Gaston

A plotting utility for Julia

v. 0.3

M. Bazdresch

May 8, 2012

Please note: Gaston is currently under development, and all functions and definitions are subject to change from one version to the next, as we figure out the best way to organize the code. Gaston has been tested on Linux (Ubuntu 10.04 and Arch), with gnuplot 4.6 and WxWindows. It only supports the wxt terminal.

1 Introduction

Gaston provides a way to plot scientific data using the Julia programming language. It accomplishes this by harnessing gnuplot, a versatile and time-tested plotting utility. Gaston also relies on gnuplot for interacting with plots (zooming and rotating a plot with the mouse, for instance).

The primary purpose of Gaston is to provide easy-to-use functions to quickly and conveniently plot the most common kinds of scientific and numeric data. It is concerned with screen output only (although we plan to support printing to a file in a future version), and supports the most common plot configurations only. Tweaking a plot to produce a specific look or producing publication-quality graphics are outside its scope.

2 Installation

To use Gaston, follow this procedure:

1. Save all files `gaston*.jl` somewhere convenient. Then, you may `cd` to that directory and start julia there, or do

   ```
   push(LOAD_PATH, "/path/to/gaston/jl")
   ```

Then, load the program with

```
load(”gaston.jl”)
```

2. To run a demo, do

   ```
   load(”gaston_demo.jl”)
   demo()
   ```

   This will create a series of figures that illustrate the current capabilities of the program. The same file may also serve as a guide on how to create different types of plots.

3 Definitions

A **figure** is an independent window, which contains a set of axes, on which one or more **curves** are plotted. A figure may contain **labels** (for instance, on each coordinate axis), a **title**, and a **legend box** which identifies each curve. Gaston supports having any number of figures open at the same time; however, gnuplot requires that only one figure is able to offer mouse interactivity at a given time. Each figure is identified by a unique **handle**. Handles are natural numbers.

A **curve** is defined by a set of coordinates. Two-dimensional curves have **x** and **y** coordinates; in three dimensions, an additional **z** coordinate must be specified. A curve also has several **properties** that specify a plotting configuration (for instance, it may have a **plotstyle**, a **linewidth**, a **linecolor**, etc), which define how the curve is to be plotted.

4 Plotting

4.1 Figures

A figure may be created by the function `figure()`. Many plotting functions create or reuse an existing figure, as needed (see each function's documentation). Called with no arguments, `figure()` creates a new figure with the smallest available handle. Called with an argument (which must be a natural number), it will create a figure with that handle if no such figure exists, or will select it (make it current) if it exists.

Selecting an existing figure may be useful, for instance, to make it mouse and keyboard interactive.

Handles may be created in any numerical order.

This function always returns the handle of the current figure.

4.2 Terminals

Gaston supports plotting to the screen as well as printing the plots to files. Two screen terminals are supported: `wxt` and `x11`. The `x11` terminal is provided for use in systems that don't support WxWindows.

For printint to files, terminals `svg` and `gif` are supported. For more details on printing, see section 5.4 below.

To set the terminal type, use the command

```
set_terminal(term)
```

where `term` is a string.

4.3 2-D plotting

There are two commands for two-dimensional plotting: `plot()` and `histogram()`.

4.3.1 plot()

The `plot()` function takes any number of arguments, with the following format:

```
plot({h,} {x,} y {, property, value,...} {...})
```

- If the optional argument `h` is provided, it is assumed to be the figure handle in which to plot.

  - If the handle doesn't exist, a new figure is created.
  - If it exists, the figure will be overwritten.
  - If it is `0`, then a new figure to plot in will be created, using the next handle available.

  If it is not provided, then the current figure will be used and overwritten.

- The `x` and `y` arguments specifiy coordinates (they must be ranges, vectors, or two-dimensional arrays with a singleton dimension). If only `y` is provided, it is assumed to be the ordinate. If `x` and `y` are provided, they are assumed to be the abscissa and the ordinate, in that order.

- The coordinates may be followed by any number of `property`, `value` pairs. These are used to set the value of any of the curve's or the axes' properties (see section Reference, below). `property`s are always strings. The `value`s must be of the appropriate type.

- The pattern `{x,} y {, property, value,...}` may be repeated any number of times.

  - Curve settings are always set for the immediately preceding curve.

  - Axes settings may be specified at any time, and in the case of repeated properties, the last one set is the one that is used.

- Properties may take the following values: `plotstyle`, `legend`, `color`, `marker`, `linewidth`, `pointsize`, `title`, `xlabel`, `ylabel`, `box`, `axis`.

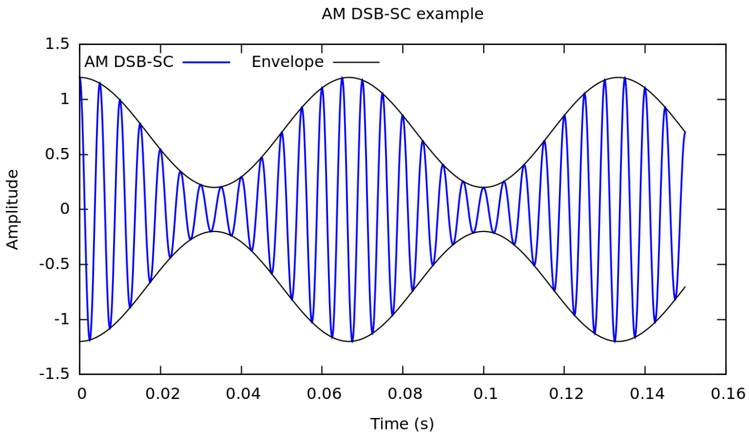
- Supported plotstyles are `lines`, `linespoints`, `points`, `impulses` and `boxes`.

- plot() returns the handle of the figure that was plotted.

As an example, to plot an amplitude modulated signal and its envelope, we may run the following code:

```
t = 0:0.0001:.15
carrier = cos(2pi*t*200)
modulator = 0.7+0.5*cos(2pi*t*15)
am = carrier.*modulator
plot(t,am,"color","blue","legend","AM DSB-SC","linewidth",1.5,
    t,modulator,"color","black","legend","Envelope",
    t,-modulator,"color","black","title","AM DSB-SC example",
    "xlabel","Time (s)","ylabel","Amplitude",
    "box","horizontal top left")
```

which produces this plot:



4.3.2 histogram()

The `histogram()` function plots a single histogram in a figure. It has the following format:

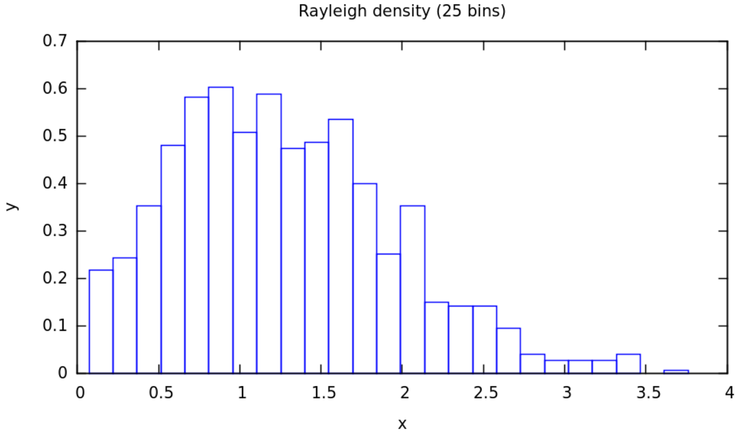
`histogram({h,} y {, "bins", bins} {, "norm", norm} {, property, value,...})`

- The optional argument h has the same meaning as in plot().
- The histogram consists of boxes, where the height of each box is given by the number of elements of y that fall in a given range.
- bins specify the number of bins (boxes) that will be plotted.
- If norm is specified, the histogram will be normalized, so that the area under the histogram is equal to norm.
- The pairs {property, value,...} have the same meaning as in plot().
- Properties may take the following values: legend, color, linewidth, title, xlabel, ylabel, box.
- histogram() returns the handle of the figure that was plotted.

As an example, to plot an approximation of a Rayleigh density, we may run the following code:

```
y = sqrt( randn(1000).^2 + randn(1000).^2 )
histogram(y,"bins",25,"norm",1,"color","blue","title","Rayleigh density (25 bins)")
```

which produces this plot:



4.4 3-D plotting

TBW

4.5 Plotting images

TBW

5 Plotting with mid-level functions

In addition to the plotting functions described in the section above, Gaston offers a "mid-level" set of functions that allow plots to be created step-by-step.

Having this mid-level layer has two benefits. One is that this layer is more flexible and allows direct control over all aspects of the plot. For instance, using high-level functions it is not currently possible to plot more than one histogram, or a histogram and another curve, on the same figure. The mid-level layer allows such combinations. Another example is an algorithm that builds figures step-by-step as data becomes available, instead of waiting until all the data needed has been produced.

A second benefit is that this layer abstracts Gaston's internal graphics representation from the high-level layer. This means Gaston's whole back-end may change without affecting the high-level functions; only the mid-level layer would have to be adapted.

Also, much of Gaston's error checking and argument validation is performed at this layer.

Please note that there is a single mid-level plot command, which is `llplot()`. According to the type of coordinates and plotstyle, it will figure out how to plot.

5.1 2-D plotting

Plotting proceeds in steps:

1. Create or select a figure with `figure(i)`, where `i` is a positive integer.
2. Add a curve (a set of coordinates plus a plot configuration), with `addcoords(x,y,conf)`. Here, `x` and `y` are vectors, and `conf` configures the plot and line styles, markers, legend, color, etc. Repeat this step for each curve you wish to include in the figure.
3. Add a configuration for the entire figure (axis), with `addconf(conf)`, where `conf` contains the figure configuration.
4. Issue the `llplot()` command.

A curve configuration is created as follows:

1. Create a default configuration with `c = CurveConf()`.
2. `c` is a structure, each of whose fields controls one aspect of the curve's configuration. These fields may be set individually. Available fields are: `legend`, `plotstyle`, `color`, `marker`, `linewidth`, and `pointsize`. For instance, to change a curve's color, do[1]

[1] Directly changing a structure's fields, instead of using setter functions, may be seen as non-idiomatic or inelegant. In this case we have decided to do the simplest thing that might possibly work.

```
c.color = "blue"
```

See gnuplot's documentation to see the range of valid options.

A figure (axis) configuration is created as follows:

1. Create a default configuration with `a = AxesConf()`
2. Just as in the case of a curve configuration, `a` is a structure whose fields may be modified. Available fields are: `title`, `xlabel`, `ylabel`, `zlabel`, `box`, `axis`.

Several rules apply:

- You can create as many figures, each with as many curves, as desired.
- Generally, if you don't provide some of the data, it will be inferred. For example, calling `addcoords` with a single vector `y` will assume the `x` coordinate is `1:length(y)` and set up the default plot configuration.
- If you call `addcoords` with matrix arguments, each column will be interpreted as a different plot.
- Calling `addcoords()` will create a new figure if none have been created yet.
- Calling `llplot()` without an axis configuration will just use one by default.
- Gnuplot only provides mouse interaction support for the current figure. To use the mouse in a previously created figure `i`, just issue command `figure(i)`. This will also bring the figure to the front.

5.1.1 2-D plotting styles

Besides simply plotting a set of `x` and `y` coordinates, Gaston supports other kinds of plots.

Error bars and lines. To add error bars or lines, just call `addcoords()` with one or two extra coordinates, and configure the plotstyle accordingly.

Histograms. To plot the histogram of a data vector `data` with `b` bins, first use the auxiliary function `(x,y) = histdata(data,b)` to create `x` and `y` coordinates that may be plotted with the `boxes` plotstyle.

5.2 3-D plotting

The same rules apply, except that `addcoords()` should be called as `addcoords(x,y,Z)`, where `Z` is a matrix whose element `j,k` corresponds to some function of `x[j],y[k]`.

For convenience, a function `Z = meshgrid(x,y,f)` is provided. Called with `x`, `y` coordinates and a function `f`, it will return a matrix that may be used to plot `f`.

5.3 Image plotting

Two image plotstyles are supported: `image` and `rbgimage`. At the moment Gaston requires that a figure contains only one image (and no other curves). This restriction will be removed in future versions.

Scalar image. To plot a matrix `Z` as a figure, use `addcoords([],[],Z)` with empty `x`, `y` coordinates and `Z` as third argument, and set the plotstyle to `image`. Element `(j,k)` of `Z` corresponds to element `(j,k)` in the plot, and its value determines the color at that point.

RGB image. To plot an RGB image, the matrix `Z` must be three dimensional. The first and second dimensions correspond to the value at each point in the image. The third dimension specifies red, green and blue values at each point. Finally, set the plotstyle to `rgbimage`.

5.4 Printing to a file

Gaston supports plotting a figure to a file instead of the screen (*printing* a file). Currently supported are SVG and GIF files. There are two ways to print figures.

5.4.1 Printing a single figure

If you have a figure on screen that you want to print, you have to change the terminal type, specify the filename, and then select the figure. For example, if you want to print the figure with handle 10 to a GIF file named `test.gif`, you would issue these commands:

```
set_terminal("gif")
set_filename("test.gif")
figure(10)
set_terminal("wxt")
```

The final `set_terminal` will allow subsequent plot commands to go to the screen again.

5.4.2 Always print to files

If you want regular plot commands to print to files instead of showing the plots on the screen, just set the terminal type and filename at the start of your session. For example, Gaston may beused in a webserver that reads SVG plots from a pipe; this may be set up as follows:

```
set_terminal("svg")
set_filename("|serverpipe")
plot(sin(-3:0.01:3),"title","SVG test")
```

The plot command in the last line will cause the SVG data representing the figure to be sent to the pipe `serverpipe`.

6 Reference

Note: In this section, at least superficial knowledge of Julia and gnuplot is assumed. Please refer to the respective documentation for more details.

6.1 Curve configuration

A given curve's configuration is stored in a structure of type `CurveConf`. This structure has the following fields:

Field	**Notes**	**Meaning in gnuplot**
`legend`	The text that appears in the legend box.	Argument of `title`.
`plotstyle`	How the curve will be plotted.	Argument of `with`.
`color`	The curve color.	Argument of `linecolor rgb`.
`marker`	The marker name.	Argument of `pointtype`.
`linewidth`	The curve line width; must be a positive number.	Argument of `linewidth`.
`pointsize`	The marker size; must be a positive number.	Argument of `pointsize`.

The types, default and valid values for each field are given in the following table.

Field	**Type**	**Default value**	**Valid values**
`legend`	`String`	`""`	Any string.
`plotstyle`	`String`	`lines`	`lines`, `linespoints`, `points`, `impulses`, `errorbars`, `errorlines`, `pm3d`, `boxes`, `image`, `rgbimage`.
`color`	`String`	`""`	`""`, any gnuplot color name, or a color specified in a string in the format `"#RRGGBB"`.
`marker`	`String`	`""`	`""`, `+`, `x`, `*`, `esquare`, `fsquare`, `ecircle`, `fcircle`, `etrianup`, `ftrianup`, `etriandn`, `ftriandn`, `edmd`, `fdmd` (run `test` in gnuplot terminal).
`linewidth`	`Real`	1	Any real number
`pointsize`	`Real`	0.5	Any real number

Notes: When `color` or `marker` are set to the empty string, gnuplot will use its own default values. Gaston does not verify that the color name provided is valid. You can see a list of valid colornames running `show colornames` in a gnuplot terminal.

The following table lists the marker names and corresponding symbols.

Marker name	Symbol
`+`	+
`x`	×
`*`	∗
`esquare`	□
`fsquare`	■
`ecircle`	○
`fcircle`	●
`etrianup`	△
`ftrianup`	▲
`etriandn`	▽
`ftriandn`	▼
`edmd`	◇
`fdmd`	◆

6.2 Axes configuration

There may be only one configuration per figure. This configuration is stored in a structure of type `AxesConf`, which has the following fields:

Field	Notes	Meaning in gnuplot
`title`	The figure's title	`title-spec`
`xlabel`	Abscissa's label	Argument to `set xlabel`
`ylabel`	Ordinate's label	Argument to `set ylabel`
`zlabel`	Z-axis label	Argument to `set zlabel`
`box`	Legend box configuration	Argument to `set key`
`axis`	Axis scale	Argument to `set logscale`

The types, default and valid values for each field are given in the following table.

Field	Type	Default value	Valid values
`title`	`String`	`"Untitled"`	Any string
`xlabel`	`String`	`"x"`	Any string
`ylabel`	`String`	`"y"`	Any string
`zlabel`	`String`	`"z"`	Any string
`box`	`String`	`"inside vertical right top"`	Any valid string
`axis`	`String`	`""`	`""`, `normal`, `semilogx`, `semilogy`, `loglog`

Notes: Gaston does not verify that box contains a valid `set key` argument. The `axis` values follow Matlab's conventions for logscale axes; Gaston translates them to gnuplot's equivalents.

6.3 Changing default configuration

Gaston provides functions to change the default values listed above. There is one function per each property, both for curves and for axes. For example, a Spanish speaker may wish to change the default value for figure titles from "Untitled" to "Sin título". This is achieved by

```
set_default_title("Sin título")
```

and, from that point on, any figure with an unspecified title will be titled thus.

Functions to change the defaults have the form `set_default_*(arg)`, where `*` is the property name and `arg` is the new default value (must be of the appropriate type).

6.4 Plot types

In this section, we describe how Gaston decides which kind of plot to produce (2-d, 3-d, or image), based on the available coordinates and specified plot style.

In the following table, a checkmark (✓) means that the coordinate has been specified by the user in either a mid-level or high-level command (`y` coordinates, if not specified, are calculated by Gaston, and are not included in the following table. The same applies to `yhigh`).

plotstyle	x	Z	ylow	Gnuplot command
`lines`	✓			`plot with lines`
	✓	✓		`splot with lines`
`linespoints`	✓			`plot with linespoints`
	✓	✓		`splot with linespoints`
`points`	✓			`plot with points`
	✓	✓		`splot with points`
`impulses`	✓			`plot with impulses`
	✓	✓		`splot with impulses`
`boxes`	✓			`plot with boxes`
`errorbars`	✓		✓	`plot with errorbars`
`errorlines`	✓		✓	`plot with errorlines`
`pm3d`		✓		`splot with pm3d`
	✓	✓		`splot with pm3d`
`image`		✓		`plot with image`
`rgbimage`		✓		`plot with rgbimage`

Any other combination of coordinates and plotstyle produces undefined behavior – we try to identify invalid combinations and produce an error, but this is not guaranteed. Please note that mixing 2-d and 3-d plots on the same figure also produces undefined behavior.

6.5 Global variables

TBW

6.6 Types

TBW

7 Testing

Gaston includes a unit test framework that verifies the program's functionality. To use it, issue

```
load("gaston_test.jl")
```

To run all tests, run:

```
run_tests()
```

This function will run all tests and print a summary to the screen.

7.1 Adding tests

New tests are easy to add. To test code that should complete successfully, use the macro `@test_success`, as in this example:

```
@test_success plot(-3:3,"title","test")
```

This code should be added to function `run_tests_success()`. To test code that should produce an error, use the macro `@test_error` inside function `run_tests_success()`.