PlotPro Manual

Christoffer Hammar Skovgaard Møller

chsm@di.ku.dk

Contents

Manual **1**

1 Savecard **1**
1.1 Supported savecard keywords 2

2 Routecard **2**
2.1 Supported routecard keywords 2

3 Titlecard **3**
3.1 Supported titlecard keywords 4

4 Data Formatting **6**

5 Examples **8**
5.1 savecard, scatter, linear split fit, uncertainty, labels 8
5.2 savecard, scatter, exp split fit, labels, fontsizes, colours 9
5.3 Deviation bar plot: savecard, deviation (with scatter), labels, colours, model, multiple data columns and axes, legend location, legendbox, devpattern 9
5.4 Subplots: scatter, linear line fit, ref, subplot, labels, colours 11
5.5 subplotframe 13
5.6 bar, labels, precalculated deviations, devpattern, rot, gridlines, tickalignment 13

Using the program **15**

6 Running the program **15**
6.1 Running on Sunray 15
6.2 Running locally 15

7 Updating Python packages **15**

8 My figures are ugly - help **16**
8.1 Legend 16
8.2 Tick marks 16

Program details 18

9 Further plans **19**

Manual

Introduction

This document serves as a set of guidelines on how to format the input file for the Python plotting program PlotPro.
The input file is structured similarly to a Gaussian input file in regards to use of special characters to denote keywords and such. The input file is structures using **blocks**.
The first block, called the **savecard**, contains information on how to save the resulting plot(s). If none is given, no file will be saved by the script. Further explanation in Section 1.
The second block, called the **routecard**, contains keywords used by the program to generate the desired plot including whether or not to make the necessary data treatment beforehand. Further details in Section 2.
The third block, called the **titlecard**, contains input such as plot title, axis labels, and legends as well as other cosmetic details of the plot. Further details in Section 3.
Now comes the data itself. Always separate data using white space (e.g. tabulations or spaces). Comma-separated values not supported (yet). Further details in Section 4.
The general framework of an input file:

```
% savecard keywords (optional)
# routecard keywords
$ titlecard keywords

data1    data2    data3
```

PlotPro also generates an output file named "inputfile_output.txt", where "inputfile" is the name of the input file. In this output file, data such as fitted parameters and registered keywords are written. This way, you, the user, can see what has been done in the program.

1 Savecard

The savecard is initiated using the % symbol and is used to specify the desired name of the saved file. The % symbol must be followed by a white space. For example:

```
% png=test_plot.png pdf=test_plot.pdf
```

This savecard asks for the figure to be saved as both a .png and a .pdf file, both with the filename "test_plot".
The savecard is **optional**, so if no savecard is provided in the input file, no figures will be saved. The figures can still be manually saved by clicking the save-icon on the opened figure.
The savecard can be more than one line, meaning that the two following examples are read alike:

```
% png=test_plot.png pdf=test_plot.pdf
```

```
% png=test_plot.png
% pdf=test_plot.pdf
```

1.1 Supported savecard keywords

The following formats are supported:

eps, pdf, pgf, png, ps, raw, rgba, svg, svgz

2 Routecard

The routecard provides PlotPro with the keywords necessary to generate the desired plots and is initiated using the # symbol followed by a white space. If more than one keyword is given, these should be separated using white space. For example:

```
# Plot=scatter fit=linear fit_plot=split
```

This routecard requests that the provided data should be plotted as a scatter plot (a.k.a. dot plot) and also requests that a linear model should be fitted the data.
The `fit_plot = split` keyword requests that the uncertainty on the fitted slope should be shown in the plot, see figure 1. The routecard can be more than one line, meaning that the two following examples are read alike:

```
# Plot=scatter fit=linear fit_plot=split
```

```
# Plot=scatter
# fit=linear
# fit_plot=split
```

All keywords are case-insensitive and a single space is allowed on both sides of the "=" sign.

2.1 Supported routecard keywords

The following keywords are supported by PlotPro:

Key	Value	Explanation
plot	scatter	scatter (dot) plot
	bar	bar plot
	dev	grouped bar or scatter plots with given deviations
fit	linear	perform linear fit
	exp	perform exponential fit
fit_plot	line	plot only fitted line
	split	plot fitted line and semi-transparent area around line signifying fitting uncertainties (see figures 1 and 2)
deviation	mae	Mean Absolute Error between e.g. a set of theoretical data and experimental data
	me	Mean Error between e.g. a set of theoretical data and experimental data

Key	Value	Explanation
	std	Standard Deviation
	maxerr	Maximum Deviation
	scatter	plots individual deviation points as scatter plot on top of bar diagram
	ref	Specifies which data column (sec. 4) is to be treated as the reference.
	type	Specifies whether the deviations should be fractional or relative. Note that if `type=frac` is used, PlotPro will remove data series, where the reference value is too close to 0 (threshold: 10^{-10}).
subplot	sticky / loose	Exclude / include whitespace between subplots in the subplot structure (see figures 4 and 5)
ref	n	The column of input data to be treated as the reference data. This column index follows 1-indexing, meaning that the first column is column 1.
uncertainty	n	The uncertainty on the y-axis data values. If the user knows no uncertainty, and has no educated guess, this keyword ought not be used.

If a routecard keyword is unspecified in the input file, it will revert to the following default value:

Key	Default
plot	scatter
fit	*blank*
fit_plot	line
deviation	*blank*
subplot	*blank*
ref	1
uncertainty	1.0

3 Titlecard

The formatting of the titlecard is similar to the formatting of the routecard. Similarly, there **must be** a white space between the initiating $ symbol and the titlecard keyword.

```
$ xaxis="test of label" yaxis="test of label" title="Test title"
```

This titlecard is equivalent to the split version:

```
$ xaxis="test of label" yaxis="test of label"
$ title="Test title"
```

Similarly, all lines starting with "$" are read as titlecards.
The text within the quotation marks are read as-is, meaning that PlotPro doesn't convert to Title Case, CamelCase, lower case or similar. Note that simple LaTeX-code can be used to input mathematical notation, Greek symbols and similar, see figure 2.
Typing the titlecard keywords in lower or upper case as well as a combination thereof makes no difference, meaning that the following examples all are read alike:

```
$ title="Test title"
$ Title="Test title"
$ TITLE="Test title"
```

A single space on either side of "=" is fine, meaning that

```
$ title="Test title"
$ title ="Test title"
$ title= "Test title"
$ title = "Test title"
```

are all equal.

3.1 Supported titlecard keywords

If a titlecard keyword is left unspecified, it reverts to the following corresponding default:

Key	Default value
xaxis	*blank*
yaxis	*blank*
title	*blank*
legend	*blank*
model	"Model 1;Model 2;Model 3"
print_fit	(0.05, 0.9)
yrange	None
rot	0
axisfontsize	18
titlefontsize	18
legendfontsize	18
tickmarksize	18
textfontsize	16
pointsize	20
linewidth	1.0
outdec	20
nbarsep	2
datacolour	"k"
fitcolour	"r"
devcolour	"mae:b;me:r;std:g;maxerr:c"
devpointcol	"w"
show_legend	"on"
legendbox	"on"
legloc	"upper right"
legcolumns	1
tickalignment	"center"
devpattern	"off"
subplotframe	"grid"
figtype	"rectangle"

Key	Default value
plotall	"off"
diagonal	"off"
gridlines	"off"
tight	"on"

The **xaxis**, **yaxis**, **title**, **legend**, and **model** keywords all take a string as input and this string is being read as written in the input (meaning that letters will be capitalised similar to the input). Simple LaTeX-code will be interpreted and printed as in LaTeX.

The **print_fit** keyword takes either "on", "off" or a coordinate tuple (e.g. "(0.05, 0.9)") of x and y coordinates. If `print_fit="off"` is used, the fitted function will not be printed on the figure. If `print_fit="on"` is used, the fitted function will be printed in the default location.

yrange allows the user to define the range of the y-axes. If used in conjunction with `deviation=scatter` - ie. for figures similar to fig. 3 - the accepted structure of `yrange` is `yrange="((y0, y1),(y2, y3))"`, where `(y0, y1)` represent the range of the left y-axis and `(y2, y3)` represent the range of the right y-axis. If used on a figure with a single y-axis, only `(y0, y1)` will be read.

The **rot** keyword allows for specification of how many degrees the (axis) text should be rotated. The default is 0, as in most cases it is not necessary to rotate (axis) text.

All the ***size** and **linewidth** keywords take integers (or floats) as input and are used to set the fontsize, tickmarksize, pointsize or linewidth of the given text, tick marks on the figure, scatter plot points or the width of the plotted line.

outdec takes integers, and controls how many decimals the results in the output file are to be given with.

nbarsep takes integers (both positive and negative), and controls how much white space should be put between groups of bars in deviation plots and bar plots. The standard is 2, meaning that the width corresponding to 2 bars is used for the white space. Note that bar groups can be placed closer than that by using e.g. `nbarsep=1`.

All the ***colour** keywords can take either one-letter abbreviations, full colour names or custom tuples of RGB-values (e.g. red can be achieved either by writing "r", "red", or "(1,0,0)". For more information, visit the Matplotlib colour documentation[1]).
devcolour requires slightly more than this - each argument is of the form "bartype:colour", and arguments are separated by a ";". The default input of "mae:b;me:r;std:g;maxerr:c" is therefore read as follows:
"The MAE bar in the deviation bar plot should be coloured 'b' (blue), the ME bar should be coloured 'r' (red), the STD bar should be coloured 'g' (green), and the MaxErr bar should be coloured 'c' (cyan)."

The **show_legend** and **legendbox** keywords take either "on" or "off" as input and are used to determine whether or not the legend should be put in the figure or whether or not the frame (or box) of the legend should be included (see fig. 3).

[1]URL: https://matplotlib.org/2.0.0/examples/color/named_colors.html

The **legloc** keyword can take input analogous to the *colour keywords. The location of the legend can be specified using a number shorthand, full text specification, or a tuple of x,y-coordinates in which to put the bottom left corner of the box. For instance, the box can be put in the lower left corner using both the input of "3", "lower left", and "(0,0)". Note here, that the quotation marks around "3" are unnecessary, but are required for the other two. For more information on legend location input, visit the Matplotlib legend handler[2].

legcolumns determines how many columns the legend is presented with, ie. are all legend entries printed in a single column or spread over 3 columns? This keyword therefore takes an integer as input (floats are converted to integers).

The **tickalignment** keyword takes 3 different inputs: "left", "center", and "right". These alignment inputs determine whether the x axis ticks should be aligned such that the tickmark is above the left, center or right side of the label, see fig. 7.

The **devpattern** keyword can take "on" and "off" or a ";"-separated string of patterns (hatches). If "off" is used, the bars will not feature any patterns (hatches), while if "on" is used, up to 4 different standard patterns (hatches) will be used: lines ("-"), grids ("+"), diagonal lines ("/"), tiles ("x"). The following input will be recognised as a pattern (hatch):
`"/", "\", "|", "-", "+", "x", "o", "O", ".", "*"`

The **subplotframe** keyword is ideally used in conjunction to the "subplot" keyword from the routecard. This titlecard is used to determine whether the requested subplot should be formatted in a line (ie. with nrows=1) or as a grid (ie. with nrows>1).
NOTE: my personal recommendation is to not use the `subplotframe="line"` on data sets which yield more than 4 subplots, as the resulting size of the subplots may be too small for preference.

The **figtype** keyword allows for specification of whether the individual figures should be squares or rectangles. Note that if this will not affect the the `subplot` routecard keyword.

The **plotall** keyword is used to plot all data series in the same figure. PlotPro will then return individual figures for each of the data series and a figure with all series in one figure. The keyword only takes "on" and "off" as input with the default being "off"

The **diagonal** keyword is used to plot a diagonal line, and is thought to be in conjunction with the routecard keyword `plot=scatter`. Note that this diagonal line is plotted from bottom left to top right, ie. with a slope of +1 and a y-intercept of 0.

The **gridlines** keyword is used to toggle which, if any, gridlines should be shown in the figure. The keyword takes "off", "x", "y", and "both" as input, where "x" means that only the gridlines passing through the x-axis are shown and similarly for "y". "both" means that the full grid is shown.

The **tight** keyword toggles the function `plt.tight_layout()`, which usually pretties up the layout of the figure by removing white space around the figure, but occasionally it messes up the layout instead. It is "on" as default, but can also be turned off by using "off" as input.

[2]URL: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html

4 Data Formatting

The data given in the input file may only start with a number or a "-". If different columns (lists) of data are provided, these are to be tabulated next to one another while separated by a whitespace (such as a space or a tabulation). In the following example, the data will be read as 3 lists of data, each containing 4 entries:

```
0       −4      28.0
1.4     5.9     −9.34
−2      6.3     10
−3.3    7       11.2
```

Note that integer and float input both are supported for both positive and negative numbers. Therefore, all real numbers should be read correctly. Testing has not been done to investigate whether or not complex numbers or irrational numbers such as π or e are supported, so don't expect them to.
Using the `ref` keyword in the routecard allows for specification of which input column is to be treated as the reference values (and thus the x-values in a scatter plot, or the reference to which the other data is compared in the case of a deviation plot).

5 Examples

5.1 savecard, scatter, linear split fit, uncertainty, labels

Input file:

```
% png=test_plot.png
% pdf=test_plot.pdf
# Plot=scatter
# fit=linear fit_plot=split
# uncertainty=20.0
$ xaxis="test of label" yaxis="test of label"
$ title="Test title"
$ legloc="lower right"
$ print_fit="on"
$ gridlines="both"

1       2.329108705
2       3.642707826
3       6.136698336
...     ...
243     474.3053311
244     580.8688169
245     542.6863531
```

(data truncated for brevity)
Output figure:

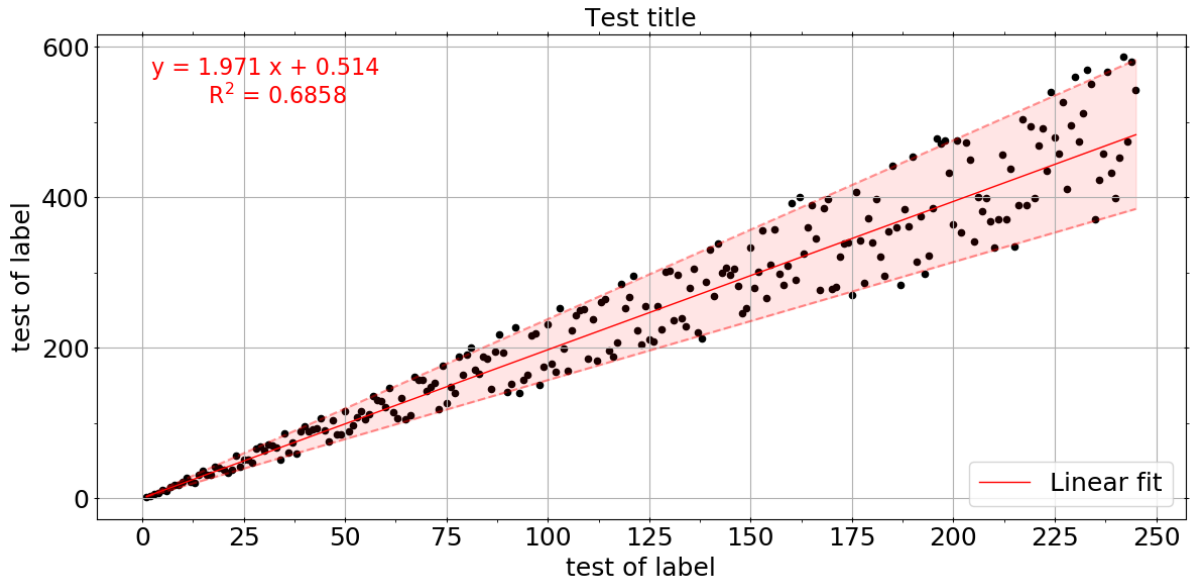

Figure 1: Test figure of scatter plot with linear fit including slope uncertainty, labeling of axes, legend positioning, and showcasing the fit function and gridlines.

5.2 savecard, scatter, exp split fit, labels, fontsizes, colours

Input file:

```
% png=exp_test.png
# Plot=scatter fit=exp fit_plot=split
$ xaxis="time [ns]" yaxis="angular Frequency, $\omega$ [Hz]"
$ title="exp test" legend="Exponential Data"
$ axisfontsize=25 titlefontsize=10 tickmarksize=20
$ datacolour="c" fitcolour="mediumpurple"

1       3684.124434
2       3243.708389
3       2716.687596
...     ...
98      147.9633327
99      176.021132
100     130.1782474
```

(data truncated for brevity)
Output figure:

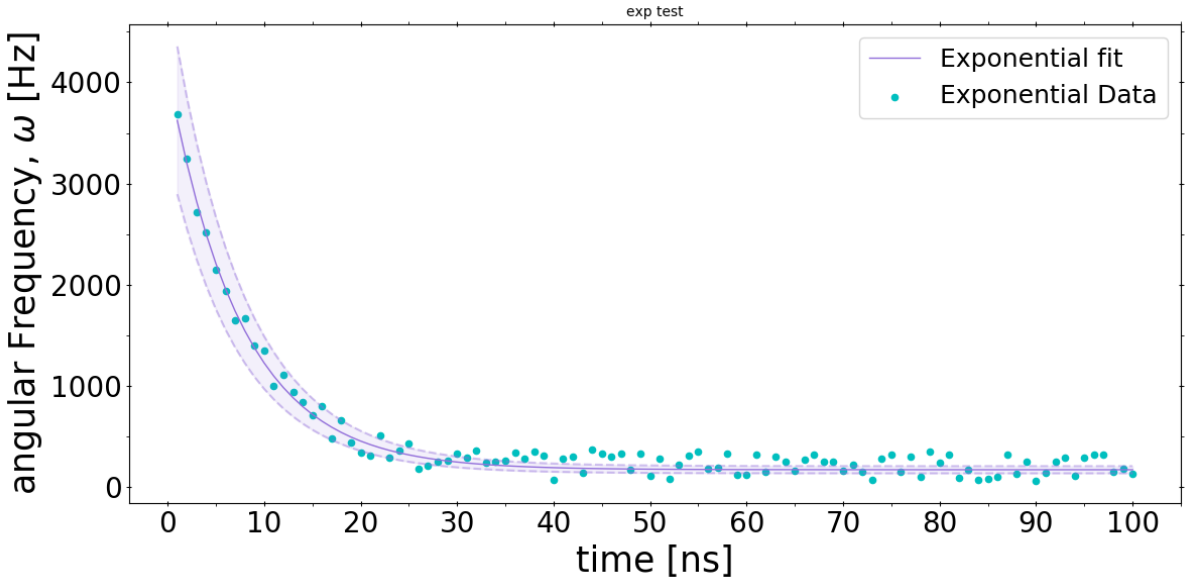

Figure 2: Test figure of scatter plot with exponential fit including uncertainties on the decay and asymptote parameters, labeling using Greek letters and LaTeXcode, legend input, font size changes and colours.

5.3 Deviation bar plot: savecard, deviation (with scatter), labels, colours, model, multiple data columns and axes, legend location, legendbox, devpattern

Input file:

```
% png=test_deviation.png
# plot=dev deviation=(mae,me,std,maxerr,scatter,ref=1)
$ xaxis="Model"
$ yaxis="Deviation (bars) [Hz];Deviations (dots) [Hz]"
$ title="Carbon-carbon spin-spin coupling constant deviations"
$ devcol="mae:g;me:(1,0,0);std:mediumpurple;maxerr:c"
$ devpattern="on"
$ model="augj SOPPA;augj HRPA(D);augj RPA(D)"
$ legloc="UPPER right"
$ pointsize=70
$ gridlines="y"
$ legendbox="off" legcolumns=2
$ legendfontsize=12

exp     augJ_SOPPA      augJ_HRPA(D)    augJ_RPA(D)
12.4    14.0225         15.0128         7.6231
28.4    30.7721         31.3256         20.1682
-8.1    -11.2483        -10.1492        -5.5872
...     ...             ...             ...
41.5    44.6432         45.3477         37.1624
31.6    33.5309         34.4587         26.0827
0.89    1.5175          1.449           0.8163
```

(data truncated for brevity)
Note that the line starting with "exp" isn't read - this is purely as a note to oneself. Also note that the `legloc="UPPER right"` is no different from writing `legloc="upper right"`, is this input is case-insensitive.
Output figure:

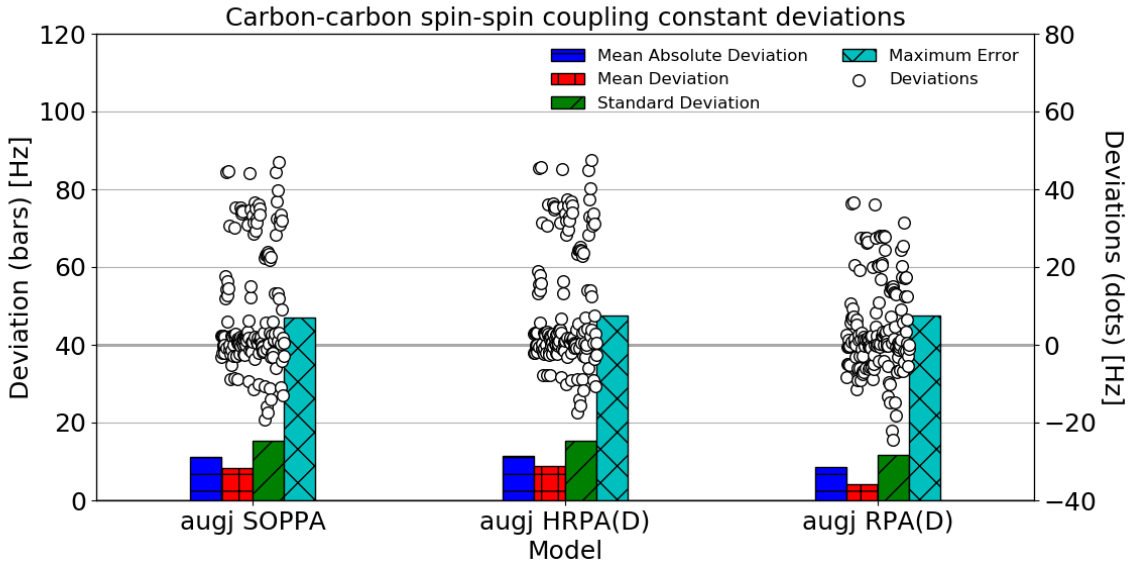

Figure 3: Test figure of deviation bar plot with MAE, ME, STD, MaxErr, and deviations, multiple yaxes, bar colours and patterns, model, legend location, scatter point size, gridlines, legendbox, legend columns, and legend font size.

Please note that there are two y-axes with different 0. This is determined by the specific values of the input data - the (bars) axis is scaled based on the size of all the bars, while the (dots) axis is scaled based on the sizes of each individual deviation. Also note, that this is specified in the input file using the ";" separator.

5.4 Subplots: scatter, linear line fit, ref, subplot, labels, colours

Input file:

```
# plot=scatter fit=linear ref=2 subplot="loose"
$ xaxis="aug-cc-pVTZ"
$ yaxis="aug-cc-pVDZ;aug-cc-pVQZ;d-aug-cc-pVTZ;Sadlej"
$ title="Correlation"
$ datacolor="darkorchid" fitcolour="black"
$ show_legend="off"

71.27   70.96   70.61   71.03   71.52
50.34   50.05   49.78   50.18   50.69
66.17   65.79   65.47   65.86   66.29
61.24   60.92   60.65   61.01   61.31
61.13   60.79   60.50   60.85   61.17
61.01   60.64   60.35   60.71   61.09
56.53   56.06   55.72   56.19   56.92
76.30   76.06   75.73   76.26   77.05
84.33   83.88   83.88   83.88   84.70
51.00   50.61   50.31   50.73   51.29
61.26   61.24   61.00   61.37   61.82
```

```
48.47    48.25    47.99    48.39    48.80
66.09    66.07    65.80    66.20    66.75
89.22    89.00    88.64    89.07    89.47
77.93    77.59    77.20    77.69    78.33
113.84   113.75   110.00   113.92   114.49
82.49    82.37    82.05    82.53    83.01
```

Here, the `ref=2` keyword signifies, that the 2nd column (ie. the column beginning with 70.96) is the reference column.
Note that the `plot=scatter` here will return 4 figures (one figure for each of the non-reference columns vs. the reference column) and the `subplot=True` yields a figure comprised of each of the above figures. The subplot figure is shown below:

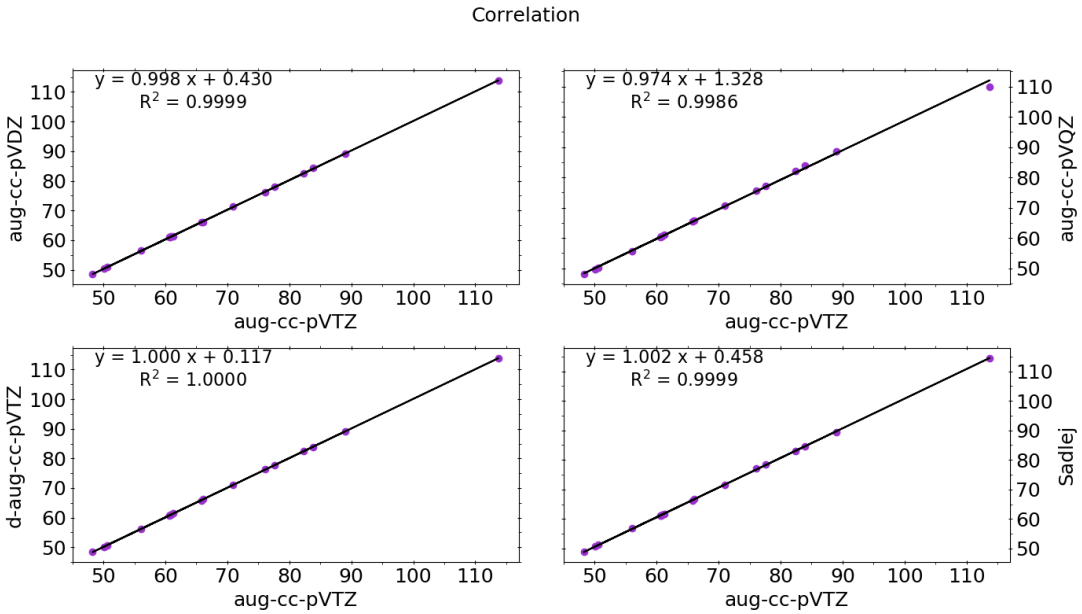

Figure 4: Test figure of loose subplots with multiple yaxes, colours, and turning off the legend.

The above figure uses the `subplot="loose"` keyword. If instead the `subplot="sticky"` keyword is used, the following subplot figure is generated:

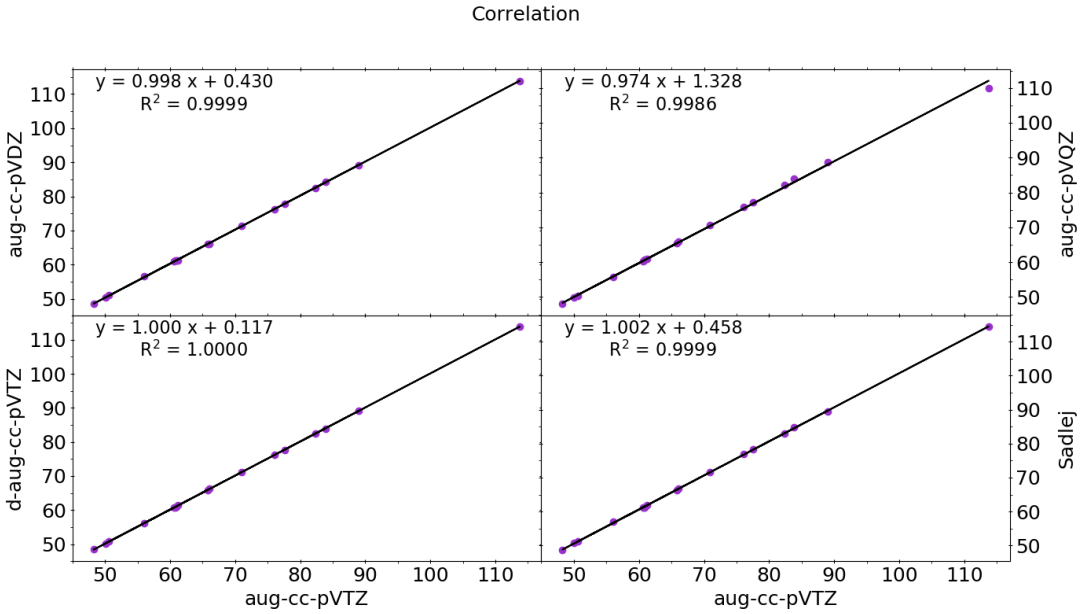

Figure 5: Test figure of sticky subplots with everything else like in figure 4.

5.5 subplotframe

In the above input file, if the titlecard keyword `subplotframe=line` is added, the resulting figure will be returned:

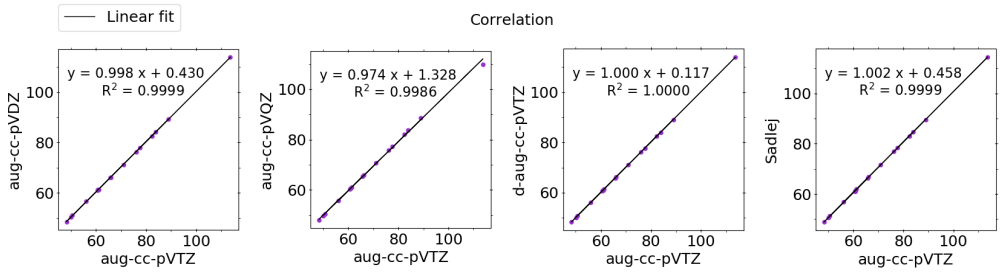

Figure 6: Test figure of subplotframe keyword with everything else as in figures 4 and 5.

From this, it should be clear that this keyword ought not be used when generating more than 4 subplots in the line.

5.6 bar, labels, precalculated deviations, devpattern, rot, gridlines, tickalignment

Input file:

```
# plot=bar
$ xaxis="Models" yaxis="Deviations [Hz]"
$ title="Deviations of select models"
$ model="HF;H2O;NH3;HFHF;H2O$\bullet$H2O;H2O$\bullet$NHaH2;
        H2O$\bullet$NHbH2;H–F;O–H;N–Ha;N–Hb"
```

```
$ legend="Mean Deviation [Hz];Mean Absolute Deviation [Hz];
        Standard Deviation [Hz]"
$ rot=30
$ devpattern="*;/;b"
$ gridlines="y"
$ tickalignment="right"

10.94505556     15.81094444     19.3914058
-1.999586111    1.999586111     3.384377945
-2.201608333    2.201608333     4.824581275
6.765416667     10.83458333     14.31503686
-1.220588889    1.600633333     1.872674722
-1.092708333    1.117325        1.356193123
-1.086605556    1.114022222     1.349878262
0.22575         6.843805556     14.45227292
0.778997222     0.778997222     4.514888475
1.1089          1.423972222     4.178511081
1.115002778     1.127147222     4.160641688
```

Note that the `model` and `legend` keywords have been modified with a linebreak in order to fit all on the page. PlotPro will not understand linebroken inputs like this!
Output figure:

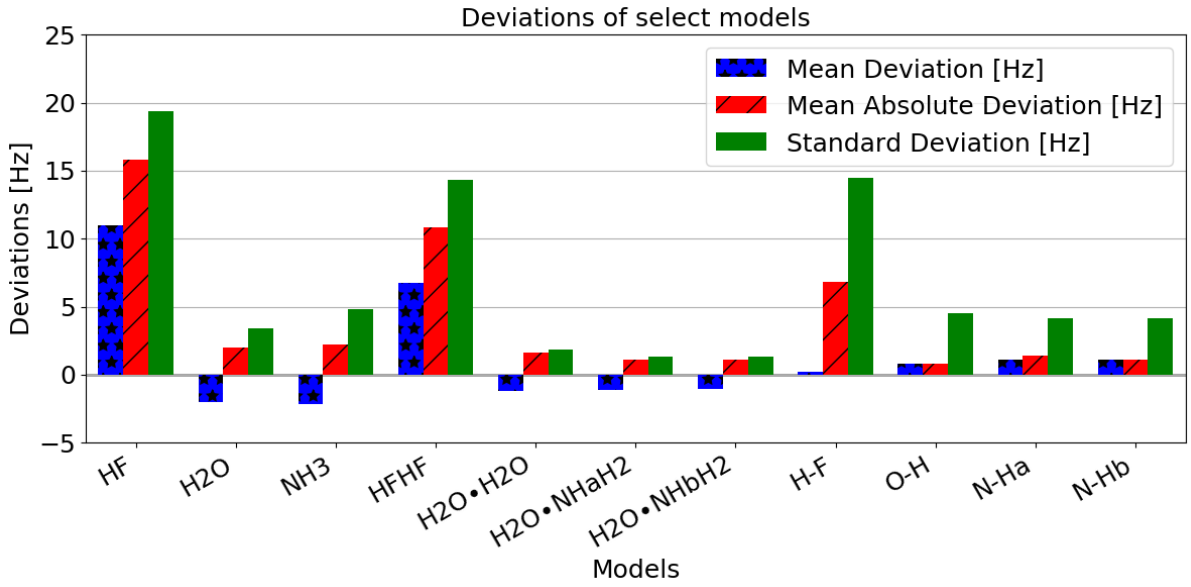

Figure 7: Test figure of deviation bars, where the input are the already calculated deviation parameters (ie. the program simply plots) including special LaTeX-characters in model names, multiple legend inputs, text rotation, custom bar patterns, gridlines, and tickalignment.

Also note that the input `devpattern="*;/;b"` yields patterns on the first two bars, but not the last as the `"b"` value is not recognised as a hatch pattern. If such an unsupported value is given, PlotPro will default to no hatch and the output file will list the supported values for future reference.

Using the program

This program was shared on Sunray, and can be found in the following directory: `/home/moller/opt/plotpro/plotpro.py`.

6 Running the program

6.1 Running on Sunray

If running the program on Sunray, it is recommended that you set up an alias for the program: when in your main folder (the one you are in upon logging in) open the `.bashrc` file. Anywhere you please, though I recommend grouping it with any existing aliases, write `alias plotpro='/home/moller/opt/plotpro/plotpro.py'`, then save and exit the `.bashrc`. From the main folder, write `source .bashrc` to "activate" the alias. You can now call the program by writing `plotpro *.txt`, where * should be replaced with the name of the input .txt file. Note that you need only be in the directory of this input file for it to work.
For the figures to be shown, an X11 tunnel has to be established. This can be done wither by using the `ssh -XC username@sunray.theory.ki.ku.dk` login option, or by installing Xming - follow this link[3] for a guide on establishing a local SSH X11 tunnel via Xming. Note that when starting a new session on your machine, Xming needs to be started for the X11 tunnel to be activated.

6.2 Running locally

To successfully run the program locally, first of all, both the .py files (**plotpro.py** and **setup_functions.py**) have to be downloaded and placed in the same folder. Next, ensure that the Python packages mentioned in the "Program Details" part are installed and up to date with the used versions (see details in section 7). When all packages are installed and sufficiently up-to-date, use the command line to enter the folder in which the .py files are located(`cd` changes directory). The .txt input file should also be in this directory. The program can then be called by writing `python plotpro.py *.txt`, where * should be replaced with the name of the input .txt file.

7 Updating Python packages

The following is written to work for Windows users. Mac/OSX users should look up "Homebrew Python", and is only relevant when running the program locally.
To check what packages are installed, open the command line (search for "cmd") and write `pip list`. This shows a list of all the installed packages as well as their versions. Cross-check with the above mentioned used packages and ensure that your installed version is at least as up-to-date as the one specified.
If one of the required packages are not installed, write `pip install * --user` where * should be replaced with the name of the package.

[3] URL: http://www.geo.mtu.edu/geoschem/docs/putty_install.html

If your version is not up-to-date, enter `pip install --upgrade * --user`, where * should be replaced with the name of the package to update.
For instance, if installing or updating NumPy, it would look like this:
`pip install numpy --user` or `pip install --upgrade numpy --user`.

8 My figures are ugly - help

In the following I will briefly discuss some ways to solve common problems that may arise during use of PlotPro.

8.1 Legend

If the legend does not look the way you want, you can try the following fixes:

- `legloc` can change the position of the legend (see sec. 3, or figs. 1 and 3). The default is `legloc=1` which is identical to `legloc="upper right"`. If extreme repositioning of the legend (ie. outside of the figure), one could use `legloc="(0.05,1.2)"`, which places the legend over the top left corner of the figure.
- `legendfontsize` changes the font size of the text in the legend and therefore also the overall size of the legend (see sec. 3, or fig. 3). The default is `legendfontsize=18`.
- `legcolumns` lets you control the number of columns that should be in the legend. The default is `legcolumns=1` meaning that all legend entries are written in one column. This keyword can be used to change the dimensionality of the legend, ie. make it a wider and shorter legend (see sec. 3, or fig. 3).
- `legendbox="off"` will turn off the box around the legend. This will slightly decrease the overall size of the legend, but if used with the `plot=dev` routecard keyword like in figure 3, you may experience that a point in the scatter plot may be close enough to the legend of the scatter points that it can cause confusion. (see sec. 3, or fig. 3).
- The easiest way to avoid the legend being in a weird place is simply not showing it. `show_legend="off"` does exactly this (see sec. 3, or fig. 4).

8.2 Tick marks

If the tick marks (and/or the labels at these tick marks) do not look the way you want, the following things can be done to change these:

- `tickmarksize` changes both the fontsize of the labels at the tick marks and the tick marks themselves (see sec. 3, or fig. 2). The default is `tickmarksize=18`.
- `rot` determines how many degrees the labels are rotated where 0 is the standard, horizontal orientation, and 90 is rotated 1/4 turn of a circle in positive orientation (counterclockwise), ie. the bottom of each letter is to the right of the top of the letter (see sec. 3, or fig. 7).

- `tickalignment` determines whether the tick labels should be anchored by the center of their text box, the left-most edge, or the right-most edge (see sec. 3, or fig. 7). The default is `tickalignment="center"`.

Program details

PlotPro was written in Python 3 using the following packages:

Python module	Functionality
NumPy (1.15.3)	General data treatment
Matplotlib/Pyplot (3.0.0)	Versatile plotting tool
Sys	Allows for communication between Python and the OS. Used for loading additional argument when calling the program.
RegEx (2018.11.7)	Regular Expressions used for some matching of supported keywords to input keywords.
ast	Abstract Syntax Trees used for literal evaluation of `colour` and `legloc` keyword specifications (so that both strings and tuples/floats can be used as input).
iminuit/Minuit (1.3.3)	Powerful fitting tool with a lot of flexibility.
probfit/Chi2Regression (1.1.0)	In conjunction with Minuit, the χ^2 Regression method means that the parameter being optimised is the χ^2 value of the fit.
SciPy/stats (1.1.0)	Used to return p-value of fit.

Using the `fit` routecard keyword, the program will attempt a fit, be it linear or exponential (or whatever function is added at a later date). This is done using a χ^2 fit in which the minimum of the n-dimensional surface of χ^2 values is found, where n is the number of free parameters (ie. 2 in the case of a linear fit). This method is somewhat sensitive to initial conditions, and these initial conditions are therefore estimated by the program based on the input data. For a linear fit, the slope is estimated as a global slope over the entire data series

$$a_{guess} = \frac{y_{end} - y_{start}}{x_{end} - x_{start}} \tag{1}$$

and the y-intercept is estimated as the difference between the given y-value and the, using a_{guess} calculated, corresponding function value of x.

$$b_{guess} = y_{start} - a_{guess} \cdot x_{start} \tag{2}$$

Thus, for correlation fits (ie. where the ideal values of a and b are 1 and 0 resp.), input fit parameters should land fairly close to the ideal values. Similarly, for δ vs. σ plots, the input slope starts near the ideal -1.
It is important to note that the parameters and uncertainties from Minuit are very sensitive to the input uncertainties as well. PlotPro uses a default error size of 1.0 to make the

fit (the input uncertainties determine the step size in the Minuit optimisation routine). The parameter most affected by this sensitivity is the y-intercept of the linear fit, which is why the relative uncertainty easily exceeds the relative uncertainty on the slope by several orders of magnitude. It is also important to note, that if the user decides to specify an uncertainty, then the fit will likely either yield a large relative uncertainty on the intercept, or a large χ^2 value. These large values shouldn't be taken to heart too much, so don't worry :) (small uncertainties may result in a large χ^2, and large uncertainties may result in large fit uncertainties).

Upon fitting, the output file will contain information on these input parameters as well as the resulting fitted parameters including their fitted uncertainties (to 20 decimal places) and their relative errors, ie. how large the uncertainty is compared to the parameter. Also included is the R^2 value, standard deviation (STD), χ^2 value, degrees of freedom, χ^2 probability (ie. the probability that the null hypothesis, e.g. that the data are linearly correlated, is "correct") and a "Yes" or "No" comment on whether or not the fit converged.

9 Further plans

My plans for further work on PlotPro include constructing a GUI, thus decreasing the need for several keywords in the input file. This should hopefully prove more intuitive for the user, and easier to change small things such as font size and legend location.