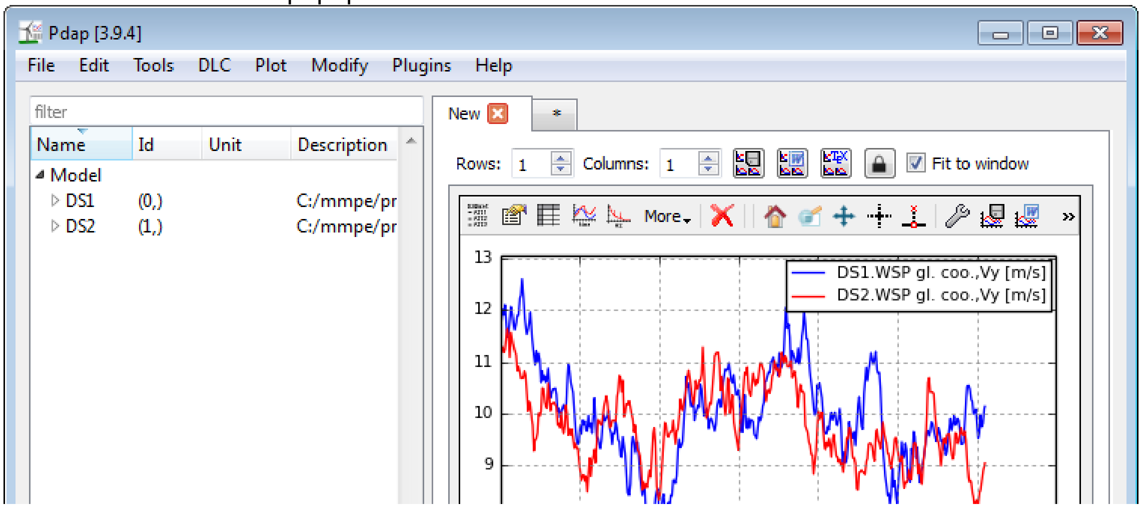

Pdap [3.9.4]
File Edit Tools DLC Plot Modify Plugins Help
filter
Name Id Unit Description
Model
DS1 (0,) C:/mmpe/pr
DS2 (1,) C:/mmpe/pr
New
Rows: 1 Columns: 1 Fit to window
More
DS1.WSP gl. coo.,Vy [m/s]
DS2.WSP gl. coo.,Vy [m/s]
13
12
11
10
9

**Figure 7 – Dataset name can be included in the legend text**

## 6.4 Modify

### 6.4.1 Manipulate

Data items can be manipulated via the Modify menu found in the main menu or in the popup menu that appears by right click on a data item, see Figure 8

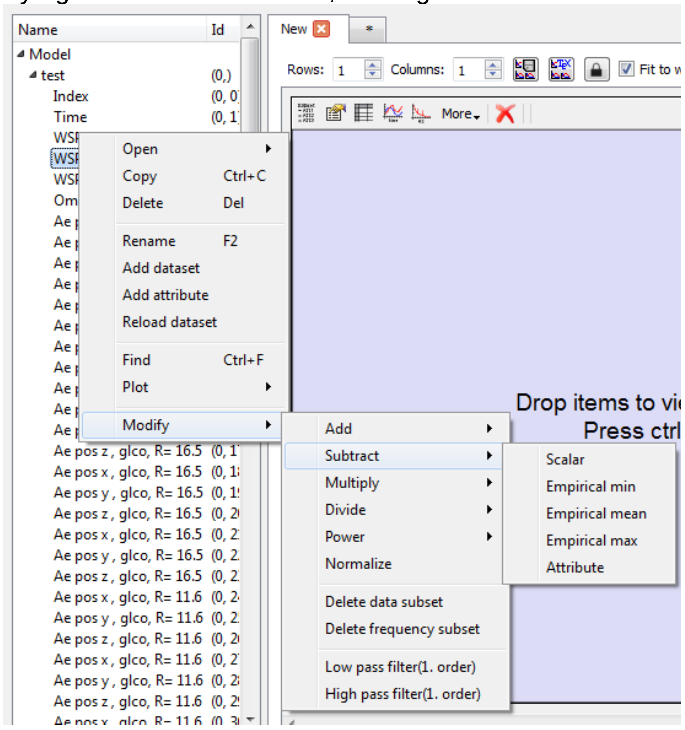


**Figure 8 – Data items can be manipulated via the Modify menu**

From this menu it is possible to add, subtract, multiply, divide or power an attribute or whole dataset by a scalar, the empirical min, mean or max, or the data of another attribute.
More over an attribute, X, can be normalized to have mean(X)=0 and std(X)=1.

**Example of manipulation**: Compute $wsp_{xy}$ from $wsp_x$ and $wsp_y$, $wsp_{xy} = \sqrt{wsp_x^2 + wsp_y^2}$:

- In order to keep original attributes copy and paste $wsp_x$ and $wsp_y$ into the dataset. The new copies are appended in the end of the attribute list
- Square $wsp_x$ and $wsp_y$:
  - Right click on the attribute, Modify – Power – scalar
  - Enter "2" and press <Ok>
- Sum squares:
  - Right click on $wsp_x$, Modify – Add – Attribute
  - Select $wsp_y$ and press <Ok>
- Compute the square root
  - Right click on $wsp_x$, Modify – Power – scalar
  - Enter "0.5" and press <Ok>
- Rename $wsp_x$ to $wsp_{xy}$
  - Right click on $wsp_x$, Rename (F2)
  - Enter name $wsp_{xy}$ and press <Ok>
- Delete temporary $wsp_y$ (squared) attribute
  - Right click on $wsp_y$, Delete (Del) and confirm

### 6.4.2 Delete data subset

A subset of the data set can be deleted e.g. the first 10 seconds or all observations with wind speed below 9 or above 11.
The tool is seen in Figure 9. Simply drag the desired attribute to the x-attribute section of the tool, use the sliders or spinbutton fields to move the limits, choose Trim(Removes $[-inf, start[$ and $]end, inf]$) or Delete(Removes $]start, end[$) and press <Delete selection>.
The tool is accessible from the Modify menu.

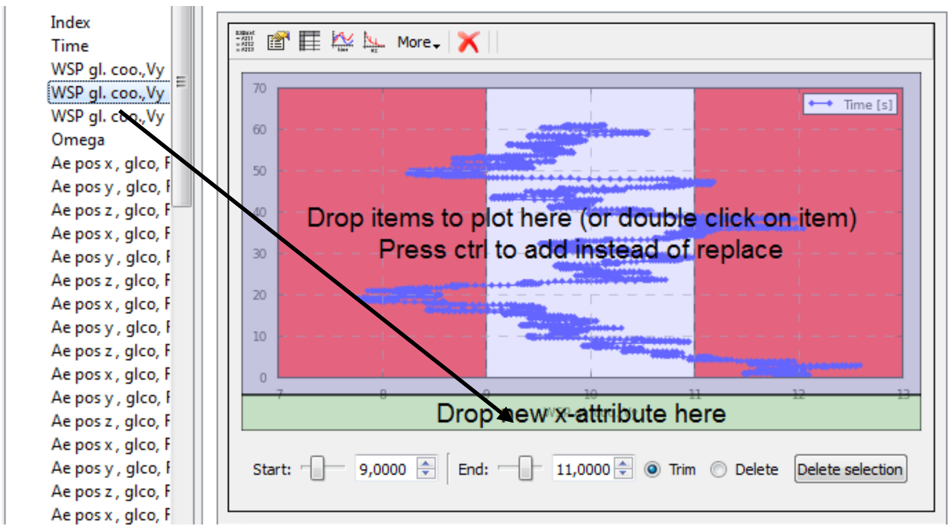


Figure 9 – Using the "Delete data subset"-tool all observations with wind speed below 9 or above 11 can be deleted

### 6.4.3 Delete frequency subset

The "Delete frequency subset" tool is similar to the "Delete data subset" tool except that it operates in frequency domain. Using this tool all frequencies between e.g. 5 Hz and 10 Hz can be suppressed, see Figure 10

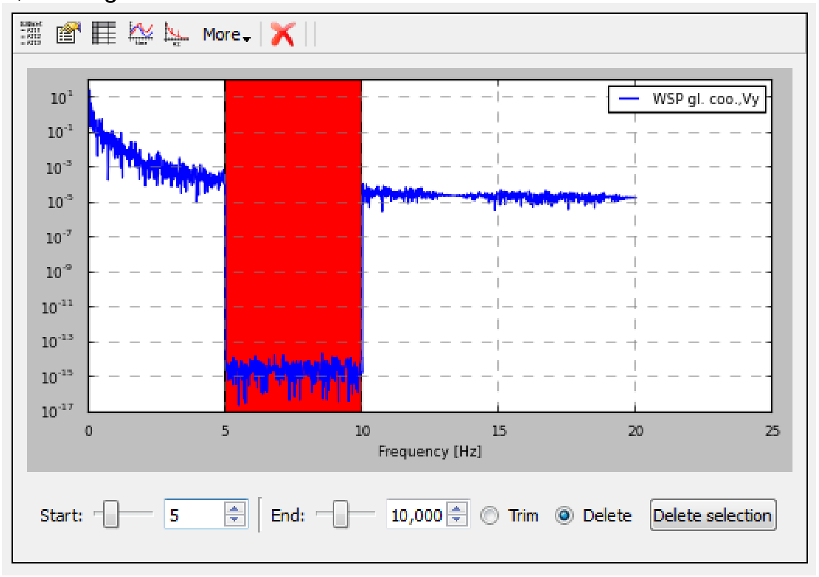


Figure 10 – Using the "Delete frequency subset"-tool all frequencies between 5 Hz and 10 Hz can be removed

### 6.4.4 First order filters

Pdap also contains a first order low and a high pass filtering tool. These tools apply smoother damping curves and a preview of the result can be seen immediately, see Figure 11.

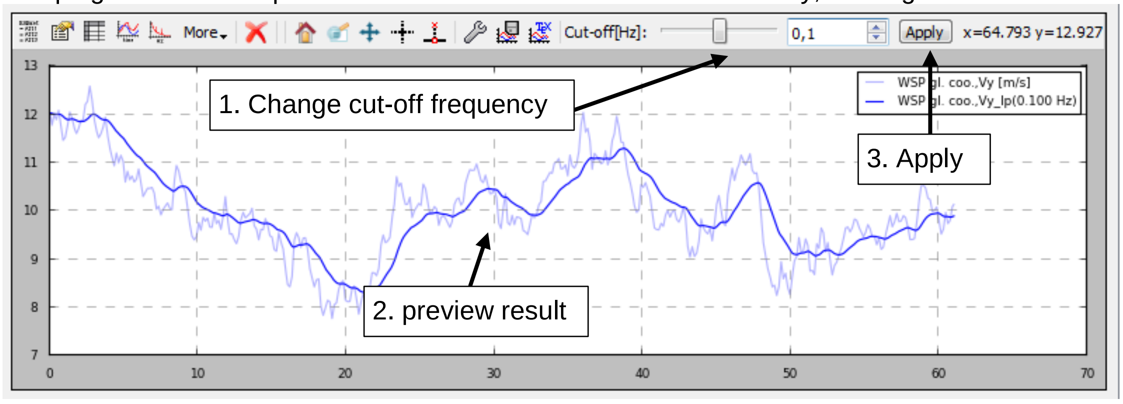


**Figure 11 – First order low pass filter tool**

## 6.5 Save and export

### 6.5.1 Save dataset

Datasets including any changes can be saved as:

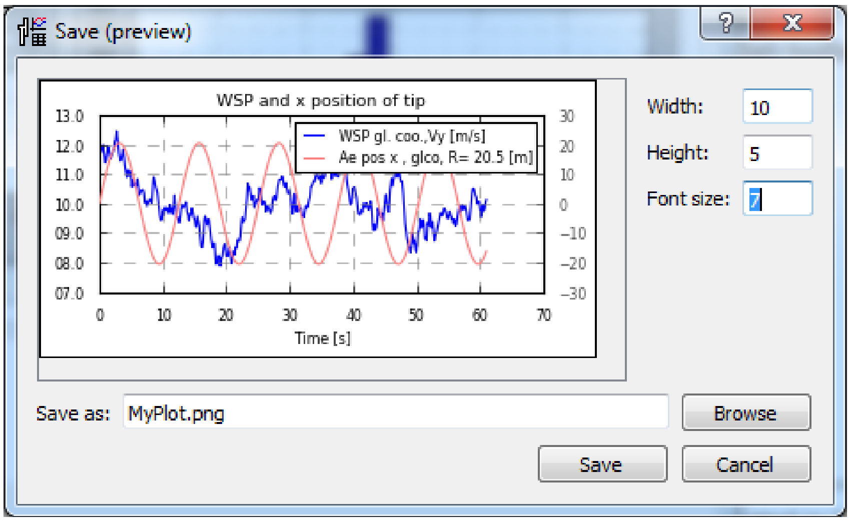
- HAWC2 (Binary or ASCII)
- CSV (Comma separated values. First line is a header containing attribute names)
- FLEX4
- DAQWin

### 6.5.2 Save as image

Pdap offers two functions that save a cell and a tab as an image respectively:

| | Save cell as image | Save tab as image |
|---|---|---|
| Icon | at cell toolbar | at tab toolbar |
| Saves | Single cell | All cells at tab |
| Supports | Cells containing plots only | All cell types |
| Size | Custom size | Size determined by window size |
| Format | Vector and raster formats | Raster formats only |

When using the cell option, a preview dialog is shown, see Figure 12. Use this dialog to change the width and height of the saved image, the font size of title, legend, labels and ticks, and the filename.



Width: 10

Height: 5

Font size: 7

Save as: MyPlot.png

**Figure 12 – Save preview dialog**

### 6.5.3 Export as LaTeX

Cells and tabs can also be exported as LaTeX using the and icons from the cell and tab toolbars respectively. Note that all cells must be plot cells.
Both options opens a settings dialog, see Figure 13.

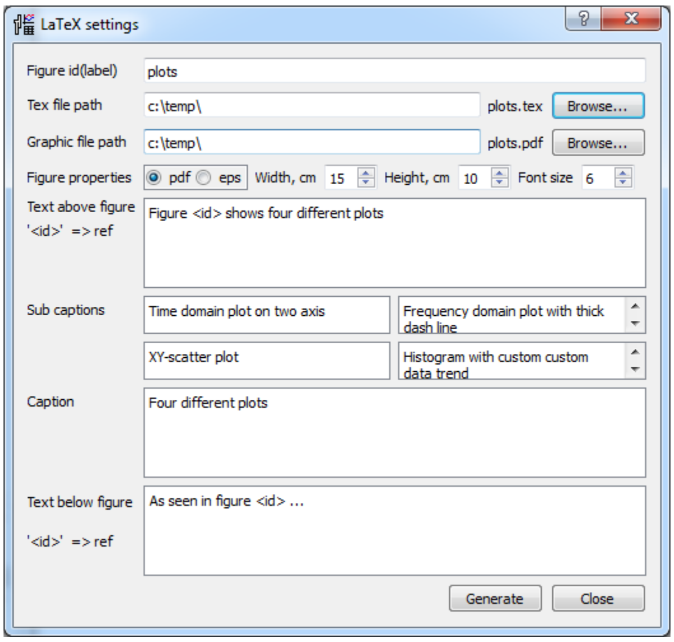

LaTeX settings

Figure id(label): plots

Tex file path: c:\temp\ plots.tex Browse...

Graphic file path: c:\temp\ plots.pdf Browse...

Figure properties: [✓ pdf] eps Width, cm 15 Height, cm 10 Font size 6

Text above figure '<id>' => ref: Figure <id> shows four different plots

Sub captions: Time domain plot on two axis | Frequency domain plot with thick dash line | XY-scatter plot | Histogram with custom custom data trend

Caption: Four different plots

Text below figure '<id>' => ref: As seen in figure <id> ...

Generate Close

**Figure 13 – LaTeX settings dialog**

Having a tab with 2x2 cells, and the settings seen in Figure 13, six files are created in the "c:\temp" folder:

| | |
|---|---|
| plots(0,0).pdf<br>plots(0,1).pdf<br>plots(1,0).pdf<br>plots(1,1).pdf | Vector graphic images of the four plots |
| plots.tex | LaTeX snippet adding the texts, images and captions |
| Example.tex | Example document that includes plot.tex |

The result of compiling Example.tex is seen in Figure 14

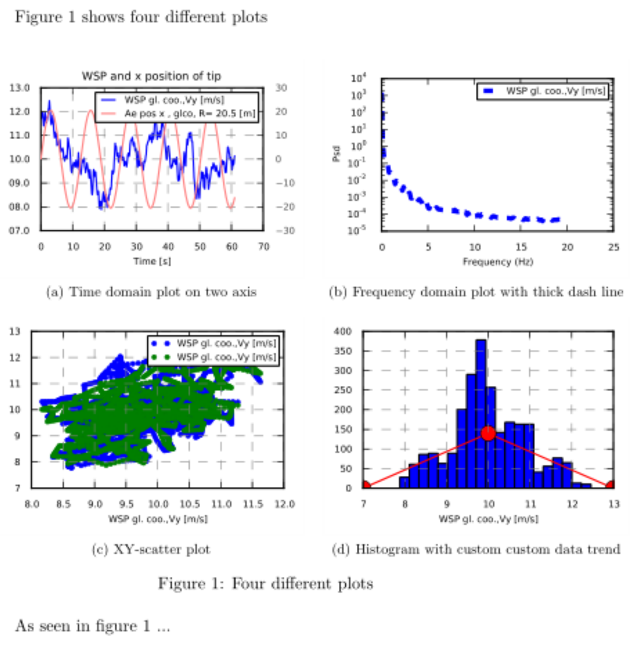


Figure 1 shows four different plots

(a) Time domain plot on two axis

(b) Frequency domain plot with thick dash line

(c) XY-scatter plot

(d) Histogram with custom custom data trend

Figure 1: Four different plots

As seen in figure 1 ...

**Figure 14 – Final pdf document**

## 6.6 Tools

The tools menu provides a few special functions

| | |
|---|---|
| Run start up script | Executes a startup script, located in "./script/startup.py" |
| Run last | Executes the command history of last session |
| Hawc2 ascii to binary | Converts a Hawc2 ascii data file to Hawc2 binary format |

# 7. Mathematical background / Implementation

## 7.1 Kaimal fit

From the plot menu, a PSD plot can be compared to a Kaimal spectrum.
Two alternative Kaimal spectrums, DS472 and IEC61400-1, are implemented. Both can be specified by the mean wind speed, U, and a length scale, L.

| DS472 | IEC61400-1 |
|---|---|
| $\frac{fL/V}{\left(1+1.5\frac{fL}{V}\right)^{5/3}}$ | $\frac{4fL/V}{\left(1+6\frac{fL}{V}\right)^{5/3}}$ |

Where

| | |
|---|---|
| f: | Frequency in Hz |
| L: | Length scale |
| V: | Wind speed (mean of 10 minutes) |

## 7.2 Fatigue

In Pdap two fatigue methods are implemented, `FatigueASTM` and `FatigueWindap`.
Both consist of a method to remove non-peak values from the signal and one to count the cycles and their sizes.

| | FatigueASTM | FatigueWindap |
|---|---|---|
| Method to remove all non-peak values of the signal | find_extremes | peak_trough |
| Method to count the cycles and their sizes. | rainflowcount | pair_range |

### 7.2.1 FatigueASTM

The implementation of the find_extremes and rainflowcount methods which are used in FatigueASTM are based on the c-implementation by Adam Nieslony [NIES].
Example:

| | signal = [-10, 0, 10, 9.7, 10, 1] |
|---|---|
| `ext_sig = find_extremes(signal)` | ext_sig = [-10, 10, 9.7, 10, 1] |
| `rfc = rainflowcount(ext_sig)` | rfc = [0.3, 0.3, 20, 9] |

### 7.2.2 FatigueWindap

FatigueWindap provides a Windap equivalent result. Before the non-peak values are removed from the signal by the `peak_trough` method, the signal is converted to 255 levels:

| | Ex: signal = [-10, 0, 10, 9.7, 10, 1] |
|---|---|
| `signal = signal – min(signal)` | signal = [0, 10, 20, 19.7, 20, 11] |
| `gain = max(signal) / 255` | gain = 0.078 |
| `signal = round(signal / gain)` | signal = [0, 128, 255, 251, 255, 140] |

The `peak_trough and pair_range` methods are implemented directly as described in [REC]. In contrast to find_extremes, the `peak_trough` method uses a threshold, `t`, such that a peak is not registered before the signal has decreased at least `t` and vice versa. In this way small cycles are ignored.
If a small threshold is used, the result of peak_trough is equal to the result of find_extremes, except for a few special cases. In the windap equivalent implementation, the threshold is 255/50 = 5.1, i.e. the example signal is reduced to [0, 255, 140].
The result of `pair_range` is assumed to be equal to the result of `rainflowcount`, as comparisons on random signals has not shown any differences except for the order of the values.
Finally the flow returned by pair_range is mapped to real amplitudes

| | |
|---|---|
| `rfc = pair_range([0,255,140])` | rfc = [255, 115] |
| `rfc = round(rfc/threshold)*threshold` | rfc = [255, 117.3] |
| `rfc = rfc*gain` | rfc = [20, 9.2] |

## 7.3 First order filters

### 7.3.1 Low-pass filter

The first order low-pass filter is an exponentially-weighted moving average filter, which is a discrete implementation of a simple RC low-pass filter.
It is implemented as

$$output_i = \alpha\ input_i + (1-\alpha) output_{i-1}$$

where *α* is a constant

$$\alpha = \frac{\Delta t}{\tau + \Delta t}$$

*Δt* is the time step between samples and τ is calculated from the cut-off frequency, *f*, in Hz:

$$\tau = \frac{1}{2\,\pi\,f}$$

### 7.3.2 High-pass filter

The first order high pass filter is a discrete implementation of a simple RC high-pass filter.
It is implemented as

$$output_i = \alpha\ output_{i-1} + \alpha(input_i - input_{i-1})$$

where *α* is a constant

$$\alpha = \frac{\tau}{\tau + \Delta t}$$

*Δt* is the time step between samples and i is calculated from the cut-off frequency, *f*, in Hz:

$$\tau = \frac{1}{2\,\pi\,f}$$

### 7.3.3 Damping curves

The damping curves for the two first order filters are seen in Figure 15. As seen the actual damping of the cut-off frequency is about 3 dB.

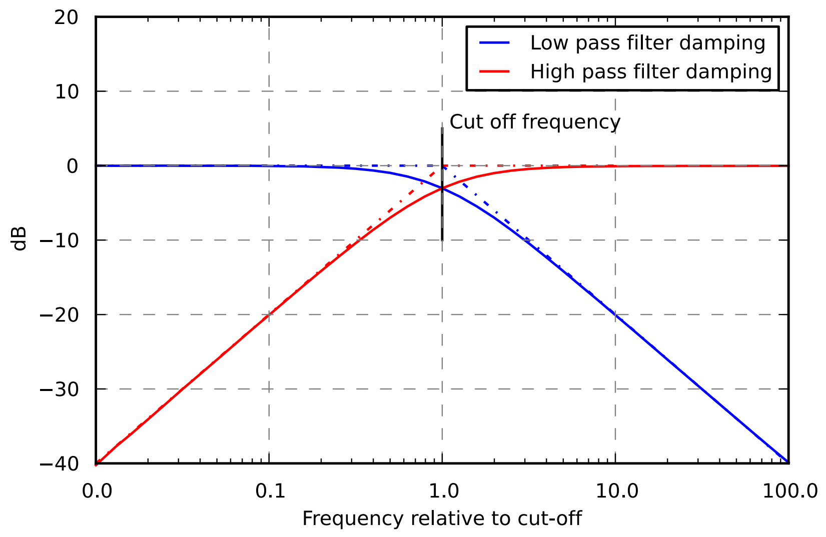
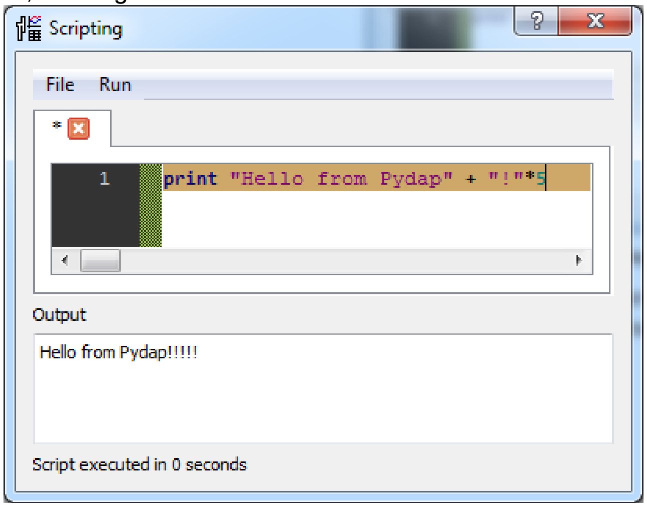


**Figure 15 – Damping curves for first order filters**

# 8. Scripting

Pdap contains a powerful python scripting environment that makes it easy to repeat operations and automate processes.
The scripting window is opened via: main menu – Tools – Scripting (F7)
The scripting window contains a menu from where scripts can be opened, saved, executed and exported as an application function (available from the main menu), a tab pane with script tabs and an output text field, see Figure 16.



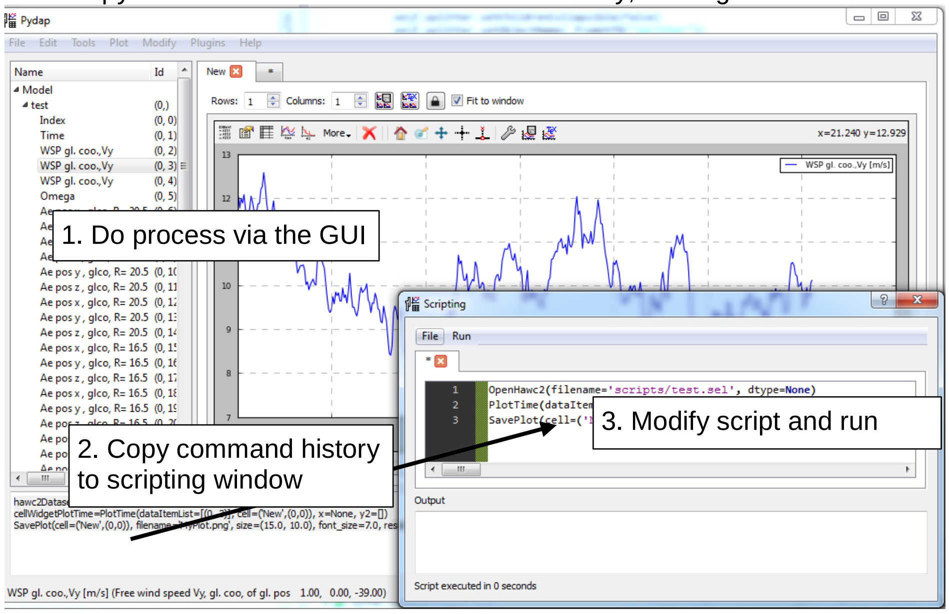
```
print "Hello from Pydap" + "!"*5
```

```
Hello from Pydap!!!!!
```

Script executed in 0 seconds

**Figure 16 – The scripting window**

The zoom level of the scripting window can be changed by pressing <ctrl> while operating the mouse wheel.
The easiest way to learn what to write in the scripting window is to do a similar process via the GUI and then copy the commands from the command history, see Figure 17.


Pydap
File Edit Tools Plot Modify Plugins Help
1. Do process via the GUI
2. Copy command history to scripting window
3. Modify script and run
Scripting
File
Run
Output

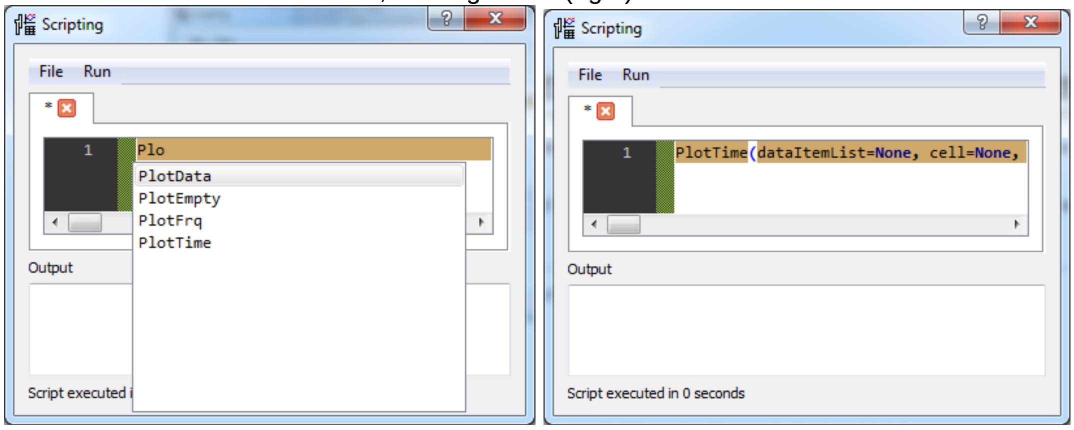
```
hawc2Datase
cellWidgetPlotTime=PlotTime(dataItemList=[(0, 3)], cell=('New',(0,0)), x=None, y2=[])
SavePlot(cell=('New',(0,0)), filename='MyPlot.png', size=(15.0, 10.0), font_size=7.0, res
```

```
OpenHawc2(filename='scripts/test.sel', dtype=None)
PlotTime(dataItem
SavePlot(cell=('N
```

Script executed in 0 seconds

**Figure 17 – Scripts can be copied from the Command history**

The scripting window supports intellisense, autocomplete and call tip insertion.
Autocomplete proposals automatically pops up after typing a few characters and pressing <Enter> inserts the selected proposal, see Figure 18 (left).
The call tip, i.e. parameter names and default values, is automatically inserted when typing "(" + <Enter> after the function name, see Figure 18 (right).



```
Plo
PlotData
PlotEmpty
PlotFrq
PlotTime
```

Script executed i

```
PlotTime(dataItemList=None, cell=None,
```

Script executed in 0 seconds

**Figure 18 – left: autocomplete list pops up after writing a few characters. Right: Call tip inserted when typing "(" + <Enter> after the function name**

The scripting environment provides access to two important objects, namely `gui` and `model` .

## 8.1 Gui

The gui object contains methods to communicate with the user and to access tabs and cells.

| | |
|---|---|
| `gui.show_message_box(msg, title="Information")` | Show info/warning/error dialog with the specified title and message |
| `gui.show_warning_box(msg, title="Warning")` | |
| `gui.show_error_box(msg, title="Error")` | |
| `gui.show_confirm_box(title, message)` | Show Yes/No dialog with specified title and message.<br>Return: Yes->True, No->False |
| `gui.get_open_filename(title, filetype_filter, file_dir=None, selected_filter=None)` | Show Open/Save dialog<br>filetype_filter: file extension, ex: "*.dit;;*.dat"<br>file_dir: default directory<br>selected_filter: default filter |
| `gui.get_save_filename(title, filetype_filter, file_dir=None, selected_filter=None):` | |
| `gui.start_wait()` | Mouse cursor set to wait cursor (i.e. hour glass or similar cursor) |
| `gui.end_wait()` | Mouse cursor set to default cursor |
| `gui.tab(tab)` | Return the specified tab.<br>tab: tab index, tab title or None(current tab returned) |
| `gui.cell(cell)` | Return the specified cell<br>cell(None): current cell<br>cell(tab): current cell at the tab returned by gui.tab(tab)<br>cell(tab,(r,c)) cell at position (r,c) at the tab returned by gui.tab(tab).<br>(cell inserted if not exists) |

## 8.2 Model

The model object contains all data items. Data items are accessed via their id(s) or names. Names and ids are shown in the data model tree

| | |
|---|---|
| **model**(dataset_id, attribute_id=None) | Return the specified data item, e.g:<br>model(0) dataset with id 0<br>model('ds1') dataset with name 'ds1'<br>model(0,3) attribute with id 3 in dataset with id 0<br>model(0,'at1') attribute with name 'at1' in dataset 0 |

### 8.2.1 Dataset

A dataset object, `ds`, holds a 2d numpy data array, where columns represent attributes and rows represent observations, and a list of attributes.

| | |
|---|---|
| **ds(**attribute_id, stop, step**)** | Return the specified attribute(s), ex:<br>ds(3) attribute with id 3<br>ds('att1') attribute with name 'att1'<br>ds(3,6) attributes with id 3,4,5<br>ds(2,7,2) attriubtes with id 2,4,6 |
| **ds[subset]** | subset: Normal indices of numpy array, ex:<br>ds[:] Whole data array<br>ds[0] First row of data array<br>ds[3:6] Row 3, 4 and 5 of data array<br>ds[2:7:2] Row 2, 4 and 6 or data array<br>ds[:,3] All rows of column 3 |
| **id** | Id of dataset |
| **name** | Name of data set |
| **description** | Description of dataset |
| **shape** | Shape of data array, i.e. (#observations, #attributes) |
| **attributes()** | List of all attributes in dataset except the auto generated index attribute |
| **sample_frq()** | Sample frequency |

### 8.2.2 Attribute

An attribute object, `att`, contains no data, but an index pointing at the corresponding column in the data array of its parent dataset.

| | |
|---|---|
| **att[subset]** | subset: Normal index of numpu array, ex:<br>att[:] Whole data column<br>att[0] First row of data column<br>att[3:6] Row 3, 4 and 5 of data column<br>att[2:7:2] Row 2, 4 and 6 or data column |
| **id** | id of attribute |
| **index** | Index of corresponding data column in parent dataset |
| **name** | Name of attribute |
| **unit** | Unit of attribute |
| **description** | Description of attribute |
| **shape** | Shape of corresponding column in data array of parent dataset |
| **sample_frq()** | Sample frequency |

## 8.3 Application functions

In Pdap functions available from the GUI are so-called "Application Functions" that behave as an interface between the user and the underlying `model`- and `gui`-objects, see Figure 19. While it is possible to invoke the underlying functions directly, it is recommended to use Application Functions whenever it is possible.

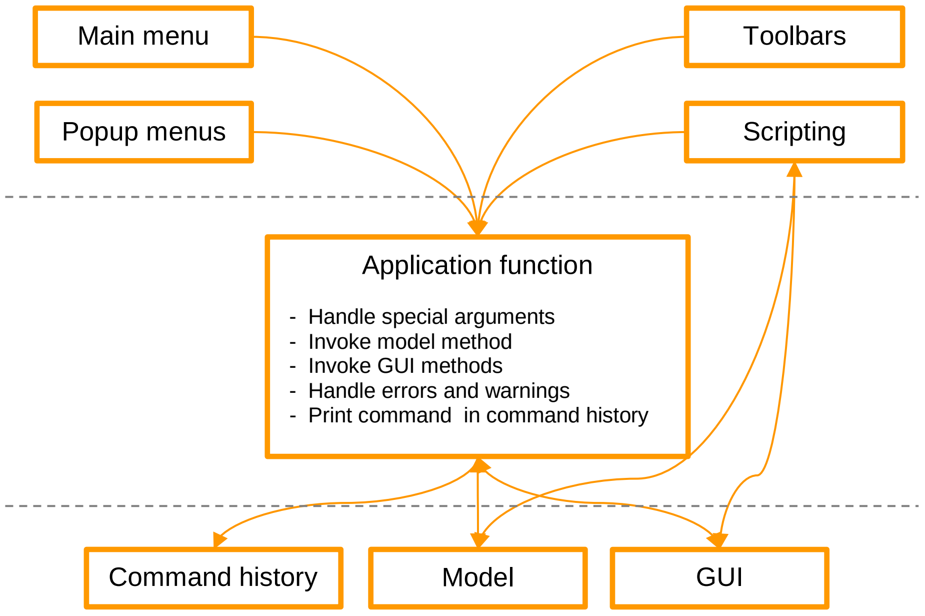


**Figure 19 – Application functions behave as a layer between the user and the underlying model and GUI.**

### 8.3.1 Example of an application function

As example consider the Application Function, `Rename`, that changes the name, unit and description of an attribute.
`Rename` takes four optional arguments, `selection`, `name`, `unit` and `description`.

`Rename` can be invoked from the main menu, from popup menus, or by pressing F2. In these cases it is called without arguments, see Figure 20.
It can also be called from a script with or without parameters

`selection` is a special argument which is automatically replaced with a `DataItemList` object instantiated from the specified value, see section 8.3.2. If the argument is not specified, it is replaced with a copy of the current data model tree selection if the function is applicable to the selection. Otherwise an error is raised.

If `name`, `unit` or `description` are not specified, `Rename` calls a function in the `gui`, that displays a dialog in which the user is requested to enter the three parameters.
Having all parameters, `Rename` invokes a function in the model that updates the `name`, `unit` and `description` of the data item referenced by the selection argument. This call causes an update of the gui.
Finally Rename prints the executed command in the command history:

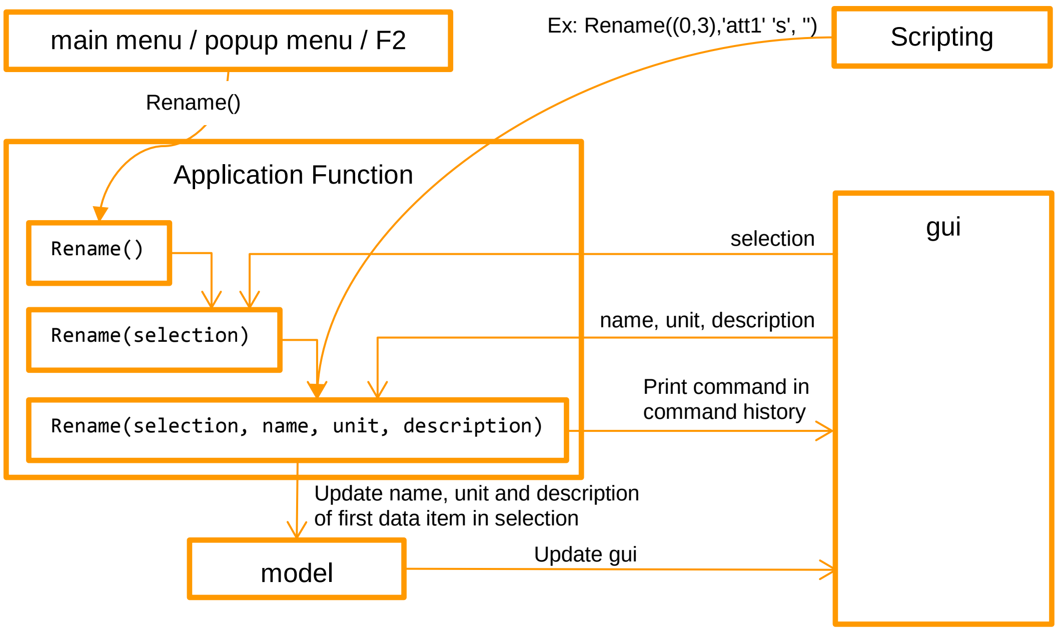

main menu / popup menu / F2
Ex: Rename((0,3),'att1' 's', '')
Scripting
Rename()
Application Function
Rename()
Rename(selection)
Rename(selection, name, unit, description)
selection
gui
name, unit, description
Print command in command history
Update name, unit and description of first data item in selection
model
Update gui

**Figure 20 – Rename, an application function**

### 8.3.2 Special arguments

There are four special arguments, which are automatically replaced if they are not specified:

| Special arguments | |
|---|---|
| `dataItemList` | The argument is replaced with a `DataItemList` object created from the specified value. The DataItemList constructor is very flexible and accepts tuples, strings, data item objects and lists of any of these, e.g.<br>(0,3) Tuple of ids<br>"(0,3)" String<br>model(0,3) DataItem object<br>[(0,3),(0,4)] List of tuples of ids<br>None (Empty DataItemList object returned) |
| `selection` | • If specified (i.e. not None): Replaced with DataItemList instantiated from specified value<br>• If not specified and function is applicable to current selection in data model tree: Replaced with copy of current selection (which is also a DataItemList)<br>• If not specified and function not applicable to current selection: An error is raised |
| `cell` | Argument replaced with the cell returned by gui.cell(<specified value>), see section 8.1 |
| `tab` | Argument replaced with the tab returned by gui.tab(<specified value>), see section 8.1 |

### 8.3.3 Open file functions

**OpenHawc2**

```
OpenHawc2(filename=None, dtype=None)
```

Open Hawc2 binary and Ascii files.

- `filename`: File name including path to file. If `None` a dialog is shown
- `dtype`: Numpy data type for data array, default is `np.float32`

Return dataset

**OpenHawc2Subset**

```
OpenHawc2Subset(filename=None,
                sensorlist=None, first_row=None, last_row=None,
                dtype=None)
```

Open subset of Hawc2 binary file.

- `filename`: File name including path to file. If `None` a dialog is shown
- `sensorlist`: List of sensors to load, e.g. [3,9] (loads sensor 3 and 9). If `None` a dialog is shown
- `first_row`: First data row to load. If `None` a dialog is shown
- `last_row`: Last data row to load. If `None` a dialog is shown
- `dtype`: Numpy data type for data array, default is `np.float32`

Return dataset

**OpenCSV**

```
OpenCSV(filename=None)
```

Open data file of comma, semicolon or whitespace separated values.

- `filename`: File name including path to file. If `None` a dialog is shown

Return dataset

**OpenFLEX4**

```
OpenFLEX4(filename=None)
```

Open FLEX4 data file

- `filename`: File name including path to file. If `None` a dialog is shown

Return dataset

**OpenDAQWin**

```
OpenCSV(filename=None, configurationfilename=None)
```

Open DAQWin data file.

- `filename`: File name including path to file. If `None` a dialog is shown
- `configurationfilename`: Filename including path to configuration file. In standard cases the configuration filename can be determined from the filename. In these cases this argument can be omitted.

Return dataset

### 8.3.4 Save file functions

**SaveHawc2Binary**

```
SaveHAWC2Binary(selection=None, filename=None)
```

Save dataset as Hawc2 binary.

- `selection: Selection containing a dataset, see section 8.3.2`
- `filename`: File name including path to file. If `None` a dialog is shown

**SaveHawc2Ascii**

```
SaveHAWC2Ascii(selection=None, filename=None)
```

Save dataset as Hawc2 ascii.

- `selection: Selection containing a dataset, see section 8.3.2`
- `filename`: File name including path to file. If `None` a dialog is shown

**SaveCSV**

```
SaveCSV(selection=None, filename=None,
        separator=",", format="%.10e", decimal_separator=".")
```

Save dataset as separated values.

- `selection: Selection containing a dataset, see section 8.3.2`
- `filename`: File name including path to file. If `None` a dialog is shown
- `separator`: String used to separate values
- `format`: format of values. For full documentation see http://docs.python.org/2/library/string.html#format-specification-mini-language
- `decimal_separator`: Decimal separator

**SaveFLEX4**

```
SaveFLEX4(selection=None, filename=None, confirm_replace_sensor_file=True)
```

Save dataset as FLEX4.

- `selection`: Selection containing a dataset, see section 8.3.2
- `filename`: File name including path to file. If `None` a dialog is shown
- `confirm_replace_sensor_file`: If True, confirmation is requested before a already existing configuration file is replaced.

**SaveFLEX4**

```
SaveDAQWin(selection=None, filename=None)
```

Save dataset as DAQWin.

- `selection`: Selection containing a dataset, see section 8.3.2
- `filename`: File name including path to file. If `None` a dialog is shown

### 8.3.5 Edit functions

**Rename**

```
Rename(selection=None, name=None, unit=None, description=None)
```

Rename a dataset or attribute

- `selection`: Selection containing a dataset or attribute, see section 8.3.2
- `name`: New name. If `None` a dialog is shown
- `unit`: New unit. Omit this argument when renaming datasets
- `description`: New description. If `None` a dialog is shown

**AddDataset**

```
AddDataset(name=None, description="", data=None)
```

Add dataset to data model

- `name`: Name of dataset. If `None` a dialog is shown
- `description`: Description of dataset
- `data`: 2D numpy array. Possible empty, e.g. `np.array([]).reshape(100,0)`

Return dataset

**AddAttribute**

`AddAttribute(selection=None, name=None, unit="-", description="", data=None)`

Add attribute to dataset

- `selection`: Selection containing a dataset, see section 8.3.2
- `name`: Attribute name. If `None` a dialog is shown
- `unit`: Attribute name
- `description`: Attribute description
- `data`: Attribute, 1D numpy array or `None`(data array padded with zeros)

Return attribute

**Concatenate**

`Concatenate(selection=None)`

Concatenate two or more datasets with the same number of attributes

- `selection`: Selection containing two or more datasets with the same number of attributes, see section 8.3.2

return concatenated dataset

**Delete**

`Delete(selection=None, confirm=True)`

Delete one or more data items

- `selection`: Selection containing one or more data items, see section 8.3.2
- `confirm`: If `True`, a confirmation is requested before deleting data items

**Reset**

`SetLineProperties(dataItemList, cell, **properties)`

Override the line properties of data items in `dataItemList` when plotted in `cell`

- `dataItemList`: See `PlotTime` section 0
- `cell`: See `PlotTime` section 0
- `**properties`: Style properties, e.g. "`linewidth=4, label='`MyLine`'`", see http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot for additional options

**ReloadDataset**

**ReloadDataset**(selection=None, confirm=True)

Reload a dataset

- selection: Selection containing a dataset or attribute(parent dataset is reloaded), see section 8.3.2
- confirm: If True, a confirmation is requested before reloading a dataset that has been modified

## 8.3.6 Plotting functions

**Plot**

**Plot**(dataItemList=None, cell=None, x=None, y2=[], **properties)
Generates a standard time domain graph plot.

- dataItemList: data items to plot, see section 8.3.2
- cell: cell to plot in, see section 8.3.2
- x: Attribute for x-axis. Any value accepted by the DataItemList constructor can be used, see section 8.3.2. If None, the basis-attribute is used, i.e. the time attribute if it exists and otherwise the autogenerated index attribute.
- y2: Attributes to plot on the secondary axis in the right side of the plot. Any value accepted by the DataItemList constructor can be used, see section 8.3.2.
- properties: E.g. "title='MyPlot', xlabel='time [s]'".
  The following keys are implemented:

| | |
|---|---|
| • title | Plot title, e.g. "title='MyPlot'" |
| • xlabel | Label of x-axis, e.g. "xlabel='Time [s]'" |
| • ylabel | See xlabel |
| • xlim | Limits of x-axis, e.g. "xlim=(0,10)" |
| • ylim | See xlim |
| • xscale | Scale of x-axis. Must be "xscale='linear'"(default) or "xscale='log'" |
| • yscale | See xscale |
| • gridlines | Show/hide gridlines, e.g. "gridlines=True" |
| • xformat | Format of ticks on x-axis, e.g. "xformat='%03d'" |
| • yformat | See xformat |
| • y2format | See xformat |

Curve and marker properties can be modified by SetLineProperties (section 0) and custom data can be appended by PlotData (section 0)

### PlotPSD

```
PlotPSD(dataItemList=None, cell=None, no_segments=None, **properties)
```

Generates a frequency domain graph plot.

- `no_segments`: Splits time signal into `no_segments` segments and apply the FFT-transformation to each segment. Finally the average PSD is plotted

For description of remaining parameters see `PlotTime` section 0
Curve and marker properties can be modified by `SetLineProperties` (section 0) and custom data can be appended by `PlotData` (section 0)

### PlotScatter

```
PlotScatter(dataItemList=None, cell=None, x=None, **properties)
```

Generates a scatter plot domain graph plot.
For description of parameters see `PlotTime` section 0
Curve and marker properties can be modified by `SetLineProperties` (section 0) and custom data can be appended by `PlotData` (section 0)

### KaimalFit

```
KaimalFit(dataItemList=None, cell=None,
          fit_args=("Kaimal(IEC 61400-1)", 10, 150),
          **properties)
```

Generates a PSDxf plot together with a Kaimal spectrum, see section 7.1.

- `fit_args`: Tuple containing <spectrum type>, <wind speed> and <length scale>.
  - spectrum type: `"Kaimail(IEC 61400-1)"` or `"Kaimal(DS472)"`.
  - wind speed: Mean wind speed in m/s
  - length scale: Length scale in m

For description of remaining parameters see `PlotTime` section 0
Curve and marker properties can be modified by `SetLineProperties` (section 0) and custom data can be appended by `PlotData` (section 0)

### Histogram

```
Histogram(dataItemList=None, cell=None, no_bins=46)
```

Generates a histogram

- no_bins: Number of bins in histogram

For description of remaining parameters see `PlotTime` section 0
Custom data can be appended by `PlotData` (section 0)

## Fatigue

```
Fatigue(dataItemList=None, cell=None, no_bins=43,
        fatigue_args=('Windap',0,[3,4,6,8,10,12]))
```

Generates a fatigue histogram with equivalent loads shown in a text box.

- `no_bins`: Number of bins in histogram
- `fatigue_args`: Tuple containing <method>, <equivalent number> and < Wöhler exponents>.
  - method: "`Windap`" or "`ASTM`". See section 7.2
  - equivalent number: if 0, the duration in seconds of the time series is used
  - Wöhler exponents: list of wöhler exponents to be shown

For description of remaining parameters see `PlotTime` section 0
Custom data can be appended by `PlotData` (section 0)

## PlotData

```
PlotData(cell=None, *args, **kwargs)
```

Plots curve of custom data to plot

- `cell`: See `PlotTime` section 0
- `*args`: ([x, ] y [, style]).
  - x: list, tuple or numpy array
  - y: list, tuple or numpy array
  - `style`: "<line><marker><color>", e.g. "-or" for at red line with circle markers see http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot
- `**kwargs`: Additional style properties, e.g. "`linewidth=4, label='`MyLine`'`", see http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot for additional options

## SetLineProperties

```
SetLineProperties(dataItemList, cell, **properties)
```

Override the line properties of data items in `dataItemList` when plotted in `cell`

- `dataItemList`: See `PlotTime` section 0
- `cell`: See `PlotTime` section 0
- `**properties`: Style properties, e.g. "`linewidth=4, label='`MyLine`'`", see http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot for additional options

### Tools function

**Hawc2Ascii2Bin**

```
Hawc2Ascii2Bin(ascii_filename=None, bin_filename=None)
```

Convert a Hawc2 ascii file to Hawc2 binary format

- `ascii_filename`: Filename including path of ascii .sel-file
- `bin_filename`: Filename for binary file. If `None` the filename will be "<ascii basename>_bin.sel"

## 8.3.7 Modify functions

**Manipulate**

```
Model(3,4)[:] = model(3,4)*0.95+0.5
```

**Normalize**

```
Normalize(selection=None)
```

Normalize an attribute or all attributes of a dataset to have mean=0 and std=1

- `selection`: Selection containing a dataset or attribute, see section 8.3.2

**DeleteDatasubset**

```
DeleteDataSubset(selection=None, cell=None,
                 value_attribute=None, from_value=None, to_value=None,
                 confirm=True)
```

Deletes a subset (i.e. some observations) of a dataset, e.g. the first 10 seconds or observations with wind speed below, between or above some limits

- `selection`: Selection containing a dataset, see section 8.3.2
- `cell`: Cell in which the DeleteDataSubsetWidget is shown if `from_value` and `to_value` is both `None`
- `value_attribute`: Attribute containing values to filter on. Any value accepted by the DataItemList constructor can be used, see section 8.3.2
- `from_value`: Lower limit. All observations where the values of `value_attribute` are above this value will be deleted. If None, all values up to `to_value` will be deleted
- `to_value`: Upper limit. All observations where the values of `value_attribute` are below this value will be deleted. If None, all values down to from_`value` will be deleted
- `confirm`: If `True`, a confirmation is requested before deleting subsets

**DeleteFrequencySubset**

```
DeleteFrequencySubset(selection=None, cell=None,
                      from_value=None, to_value=None,
                      confirm=True)
```

Deletes a subset of frequencies from an attribute

- selection: Selection containing an attribute, see section 8.3.2
- cell: Cell in which the DeleteDataSubsetWidget is shown if from_value and to_value is both None
- from_value: Lower limit. All frequencies above this value will be deleted. If None, all frequencies from 0 to to_value will be deleted
- to_value: Upper limit. All frequencies below this value will be deleted. If None, all frequencies above from_value will be deleted
- confirm: If True, a confirmation is requested before deleting frequencies

**FirstOrderLowPassFilter**

```
FirstOrderLowPassFilter(selection=None, cell=None,
                        cut_off=None, confirm=True)
```

Applies a first order low pass filter to an attribute

- selection: Selection containing an attribute, see section 8.3.2
- cell: Cell in which the FilterWidget is shown if cut_off is None
- cut_off: Cut off frequency in Hz
- confirm: If True, a confirmation is requested before applying filter

**FirstOrderHighPassFilter**

```
FirstOrderHighPassFilter(selection=None, cell=None,
                         cut_off=None, confirm=True)
```

Applies a first order high pass filter to an attribute

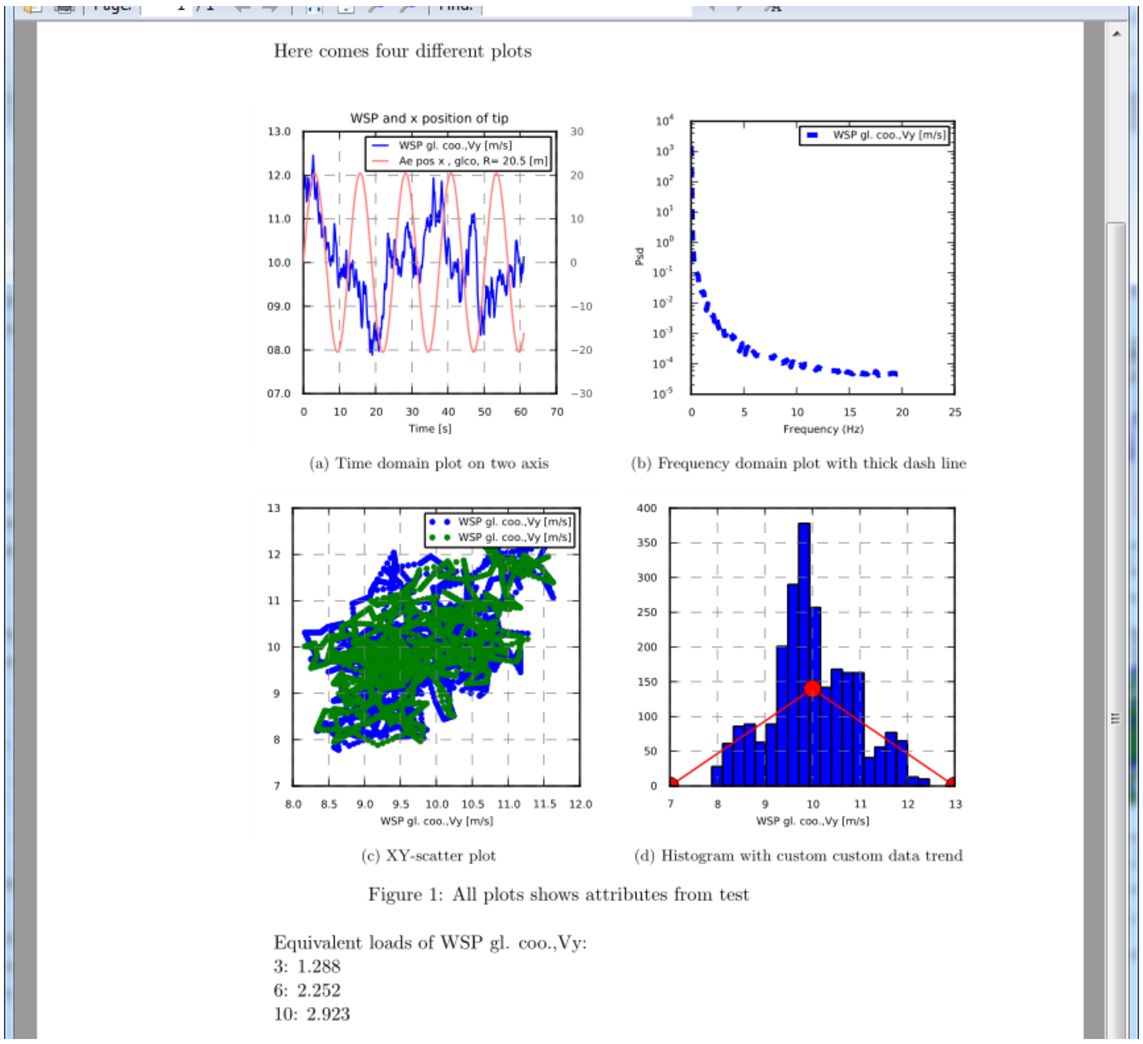
- selection: Selection containing an attribute, see section 8.3.2
- cell: Cell in which the FilterWidget is shown if cut_off is None
- cut_off: Cut off frequency in Hz
- confirm: If True, a confirmation is requested before applying filter

## 8.4 Example script 1

The following script demonstrates how to

- Reset
- Load dataset
- Access dataset, attributes and observations
- Manipulate attribute
- Make different plots
- Export as Latex

```
#Reset. Closes all tabs and removes datasets opened in previous executions
Reset(confirm=False)

# open dataset
hawc2Dataset=OpenHawc2(filename='scripts/test.sel', dtype=None)

"""
dataset now accessible as:
- ds = hawc2Dataset   #
- ds = model(0)       # if no dataset has been loaded before.
- ds = model('test')  # if no other datasets with name 'test' exists in model
"""
ds = hawc2Dataset

"""
Attributes now accessible as:
- att = ds('Time')
- att = ds(1)

When passed as argument to application function the following can be used too:
att_arg = [(0,1)]          # Option printed in command history
att_arg = (0,1)
att_arg = (0,'time')
att_arg = ('test',1)
att_arg = ('test','time')
"""
att = ds(3)

# Print max value of att
print (np.max(att[:]))


# Calibrate attribute
att[:] = att[:]*0.95+0.5

# Define a function to make some plots and save as latex
def plot_and_save(ds, tab):

    # Plot wsp on y-axis and tip_pos_x on y2-axis
    # Set title and change format of ticks on y-axis
    Plot(dataItemList=ds(3), cell=(tab, (0,0)), y2=[ds(6)],
             title="WSP and x position of tip", yformat="%04.1f")

    # Plot psd of wsp (average of 8 segments) without gridlines
    # Set line properties to thick dashes
    PlotPSD(dataItemList=ds(3), cell=(tab,(0,1)), no_segments=8, gridlines=False)
    SetLineProperties(dataItemList=ds(3), cell=(tab, (0,1)),
                      linewidth=3, linestyle="--")
    # Make scatterplot of wsp1 and wsp2 vs wsp3
    PlotScatter(dataItemList=[(0,2), (0, 3)], cell=(tab,(1,0)), x=[(0, 4)])

    # Plot 20 bins histogram of wsp
    # Append custom data "trend"
    Histogram(dataItemList=ds(3), cell=(tab,(1,1)), no_bins=20)
    PlotData((tab,(1,1)), [7,10,13],[0,140,0],'r-o', markersize=10)

    # Calculate equivalent loads
    m=[3, 4, 6, 8, 10, 12]
    neq = float(ds.shape[0]+1)/ds.sample_frq
    eq_loads = eq_load(ds(3)[:], m=m, neq=neq)[0]

    # Generate string
    eq_load_str = "\n".join(["%d: %.3f" % (m, eq) for m, eq in zip(m, eq_loads)])

    Tab2Tex(tab=tab,
           label=tab, tex_path='c:/temp/', graphic_path='c:/temp/',
             text_above='Here comes four different plots',
             sub_captions=['Time domain plot on two axis',
                           'Frequency domain plot with thick dash line',
                                'XY-scatter plot',
                                'Histogram with custom custom data trend'],
             caption='All plots shows attributes from %s' % ds.name,
             text_below='Equivalent loads of %s:\n%s' % (ds(3).name, eq_load_str),
             size=(15, 15), font_size=6, graphic_format='pdf')
plot_and_save(ds, "plots")
```
The exported Latex snipped is shown in Figure 21



Here comes four different plots

(a) Time domain plot on two axis

(b) Frequency domain plot with thick dash line

(c) XY-scatter plot

(d) Histogram with custom custom data trend

Figure 1: All plots shows attributes from test

Equivalent loads of WSP gl. coo.,Vy:
3: 1.288
6: 2.252
10: 2.923

**Figure 21 – LaTeX document generated by script**

## 8.5 Example script 2

The `plot_and_save` function can also be used within a loop. The following script opens the dataset twice and applies the function to the first and second 30 seconds subset.

```
for start, end in [(0,30), (30,60)]:
    ds = hawc2Dataset=OpenHawc2(filename='scripts/test.sel')

    #Extract first or second 30 sec, i.e. delete (0 to start) and (end to max)
    DeleteDataSubset(selection=ds,
                     value_attribute=ds('Time'), to_value=start, confirm=False)
    DeleteDataSubset(selection=ds,
                     value_attribute=ds('Time'), from_value=end, confirm=False)

    #Rename dataset
    Rename(selection=ds,
           name="MyDataset_%d_%d" % (start, end),
           description="30 seconds of %s" % ds.name)

    plot_and_save(ds, "plots_%d_%d" % (start, end))
```

# References

| | |
|---|---|
| [NIES] | Adam Nieslony: Rainflow Counting Algorithm, http://www.mathworks.se/matlabcentral/fileexchange/3026-rainflow-counting-algorithm&watching=3026 |
| [REC] | Recommended Practices for Wind Turbine Testing - 3. Fatigue Loads, 2. edition 1990, Appendix A |

**Danmarks Tekniske Universitet**
DTU Vindenergi
Nils Koppels Allé
Bygning 403
2800 Kgs. Lyngby
Telephone  45 25 25 25

info@vindenergi.dtu.dk
www.vindenergi.dtu.dk