Plotting Tools

GNUPlot, Desmos, & VegaLite

1 GNUPlot

General script pattern involves an arbitrary number of **set** *commands for setup of axes, titles, fonts, and other styles; followed by one or more* **plot** *commands, each of which renders output to the default or designated "terminal". See here for numerous demos, or in <src>/demo directory.*

Invocation

```
gnuplot [options]* [-c script-file]?
```

GNUPlot is invoked in either batch mode (script given to STDIN); or in interactive mode, which exposes a prompt. Default settings can be overridden at startup by initialization files in either <install>/.gnuplotrc or ~/.gnuplot. These files can invoke normal **set** *commands as well as setting environment variables such as:* `GNUTERM`, `GDFONTPATH`, `GNUPLOT_DEFAULT_GDFONT`.

Interactive (command-line) commands include:

break	cd	call	clear
contin.	do	evaluate	exit
fit	help	history	if
for	import	load	lower
pause	**[s]plot**	print	printerr
pwd	quit	raise	refresh
replot	reread	reset	save
set	show	shell	stats
system	test	toggle	undef.
unset	update	vclear	vfill
while			

Plotting

```
[s]plot [<\ul{p}lot\ul{e}lement>]+
```

replot *and* **refresh** *repeat the last* **plot** *command. PE has the following elaborate structure:*

```
|- axes ... to display ∈ []
|- <data> ... ∈ <file>|''|'-'|'+'|<"sampled">
      |- csv ... datafile is in csv format
      |- index ... [m, [n, [p]]] select from datasets
      |- every ... periodic row selection
      |- skip ... used to skip col headers
      |- using ... map data to plotting cols
            |- key ... row-based titles
            |- bins ... segregate x into buckets
            |- smooth ... interpolation ∈:
                 unique | frequency | cumulative |
                 bins | kdensity | [a|m|]csplines |
                 [s]bezier | unwrap | zsort
            |- [x|x2|y|y2|cb]ticlabels ...
|- title ... [columnheader[(N)]] <title> [at ... ]
|- with ... (see "Plot Types" section)
```

Plot examples:

```
plot sin(x)                  # "sampled" <data>
plot '-'<< EOD ...           # inline "here-doc"
plot for [j=1:3] sin(j*x)    # for clause
plot for [i in var] i.".dat" # for-in syntax
plot "file" with lines,      # with clause
    "file2" with points      # second PE
```

```
plot [-pi:pi*2] [-5:5] tan(x)   # x, y ranges
plot [t=1:10] tan(t), x**t      # parametric
plot "file"using (tan($2)):($3) smooth csplines \
    axes x1y2 title "parametric"with lines
```

using *phrase maps <data> to the abscissa & ordinate (or angle and radius). For example:*

```
... using 1:($2+$3)             # derived fields
... using 1:($3>7 ? $2 : 1/7)   # ternary filtered
... using 1:5:6 '%lf,%lf,%lf'   # comma sep
... using 2:4:xtic(1):ytic(3)   # labels
```

Set Command

Use **help** *to learn about specific set subcommand parameters. Some, like datafile, palette, pm3d, and style, have a number of subsub-commands. Use* **show <cmd>|all** *to see current settings, eg:*

```
set angle radians      # a typical imperative
show angles            # now in radians
```

angles	arrow	autoscale	bind
bmargin	border	boxwidth	boxdepth
color	colorseq.	clabel	clip
cntrlbl	cntrparam	colorbox	col.names
contour	dashtype	datafile	decimalsgn
dgrid3d	dummy	encoding	errorbars
fit	fontpath	format	functions
grid	hidden3d	hist.size	history
isosamp.	isosurf.	jitter	key
label	linetype	link	lmargin
loadpath	locale	logscale	macros
mapping	margin	micro	minussign
monoch.m	mouse	mttics	multiplot
mx2tics	mxtics	my2tics	mztics
nonlinear	object	offsets	origin
output	overflow	palette	parametric
paxis	pixmap	plot	pm3d
p.ntint.	pointsize	polar	print
psdir	raxis	rgbmax	rlabel
rmargin	rrange	rtics	samples
size	spiderp.t	style	surface
table	terminal	termopt.	theta
tics	ticslevel	ticscale	timestamp
timefmt	title	tmargin	trange
ttics	urange	variables	version
vgrid	view	v?range	walls
?data	?dtics	?label	?mtics
?range	?tics	?plane	?zeroax.

(? denotes an axis name, eg xtics.)

Script Syntax

Expressions:

```
var1 + var2            # variables
sin(x) + 2             # function call
{1.0, -3.4}            # constant ∈ ℂ
~var                   # one's comp.
var1 && var2           # logical AND
```

Operators:

**	*	/	%	+	-	==	!=
<	<=	>	>=	<<	>>	&	^
|	&&	||	=	,	.	eq	ne

Conditionals:

```
if(...){...} else {...}              # conditional
plot for [file in "A B C"] file ...  # plot for
set for [i = 1:10] style line i lc ...# set for
while(<expr>){<cmd>*}                # while syntax
do for [i in "A B C"] { <cmd>*}      # repet. cmds
```

do *and* **while** *admit* **break** *and* **continue** *statements, as you might imagine.*

Strings:

```
sprintf("Title, plot #%d",n)   # functions
"Title for plot #"."4"         # concatenation
if ("A"."B" eq "AB")           # operators
strlen("αβγ")                  # =3 ... utf-8
"generated on `date`"          # cmd-line subst.
# I'm a comment                # comments
range = "1:3"; plot ... using @range # macros
```

Built-in Functions

Math Functions (data-type default is $\mathbb{C}$ *then* $\mathbb{R}$*):*

abs	acos[h]	airy	arg
asin[h]	atan[2|h|]	Elliptic	besj[0|1|n]
besy[0|1|n]	besi[0|1|n]	ceil	cos[h]
erf[c]	exp	expint	floor
gamma	ibeta	inverf	igamma
imag	invnorm	int	lambertw
lgamma	log[10]	norm	rand
real	sgn	sin[h]	sqrt
tan[h]	vogt		

String, Data, & Date Functions:

column	col.head	exists	hsv2rgb
palette	sprintf	stringcol.	strlen
strstrt	substr	str[f|p]time	system
timecol.	tm_hour	tm_mday	tm_min
tm_mon	tm_sec	tm_wday	tm_yday
tm_year	time	trim	valid
value	voxel	word[s]	

Stylizing: lines, text & color

Text:

```
x^2, y_2 {/Times abc}          # enhanced text
\U+221E                        # unicode (∞)
set term pdfcairo font "Times,12# font
```

Lines (linetypes, linecolors, textcolors, fillcolors, linestyles, dashtypes):

```
plot "foo", "bar"              # use default lt'1,2
plot sin(x) linetype 4         # explicit LT
plot sin(x) lt rgb "violet"    # inline LT def
plot ... using 1:2:3 lc variable   # data-def'd col
plot ... dt 4                  # numbered dt
plot ... dt (s1,e1,...)        # pattern-spec'd DT
set style line 5 lt ... lw ...# build LS
plot sin(x) ... ls 5           # use the above
```

Color specification (for lines, text):

```
plot ... [lc|lt|fc] <col_spec>  # coloring syntax
plot ... lc 0xFF00FF            # RGB-based CS
plot ... lc "#FF00FF"           # 'x11'-based CS
plot ... lc palette frac .2     # fraction ∈ [0,1]
```

Plot Types ("styles")

```
plot <data> using \dots with <style>
```

Each PE in a **plot** *command admits an optional override to the default <style>, which then renders an eponymous plot type. The PE's associated* **using** *must yield an acceptable number of data fields, as indicated next each <style> below.*

- **arrows** ... x, y, length, angle
- **boxerrorbars** ...
- **boxes** ... x,y,[xwidth]
- **boxes** (3d) ... x, y, z, [xwidth], [color]
- **boxplot** ... x,y,[?,?]
- **boxxyerror** ... x, y, [xδ,yδ]|[xmin,xmax,ymin,ymax]
- **candlesticks** ... x, min, wsk_min, wsk_max, max
- **circles** ... x, y, [rad,[arc_beg, [arc_end]]], [col]
- **ellipses** ... x, y, [major, [minor, [angle]]]
- **dots** ... [x], y, [z]
- **filledcurves** ... x, y, yerror
- **financebars** ... date, open, low, high, close
- **histeps** ... [x], y, [z]
- **histogram** ... y, [yerr|[ymin, ymax]]
- **image** ... bitmap-image
- **impulses** ... [x], y, [z]
- **labels** ... x, y, [z], string
- **lines** ... [x], y, [z]
- **linespoints** ... [x], y, [z]
- **parallelaxes** ... *(one per axis)*
- **polar** ... angle, radius
- **points** ... [x], y, [z]
- **polygons** ... <polygon>
- **spiderplot** ... *(one per axis)*
- **[f|fill|]steps** ... x, y
- **rgb[alpha|image]** ... (see image)
- **vectors** ... x, y, [z], xΔ, yΔ, [zΔ]
- **[x|xy|y]errorbars** ... x, y, [xΔ | [xlow, xhigh]]
- **[x|xy|y]errorlines** ... x, y, [xΔ | [xlow, xhigh]]
- **pm3d** ... *(see documentation)*
- **isosurface** ... <voxel-grid-file>
- **zerrorfill** ... x, y, z, [zΔ | [zlow, zhigh]]

∃ variations on a theme, eg to create a "bee swarm" plot, use **set jitter**, *then* **with points**. *Similarly, for "fence plots" use the* **zerrorfill** *style. Some styles admin an additional qualifier, eg* **histogram**
.

Terminals (Output)

```
set term [terminal-name] [term-option]*
```

Output can be rendered as code for external compilation (eg, tikz, svg, HTML canvas, etc); as image binaries (jpeg, gif, pdf, etc); within a specified terminal (qt, x11, etc); or directly to a supported printer (eg, epson, hp, etc). My preferred terminals are: cairolatex, canvas, epslatex, gif, pdfcairo, pngcairo, pstricks, svg, tikz, wxt, x11. For example:

```
set output 'file.png'           # save to file
set term pngcairo size 400,600  # png
```

See more examples here. Each terminal has its own [unfortunately] distinct interface to control options such as the following (more common ones):

background <color>	color | monochrome
[no]enhanced	font <fontname[,size]>
fontscale <scale>	[no]header <header>
[input|standalone]	[no]inverted
linewidth <lw>	[rounded|butt|square]
title <title>	[no]transparent
size <XX>,<YY>	resolution <dpi>

Command-line Shortcuts

^B	back char	^F	forwd char
^A	begin line	^E	end line
^H	del prev char	DEL	del curr char
^D	del curr char	^K	del to EOL
^L	redraw line	^U	delete line
^W	del prev word	^V	inhibit
TAB	file-complet'n	^P	back history
^N	forward hist	^R	back-search

2 Desmos

Click icon on left ( ) to toggle a single graphical element's display on/off; click and hold to set its presentation attributes. Set display using wrench . Animate with sliders ( ).

Functions

```
f(x) = ...                      # function
```
$\{-6 \le x < -2 : \frac{1}{2}x, -2 \le x : 4 - x^2\}$ # piecewise
```
(4 sin(4 t),3 cos(3t))          # parametrics
```
$r = \theta/2$ # Polar

Restrictions

$f(x)\{0<x<5\}$ # simple

$c_1\{0<y<10\}$ # y range

$x^2\{y<3\}\{x>0\}$ # compound

Statistics

Use settings icon to turn a listing of discrete points into a table.

```
(1,2),(2,1)                     # points
```
$a = [-100, -98, \ldots 100]$ # list
```
mean(a)                         # stats
```
∃ many built-in statistical functions:

total	length	mean
median	min	max
quantile	mad	stdevp
var		

Calculus

```
d/dx f(x)                       # derivative
```
$\int_0^x (1-t^2)dt$ # integrals

Keyboard Shortcuts

Type word like ***sum*** *and* $\sum$ *"template" will appear.*

sqrt ($\sqrt{}$)	sum ($\sum$)	pi (π)
theta (θ)	prod ($\prod$)	int ($\int$)

3 Plots.jl

Frontends: Plots, Gadfly, and Winston (obsolete). "Backend" packages for Plots: PlotlyJS, PyPlot, GR, PGFPlotsX. (Eg, `gr()`*). See here for attributes supported per backend.*

Plotting

```
p = plot(x,y)              # visual side-effect
plot!(x,y)                 # add to current plot
plot(p,x,y)                # ≡ to above (adds to p)
z = rand(10,2); plot(x,z) #  mult series
plotly()                   # set plotly backend
gr()                       # set gr backend
display(plot(x, y))        #  required in scripts
plot(x,x->sin(x))          # plot anon fn
tvec = range(0, 6.28, length = 100)# setup...
plot(sin,cos,tvec)         #  parametric plot
```

Stylizing

Choose a color scheme from here, a theme from here, and then fine-tune by hand like:

```
plot!(p,title="...")                    # title
plot!(p,label = ["Line 1"  "Line 2"])# attrs
xlabel!("My x")                         # alternat.
clibrary(<clib>)  # import color library ∈:
```
:Plots, :cmocean, :misc, :colorcet, :colorbrewer
```
plotattr()                 # query params
```

Plot-level attributes (more here):

bg (color)	size	dpi
fontfamily	title	legend
framestyle	aspect_ratio	camera
palette		

Grid attributes (more here):

grid	gridlinewidth	link
[x|y|z]lims	[x|y|z]ticks	[x|y|z]scale
[x|y]guide	[x|y]label	

Series-level attributes (more here):

Points	Lines	Surfaces
markercolor	linecolor	fillrange
markeralpha	linealpha	fillcolor
markersize	linestyle	fillalpha
markershape	linewidth	
markerstroke-		
-color		
-alpha		
-width		

Plot Types

```
plot!(p,seriestype = :scatter)
```

Where seriestype ∈:

line	path	steppre
steppost	sticks	scatter
heatmap	hexbin	barbins
barhist	stephist	bins2d
histogram2d	histogram3d	density
bar	hline	vline
contour	pie	shape
image	path3d	scatter3d
surface	wireframe	contour3d
volume		

Screen Layout

```
plot(x, y, layout = (4, 1))          # 4x1 vertically
plot(p1,p2,p3,p4,layout=(2,2))# saved p's
l = @layout [a{0.6h} b{0.6w} c]# advanced
plot(x, y, layout = l)               # use above
BB = (x1,x2,y1,y2)           # set boundingbox
plot(x,y,inset=(1,BB))       # insetting
```

Exporting & Importing

Save to .eps, .html, .pdf, .png, .ps, .svg, .tex, .text:

```
savefig("myplot.png")      # from screen
savefig(p, "myplot.pdf") # from var p
png(fn)                    # shorthand save as
img = load("a.png")        # load image
plot(x,y,img)              # plot an image
```

Animations

See here for more examples.

```
p = plot([sin, cos], zeros(0), leg = false)
anim = Animation()
for x = range(0, stop = 10π, length = 100)
    push!(p, x, Float64[sin(x), cos(x)])
    frame(anim)
end
```

Extensions

Use or create recipes for often-generated plot-types. Eg, StatsPlot allows visualization of data frames, distributions, boxplots, etc. Also see GraphRecipes for help plotting graphs. Alternatively, browse the "ecosystem".

```
@df iris scatter(          # using Dataframes
    :SepalLength, :SepalWidth)
plot(Normal(3, 5))         # using Distributions
```

4 Observable Plot

Inspired by Grammar of Graphics, built atop D3, and tightly integrated with Observable. Plot is opinionated, full of default options and [overridably] programmatic means of inferring the plot-author's intent. Eg, scales (their type, domain, and range) are often inferred from data and screen context.

Plotting

To plot, issue one of the following calls

```
Plot.plot(<options>)
Plot.<mark>(<data>, <options>).plot()
```

Plot-level style options include:

marks	marginTop	marginRight
marginBottom	marginLeft	width
height		

Options & Channels

Ubiquitous as arguments in the signature of Plot's modules, each set of options is associated with one of the major domain entities: a mark, a scale, a facet, a transform, or the plot itself. "Channels" are a special kind of option, exclusively for marks, that vary according to the associated mark's data.

Marks

Importantly, **Plot** *eschews plot-types in favor of "marks," which represent graphical elements to display. Each mark type has options, some of which are common to all mark-types. Mark types ∈:*

area[X|Y]	bar(X|Y)	cell[X|Y]	dot[X|Y]
line[X|Y]	link	rect[X|Y]	rule(X|Y)
text[X|Y]	tick(X|Y)		

Style options (some only for rect. marks):

fill	fillOpacity	stroke
strokeWidth	strokeOpacity	strokeLinejoin
strokeMiterlimit	strokeDasharray	mixBlendmode
insetTop^r	insetRight^r	insetBottom^r
rx^r	ry^r	

Mark options that vary with data are "channels."
Channels that are available for all mark types ∈:

fill	fillOpacity	stroke
strokeOpacity	title	

Channels by plot type (optional / required indicated for <mark>X, <mark>Y respectively):

	x	x1	x2	y	y1	y2	z	r	text	fontSize	rotate
area		rr	oo		rr	oo	oo				
bar	-o	r-	r-	o-	-r	-r					
cell	oo			oo							
dot	oo			oo				oo			
line	rr			rr				oo			
link		r	r	r	r						
rect		rr	rr	rr	rr						
rule	o-	-o	-o	-o	o-	o-					
text	oo			oo					r	oo	oo
tick	ro			or							

Scales `[<scale>]`

A means of encoding data into graphical content (screen position, color, etc). Scales map from an input domain to an output range. Options below are uniformly accessible or specific to scale type, as indicated in superscript: quantitative, positional, ordinal, band, axis, color.

type	domain	range
reverse	transform	clamp^q
nice^q	zero^q	percent^q
inset^p	round^p	padding^o
paddingInner^b	paddingOuter^b	axis^a
ticks^a	tickSize^a	tickFormat^a
tickRotate^a	grid^a	label^a
labelAnchor^a	labelOffset^a	scheme^c
interpolate^c		

Facets

Small multiples, laid out in grid fashion. Choose 2 of 3 channels from `data`, `x`, `y`, *and any of the following optional styles:*

marginTop	marginRight	marginBottom
marginLeft	grid	

Transforms

Take stipulated "outputs" and "options" as inputs and produce new, transformed options that can be used as arguments to `.plot()` *or* `<mark>()`

```
Plot.<trnsfrm>(<outputs>, <options>)
```

bin[X|Y]	group[X|Y]	select(First|Last)
normalize(X|Y)	window(X|Y)	select(Min|Max)(X|Y)
map[X|Y]		stack(X|Y)[1|2]

Transform-specific options:

bin	map, et al	stack
thresholds	k	offset
domain	shift	order
cumulative	reduce	reduce