
ccsgp_get_started Documentation

Release 0.0

Patrick Huck

Sep 27, 2017

Contents

1	Examples Module	3
2	ccsgp	9
2.1	User Functions	11
2.2	Base Class	13
2.3	Config & Utils	17
3	Indices and tables	19
	Python Module Index	21

Authors Patrick Huck ([GitHub](#)), original `default_colors` in `ccsgp_get_started.ccsqp.config` contributed by Johanna Huck

Date Sep 27, 2017

get started with the ccsqp plotting library

CHAPTER 1

Examples Module

The examples are based on a dataset of World Bank [Indicators](#). You can use the dataset yourself to play around¹. See the `genExDat.sh` script in the same directory on how I extracted the data into the correct format for `ccsgp`. To generate all example plots based on `ccsgp_get_started_data` you can run:

```
$ python -m ccsgp_get_started
```

Alternatively, you can run a specific module, for instance:

```
$ python -m ccsgp_get_started.examples.gp_datdir [--log] <country-initial> <#-most-  
↪populated>
```

and this way plot specific country initials. You can open all resulting pictures via `$ open examplesDir/examples/gp_datdir/*.pdf` or use `pdftup` to put multiple plots on one page. To start on your own read the documentation below or the source code and use one of the examples as a template.

`ccsgp_get_started.examples.gp_datdir.gp_datdir(initial, topN)`
example for plotting from a text file via `numpy.loadtxt`

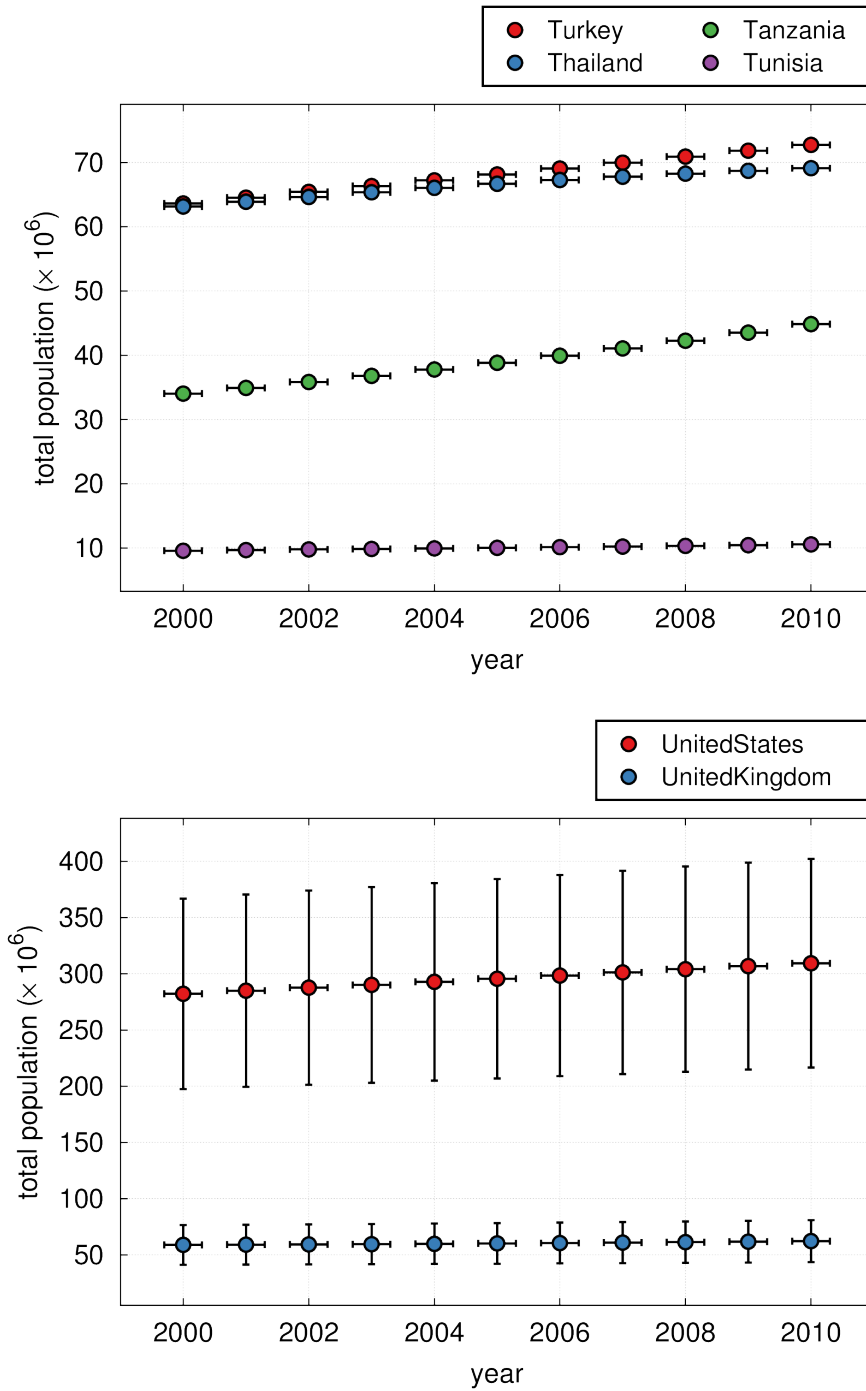
- 1.prepare input/output directories
- 2.load the data into an `OrderedDict()` [adjust axes units]
- 3.sort countries from highest to lowest population
- 4.select the <topN> most populated countries
- 5.call `ccsgp.make_plot` with data from 4

Below is an output image for country initial T and the 4 most populated countries for this initial (click to enlarge). Also see:

```
$ python -m ccsgp_get_started.examples.gp_datdir -h
```

for help on the command line options.

¹ `ccsgp_get_started_data/input/examples/gp_datdir/{WorldBankIndicators.csv, genExDat.sh}`



Parameters

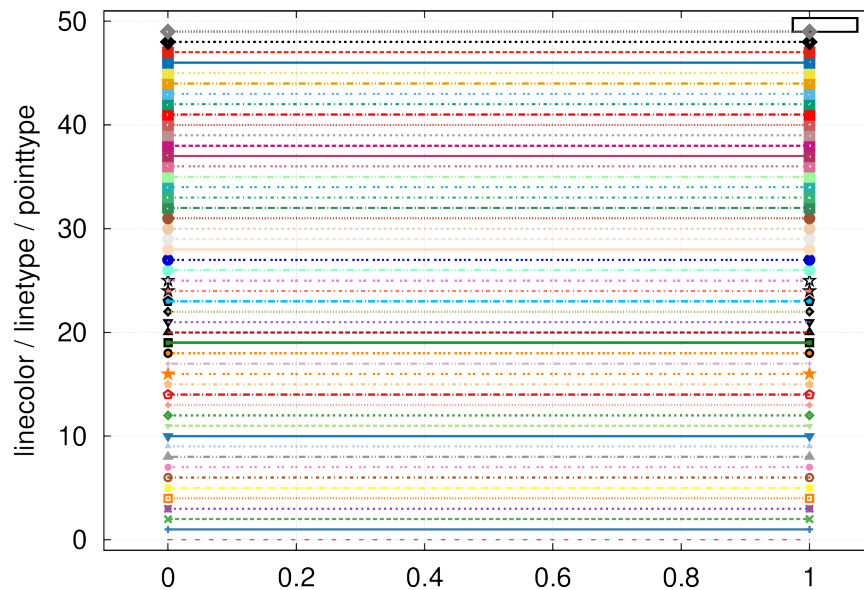
- **initial** (*str*) – country initial
- **topN** (*int*) – number of most populated countries to plot

Variables

- **inDir** – input directory according to package structure and initial
- **outDir** – output directory according to package structure

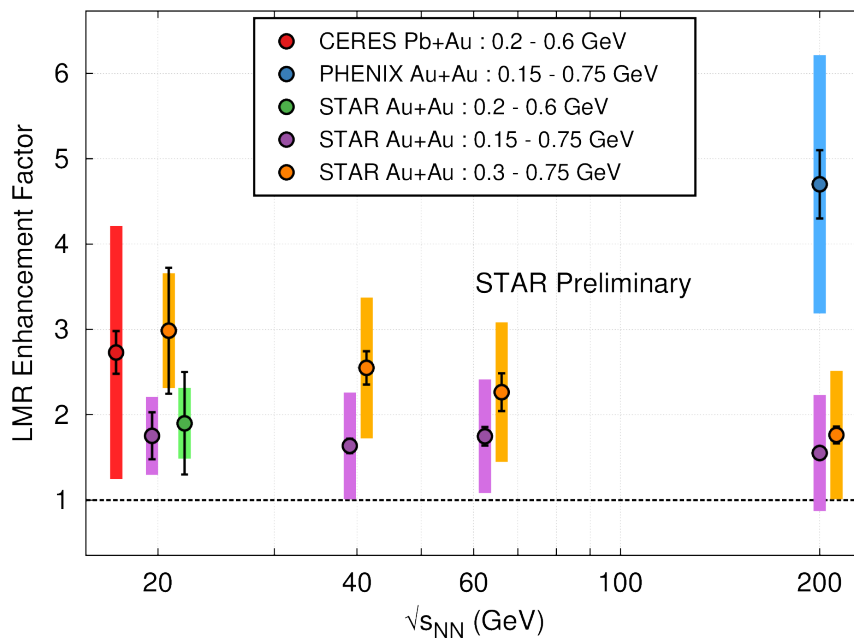
- **data** – OrderedDict with datasets to plot as separate keys
- **file** – data input file for specific country, format: [x y] OR [x y dx dy]
- **country** – country, filename stem of input file
- **file_url** – absolute url to input file
- **nSets** – number of datasets

`ccsgp_get_started.examples.gp_lcltpt.gp_lcltpt()`
example plot to display linecolors, linetypes and pointtypes



`ccsgp_get_started.examples.gp_xfac.gp_xfac()`
example using QM12 enhancement factors

- uses *gpcalls* kwarg to reset xtics
- `numpy.loadtxt` needs reshaping for input files w/ only one datapoint
- according poster presentations see [QM12](#) & [NSD](#) review

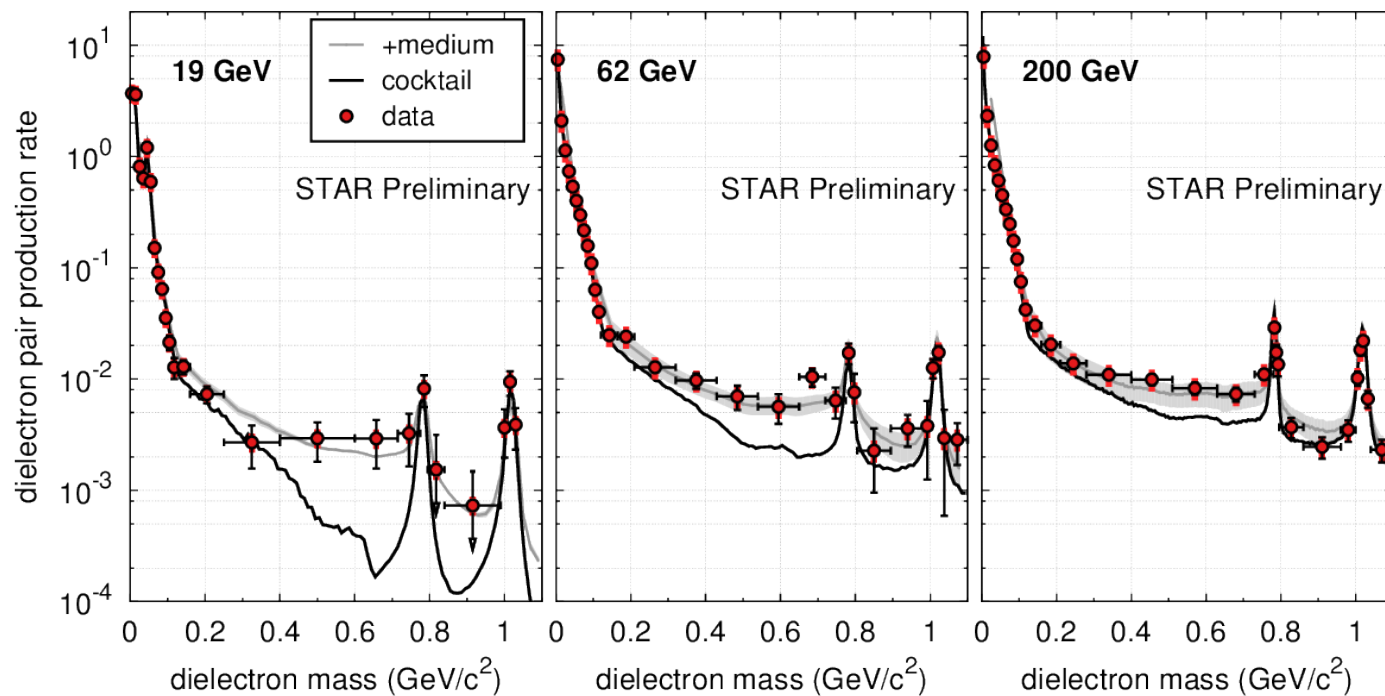


Variables

- **key** – translates filename into legend/key label
- **shift** – slightly shift selected data points

`ccsgp_get_started.examples.gp_panel.gp_panel` (*version*, *skip*)

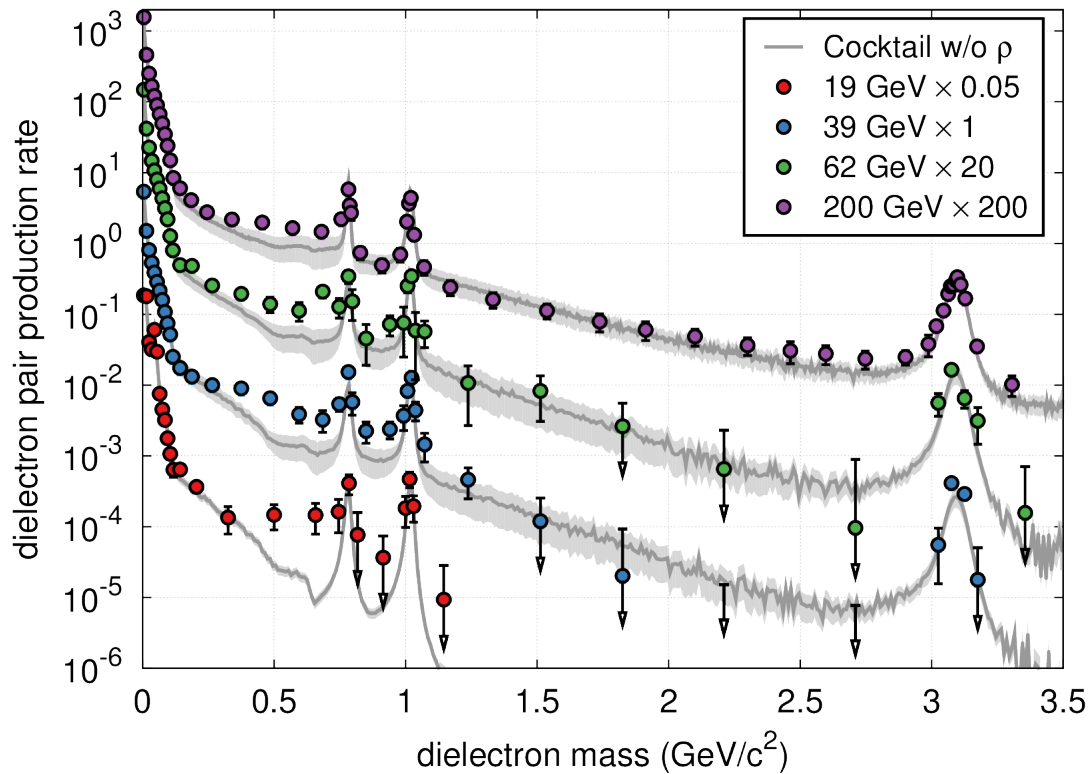
example for a panel plot using QM12 data (see `gp_xfac`)



Parameters *version* (*str*) – plot version / input subdir name

`ccsgp_get_started.examples.gp_stack.gp_stack` (*version, energies, inclMed, inclFits*)
 example for a plot w/ stacked graphs using QM12 data (see `gp_panel`)

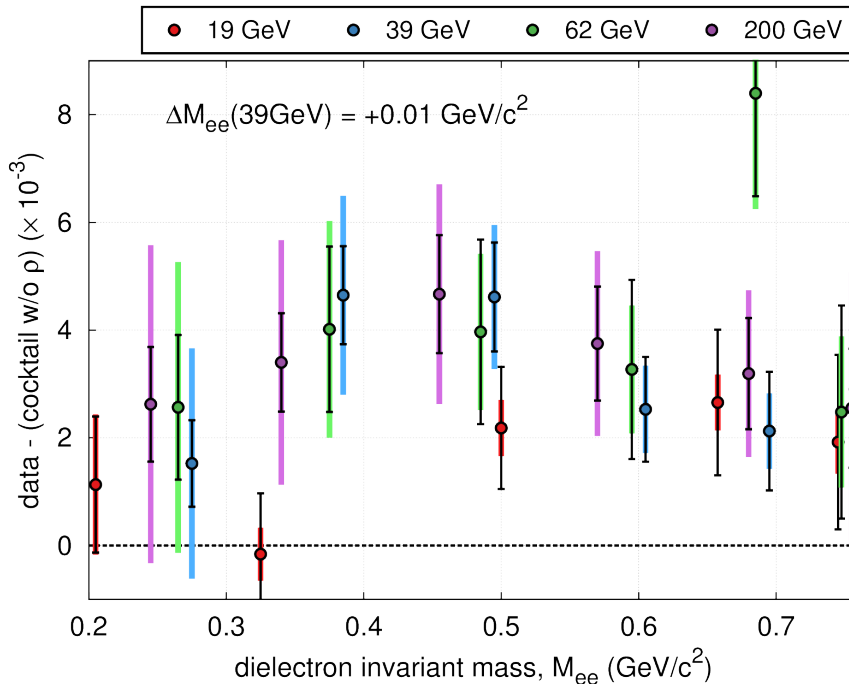
- how to omit keys from the legend
- manually add legend entries
- automatically plot arrows for error bars larger than data point value



Parameters `version` (*str*) – plot version / input subdir name

`ccsgp_get_started.examples.gp_rdiff.gp_rdiff` (*version, nomed, noxerr, diffRel, divdNdy*)
 example for ratio or difference plots using QM12 data (see `gp_panel`)

- uses uncertainties package for easier error propagation and rebinning
- stat. error for medium = 0!
- stat. error for cocktail ~ 0!
- statistical error bar on data stays the same for diff
- TODO: implement ratio!
- TODO: adjust statistical error on data for ratio!
- TODO: adjust name and ylabel for ratio



Parameters

- **version** (*str*) – plot version
- **nomed** (*bool*) – don't plot medium
- **noxerr** (*bool*) – don't plot x-errors

`ccsgp_get_started.examples.gp_ptspec.gp_ptspec()`
example for a 2D-panel plot etc.

`ccsgp_get_started.examples.utils.enumzipEdges(eArr)`
zip and enumerate edges into pairs of lower and upper limits

`ccsgp_get_started.examples.utils.getCocktailSum(e0, e1, eCocktail, uCocktail)`
get the cocktail sum for a given data bin range

`ccsgp_get_started.examples.utils.getEdges(npArr)`
get np array of bin edges

`ccsgp_get_started.examples.utils.getErrorComponent(result, tag)`
get total error contribution for component with specific tag

`ccsgp_get_started.examples.utils.getMaskIndices(mask)`
get lower and upper index of mask

`ccsgp_get_started.examples.utils.getUArray(npArr)`
uncertainty array multiplied by binwidth (col2 = dx)

`ccsgp_get_started.examples.utils.getWorkDirs()`
get input/output dirs (same input/output layout as for package)

ccsgp is a plotting library based on [gnuplot-py](#) which wraps the necessary calls to gnuplot-py into one function called `make_plot`. The keyword arguments to `make_plot` provide easy control over the plot-by-plot dependent options while reasonable defaults for legend, grid, borders, font sizes, terminal etc. are handled internally. By providing the data in a default and reasonable format, the user does not need to deal with the details of “gnuplot’ing” nor the internals of the gnuplot-py interface library. Every call of `make_plot` dumps an ascii representation of the plot in the terminal and generates the eps hardcopy original. The eps figure is also converted automatically into pdf, png and jpg formats for easy inclusion in presentations and papers. In addition, the user can decide to save the data contained in each image into hdf5 files for easy access via numpy. The function *repeat_plot* allows the user replot a specific graph with different properties, like axis ranges for instance. The `make_panel` user function facilitates plotting of 1D- or 2D-panel images with merged axes.

The name *ccsgp* stands for “Carbon Capture and Sequestration GnuPlot” as this library started off in the context of my wife’s [research](#). I knew how to produce nice-looking plots using gnuplot but wanted to hook it up to python directly. The resulting library let’s me generate identical plots independent of the data input source (ROOT, YAML, txt, pickle, hdf5, ...) using the full power of python. Gnuplot – A pipe-based interface to the gnuplot plotting program.

This is the main module of the Gnuplot package.

Written by “Michael Haggerty”, <mailto:mhagger@alum.mit.edu>. Inspired by and partly derived from an earlier version by “Konrad Hinsén”, <mailto:hinsen@ibs.ibs.fr>. If you find a problem or have a suggestion, please “let me know”, <mailto:mhagger@alum.mit.edu>. Other feedback would also be appreciated.

The Gnuplot.py home page is at

“Gnuplot.py”, <http://gnuplot-py.sourceforge.net>

For information about how to use this module:

1. Check the README file.
2. Look at the example code in `demo.py` and try running it by typing ‘python `demo.py`’ or ‘python `__init__.py`’.
3. For more details see the extensive documentation strings throughout the python source files, especially this file, `_Gnuplot.py`, `PlotItems.py`, and `gp_unix.py`.
4. The docstrings have also been turned into html which can be read “here”, <http://gnuplot-py.sourceforge.net/doc>. However, the formatting is not perfect; when in doubt, double-check the docstrings.

You should import this file with ‘import Gnuplot’, not with ‘from Gnuplot import *’, because the module and the main class have the same name, ‘Gnuplot’.

To obtain the gnuplot plotting program itself, see “the gnuplot FAQ”, [ftp://ftp.gnuplot.vt.edu/pub/gnuplot/faq/index.html](http://ftp.gnuplot.vt.edu/pub/gnuplot/faq/index.html). Obviously you need to have gnuplot installed if you want to use Gnuplot.py.

The old command-based interface to gnuplot (previously supported as ‘oldplot.py’) has been removed from the package.

Features:

- o **Allows the creation of two or three dimensional plots from python.**
- o **A gnuplot session is an instance of class ‘Gnuplot’.** Multiple sessions can be open at once. For example:

```
g1 = Gnuplot.Gnuplot()
g2 = Gnuplot.Gnuplot()
```

Note that due to limitations on those platforms, opening multiple simultaneous sessions on Windows or Macintosh may not work correctly. (Feedback?)

- o **The implicitly-generated gnuplot commands can be stored to a file** instead of executed immediately:

```
g = Gnuplot.Gnuplot('commands.txt')
```

The ‘commands.txt’ file can then be run later with gnuplot’s ‘load’ command. Beware, however: the plot commands may depend on the existence of temporary files, which will probably be deleted before you use the command file.

- o Can pass arbitrary commands to the gnuplot command interpreter:

```
g('set pointsize 2')
```

(If this **is** all you want to do, you might consider using the lightweight GnuplotProcess **class defined in** gp.py.)

- o **A Gnuplot object knows how to plot objects of type ‘PlotItem’.** Any PlotItem can have optional ‘title’ and/or ‘with’ suboptions. Builtin PlotItem types:

- **‘Data(array1)’ – data from a Python list or NumPy array** (permits additional option ‘cols’)
- **‘File(‘filename’)’ – data from an existing data file** (permits additional option ‘using’)
- **‘Func(‘exp(4.0 * sin(x))’)** – functions (passed as a string, evaluated by gnuplot)
- **‘GridData(m, x, y)’ – data tabulated on a grid of (x,y) values** (usually to be plotted in 3-D)

See the documentation strings for those classes for more details.

- o **PlotItems are implemented as objects that can be assigned to variables** and plotted repeatedly. Most of their plot options can also be changed with the new ‘set_option()’ member functions then they can be replotted with their new options.

- o **Communication of commands to gnuplot is via a one-way pipe.** Communication of data from python to gnuplot is via inline data (through the command pipe) or via temporary files. Temp files are deleted automatically when their associated ‘PlotItem’ is deleted. The PlotItems in use by a Gnuplot object at any given time are stored in an internal list so that they won’t be deleted prematurely.

- o Can use ‘replot’ method to add datasets to an existing plot.

- o Can make persistent gnuplot windows by using the constructor option 'persist=1'. Such windows stay around even after the gnuplot program is exited. Note that only newer version of gnuplot support this option.
- o Can plot either directly to a postscript printer or to a postscript file via the 'hardcopy' method.
- o Grid data for the splot command can be sent to gnuplot in binary format, saving time and disk space.
- o Should work under Unix, Macintosh, and Windows.

Restrictions:

- Relies on the numpy Python extension. This can be obtained from the Scipy group at <http://www.scipy.org/Download>. If you're interested in gnuplot, you would probably also want numpy anyway.
- Only a small fraction of gnuplot functionality is implemented as explicit method functions. However, you can give arbitrary commands to gnuplot manually:

```
g = Gnuplot.Gnuplot()
g('set data style linespoints')
g('set pointsize 5')
```

- There is no provision for missing data points in array data (which gnuplot allows via the 'set missing' command).

Bugs:

- No attempt is made to check for errors reported by gnuplot. On unix any gnuplot error messages simply appear on stderr. (I don't know what happens under Windows.)
- All of these classes perform their resource deallocation when '__del__' is called. Normally this works fine, but there are well-known cases when Python's automatic resource deallocation fails, which can leave temporary files around.

User Functions

`ccsgp_get_started.ccsdp.ccsdp.make_panel(dpt_dict, **kwargs)`
make a panel plot

- name/title/debug are global options used once to initialize the multiplot
- x, yr/x, ylog/lines/labels/gpcalls are applied on each subplot
- key/ylabel are only plotted in first subplot
- xlabel is centered over entire panel
- same for r, l, b, tmargin where r, lmargin will be reset, however, to allow for merged y-axes
- input: OrderedDict w/ subplot titles as keys and lists of make_plot's data/properties/titles as values, see below
- layout = '<cols>x<rows>', defaults to horizontal panel if omitted
- key_subplot_id sets the desired subplot to put the key in

Parameters `dpt_dict` (*dict*) – OrderedDict('subplot-title': [data, properties, titles], ...)

`ccsgp_get_started.ccsdp.ccsdp.make_plot(data, properties, titles, **kwargs)`
main function to generate a 1D plot

- each dataset is represented by a numpy array consisting of data points in the format `[x, y, dx, dy1, dy2]`, `dy1` = statistical error, `dy2` = systematic uncertainty
- for symbol numbers to use in labels see <http://bit.ly/1erBgIk>
- lines format: `'<x/y>=<value>': '<gnuplot options>'`, horizontal = (along) x, vertical = (along) y
- labels format: `'label text': [x, y, abs. placement true/false]`
- arrows format: `[<x0>, <y0>], [<x1>, <y1>], '<gnuplot props>'`

Parameters

- **data** (*list*) – datasets
- **properties** (*list*) – gnuplot property strings for each dataset (lc, lw, pt ...)
- **titles** (*list*) – legend/key titles for each dataset
- **name** (*str*) – basename of output files
- **title** (*str*) – image title
- **debug** (*bool*) – flag to switch to debug/verbose mode
- **key** (*list*) – legend/key options to be applied on top of default_key
- **xlabel** (*str*) – label for x-axis
- **ylabel** (*str*) – label for y-axis
- **xr** (*list*) – x-axis range
- **yr** (*list*) – y-axis range
- **xlog** (*bool*) – make x-axis logarithmic
- **ylog** (*bool*) – make y-axis logarithmic
- **lines** (*dict*) – vertical and horizontal lines
- **arrows** (*list*) – arrows
- **labels** (*dict*) – labels
- **lmargin** (*float*) – defines left margin size (relative to screen)
- **bmarg** (*float*) – defines bottom margin size
- **rmarg** (*float*) – defines right margin size
- **tmarg** (*float*) – defines top margin size
- **arrow_offset** (*float*) – offset from data point for special error bars (see gp_panel)
- **arrow_length** (*float*) – length of arrow from data point towards zero for special error bars (see gp_panel)
- **arrow_bar** (*float*) – width of vertical bar at end of special error bars (see gp_panel)
- **gpcalls** (*list*) – execute arbitrary gnuplot set commands

Returns MyPlot

`ccsgp_get_started.ccsdp.ccsdp.repeat_plot (plt, name, **kwargs)`
repeat a plot with different properties (kwargs see make_plot)

Parameters

- `plt` (`MyPlot`) – plot to repeat
- `name` (`str`) – basename of new output file(s)

Returns `plt`

Base Class

`class ccsgp_get_started.ccsgp.mypplot.MyPlot (name='test', title='', debug=0)`
base class

- basic gnuplot setup (bars, grid, title, key, terminal, multiplot)
- utility functions for general plotting

Parameters

- `title` (`str`) – image title
- `name` (`str`) – basename used for output files
- `debug` (`bool`) – debug flag for verbose gnuplot output

Variables

- `name` – basename for output files
- `epsname` – basename + '.eps'
- `gp` – Gnuplot.Gnuplot instance
- `nPanels` – number of panels in a multiplot
- `nVertLines` – number of vertical lines
- `nLabels` – number of labels
- `nArrows` – number of arrows
- `axisLog` – flags for logarithmic axes
- `axisRange` – axis range for respective axis (set in `setAxisRange`)

`_ascii()`

write ascii file(s) w/ data contained in plot

`_clamp(val, minimum=0, maximum=255)`

convenience function to clamp number into min..max range

`_colorscale(hexstr, scalefactor=1.4)`

Scales a hex string by `scalefactor`. Returns scaled hex string.

- taken from T. Burgess ([source](#))
- To darken the color, use a float value between 0 and 1.
- To brighten the color, use a float value greater than 1.

```
>>> colorscale("#DF3C3C", .5)
#6F1E1E
>>> colorscale("#52D24F", 1.6)
#83FF7E
>>> colorscale("#4F75D2", 1)
#4F75D2
```

`_convert()`
convert eps/ps original into pdf, png and jpg format

`_get_style_mod_prop(prop)`
get style and modified property string

`_hardcopy()`
generate eps, convert to other formats and write data to hdf5

`_hdf5()`
write data contained in plot to HDF5 file

- easy numpy import -> (savetxt) -> gnuplot
- export to ROOT objects

h5py howto (see <http://www.h5py.org/docs/intro/quick.html>):

- open file: `f = h5py.File(name, 'r')`
- list datasets: `list(f)`
- load entire dataset as np array: `arr = f['dset_name'][...]`
- NOTE: literally type the 3 dots, replace `dset_name`
- np.savetxt format: `fmt = '%.4f %.3e %.3e %.3e %.3e'`
- save array to txt file: `np.savetxt('arr.dat', arr, fmt=fmt)`

Raises ImportError

`_plot_errs(data)`
determine whether to plot primary errors separately
plot errorbars if data has more than two columns which are not all zero

Parameters `data` (`numpy.array`) – one dataset

Variables `error_sums` – sum of x and y errors

Returns True or False

`_plot_syserrs(data)`
determine whether to plot secondary errors

Parameters `data` (`numpy.array`) – one dataset

Returns True or False

`_prettify(str)`
prettify string, remove special symbols

`_setter(list)`
convenience function to set a list of gnuplot options

Parameters `list` (`list`) – list of strings given to gnuplot's set command

`_sum_errs(data, i)`
convenience function to calculate sum of i-th column

`_using(data, prop=None)`
determine string with columns to use

Parameters

- **data** (*numpy.array*) – one dataset
- **prop** (*str*) – property string of a dataset

Returns ‘1:2:3’, ‘1:2:4’ or ‘1:2:3:4’

_with_errs (*data, prop*)

generate special property string for primary errors

- currently error bars are drawn in black
- use same linewidth as for points
- TODO: give user the option to draw error bars in lighter color according to the respective data points

Parameters

- **data** (*numpy.array*) – one dataset
- **prop** (*str*) – property string of a dataset

Returns property string for primary errors

_with_main (*prop*)

get the correct property string for main data

_with_syserrs (*prop*)

generate special property string for secondary errors

- draw box in lighter color than point/line color
- does not support integer line colors, only hex

Parameters **prop** (*str*) – property string of a dataset

Returns property string for secondary errors

addHorizontalLine (*y, opts*)

draw horizontal line

Parameters

- **y** (*float*) – y-position
- **opts** (*str*) – line draw options

initData (*data, properties, titles, subplot_title=None*)

initialize the data

- all lists given as parameters must have the same length.
- each data set is drawn twice to allow for different colors for the errorbars
- error bars use the same linewidth as data points and line color black
- use ‘boxwidth 0.03 absolute’ in gp_calls to set the width of the uncertainty boxes
- use alternative gnuplot style if *properties* contains a style specification in the form with <style> and if the style is in `ccsgp.config.supported_styles` (style specification has to be at the beginning of the property string!)

Parameters

- **data** (*list of numpy arrays*) – data points w/ format [x, y, dx, dy] for each dataset
- **properties** (*list of str*) – plot properties for each dataset (pt/lw/ps/lc...)
- **titles** (*list of strings*) – key/legend titles for each dataset
- **subplot_title** (*str*) – subplot title for panel plot case

Variables

- **dataSets** – zipped titles and data for hdf5/ascii output and setAxisRange
- **data** – list of Gnuplot.Data including extra data sets for error plotting

plot (*hardcopy=True*)

plot and generate output files

prepare_plot (*margins=True, **kwargs*)

prepare for plotting (calls all members of MyPlot)

setArrow (*p0, p1, prop*)

draw an arrow into the figure

Parameters

- **p0** (*list*) – start point [x, y]
- **p1** (*list*) – end point [x, y]
- **prop** (*str*) – gnuplot property string for the arrow

setAxisLabel (*label, axis='x'*)

set label for specified axis

Parameters

- **label** (*str*) – label
- **axis** (*str*) – axis which to label

setAxisLog (*log, axis='x'*)

set logarithmic scale for specified axis

Parameters

- **log** (*bool*) – whether to set logarithmic
- **axis** (*str*) – axis which to set logarithmic

setAxisLogs (***kwargs*)

set axes logarithmic if requested

setAxisRange (*rng, axis='x'*)

set range for specified axis

- automatically determines axis range to include all data points if range is not given.
- logscale and secondary errors taken into account
- y-axis range determined for points within given x-axis range

Parameters

- **rng** (*list*) – lower and upper range limits
- **axis** (*str*) – axis to which to apply range

setErrorArrows (***kwargs*)

reset properties of arrows used to plot special errors

setKeyOptions (*key_opts*)

set key options

Parameters **key_opts** (*list*) – strings for key/legend options

setLabel (*label, pos, abs_place=False*)

draw a label into the figure

Parameters

- **label** (*str*) – label
- **pos** (*list*) – x,y - position
- **abs_place** (*bool*) – absolute or relative placement

setMargins (***kwargs*)

set the margins

- keys other than l(b,t,r)margin are ignored
- if margin not given leave to gnuplot

setVerticalLine (*x, opts*)

draw a vertical line

Parameters

- **x** (*float*) – position on x-axis
- **opts** (*str*) – line draw options

Config & Utils

var default_key default options for legend/key

var basic_setup bars, grid, terminal and default_key

var default_margins default margins to define plot area

var xPanProps xscale, xsize, xoffset for panel plots

var default_colors provides a reasonable color selection (see [palette](#))

`ccsgp_get_started.ccsqp.utils.getOpts` (*i*)

convenience function for easy access to gnuplot property string

`ccsgp_get_started.ccsqp.utils.zip_flat` (*a, b, c=None, d=None*)

zips 2-4 lists and flattens the result

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

C

- `ccsgp_get_started`, 1
- `ccsgp_get_started.ccsqp`, 9
- `ccsgp_get_started.ccsqp.ccsqp`, 11
- `ccsgp_get_started.ccsqp.config`, 17
- `ccsgp_get_started.ccsqp.Gnuplot`, 9
- `ccsgp_get_started.ccsqp.mypplot`, 13
- `ccsgp_get_started.ccsqp.utils`, 17
- `ccsgp_get_started.examples`, 3
- `ccsgp_get_started.examples.gp_datdir`, 3
- `ccsgp_get_started.examples.gp_lcltpt`, 5
- `ccsgp_get_started.examples.gp_panel`, 6
- `ccsgp_get_started.examples.gp_ptspec`, 8
- `ccsgp_get_started.examples.gp_rdiff`, 7
- `ccsgp_get_started.examples.gp_stack`, 6
- `ccsgp_get_started.examples.gp_xfac`, 5
- `ccsgp_get_started.examples.utils`, 8

Symbols

`_ascii()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 13
`_clamp()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 13
`_colormap()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 13
`_convert()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_get_style_mod_prop()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_hardcopy()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_hdf5()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_plot_errs()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_plot_syserrs()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_prettify()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_setter()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_sum_errs()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_using()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 14
`_with_errs()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 15
`_with_main()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 15
`_with_syserrs()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 15

A

`addHorizontalLine()` (ccsgp_get_started.ccsdp.myplot.MyPlot method), 15

C

`ccsgp_get_started` (module), 1
`ccsgp_get_started.ccsdp` (module), 9
`ccsgp_get_started.ccsdp.ccsdp` (module), 11
`ccsgp_get_started.ccsdp.config` (module), 17
`ccsgp_get_started.ccsdp.Gnuplot` (module), 9
`ccsgp_get_started.ccsdp.myplot` (module), 13
`ccsgp_get_started.ccsdp.utils` (module), 17
`ccsgp_get_started.examples` (module), 3
`ccsgp_get_started.examples.gp_datdir` (module), 3
`ccsgp_get_started.examples.gp_lcltpt` (module), 5
`ccsgp_get_started.examples.gp_panel` (module), 6
`ccsgp_get_started.examples.gp_ptspec` (module), 8
`ccsgp_get_started.examples.gp_rdiff` (module), 7
`ccsgp_get_started.examples.gp_stack` (module), 6
`ccsgp_get_started.examples.gp_xfac` (module), 5
`ccsgp_get_started.examples.utils` (module), 8

E

`enumzipEdges()` (in module `ccsgp_get_started.examples.utils`), 8

G

`getCocktailSum()` (in module `ccsgp_get_started.examples.utils`), 8
`getEdges()` (in module `ccsgp_get_started.examples.utils`), 8
`getErrorComponent()` (in module `ccsgp_get_started.examples.utils`), 8
`getMaskIndices()` (in module `ccsgp_get_started.examples.utils`), 8
`getOpts()` (in module `ccsgp_get_started.ccsdp.utils`), 17
`getUArray()` (in module `ccsgp_get_started.examples.utils`), 8
`getWorkDirs()` (in module `ccsgp_get_started.examples.utils`), 8
`gp_datdir()` (in module `ccsgp_get_started.examples.gp_datdir`), 3

`gp_lcltpt()` (in module `ccsgp_get_started.examples.gp_lcltpt`), [5](#)
`gp_panel()` (in module `ccsgp_get_started.examples.gp_panel`), [6](#)
`gp_ptspec()` (in module `ccsgp_get_started.examples.gp_ptspec`), [8](#)
`gp_rdiff()` (in module `ccsgp_get_started.examples.gp_rdiff`), [7](#)
`gp_stack()` (in module `ccsgp_get_started.examples.gp_stack`), [6](#)
`gp_xfac()` (in module `ccsgp_get_started.examples.gp_xfac`), [5](#)

`cc-` `setVerticalLine()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [17](#)

Z

`cc-` `zip_flat()` (in module `ccsgp_get_started.ccsdp.utils`), [17](#)

I

`initData()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [15](#)

M

`make_panel()` (in module `ccsgp_get_started.ccsdp.ccsdp`), [11](#)
`make_plot()` (in module `ccsgp_get_started.ccsdp.ccsdp`), [11](#)
`MyPlot` (class in `ccsgp_get_started.ccsdp.mypplot`), [13](#)

P

`plot()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`prepare_plot()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)

R

`repeat_plot()` (in module `ccsgp_get_started.ccsdp.ccsdp`), [12](#)

S

`setArrow()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setAxisLabel()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setAxisLog()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setAxisLogs()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setAxisRange()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setErrorArrows()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [16](#)
`setKeyOptions()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [17](#)
`setLabel()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [17](#)
`setMargins()` (`ccsgp_get_started.ccsdp.mypplot.MyPlot` method), [17](#)