

The GPlotFun function package for gretl

Artur Tarassow

Version 0.2

Changelog

- Version 0.2 (May 2023)
 - Add support for frequency plot
- Version 0.1 (Feb, 2019)
 - initial version

Contents

1	Introduction	2
2	Get the package	2
3	Examples	2
3.1	Time-series plot	3
3.2	Scatter plot	4
3.3	Stacked barplot	5
3.4	Heatmap	6
3.5	Frequency plot	7
4	The GPlotFun() function and plotting types	7
4.1	The “standard” plotting option	7
4.2	The “scatter” plotting option	8
4.3	The “stackedbar” plotting option	8
4.4	The “heatmap” plotting option	9
4.5	The “frequency” plotting option	9
5	Plotting settings	9
5.1	Font related	9
5.2	linetype and linewidth	9
5.3	Color format, transparency, grid and borders	10
5.4	Date format, scales and xticks	10

5.5	Legend and labels	11
5.6	Output format	11

1 Introduction

The **GPlotFun** package is a collection of gretl scripts and functions for advanced plotting methods. Gretl’s built-in plotting facility makes heavy use of the well-known open-source graphing utility *Gnuplot* (URL: <http://www.gnuplot.info/>). Gretl already offers a plotting facility, as described in Ch. 8 in the manual. However, some types of plots are either hard to realize using the current state or not supported at all. This package is an attempt to bridge the gap between Gretl’s current state of employing gnuplot and gnuplot’s huge capability to draw very different and advanced plots.

The **GPlotFun** package makes heavily use of gretl’s support of bundles which offer a highly flexible way of programming. This initial package version is a suggestion, and the author is looking forward to collaborate with interested people to continue its development.

The **GPlotFun** package comprises the following features:

- Time-series plots with vertical and horizontal lines, with vertical and horizontal shaded areas, and impulse as well as point plots.
- Scatter plots supporting classes of points and lines.
- Stacked barplots combined with line plots.
- Heatmaps
- Frequency plot
- Many parameters can be controlled for obtaining high-quality graphs.

In the following sections, we will inform the interested user or potential supporter where to download the package. Next, we will introduce some examples and show how to control the output before a more detailed overview about the various options will be shown.

2 Get the package

The **GPlotFun** package is currently only available on the author’s github repository: <https://github.com/atecon/GPlotFun/tree/feature/progress>. Interested people can find the functions and the manual in Lyx- and pdf-format there. The author seeks to collaborate on this project with other people. Github provides a professional environment for joint developments by means of branches and pull requests.

3 Examples

For illustration we use a time-series data set comprising annual macroeconomic data of the UK between 1873 and 1991. We will show how to draw simple as well as more advanced different types of plots using the package.

3.1 Time-series plot

The sample script opens the sample data set and plots two series of interest over time. The package's functions are accessed through bundles which are initially filled with some minimum information before passed to the main function `GPlotFun()`. In the example, we set up an empty bundle `b` and define a $T \times 2$ matrix named `lines` with two series. Furthermore, we pass a matrix named `obsdate`. A matrix named like this is assumed holding timestamps which will be converted into date strings and which are printed at the x-axis of the graph. If no such a matrix is passed, the x-axis will be a sequence of integers starting at '1'.

Per default, the graph will be immediately depicted on the screen but of course the user can store it in various formats as shown below.

```
open hendry_jae.gdt -q
bundle b = null
b.obsdate = {$obsdate}          # access timestamp
matrix b.lines = {pgdp} ~ {gdp}
GPlotFun(&b)
```

The output will be the plot on the left-hand in Figure 1 which will be directly shown on the screen.

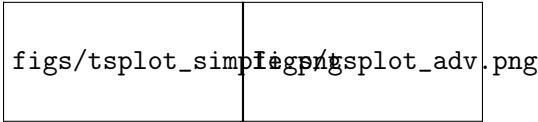


Figure 1: Default time-series plot (left) and time-series plot with advanced settings.

We can easily add further features to the plot. The following script shows how to add a horizontal shaded area around one of the lines.¹ Furthermore, we will add a vertical line at some specific dates, add a recession bar (vertical shaded area) and some point as well as impulse plot. Lastly, we will control some of the labels and change some further details. These commands result in the right-hand side plot of Figure 1.

¹At the moment only a single horizontal shaded area can be drawn.

```

bundle b = null
b.obsdate = {$obsdate}          # access timestamp
matrix b.lines = {pgdp}

# shaded area around gdp observations
b.hfilled = {gdp-1} ~ {gdp+1}

# recession bar for some period
b.vfilled = {obsnum(1900)} ~ {obsnum(1905)}
b.vfilled_alpha = 0.3           # set transparency (optional)

# vertical line at two observations
b.vlines = {obsnum(1917)} ~ {obsnum(1945)}
b.vlines_dt = seq(1,2)         # set line type (solid/ dashed)

# points on rhs-axis
b.points_rhs = {diff(pgdp)/pgdp(-1)*100}

# impulses
b.impulses = {diff(gdp)/gdp(-1)*100}

# restrict range of y-axis
b.ymin = -1
b.ymax = 18
# change dimension of the plot
b.width = 800
b.height = 420

# add legend names and set labels + title
b.StrLeg = defarray("1", "2", "3 (rhs)")
b.title = "This is a nice plot..."
b.ylabel = "y-axis"

GPlotFun(&b)

```

3.2 Scatter plot

The following script creates a scatterplot with points and lines using the `which='scatter'` option. The default `which`-Parameter is `which='standard'` which calls the type of plot as presented in the previous section.

In the example, we will plot two distinguished pairs of points. First, we plot the `pgdp` vs. the `gdp` series and also the `pw` vs. `gdp` series, respectively by passing the `points` matrix. This matrix holds the values for the x-axis (this is only the case for “scatter”-plots) in the first column while the remaining columns refer to variables which are mapped onto the y-axis. Similarly, the `lines` matrix also holds in the first column the values plotted on the x-axis followed by the series mapped onto the y-axis.

We also add a y-label and control the point size by setting the entries in vector `ps` setting the

point size for each of the $k - 1$ columns of matrix **points**. The specific name **legend_str** refers to a string array holding a description of the separate lines followed by the description of the points (exactly in this order). Per default, this string array is empty.

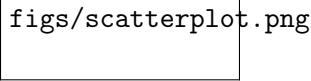
```
bundle b = null
b.points = {gdp} ~ {pgdp} ~ {pw}           # 1st col. refers to x-axis values
b.lines = {gdp} ~ {pgdp}                   # 1st col. refers to x-axis values

b.which = "scatter"                         # set scatterplot
b.xlabel = ""
b.ylabel = "pgdp \& pw"

# style
#   b.colors = "jet"                       # set different color scheme.
b.ps = ones(2,1)                           # point size
b.legend_str = defarray("pgdp vs. gdp (line)", "pgdp vs. gdp", "w vs. gdp")
b.fs = 14                                   # font size

GPlotFun(&b)
```

The resulting plot is depicted in Figure 2.



figs/scatterplot.png

Figure 2: Scatter plot with points and lines.

3.3 Stacked barplot

The stacked barplot is another type of plot. This kind of graph can be combined with horizontal lines and is called by the **which="stackedbar"** parameter. The following sample script shows how to setup such a graph. We define a matrix named **barstacked** holding each series as a column vector before passing another matrix named **lines** holding series which are plotted as horizontal lines. Again the user can plot the values against time if the **obsdate** matrix is passed to the bundle.

```

bundle b = null
b.barstacked = {gdp} ~ {pgdp} ~ {pw}      # rows: x-axis; cols: y-axis
b.obsdate = {$obsdate}
b.lines = {m}
b.xmin = 0                                # starting value at x-axis

# add legend
b.legend_str = defarray("gdp", "pgdp", "pw", "m (line)")

# control output
b.width = 800
b.height = 420
b.which = "stackedbar"
GPlotFun(&b)

```

The resulting plot is depicted in Figure 3. The pattern of the stacked bars cannot be controlled by the user at the moment.

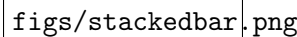


Figure 3: Stacked barplot with horizontal line

3.4 Heatmap

The following example shows how to plot a heatmap which can be a useful tool for visualizing complex information. Heatmaps are set by the `which='heatmap'` parameter. We show in the following script how to store the output as a png-file.²

```

bundle b = null
b.xmin = 0                                # minimum x-value, optional
b.ymin = 0                                # minimum y-value, optional
b.width = 800
b.height = 420
b.title = "This is a heatmap"

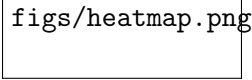
list L = 1 2 3 4 5 6 7 8 9 10 11 12      # list of series
b.heat = {L}'                            #alternatively: mnormal(30,100)
b.obsdate = {$obsdate}

b.which = "heatmap"
string fname = sprintf("%s/heatmap.png", $dotdir)    # "display" doesn't work, yet
b.fname = fname                                    # store plot
GPlotFun(&b)

```

²Unfortunately, the user has to save the plot as the direct output on the screen using the “display” option does not work for this kind of plot.

We pass a list of 12 series to a matrix named **heat**. Row values of the **heat** matrix are plotted on the y-axis and column values at the y-axis. Optionally, we also pass the **obsdate** matrix for printing date strings at the x-axis.



figs/heatmap.png

Figure 4: Heatmap plot.

3.5 Frequency plot

The following example shows how to plot a frequency plot. While gretl has the built-in command **freq** for drawing a frequency plot, it cannot really be tweaked at the moment. The plot can be called by the **which='frequency'** parameter. We show in the following script how to store the output as a png-file.

```
bundle b = null
b.which = "frequency"
b.data = {pgdp} # single column vector
b.data[2,1] = NA
b.title = "This is a frequency plot"
string b.fname = sprintf("frequency_plot.png")
GPlotFun(&b)
```

4 The GPlotFun() function and plotting types

The main vehicle is the GPlotFun() function through which the user can set and modify various plots.

```
GPlotFun(bundle *b)
```

Return type: void

The function argument is a single bundle element which must include some basic information.

4.1 The “standard” plotting option

The most basic type of plotting option is the “standard” option — illustrated in section 3.1. This option covers various styles such as (i) lines, (ii) lineplots, (iii) impulses, (iv) vertical bars and (v) horizontally shaded areas.

In order to plot numerical values in a certain style, the user has to pass specifically named matrices to GPlotFun() as shown in Table 1. This is a flexible framework which could be easily extended in future.

As can be seen, currently only horizontal lines and impulses can be drawn on the right-hand side y-axis.

Style	Lines(-plots) (lhs)	Lines(-plots) (rhs)	Vert. lines (lhs)	Impulses	Impulses (rhs)
Matrix name	lines	lines_rhs	vlines	impulses	impulses_rhs
Dimension	$T \times k$	$T \times k$	$1 \times k$	$T \times k$	$T \times k$
Style	Horiz. filled area	Vert. filled area			
Matrix name	hfilled	vfilled			
Dimension	$T \times 2$	$k \times 2$			

Table 1: Overview style options and name of respective matrices for the “standard” plotting option.

Style	Lines(-plots) (lhs)	Lines(-plots) (lrhs)	Points (lhs)	Points (rhs)
Matrix name	lines	lines_rhs	points	points_rhs
Dimension	$T \times (1 + k)$	$T \times (1 + k)$	$T \times (1 + k)$	$T \times (1 + k)$

Table 2: Overview style options and name of respective matrices for the “scatter” plotting option.

- The matrices **lines**, **lines_rhs**, **impulses** and **impulses_rhs** are of dimension $T \times k$ holding x-axis values as rows and y-axis values as columns.
- Information for drawing vertical lines are put into a $1 \times k$ vector named **vlines** where each element refers to a specific observation number, e.g. if ‘year’ 1971 is the 51th observation one writes the value ‘51’ into one of the k columns.
- At the moment one can only draw a single horizontal shaded area. Matrix **hfilled** holds on the first (second) column the lower (upper) bound of the shaded area while the rows refer to some x-axis value.
- Matrix **vfilled** is a $k \times 2$ matrix holding the starting and end point for each of the k vertically shaded areas in the column 1 and 2, respectively. Again, if one wants to draw a shaded area between the 10th and 15th observation (drawn at on x-axis), one constructs a matrix such as **b.vfilled** = {10,15}.

4.2 The “scatter” plotting option

The ‘scatter’ option allows plotting scatter plots, and covers (i) lines as well as (ii) points. The user has to pass specifically named matrices to **GPlotFun()** as shown in Table 2.

For each of these matrices, the first column is always assumed to hold the x-axis values such that drawing $k = 2$ lines or distinguished point plots, the respective matrix must be of size $T \times (1 + 2)$.

4.3 The “stackedbar” plotting option

The “stackedbar” option currently covers three types of plots: (i) horizontal lines, (ii) stacked bars (in gnuplot terms these are ‘row-stacked histograms’) and (iii) vertical lines. Again the naming convention of the matrices one has to pass is as shown in Table 3.

Each column of the respective matrix specifies another series/ variable drawn for each of the T rows. For the **vstacked** matrix, the first column specifies the ‘base’-bar while the remaining series

Style	Lines(-plots) (rhs)	Lines(-plots) (lhs)	Stacked bars (rhs)	Vertical lines
Matrix name	lines	lines_rhs	vstacked	vlines
Dimension	$T \times k$	$T \times k$	$T \times k$	$1 \times k$

Table 3: Overview style options and name of respective matrices for the “stackedbar” plotting option.

are stacked on top of the first series for each of the T observations. Vector **vlines**, however, denotes the observation number for which a vertical lines is drawn (**obsnum()** gretl equivalent).

4.4 The “heatmap” plotting option

The “heatmap” option currently only supports a single style namely the heatmap. Again the naming convention of the matrices one has to pass is as shown in Table 4.

Style	heatmap
Matrix name	heat
Dimension	$k \times T$

Table 4: Overview style options and name of respective matrices for the “heatmap” plotting option.

Values of the $k \times T$ **heat** matrix are plotted exactly in this dimension meaning that rows are put on the y-axis and columns on the x-axis.

4.5 The “frequency” plotting option

Called by the **which** parameter “frequency”. Again the naming convention of the matrices one has to pass is as shown in Table 5.

Style	frequency
Matrix name	data
Dimension	$T \times 1$

Table 5: Overview style options and name of respective matrices for the “frequency” plotting option.

5 Plotting settings

5.1 Font related

Options for setting the font style and font size are available (see Table 6).

5.2 linetype and linewidth

Table 7 reports the options for controlling the linewidth for different styles. Further control parameters are shown in Table 8.

Option	
fstyle	string, font style applied to whole plot (default: <i>serif</i>)
fs	scalar, font size applied to all elements of the plot (default: 12)
vlines_fs	scalar, font size for optional labels of vertical lines (default: 10)

Table 6: Overview of font related options.

Option	
lines_lw	matrix, specify line width of lhs lines (default: 1 for 12 lines)
lines_rhs_lw	matrix, specify line width of rhs lines (default: 1 for 12 lines)
vlines_lw	matrix, specify line width of lhs vertical lines (default: 1 for 12 lines)
impulses_lw	matrix, specify line width of lhs impulses (default: 1 for 12 impulses)
impulses_rhs_lw	matrix, specify line width of rhs impulses (default: 1 for 12 impulses)
vlines_lw	matrix, specify line width of rhs vertical lines (default: 1 for 12 lines)

Table 7: Overview of linewidth-related options.

5.3 Color format, transparency, grid and borders

Table 9 shows the various control parameters for specifying color for lines, point etc., controlling transparency of some of the elements and so on.

For colors the user can either choose among 3 pre-defined color pallets³ (not for heatmaps though!)

1. `dark2`
2. `greys`
3. `jet`

by setting the color parameter e.g. to `b.color = "jet"`, or set a string array as for instance by `b.colors = defarray("black", "blue")`.

5.4 Date format, scales and xticks

Table 10 shows the various parameter for controlling date strings, scales and the axis.

Once the user has passed an `obsdate` vector to the bundle, one can select between the following 5 different formats for showing date strings on the x-axis by setting `dateform`:

- 0= YYYY/MM/DD
- 1= YY/MM/DD
- 2= YYYY/MM
- 3= YY/MM
- 4= MM/DD
- 5= MM/YY

³These color pallets are taken from <https://github.com/Gnuplotting/gnuplot-palettes/>.

Option	
ps	matrix, specify point size of points or lines (default: 0 for 12 lines)
pt	matrix, specify point size of points or lines (default: 1 to 6 for 12 lines)
lines_pi	scalar, specify that only the pival-th point of the plot is given a symbol
lines_dt	matrix, set dash type for lines (default: solid lines (=1) for 12 lines)
vlines_dt	matrix, set dash type for vertical lines (default: solid lines (=1) for 12 lines)
pat	matrix, pattern styles for stacked bars (default {1,4,2,3,8})

Table 8: Overview of line-/ point-types-related options.

Option	
colors	string array, specify line/ point color either as RGB color (default: dark2_pal palette)
vcolors	string array, specify vertical line color either as RGB color (default: jet_pal palette)
vfilled_alpha	scalar, specify the degree of transparency of vertical shaded areas (default: 0.3)
hfilled_alpha	scalar, specify the degree of transparency of shaded shaded areas (default: 0.9)
grid	bool, plot grid (default: on (1))

Table 9: Overview of color-related options.

5.5 Legend and labels

Table 11 lists legend-related options.

The legend can be put at four different positions through the **legend_pos** option:

- “outside below horizontal” = 1
- “top left” = 2
- “top right” = 3
- “bottom left” = 4
- “bottom right” = 5

5.6 Output format

Table 12 shows the options for specifying the size of the plot and the file name plus format. Per default, the option **fname** is “display” which immediately shows the plot on the screen.

Figures can be stored on various formats such as:

1. png: “*.png”
2. pdf: “*.pdf”
3. eps: “*.eps”
4. svg: “*-svg”

Option	
obsdate	series, YYYYMMDD ISO 8601 “basic” date format (optional)
dateform	int, in case obsdate is passed, specify date format on the x-axis (default: 1)
skipx	int, number of entries on the x-axis skipped before the next one shown (default: 4)
ymin	scalar, specify minimum value on lhs y-axis (optional)
ymax	scalar, specify maximum value on lhs y-axis (optional)

Table 10: Overview of legend-related options.

Option	
legend_show	bool, show legend/ key (default: true(=1))
legend_pos	integer, choose where to put legend (default: 1=outside below horizontal)
legend_str	string array, set label for each object (lines, points, etc.) (default: empty)
nocolorbox	integer, select whether to show the colorbox of a heatmap or not (default: 1 show)
title	string, specify title (optional)
xlabel	string, specify xlabel (optional)
ylabel	string, specify lhs ylabel (optional)
ylabel_rhs	string, specify rhs ylabel (optional)
add_vline_label	bool, show number label (consecutive numbering) for vertical lines (default: true(=1))
vlines_shift_fraction	scalar, shift parameter number labels of vertical lines (negative: DOWN, default=0.0)

Table 11: Overview of legend-/label-related options.

Option	
height	scalar, specify height of the plot (default: 480)
width	scalar, specify width of the plot (default: 640)
fname	string, path + filename + format of the plot (default: “display” for immediate plot on the screen)

Table 12: Overview of output-related options.