

plotspectrum

Benjamin Bulheller

September 7, 2008

Contents

Introduction	2
Installation	3
Input, Output, Command and Configuration Files	3
Title and Axes Labels	5
<i>x</i> and <i>y</i> Ranges	6
3D Plots	6
Line Styles and Graph Types	6
Curve Smoothing	8
Plotting Specific Columns	8
Legend or Key	9

Introduction

gnuplot^[2] is an extremely powerful interactive data and function plotting utility. Apart from being cross-platform (available for virtually every operating system out there) and command-line driven it is scriptable and supports all conceivable types of graphs and functions.

`plotspectrum` was started to quickly plot circular and linear dichroism spectra (CD, LD) from *xy*-data during a computational chemistry PhD project. One additional option here, another one there, ... and it basically grew into a gnuplot front-end which can be either used to create the rough gnuplot command file in order to edit and perfect afterwards and also comes in handy in scripts when dozens or hundreds of plots need to be produced. It offers lots of command line options to produce quite fine-tunable 2D and 3D plots from one or multiple *xy*-data files using just one (admittedly quite long) command line.

The powers of gnuplot clearly go beyond the possibility of using it via the command line only and it is *strongly* recommended to become familiar with the syntax and commands of gnuplot command files to harness these possibilities. `plotspectrum` apart from conveniently plotting spectra from the command line also produces the gnuplot command file which makes it possible to further improve the plots and to learn the syntax and commands this way.

When the number of command line options was found to be headed towards 50 and the usage was spread over two screens this manual was started. Each section generally starts with the usage output of the script, that is all parameters needed for this section. Each parameter is then explained in detail.

The general usage of `plotspectrum` is

```
plotspectrum file1 [file2] [...] [options]
```

where `fileX` contains the *xy*-data in tab- or spaces-separated columns like this:

#	x-values	y-values
	441.600000	0.038725
	441.700000	0.037447
	441.800000	0.036205
	441.900000	0.035000
	442.000000	0.033831

Comment lines are possible and have to start with a hash-symbol (#). One or multiple files can be given and are all plotted in the same plot using different line styles by default. `[options]` represents one or more of the various command line parameters which will be discussed in detail.

Installation

The script was written in Perl which is an interpreted language and therefore does not require a compilation. Linux and Mac OS have Perl automatically installed, Windows users need to install ActivePerl^[1] and have to do some adjustments (e.g. the path to the Perl executable in the first line of the script). However, the script has not been tested under Windows and may not run especially due to possible differences when running gnuplot via the command line.

Under Linux and Mac OS the file needs to be made executable:

```
chmod +x plotspectrum
```

It should then be copied into a folder which is included in the `PATH` so that it can be executed from anywhere in the system without always typing the absolute location of the script. A common folder for user scripts is `~/bin` for example.

The script requires several Perl libraries from which it uses routines. In particular these are

- `GetParameters.pm` to parse the command line parameters
- `ReadSpectrum.pm` to read in xy data
- `GetBaseName.pm` to split a filename into base name and extension
- `DebugPrint.pm` which comes in handy during debugging

These libraries need to be either in one of Perl's library directories, `~/bin/perl/lib` or in the directory of `plotspectrum` itself.

If the script is found and no problems occur with the required Perl libraries, issuing the command `plotspectrum` should display the usage information.

Input, Output, Command and Configuration Files

```
plotspectrum file1 [file2] [...] [options]
```

<code>file1</code>	the extension <code>.cd</code> or <code>.ld</code> is added if omitted
<code>-cmd</code>	keep the gnuplot command file for later changes
<code>-cfg</code>	create a <code>.cfg</code> file with the used command line
<code>-pdf</code>	create a pdf from the postscript
<code>-gif</code>	create a GIF file instead of postscript (automatically, if <code>.gif</code> extension is given with <code>-o</code>)

```

-res          defines the gif resolution xres yres (default 1024x768)

-o           define a file name of the output file
-p           preview mode: open gnuplot instead of writing to a file
-s           define the size of the plot (default is $Options->{s})

```

One or more input files (`file1`, `file2`, etc.) can be given and will all be plotted in the same plot with different line styles. The files need to contain *xy* data, that is two columns with numbers separated by tabs or spaces. Comment lines are possible starting with a hash symbol (`#`). Every file is checked for existence and if not found the extensions `.cd` and `.ld` are tested. That is, these extensions may be omitted on the command line.

If the option `-cmd` is given, the script also creates the gnuplot command file which was used to create the output. This is very handy to use `plotspectrum` to create the rough plot and then fine-tune in the gnuplot command file.

Another way to save the information of how the plot was created (for later re-use) is via a configuration file. This is requested by `-cfg` basically creates a shell script (already executable) with the command line that was used. The file name is the base name of the output file with the extension `.cfg`.

Via `-o outfile` an output file can be defined (the extension `.ps` is automatically added if it is omitted). If `-o` is not given then the base name of the first input file is used (output to a postscript files is default). To output to a GIF file, the parameter `-gif` can be given. If an output file with `.gif` extension is defined the `-gif` option can actually be omitted. By default the resolution is 1024x768, this can be changed via `-res xres,yres`, e.g. `-res 2048,1536` (mind that this must be one string, that is no blank between the numbers and the comma).

The parameter `-pdf` first creates a postscript file (gnuplot cannot directly produce PDF) and then runs `pstopdf` on it to create the PDF. The name of this command is depending on the OS and \LaTeX distribution and may also be `ps2pdf` and therefore might need to be changed in the source.

In order not to write to postscript or GIF, the option `-p` (preview) can be given which uses the X11 terminal to show the plot. Mind that each of the three terminals displays the linestyles differently.

The parameter `-s` sets the size of the the plot. By default the size is 1,1, this can be changed via `-size x,y`, e.g. `-size 1.2,0.5` (mind that this must be one string, that is no blank between the numbers and the comma).

Title and Axes Labels

<code>-t</code>	specify a title for the plot
<code>-a</code>	use absorbance units for y-axis
<code>-e</code>	use ellipticity units for y-axis
<code>-ld</code>	use LD in absorbance units for the y-axis
<code>-dr</code>	use the dichroic ratio in absorbance units for the y-axis
<code>-ldr</code>	use the reduced LD in absorbance units for the y-axis
<code>-w</code>	use wavelength units for x-axis
<code>-xlabel</code>	specify a label for the x axis
<code>-ylabel</code>	specify a label for the y axis

Via `-t` "Plot Title" it is possible to define a title for the plot. All fancy gnuplot moves may certainly be used, e.g. for using subscript, superscript or symbols like Greek characters.

The axes labels can be freely configured via the command line parameters `-xlabel` and `-ylabel`. Since `plotspectrum` was developed for plotting circular and linear dichroism spectra (and actually spectra in general), there are a few default specifically for these types of files:

- `-w`: use wavelengths for the x -axis (default for CD, LD and absorbance spectra)
→ wavelength λ / nm
- `-wn`: use wavenumbers for the x -axis
→ wavenumbers cm^{-1}
- `-a`: use absorbance units for the y -axis (this is default for `.abl` and `.ab` files)
→ absorbance / $\text{mol}^{-1} \text{dm}^3 \text{cm}^{-1}$
- `-e`: use ellipticity units for the y -axis (this is default for `.cdl` and `.cd` files)
→ $[\theta]$ / $\text{deg cm}^2 \text{dmol}^{-1}$
- `-ld`: for linear dichroism spectra in absorbance units (default for `.ld` files)
→ LD / $\text{mol}^{-1} \text{dm}^3 \text{cm}^{-1}$
- `-dr`: for plotting the dichroic ratio in absorbance units (default for `.dr` files)
→ Dichroic Ratio / $\text{mol}^{-1} \text{dm}^3 \text{cm}^{-1}$
- `-ldr`: for plotting the reduced linear dichroism in absorbance units (default for `.ldr` files)
→ LD_r / $\text{mol}^{-1} \text{dm}^3 \text{cm}^{-1}$

x and *y* Ranges

<code>-x</code>	the <i>x</i> -range, e.g. 150 250, default is <code>\$xMin \$xMax</code> for <code>.cd/.ld</code> files
<code>-xtics</code>	the distance of the tics on the <i>x</i> -axis
<code>-mxtics</code>	defines the minor tics on the <i>x</i> -axis
<code>-noxtics</code>	disable tics of the <i>x</i> axis
<code>-y</code>	the <i>y</i> -range, e.g. -1000 1000 or 1000
<code>-ytics</code>	the distance of the tics on the <i>y</i> -axis
<code>-mytics</code>	defined the minor tics on the <i>y</i> -axis
<code>-noytics</code>	disable tics of the <i>y</i> axis

If no parameters concerning the *x* and *y* axes are given, the script determines a lot by itself. Due to its history to be used for CD and LD spectra the default for the *x*-range is 150–250. These are defined at the very beginning of the script under **Configuration Variables** and it is recommended to change these and other default values to one's needs.

However, while checking for the existence of each file, the actual minimum and maximum *x* values of all files are determined. If these do not fall in between the default *x*-range it is automatically adjusted so that the data of all files is actually shown in the plots.

To define an *x*-range, two numbers must be given to the `-x` option. For the *y*-range (due to the history of the script), however, it is possible to provide only one parameter which will then be used to define the range using its negative and positive value. That is, `-y 1000` will produce a *y*-range of -1000 .. 1000.

The main and minor tics on the axes (labelled tics and unlabelled tics in between) can be set using the `-?tics` and `m?tics` options, respectively. Otherwise the script tries to determine a reasonable tic scale by itself.

3D Plots

The basic difference between plotting 2D or 3D is using the `plot` or `splot` command, respectively. By default the script will produce 2D graphs. For 3D plots add the `-splot` option which selects the respective command in `gnuplot`.

It is often necessary to define the columns to be used for 3D plots, see further down for how to use the `-using` option to achieve this (page 8).

Line Styles and Graph Types

<code>-lines</code>	plot a line spectrum instead of a curve (used automatically for <code>.cdl</code> and <code>.abl</code>)
<code>-impulses</code>	same as <code>"-lines"</code>
<code>-points</code>	use points for plotting
<code>-dots</code>	use dots for plotting
<code>-lw</code>	define the line width (default 4)
<code>-nocolor</code>	do not use colors (depends on the terminal, i.e. X11 will still be colored)
<code>-solid</code>	use only solid lines
<code>-ls</code>	A list of numbers from 1 to 15 which defines the line style (changeable in the source). The first style is assigned to the first spectrum and so on. If a style is marked with an asterisk (<code>-ls 2*</code>), it is assigned to all other graphs, for which no style is given. If the style contains points, <code>"lines"</code> is automatically switched to <code>"linespoints"</code>
<code>-every</code>	print only every nth data point of all graphs .
<code>-pd</code>	point distance: default is 5, it detects too many datapoints per x-value automatically if <code>linespoints</code> are used and sets <code>every</code> to the given value for that graph

The term “lines” is a bit confusing since gnuplot uses “impulses” for line spectra and “lines” for curve spectra. However, since usually in science it is differentiated between line- and curve-spectra, both `-lines` and `-impulses` lead to the same effect of producing a line spectrum (i.e. using “impulses” to plot). To use points to plot, the `-points` option can be given and accordingly `-dots` in order to use dots instead.

Several parameters allow the configuration of the line style used to plot the graphs. `-lw` selects the line width while `-nocolor` uses grey and black only in the postscript files (X11 preview will still show colors). `-solid` disables the use of dashed and dotted lines for postscript output.

gnuplot uses predefined line styles when multiple graphs are plotted without specifically defining the used style. `plotspectrum` uses style which are only subtly different from the original definitions but can be easily changed to personal liking. The styles are defined in the subroutine `CreateCommandFile` and are certainly also written to the gnuplot file (if `-cmd` was given) to make it easy to change the line styles in one place.

Just as gnuplot the line styles are applied one after another to the different graphs. Using the `-ls` option it is possible to select a specific line style for the individual graphs:

```
plotspectrum file1 file2 file3 -ls 2 5 8
```

The latter will assign the line styles 2, 5 and 8 in this order to the given files. If more files are given than line styles are assigned, `plotspectrum` uses its normal routine

and assigns styles in a consecutive way to the files. If it is necessary to make some of the curves stand out from all others it is possible to override this auto-assignment and define a default style:

```
plotspectrum file1 file2 file3 unimp1 unimp2 unimp3 -ls 2 5 8 1*
```

For the first three files a specific style is assigned while for all other files the line style 1 is selected. `plotspectrum` automatically takes care of selecting the appropriate graph type, that is “lines” if the line style does not define a point style and “linespoints” if a point style definition is found.

The gnuplot “linespoints” styles are very important if colors and solid/dotted/dashed styles are not enough to differentiate between curves (mainly in the legend which states which curve is which). However, if the spectra are of a very high “resolution”, like every 0.1 units on the x -scale, one may be left with a curve made up only of the points, with no space in between them. Two parameters allow dealing with such cases. `-every` sets the respective gnuplot option with the same keyword, plotting only every n^{th} value of the graph.

However, `-every` also affects all other graphs and may result in low resolutions of curves which would look much smoother and graphs of different resolutions may require different settings for `every`. A better solution is offered by the option `-pd` (the point distance). On the one hand it affects only curves which actually use linespoints in the first place and the value of `every` is determined for each graph individually.

If the graph is rather coarse and the point distance per x -unit is greater than the user-defined value, the setting is ignored and all points are plotted. Otherwise too little points would be used and would even falsify the output in the worst case.

Curve Smoothing

Five curve smoothing algorithms provided by gnuplot can be triggered using different options. This only works only globally for the plot, though, that is if the option `-bezier` is given then it will be used for all the plotted curves.

<code>-csplines</code>	use splines to connect the points smoothly
<code>-acsplines</code>	use weighted splines to connect the points smoothly (third column with weighting factors needed)
<code>-bezier</code>	use bezier curves to connect the points smoothly
<code>-sbezier</code>	use sbezier curves to connect the points smoothly
<code>-unique</code>	use the 'unique' fitting routine to smooth curves

Plotting Specific Columns

By default `plotspectrum` will always plot the first two columns because `gnuplot` uses these even if more columns are present. With the `-using` option it is possible to plot specific columns of an input file, the given settings are directly added to the `using` directive of the `plot` command. This only works for “plain” column numbers (counting starts at 1) like `-using 1:3` to plot columns 1 and 3. In `gnuplot` itself settings like `1:($2+$3)` are also possible but such definitions have to be added to the command file after creating it using `-cmd`.

For 3D plots using the `-splot` option it is certainly possible to define three columns like `-splot -using 1:4:5`.

The `-using` option works like `-ls` and reads in a list of settings which are applied to the files in the given order. To plot e.g. different columns of the same file one would plot it “twice” with different settings:

```
plotspectrum file1 file1 -using 1:2 1:3
```

A trailing asterisk marks a setting as default value:

```
plotspectrum file1 file2 file3 file4 -using 1:2* 1:3
```

This would plot `file2` using the first and third column and all others with the first and second.

Legend or Key

<code>-nokey</code>	do not display the key (legend)
<code>-k</code>	keys for each graph used in the legend (order matters!)
<code>-ktop</code>	place key at the top
<code>-kbottom</code>	place key at the bottom
<code>-kbelow</code>	place key below the plot
<code>-kinside</code>	place key inside the plot
<code>-koutside</code>	place key outside the plot
<code>-kreverse</code>	place line sample to the left of the keys
<code>-knoreverse</code>	place line sample to the right of the keys
<code>-kleft</code>	place key left
<code>-kright</code>	place key right
<code>-kLeft</code>	justify key labels left
<code>-kRight</code>	justify key labels right
<code>-ksamplen</code>	define the length of the samples in the key (default 2)
<code>-kspacing</code>	define the spacing of the samples in the key (default 1.25)

The above commands allow a configuration of the legend (or “key” in gnuplot jargon). Most of the parameters should be self-explanatory. The option `-k` takes a list of strings (use quotation marks if blanks are needed within them) which are assigned as keys to the given files in that order:

```
plotspectrum file1 file2 file3 -k "file 1" "file 2" "the last file"
```

Mind that the keys are assigned to the files in the same order as given. If no key titles are given, the base names of the files are used instead.

References

- [1] ActivePerl. <http://www.activeperl.com>.
- [2] gnuplot. <http://www.gnuplot.org>.