Gnuplot bindings

Author: Kay-Uwe Kirstein
Version: 0.1

Copying

Introduction

This module contains a pure binding to gnuplot. Communication to gnuplot is performed via pipes. The usual work flow to generate plot via gnuplot is the following:

1. open pipe via `open`
2. send plot commands, e.g., with `plot`
3. close pipe with `close`

Function Reference

Open / Closing Functions

```
gplot::open cmd;
```

opens a pipe to gnuplot, using *cmd*. *cmd* usually is something like `gnuplot` or `/path/to/gnuplot/bin/gnuplot` depending on your path configuration. `open` returns a pointer to the actual pipe for later usage, so a typical call to open might look like this:

```
let gp = gplot::open "/path_to_gnuplot/gnuplot";
```

gplot::GPLOT_EXE is a predefined variable with the standard Gnuplot executable. It is set to `pgnuplot` on Windows and to `gnuplot` otherwise and can be overridden bythe GPLOT_EXE environment variable. (`pgnuplot.exe` is a special executable for Windows, which is capable of stdin pipes in contrast to the normal `gnuplot.exe`). Usage of `gplot::GPLOT_EXE` might look like this:

```
let gp = gplot::open gplot::GPLOT_EXE;
gplot::close gp;
```

closes a gnuplot session, given by the handle *gp*.

Low-Level Commands

```
gplot::puts_no_echo string gp;
```

sends the string to the gnuplot session *gp* points to. As the name states, there is no echo read back from gnuplot (Don't know whether *gnuplot* or *pgnuplot.exe* supports reading/bidirectional pipes at all).

```
gplot::puts string gp;
```

is a convenience wrapper to `gplot::puts_no_echo`.

Plot Commands

The main (versatile) function to generate plots is the simple plot command, which expects a list of the data to be plotted.

```
gplot::plot gp data opt;
```

where *gp* is the pointer to the gnuplot session, *data* is a list containing the data to be plotted and *opt* is a tuple, containing options for the plot. *opt* might be empty () or `DEFAULT` for default options (refer to gnuplot for them).

If data for the x-axis (ordinate) should be explicitely given *plotxy* should be used instead:

```
gplot::plotxy_deprecated gp (xdata, ydata) opt;
gplot::plotxy gp (xdata, ydata) opt [];
```

Multiple datasets can be plotted into a single graph by combining them to tuples of lists:

```
gplot::plotxy gp (xdata, y1data, y2data, ..)  opt;
gplot::plotxy gp (xdata, y1data, y2data, ..)  opt [];
gplot::plotxy gp (xdata, y1data, y2data, ..)  opt titles;
```

where the latter form gives additional titles for each y-data set.

Plot Options

```
gplot::xtics gp list_of_tic_labels;
```

Sets the tic labels of the x-axis to the given text labels. The labels can be given aas a simple list of strings, which are taken as successive labels or as a list of tuples with the form `(value, label)`, in which case each label is placed at its `value` position.

`gplot::xtics gp ()` or `gplot::xtics gp "default";`

This restores the default tics on the y-axis.

`gplot::title t;`

Sets a title string on top of the plot (default location)

`gplot::output gp terminal name;`

Sets the terminal and output name for the successive plots. For some terminal additional options might be given:

`gplot::output gp (terminal, options) name`.

For terminals like x11 or windows, name can be empty `()`.

`gplot::xlabel gp name` or `gplot::ylabel gp name`

Adds labels to the x- or y-axis, respectively. An empty name removes the label for successive plots, e.g., `gplot::xlabel gp ""`.

Private Functions

`gpdata data`, `gpxydata (xdata, y1data, ..)`

Internal functions to handle lists of data point (`gpdata`) or tuples of lists of data points (`gpxydata`) and convert them to be understood by Gnuplot.

`gpxycmd`, `gpxycmdtitle`

Internal function to generate the plotting command for multiple datasets. `gpxycmdtitle` adds titles to each dataset, a.k.a plot legend.

`gplot::gpopt ("style", style, args);`

Internal function to convert a plot style to the respective gnuplot syntax

`gplot::gptitle t;`

Internal function to generate title information for individual datasets