A VISUAL REFERENCE · gnuplot

gnuplot, one change at a time

From a first curve to layout, plot families and reusable commands. Complete source. A small change. Its actual result.

Source on Codeberg - Contact: info@emaix.org.

The Org file is the canonical source. Read the HTML guide beside your work, or download its PDF and source for offline reading.

> **Last changes: 2026-10-31.** Added violin plots, ECDFs, hexbin density, Sankey diagrams, waterfall variants, arc diagrams and dumbbell charts. Unified figure galleries and source styling.

400
worked examples

37
topic families

1
canonical Org source

READING THE TOUR

Three ways to see a change

Plotting code

Read the complete commands and the explanation of new parameters.

Actual figure

Inspect the generated plot alongside its code. Open a thumbnail to jump to its example.

One change

Compare the named baseline with Before / After controls and highlighted source lines.

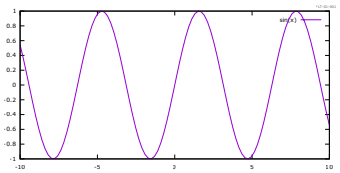
Follow the chapters from first plots, variables and data to axes, styling, reusable commands and statistics. Then choose a plot family: bars and grouping, distributions and scatter plots, colour maps and heatmaps, surfaces and radar charts, sectors and waterfalls. Chapters 31–37 develop violin plots, ECDFs, hexbin density, Sankey diagrams, waterfall variants, arc diagrams and dumbbell comparisons. Every comparison points to an earlier example.

Each example uses the plotting commands shown beside its figure. Chapter 30 explains custom styles and shows the complete files to save before loading them. Its load commands are visible; earlier examples write any chosen RWTH colours as hexadecimal values directly in their commands and use no additional style files.

Start with `plot sin(x)`. There are no exercises and no hidden accumulating session: every plot block starts its own gnuplot process. Blue lines differ from the named comparison. The HTML has Before / After switches; the PDF shows the resulting plot. Some refactoring examples deliberately produce the same picture. Read each plotting command beside its resulting figure.

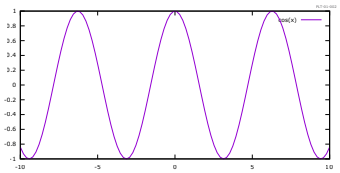
Read from the beginning or jump to a stable example ID. Each example includes its complete input and captured result. Copy link preserves the address of the guide you are reading, including its current domain and path.

01 · First picture



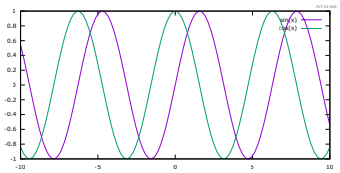
PLT-01-001

Start with one command



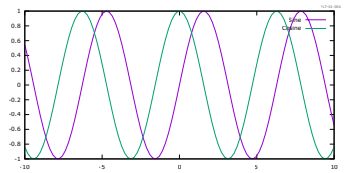
PLT-01-002

Change the function



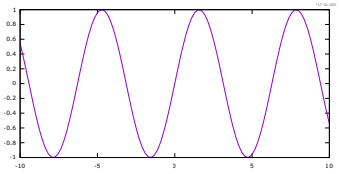
PLT-01-003

Add a second curve



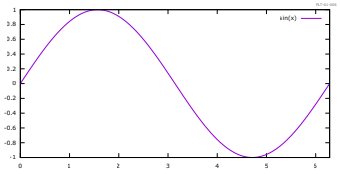
PLT-01-004

Name the curves



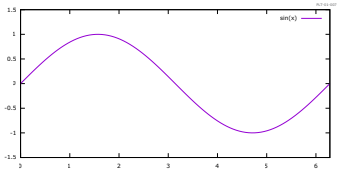
PLT-01-005

Hide the legend



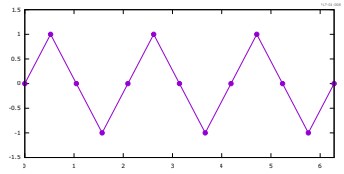
PLT-01-006

Choose the horizontal interval



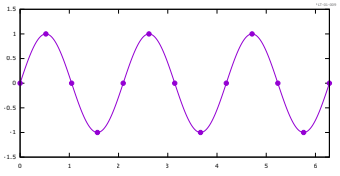
PLT-01-007

Leave room above and below



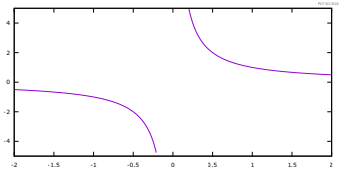
PLT-01-008

See coarse sampling of a function



PLT-01-009

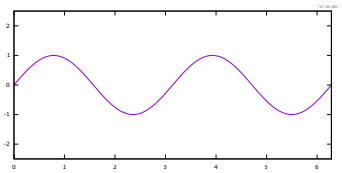
Increase samples without changing the function



PLT-01-010

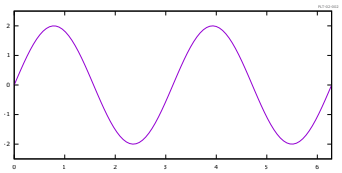
Break a line at an excluded interval

02 · Variables, functions and arithmetic



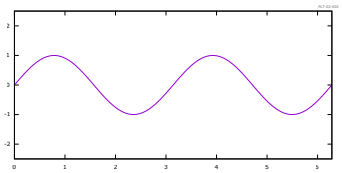
PLT-02-001

Use variables for repeated numbers



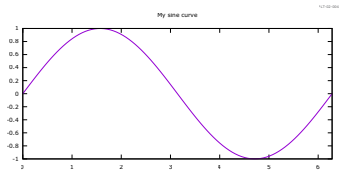
PLT-02-002

Change one variable



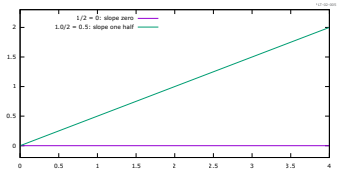
PLT-02-003

Name the expression as a function



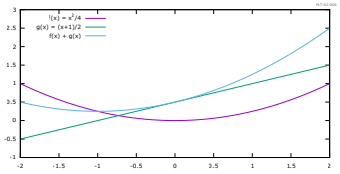
PLT-02-004

Use a string variable directly



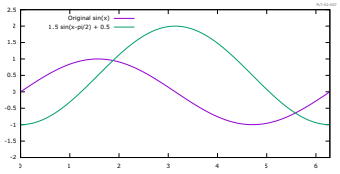
PLT-02-005

Avoid accidental integer division



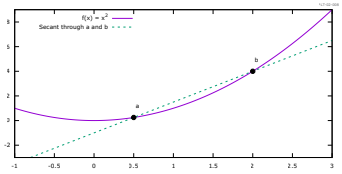
PLT-02-006

Calculate and combine functions



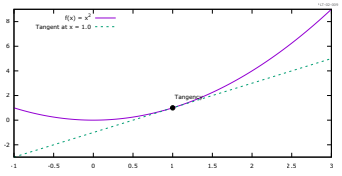
PLT-02-007

Scale and shift a function with arithmetic



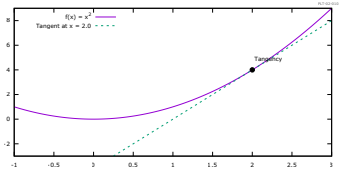
PLT-02-008

Calculate the secant through two points



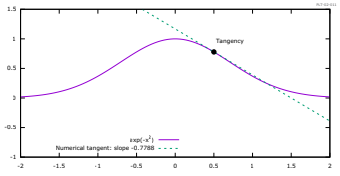
PLT-02-009

Calculate a tangent from the known derivative



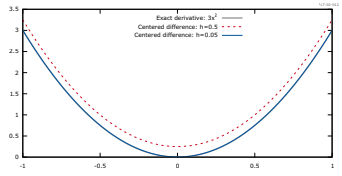
PLT-02-010

Move the tangent by changing one variable



PLT-02-011

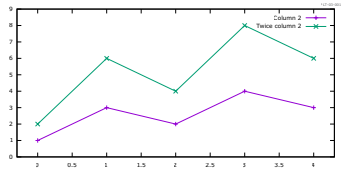
Estimate a tangent with a numerical derivative



PLT-02-012

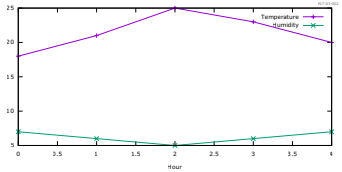
Compare numerical derivative step sizes

03 · Reading and selecting data



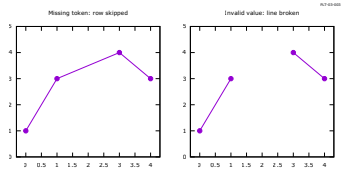
PLT-03-001

Read columns and calculate plotted values



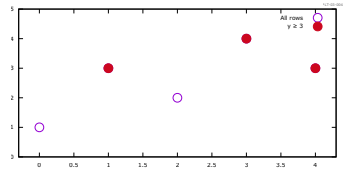
PLT-03-002

Read CSV headers and use them in the legend



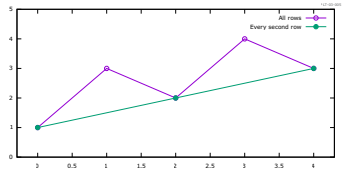
PLT-03-003

Compare missing values with invalid values



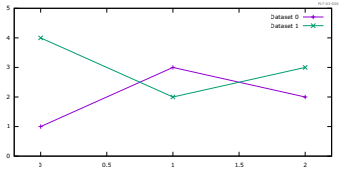
PLT-03-004

Filter observations with a column expression



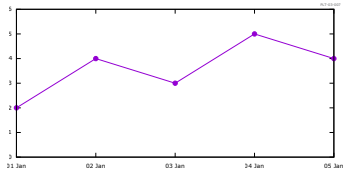
PLT-03-005

Select every second data row



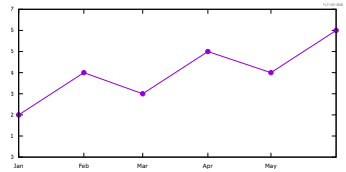
PLT-03-006

Select datasets separated by blank records



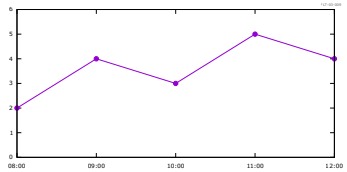
PLT-03-007

Read dates and format their tick labels



PLT-03-008

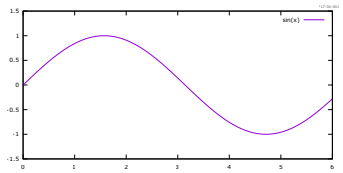
Place ticks at calendar-month intervals



PLT-03-009

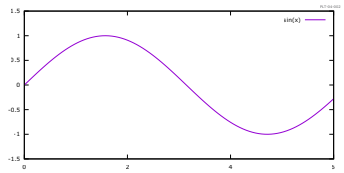
Read timestamps and show hourly ticks

04 · Major tick marks



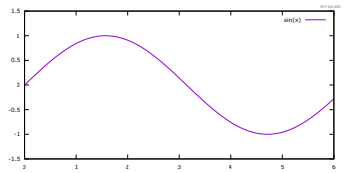
PLT-04-001

Prepare a simple tick example



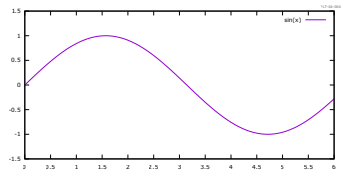
PLT-04-002

Place a tick every two units



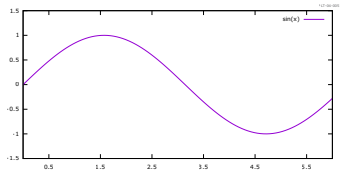
PLT-04-003

Halve the tick interval



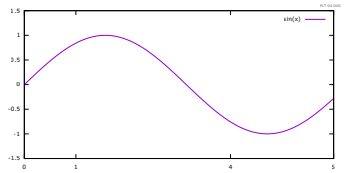
PLT-04-004

Use half-unit ticks



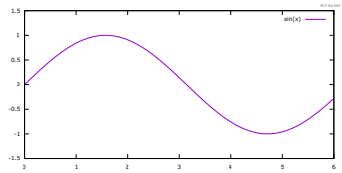
PLT-04-005

Set start, increment and end



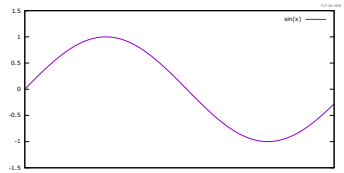
PLT-04-006

Choose irregular tick positions



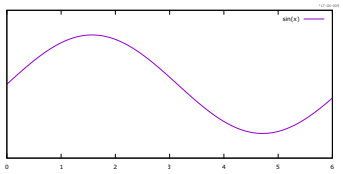
PLT-04-007

Change y ticks independently



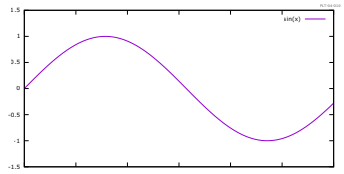
PLT-04-008

Remove x ticks



PLT-04-009

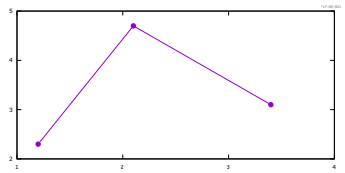
Remove y ticks



PLT-04-010

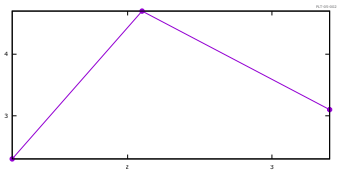
Keep marks but hide their labels

05 · Axis ranges and scales



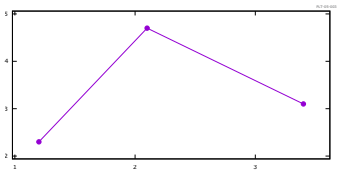
PLT-05-001

Observe automatic range rounding



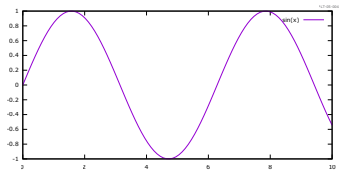
PLT-05-002

Keep automatic ranges at the exact extrema



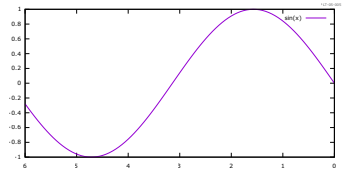
PLT-05-003

Pad the data range with offsets



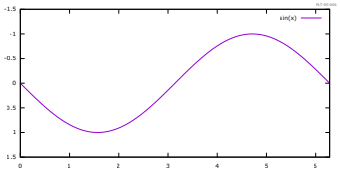
PLT-05-004

Let the upper x limit be automatic



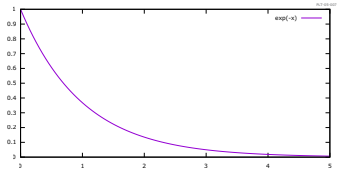
PLT-05-005

Reverse the horizontal direction



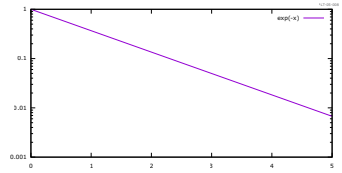
PLT-05-006

Reverse the vertical direction



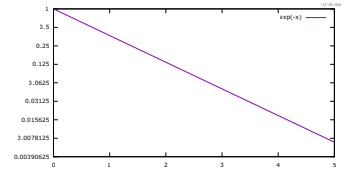
PLT-05-007

Begin with a positive function



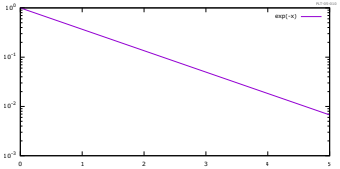
PLT-05-008

Use a base-10 logarithmic y axis



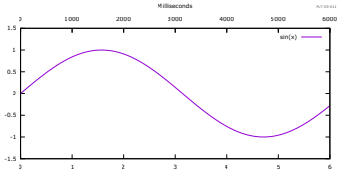
PLT-05-009

Use a base-2 logarithmic y axis



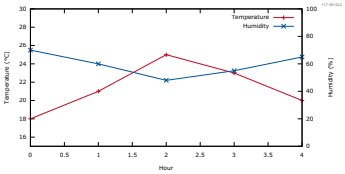
PLT-05-010

Format logarithmic ticks as powers



PLT-05-011

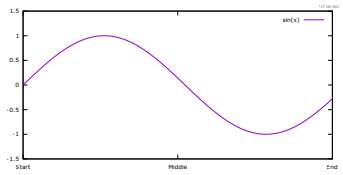
Link a second axis in different units



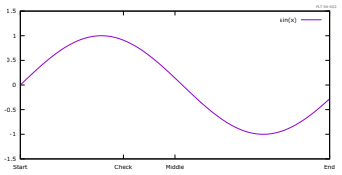
PLT-05-012

Plot a separate quantity on the right axis

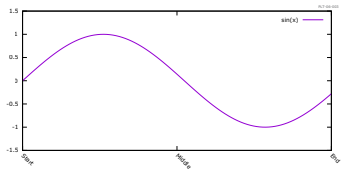
06 · Tick labels and formatting



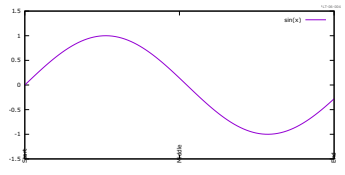
PLT-06-001
Replace numbers with words



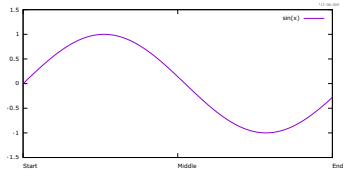
PLT-06-002
Add one labelled tick



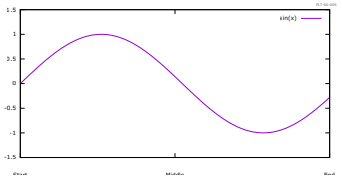
PLT-06-003
Rotate tick labels



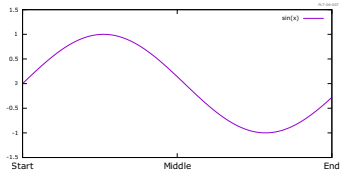
PLT-06-004
Stand tick labels vertically



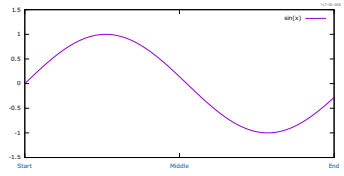
PLT-06-005
Left-align horizontal tick labels



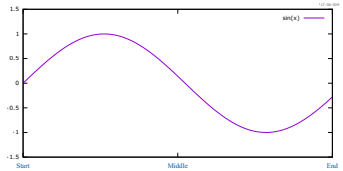
PLT-06-006
Offset tick labels



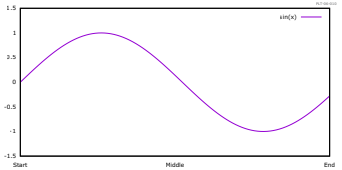
PLT-06-007
Increase tick-label size



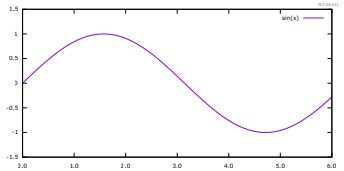
PLT-06-008
Colour tick-label text



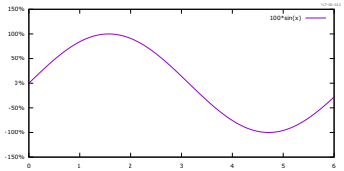
PLT-06-009
Use a different tick-label font



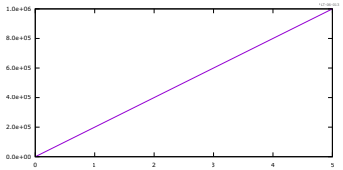
PLT-06-010
Keep labels and hide tick strokes



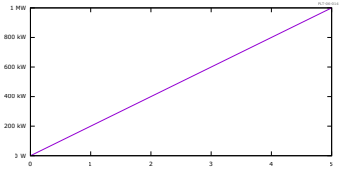
PLT-06-011
Print one decimal place



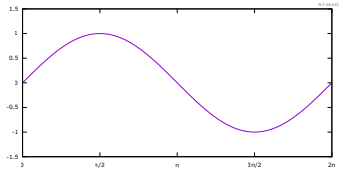
PLT-06-012
Format values as percentages



PLT-06-013
Use scientific tick notation

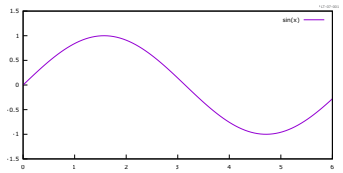


PLT-06-014
Use engineering prefixes on ticks

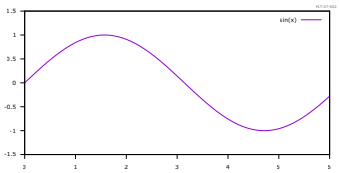


PLT-06-015
Label an angular axis in multiples of pi

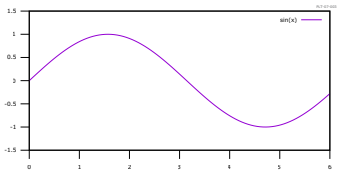
07 · Minor ticks and grids



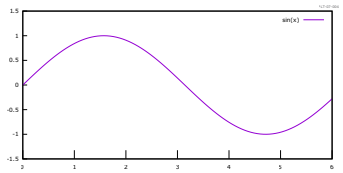
PLT-07-001
Remove mirrored ticks



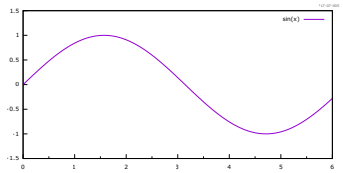
PLT-07-002
Point ticks outward



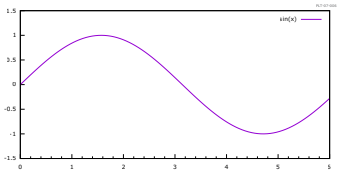
PLT-07-003
Lengthen major ticks



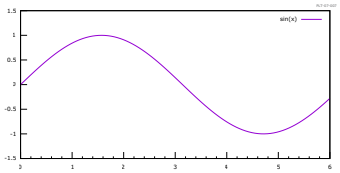
PLT-07-004
Give the two axes different tick lengths



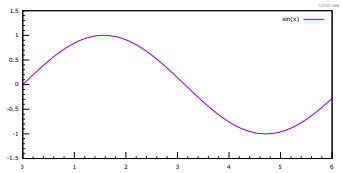
PLT-07-005
Add a minor tick halfway



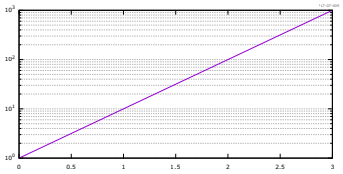
PLT-07-006
Add four minor ticks



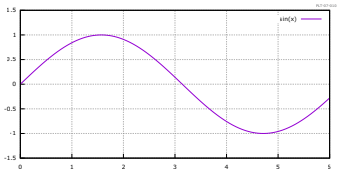
PLT-07-007
Set major and minor lengths



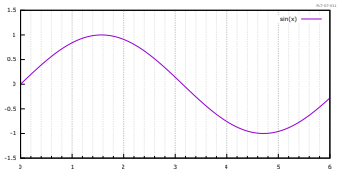
PLT-07-008
Subdivide the vertical axis



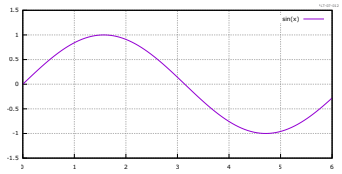
PLT-07-009
Show minor ticks on a logarithmic axis



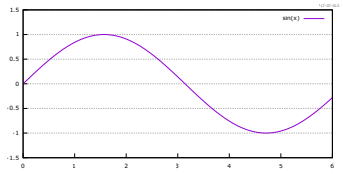
PLT-07-010
Add a major grid



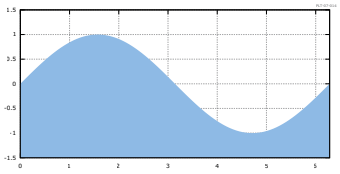
PLT-07-011
Show a lighter minor grid



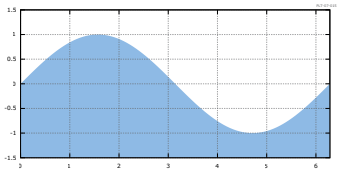
PLT-07-012
Use dashed grid lines



PLT-07-013
Keep only horizontal grid lines

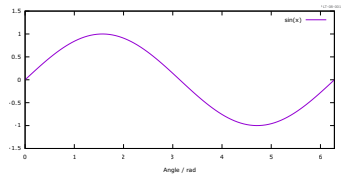


PLT-07-014
Draw the grid at the back



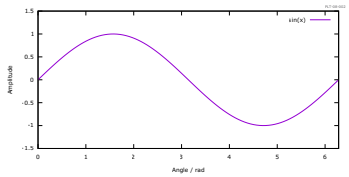
PLT-07-015
Draw the grid at the front

08 · Axis labels and titles



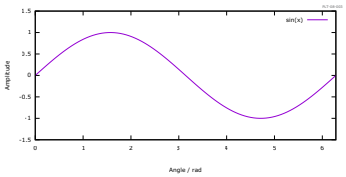
PLT-08-001

Name the x axis



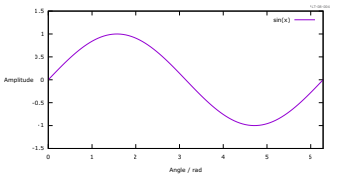
PLT-08-002

Name the y axis



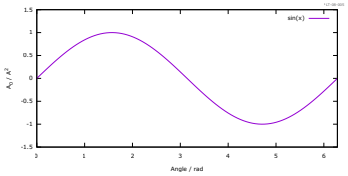
PLT-08-003

Move the x label downward



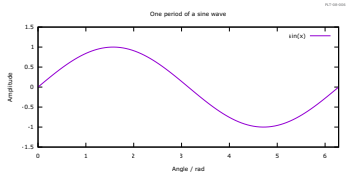
PLT-08-004

Make the y label horizontal



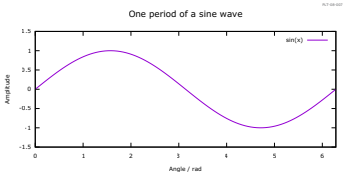
PLT-08-005

Use subscripts and superscripts



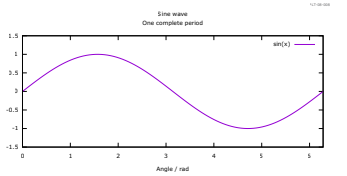
PLT-08-006

Add a figure title



PLT-08-007

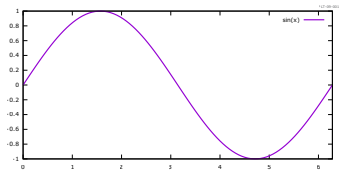
Increase the title font



PLT-08-008

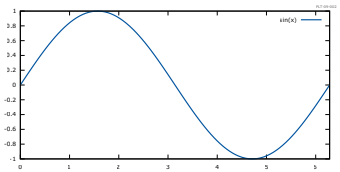
Write a two-line title

09 · Line, point and fill styles



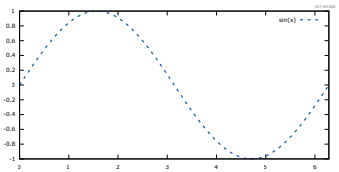
PLT-09-001

Increase line width



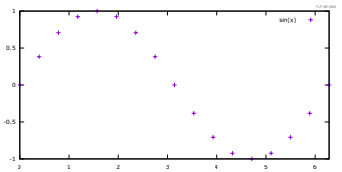
PLT-09-002

Choose an explicit line colour



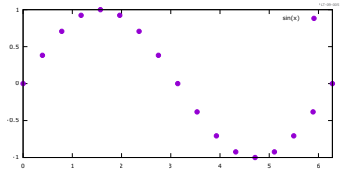
PLT-09-003

Use a dashed line



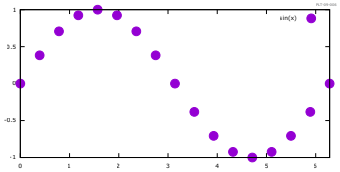
PLT-09-004

Plot sampled points



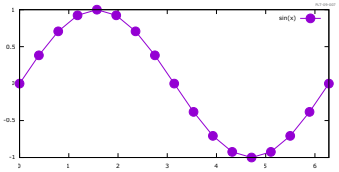
PLT-09-005

Choose the point symbol



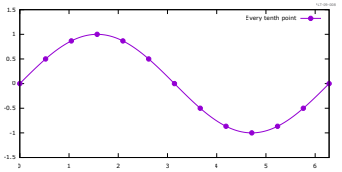
PLT-09-006

Increase point size



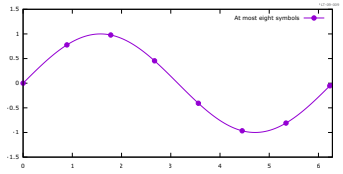
PLT-09-007

Connect the points



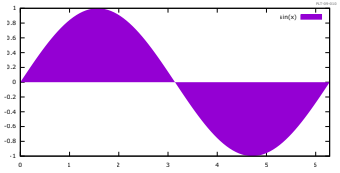
PLT-09-008

Show a symbol at every tenth sample



PLT-09-009

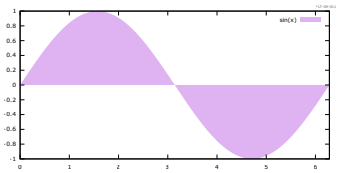
Limit the total number of point symbols



PLT-09-010

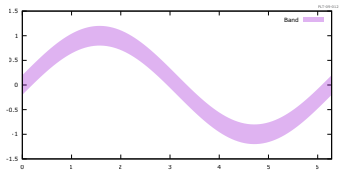
Fill down to zero

09 · Line, point and fill styles (continued)



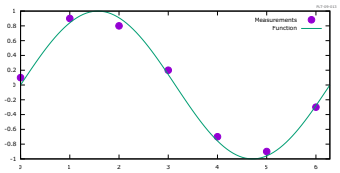
PLT-09-011

Make the fill translucent



PLT-09-012

Fill between two curves



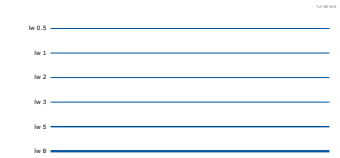
PLT-09-013

Set different defaults for data and functions



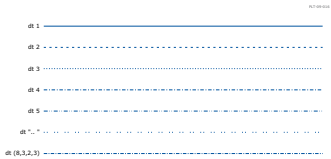
PLT-09-014

Quick reference: line types



PLT-09-015

Quick reference: line widths



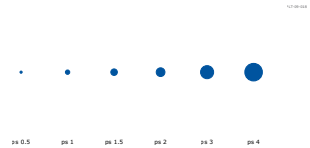
PLT-09-016

Quick reference: dash patterns



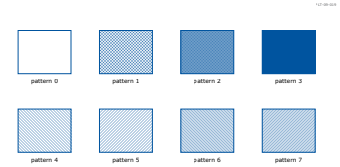
PLT-09-017

Quick reference: point symbols



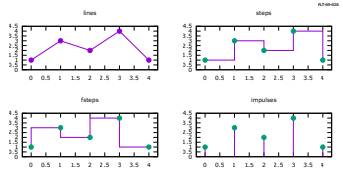
PLT-09-018

Quick reference: point sizes



PLT-09-019

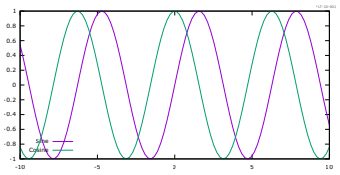
Quick reference: fill patterns



PLT-09-020

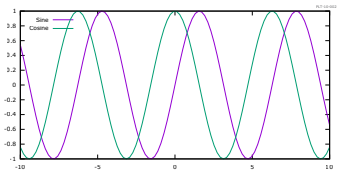
Quick reference: lines, steps and impulses

10 · Legend position and layout



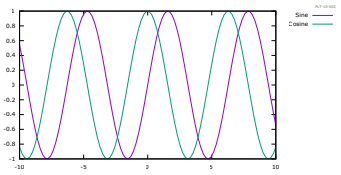
PLT-10-001

Move the legend to the lower left



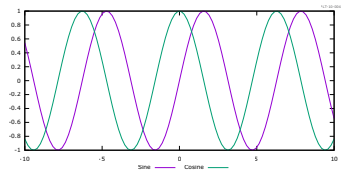
PLT-10-002

Move the legend to the upper left



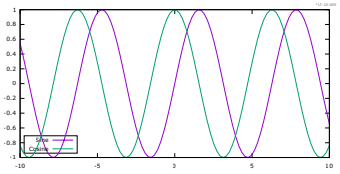
PLT-10-003

Move the legend outside



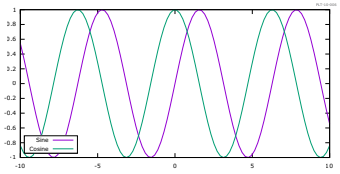
PLT-10-004

Place a horizontal legend below



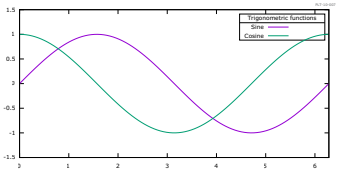
PLT-10-005

Draw a legend box



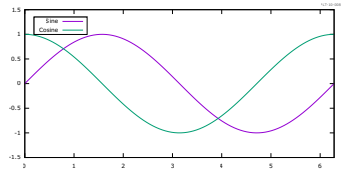
PLT-10-006

Give the legend an opaque background



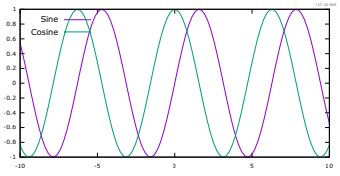
PLT-10-007

Give the legend a title



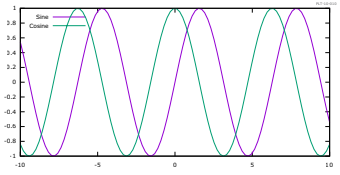
PLT-10-008

Anchor the legend with graph coordinates



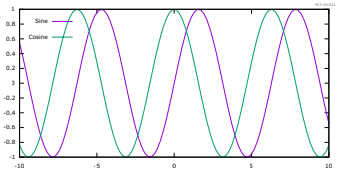
PLT-10-009

Enlarge legend text



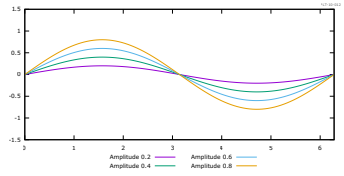
PLT-10-010

Lengthen legend samples



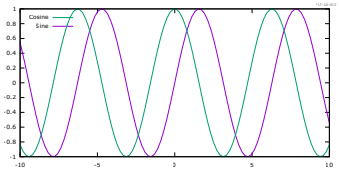
PLT-10-011

Increase spacing between entries



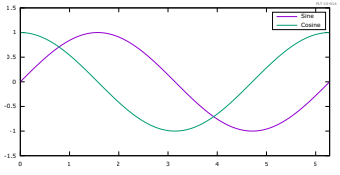
PLT-10-012

Arrange the legend in two columns



PLT-10-013

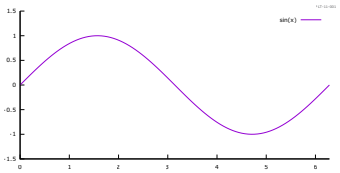
Reverse legend entry order



PLT-10-014

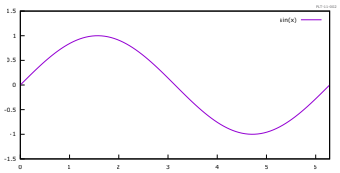
Reverse sample placement without reordering entries

11 · Borders and reference lines



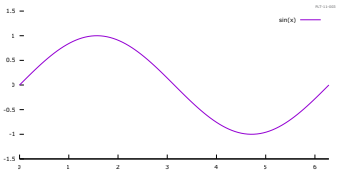
PLT-11-001

Keep only left and bottom borders



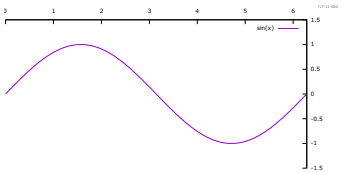
PLT-11-002

Restore all four borders



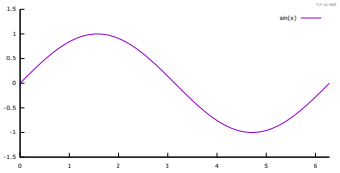
PLT-11-003

Keep just the bottom border



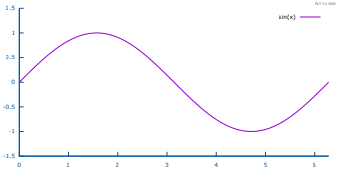
PLT-11-004

Put the frame and ticks at the top and right



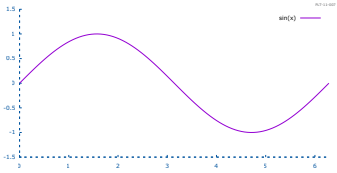
PLT-11-005

Thicken the remaining borders



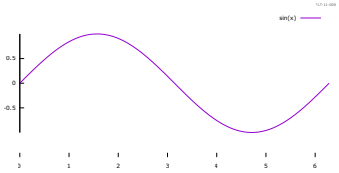
PLT-11-006

Colour the border and tick strokes



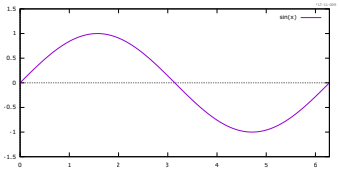
PLT-11-007

Use a dashed border



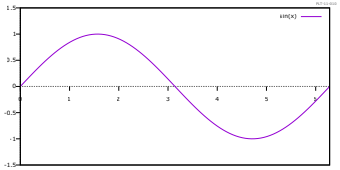
PLT-11-008

Shorten borders to the data extent



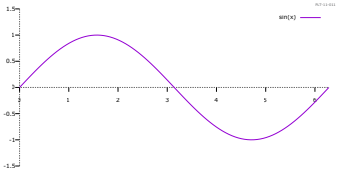
PLT-11-009

Draw the zero axes



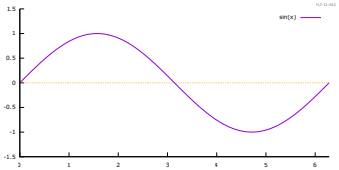
PLT-11-010

Put ticks on the zero axes



PLT-11-011

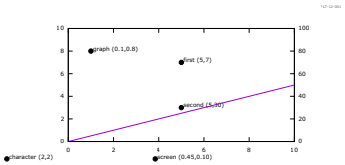
Remove the outer frame



PLT-11-012

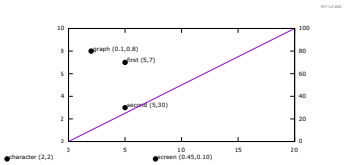
Style the horizontal zero reference

12 · Labels, arrows and shaded regions



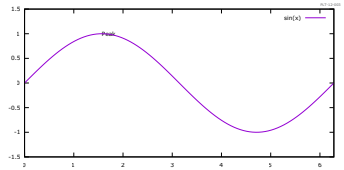
PLT-12-001

Compare five coordinate systems



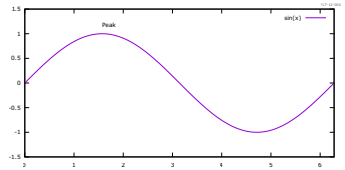
PLT-12-002

Change ranges while retaining annotation anchors



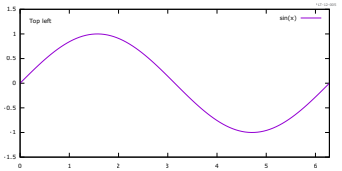
PLT-12-003

Place text at a data point



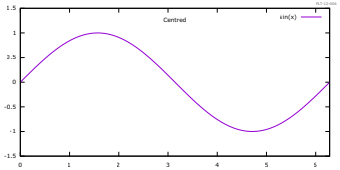
PLT-12-004

Offset a label from its point



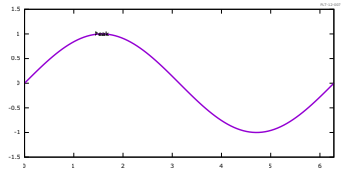
PLT-12-005

Anchor text to the plot region



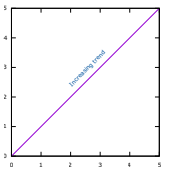
PLT-12-006

Centre text on its anchor



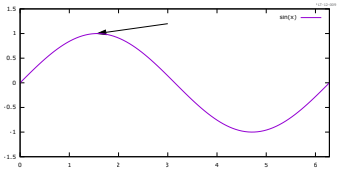
PLT-12-007

Give a label an opaque text box



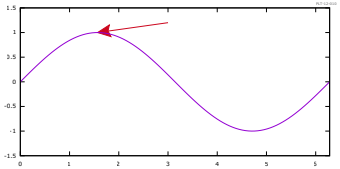
PLT-12-008

Rotate an annotation along a trend



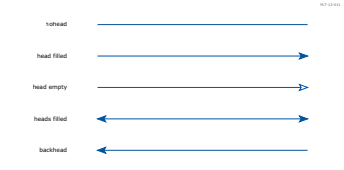
PLT-12-009

Draw an arrow between points



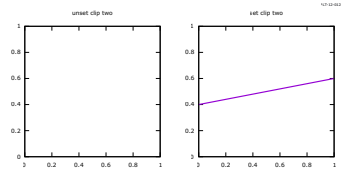
PLT-12-010

Control arrowhead size and angles



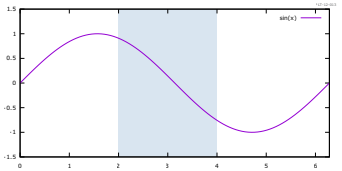
PLT-12-011

Quick reference: arrowheads



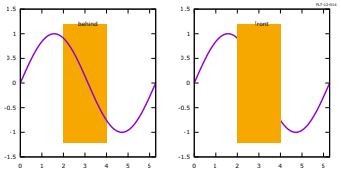
PLT-12-012

Clip a segment whose endpoints are both outside



PLT-12-013

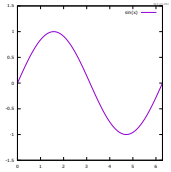
Highlight an x interval



PLT-12-014

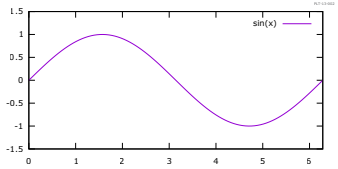
Compare object layers behind and in front

13 · Space, proportions and margins



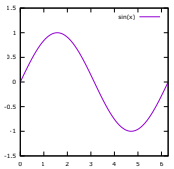
PLT-13-001

Change the canvas shape



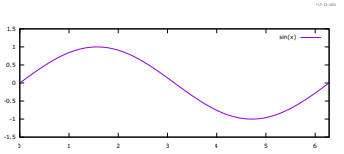
PLT-13-002

Change the overall font size



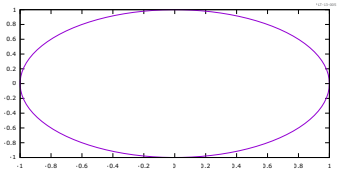
PLT-13-003

Make the plot region square



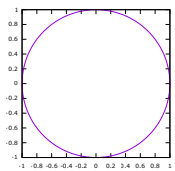
PLT-13-004

Set an explicit plot aspect ratio



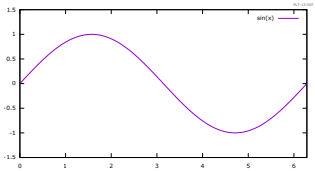
PLT-13-005

Compare geometry on automatic axes



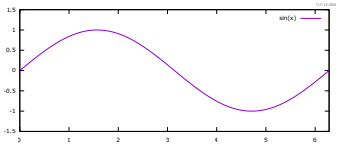
PLT-13-006

Give x and y equal physical units



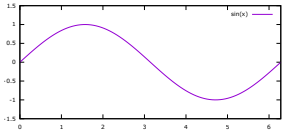
PLT-13-007

Reserve more left margin



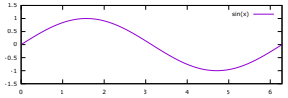
PLT-13-008

Reserve more bottom margin



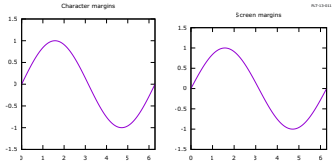
PLT-13-009

Place the plot borders explicitly



PLT-13-010

Reserve space above the plot



PLT-13-011

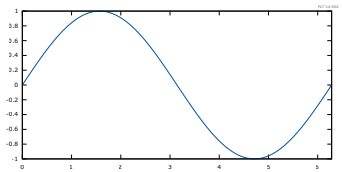
Compare character margins with screen margins

14 · Reusable styles, loops and command blocks



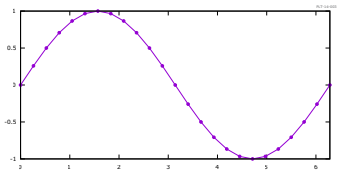
PLT-14-001

Distinguish line types from user line styles



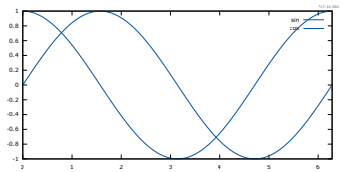
PLT-14-002

Replace a style fragment with a macro



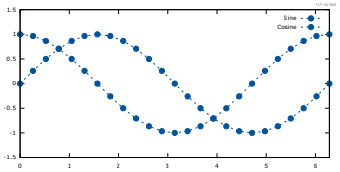
PLT-14-003

Change the macro in one place



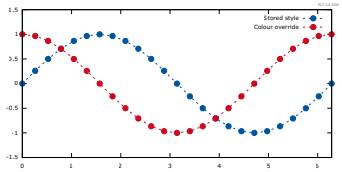
PLT-14-004

Reuse one style for two curves



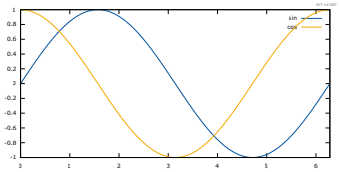
PLT-14-005

Define and reuse a numbered line style



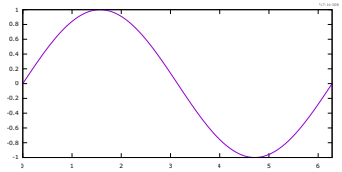
PLT-14-006

Override one property of a reusable style



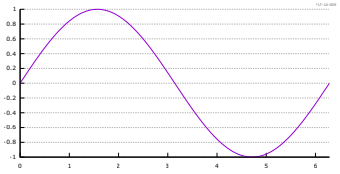
PLT-14-007

Share geometry but vary colour



PLT-14-008

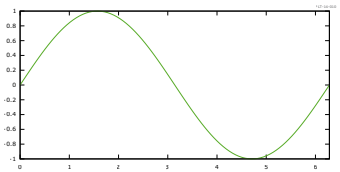
Put a range into a macro



PLT-14-009

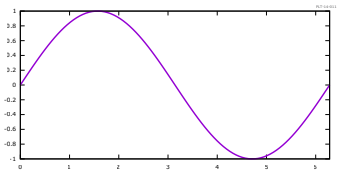
Bundle several setup commands

14 · Reusable styles, loops and command blocks (continued)



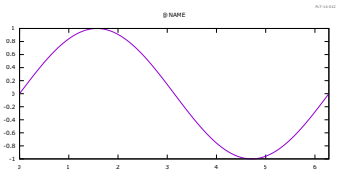
PLT-14-010

Build a macro with string concatenation



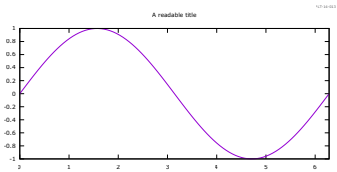
PLT-14-011

Format a numeric macro parameter



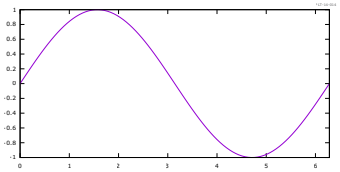
PLT-14-012

See where a macro does not expand



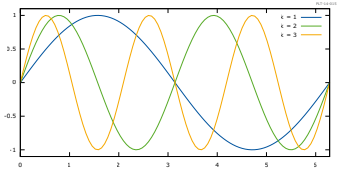
PLT-14-013

Pass the title variable correctly



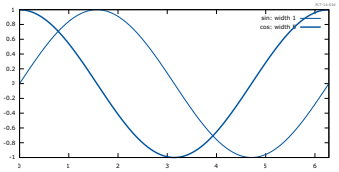
PLT-14-014

Execute constructed text with eval



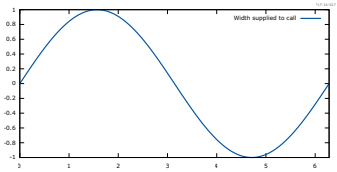
PLT-14-015

Combine a plot loop with a style macro



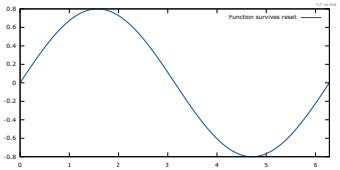
PLT-14-016

Build macro text with a function parameter



PLT-14-017

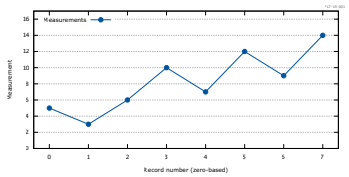
Pass parameters to a reusable command block



PLT-14-018

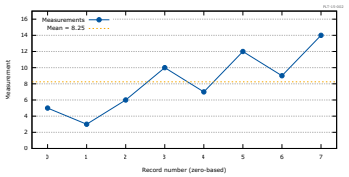
Reset plot settings within a session

15 · Statistics before plotting



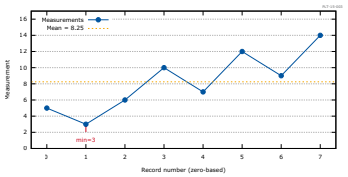
PLT-15-001

Plot measurements by record number



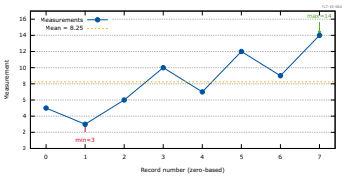
PLT-15-002

Calculate and draw the mean



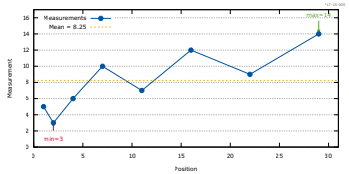
PLT-15-003

Locate the minimum by its record index



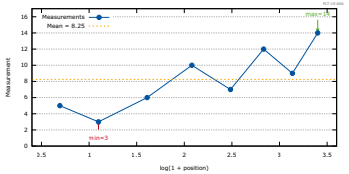
PLT-15-004

Locate the maximum as well



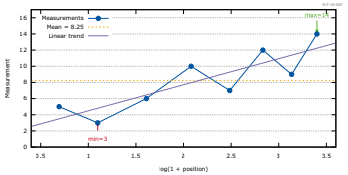
PLT-15-005

Use the actual x positions



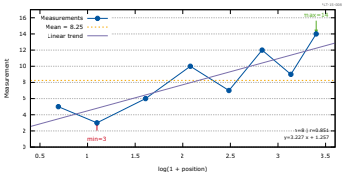
PLT-15-006

Transform x consistently for statistics and plotting



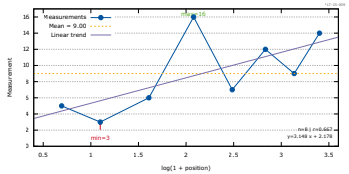
PLT-15-007

Draw the linear trend from two-column statistics



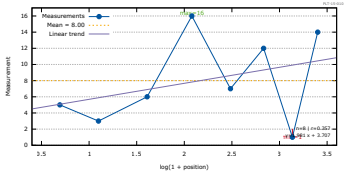
PLT-15-008

Show the computed summary beside the plot



PLT-15-009

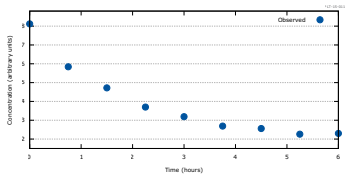
Move the maximum by changing only the data



PLT-15-010

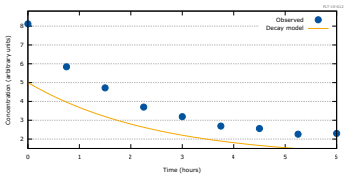
Move the minimum without editing annotations

15 · Statistics before plotting (continued)



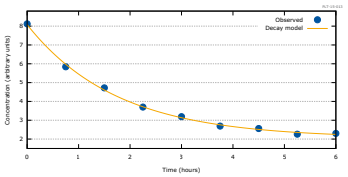
PLT-15-011

Begin with observations for an exponential fit



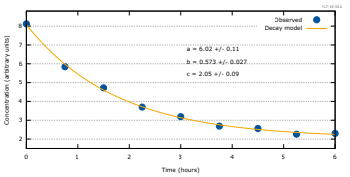
PLT-15-012

Overlay an initial decay model



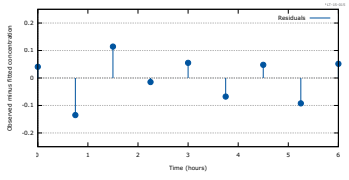
PLT-15-013

Let fit estimate the decay parameters



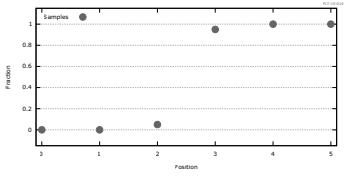
PLT-15-014

Display fitted values and their uncertainties



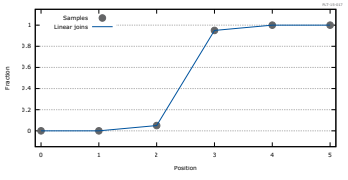
PLT-15-015

Inspect residuals after fitting



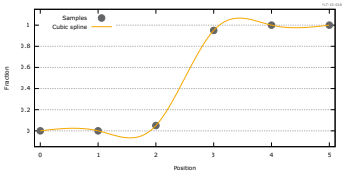
PLT-15-016

Start an interpolation with six samples



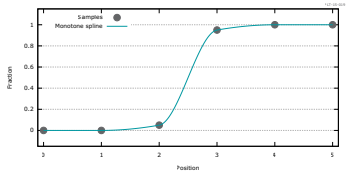
PLT-15-017

Connect samples with straight lines



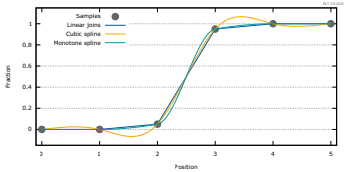
PLT-15-018

Replace joins with a cubic spline



PLT-15-019

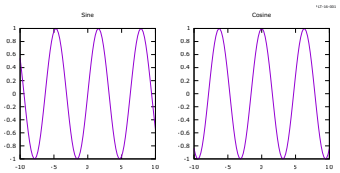
Preserve monotonicity with mcsplines



PLT-15-020

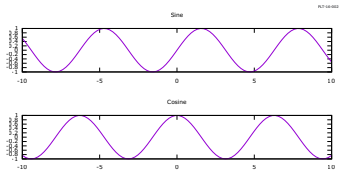
Compare interpolation choices in one figure

16 · Multiple panels



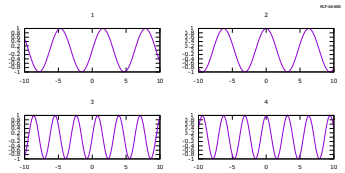
PLT-16-001

Create two side-by-side panels



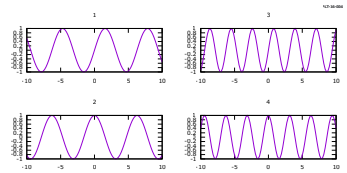
PLT-16-002

Stack the same panels vertically



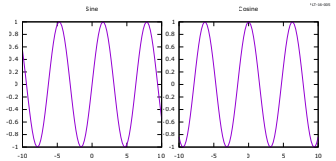
PLT-16-003

Use a two-by-two layout



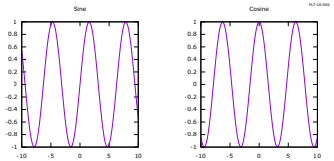
PLT-16-004

Fill panels down columns first



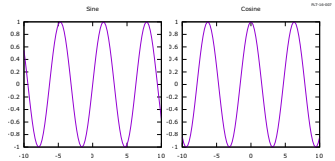
PLT-16-005

Specify shared outer margins



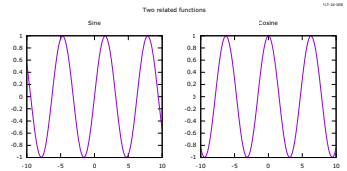
PLT-16-006

Increase the gap between columns



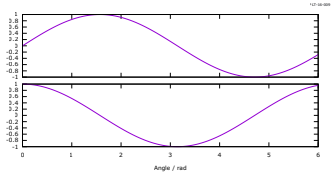
PLT-16-007

Reduce the gap again



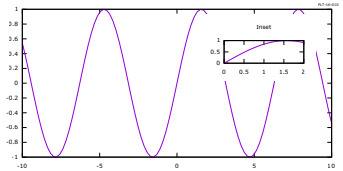
PLT-16-008

Add a shared figure title



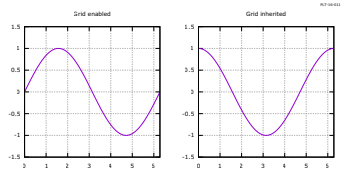
PLT-16-009

Share an x scale across stacked panels



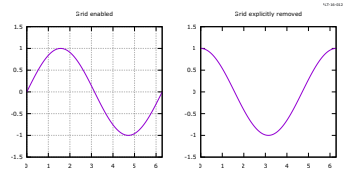
PLT-16-010

Create an inset manually



PLT-16-011

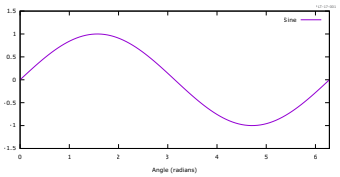
See settings persist between panels



PLT-16-012

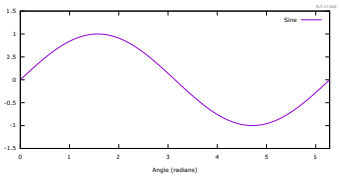
Clear a setting before the next panel

17 · Terminals and output formats



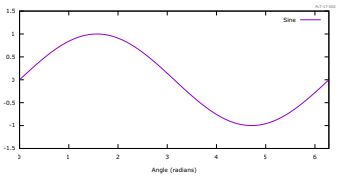
PLT-17-001

Write an SVG with explicit terminal settings



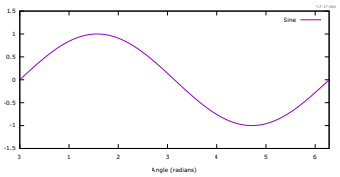
PLT-17-002

Export the same plot as a PNG image



PLT-17-003

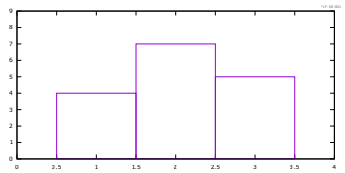
Export a PDF at a physical page size



PLT-17-004

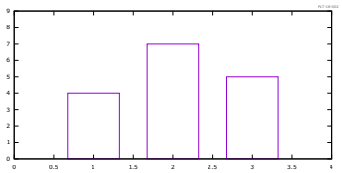
Export a PNG with a transparent background

18 · Bars and categorical histograms



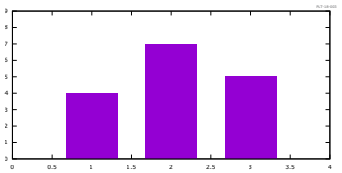
PLT-18-001

Start with three bars



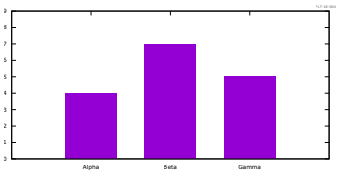
PLT-18-002

Set an exact bar width



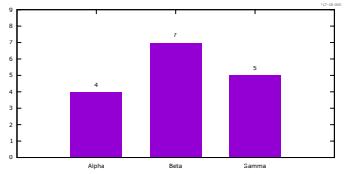
PLT-18-003

Fill the bars



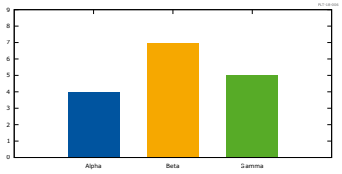
PLT-18-004

Use category names



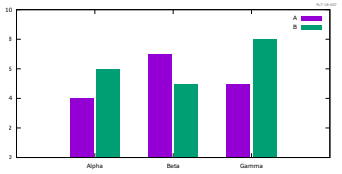
PLT-18-005

Write values above the bars



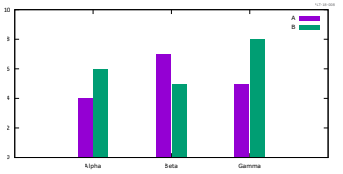
PLT-18-006

Colour each category



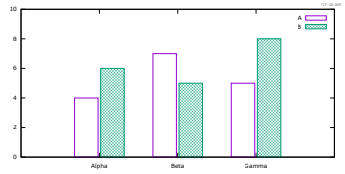
PLT-18-007

Put two series side by side



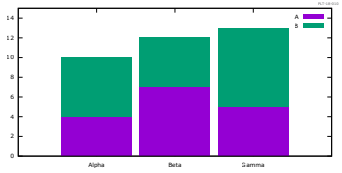
PLT-18-008

Widen the gap between groups



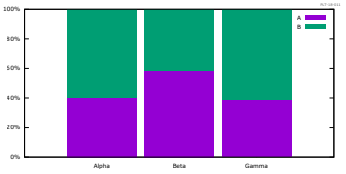
PLT-18-009

Use hatching for the series



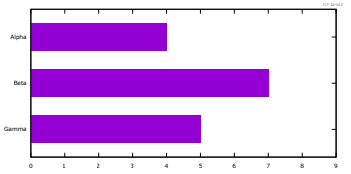
PLT-18-010

Stack the series by row



PLT-18-011

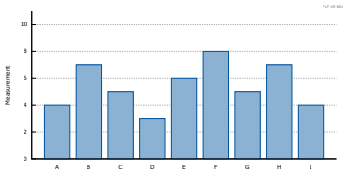
Normalize every stack to 100 percent



PLT-18-012

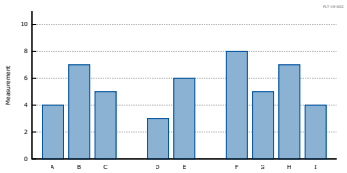
Turn bars horizontally

19 · Data-driven grouping brackets



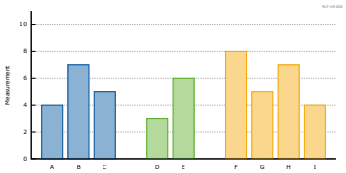
PLT-19-001

Start with labelled bars



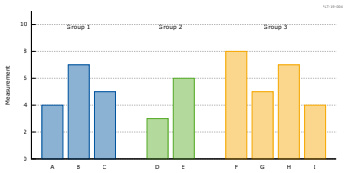
PLT-19-002

Leave one empty position between groups



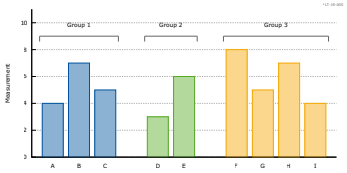
PLT-19-003

Colour bars by group



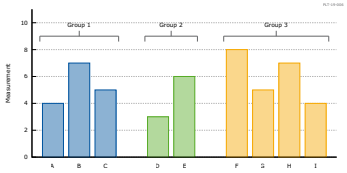
PLT-19-004

Derive group labels and widths from the data



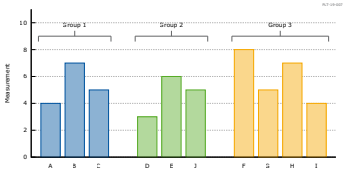
PLT-19-005

Scale a bracket to each group



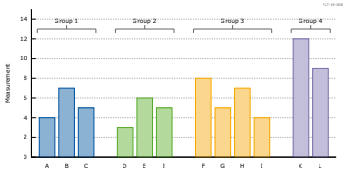
PLT-19-006

Add a centred stem to the reusable shape



PLT-19-007

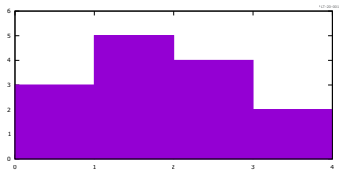
Widen a group by adding only a row



PLT-19-008

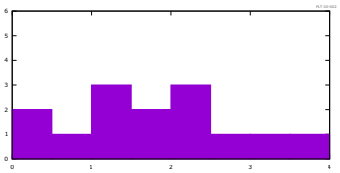
Detect an extra group and a new maximum

20 · Frequency histograms and binning



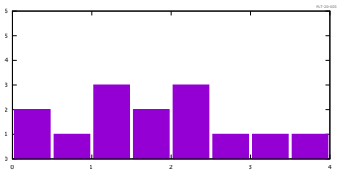
PLT-20-001

Bin raw observations into counts



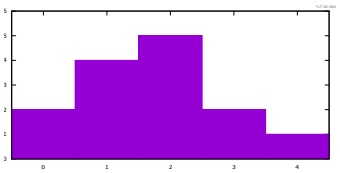
PLT-20-002

Halve the bin width



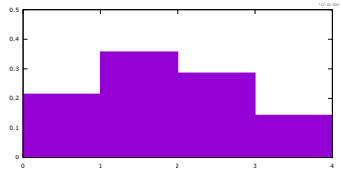
PLT-20-003

Leave small gaps between bins



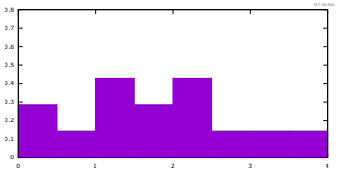
PLT-20-004

Shift the bin boundaries



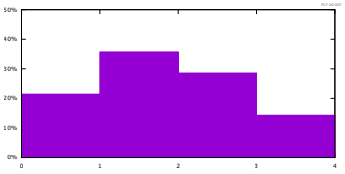
PLT-20-005

Show probability per bin



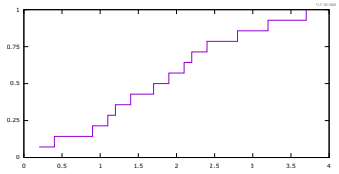
PLT-20-006

Make the histogram a density



PLT-20-007

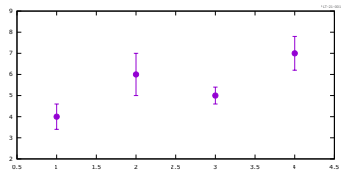
Display bin shares as percentages



PLT-20-008

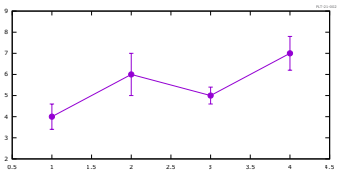
Show the empirical cumulative distribution

21 · Uncertainty, distributions and symbols



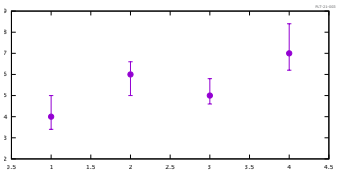
PLT-21-001

Add symmetric error bars



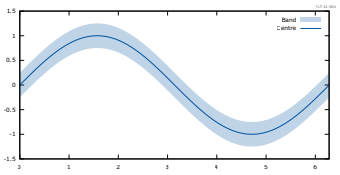
PLT-21-002

Connect the uncertain measurements



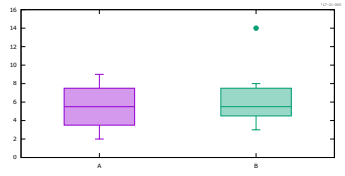
PLT-21-003

Use asymmetric bounds



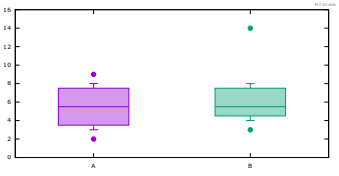
PLT-21-004

Show uncertainty as a ribbon



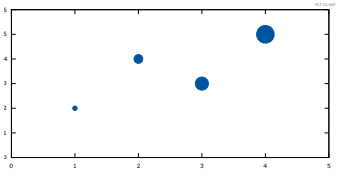
PLT-21-005

Summarize groups with boxplots



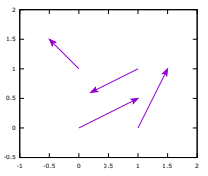
PLT-21-006

Change the boxplot whisker rule



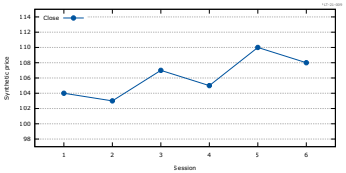
PLT-21-007

Encode a third quantity as bubble area



PLT-21-008

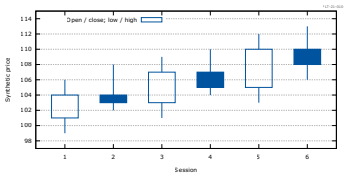
Draw vectors from component data



PLT-21-009

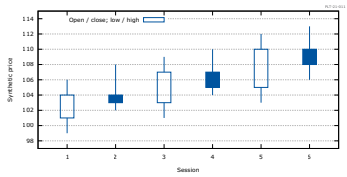
Begin a price chart with closing values

21 · Uncertainty, distributions and symbols (continued)



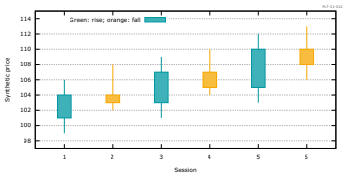
PLT-21-010

Show open, close, low and high as candlesticks



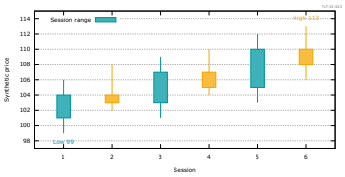
PLT-21-011

Narrow the candlestick bodies



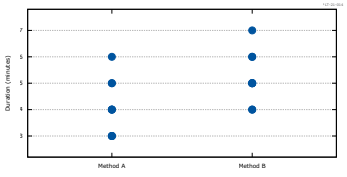
PLT-21-012

Colour rising and falling sessions from the data



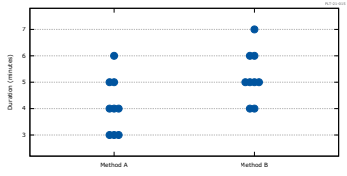
PLT-21-013

Annotate the automatically detected high and low



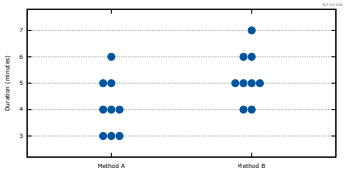
PLT-21-014

See why repeated points hide observations



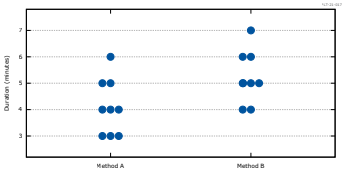
PLT-21-015

Separate overlapping observations with jitter



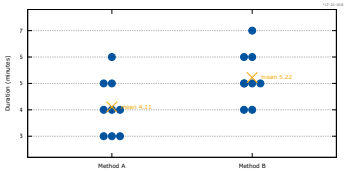
PLT-21-016

Increase the swarm separation



PLT-21-017

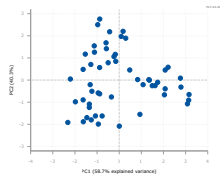
Limit horizontal displacement with wrap



PLT-21-018

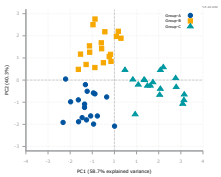
Overlay automatically computed group means

22 · Scatter plots and PCA



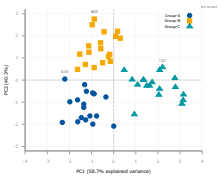
PLT-22-001

Plot prepared PCA scores



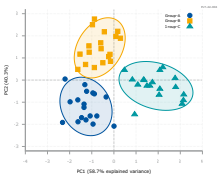
PLT-22-002

Distinguish PCA groups by colour and symbol



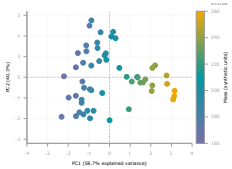
PLT-22-003

Label selected PCA samples



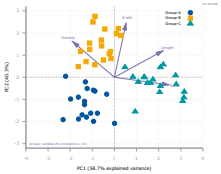
PLT-22-004

Draw PCA group-spread ellipses



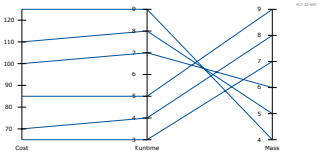
PLT-22-005

Colour PCA points by a continuous measurement



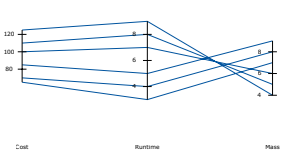
PLT-22-006

Overlay variable arrows in a PCA biplot



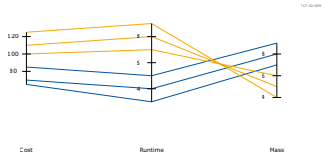
PLT-22-007

Connect multiple measurements with parallel axes



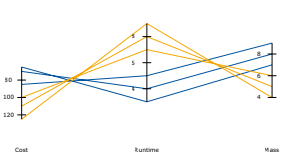
PLT-22-008

Fix meaningful ranges for each parallel axis



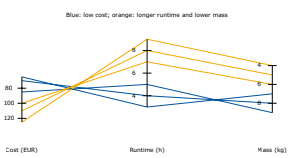
PLT-22-009

Colour parallel lines by design group



PLT-22-010

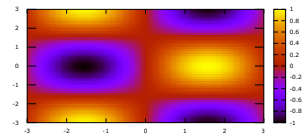
Reverse the cost axis to align preferences



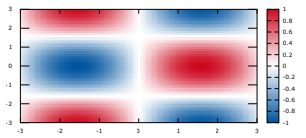
PLT-22-011

Align all axes so higher means preferable

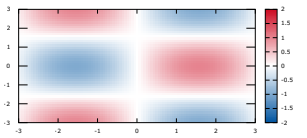
23 · Colour scales and map layout



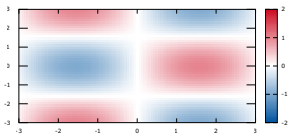
PLT-23-001
Start with a simple colour map



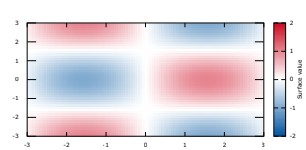
PLT-23-002
Choose a diverging palette



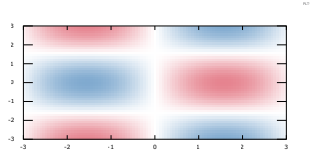
PLT-23-003
Expand the colour range



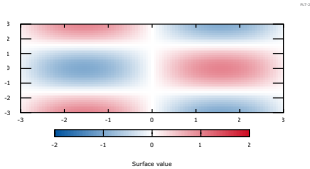
PLT-23-004
Set colourbar tick spacing



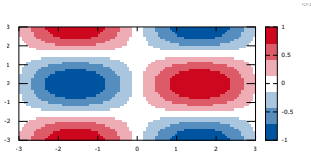
PLT-23-005
Label the colourbar



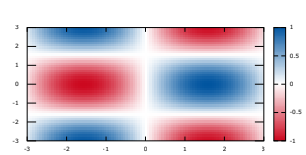
PLT-23-006
Hide the colourbar



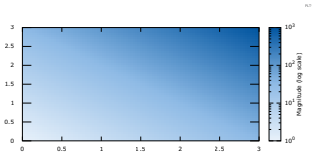
PLT-23-007
Place a horizontal colourbar



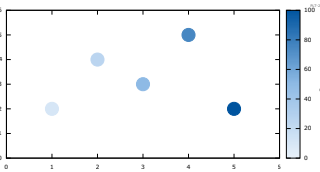
PLT-23-008
Reduce a palette to discrete colour bands



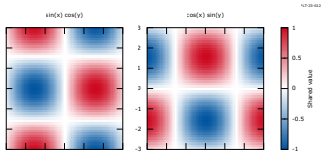
PLT-23-009
Reverse the palette direction



PLT-23-010
Use a logarithmic colour scale

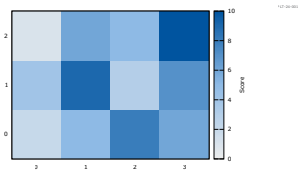


PLT-23-011
Colour points using a third data column



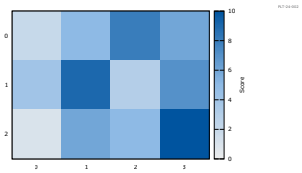
PLT-23-012
Share one colourbar across two panels

24 · Matrix heatmaps



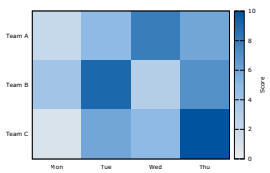
PLT-24-001

Draw a heatmap from a matrix



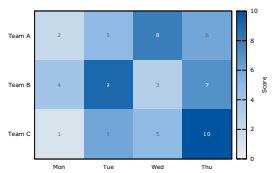
PLT-24-002

Put the first matrix row at the top



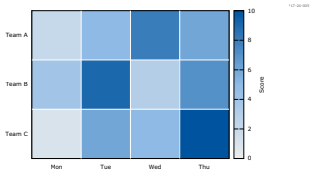
PLT-24-003

Label heatmap rows and columns



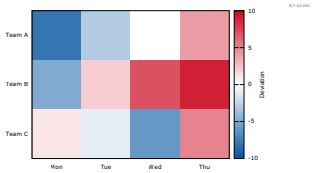
PLT-24-004

Print the value inside every cell



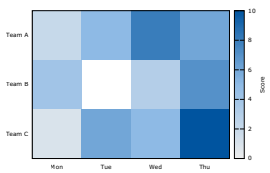
PLT-24-005

Draw lines between heatmap cells



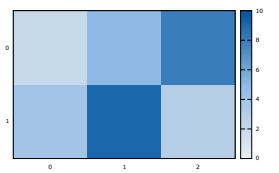
PLT-24-006

Use a symmetric palette for signed values



PLT-24-007

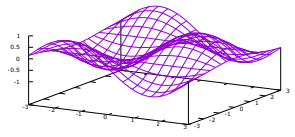
Leave missing heatmap cells blank



PLT-24-008

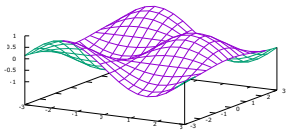
Read a heatmap as x, y and value triples

25 · Surfaces, contours and polar curves



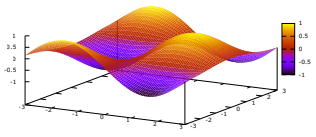
PLT-25-001

Draw a wireframe surface



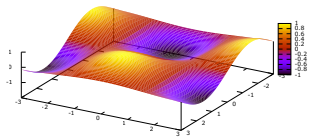
PLT-25-002

Remove hidden wireframe lines



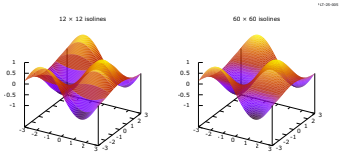
PLT-25-003

Colour the surface height



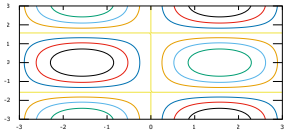
PLT-25-004

Change the surface viewing angle



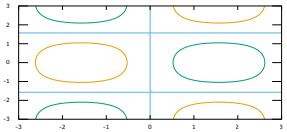
PLT-25-005

Compare coarse and fine surface sampling



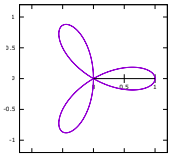
PLT-25-006

Project contour lines into a map



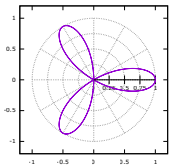
PLT-25-007

Choose only specific contour levels



PLT-25-008

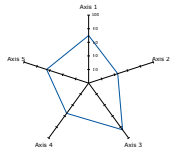
Draw a polar curve



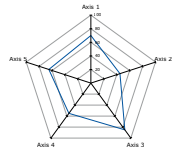
PLT-25-009

Add a polar grid

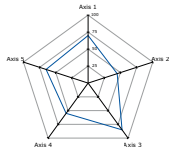
26 · Spider plots and radar charts



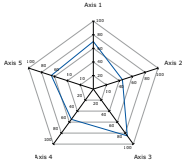
PLT-26-001
Start a native spider plot



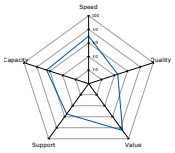
PLT-26-002
Add the polygonal spiderweb



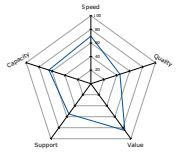
PLT-26-003
Change the number of web rings



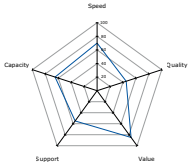
PLT-26-004
Show numeric notation on every spoke



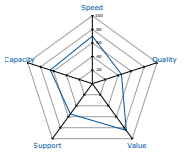
PLT-26-005
Give each spoke a meaningful name



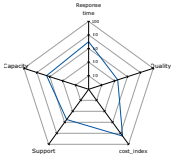
PLT-26-006
Rotate individual spoke labels



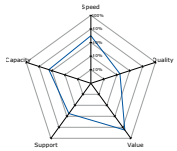
PLT-26-007
Move spoke names radially outward



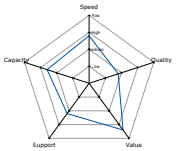
PLT-26-008
Style the spoke-name font and colour



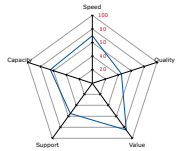
PLT-26-009
Use multiline and literal spoke names



PLT-26-010
Format a spoke scale as percentages

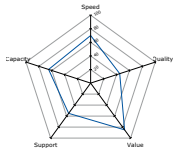


PLT-26-011
Use words for radial tick levels



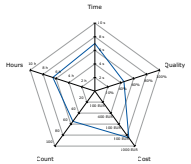
PLT-26-012
Change numeric tick text and offset

26 · Spider plots and radar charts (continued)



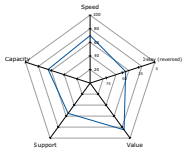
PLT-26-013

Rotate numeric tick labels explicitly



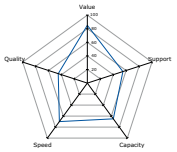
PLT-26-014

Give spokes different units and ranges



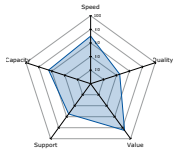
PLT-26-015

Reverse a lower-is-better spoke



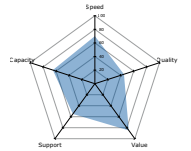
PLT-26-016

Choose which variable starts at the top



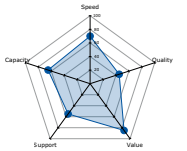
PLT-26-017

Fill the polygon transparently



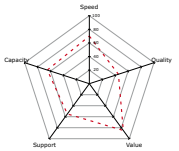
PLT-26-018

Remove the polygon border



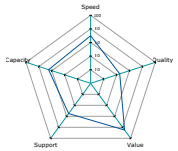
PLT-26-019

Mark every polygon vertex



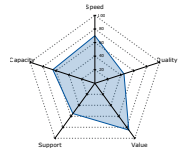
PLT-26-020

Choose a dashed polygon outline



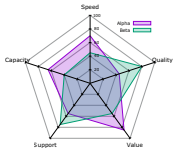
PLT-26-021

Style the radial spoke lines



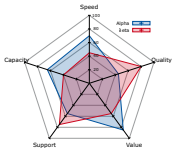
PLT-26-022

Style and layer the web grid



PLT-26-023

Compare several rows and name the series



PLT-26-024

Combine separate datasets with newspiderplot

27 · Sectors, pies and donuts



PLT-27-001
Draw a single sector



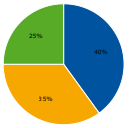
PLT-27-002
Complete a three-slice pie



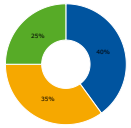
PLT-27-003
Assign distinct slice colours



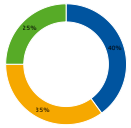
PLT-27-004
Separate slices with white borders



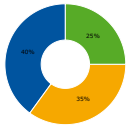
PLT-27-005
Label the percentages



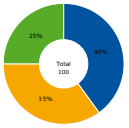
PLT-27-006
Open a donut hole



PLT-27-007
Make the donut ring thinner

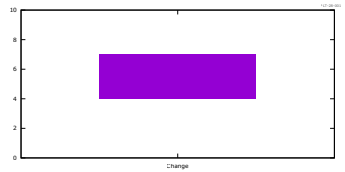


PLT-27-008
Reverse the angular direction



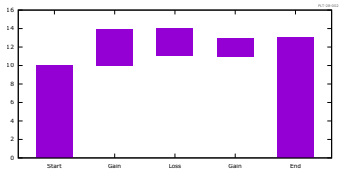
PLT-27-009
Add a central total

28 · Waterfall totals and floating bars



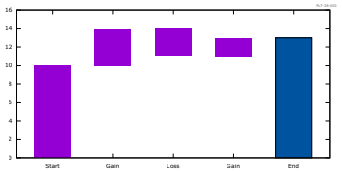
PLT-28-001

Start with one floating rectangle



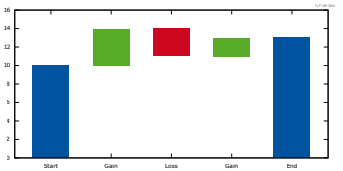
PLT-28-002

Arrange a sequence of intervals



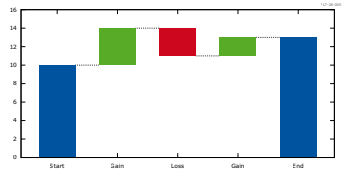
PLT-28-003

Derive intervals from signed changes



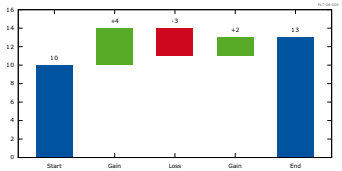
PLT-28-004

Colour totals, gains and losses



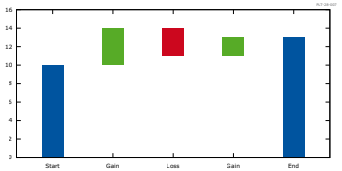
PLT-28-005

Connect the running levels



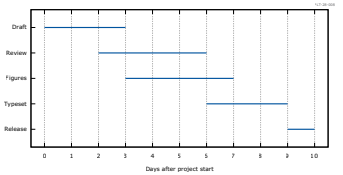
PLT-28-006

Print the signed changes



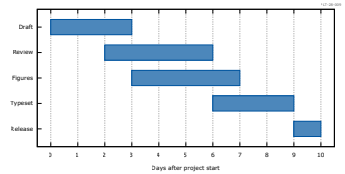
PLT-28-007

Narrow the waterfall bars



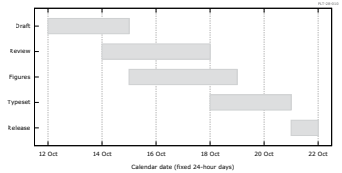
PLT-28-008

Draw a small schedule from task intervals



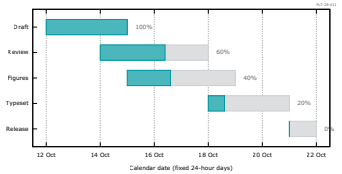
PLT-28-009

Replace schedule lines with interval boxes



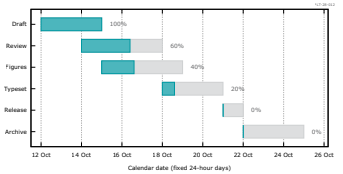
PLT-28-010

Show calendar dates on the schedule



PLT-28-011

Overlay the completed fraction of each task



PLT-28-012

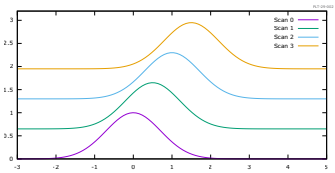
Adapt the schedule range when a task is added

29 · Offset curves and signal waterfalls



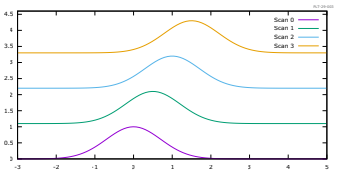
PLT-29-001

Draw a family of related curves



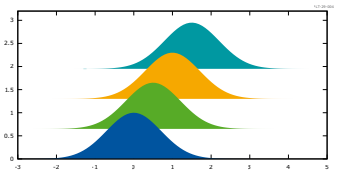
PLT-29-002

Separate curves with a vertical offset



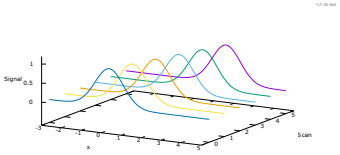
PLT-29-003

Increase the separation



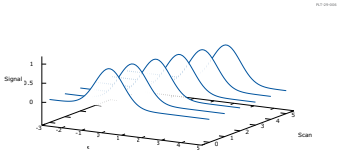
PLT-29-004

Fill offset curves back to front



PLT-29-005

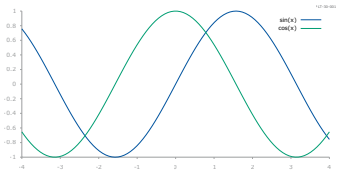
Move the signal waterfall into 3D



PLT-29-006

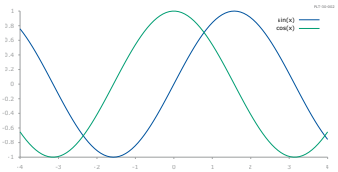
Fill the 3D waterfall slices

30 · Custom styles and loading



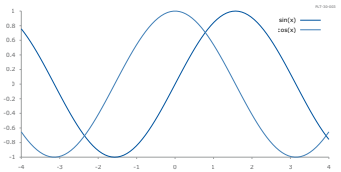
PLT-30-001

Define a reusable style file



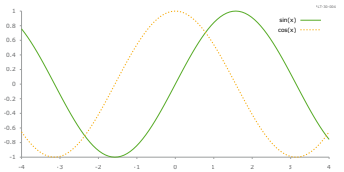
PLT-30-002

Load a project or personal style



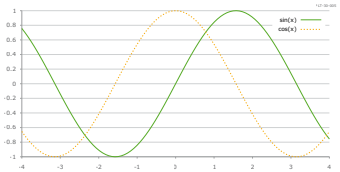
PLT-30-003

Load RWTH–Tufte for a line plot



PLT-30-004

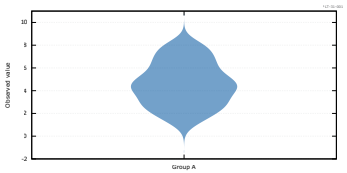
Use the library’s named colours



PLT-30-005

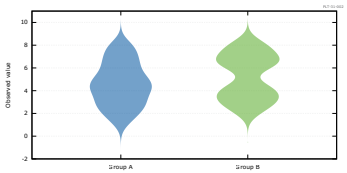
Override a style for one plot

31 · Violin plots and density choices



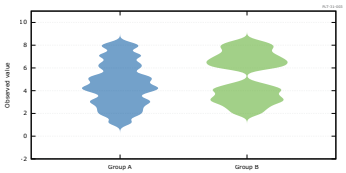
PLT-31-001

Mirror one empirical density



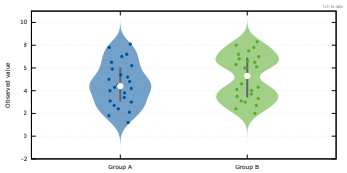
PLT-31-002

Compare two groups on a common scale



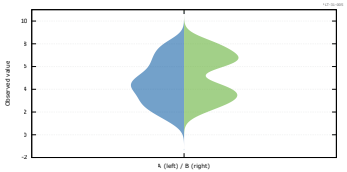
PLT-31-003

Expose the effect of a smaller bandwidth



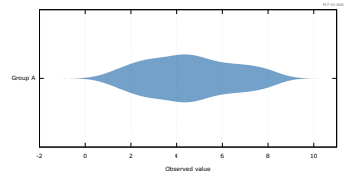
PLT-31-004

Overlay observations and quartiles



PLT-31-005

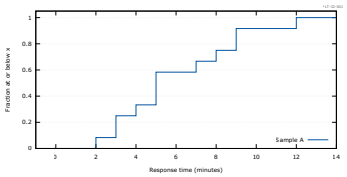
Split two groups across one violin



PLT-31-006

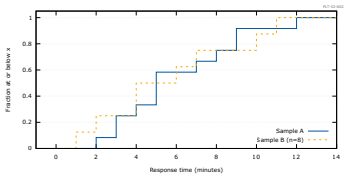
Turn the density sideways with native kdensity

32 · ECDFs, exceedance and probability bands



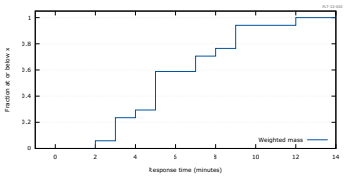
PLT-32-001

Draw a right-continuous ECDF with ties



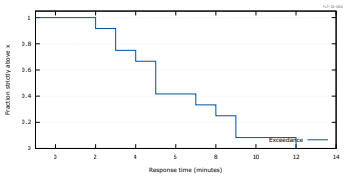
PLT-32-002

Compare samples of different sizes



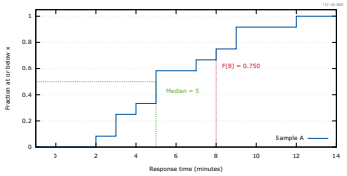
PLT-32-003

Accumulate observation weights



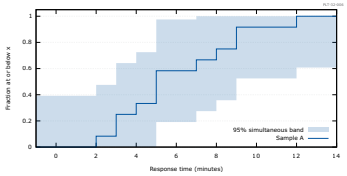
PLT-32-004

Plot the fraction strictly above a threshold



PLT-32-005

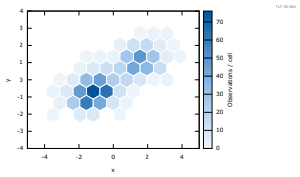
Read the empirical median and a service threshold



PLT-32-006

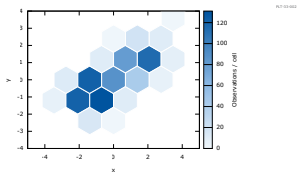
Add a simultaneous DKW confidence band

33 · Hexbin density from scattered observations



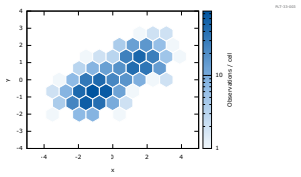
PLT-33-001

Count observations in true hexagonal bins



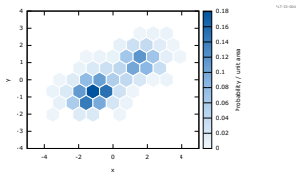
PLT-33-002

Trade detail for larger bins



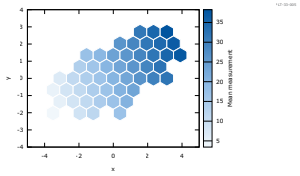
PLT-33-003

Reveal sparse cells with logarithmic colour



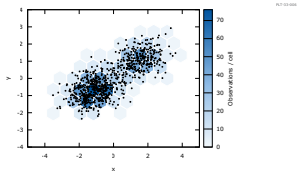
PLT-33-004

Normalize counts to probability density



PLT-33-005

Map a measurement mean instead of frequency



PLT-33-006

Overlay the original observations

34 · Sankey diagrams with conserved flows



PLT-34-001

Connect two nodes with one flow



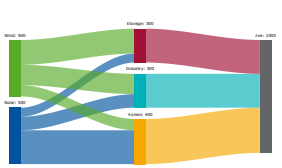
PLT-34-002

Split a source into three destinations



PLT-34-003

Combine multiple sources and destinations



PLT-34-004

Carry the same flow through a third stage



PLT-34-005

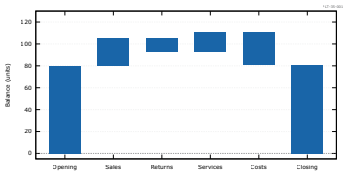
Change node order to inspect crossings



PLT-34-006

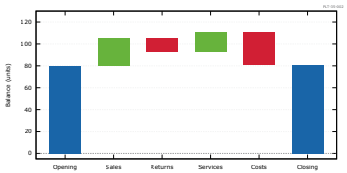
Normalize the flow to a 100-percent budget

35 · Waterfall chart variants and subtotals



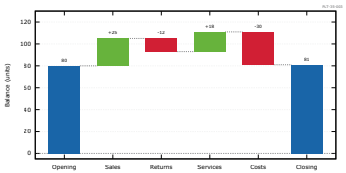
PLT-35-001

Compute a bridge from signed contributions



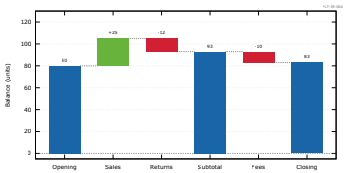
PLT-35-002

Distinguish increases, decreases and totals



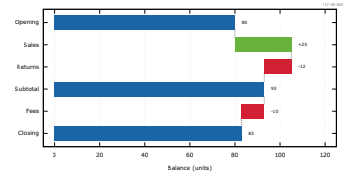
PLT-35-003

Connect levels and label every contribution



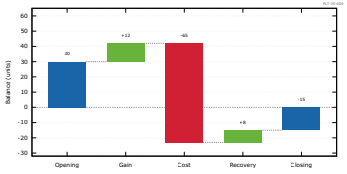
PLT-35-004

Insert a subtotal without double counting



PLT-35-005

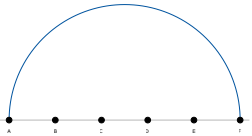
Turn the bridge into horizontal bars



PLT-35-006

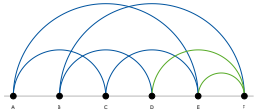
Keep negative totals and zero crossings correct

36 · Arc diagrams for relationships



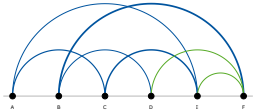
PLT-36-001

Connect two ordered nodes with a semicircle



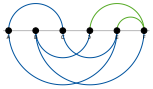
PLT-36-002

Display a network as nested arcs



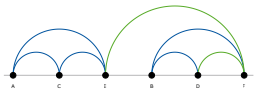
PLT-36-003

Encode relationship weight with stroke width



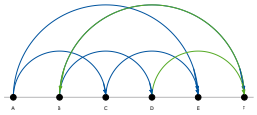
PLT-36-004

Separate within-group and cross-group links



PLT-36-005

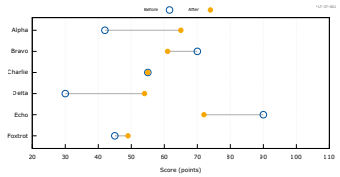
Change the node order explicitly



PLT-36-006

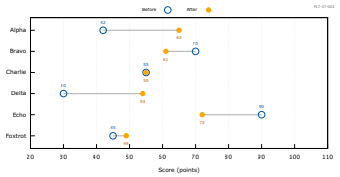
Add direction at the target endpoints

37 · Dumbbell charts and paired comparisons



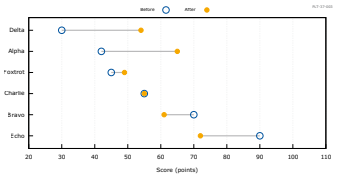
PLT-37-001

Connect before and after values horizontally



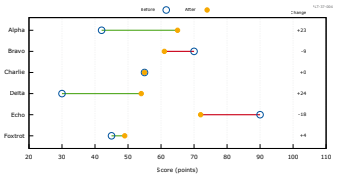
PLT-37-002

Label both endpoints directly



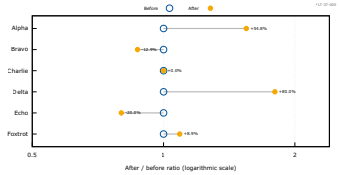
PLT-37-003

Sort categories by their signed change



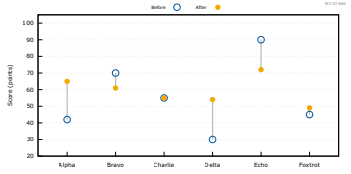
PLT-37-004

Colour increases, decreases and unchanged pairs



PLT-37-005

Compare relative changes on a logarithmic scale



PLT-37-006

Turn paired comparisons into vertical dumbbells

Complete plotting commands

The main panel highlights the plotting commands that make each figure. Expand the complete Bash block to see the setup as well. A quoted heredoc passes gnuplot expressions literally, and each example starts a fresh plotting process.

Plot setup explained

`gnuplot -d`	Start a fresh process and ignore startup configuration files.
`<<'GP' ... GP`	Quoted Bash heredoc: pass $D, $1 and other gnuplot syntax literally. The closing GP stands alone.
`terminal svg size 1200,600`	Use an SVG canvas 1200 pixels wide and 600 high, unless an example explicitly changes it.
`enhanced font "sans,20"`	Enable enhanced text; choose nominal font size 20. Rendering depends on the terminal and installed fonts.
`encoding utf8; white background`	Use UTF-8 text and an opaque white canvas.
`set output 'figures/ID.svg'`	Select the output filename; unset output closes it.
`label 999 at screen 0.985,0.975`	Reserved ID near the upper-right corner. Screen positions are canvas fractions; right aligns text, front draws it above other layers.
`font ",12"; textcolor #646567`	Use the current font family at nominal size 12, in RWTH grey.

IDs identify plots; page numbers locate them

An ID such as `PLT-15-003` identifies the tour (`PLT`), chapter (`15`) and example within that chapter (`003`). Examples start at `001` in each chapter; `000` is the chapter anchor. IDs are independent of PDF pages.

Use an ID to open an example directly, follow a comparison or find its downloadable files. The plot index maps IDs to topics and filenames.

Topic	ID range	Plots
01 · First picture	PLT-01-001 … PLT-01-010	10
02 · Variables, functions and arithmetic	PLT-02-001 … PLT-02-012	12
03 · Reading and selecting data	PLT-03-001 … PLT-03-009	9
04 · Major tick marks	PLT-04-001 … PLT-04-010	10
05 · Axis ranges and scales	PLT-05-001 … PLT-05-012	12
06 · Tick labels and formatting	PLT-06-001 … PLT-06-015	15
07 · Minor ticks and grids	PLT-07-001 … PLT-07-015	15
08 · Axis labels and titles	PLT-08-001 … PLT-08-008	8
09 · Line, point and fill styles	PLT-09-001 … PLT-09-020	20
10 · Legend position and layout	PLT-10-001 … PLT-10-014	14
11 · Borders and reference lines	PLT-11-001 … PLT-11-012	12
12 · Labels, arrows and shaded regions	PLT-12-001 … PLT-12-014	14
13 · Space, proportions and margins	PLT-13-001 … PLT-13-011	11
14 · Reusable styles, loops and command blocks	PLT-14-001 … PLT-14-018	18
15 · Statistics before plotting	PLT-15-001 … PLT-15-020	20
16 · Multiple panels	PLT-16-001 … PLT-16-012	12
17 · Terminals and output formats	PLT-17-001 … PLT-17-004	4
18 · Bars and categorical histograms	PLT-18-001 … PLT-18-012	12
19 · Data-driven grouping brackets	PLT-19-001 … PLT-19-008	8
20 · Frequency histograms and binning	PLT-20-001 … PLT-20-008	8
21 · Uncertainty, distributions and symbols	PLT-21-001 … PLT-21-018	18
22 · Scatter plots and PCA	PLT-22-001 … PLT-22-011	11
23 · Colour scales and map layout	PLT-23-001 … PLT-23-012	12
24 · Matrix heatmaps	PLT-24-001 … PLT-24-008	8
25 · Surfaces, contours and polar curves	PLT-25-001 … PLT-25-009	9
26 · Spider plots and radar charts	PLT-26-001 … PLT-26-024	24
27 · Sectors, pies and donuts	PLT-27-001 … PLT-27-009	9
28 · Waterfall totals and floating bars	PLT-28-001 … PLT-28-012	12
29 · Offset curves and signal waterfalls	PLT-29-001 … PLT-29-006	6

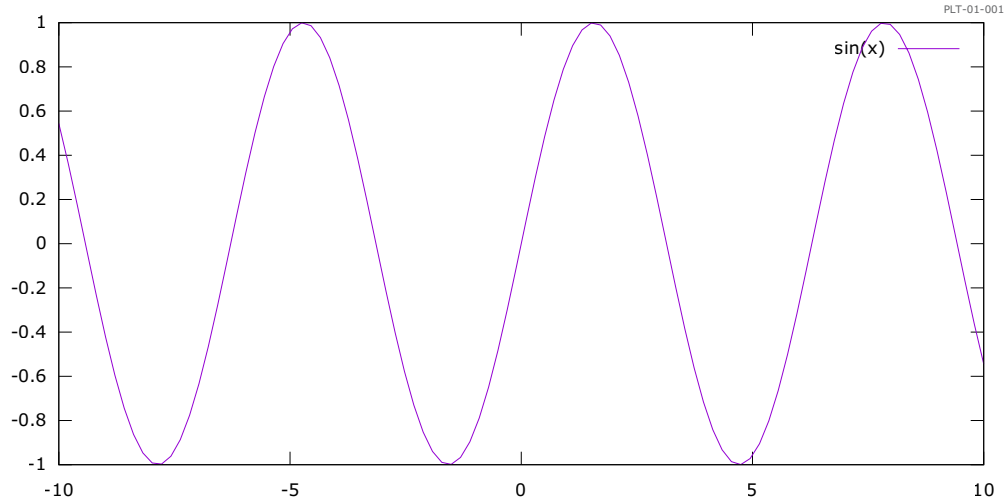
Topic	ID range	Plots
30 · Custom styles and loading	PLT-30-001 ... PLT-30-005	5
31 · Violin plots and density choices	PLT-31-001 ... PLT-31-006	6
32 · ECDFs, exceedance and probability bands	PLT-32-001 ... PLT-32-006	6
33 · Hexbin density from scattered observations	PLT-33-001 ... PLT-33-006	6
34 · Sankey diagrams with conserved flows	PLT-34-001 ... PLT-34-006	6
35 · Waterfall chart variants and subtotals	PLT-35-001 ... PLT-35-006	6
36 · Arc diagrams for relationships	PLT-36-001 ... PLT-36-006	6
37 · Dumbbell charts and paired comparisons	PLT-37-001 ... PLT-37-006	6

Start with one command

New baseline: the commands below define this example.

gnuplot can draw a mathematical expression directly. The automatic axes, border and legend are a useful starting point.

```
plot sin(x)
```



PLT-01-001 · Generated from the complete Bash block

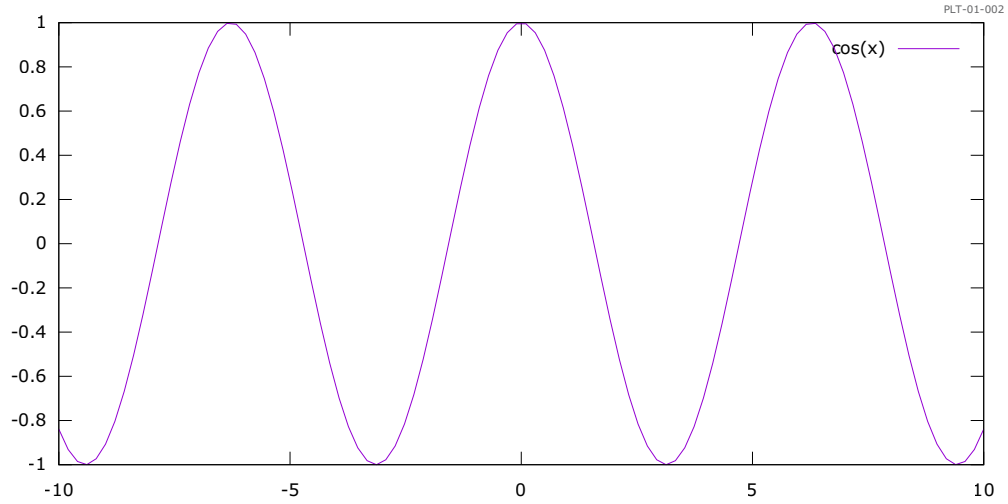
`plot`	Draw a two-dimensional graph.
`sin(x)`	Evaluate the sine function; its argument is in radians.
`x`	The independent variable sampled along the horizontal axis.

Change the function

Compare with PLT-01-001 · Start with one command.

Replace sine with cosine; the plotting command stays the same.

```
plot cos(x)
```



PLT-01-002 · Generated from the complete Bash block

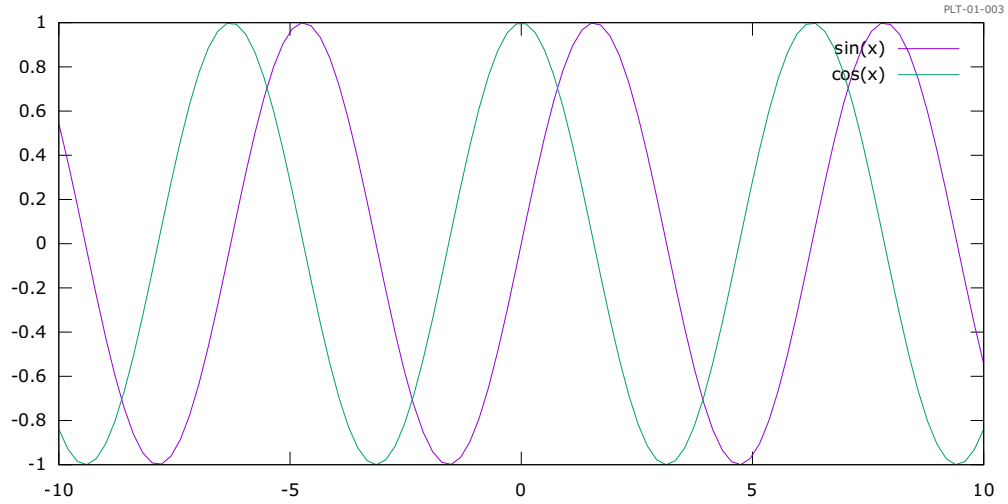
`cos(x)`	The cosine function, also using radians.

Add a second curve

Compare with PLT-01-001 · Start with one command.

A comma adds another curve to the same coordinate system.

```
plot sin(x), cos(x)
```



PLT-01-003 · Generated from the complete Bash block

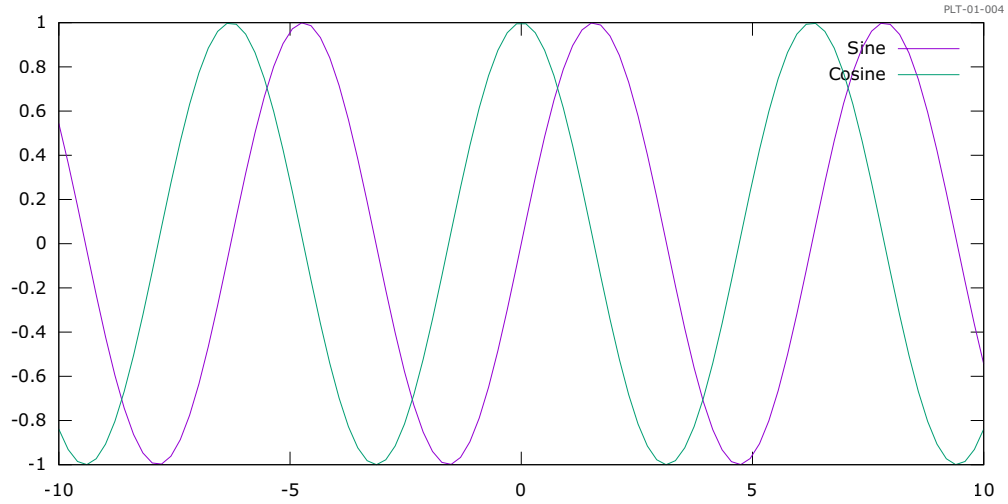
,	Separates plot elements within a single plot command.

Name the curves

Compare with PLT-01-003 · Add a second curve.

Explicit titles replace the expression text in the legend.

```
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-01-004 · Generated from the complete Bash block

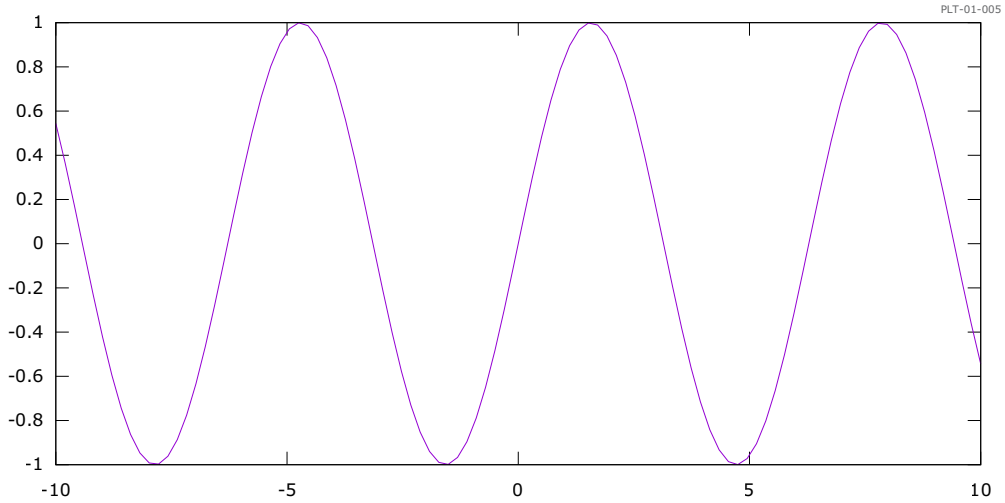
`title`	Set the legend text for that curve.
`'Sine', 'Cosine'`	Quoted text is a string. Each title belongs to the preceding expression.

Hide the legend

Compare with PLT-01-001 · Start with one command.

Remove the legend while preserving the curve.

```
unset key
plot sin(x)
```



PLT-01-005 · Generated from the complete Bash block

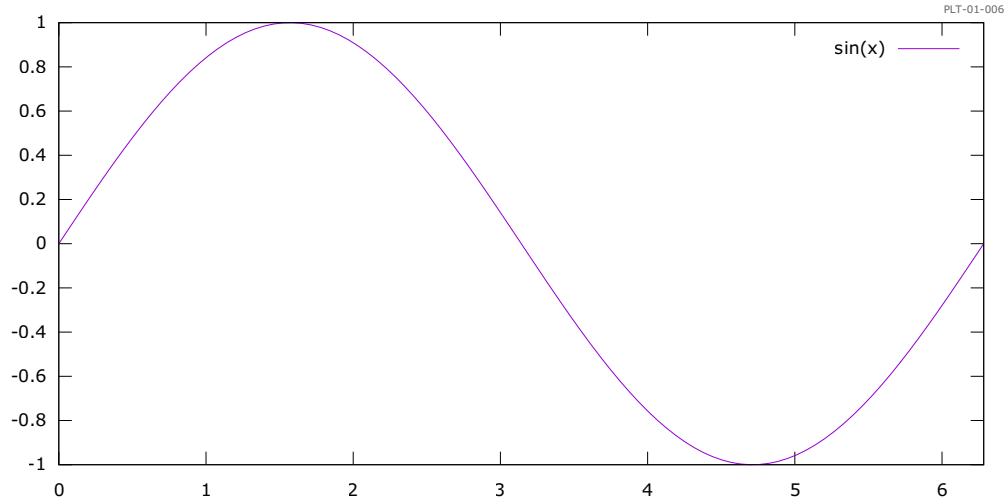
key	gnuplot's name for the legend.
unset	Disable the named setting.

Choose the horizontal interval

Compare with PLT-01-001 · Start with one command.

Restrict the plot to one complete sine period.

```
set xrange [0:2*pi]
plot sin(x)
```



PLT-01-006 · Generated from the complete Bash block

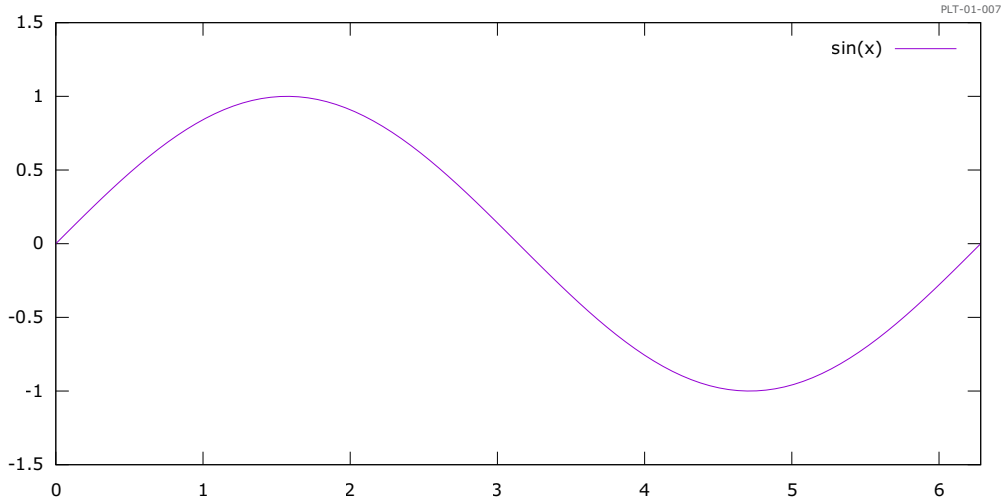
`set xrange`	Set the horizontal axis limits.
`[0:2*pi]`	Lower limit 0, upper limit 2π. The colon separates the two bounds.
`pi; *`	pi is the built-in constant; * means multiplication.

Leave room above and below

Compare with PLT-01-006 · Choose the horizontal interval.

Fix the vertical range so the curve no longer touches the top and bottom.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
plot sin(x)
```



PLT-01-007 · Generated from the complete Bash block

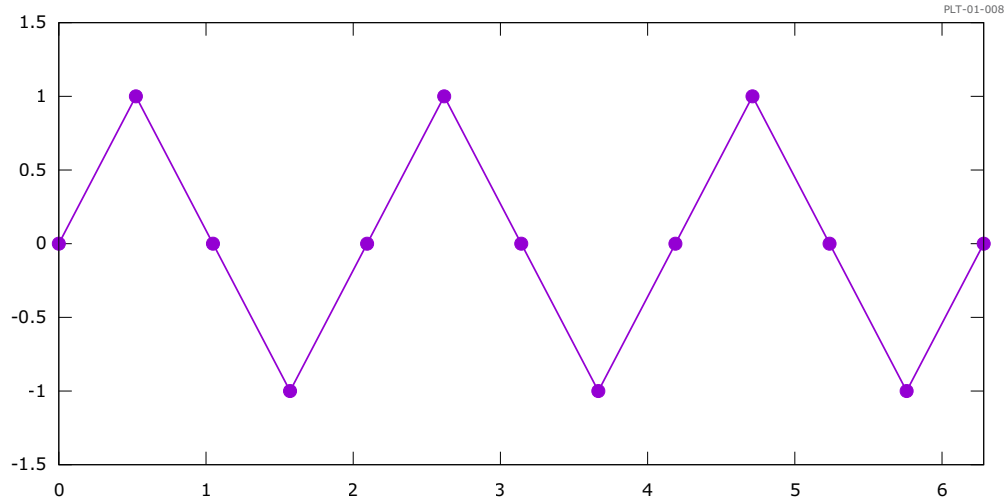
`yrange`	The vertical axis range.
`-1.5:1.5`	Lower and upper y limits, in data units.

See coarse sampling of a function

New baseline: the commands below define this example.

gnuplot draws analytic functions by evaluating a finite set of points and connecting them. Too few samples can make a smooth function appear angular or miss features.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 13
plot sin(3*x) with linespoints pt 7 lw 2 notitle
```



PLT-01-008 · Generated from the complete Bash block

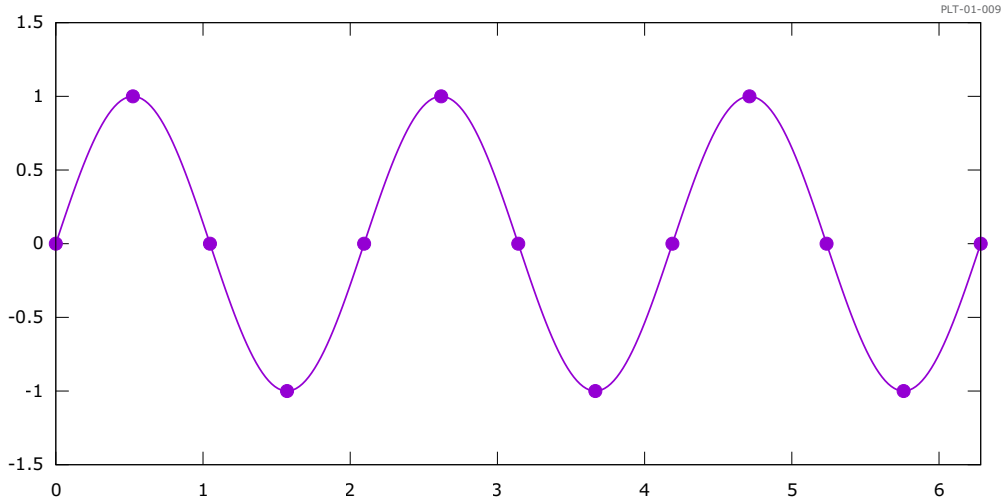
`samples 13`	Evaluate the function at thirteen x positions.
`linespoints`	Expose the sample positions and the connecting straight segments.

Increase samples without changing the function

Compare with PLT-01-008 · See coarse sampling of a function.

More samples improve the approximation and take more work. This setting controls function evaluation; it does not add observations to a data file.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 301
plot sin(3*x) with linespoints pt 7 pi 25 lw 2 notitle
```



PLT-01-009 · Generated from the complete Bash block

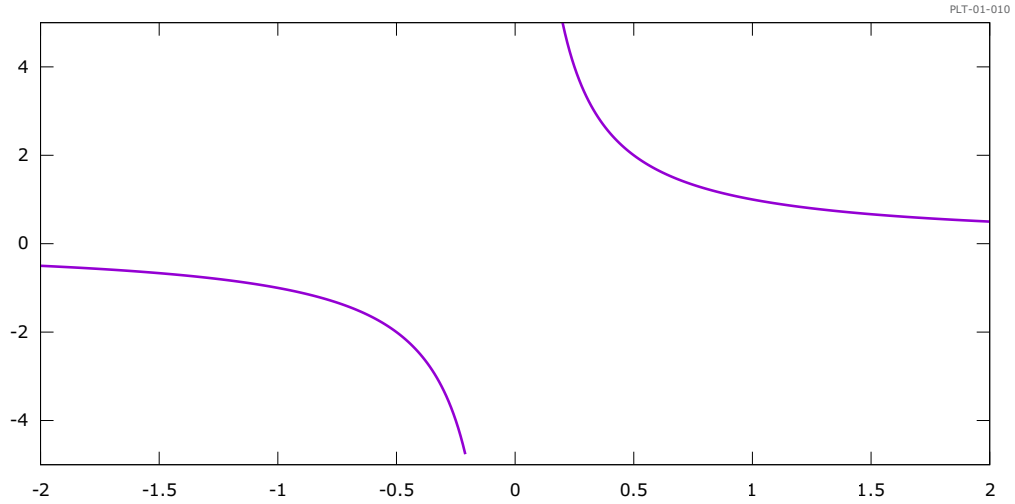
`samples 301`	Use a denser function-evaluation grid.
`pi 25`	Keep markers sparse while retaining the dense line.
`data plots`	Their resolution depends on the supplied data, not set samples.

Break a line at an excluded interval

New baseline: the commands below define this example.

Conditional undefined values create a gap. More samples alone do not reliably prevent spurious lines across discontinuities or excluded regions.

```
set xrange [-2:2]
set yrange [-5:5]
set samples 401
f(x) = abs(x)<0.2 ? 1/0 : 1/x
plot f(x) with lines lw 3 notitle
```



PLT-01-010 · Generated from the complete Bash block

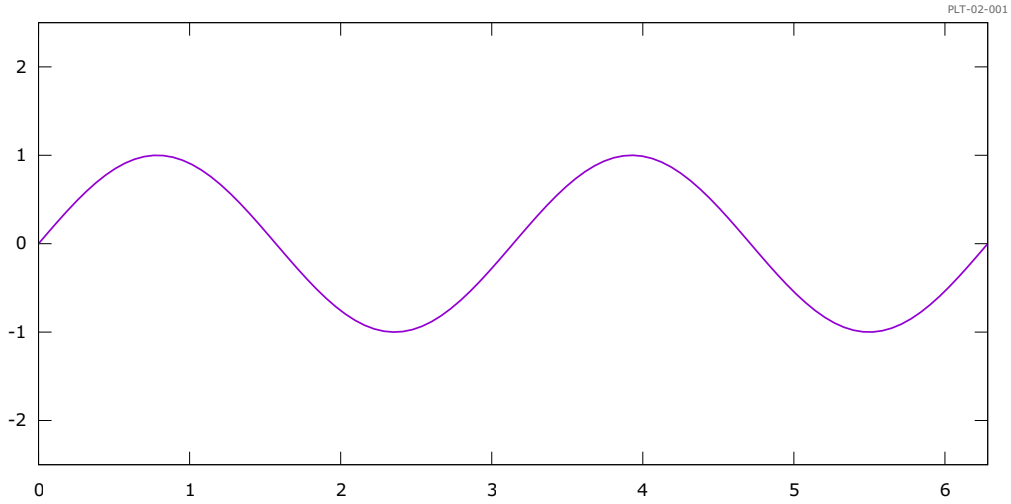
`condition ? 1/0 : value`	Return an undefined value where the curve must be omitted.
`abs(x)<0.2`	Exclude a visible interval around the singularity.
`1/0`	gnuplot treats this expression as undefined for plotting.

Use variables for repeated numbers

New baseline: the commands below define this example.

Start with ordinary numeric variables. A variable holds a value; no textual macro expansion is needed.

```
A = 1.0
w = 2.0
set xrange [0:2*pi]
set yrange [-2.5:2.5]
plot A*sin(w*x) with lines linewidth 2 notitle
```



PLT-02-001 · Generated from the complete Bash block

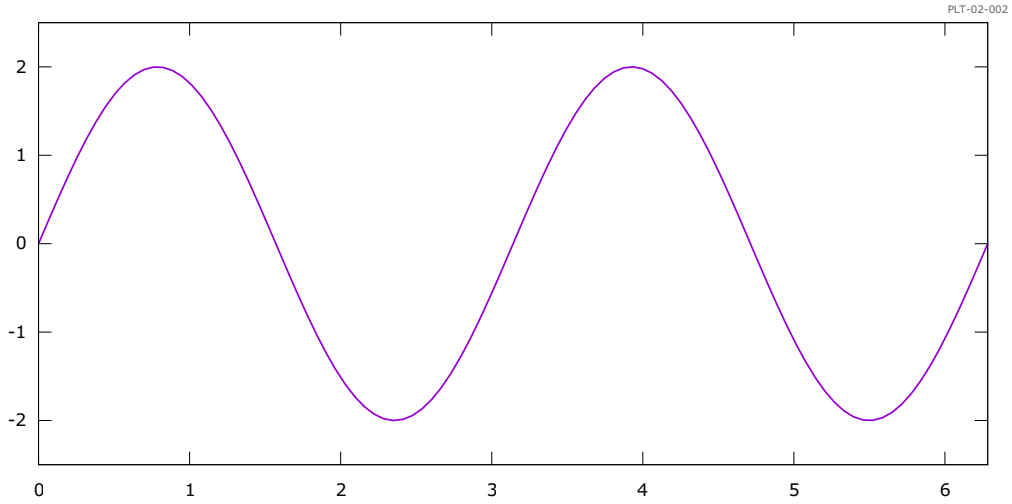
`A`	Amplitude: scales the vertical height of the sine wave.
`w`	Angular frequency: w=2 produces two cycles over 0 to 2*pi.
`A*sin(w*x)`	Read the variables when the expression is evaluated.

Change one variable

Compare with PLT-02-001 · Use variables for repeated numbers.

Doubling amplitude changes only the vertical scale of the function.

```
A = 2.0
w = 2.0
set xrange [0:2*pi]
set yrange [-2.5:2.5]
plot A*sin(w*x) with lines linewidth 2 notitle
```



PLT-02-002 · Generated from the complete Bash block

`A = 2.0`

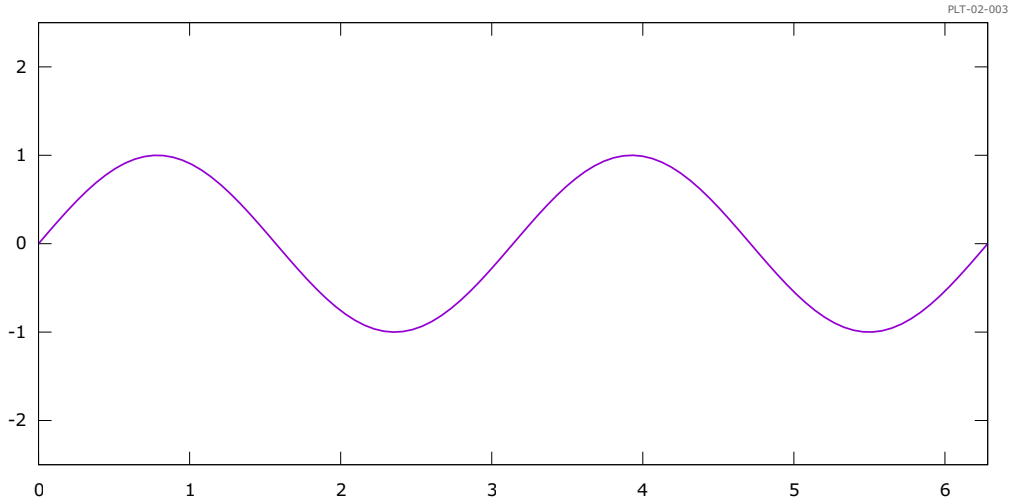
Peaks become +2 and -2; the displayed axis range stays fixed.

Name the expression as a function

Compare with PLT-02-001 · Use variables for repeated numbers.

The result is intentionally unchanged. A function packages numeric calculation, while a macro will package command text.

```
A = 1.0
w = 2.0
set xrange [0:2*pi]
set yrange [-2.5:2.5]
f(x) = A*sin(w*x)
plot f(x) with lines linewidth 2 notitle
```



PLT-02-003 · Generated from the complete Bash block

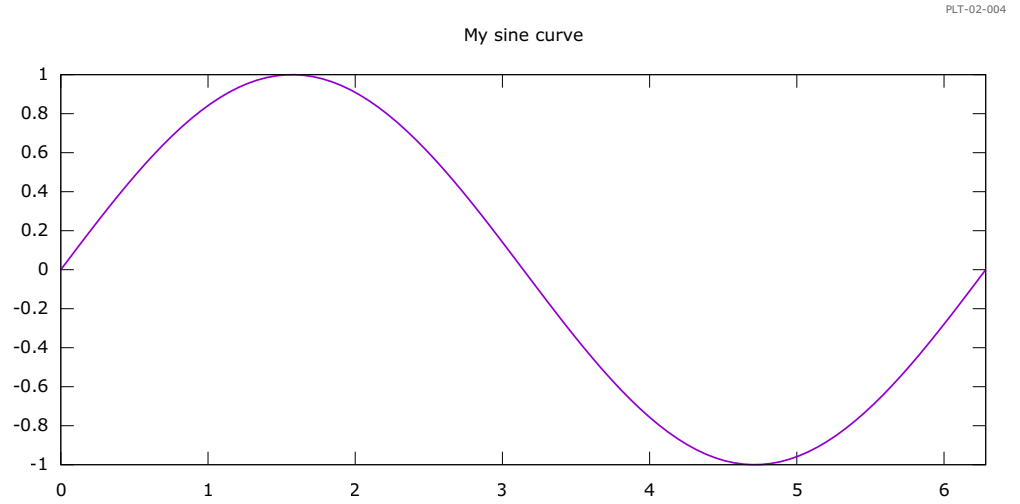
`f(x) = ...`	Define a numeric function with one argument; it can also read A and w.

Use a string variable directly

New baseline: the commands below define this example.

Many commands accept string expressions already. Titles are a simple example.

```
TITLE = "My sine curve"
set title TITLE
set xrange [0:2*pi]
plot sin(x) with lines linewidth 2 notitle
```



PLT-02-004 · Generated from the complete Bash block

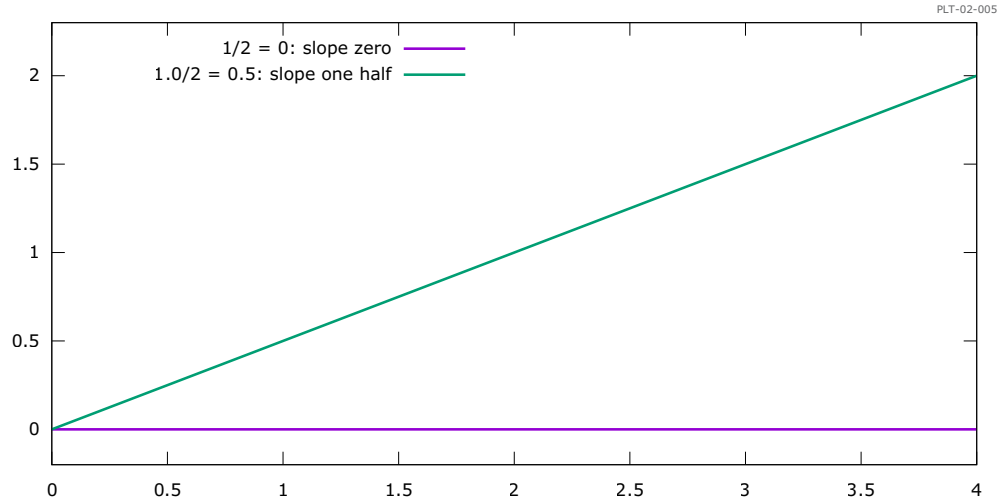
`TITLE = "..."`	Assign text to a string variable; variable names are case-sensitive.
`set title TITLE`	Pass the string value directly to the title command. No @ is necessary.

Avoid accidental integer division

New baseline: the commands below define this example.

When both operands are integers, division truncates its result before later calculations. At least one real operand gives real division. Multiplying an already truncated result by x cannot recover the fraction.

```
integer_ratio = 1/2
real_ratio = 1.0/2
set xrange [0:4]
set yrange [-0.2:2.3]
set key top left
plot integer_ratio*x with lines lw 3 \
     title sprintf("1/2 = %d: slope zero",integer_ratio), \
     real_ratio*x with lines lw 3 \
     title sprintf("1.0/2 = %.1f: slope one half",real_ratio)
```



PLT-02-005 · Generated from the complete Bash block

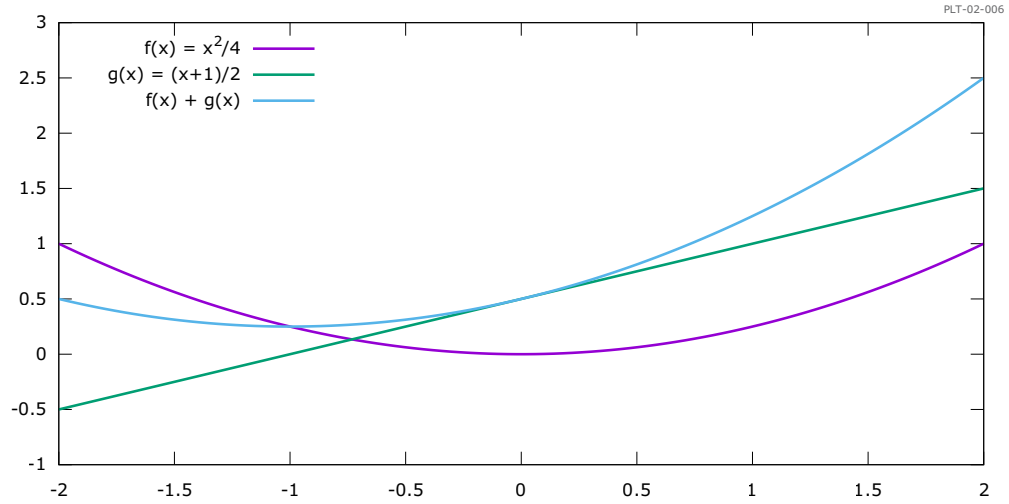
`1/2`	Both operands are integers, so the result is zero.
`1.0/2`	A real operand makes the result 0.5.
`real_ratio*x`	Use the calculated fraction as the slope of a line.
`%d; %.1f`	Print the integer result and the real result with one decimal digit.

Calculate and combine functions

New baseline: the commands below define this example.

gnuplot evaluates arithmetic expressions directly. Define two functions, then add their values at each x position. Parentheses control evaluation order; ** means exponentiation, not multiplication.

```
f(x) = x**2/4.0
g(x) = (x+1)*0.5
sum_curve(x) = f(x)+g(x)
set xrange [-2:2]
set yrange [-1:3]
set key top left
plot f(x) with lines lw 3 title "f(x) = x^2/4", \
     g(x) with lines lw 3 title "g(x) = (x+1)/2", \
     sum_curve(x) with lines lw 3 title "f(x) + g(x)"
```



PLT-02-006 · Generated from the complete Bash block

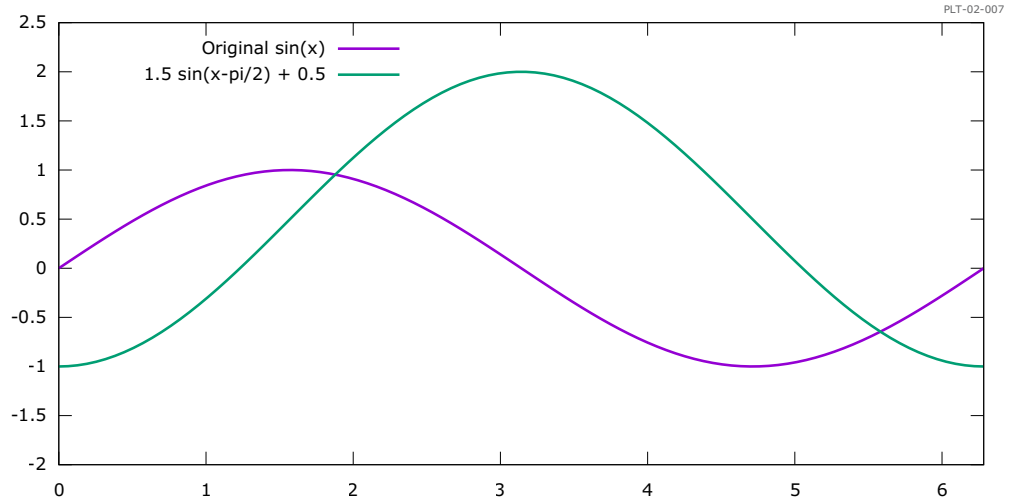
`x**2`	Square x; exponentiation uses **.
`+; -; *; /`	Add, subtract, multiply and divide numeric expressions.
`(x+1)*0.5`	Add one first, then multiply the entire result by one half.
`sum_curve(x)`	A user function can call other user functions.

Scale and shift a function with arithmetic

New baseline: the commands below define this example.

Transform a curve by calculating new arguments and values. Subtracting a positive shift inside f moves it right; multiplying its result changes its amplitude; adding an offset moves it upward.

```
f(x) = sin(x)
amplitude = 1.5
shift = pi/2
offset = 0.5
transformed(x) = amplitude*f(x-shift)+offset
set xrange [0:2*pi]
set yrange [-2:2.5]
set key top left
plot f(x) with lines lw 3 title "Original sin(x)", \
     transformed(x) with lines lw 3 \
     title "1.5 sin(x-pi/2) + 0.5"
```



PLT-02-007 · Generated from the complete Bash block

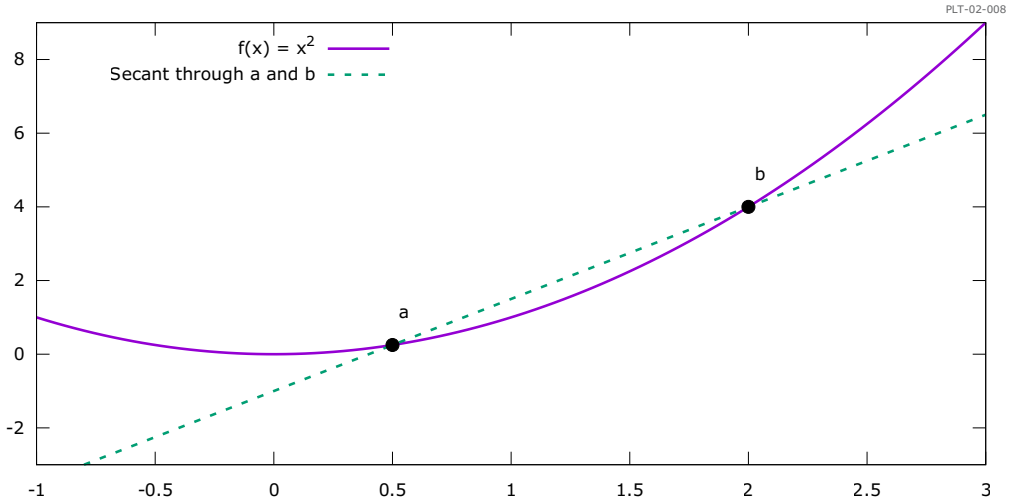
`f(x-shift)`	Shift the curve right by pi/2 radians.
`amplitude*f(...)`	Scale the vertical variation by 1.5.
`+offset`	Add 0.5 to every plotted y value.
`pi`	Built-in constant; trigonometric arguments are radians by default.

Calculate the secant through two points

New baseline: the commands below define this example.

A secant joins two points on the function. Its slope is the change in y divided by the change in x. The point-slope equation builds the entire line from that slope and the first point.

```
f(x) = x**2
a = 0.5
b = 2.0
slope = (f(b)-f(a))/(b-a)
secant(x) = f(a)+slope*(x-a)
set xrange [-1:3]
set yrange [-3:9]
set key top left
set label 1 "a" at a,f(a) offset 0,1 point pt 7 front
set label 2 "b" at b,f(b) offset 0,1 point pt 7 front
plot f(x) with lines lw 3 title "f(x) = x^2", \
     secant(x) with lines lw 3 dt 2 title "Secant through a and b"
```



PLT-02-008 · Generated from the complete Bash block

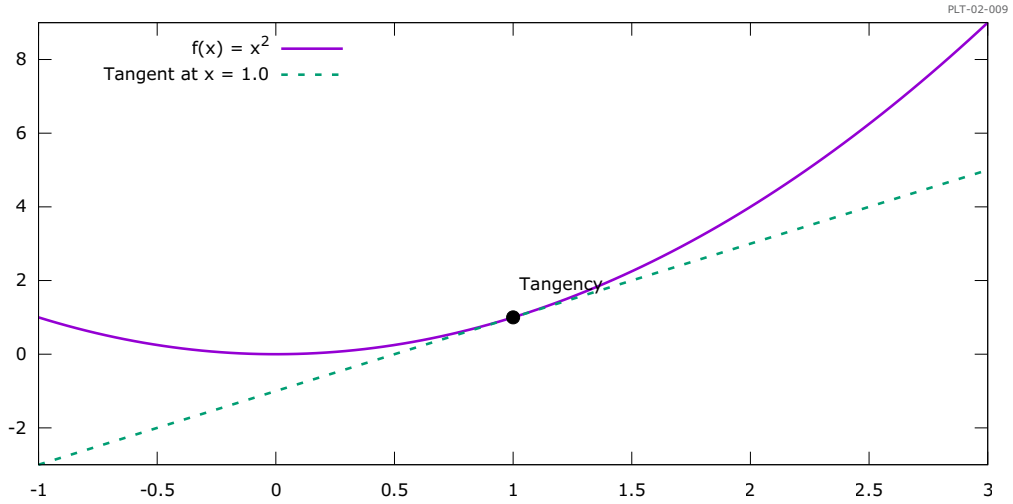
`a; b`	Choose distinct x positions; b-a must not be zero.
`(f(b)-f(a))/(b-a)`	Calculate the average slope between the two points.
`f(a)+slope*(x-a)`	A line with the calculated slope passing through (a,f(a)).
`label ... point`	Mark both points using their calculated y coordinates.

Calculate a tangent from the known derivative

Compare with PLT-02-008 · Calculate the secant through two points.

The tangent uses the local derivative instead of a secant slope. Supply the derivative explicitly, then calculate t(x)=f(a)+f'(a)*(x-a). For x squared at a=1, the result is 2*x-1.

```
f(x) = x**2
df(x) = 2*x
a = 1.0
slope = df(a)
tangent(x) = f(a)+slope*(x-a)
set xrange [-1:3]
set yrange [-3:9]
set key top left
set label 1 "Tangency" at a,f(a) offset 0,1 point pt 7 front
plot f(x) with lines lw 3 title "f(x) = x^2", \
     tangent(x) with lines lw 3 dt 2 \
     title sprintf("Tangent at x = %.1f",a)
```



PLT-02-009 · Generated from the complete Bash block

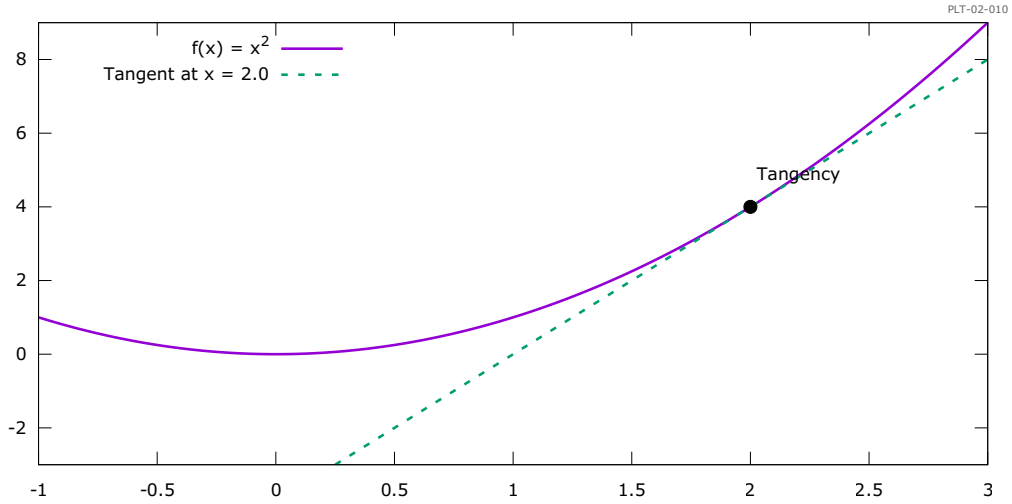
`df(x) = 2*x`	Define the known derivative of x squared; its name has no special meaning to gnuplot.
`slope = df(a)`	Evaluate the derivative at the point of tangency.
`tangent(x)`	Use point-slope form to calculate the tangent line.
`a = 1.0`	The contact point is (1,1) and its slope is two.

Move the tangent by changing one variable

Compare with PLT-02-009 · Calculate a tangent from the known derivative.

Change only a and recalculate the slope. At a=2, the contact point becomes (2,4), the slope becomes four, and the tangent becomes 4*x-4. Assigned numeric variables are evaluated when their assignment executes.

```
f(x) = x**2
df(x) = 2*x
a = 2.0
slope = df(a)
tangent(x) = f(a)+slope*(x-a)
set xrange [-1:3]
set yrange [-3:9]
set key top left
set label 1 "Tangency" at a,f(a) offset 0,1 point pt 7 front
plot f(x) with lines lw 3 title "f(x) = x^2", \
     tangent(x) with lines lw 3 dt 2 \
     title sprintf("Tangent at x = %.1f",a)
```



PLT-02-010 · Generated from the complete Bash block

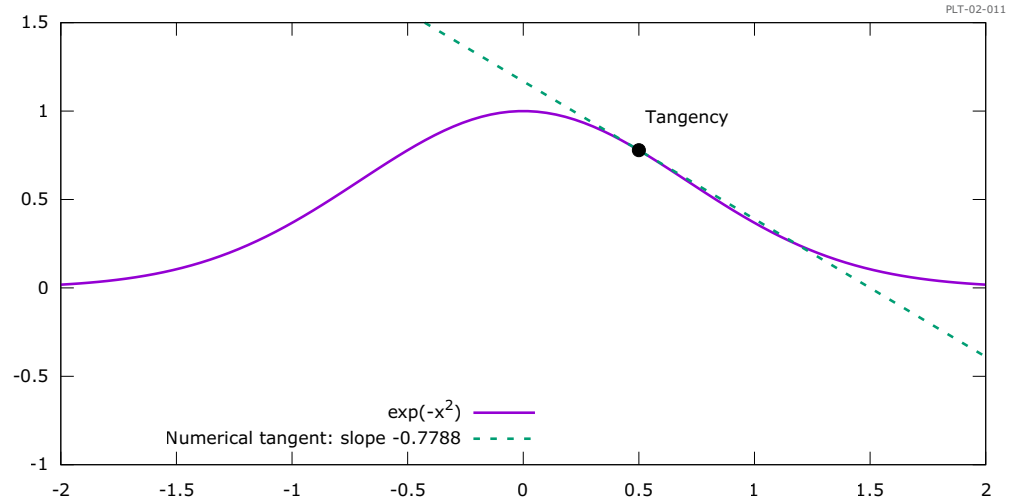
`a = 2.0`	Move the point of tangency to x=2.
`slope = df(a)`	Re-evaluate this assignment after changing a.
`tangent(x)`	A user function evaluates its expression using the current variable values.

Estimate a tangent with a numerical derivative

New baseline: the commands below define this example.

For a smooth function, a centered difference approximates the local slope using two nearby evaluations. This calculates an estimate rather than symbolically differentiating f. The step h must be nonzero and suitable for the function scale.

```
f(x) = exp(-x**2)
h = 1e-5
df(x) = (f(x+h)-f(x-h))/(2*h)
a = 0.5
slope = df(a)
tangent(x) = f(a)+slope*(x-a)
set xrange [-2:2]
set yrange [-1:1.5]
set key bottom left
set label 1 "Tangency" at a,f(a) offset 0,1 point pt 7 front
plot f(x) with lines lw 3 title "exp(-x^2)", \
     tangent(x) with lines lw 3 dt 2 \
     title sprintf("Numerical tangent: slope %.4f",slope)
```



PLT-02-011 · Generated from the complete Bash block

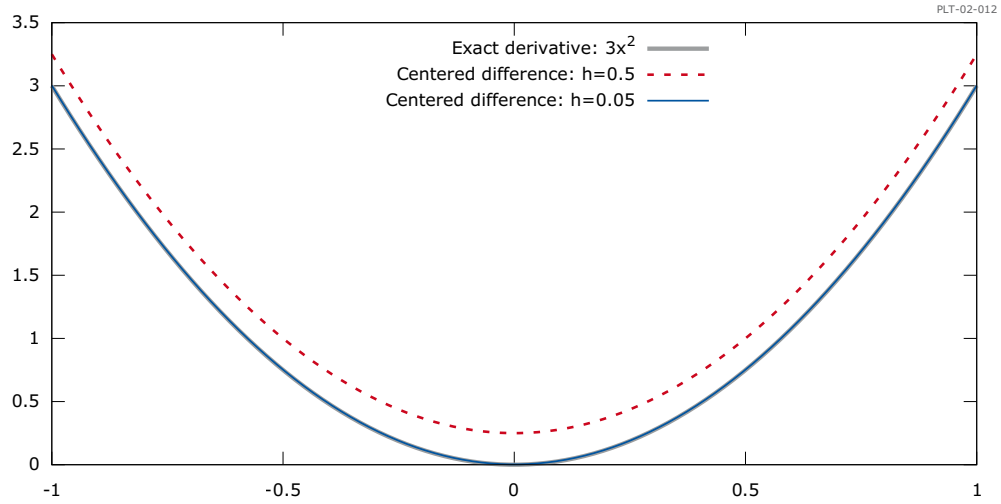
`exp(-x**2)`	Evaluate the exponential function at minus x squared.
`1e-5`	Scientific notation for 0.00001.
`(f(x+h)-f(x-h))/(2*h)`	Centered finite difference: change in function value divided by the full interval width.
`h`	Controls the numerical approximation; this fixed step is an example, not a universal choice.
`differentiability`	A finite difference does not guarantee a tangent exists at a cusp or discontinuity.

Compare numerical derivative step sizes

Compare with PLT-02-011 · Estimate a tangent with a numerical derivative.

For f(x)=x cubed, the centered difference equals 3*x squared plus h squared. A large step therefore produces a visible upward error. A smaller step reduces that error here; extremely small steps can instead lose accuracy through floating-point cancellation.

```
f(x) = x**3
exact(x) = 3*x**2
estimate(x,h) = (f(x+h)-f(x-h))/(2*h)
set xrange [-1:1]
set yrange [0:3.5]
set key top center
plot exact(x) with lines lw 5 lc rgb "#9C9E9F" \
     title "Exact derivative: 3x^2", \
     estimate(x,0.5) with lines lw 3 dt 2 lc rgb "#CC071E" \
     title "Centered difference: h=0.5", \
     estimate(x,0.05) with lines lw 2 lc rgb "#00549F" \
     title "Centered difference: h=0.05"
```



PLT-02-012 · Generated from the complete Bash block

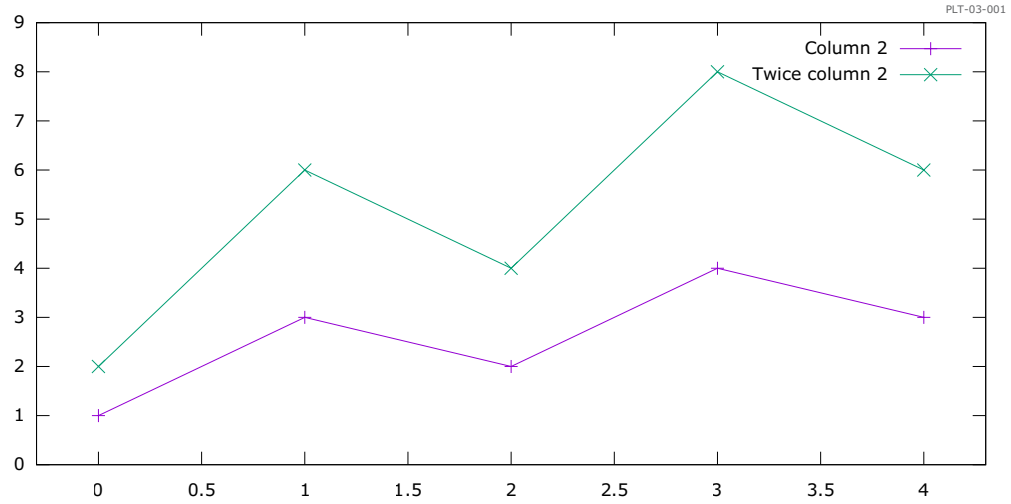
`estimate(x,h)`	A function with two parameters makes the step size explicit.
`h=0.5`	For this cubic, the derivative estimate is too high by 0.25.
`h=0.05`	Reduce that error to 0.0025; the blue and exact curves nearly coincide.
`truncation versus rounding`	A smaller step reduces approximation error until floating-point effects become significant.

Read columns and calculate plotted values

New baseline: the commands below define this example.

Using selects columns for the plot style. Parenthesized expressions can transform a column before plotting. This inline datablock behaves like a whitespace-separated data file.

```
$D << EOD
0 1
1 3
2 2
3 4
4 3
EOD
set xrange [-0.3:4.3]
set yrange [0:9]
plot $D using 1:2 with linespoints title "Column 2", \
    $D using 1:($2*2) with linespoints title "Twice column 2"
```



PLT-03-001 · Generated from the complete Bash block

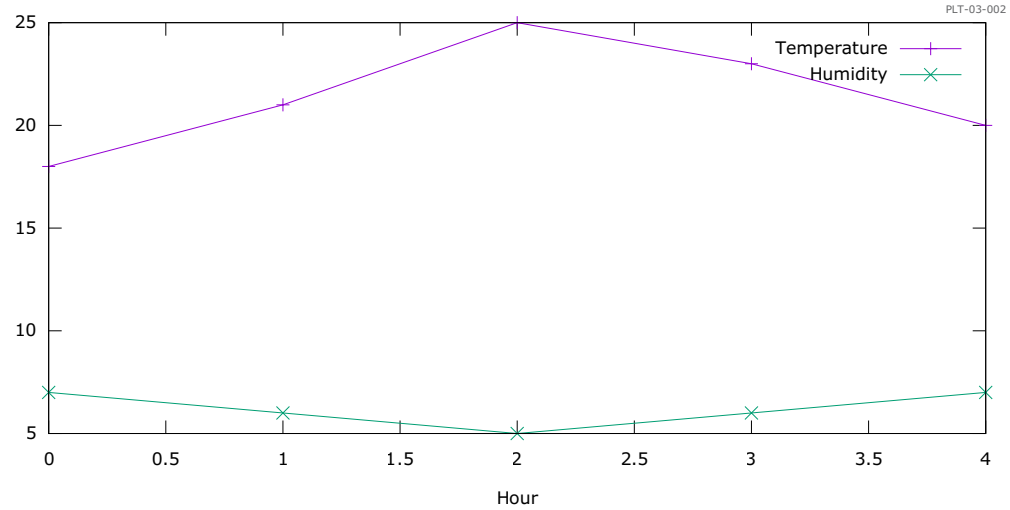
`using 1:2`	Read x from column 1 and y from column 2.
`using 1:($2*2)`	Compute y from a column expression.
`$D`	Named inline data; a quoted filename can be used in its place.
`$2`	Column 2 of the current data row inside a using expression.

Read CSV headers and use them in the legend

New baseline: the commands below define this example.

Set the separator before reading CSV. Column headers name the quantities and can automatically supply legend entries. The data are embedded here so the example stays self-contained.

```
$CSV << EOD
"Hour","Temperature","Humidity"
0,18,7
1,21,6
2,25,5
3,23,6
4,20,7
EOD
set datafile separator comma
set datafile columnheaders
set key autotitle columnhead
set xlabel "Hour"
plot $CSV using "Hour":"Temperature" with linespoints, \
    $CSV using "Hour":"Humidity" with linespoints
```



PLT-03-002 · Generated from the complete Bash block

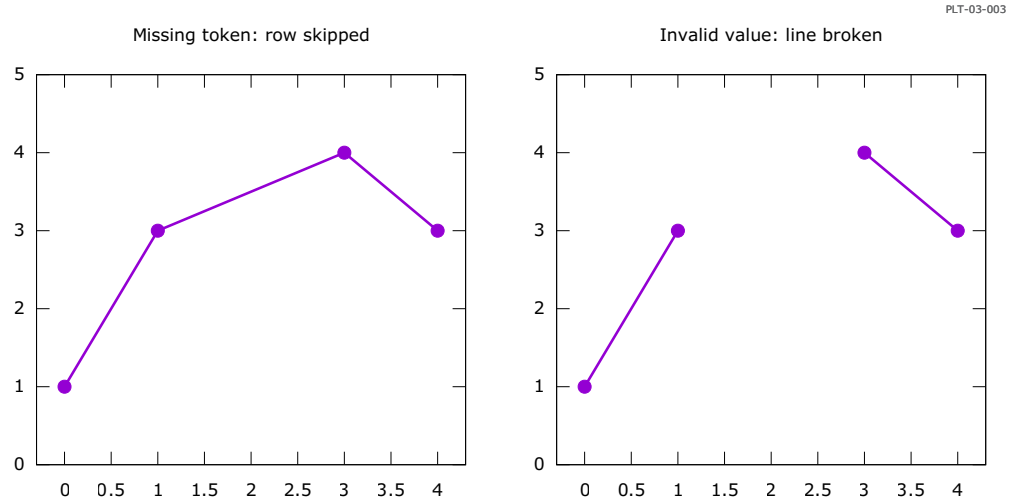
`separator comma`	Split fields at commas rather than whitespace.
`datafile columnheaders`	Treat the first row as column names.
`using "Hour":"Temperature"`	Select columns by their header names.
`autotitle columnhead`	Use the y-column header as the legend title.

Compare missing values with invalid values

Compare with PLT-03-001 · Read columns and calculate plotted values.

gnuplot distinguishes missing data from invalid data. A configured missing token skips the row and connects the surrounding observations. An invalid value breaks the line. Use explicit gaps when continuity would be misleading.

```
$D << EOD
0 1
1 3
2 NaN
3 4
4 3
EOD
set multiplot layout 1,2
set xrange [-0.3:4.3]
set yrange [0:5]
set datafile missing "NaN"
set title "Missing token: row skipped"
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
unset datafile missing
set title "Invalid value: line broken"
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
unset multiplot
```



PLT-03-003 · Generated from the complete Bash block

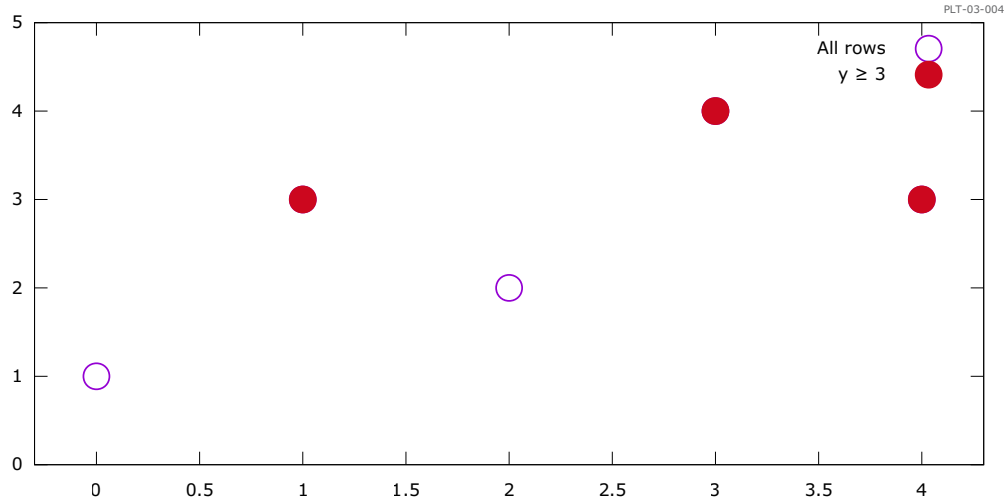
`datafile missing "NaN"`	Skip rows containing this token; connect across the missing observation.
`unset datafile missing`	Treat NaN as invalid again, retaining the row as a break in the line.
`blank line`	A blank record can also separate connected data segments.

Filter observations with a column expression

Compare with PLT-03-001 · Read columns and calculate plotted values.

The ternary expression keeps rows that satisfy a condition and returns an undefined value for the others. Points show the retained observations without implying continuity.

```
$D << EOD
0 1
1 3
2 2
3 4
4 3
EOD
set xrange [-0.3:4.3]
set yrange [0:5]
plot $D using 1:2 with points pt 6 ps 2 title "All rows", \
    $D using 1:($2>=3 ? $2 : 1/0) with points \
    pt 7 ps 2 lc rgb "#CC071E" title "y ≥ 3"
```



PLT-03-004 · Generated from the complete Bash block

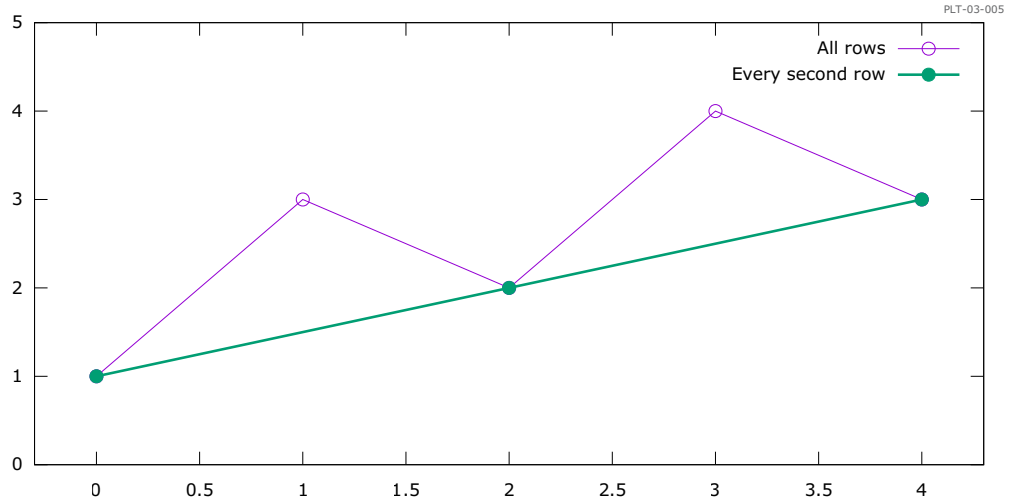
`$2>=3 ? $2 : 1/0`	Keep the value only if column 2 is at least three.
`1/0`	Skip a row by making its plotted value undefined.
`points`	Show the selected observations without connecting gaps.

Select every second data row

Compare with PLT-03-001 · Read columns and calculate plotted values.

Every selects input records before plotting. It reduces the data used for both the line and its symbols, unlike pointinterval which only thins symbols.

```
$D << EOD
0 1
1 3
2 2
3 4
4 3
EOD
set xrange [-0.3:4.3]
set yrange [0:5]
plot $D using 1:2 with linespoints pt 6 title "All rows", \
    $D every 2 using 1:2 with linespoints pt 7 lw 3 \
    title "Every second row"
```



PLT-03-005 · Generated from the complete Bash block

`every 2`	Select rows 0, 2, 4, and so on.
`every versus pointinterval`	Every changes the input observations; pointinterval only changes displayed markers.

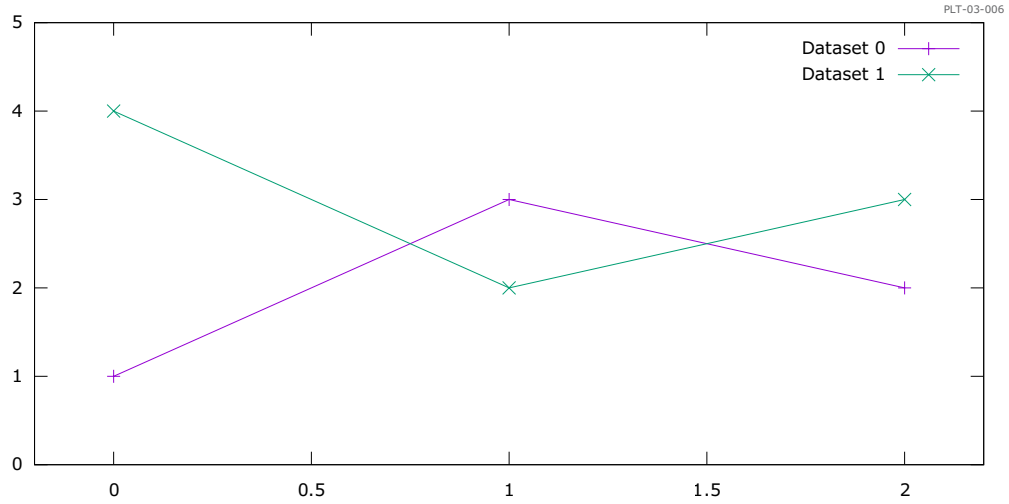
Select datasets separated by blank records

New baseline: the commands below define this example.

Two blank records separate indexed datasets. A single blank record breaks a line within one dataset. Index numbering begins at zero.

```
$D << EOD
0 1
1 3
2 2


0 4
1 2
2 3
EOD
set xrange [-0.2:2.2]
set yrange [0:5]
plot $D index 0 using 1:2 with linespoints title "Dataset 0", \
    $D index 1 using 1:2 with linespoints title "Dataset 1"
```



PLT-03-006 · Generated from the complete Bash block

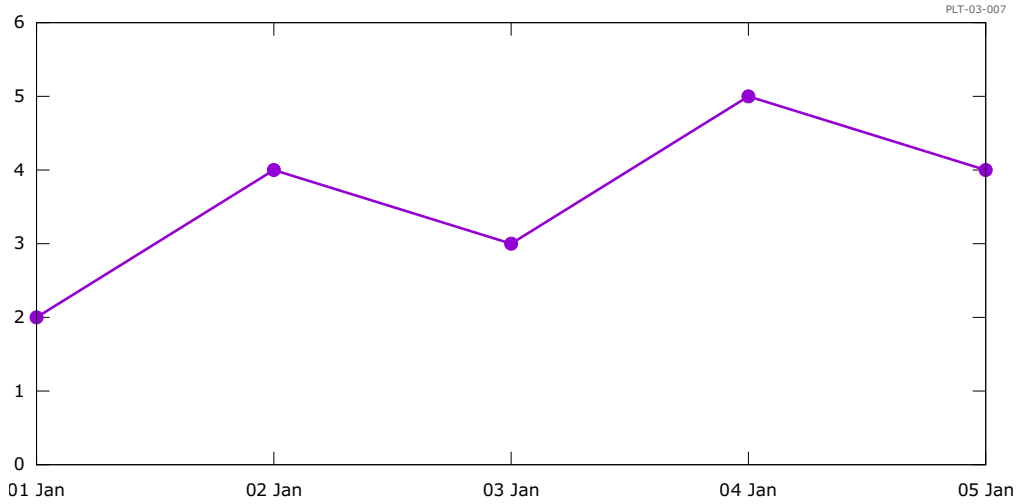
`index 0; index 1`	Select the first and second datasets.
`two blank records`	Start a new indexed dataset.
`one blank record`	Separate connected segments without starting a new index.

Read dates and format their tick labels

New baseline: the commands below define this example.

Input parsing and displayed tick labels use separate formats. gnuplot stores time coordinates as seconds; a tick interval of 86400 therefore represents one day.

```
$D << EOD
2026-01-01 2
2026-01-02 4
2026-01-03 3
2026-01-04 5
2026-01-05 4
EOD
set xdata time
set timefmt "%Y-%m-%d"
set format x "%d %b"
set xrange ["2026-01-01":"2026-01-05"]
set xtics 86400
set yrange [0:6]
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-03-007 · Generated from the complete Bash block

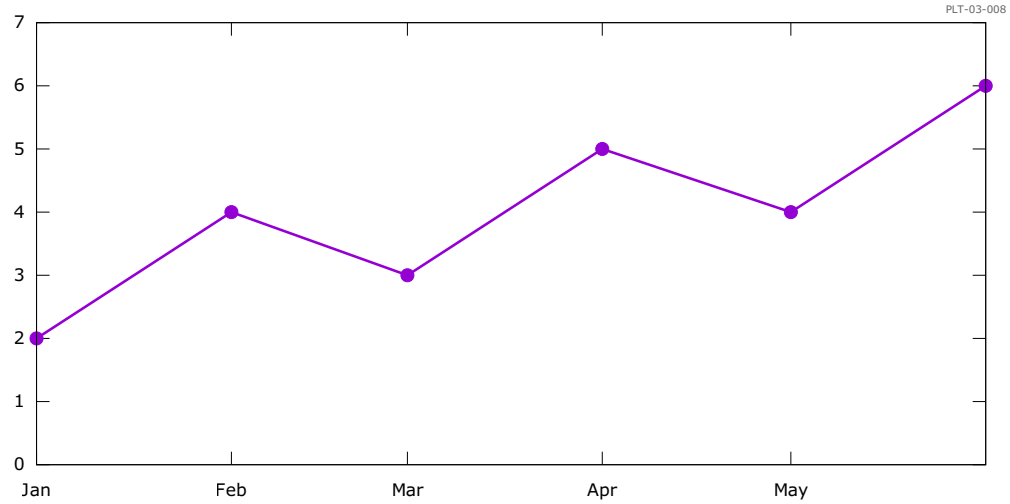
`xdata time`	Interpret x input and quoted range limits as dates/times.
`timefmt "%Y-%m-%d"`	Read the year-month-day syntax in the data.
`format x "%d %b"`	Display day and abbreviated month on the ticks.
`xtics 86400`	Use a one-day interval measured in seconds.

Place ticks at calendar-month intervals

Compare with PLT-03-007 · Read dates and format their tick labels.

Months have different lengths, so a fixed number of seconds is not a calendar month. The month interval syntax follows calendar boundaries.

```
$D << EOD
2026-01-01 2
2026-02-01 4
2026-03-01 3
2026-04-01 5
2026-05-01 4
2026-06-01 6
EOD
set xdata time
set timefmt "%Y-%m-%d"
set format x "%b"
set xrange ["2026-01-01":"2026-06-01"]
set xtics "2026-01-01", 1 month, "2026-06-01"
set yrange [0:7]
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-03-008 · Generated from the complete Bash block

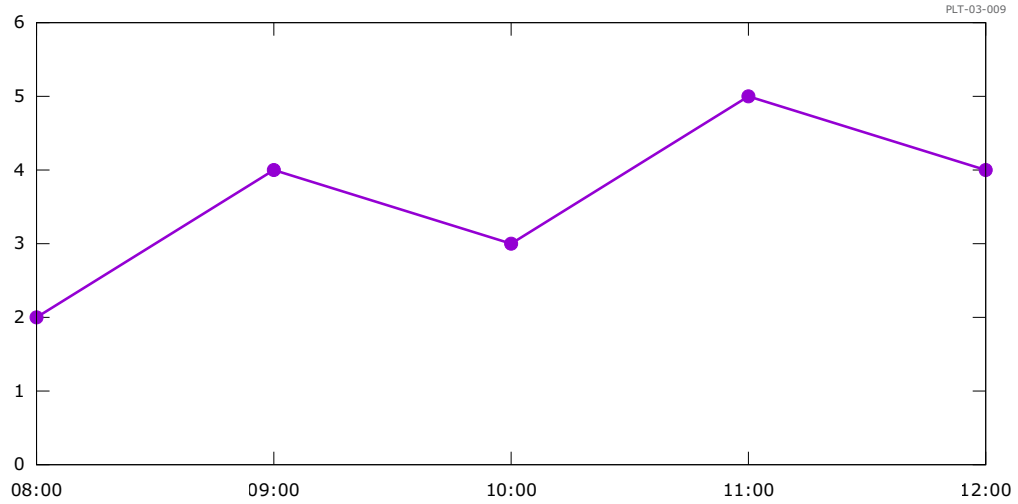
`1 month`	Advance by calendar months rather than a fixed second count.
`format x "%b"`	Print abbreviated month names.
`quoted start and end`	Parse tick boundaries using timefmt.

Read timestamps and show hourly ticks

Compare with PLT-03-007 · Read dates and format their tick labels.

A timestamp separated with T stays in one data field. gnuplot uses UTC-style time coordinates and does not apply local timezone or daylight-saving conversions.

```
$D << EOD
2026-01-01T08:00 2
2026-01-01T09:00 4
2026-01-01T10:00 3
2026-01-01T11:00 5
2026-01-01T12:00 4
EOD
set xdata time
set timefmt "%Y-%m-%dT%H:%M"
set format x "%H:%M"
set xtics 3600
set yrange [0:6]
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-03-009 · Generated from the complete Bash block

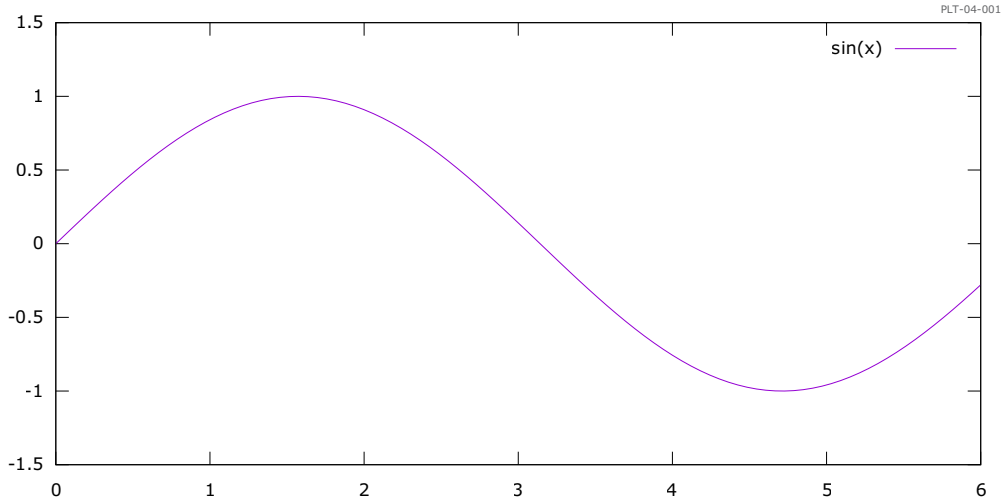
`T`	Literal separator matched by the input format.
`%H:%M`	Parse or display hours and minutes, according to the command.
`xtics 3600`	Place ticks one hour apart.
`timezone`	Normalize timestamps to the intended time standard before plotting.

Prepare a simple tick example

Compare with PLT-01-007 · Leave room above and below.

Use an interval from 0 to 6. Subsequent tick examples refer back to this unchanged curve and range.

```
set yrange [-1.5:1.5]
set xrange [0:6]
plot sin(x)
```



PLT-04-001 · Generated from the complete Bash block

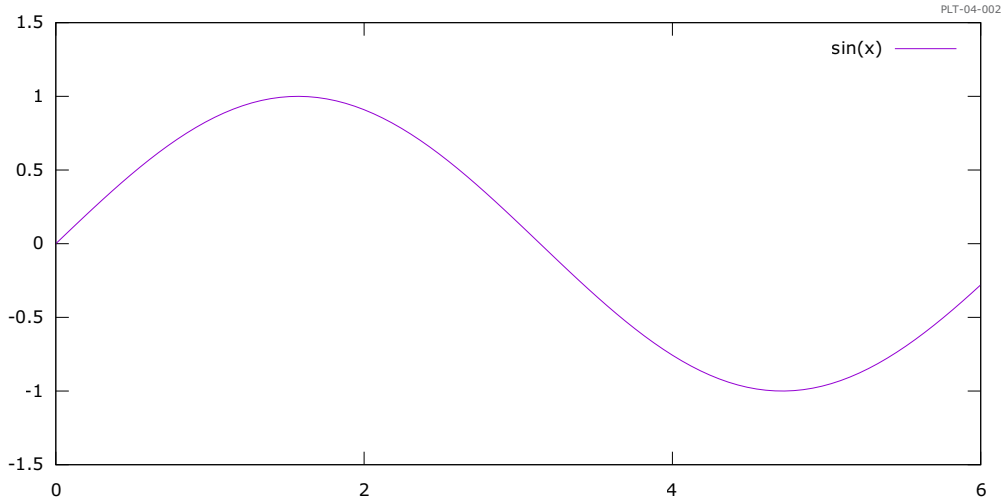
`[0:6]` Only the horizontal range changes. Ticks are still placed automatically.

Place a tick every two units

Compare with PLT-04-001 · Prepare a simple tick example.

Set an explicit interval between major x ticks.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 2
plot sin(x)
```



PLT-04-002 · Generated from the complete Bash block

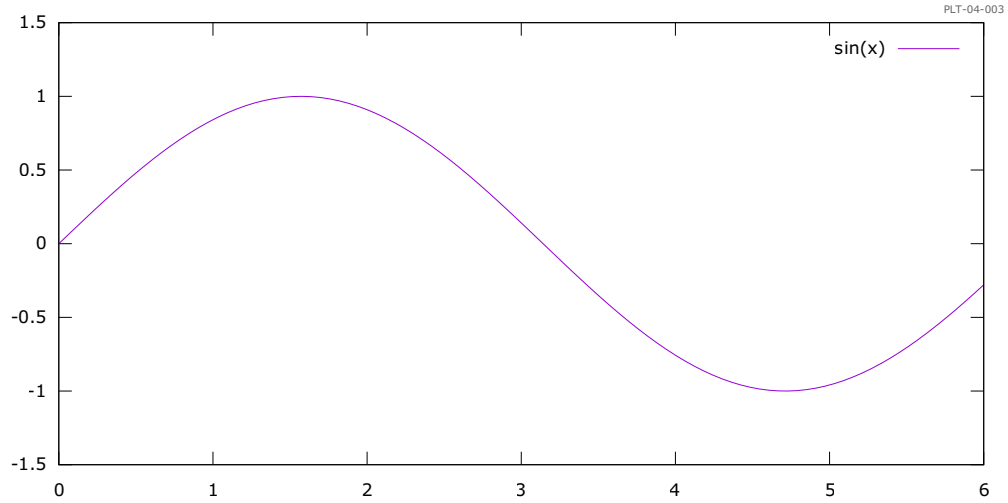
`xtics`	Configure major ticks on the lower x axis.
`2`	Tick increment in x data units.

Halve the tick interval

Compare with PLT-04-002 · Place a tick every two units.

Change only the increment: more tick marks now appear.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
plot sin(x)
```



PLT-04-003 · Generated from the complete Bash block

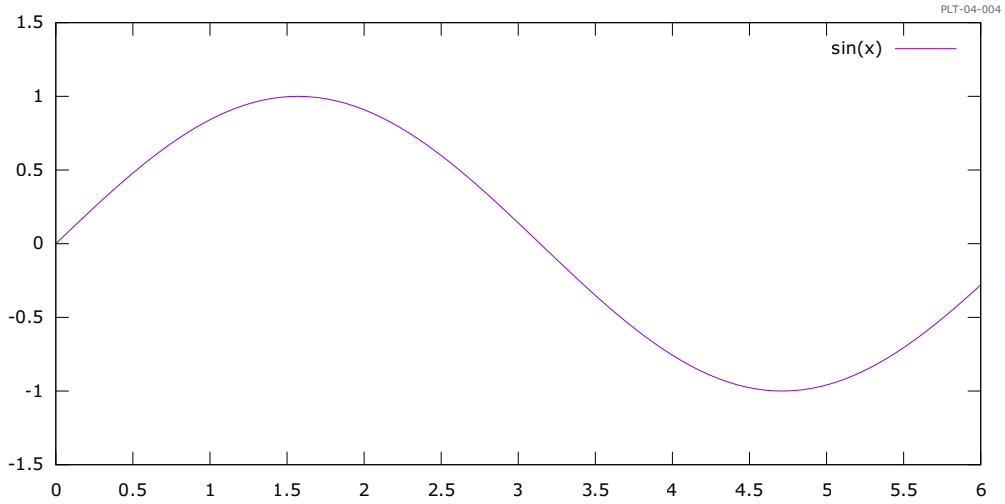
1 A tick every one x unit, rather than every two.

Use half-unit ticks

Compare with PLT-04-003 · Halve the tick interval.

A fractional increment gives a denser scale.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 0.5
plot sin(x)
```



PLT-04-004 · Generated from the complete Bash block

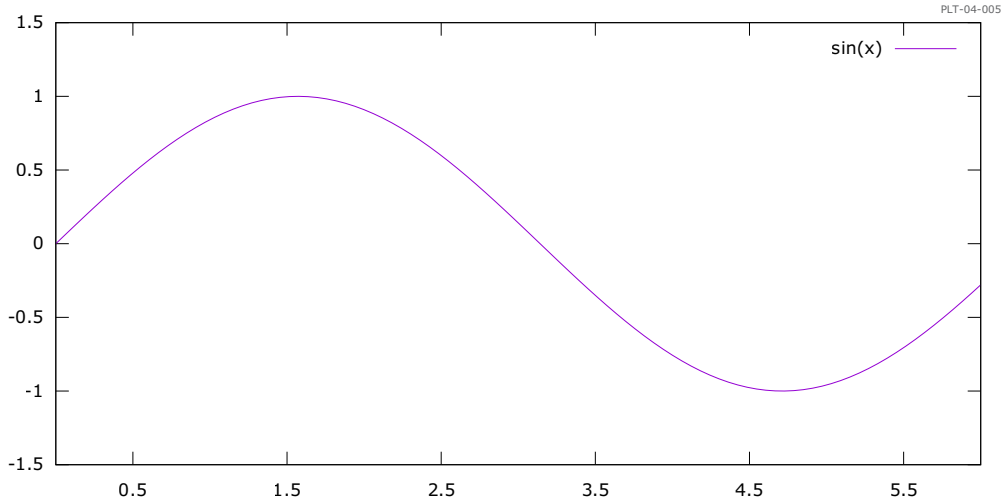
`0.5` A tick every half unit. Decimal values are allowed.

Set start, increment and end

Compare with PLT-04-003 · Halve the tick interval.

Three values restrict where the regularly spaced ticks begin and end.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 0.5,1,5.5
plot sin(x)
```



PLT-04-005 · Generated from the complete Bash block

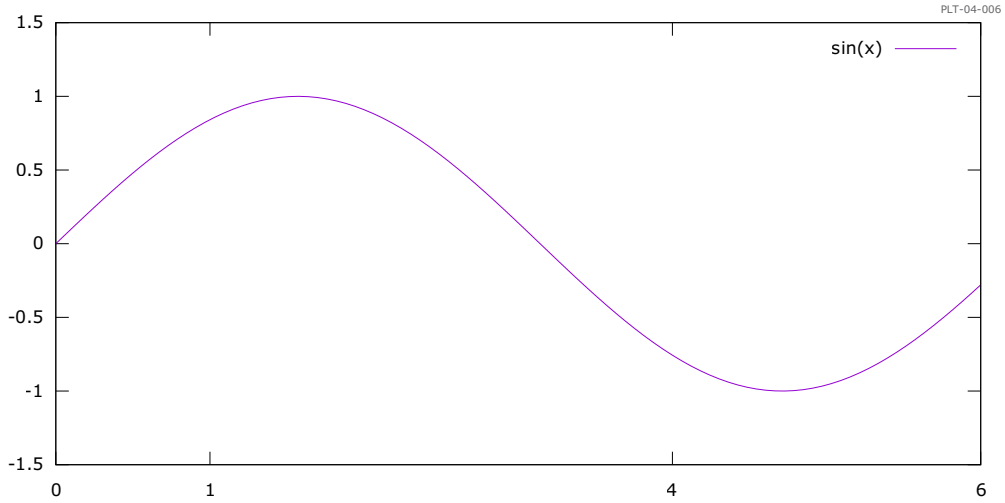
`0.5`	First tick position.
`1`	Increment between ticks.
`5.5`	Last permitted tick position. The axis range itself is unchanged.

Choose irregular tick positions

Compare with PLT-04-003 · Halve the tick interval.

A parenthesised list replaces the regular sequence.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics (0,1,4,6)
plot sin(x)
```



PLT-04-006 · Generated from the complete Bash block

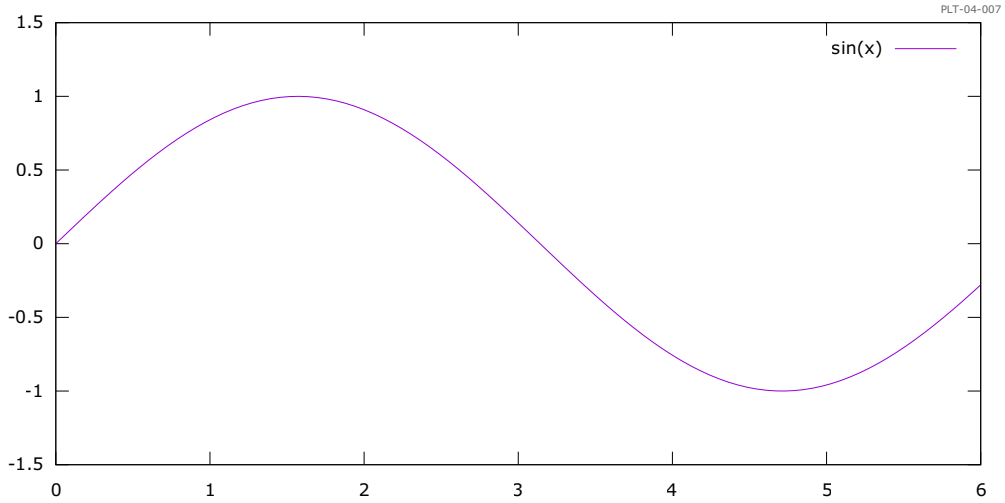
`(0,1,4,6)` Explicit x positions; they need not be evenly spaced.

Change y ticks independently

Compare with PLT-04-003 · Halve the tick interval.

Keep the x ticks and change only the vertical increment.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set ytics 0.5
plot sin(x)
```



PLT-04-007 · Generated from the complete Bash block

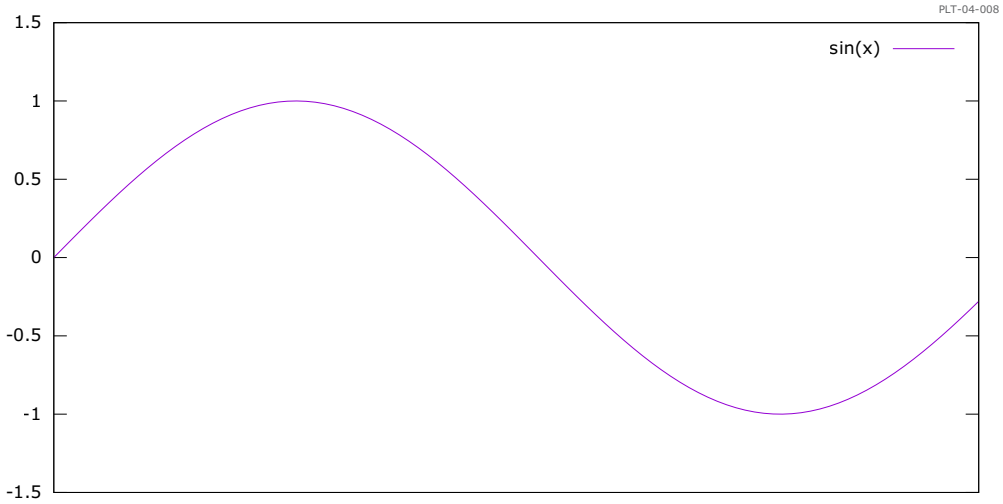
`ytics`	Configure ticks on the primary y axis.
`0.5`	Vertical tick increment, in y data units.

Remove x ticks

Compare with PLT-04-003 · Halve the tick interval.

The curve, x axis and y ticks remain visible.

```
set yrange [-1.5:1.5]
set xrange [0:6]
unset xtics
plot sin(x)
```



PLT-04-008 · Generated from the complete Bash block

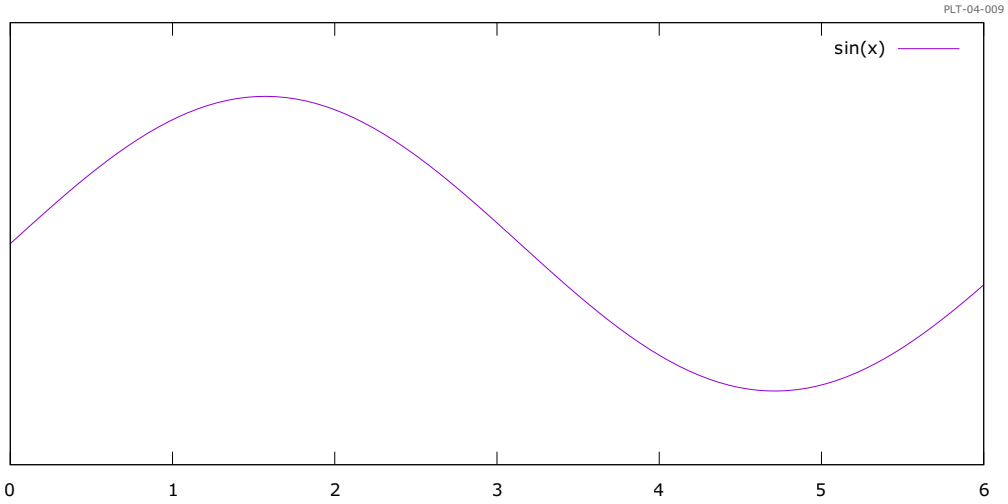
`unset xtics`	Remove both the x tick marks and their labels.

Remove y ticks

Compare with PLT-04-003 · Halve the tick interval.

Hide the vertical scale while retaining the horizontal one.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
unset ytics
plot sin(x)
```



PLT-04-009 · Generated from the complete Bash block

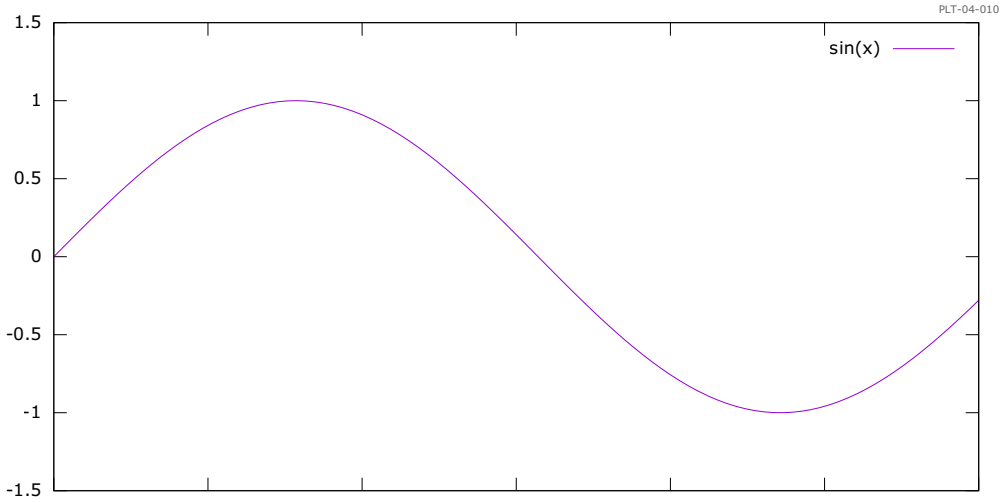
`unset ytics`	Remove the primary y tick marks and labels.

Keep marks but hide their labels

Compare with PLT-04-003 · Halve the tick interval.

An empty format string removes the text while leaving the tick marks.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set format x ""
plot sin(x)
```



PLT-04-010 · Generated from the complete Bash block

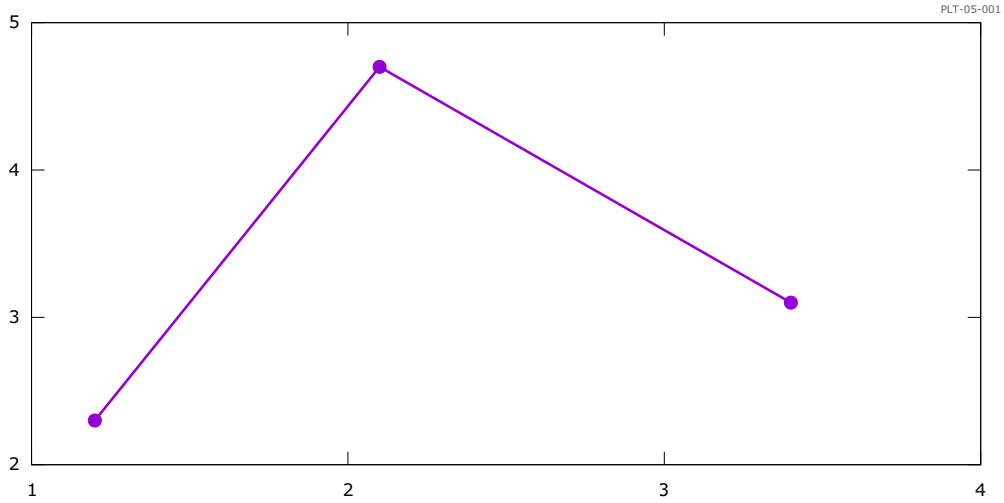
`format x`	Control how x tick values are printed.
`""`	An empty string: print no label text.

Observe automatic range rounding

New baseline: the commands below define this example.

With data input, autoscaling finds the extrema and normally extends the range to convenient major ticks. This leaves some space between the extreme points and the frame.

```
$D << EOD
1.2 2.3
2.1 4.7
3.4 3.1
EOD
set autoscale xy
set xtics 1
set ytics 1
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-05-001 · Generated from the complete Bash block

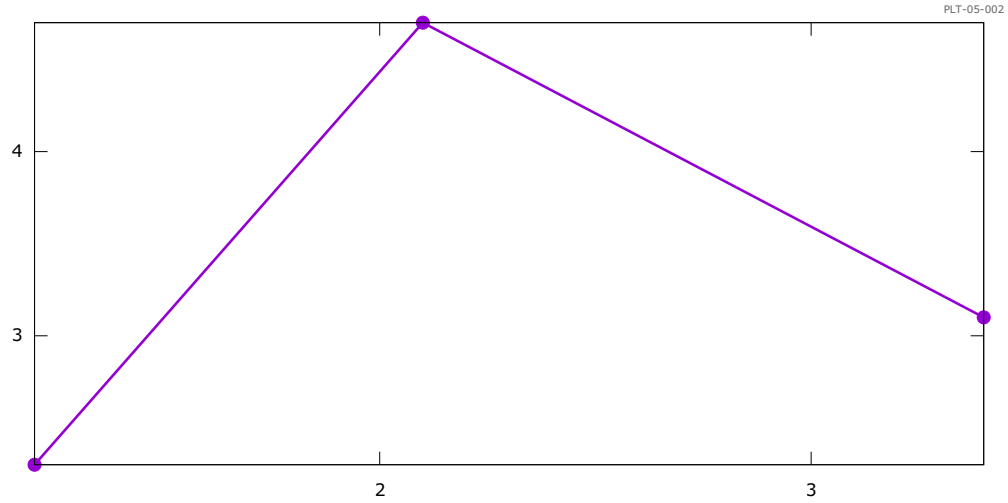
`autoscale xy`	Choose x and y ranges from the plotted data.
`xtics 1; ytics 1`	Major tick positions guide the default extension of the ranges.

Keep automatic ranges at the exact extrema

Compare with PLT-05-001 · Observe automatic range rounding.

The fix modifier disables extension to the next major tick. Large point symbols may touch the border because the coordinates now end exactly at the data extrema.

```
$D << EOD
1.2 2.3
2.1 4.7
3.4 3.1
EOD
set autoscale xfix
set autoscale yfix
set xtics 1
set ytics 1
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-05-002 · Generated from the complete Bash block

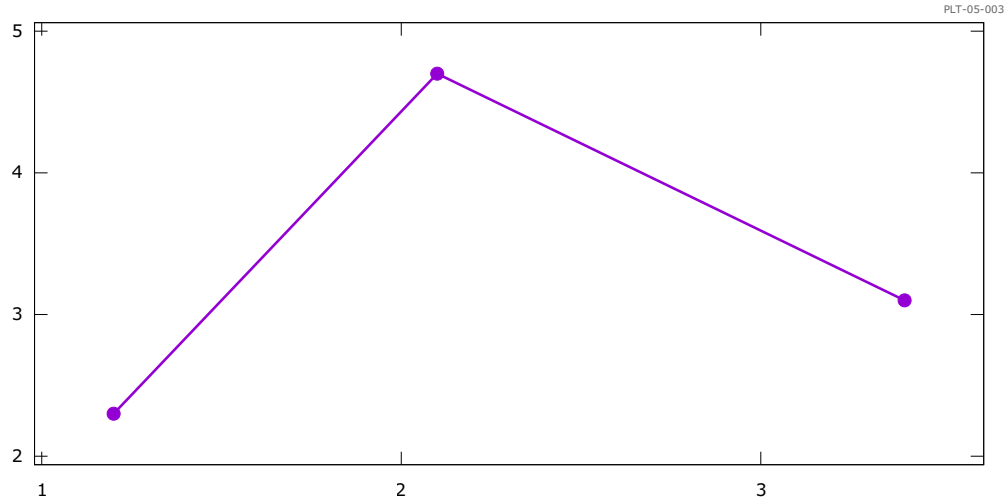
`autoscale xfix; autoscale yfix`	Autoscale both axes without rounding outward to major ticks.
`fix`	Controls extension of an automatic range; it does not make the range manually fixed.

Pad the data range with offsets

Compare with PLT-05-002 · Keep automatic ranges at the exact extrema.

Offsets add room inside the coordinate frame. Margins instead reserve space outside the frame for labels and other decorations.

```
$D << EOD
1.2 2.3
2.1 4.7
3.4 3.1
EOD
set autoscale xfix
set autoscale yfix
set xtics 1
set ytics 1
set offsets graph 0.1, graph 0.1, graph 0.15, graph 0.15
plot $D using 1:2 with linespoints pt 7 lw 3 notitle
```



PLT-05-003 · Generated from the complete Bash block

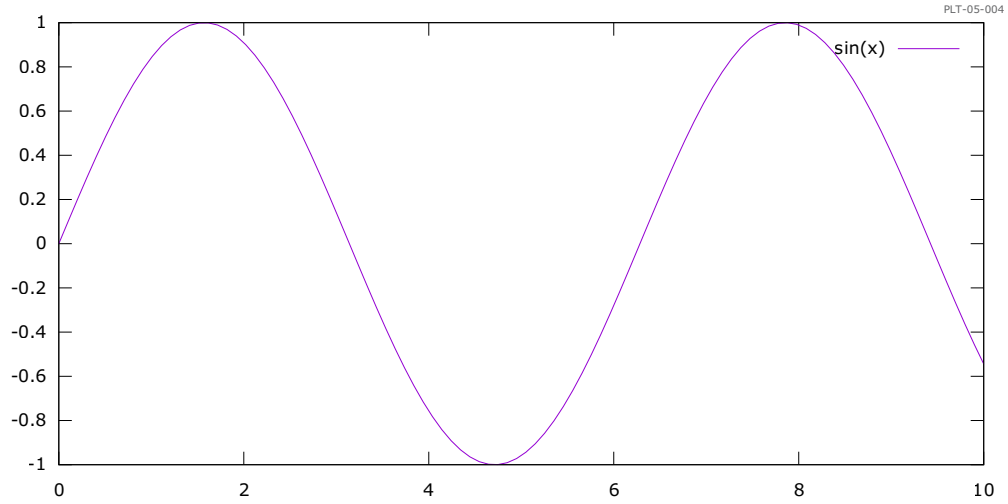
`offsets left,right,top,bottom`	Order of the four extensions around the data range.
`graph 0.1`	Extend by a fraction of the unpadded axis span.
`offsets versus margins`	Offsets enlarge the coordinate range; margins position the frame on the canvas.

Let the upper x limit be automatic

Compare with PLT-01-006 · Choose the horizontal interval.

A star leaves one bound to autoscaling.

```
set xrange [0:*]
plot sin(x)
```



PLT-05-004 · Generated from the complete Bash block

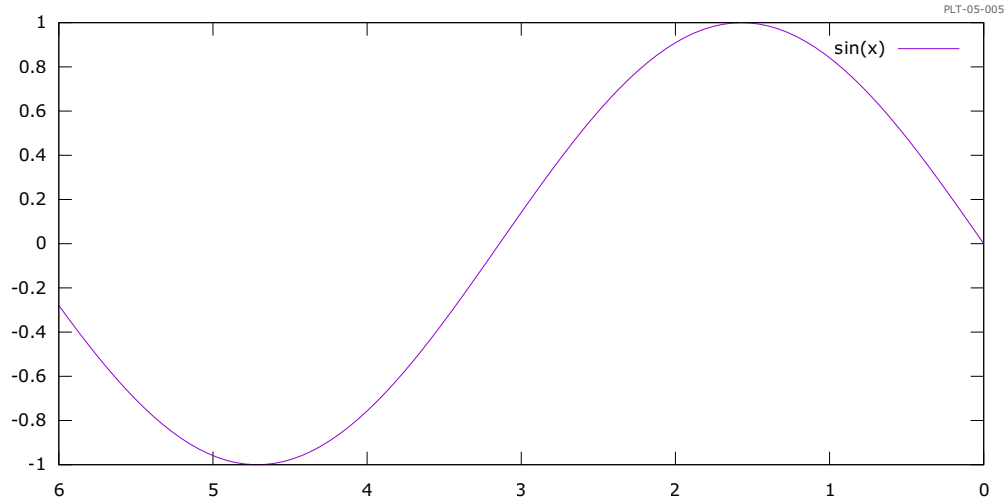
`0`	Fixed lower x limit.
`*`	An automatically chosen upper bound; for a function it uses the active default sampling range.

Reverse the horizontal direction

Compare with PLT-01-006 · Choose the horizontal interval.

An explicit descending range reverses the axis.

```
set xrange [6:0]
plot sin(x)
```



PLT-05-005 · Generated from the complete Bash block

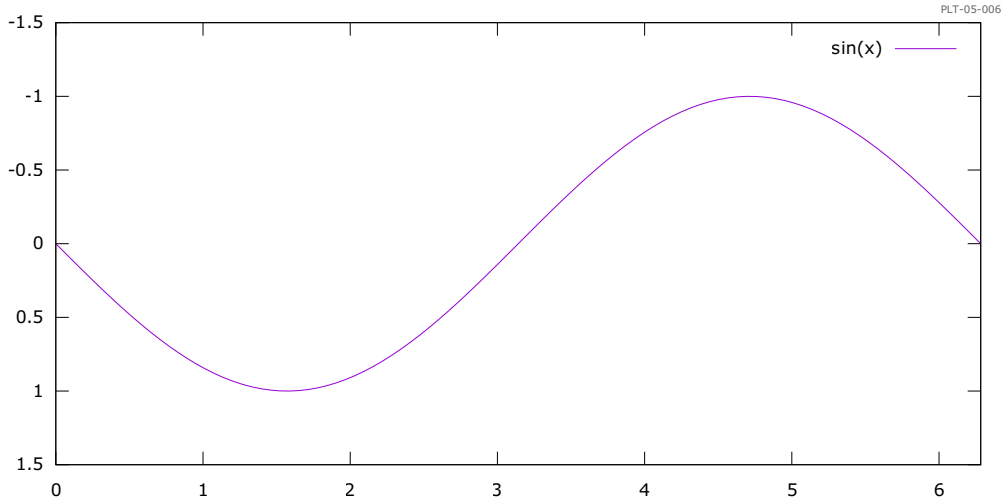
`[6:0]`	Left bound 6, right bound 0; x now increases toward the left.

Reverse the vertical direction

Compare with PLT-01-007 · Leave room above and below.

Reverse the visible y axis while retaining the same function.

```
set xrange [0:2*pi]
set yrange [1.5:-1.5]
plot sin(x)
```



PLT-05-006 · Generated from the complete Bash block

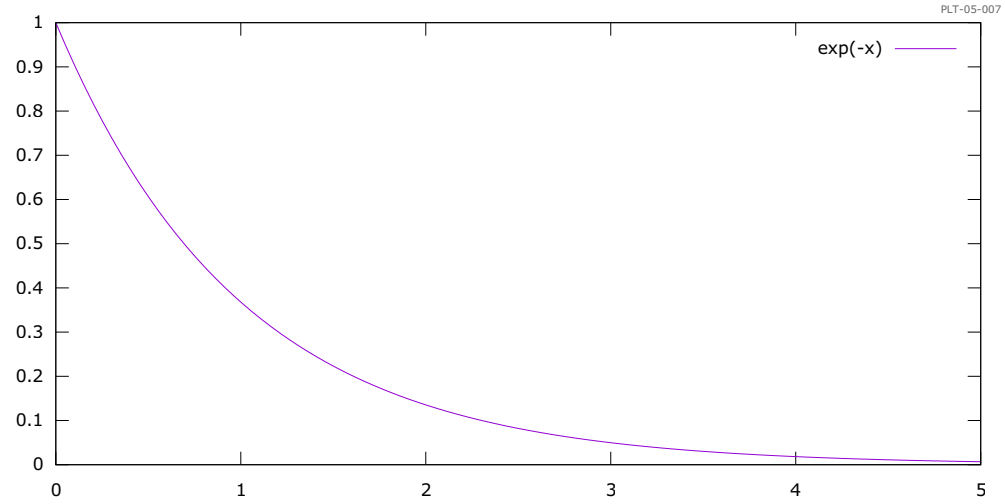
`[1.5:-1.5]` Lower screen edge corresponds to 1.5, upper edge to −1.5.

Begin with a positive function

New baseline: the commands below define this example.

Use an exponential decay as the reference for logarithmic scaling.

```
set xrange [0:5]
plot exp(-x)
```



PLT-05-007 · Generated from the complete Bash block

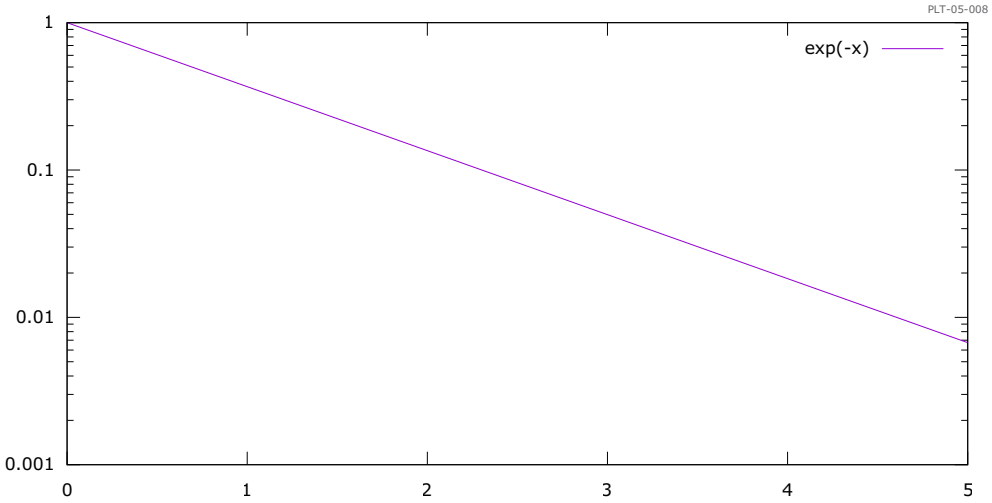
`exp(-x)`	The exponential function e raised to −x; all its values are positive.

Use a base-10 logarithmic y axis

Compare with PLT-05-007 · Begin with a positive function.

Equal distances now represent equal ratios, making the exponential a straight line.

```
set xrange [0:5]
set logscale y 10
plot exp(-x)
```



PLT-05-008 · Generated from the complete Bash block

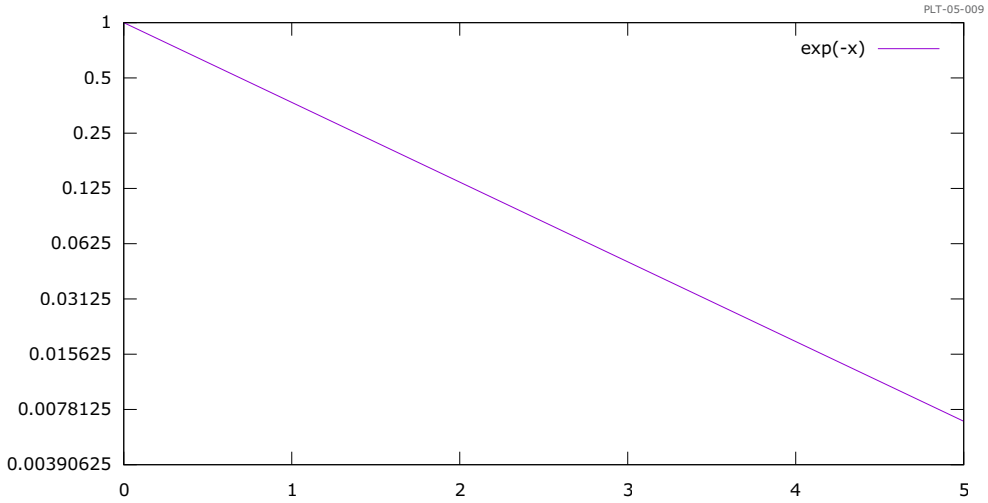
`logscale y`	Transform the vertical axis logarithmically.
`10`	Logarithm base. Plotted y values must be strictly positive.

Use a base-2 logarithmic y axis

Compare with PLT-05-007 · Begin with a positive function.

Change the logarithm base to show repeated halving naturally.

```
set xrange [0:5]
set logscale y 2
plot exp(-x)
```



PLT-05-009 · Generated from the complete Bash block

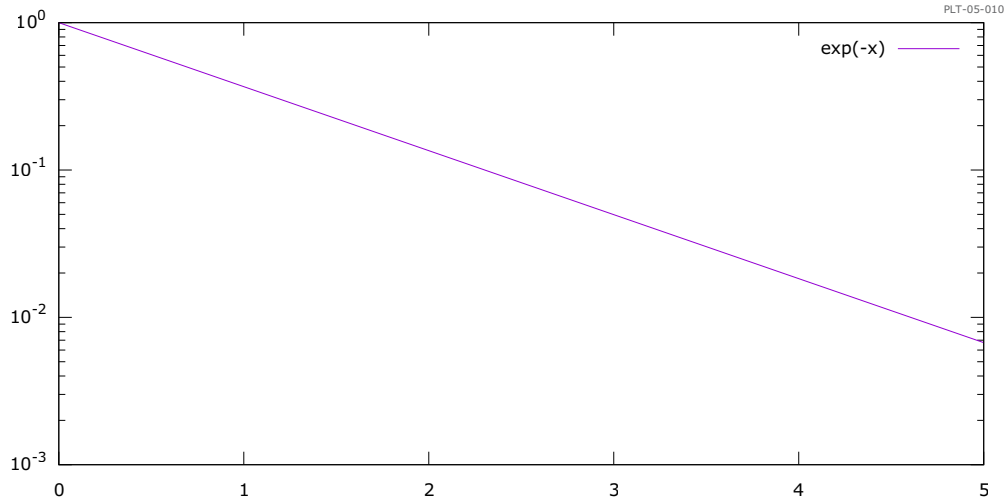
2 Base 2; neighbouring powers differ by a factor of two.

Format logarithmic ticks as powers

Compare with PLT-05-008 · Use a base-10 logarithmic y axis.

Enhanced text puts the exponent above the baseline.

```
set xrange [0:5]
set logscale y 10
set format y "10^{%L}"
plot exp(-x)
```



PLT-05-010 · Generated from the complete Bash block

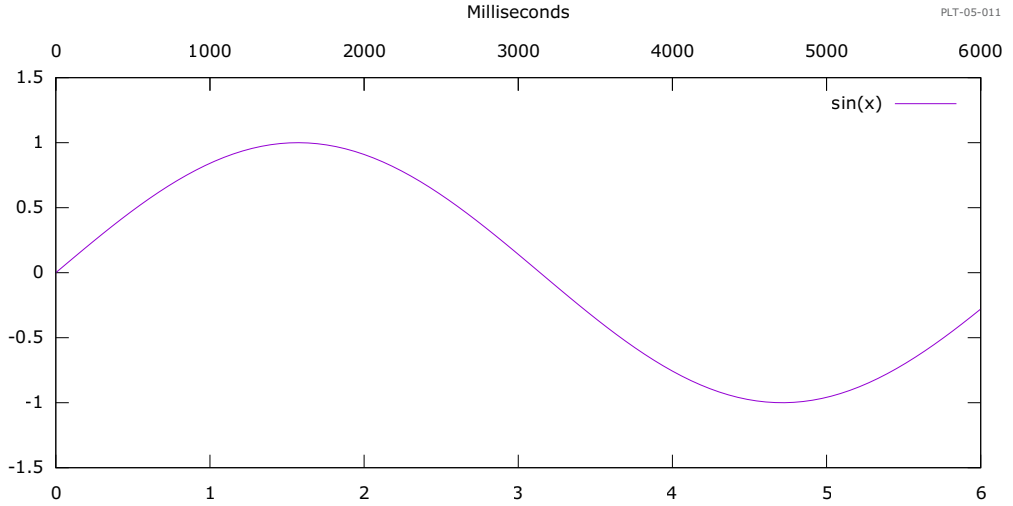
`%L`	The base-10 exponent used by this logarithmic tick formatter.
`^{...}`	Enhanced-text superscript syntax; it changes typesetting, not the data.

Link a second axis in different units

Compare with PLT-04-003 · Halve the tick interval.

The top axis represents the same coordinate in milliseconds instead of seconds.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set link x2 via x*1000 inverse x/1000
set x2tics
set x2label "Milliseconds"
plot sin(x)
```



PLT-05-011 · Generated from the complete Bash block

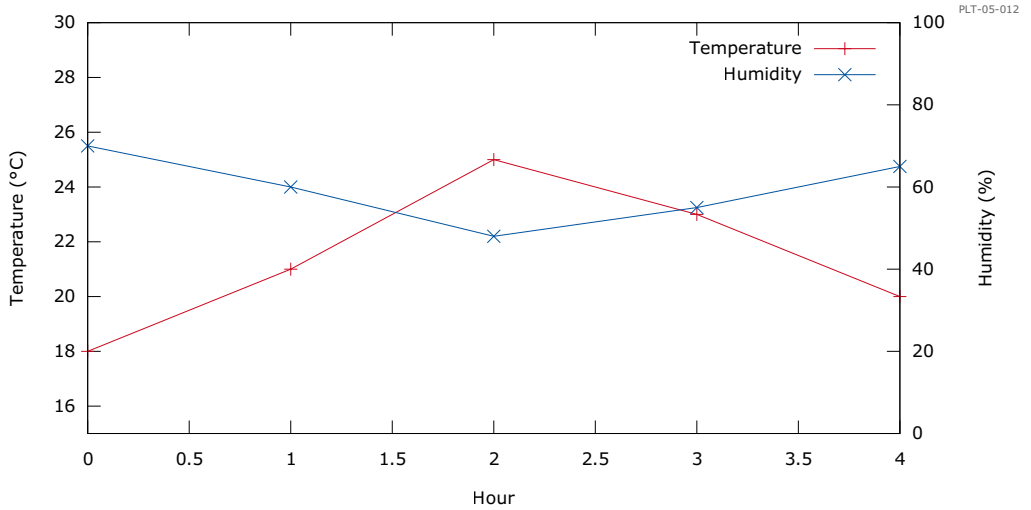
`x2`	The secondary, upper x axis.
`via x*1000`	Forward conversion from the primary axis.
`inverse x/1000`	Reverse conversion back to primary coordinates.
`set x2tics; x2label`	Show the upper tick scale and label it.

Plot a separate quantity on the right axis

New baseline: the commands below define this example.

An independent right axis can represent a second physical quantity. Both curves share x positions, but their heights use different scales; apparent crossings do not mean equal values.

```
$D << EOD
0 18 70
1 21 60
2 25 48
3 23 55
4 20 65
EOD
set xlabel "Hour"
set ylabel "Temperature (°C)"
set y2label "Humidity (%)"
set ytics nomirror
set y2tics
set yrange [15:30]
set y2range [0:100]
plot $D using 1:2 axes x1y1 with linespoints \
    lc rgb "#CC071E" title "Temperature", \
    $D using 1:3 axes x1y2 with linespoints \
    lc rgb "#00549F" title "Humidity"
```



PLT-05-012 · Generated from the complete Bash block

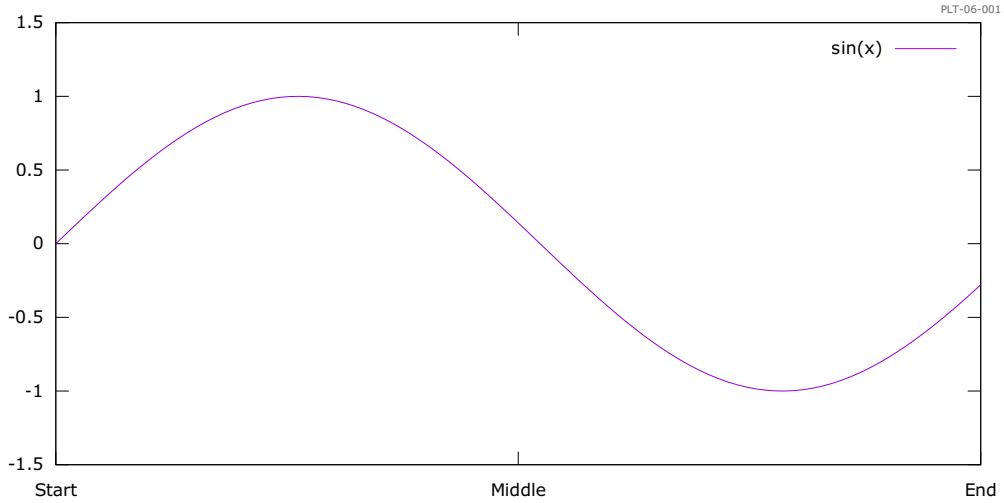
`axes x1y2`	Use the bottom x axis and independent right y axis.
`y2range; y2tics; y2label`	Configure the secondary scale explicitly.
`ytics nomirror`	Avoid copying primary ticks to the right border.

Replace numbers with words

Compare with PLT-04-003 · Halve the tick interval.

A tick list can pair arbitrary labels with numeric positions.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
plot sin(x)
```



PLT-06-001 · Generated from the complete Bash block

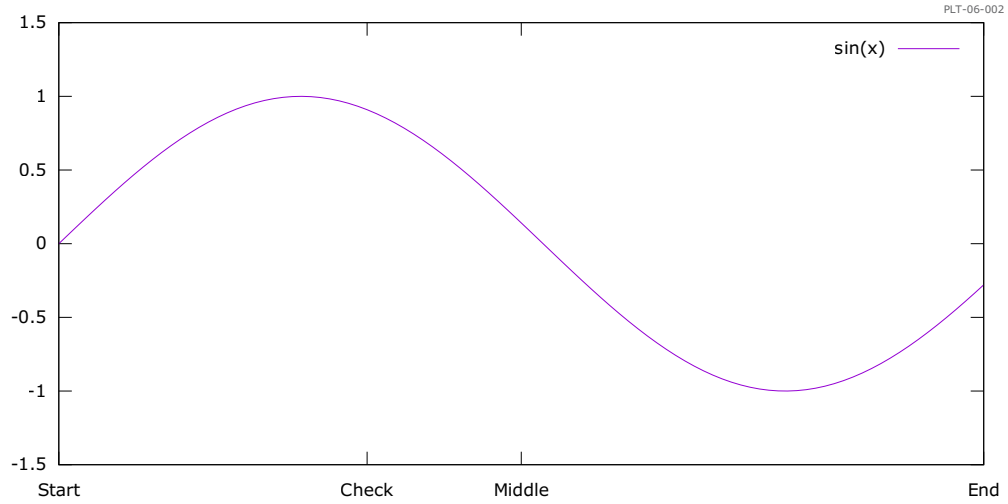
`"Start" 0`	Display Start at x=0.
`"Middle" 3; "End" 6`	Each quoted label is followed by its data position.

Add one labelled tick

Compare with PLT-06-001 · Replace numbers with words.

add preserves the existing ticks and appends a new one.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics add ("Check" 2)
plot sin(x)
```



PLT-06-002 · Generated from the complete Bash block

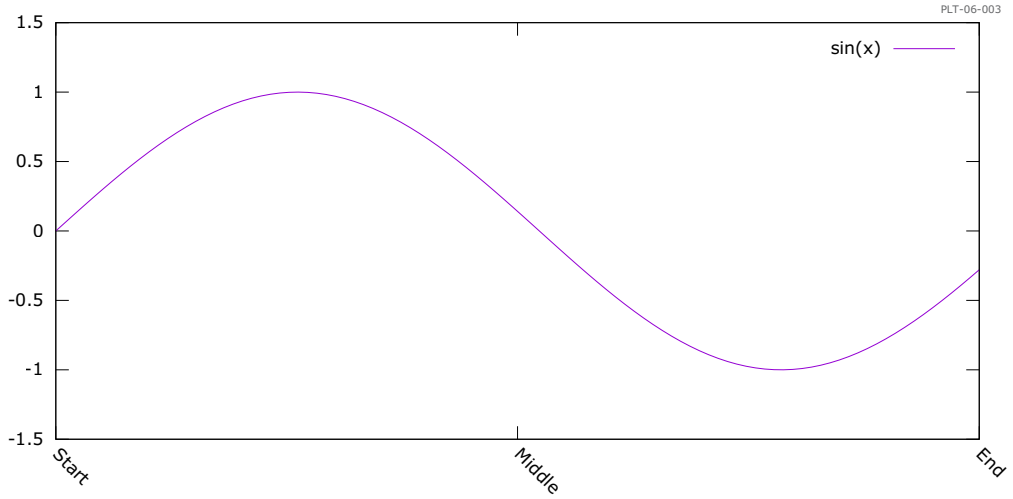
add	Extend the current tick list instead of replacing it.
"Check" 2	Text and x position of the additional tick.

Rotate tick labels

Compare with PLT-06-001 · Replace numbers with words.

Rotate the text without moving the tick positions.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics rotate by -45
plot sin(x)
```



PLT-06-003 · Generated from the complete Bash block

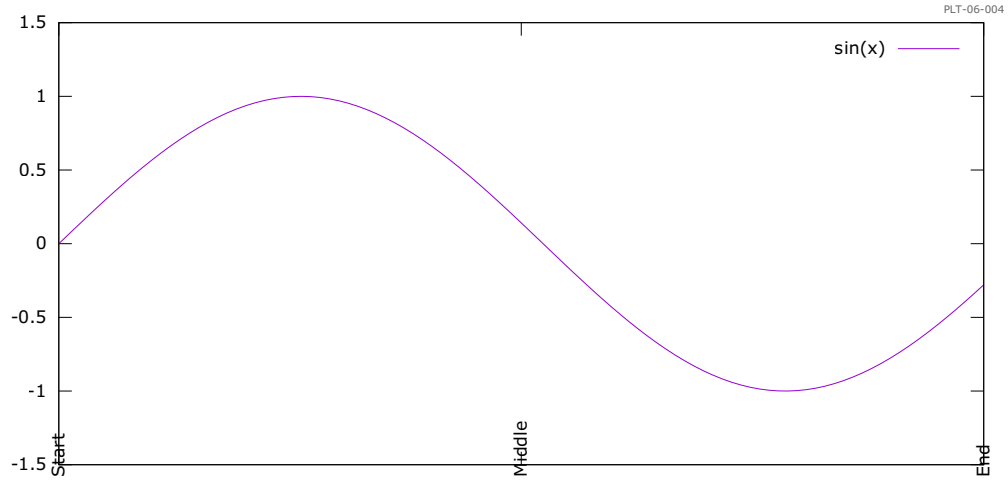
`rotate by`	Set the label rotation in degrees.
`-45`	Tilt clockwise by 45 degrees.

Stand tick labels vertically

Compare with PLT-06-003 · Rotate tick labels.

Rotate the same three labels through 90 degrees. This can help when long category names compete for horizontal space.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics rotate by 90
plot sin(x)
```



PLT-06-004 · Generated from the complete Bash block

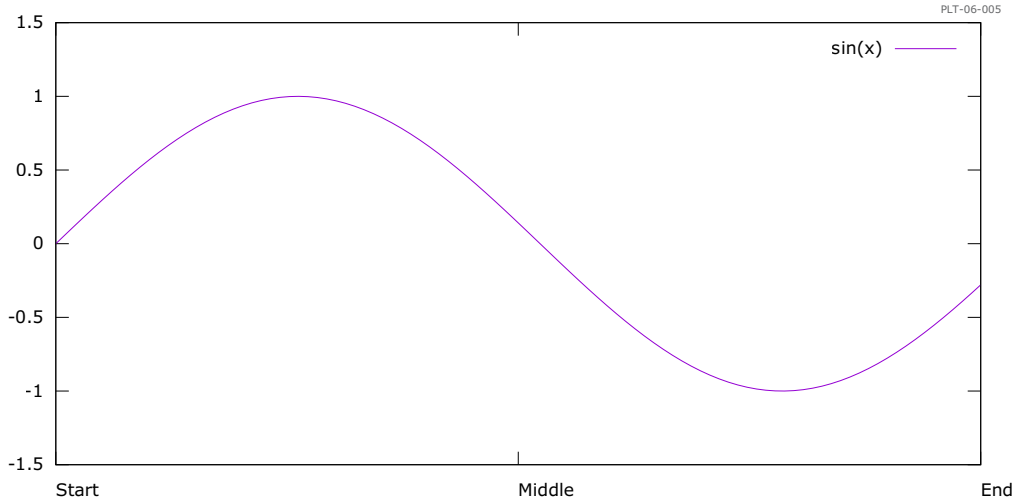
`rotate by 90`	Rotate tick text counterclockwise by 90 degrees.

Left-align horizontal tick labels

Compare with PLT-06-001 · Replace numbers with words.

Keep the tick positions fixed and change how their labels align with them. Each word starts at its tick instead of being centred on it.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics left
plot sin(x)
```



PLT-06-005 · Generated from the complete Bash block

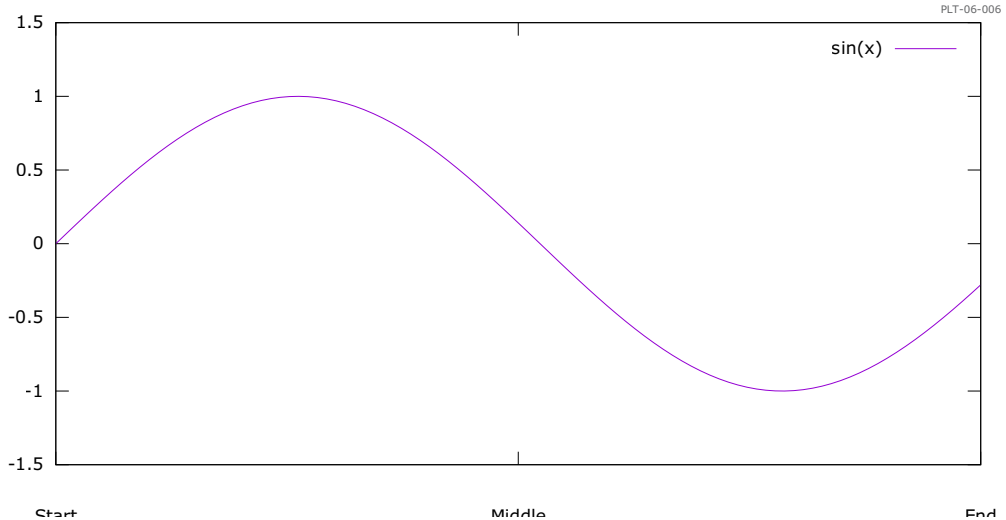
`left`	Align the left edge of each x tick label with its tick position.

Offset tick labels

Compare with PLT-06-001 · Replace numbers with words.

Move the label text without moving the tick marks.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics offset 0,-1
plot sin(x)
```



PLT-06-006 · Generated from the complete Bash block

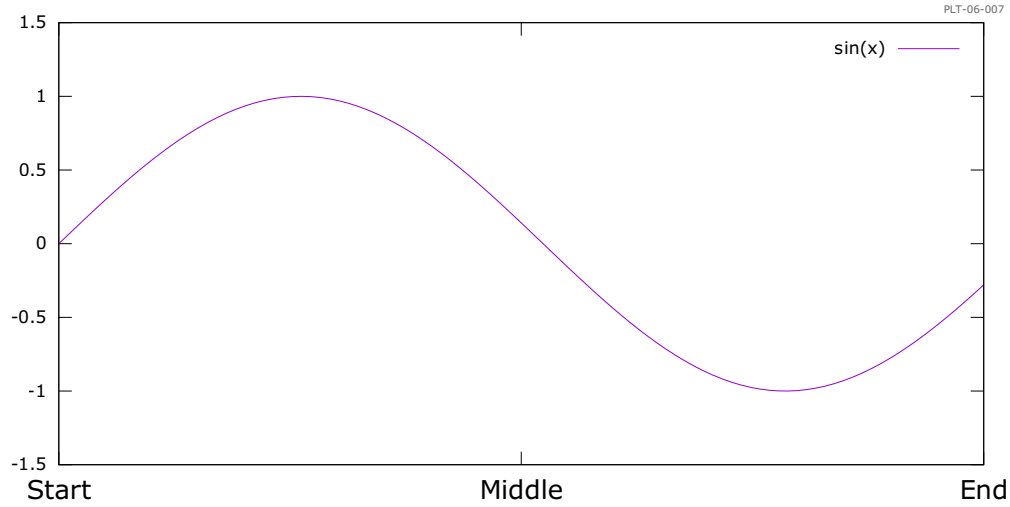
`offset 0,-1`	Horizontal shift 0; vertical shift −1.
`Default offset units`	Character widths horizontally and character heights vertically; these are not data units.

Increase tick-label size

Compare with PLT-06-001 · Replace numbers with words.

A font string can change just the size while retaining the current font family.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics font ",30"
plot sin(x)
```



PLT-06-007 · Generated from the complete Bash block

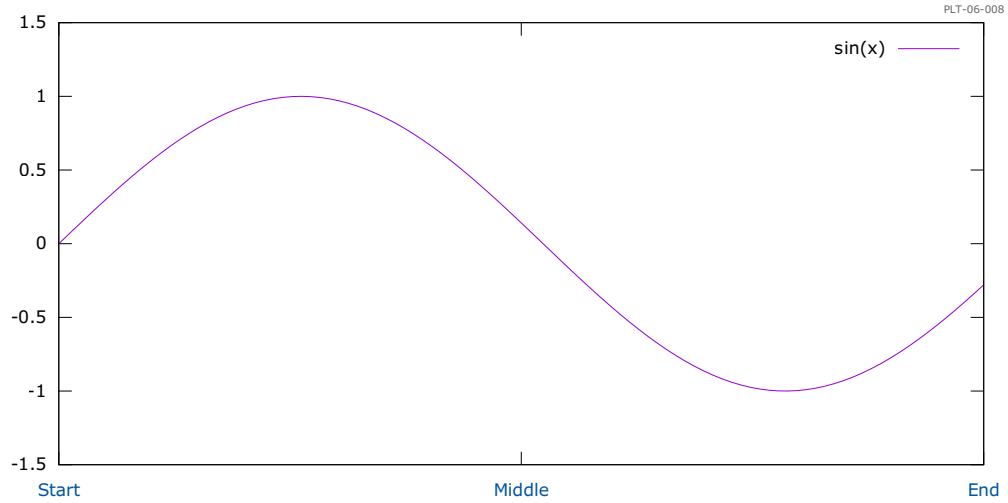
`font`	Choose the tick-label font.
`",30"`	Empty family name means retain it; 30 is the nominal font size. Its physical appearance depends on the terminal.

Colour tick-label text

Compare with PLT-06-001 · Replace numbers with words.

Text colour is independent of the curve colour.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics textcolor rgb "#00549F"
plot sin(x)
```



PLT-06-008 · Generated from the complete Bash block

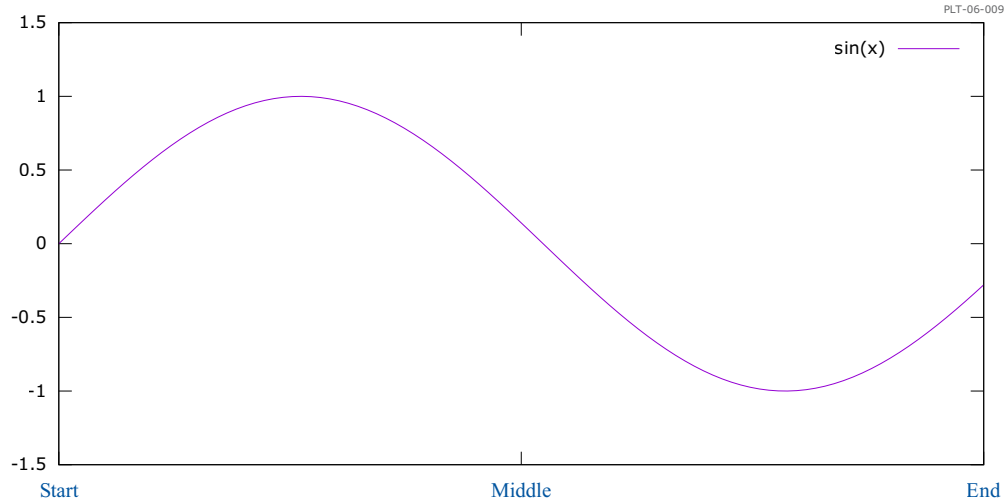
`textcolor`	Set the tick-label colour.
`rgb "#00549F"`	An explicit red/green/blue hexadecimal colour.

Use a different tick-label font

Compare with PLT-06-008 · Colour tick-label text.

Change the x tick labels to a larger serif face while retaining their colour. Font-family availability depends on the viewer and installed fonts.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set xtics textcolor rgb "#00549F" font "serif,24"
plot sin(x)
```



PLT-06-009 · Generated from the complete Bash block

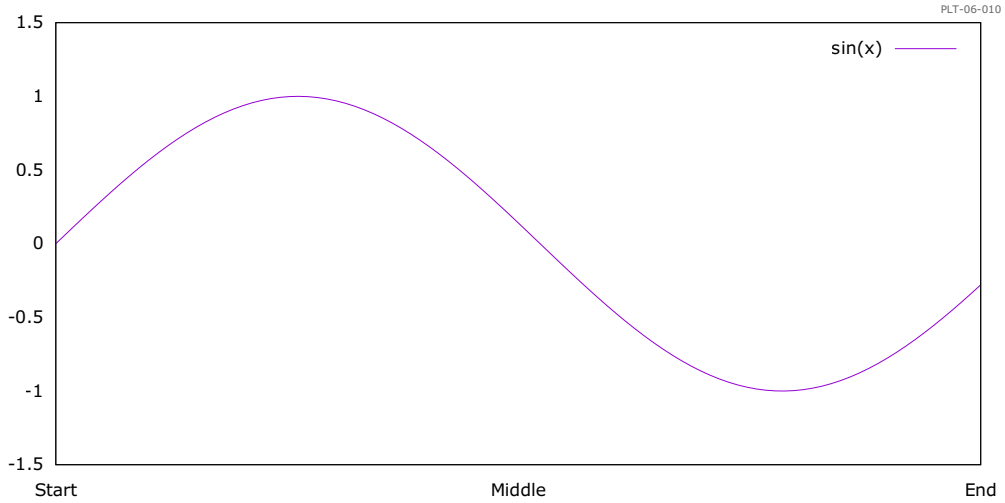
`font "serif,24"`	Request a serif font at nominal size 24 for x tick labels only.

Keep labels and hide tick strokes

Compare with PLT-06-001 · Replace numbers with words.

Zero-length tick marks leave the labels and border visible. Unlike unsetting ticks, this retains the numeric or categorical coordinate labels.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics ("Start" 0,"Middle" 3,"End" 6)
set tics scale 0
plot sin(x)
```



PLT-06-010 · Generated from the complete Bash block

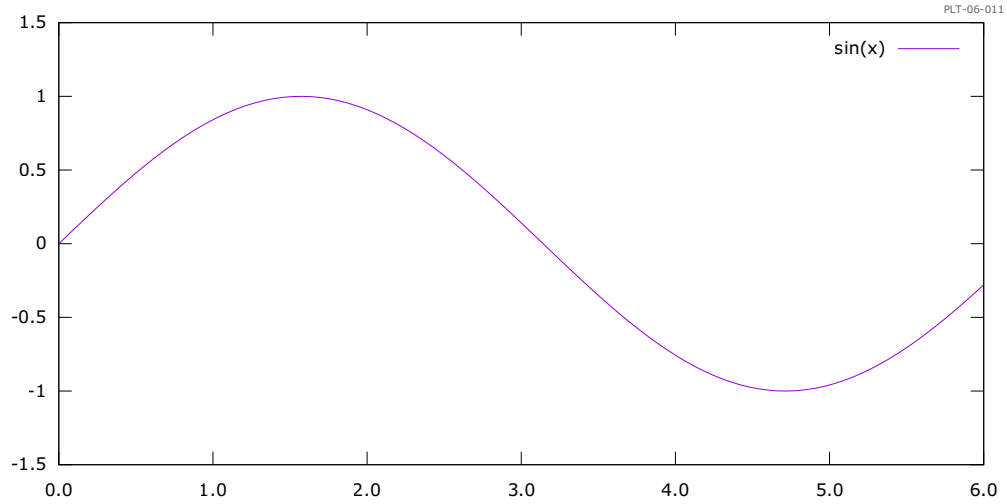
`scale 0`	Set the length of major tick strokes to zero; default minor length follows the major scale.

Print one decimal place

Compare with PLT-04-003 · Halve the tick interval.

The numeric tick positions stay the same; only their formatting changes.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set format x "%.1f"
plot sin(x)
```



PLT-06-011 · Generated from the complete Bash block

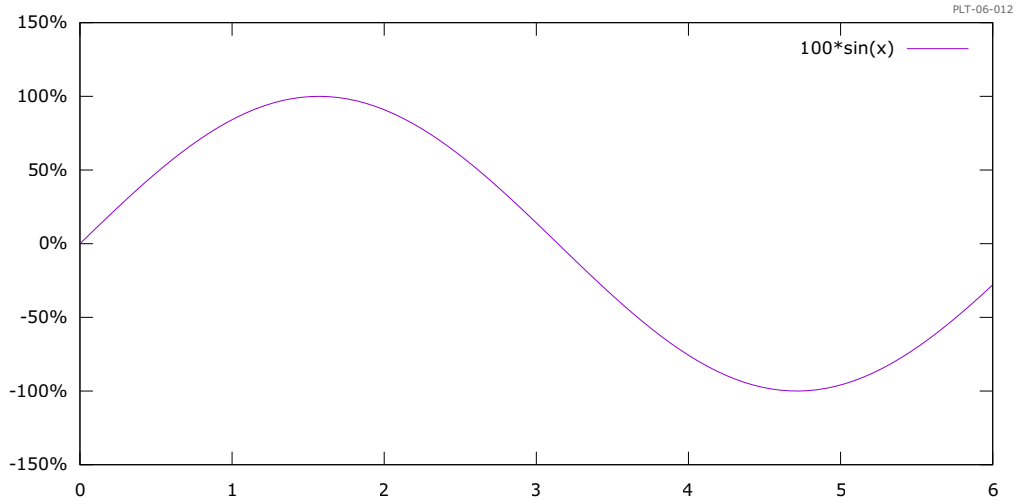
`%f`	Print a fixed-point decimal number.
`.1`	Keep one digit after the decimal point.

Format values as percentages

Compare with PLT-04-007 · Change y ticks independently.

Multiply the function by 100 and append a literal percent sign to each y label.

```
set xrange [0:6]
set xtics 1
set yrange [-150:150]
set ytics 50
set format y "%.0f%%"
plot 100*sin(x)
```



PLT-06-012 · Generated from the complete Bash block

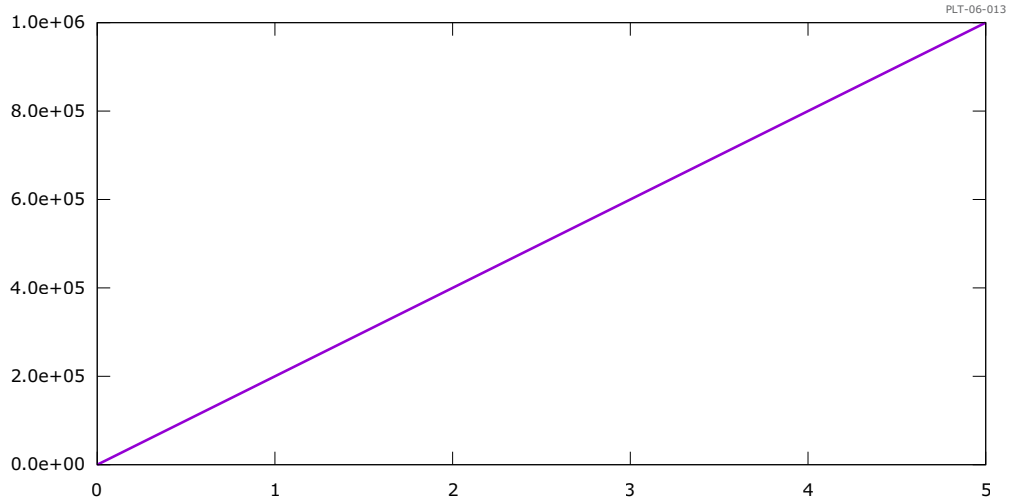
`100*sin(x); [-150:150]`	Scale the plotted values and their visible range.
`ytics 50`	A major tick every 50 y units.
`%.0f%%`	No decimal digits; %% prints one literal percent sign.

Use scientific tick notation

New baseline: the commands below define this example.

Formatting changes the printed labels, not the coordinates. Scientific notation keeps large values compact without changing the axis to a logarithmic scale.

```
set xrange [0:5]
set yrange [0:1000000]
set format y "%.1e"
plot 200000*x with lines lw 3 notitle
```



PLT-06-013 · Generated from the complete Bash block

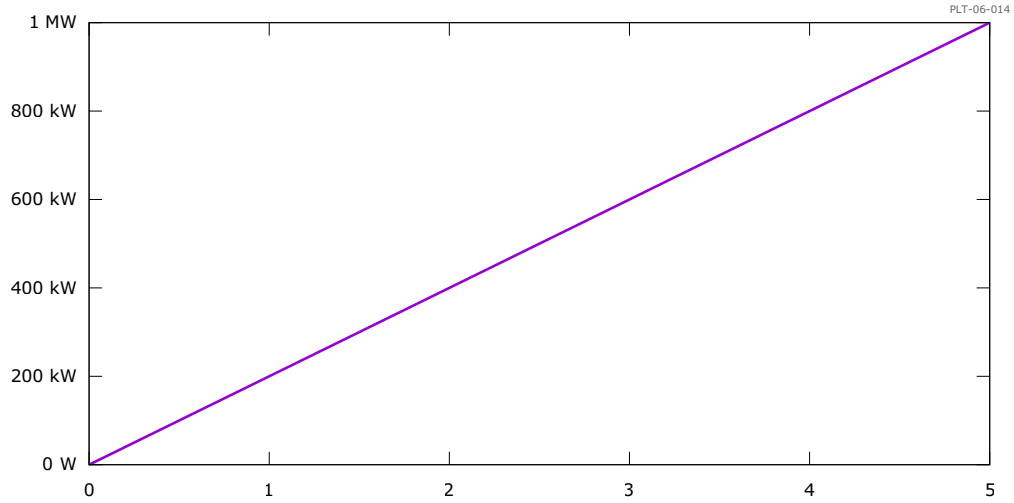
`format y "%.1e"`	Print one decimal digit and a base-10 exponent.
`yrange`	The axis remains linear; only its labels change.

Use engineering prefixes on ticks

Compare with PLT-06-013 · Use scientific tick notation.

gnuplot has tick-format extensions beyond ordinary printf. Pair the engineering mantissa with an SI prefix and a physical unit.

```
set xrange [0:5]
set yrange [0:1000000]
set format y "%.0s %cW"
plot 200000*x with lines lw 3 notitle
```



PLT-06-014 · Generated from the complete Bash block

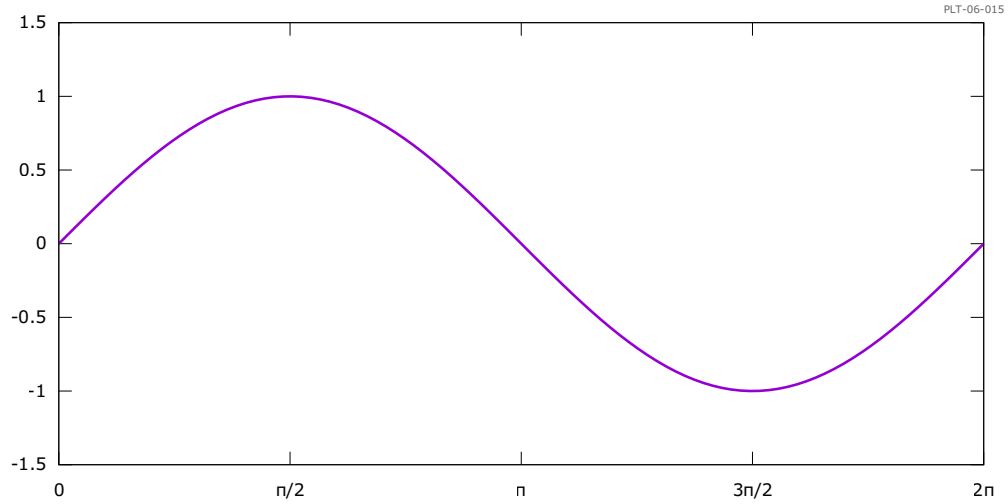
`%.0s`	Mantissa using a power of ten that is a multiple of three.
`%c`	Replace that power with an SI prefix, such as k or M.
`W`	Append the literal unit; format does not convert the data.

Label an angular axis in multiples of pi

Compare with PLT-01-007 · Leave room above and below.

Explicit positions give familiar angular labels. The function still receives radians; writing pi in a label does not convert its input.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xtics ("0" 0, "π/2" pi/2, "π" pi, \
    "3π/2" 3*pi/2, "2π" 2*pi)
plot sin(x) with lines lw 3 notitle
```



PLT-06-015 · Generated from the complete Bash block

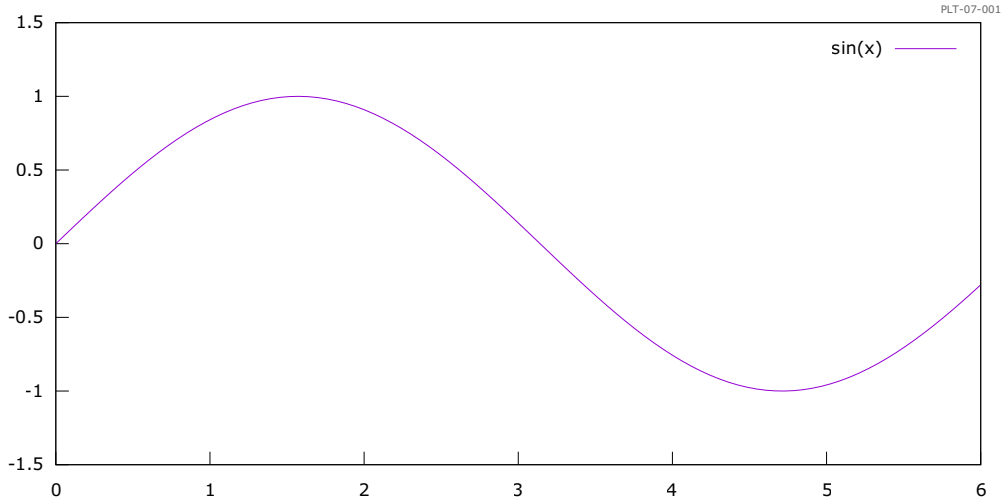
`"π/2" pi/2`	The quoted text is the label; the expression is its numeric position.
`pi`	Built-in constant used for tick positions and the function domain.

Remove mirrored ticks

Compare with PLT-04-003 · Halve the tick interval.

Keep lower and left ticks; remove their copies on the top and right borders.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
plot sin(x)
```



PLT-07-001 · Generated from the complete Bash block

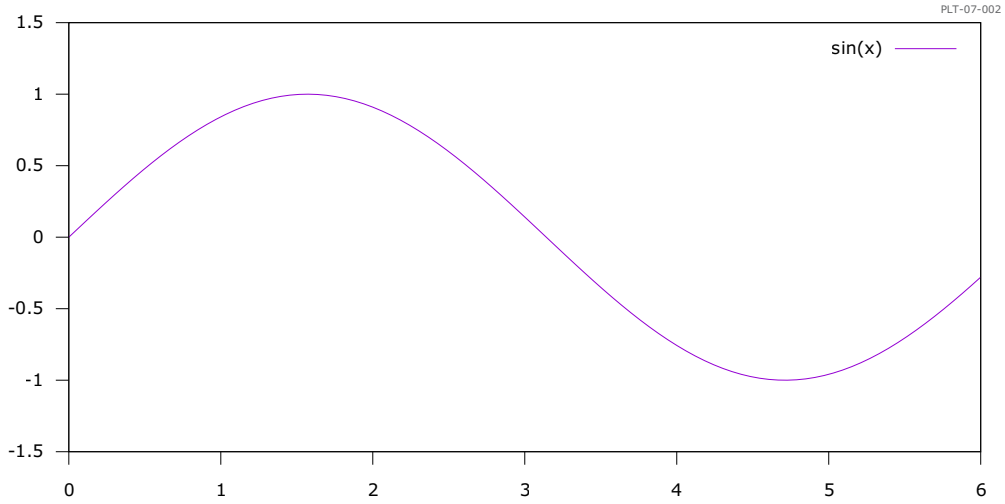
`tics`	Apply to the tick axes together.
`nomirror`	Do not repeat primary ticks on the opposite borders.

Point ticks outward

Compare with PLT-07-001 · Remove mirrored ticks.

Change the direction of the marks while keeping labels in place.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set tics out
plot sin(x)
```



PLT-07-002 · Generated from the complete Bash block

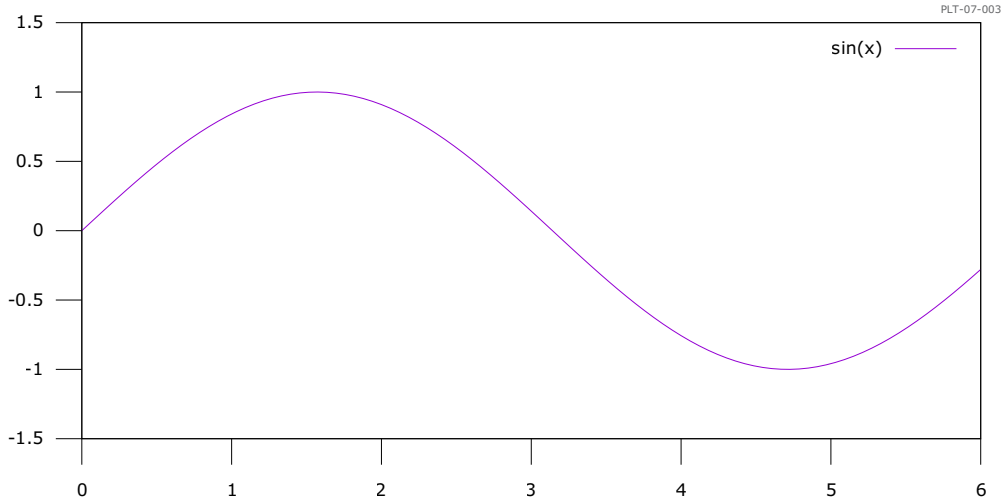
`out`	Draw ticks outside the plot region; in selects the opposite direction.

Lengthen major ticks

Compare with PLT-07-002 · Point ticks outward.

Tick scale is a relative length, not a change to the axis scale.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set tics out
set tics scale 2
plot sin(x)
```



PLT-07-003 · Generated from the complete Bash block

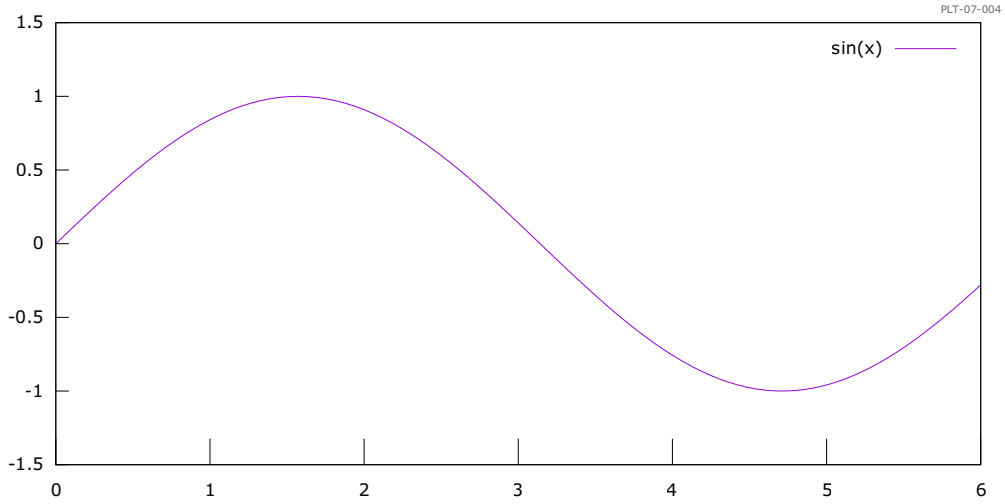
`scale 2`	Make major tick marks twice the normal length.

Give the two axes different tick lengths

Compare with PLT-07-001 · Remove mirrored ticks.

Lengthen only the x tick marks. The y tick marks retain their default length, making the independent axis stand out.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set xtics scale 2
plot sin(x)
```



PLT-07-004 · Generated from the complete Bash block

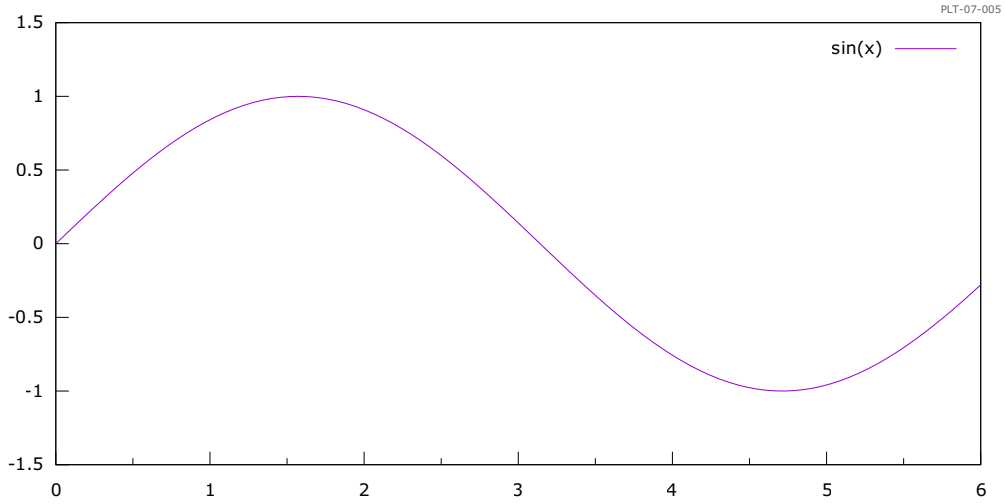
`xtics scale 2`	Draw x major ticks at twice the default length without changing y ticks.

Add a minor tick halfway

Compare with PLT-07-001 · Remove mirrored ticks.

Minor ticks divide major intervals without adding labels.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set mxtics 2
plot sin(x)
```



PLT-07-005 · Generated from the complete Bash block

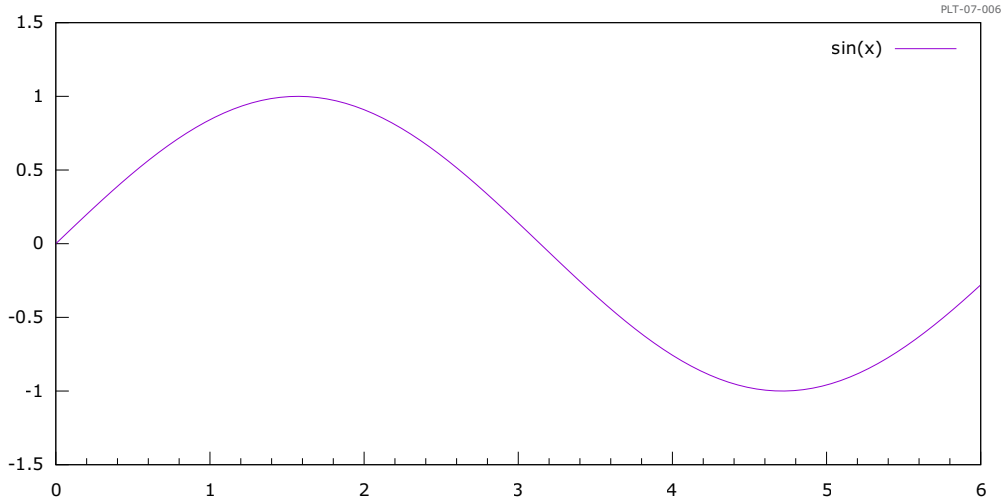
`mxtics`	Configure minor x ticks.
`2`	Divide each major interval into two parts: one minor tick halfway.

Add four minor ticks

Compare with PLT-07-005 · Add a minor tick halfway.

More subdivisions give finer visual guidance without more numbers.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set mxtics 5
plot sin(x)
```



PLT-07-006 · Generated from the complete Bash block

5

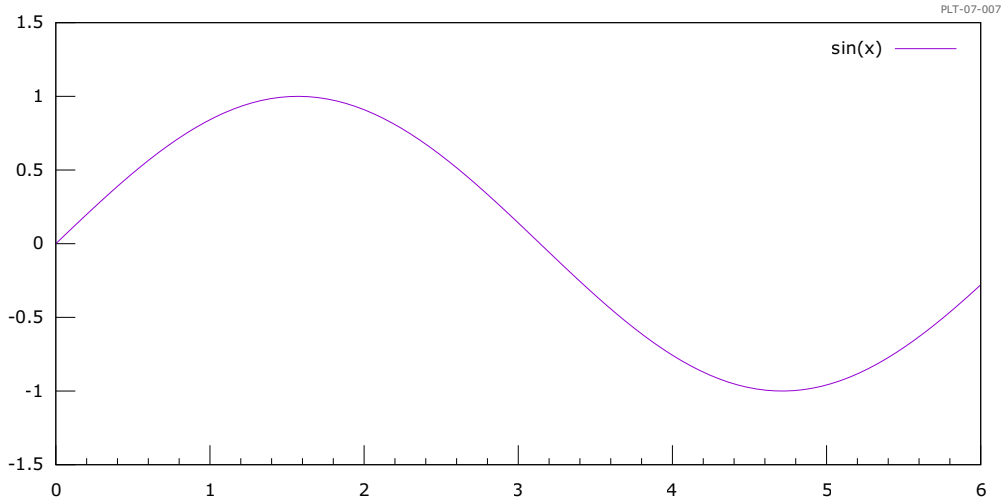
Divide each major interval into five parts, giving four interior minor ticks.

Set major and minor lengths

Compare with PLT-07-006 · Add four minor ticks.

Two scale values distinguish the two kinds of marks.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set mxtics 5
set tics scale 1.5,0.5
plot sin(x)
```



PLT-07-007 · Generated from the complete Bash block

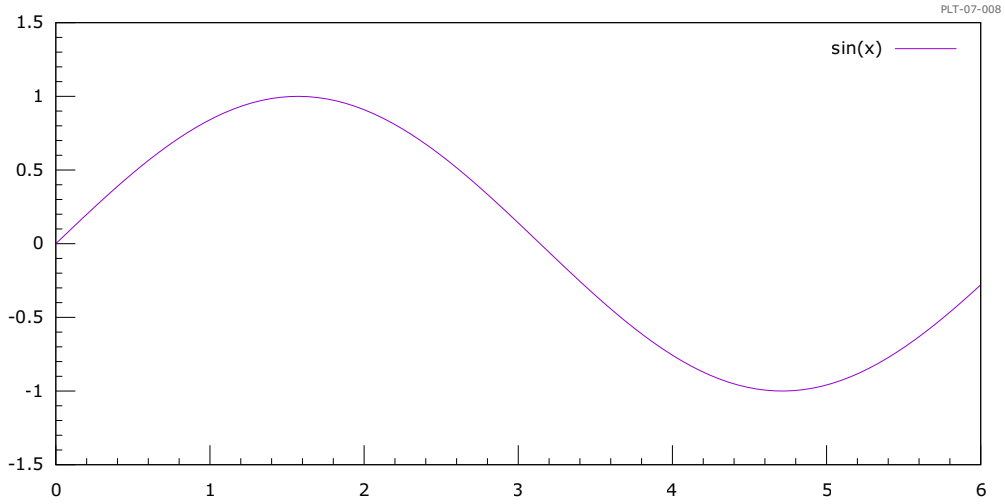
`1.5`	Major tick length multiplier.
`0.5`	Minor tick length multiplier; the second value is independent of the first.

Subdivide the vertical axis

Compare with PLT-07-007 · Set major and minor lengths.

Add minor ticks on the y axis as well as the x axis. Each major y interval is split into five equal intervals.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set mxtics 5
set tics scale 1.5,0.5
set mytics 5
plot sin(x)
```



PLT-07-008 · Generated from the complete Bash block

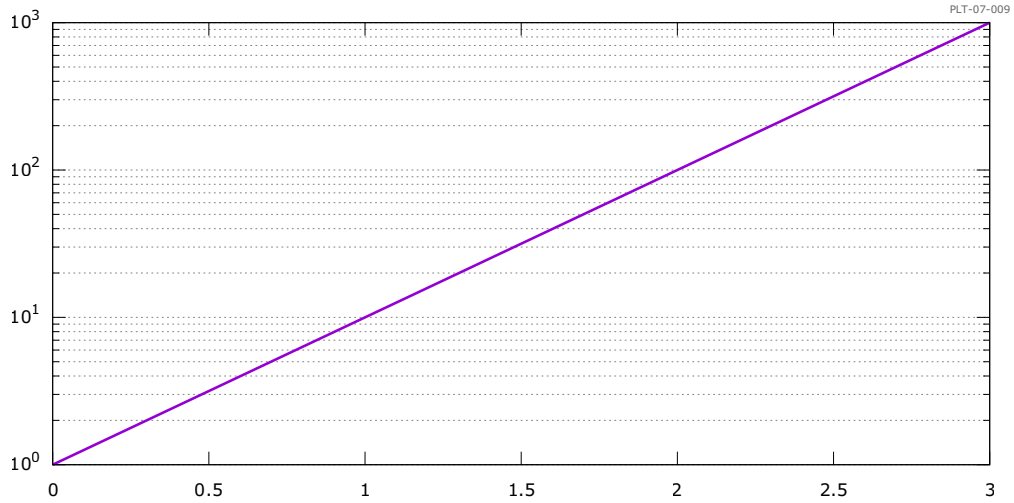
`mytics 5` Place four minor ticks between successive major y ticks.

Show minor ticks on a logarithmic axis

Compare with PLT-05-010 · Format logarithmic ticks as powers.

Logarithmic minor ticks follow the multiplicative scale. The default minor subdivisions for base ten mark values such as 2 through 9 within each decade.

```
set xrange [0:3]
set yrange [1:1000]
set logscale y 10
set format y "10^{%L}"
set mytics default
set grid ytics mytics
plot 10**x with lines lw 3 notitle
```



PLT-07-009 · Generated from the complete Bash block

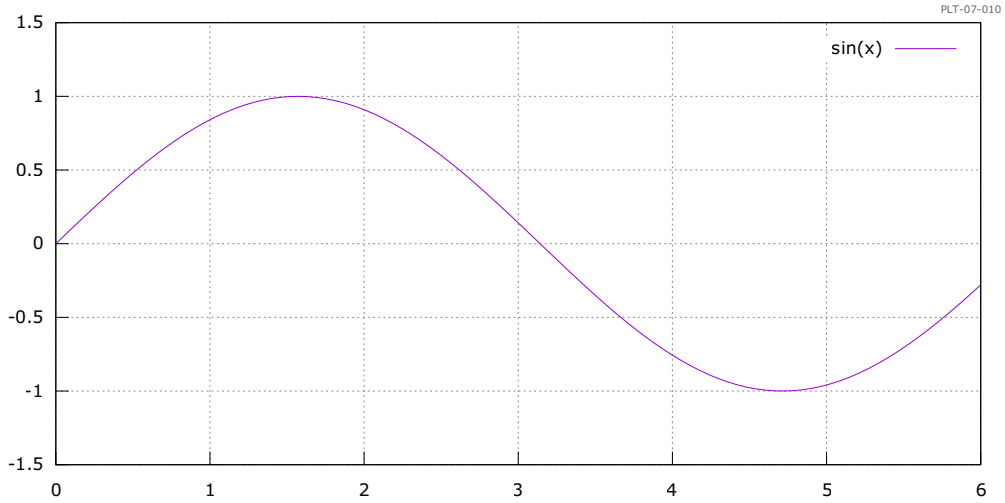
`mytics default`	Use the default subdivisions appropriate to the logarithmic axis.
`grid ytics mytics`	Draw both major and minor horizontal grid lines.
`10^{%L}`	Use enhanced text to display powers of the logscale base.

Add a major grid

Compare with PLT-07-001 · Remove mirrored ticks.

Grid lines follow the selected tick positions.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set grid xtics ytics
plot sin(x)
```



PLT-07-010 · Generated from the complete Bash block

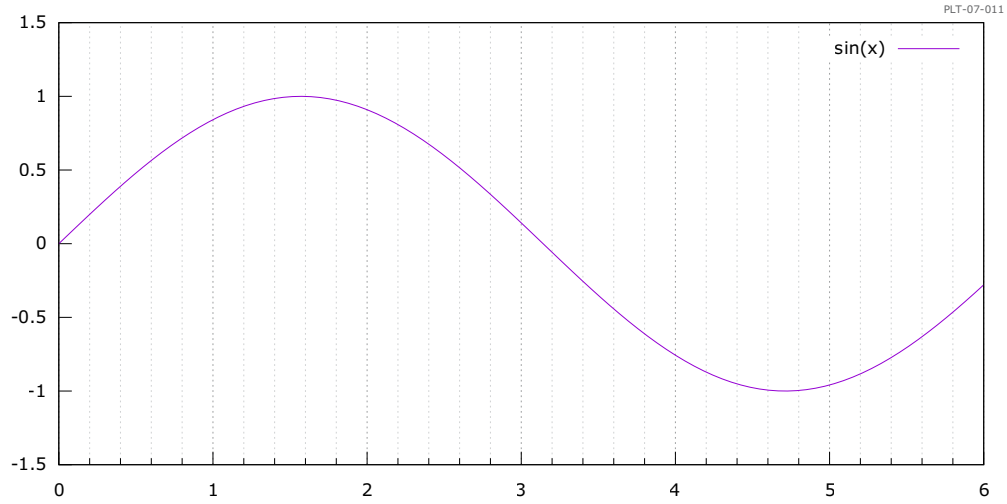
`grid`	Enable grid lines.
`xtics ytics`	Draw lines through major x and major y ticks.

Show a lighter minor grid

Compare with PLT-07-006 · Add four minor ticks.

The first style applies to major lines; the second applies to minor lines.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set mxtics 5
set grid xtics mxtics \
    linewidth 1 linecolor rgb "#9C9E9F", \
    linewidth 0.5 linecolor rgb "#CFD1D2"
plot sin(x)
```



PLT-07-011 · Generated from the complete Bash block

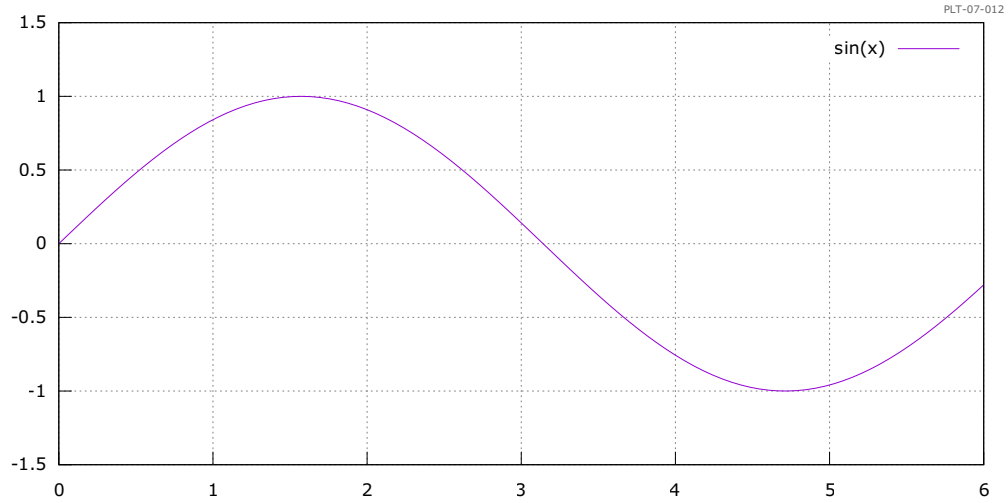
`mxtics`	Also draw through minor x tick positions.
`linewidth 1; 0.5`	Relative widths for the major and minor grid lines.
`linecolor rgb`	Explicit colours; the comma separates the two grid styles.

Use dashed grid lines

Compare with PLT-07-010 · Add a major grid.

Keep the same grid positions but replace solid grid strokes with a dash pattern.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set grid xtics ytics dashtype 2
plot sin(x)
```



PLT-07-012 · Generated from the complete Bash block

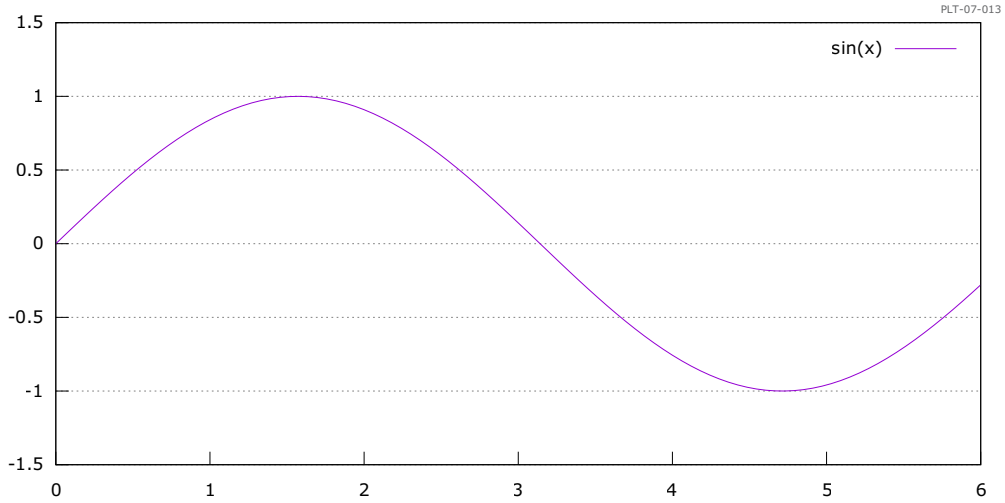
`dashtype 2`	Select dash pattern 2 for the grid; patterns depend on the terminal.

Keep only horizontal grid lines

Compare with PLT-07-010 · Add a major grid.

Remove the vertical grid lines so that horizontal guides alone support comparisons of y values.

```
set yrange [-1.5:1.5]
set xrange [0:6]
set xtics 1
set tics nomirror
set grid noxtics ytics
plot sin(x)
```



PLT-07-013 · Generated from the complete Bash block

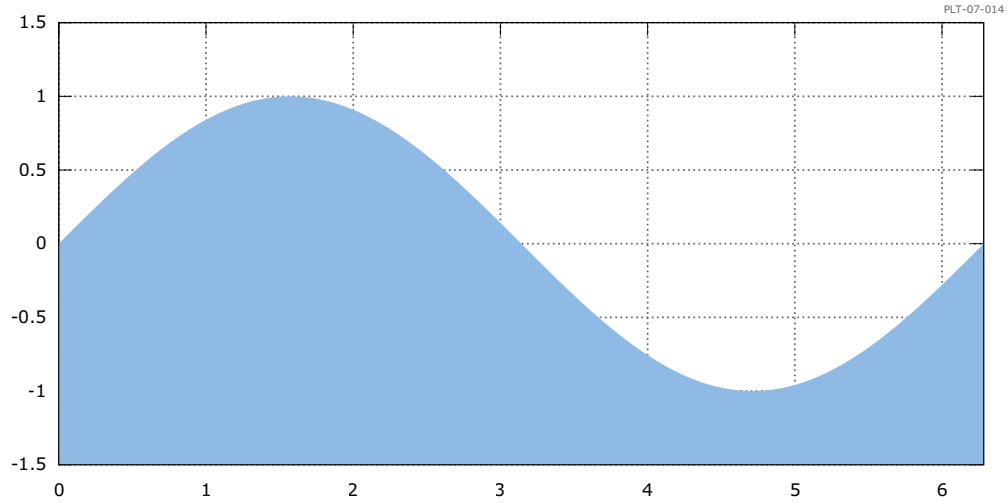
`noxtics`	Exclude x tick positions from the grid.
`ytics`	Draw horizontal grid lines at major y tick positions.

Draw the grid at the back

New baseline: the commands below define this example.

An opaque fill makes drawing order visible. Grid lines at the back are covered by the fill; moving the grid to the front draws them over it.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set grid xtics ytics back lc rgb "#646567" lw 2
plot sin(x) with filledcurves y1=-1.5 \
    fc rgb "#8EBAE5" fs solid 1.0 notitle
```



PLT-07-014 · Generated from the complete Bash block

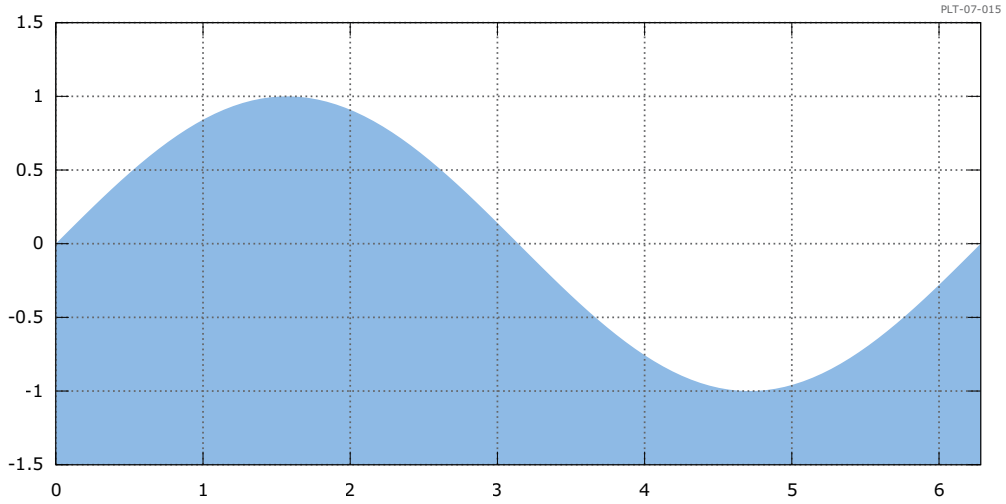
`grid back`	Choose whether the grid is drawn behind or over the plotted data.
`fs solid 1.0`	Use an opaque fill so that the layer change is easy to see.

Draw the grid at the front

Compare with PLT-07-014 · Draw the grid at the back.

An opaque fill makes drawing order visible. Grid lines at the back are covered by the fill; moving the grid to the front draws them over it.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set grid xtics ytics front lc rgb "#646567" lw 2
plot sin(x) with filledcurves y1=-1.5 \
    fc rgb "#8EBAE5" fs solid 1.0 notitle
```



PLT-07-015 · Generated from the complete Bash block

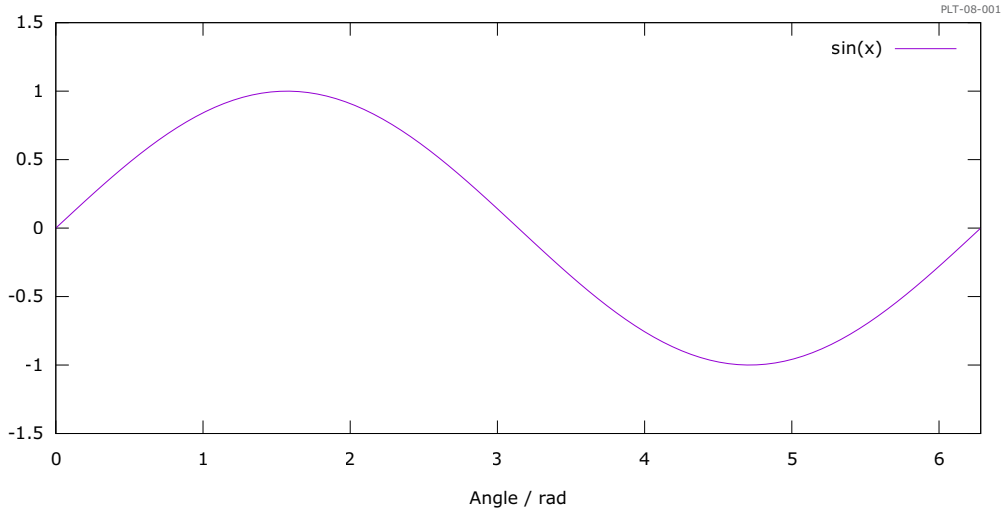
`grid front`	Choose whether the grid is drawn behind or over the plotted data.
`fs solid 1.0`	Use an opaque fill so that the layer change is easy to see.

Name the x axis

Compare with PLT-01-007 · Leave room above and below.

Axis labels describe the plotted quantity and its unit.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
plot sin(x)
```



PLT-08-001 · Generated from the complete Bash block

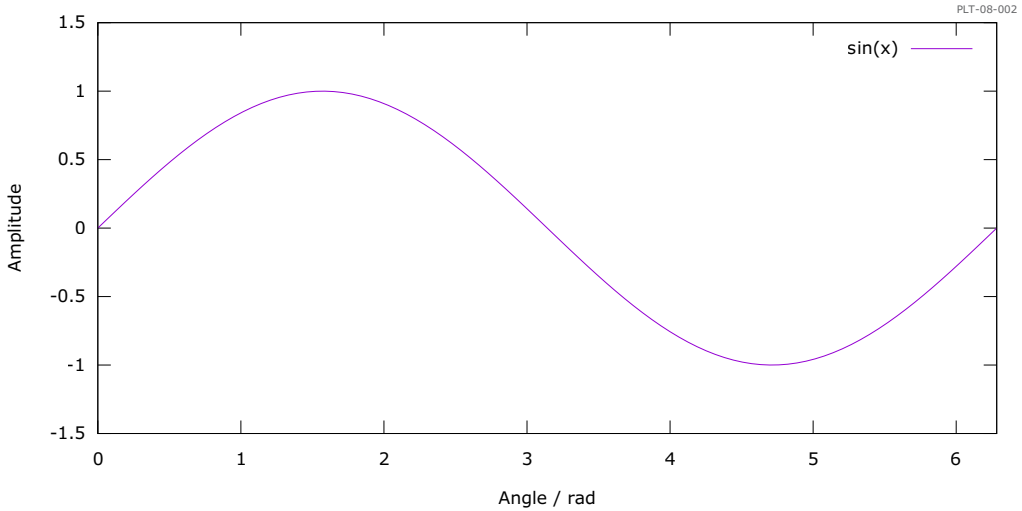
`xlabel`	Text label for the primary x axis.
`"Angle / rad"`	Literal label text; the slash is part of the text, not division.

Name the y axis

Compare with PLT-08-001 · Name the x axis.

Add a label to the left-hand scale.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "Amplitude"
plot sin(x)
```



PLT-08-002 · Generated from the complete Bash block

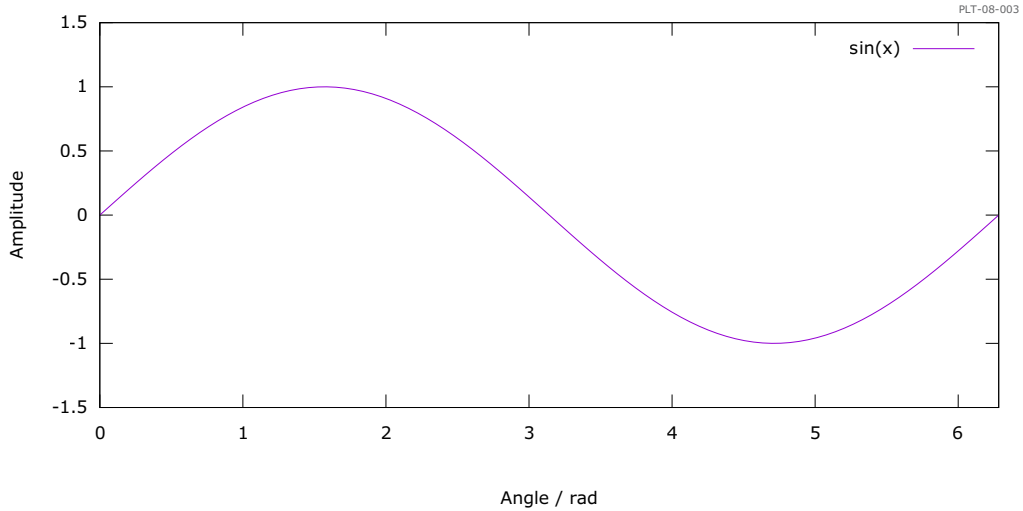
`ylabel`	Text label for the primary y axis.

Move the x label downward

Compare with PLT-08-002 · Name the y axis.

A negative vertical offset increases the distance below the axis.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set ylabel "Amplitude"
set xlabel "Angle / rad" offset 0,-1
plot sin(x)
```



PLT-08-003 · Generated from the complete Bash block

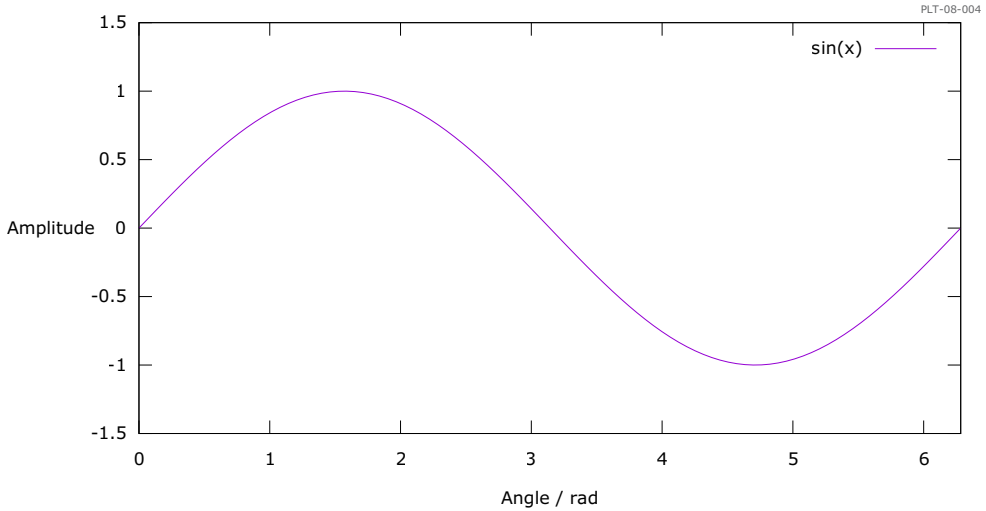
`offset 0,-1` — Shift zero character widths horizontally and one character height downward.

Make the y label horizontal

Compare with PLT-08-002 · Name the y axis.

Some short y labels read better horizontally. The left margin may need more room.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "Amplitude" norotate
set lmargin 15
plot sin(x)
```



PLT-08-004 · Generated from the complete Bash block

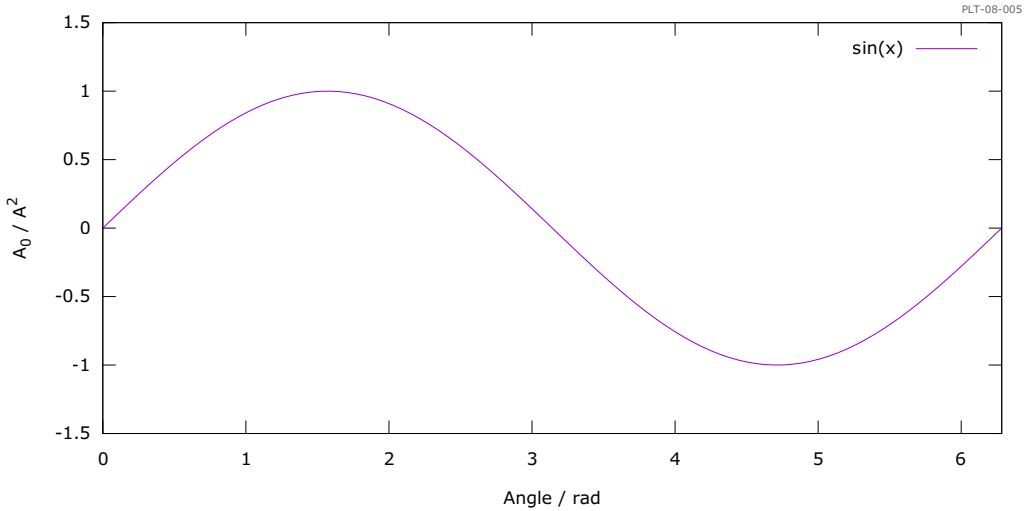
`norotate`	Render the y-axis label horizontally.
`lmargin 15`	Reserve a left margin of 15 character widths.

Use subscripts and superscripts

Compare with PLT-08-002 · Name the y axis.

Enhanced text supports compact mathematical notation in labels.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "A_0 / A^{2}"
plot sin(x)
```



PLT-08-005 · Generated from the complete Bash block

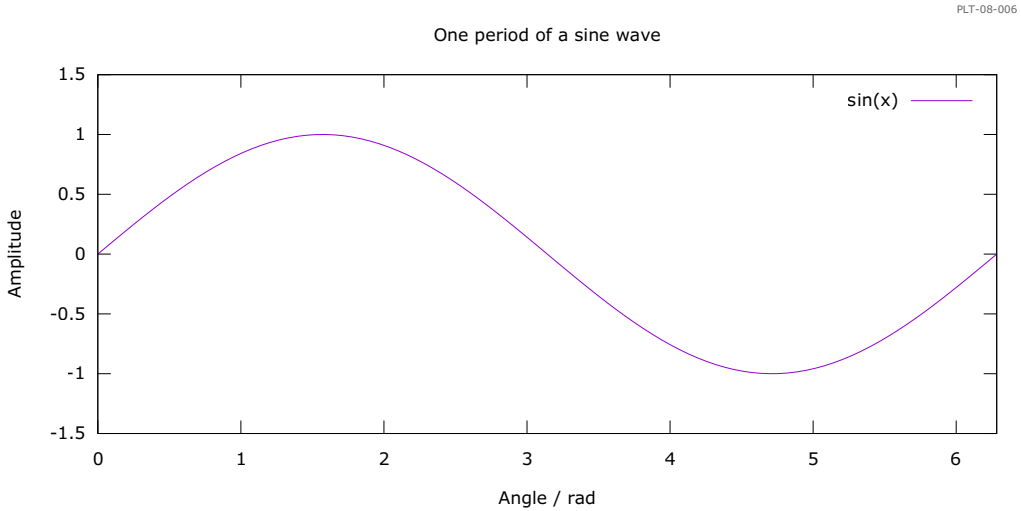
`_0`	Render 0 as a subscript.
`^{2}`	Render 2 as a superscript. Braces group the superscript content.

Add a figure title

Compare with PLT-08-002 · Name the y axis.

A plot title sits above the plotting region and is separate from curve titles in the legend.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "Amplitude"
set title "One period of a sine wave"
plot sin(x)
```



PLT-08-006 · Generated from the complete Bash block

`set title`	Set the overall plot title; plot ... title names a curve instead.

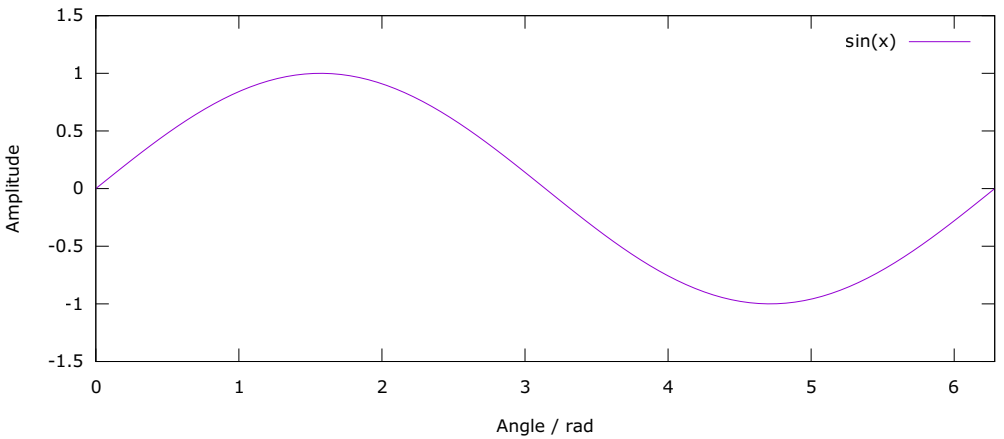
Increase the title font

Compare with PLT-08-006 · Add a figure title.

Change only the title typography.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "Amplitude"
set title "One period of a sine wave" font ",30"
plot sin(x)
```

PLT-08-007



PLT-08-007 · Generated from the complete Bash block

`font ",30"` Retain the font family and set the title’s nominal size to 30.

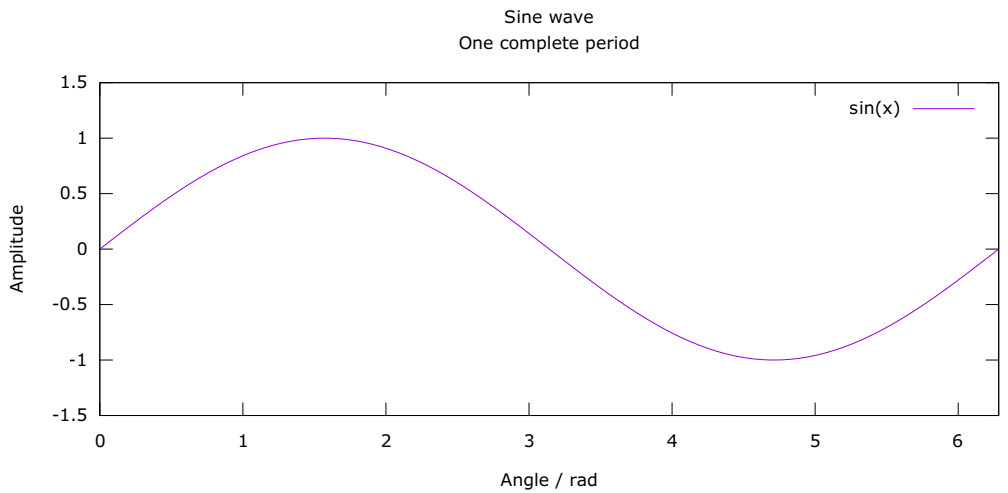
Write a two-line title

Compare with PLT-08-006 · Add a figure title.

A double-quoted string interprets the newline escape.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle / rad"
set ylabel "Amplitude"
set title "Sine wave\nOne complete period"
plot sin(x)
```

PLT-08-008



PLT-08-008 · Generated from the complete Bash block

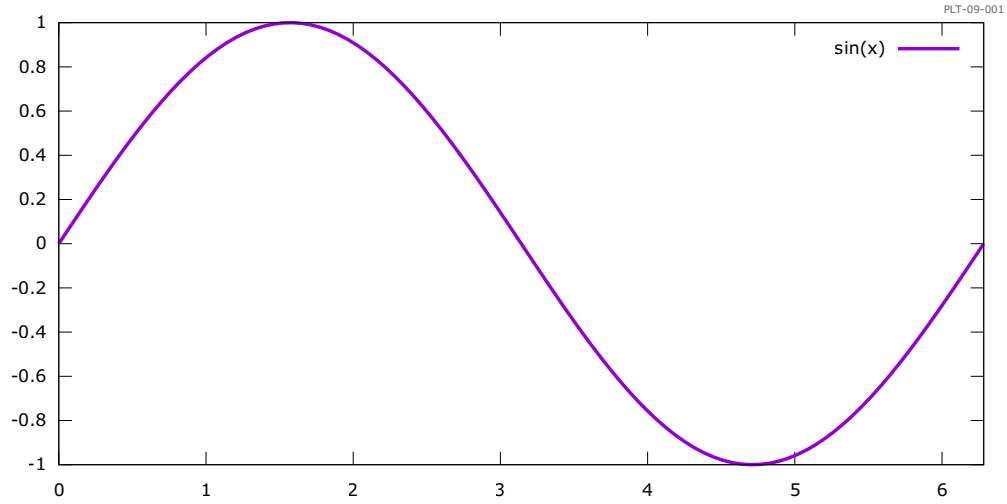
`\n`	A newline escape inside the double-quoted gnuplot string.
`Two title lines`	The automatic top margin expands to accommodate them.

Increase line width

Compare with PLT-01-006 · Choose the horizontal interval.

A larger line width makes the same curve more prominent.

```
set xrange [0:2*pi]
plot sin(x) linewidth 4
```



PLT-09-001 · Generated from the complete Bash block

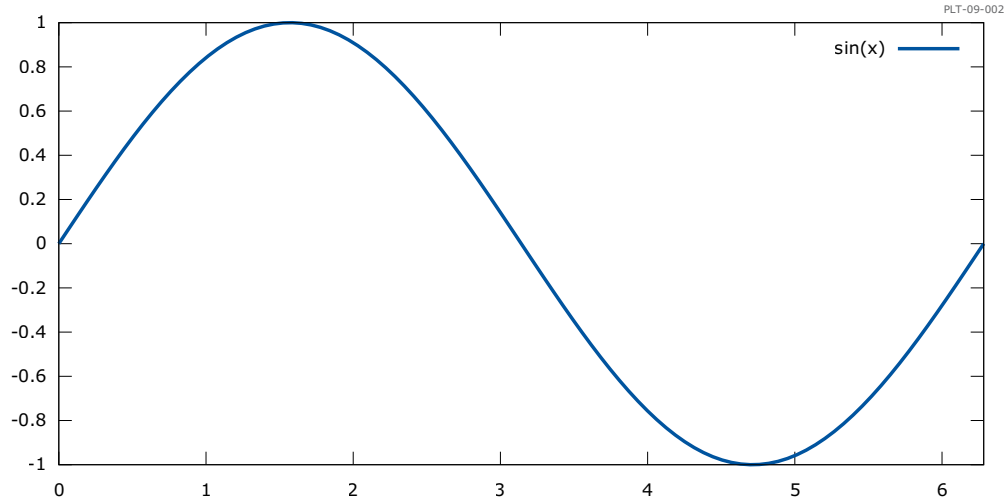
`linewidth 4`	Set the relative line width to 4; terminal rendering determines the physical width.

Choose an explicit line colour

Compare with PLT-09-001 · Increase line width.

The explicit RWTH blue hexadecimal value replaces the default colour. No style file is needed for this colour choice.

```
set xrange [0:2*pi]
plot sin(x) linewidth 4 linecolor rgb "#00549F"
```



PLT-09-002 · Generated from the complete Bash block

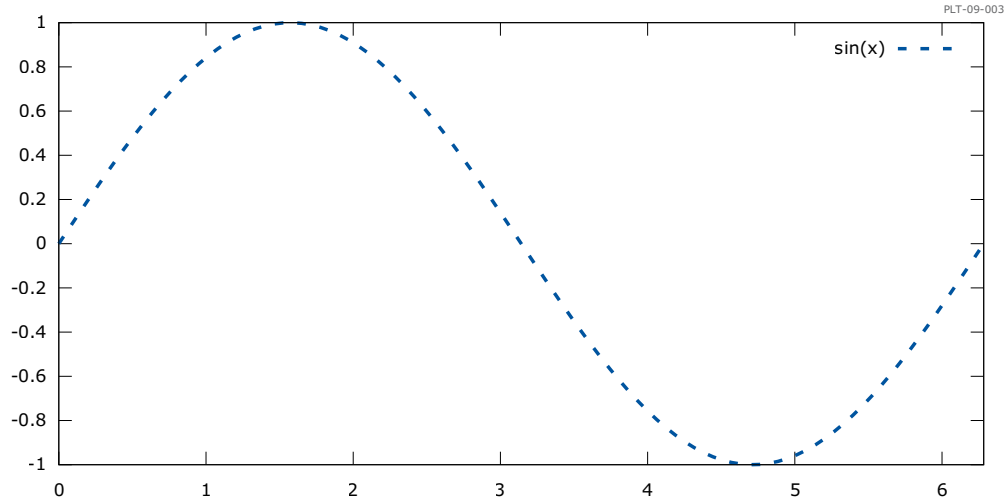
`linecolor`	Set this curve’s colour.
`rgb "#00549F"`	RWTH blue as six hexadecimal digits: two digits per RGB channel.

Use a dashed line

Compare with PLT-09-002 · Choose an explicit line colour.

Dash patterns distinguish curves without relying on colour.

```
set xrange [0:2*pi]
plot sin(x) linewidth 4 linecolor rgb "#00549F" dashtype 2
```



PLT-09-003 · Generated from the complete Bash block

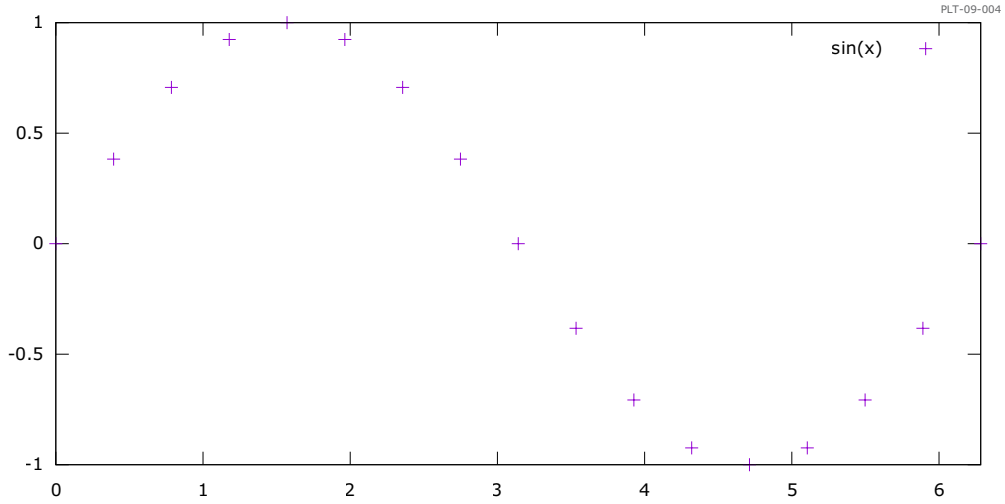
`dashtype 2`	Select predefined dash pattern 2; 1 is solid.

Plot sampled points

Compare with PLT-01-006 · Choose the horizontal interval.

The function is still analytic, but its samples are displayed as points.

```
set xrange [0:2*pi]
set samples 17
plot sin(x) with points
```



PLT-09-004 · Generated from the complete Bash block

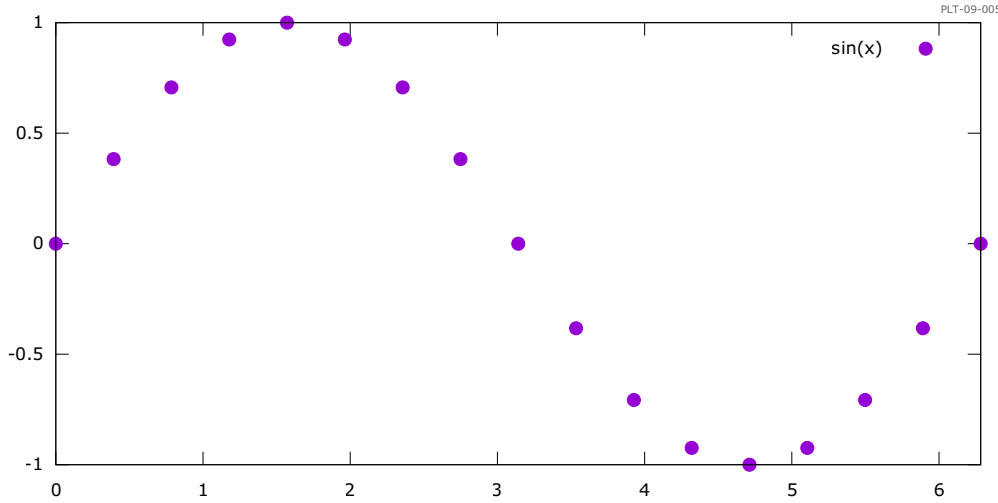
`samples 17`	Sample the expression at 17 x positions.
`with points`	Draw symbols rather than connecting line segments.

Choose the point symbol

Compare with PLT-09-004 · Plot sampled points.

Point type numbers select terminal-supported symbols.

```
set xrange [0:2*pi]
set samples 17
plot sin(x) with points pointtype 7
```



PLT-09-005 · Generated from the complete Bash block

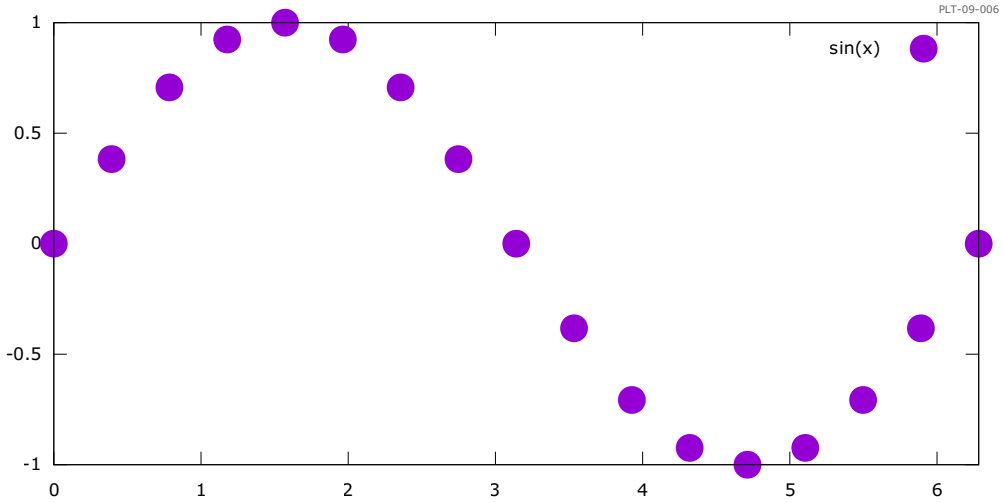
`pointtype 7`	Select symbol 7, a filled circle in the SVG terminal used here.

Increase point size

Compare with PLT-09-005 · Choose the point symbol.

Scale the symbol size without changing point positions.

```
set xrange [0:2*pi]
set samples 17
plot sin(x) with points pointtype 7 pointsize 2
```



PLT-09-006 · Generated from the complete Bash block

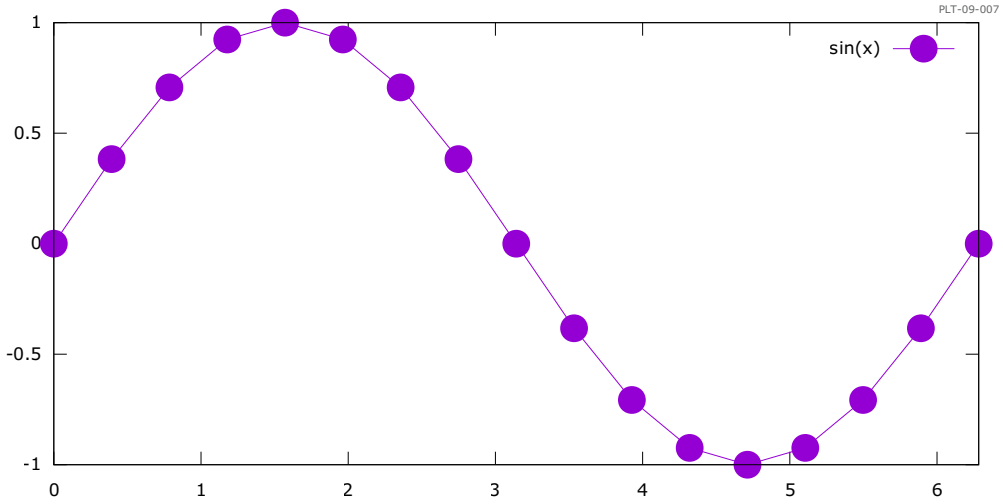
`pointsize 2` Draw symbols at twice the default point-size scale.

Connect the points

Compare with PLT-09-006 · Increase point size.

linespoints combines a connecting line with the selected symbols.

```
set xrange [0:2*pi]
set samples 17
plot sin(x) with linespoints pointtype 7 pointsize 2
```



PLT-09-007 · Generated from the complete Bash block

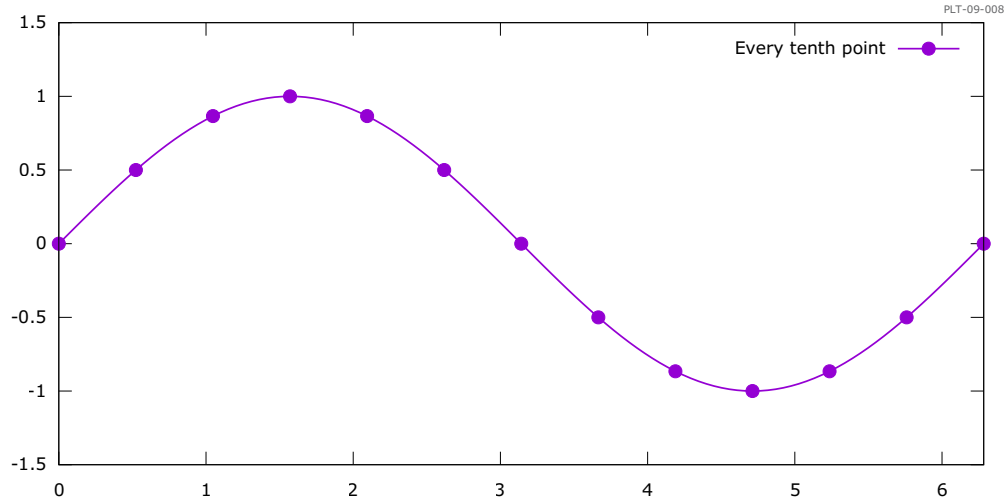
`with linespoints`	Draw both line segments and point symbols.

Show a symbol at every tenth sample

Compare with PLT-09-007 · Connect the points.

Dense line samples make smooth curves, but drawing a marker at every sample creates clutter. Pointinterval changes marker frequency while retaining every line segment.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 121
plot sin(x) with linespoints pt 7 pi 10 lw 2 \
    title "Every tenth point"
```



PLT-09-008 · Generated from the complete Bash block

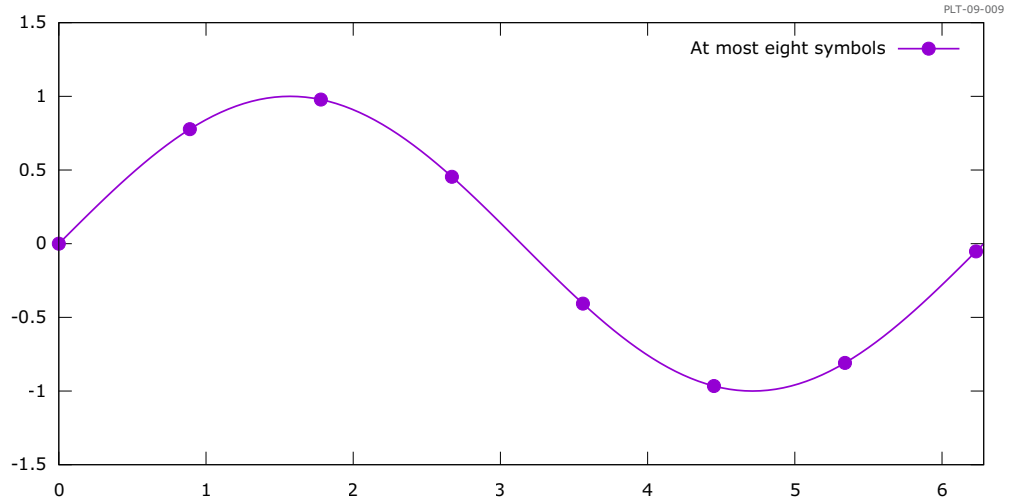
`pi 10`	Draw a symbol at every tenth sample; short for pointinterval.
`samples 121`	Keep all 121 samples for the connected line.
`pi -10`	Also blank a background patch under each selected symbol, on supporting terminals.

Limit the total number of point symbols

Compare with PLT-09-008 · Show a symbol at every tenth sample.

Pointnumber expresses a desired upper limit on markers rather than a fixed sample interval. The line still uses all sampled points.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 121
plot sin(x) with linespoints pt 7 pn 8 lw 2 \
    title "At most eight symbols"
```



PLT-09-009 · Generated from the complete Bash block

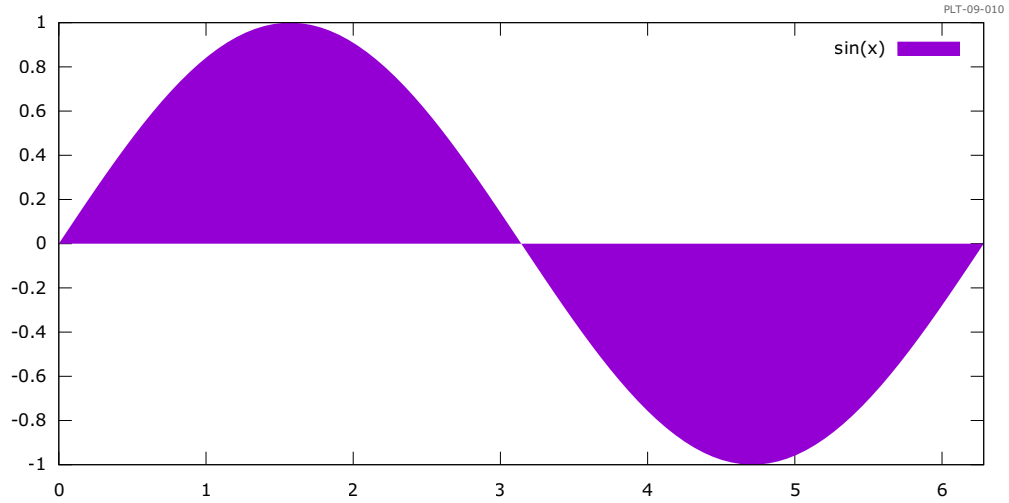
`pn 8`	Limit the number of markers to eight; short for pointnumber.
`pi versus pn`	Pointinterval sets spacing in samples; pointnumber sets a total limit.

Fill down to zero

Compare with PLT-01-006 · Choose the horizontal interval.

filledcurves can close the curve against an axis-parallel reference line.

```
set xrange [0:2*pi]
plot sin(x) with filledcurves y1=0
```



PLT-09-010 · Generated from the complete Bash block

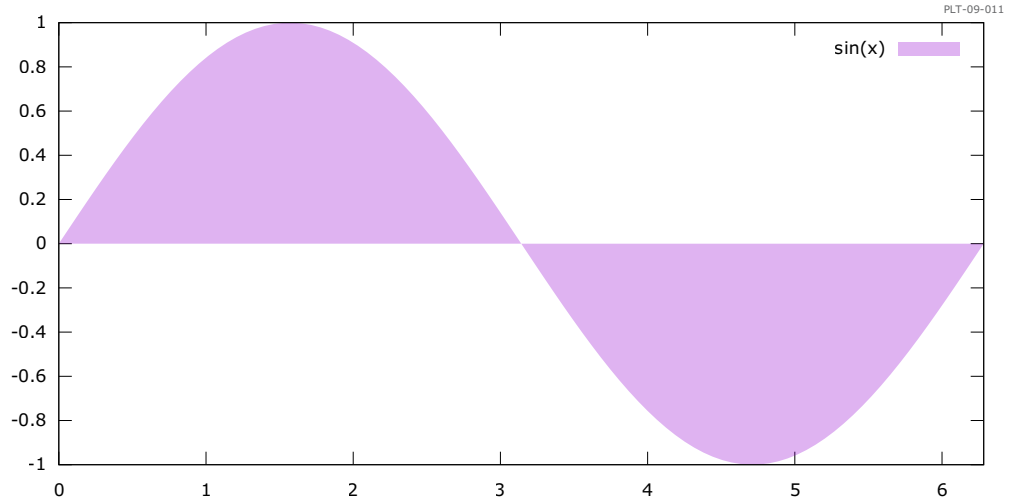
`filledcurves`	Fill a closed area.
`y1=0`	Close it against y=0 on the primary y axis.

Make the fill translucent

Compare with PLT-09-010 · Fill down to zero.

Transparency lets other graphical elements remain visible through the area.

```
set xrange [0:2*pi]
set style fill transparent solid 0.3 noborder
plot sin(x) with filledcurves y1=0
```



PLT-09-011 · Generated from the complete Bash block

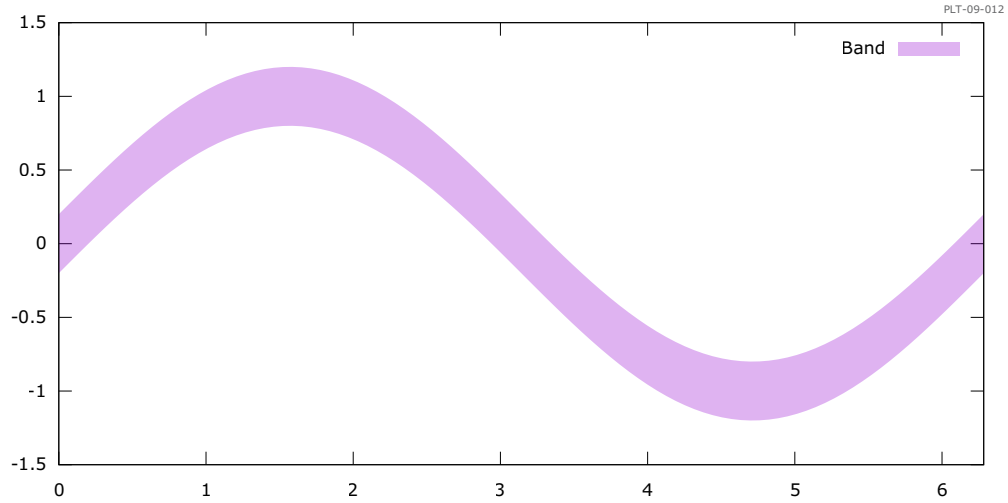
`style fill`	Default fill settings for filled plot styles.
`transparent solid 0.3`	Use a solid fill with opacity 0.3; 0 is transparent, 1 is opaque.
`noborder`	Do not draw an extra outline around the filled area.

Fill between two curves

Compare with PLT-09-011 · Make the fill translucent.

The same idea can shade a corridor between explicit lower and upper values.

```
set xrange [0:2*pi]
set style fill transparent solid 0.3 noborder
plot "+" using 1:(sin($1)-0.2):(sin($1)+0.2) \
    with filledcurves title "Band"
```



PLT-09-012 · Generated from the complete Bash block

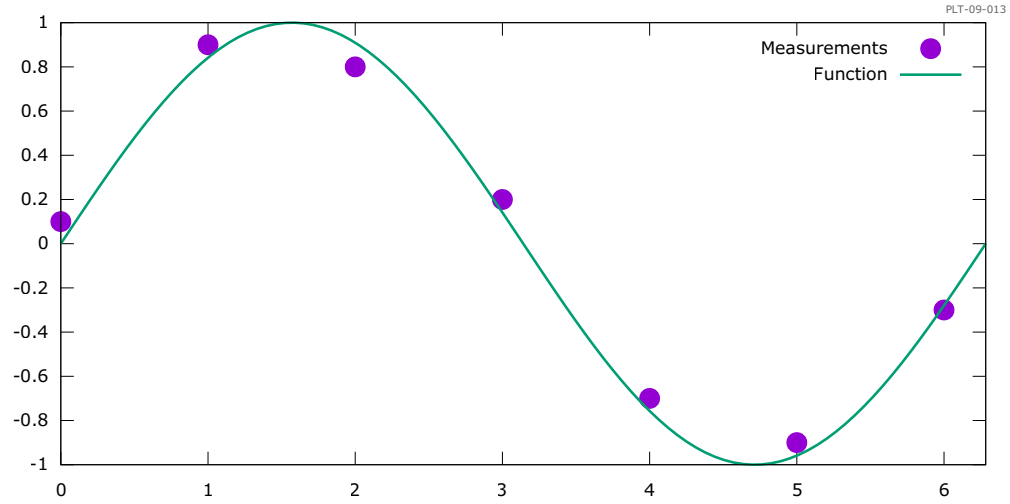
`"+"`	A special data source that supplies sampled x positions.
`using 1:...:...`	Map x, lower bound and upper bound to the fill.
`$1`	Value in the first supplied column.
`-0.2; +0.2`	The lower and upper offsets around sine.

Set different defaults for data and functions

New baseline: the commands below define this example.

The default plot style can be set independently for data and analytic functions. Explicit with clauses on individual elements take precedence.

```
$D << EOD
0 0.1
1 0.9
2 0.8
3 0.2
4 -0.7
5 -0.9
6 -0.3
EOD
set xrange [0:2*pi]
set style data points
set style function lines
plot $D using 1:2 pt 7 ps 1.5 title "Measurements", \
    sin(x) lw 3 title "Function"
```



PLT-09-013 · Generated from the complete Bash block

`style data points`	Choose points when a data plot omits its with clause.
`style function lines`	Choose lines when an analytic function omits its with clause.
`with ...`	Overrides the default for that plot element.

Quick reference: line types

New baseline: the commands below define this example.

Compare the first eight default line types in one plot. A line type selects a reusable style, including colour; it is not the dash-pattern number. These swatches show the defaults of this fresh gnuplot process. Defaults can be redefined, and rendering depends on the terminal.

```
unset key
unset border
unset xtics
set ytics nomirror scale 0
set xrange [0:1]
set lmargin 14
set rmargin 3
set yrange [8.7:0.3]
set ytics ()
set for [i=1:8] ytics add (sprintf("lt %d",i) i)
plot for [i=1:8] i with lines linetype i linewidth 3
```



PLT-09-014 · Generated from the complete Bash block

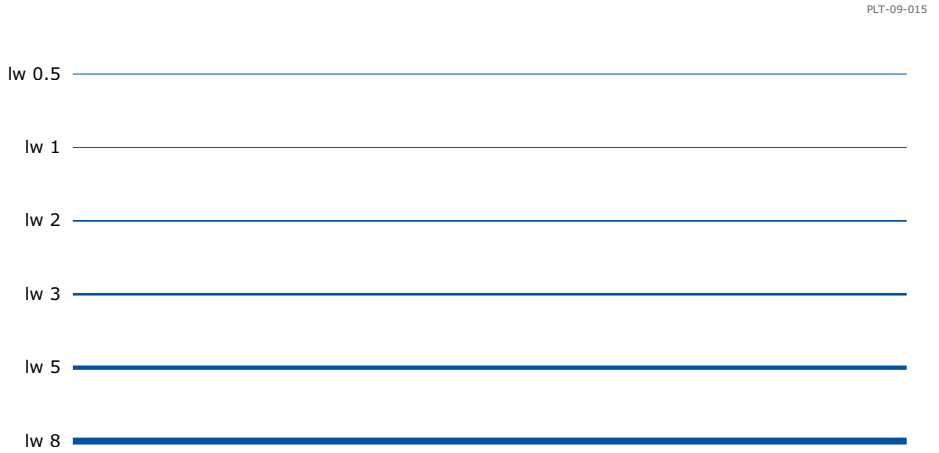
`for [i=1:8]`	Repeat the plot element for each style number; y=i places each sample on its own row.
`linetype i (lt i)`	Select line style i, including its default colour and other stored properties.
`sprintf("lt %d",i)`	Label each row with the exact style number.
`linewidth 3`	Hold stroke width constant so the line types are easy to compare.

Quick reference: line widths

New baseline: the commands below define this example.

Compare six stroke widths at the same colour and dash pattern. Widths are relative multipliers, not a promise of an exact printed thickness.

```
unset key
unset border
unset xtics
set ytics nomirror scale 0
set xrange [0:1]
set lmargin 14
set rmargin 3
array W[6] = [0.5,1,2,3,5,8]
set yrange [6.7:0.3]
set ytics ()
set for [i=1:6] ytics add (sprintf("lw %g",W[i]) i)
plot for [i=1:6] i with lines linewidth W[i] \
    linecolor rgb "#00549F"
```



PLT-09-015 · Generated from the complete Bash block

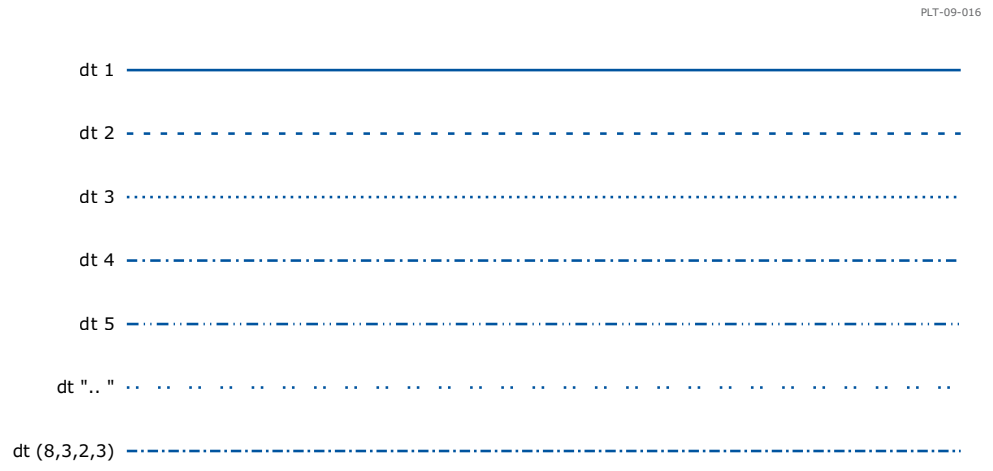
`array W[6]`	Store the six widths in a one-based array.
`linewidth W[i] (lw W[i])`	Use the current array value as the stroke-width multiplier.
`%g`	Format each label without unnecessary trailing zeroes.

Quick reference: dash patterns

New baseline: the commands below define this example.

Compare five numbered dash patterns and two custom definitions while keeping colour and width fixed. Dash patterns are independent of line types. The numbered patterns shown here are those of the SVG terminal.

```
unset key
unset border
unset xtics
set ytics nomirror scale 0
set xrange [0:1]
set lmargin 14
set rmargin 3
set yrange [7.7:0.3]
set ytics ()
set for [i=1:5] ytics add (sprintf("dt %d",i) i)
set ytics add ('dt ".. "' 6, 'dt (8,3,2,3)' 7)
set dashtype 6 ".. "
set dashtype 7 (8,3,2,3)
plot for [i=1:7] i with lines dashtype i \
    linewidth 3 linecolor rgb "#00549F"
```



PLT-09-016 · Generated from the complete Bash block

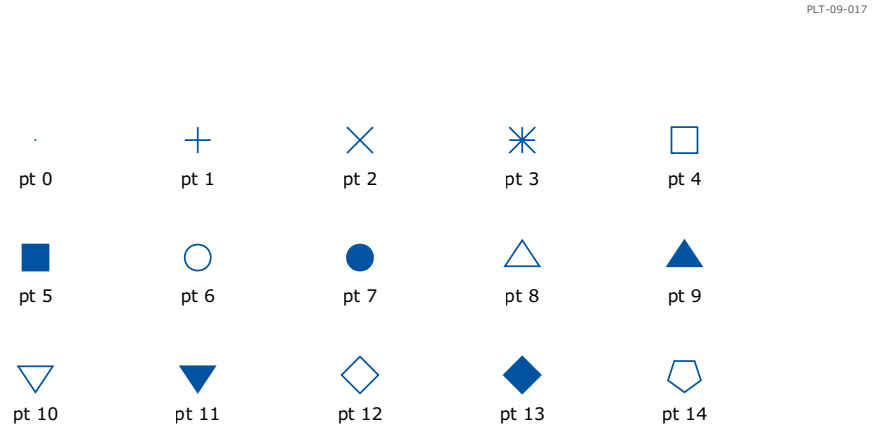
`dashtype i (dt i)`	Select a dash pattern independently of the line colour; type 1 is solid here.
`set dashtype 6 ".. "`	Define pattern 6 from two dots and an extra space.
`(8,3,2,3)`	Alternate stroke and gap lengths: long stroke, gap, short stroke, gap.
`linecolor rgb (lc rgb)`	Use the same blue for every sample.

Quick reference: point symbols

New baseline: the commands below define this example.

Choose a point symbol from this labelled contact sheet. Every symbol uses the same colour and size. Symbol shapes and the available range can vary by terminal; these are SVG point types 0 through 14.

```
unset key
unset border
unset tics
set xrange [0:6]
set yrange [0:4]
$P << EOD
0 0
EOD
set for [i=0:14] label i+1 sprintf("pt %d",i) \
    at 1+i%5,3-int(i/5) center offset 0,-1.5
plot for [i=0:14] $P using (1+i%5):(3-int(i/5)) \
    with points pointtype i pointsize 2 \
    linecolor rgb "#00549F"
```



PLT-09-017 · Generated from the complete Bash block

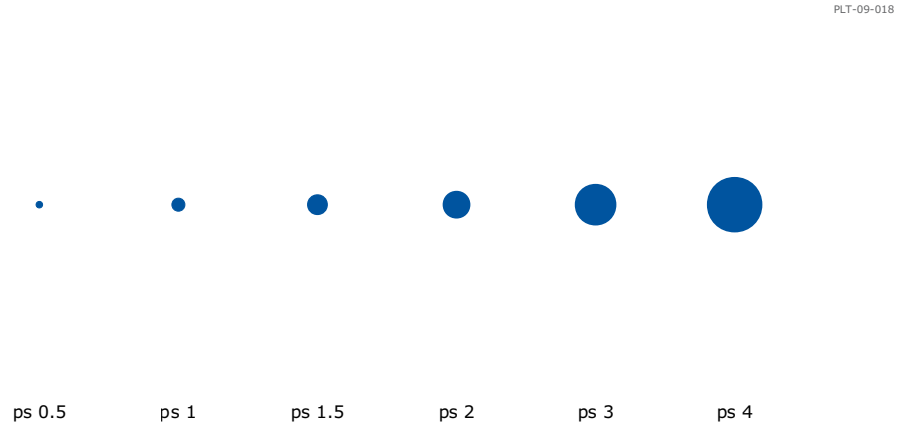
`pointtype i (pt i)`	Select the symbol by its integer number; type 0 is a small dot.
`pointsize 2 (ps 2)`	Use the same relative marker size for every symbol.
`i%5; int(i/5)`	Arrange the symbols in five columns and three rows.
`$P; using (...)`	Reuse one inline data row and calculate each symbol position.

Quick reference: point sizes

New baseline: the commands below define this example.

Compare six sizes of the same filled-circle marker. Point size is a relative multiplier, not a data-space radius. Its visual size depends on the terminal.

```
unset key
unset border
unset ytics
set xtics nomirror scale 0
set xrange [0:7]
set yrange [0:2]
set bmargin 5
array S[6] = [0.5,1,1.5,2,3,4]
set xtics ()
set for [i=1:6] xtics add (sprintf("ps %g",S[i]) i)
$P << EOD
0 0
EOD
plot for [i=1:6] $P using (i):(1) \
    with points pointtype 7 pointsize S[i] \
    linecolor rgb "#00549F"
```



PLT-09-018 · Generated from the complete Bash block

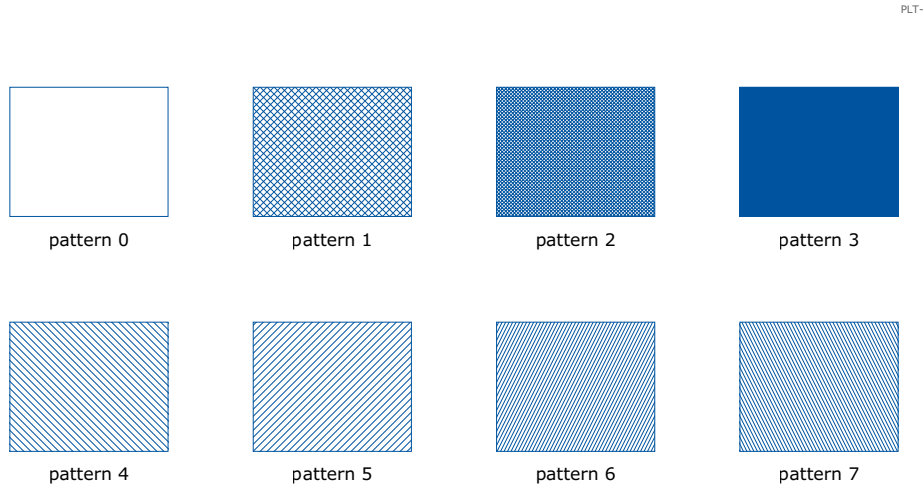
`pointtype 7`	Use the filled-circle symbol of the SVG terminal.
`pointsize S[i] (ps S[i])`	Select each relative size from the array; positions and colour stay fixed.
`set xtics add`	Put a size label below each sample without drawing tick strokes.

Quick reference: fill patterns

New baseline: the commands below define this example.

Compare eight numbered hatch fills in a compact two-row sheet. These rectangles use the same fillstyle syntax as filled plot elements such as boxes. Available patterns depend on the terminal and may repeat beyond its pattern set.

```
unset key
unset border
unset tics
set xrange [0:8]
set yrange [0:4]
x(i) = 0.35 + 2*(i%4)
y(i) = 2.35 - 2*int(i/4)
set for [i=0:7] object i+1 rectangle \
    from x(i),y(i) to x(i)+1.3,y(i)+1.1 \
    fillcolor rgb "#00549F" fillstyle pattern i \
    border linecolor rgb "#00549F"
set for [i=0:7] label i+1 sprintf("pattern %d",i) \
    at x(i)+0.65,y(i)-0.2 center
plot NaN notitle
```



PLT-09-019 · Generated from the complete Bash block

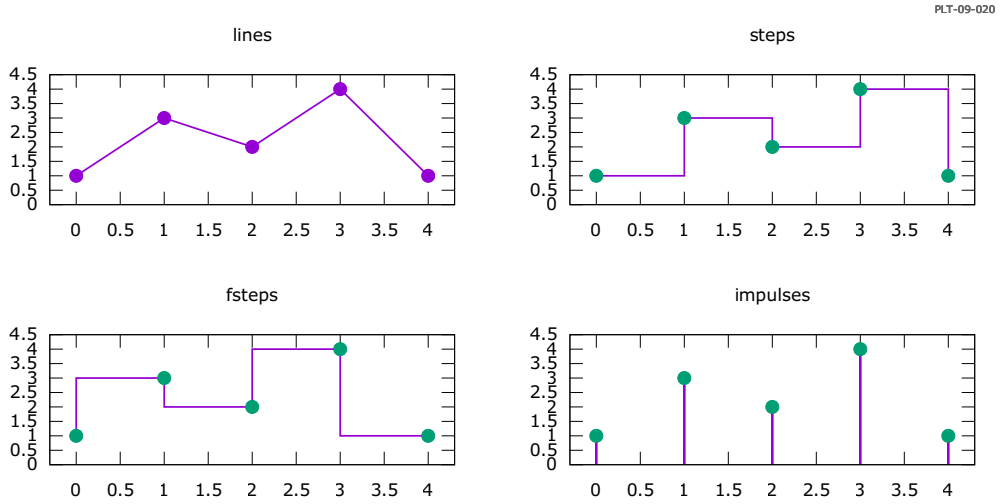
`fillstyle pattern i (fs pattern i)`	Fill each rectangle with terminal pattern i.
`fillcolor rgb (fc rgb)`	Choose the colour used to draw the pattern.
`border linecolor rgb`	Outline each rectangle so even an empty pattern remains visible.
`plot NaN`	Draw the configured objects and labels without a data curve; both ranges are explicit.

Quick reference: lines, steps and impulses

New baseline: the commands below define this example.

The same observations produce different connecting conventions. Vertical impulses emphasize the individual values rather than implying a continuous path.

```
$D << EOD
0 1
1 3
2 2
3 4
4 1
EOD
set multiplot layout 2,2
unset key
set xrange [-0.3:4.3]
set yrange [0:4.5]
set title "lines"
plot $D with linespoints pt 7 lw 2
set title "steps"
plot $D with steps lw 2, $D with points pt 7
set title "fsteps"
plot $D with fsteps lw 2, $D with points pt 7
set title "impulses"
plot $D with impulses lw 3, $D with points pt 7
unset multiplot
```



PLT-09-020 · Generated from the complete Bash block

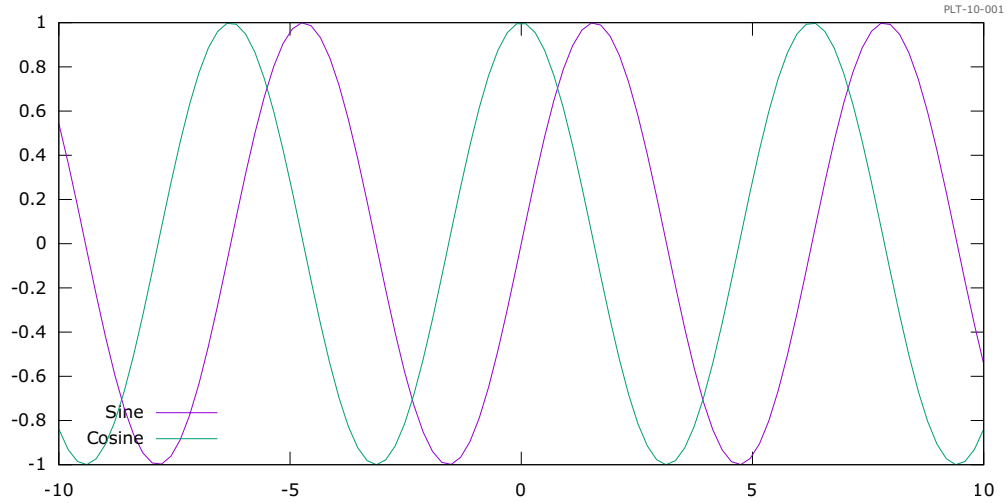
`steps`	Hold the previous value until reaching the next x position.
`fsteps`	Change to the next value at the previous x position.
`impulses`	Draw a vertical stroke from y=0 to each observation.
`linespoints`	Connect adjacent observations with straight segments.

Move the legend to the lower left

Compare with PLT-01-004 · Name the curves.

Set horizontal and vertical placement explicitly.

```
set key left bottom
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-001 · Generated from the complete Bash block

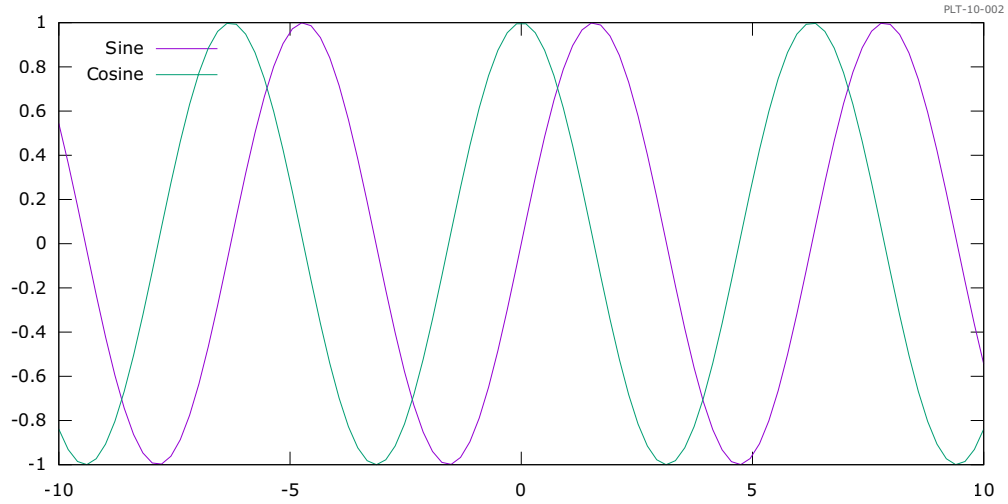
`key`	The legend.
`left bottom`	Horizontal and vertical anchors inside the plot.

Move the legend to the upper left

Compare with PLT-10-001 · Move the legend to the lower left.

Only the vertical anchor changes.

```
set key left top
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-002 · Generated from the complete Bash block

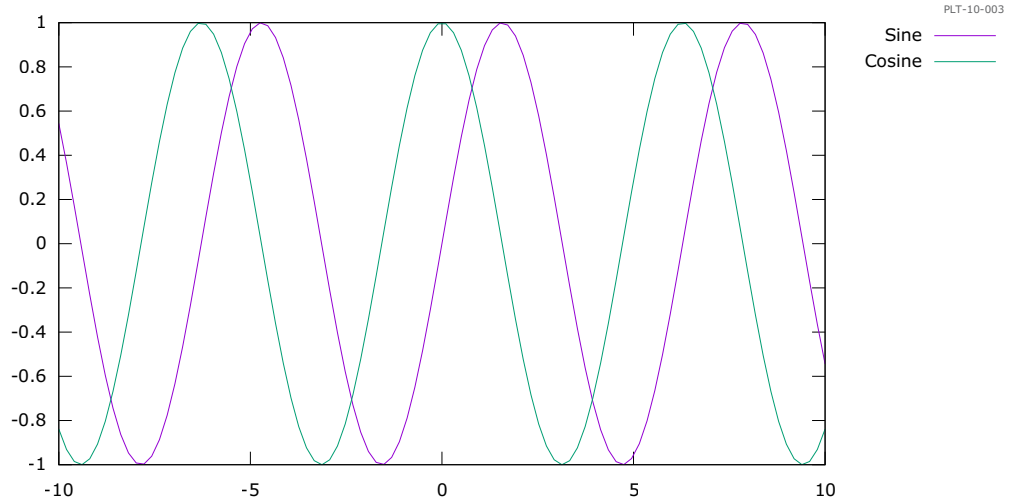
`top`	Use the top edge instead of the bottom edge.

Move the legend outside

Compare with PLT-10-002 · Move the legend to the upper left.

Placing a legend outside can free the data region, but consumes canvas space.

```
set key outside right top
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-003 · Generated from the complete Bash block

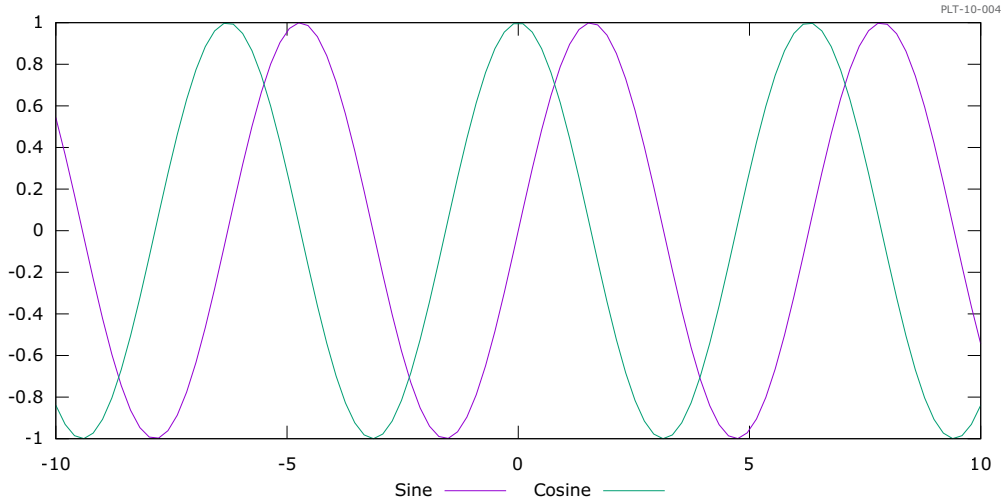
`outside`	Place the legend outside the plot borders.
`right top`	Use the right-hand side and align it to the top.

Place a horizontal legend below

Compare with PLT-01-004 · Name the curves.

Horizontal arrangement uses columns instead of a vertical stack.

```
set key outside bottom center horizontal
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-004 · Generated from the complete Bash block

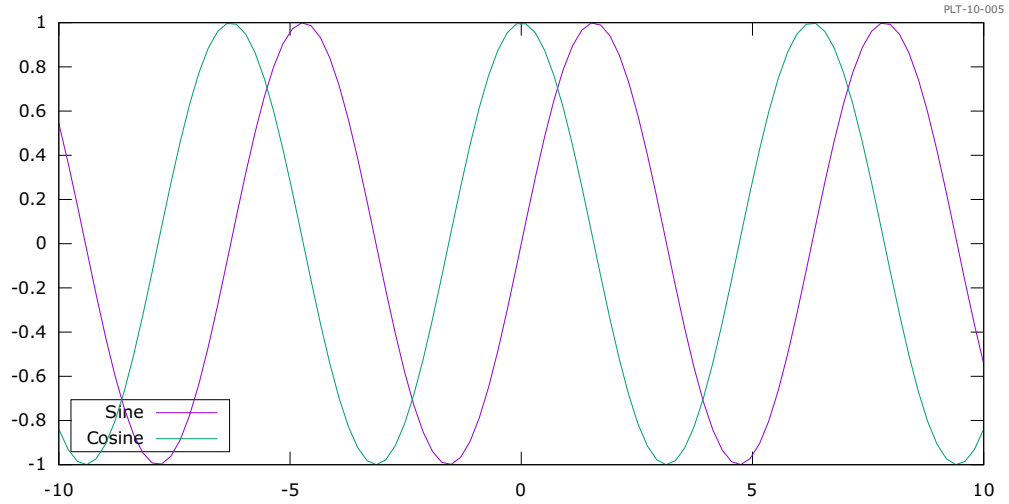
`outside bottom center`	Centre the legend below the plot.
`horizontal`	Arrange entries across a row when space allows.

Draw a legend box

Compare with PLT-10-001 · Move the legend to the lower left.

A border helps separate a legend from surrounding data.

```
set key left bottom
set key box
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-005 · Generated from the complete Bash block

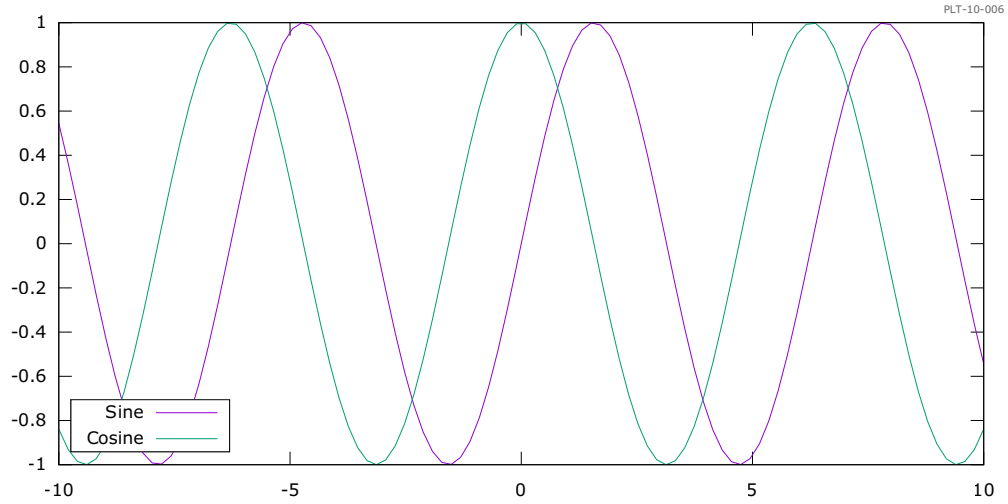
box	Draw a box around the existing legend position.

Give the legend an opaque background

Compare with PLT-10-005 · Draw a legend box.

The legend now covers lines behind it. Compare the line passing through its area.

```
set key left bottom
set key box
set key opaque
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-006 · Generated from the complete Bash block

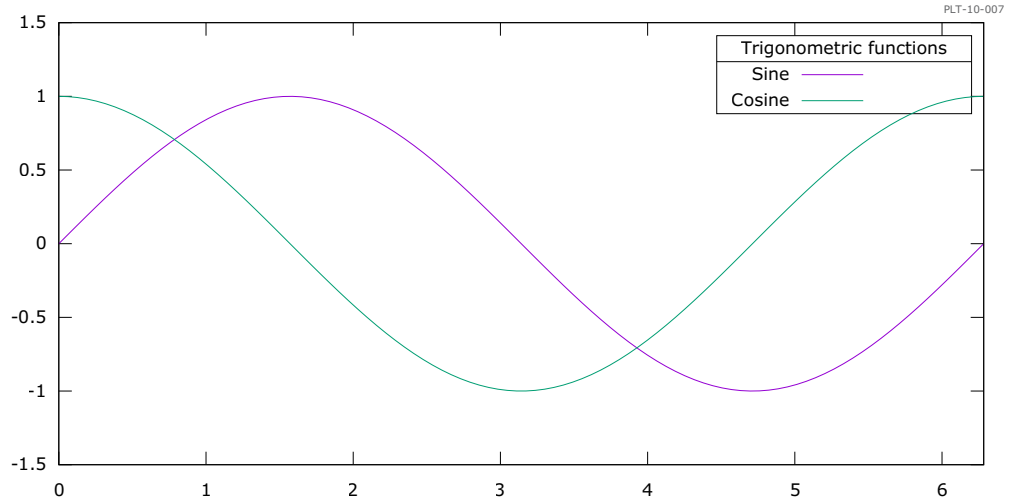
opaque	Fill behind the legend with the background colour.

Give the legend a title

Compare with PLT-10-005 · Draw a legend box.

A legend title describes the collection of entries. It is separate from the figure title and from the titles of individual curves.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set key top right title "Trigonometric functions" box
plot sin(x) title "Sine", cos(x) title "Cosine"
```



PLT-10-007 · Generated from the complete Bash block

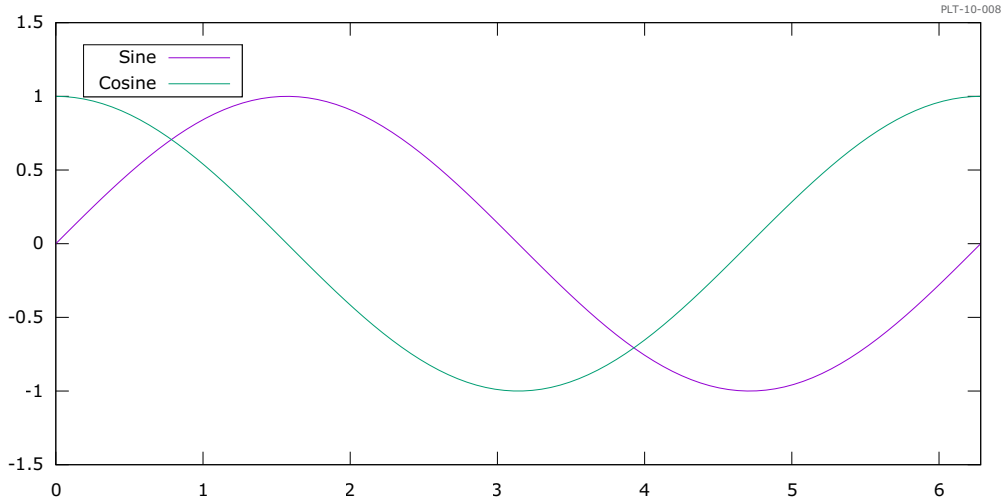
`key title`	Heading inside the legend.
`plot ... title`	Label for one curve in that legend.
`box`	Draw a frame around the legend and its title.

Anchor the legend with graph coordinates

Compare with PLT-10-007 · Give the legend a title.

Explicit placement is useful when the named corner positions are too coarse. Graph coordinates track the plot region as its margins change.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set key at graph 0.03,0.95 left top opaque box
plot sin(x) title "Sine", cos(x) title "Cosine"
```



PLT-10-008 · Generated from the complete Bash block

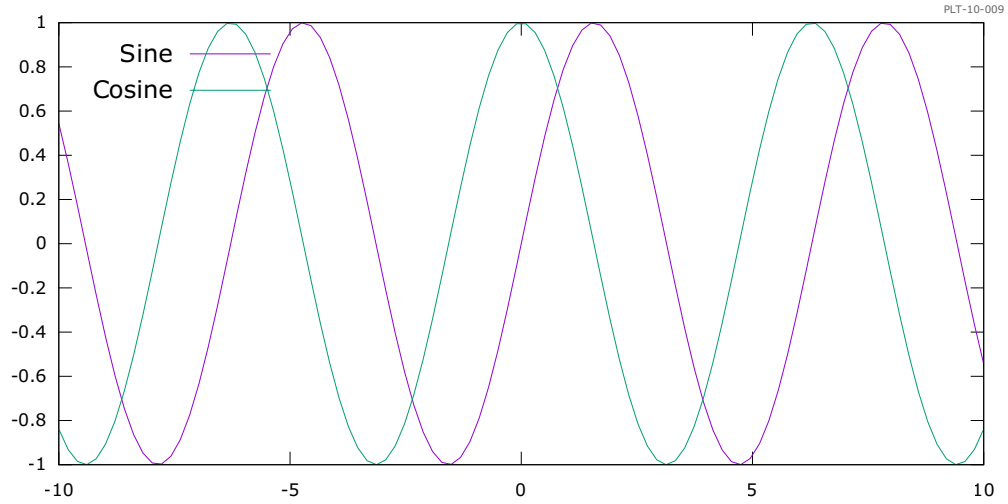
`at graph 0.03,0.95`	Place the legend anchor near the upper-left of the plot region.
`left top`	Align the legend rectangle relative to that anchor.
`opaque`	Cover plotted data underneath the legend.

Enlarge legend text

Compare with PLT-10-002 · Move the legend to the upper left.

Legend text has its own font setting.

```
set key left top
set key font ",28"
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-009 · Generated from the complete Bash block

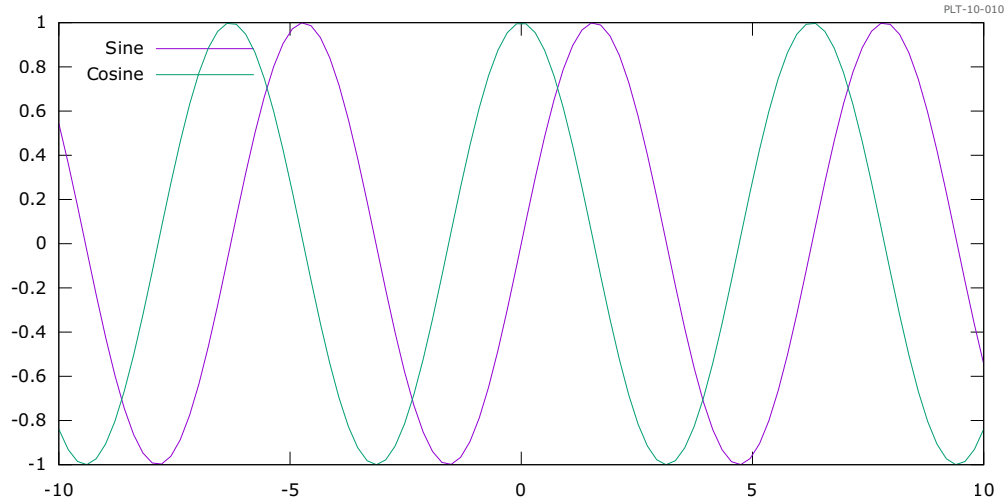
`font ",28"` Keep the family; use nominal size 28 for legend text.

Lengthen legend samples

Compare with PLT-10-002 · Move the legend to the upper left.

The sample strokes can be adjusted independently of the text.

```
set key left top
set key samplen 7
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-010 · Generated from the complete Bash block

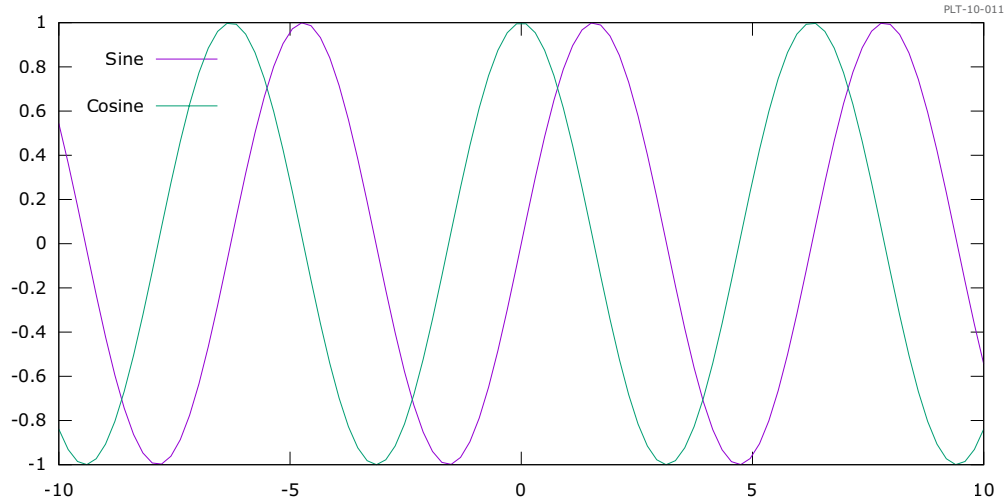
`samplen 7`	Sample length, approximately in character widths; the default is shorter.

Increase spacing between entries

Compare with PLT-10-002 · Move the legend to the upper left.

More vertical spacing separates the legend rows.

```
set key left top
set key spacing 1.8
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-011 · Generated from the complete Bash block

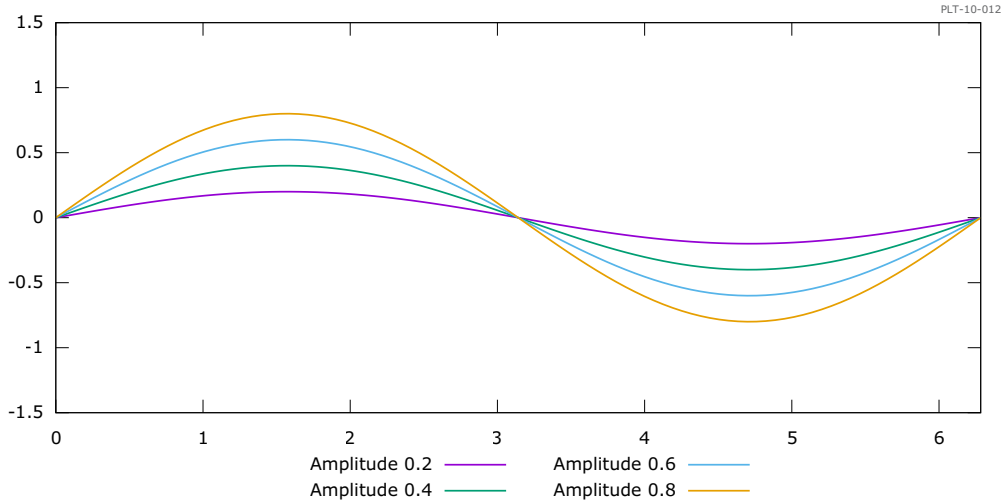
`spacing 1.8` Multiply the normal row spacing by 1.8.

Arrange the legend in two columns

New baseline: the commands below define this example.

A horizontal legend fills across its rows. Limit the column count to keep several entries compact below the plot.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set key outside below horizontal maxcols 2
plot for [i=1:4] 0.2*i*sin(x) \
    title sprintf("Amplitude %.1f",0.2*i) lw 2
```



PLT-10-012 · Generated from the complete Bash block

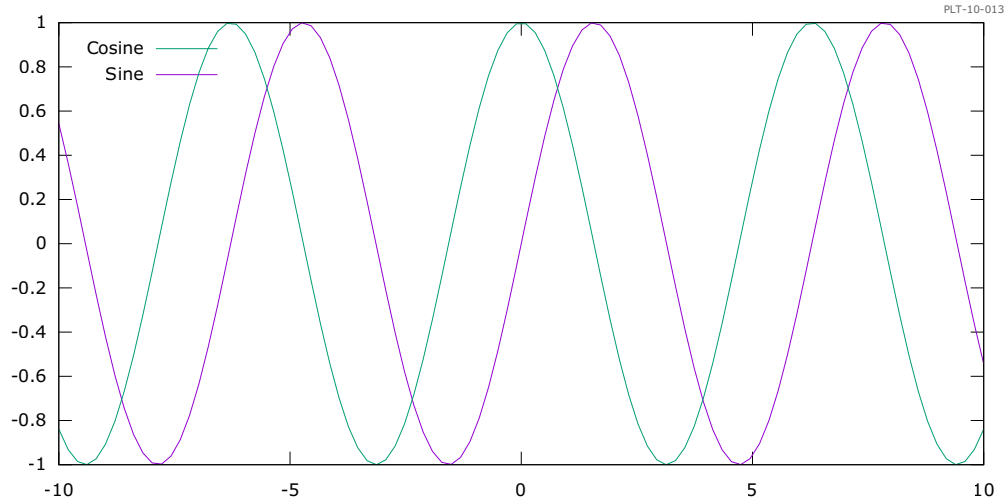
`horizontal maxcols 2`	Lay out entries horizontally using at most two columns.
`outside below`	Reserve legend space below the plot.
`maxrows`	The complementary limit for the number of rows.

Reverse legend entry order

Compare with PLT-10-002 · Move the legend to the upper left.

Legend order can change without changing which curve is drawn first.

```
set key left top
set key invert
plot sin(x) title 'Sine', cos(x) title 'Cosine'
```



PLT-10-013 · Generated from the complete Bash block

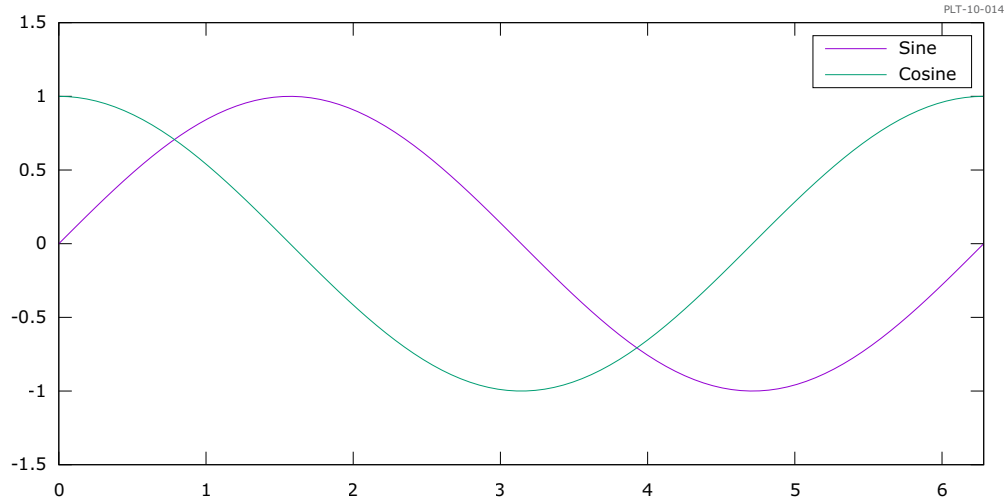
`invert`	Reverse the vertical order of entries; it does not reverse x or the data.

Reverse sample placement without reordering entries

Compare with PLT-10-005 · Draw a legend box.

Reverse exchanges the text and sample positions within each legend entry. Invert changes the order of the entries, as shown in PLT-10-013.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set key top right reverse Left box
plot sin(x) title "Sine", cos(x) title "Cosine"
```



PLT-10-014 · Generated from the complete Bash block

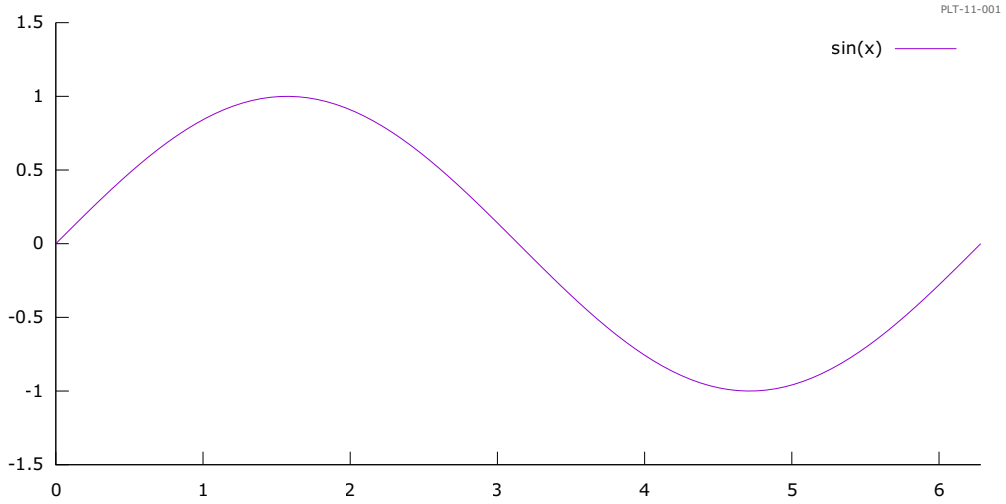
`reverse`	Put the sample on the opposite side of each label.
`Left`	Left-align the legend text; capitalization matters here.
`invert`	A separate option that reverses the order of the entries.

Keep only left and bottom borders

Compare with PLT-01-007 · Leave room above and below.

The border value is a sum of side flags. Matching tick settings prevent marks on missing sides.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set border 3
set tics nomirror
plot sin(x)
```



PLT-11-001 · Generated from the complete Bash block

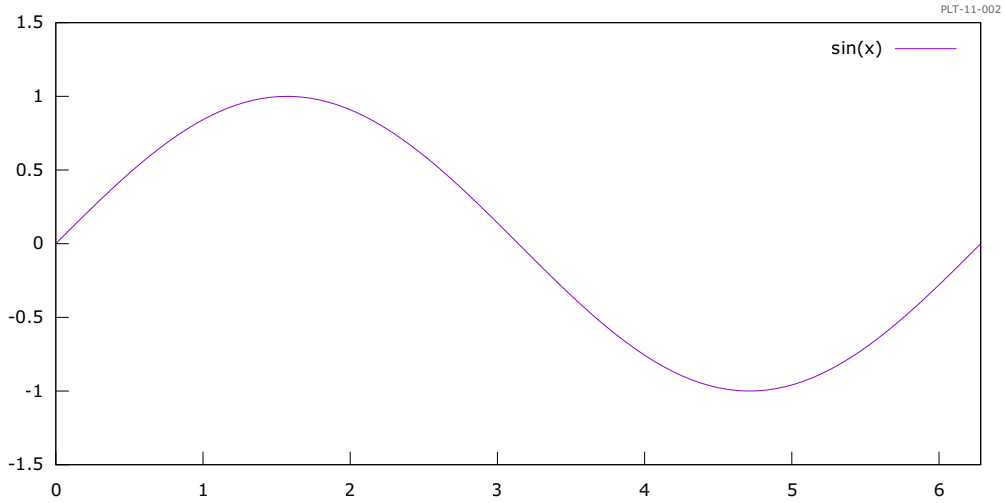
`border 3`	1 (bottom) + 2 (left). In 2D, 4 is top and 8 is right.
`tics nomirror`	Remove tick copies on the omitted sides.

Restore all four borders

Compare with PLT-11-001 · Keep only left and bottom borders.

Change only the border bit mask.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set tics nomirror
set border 15
plot sin(x)
```



PLT-11-002 · Generated from the complete Bash block

15

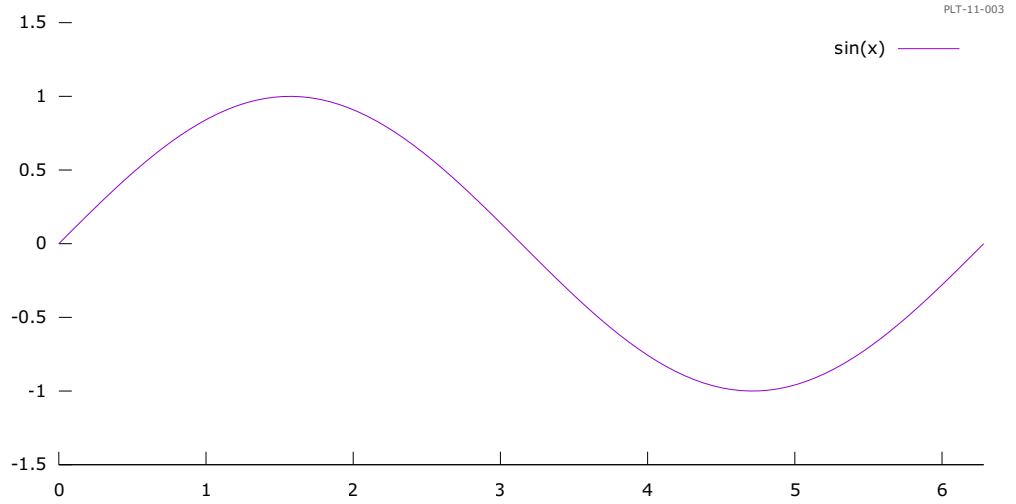
1 + 2 + 4 + 8: draw all four sides. Tick mirroring remains disabled.

Keep just the bottom border

Compare with PLT-11-001 · Keep only left and bottom borders.

The border mask selects individual sides independently. Keeping just the bottom side creates an open frame; the existing y ticks remain.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set border 1
set tics nomirror
plot sin(x)
```



PLT-11-003 · Generated from the complete Bash block

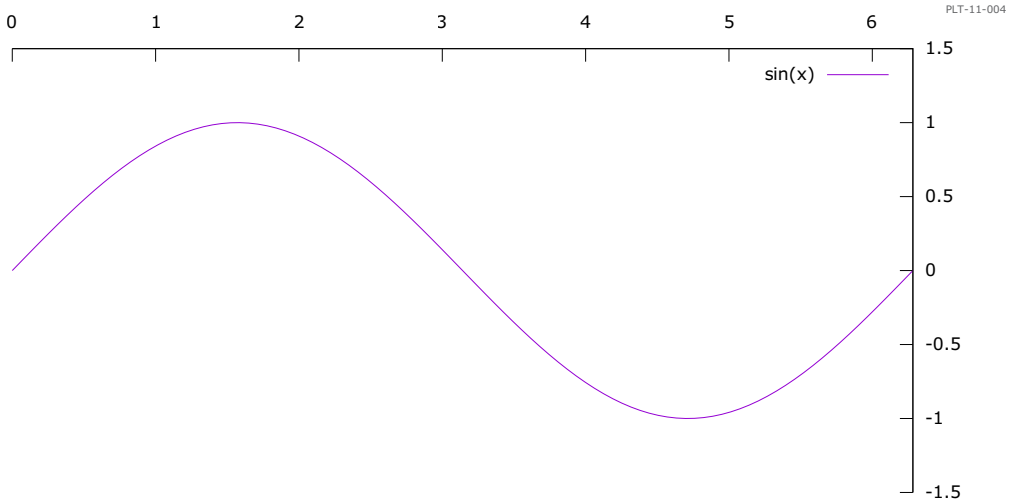
`border 1`	Draw the bottom side only. In 2D, bottom=1, left=2, top=4 and right=8; add values to combine sides.

Put the frame and ticks at the top and right

Compare with PLT-11-001 · Keep only left and bottom borders.

Move the coordinate frame to the opposite sides. Match the secondary ranges to the primary axes so that the new labels describe the same curve.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set border 12
unset xtics
unset ytics
set x2range [0:2*pi]
set y2range [-1.5:1.5]
set x2tics nomirror
set y2tics nomirror
plot sin(x)
```



PLT-11-004 · Generated from the complete Bash block

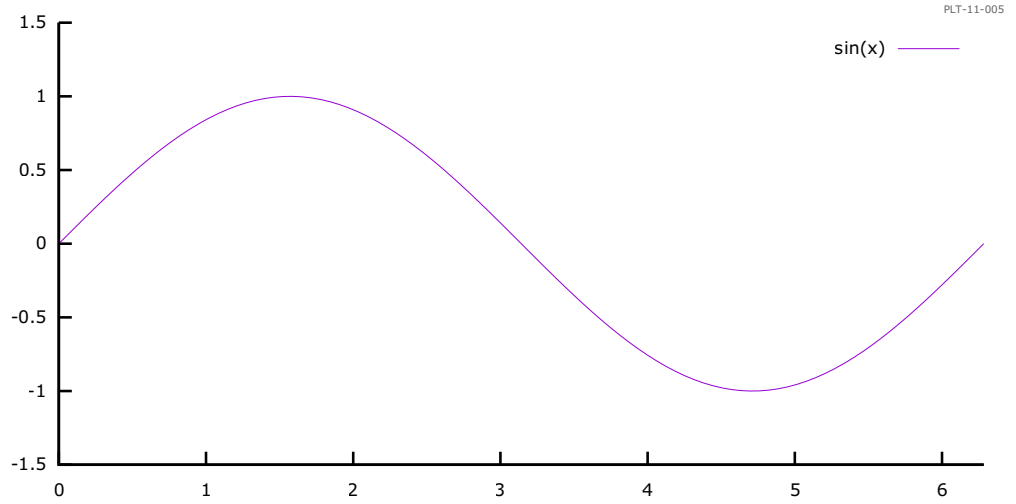
`border 12`	Draw top (4) and right (8) borders.
`x2range; y2range`	Match the secondary axes to the primary x and y ranges.
`x2tics; y2tics`	Show ticks on top and right; disable primary ticks to remove bottom and left labels.

Thicken the remaining borders

Compare with PLT-11-001 · Keep only left and bottom borders.

The border has an independent line style.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set tics nomirror
set border 3 linewidth 3
plot sin(x)
```



PLT-11-005 · Generated from the complete Bash block

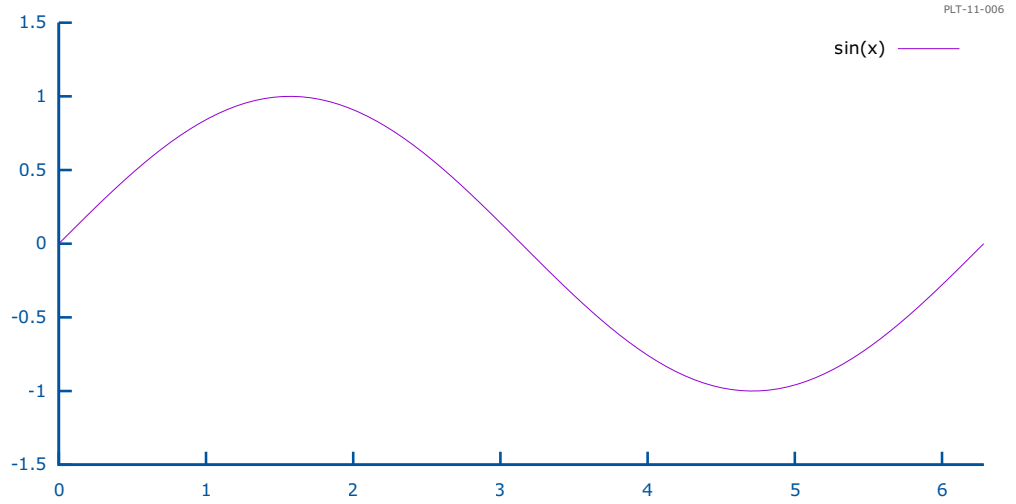
`linewidth 3` Relative width of the chosen border sides.

Colour the border and tick strokes

Compare with PLT-11-005 · Thicken the remaining borders.

Apply a blue line colour to the left and bottom frame. Tick strokes inherit the border style; tick-label text can be styled separately.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set tics nomirror
set border 3 linewidth 3 linecolor rgb "#00549F"
plot sin(x)
```



PLT-11-006 · Generated from the complete Bash block

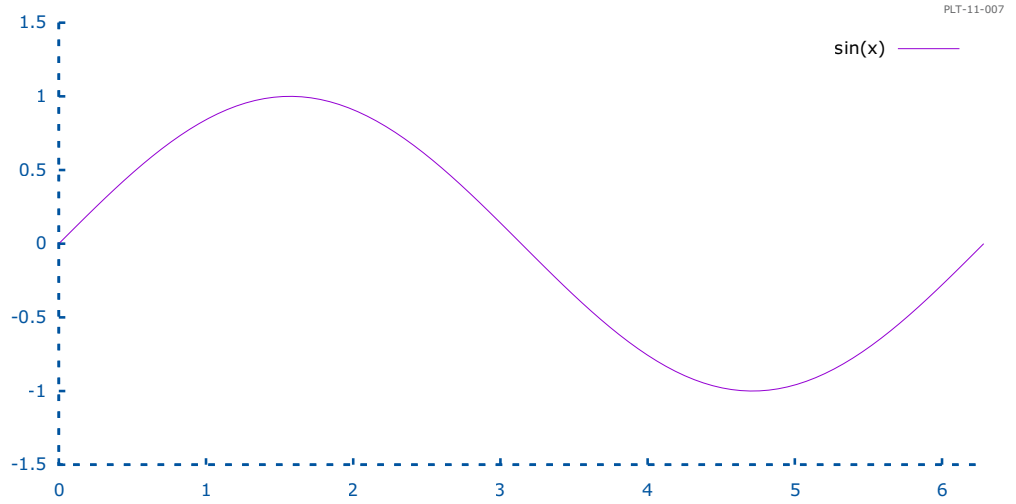
`linecolor rgb "#00549F"`	Use an explicit blue RGB colour for the border line style.

Use a dashed border

Compare with PLT-11-006 · Colour the border and tick strokes.

Give the existing blue border a dashed stroke while preserving its width and selected sides.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set tics nomirror
set border 3 linewidth 3 linecolor rgb "#00549F" dashtype 2
plot sin(x)
```



PLT-11-007 · Generated from the complete Bash block

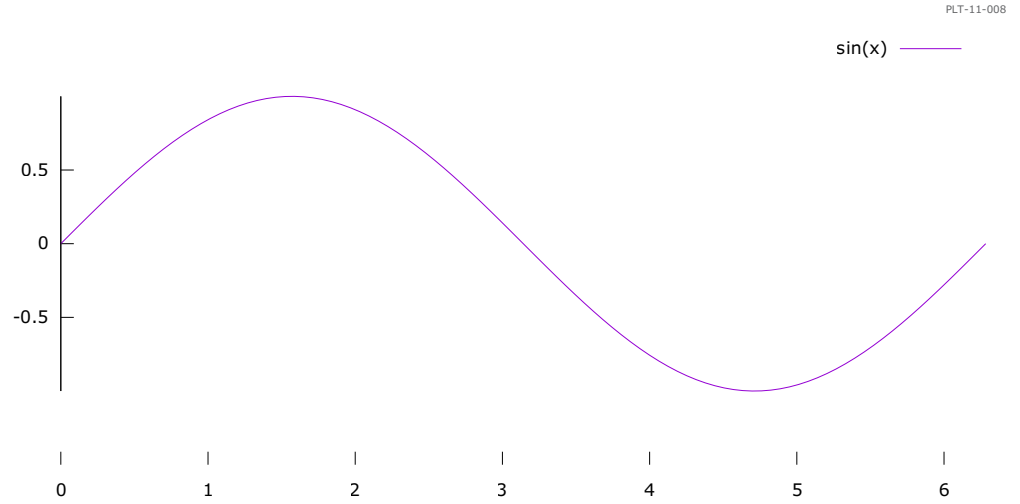
`dashtype 2`	Use dash pattern 2 for the border line style.

Shorten borders to the data extent

Compare with PLT-11-001 · Keep only left and bottom borders.

Limit ticks and borders to the values present in the plotted data. The sine curve only reaches -1 to 1, so the left border becomes shorter even though the y range remains -1.5 to 1.5.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set border 3
set xtics rangelimited
set ytics rangelimited
set tics nomirror
plot sin(x)
```



PLT-11-008 · Generated from the complete Bash block

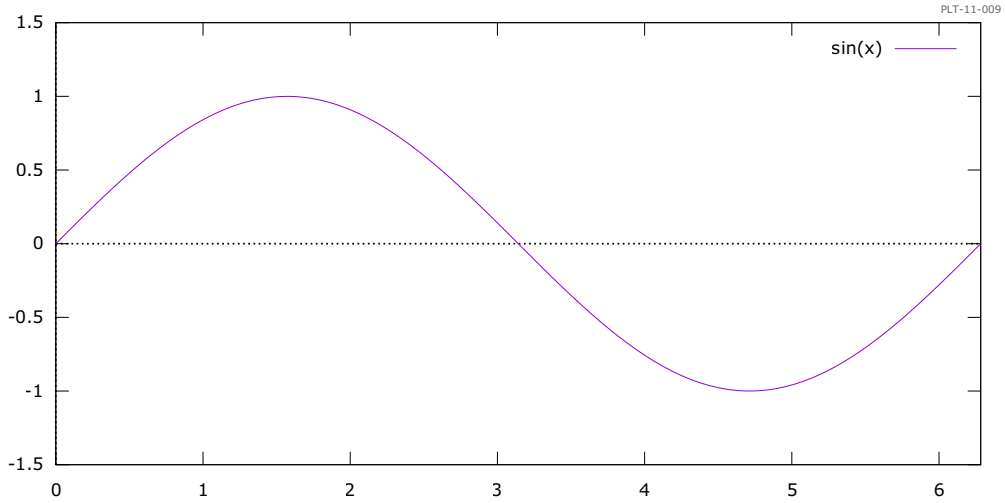
`rangelimited`	Limit automatic ticks and their border to plotted data values; apply separately to x and y ticks.

Draw the zero axes

Compare with PLT-01-007 · Leave room above and below.

Reference axes through zero can be useful even when ticks stay on the border.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set zeroaxis linewidth 2
plot sin(x)
```



PLT-11-009 · Generated from the complete Bash block

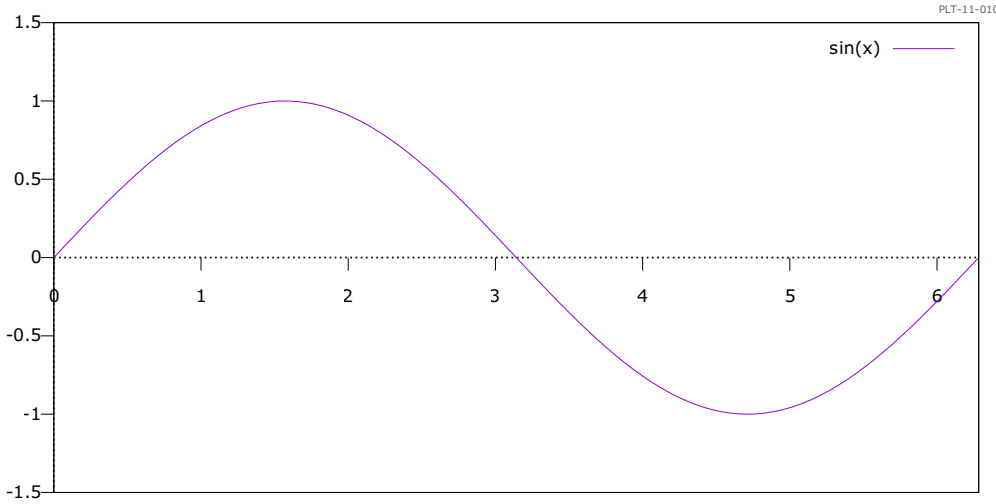
`zeroaxis`	Draw the axes at zero where they fall within the ranges.
`linewidth 2`	Relative width of those reference axes.

Put ticks on the zero axes

Compare with PLT-11-009 · Draw the zero axes.

Tick position can move from the frame to the axes themselves.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set zeroaxis linewidth 2
set xtics axis nomirror
set ytics axis nomirror
plot sin(x)
```



PLT-11-010 · Generated from the complete Bash block

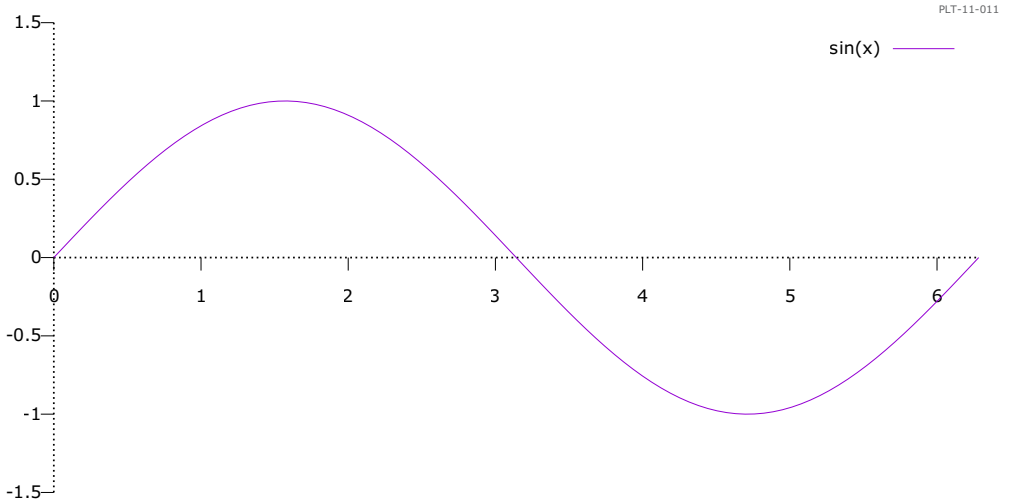
`axis`	Place ticks and labels on the zero axis when it lies in range.
`nomirror`	Do not repeat them on the opposite frame.

Remove the outer frame

Compare with PLT-11-010 · Put ticks on the zero axes.

The zero axes and their ticks remain as the coordinate reference.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set zeroaxis linewidth 2
set xtics axis nomirror
set ytics axis nomirror
unset border
plot sin(x)
```



PLT-11-011 · Generated from the complete Bash block

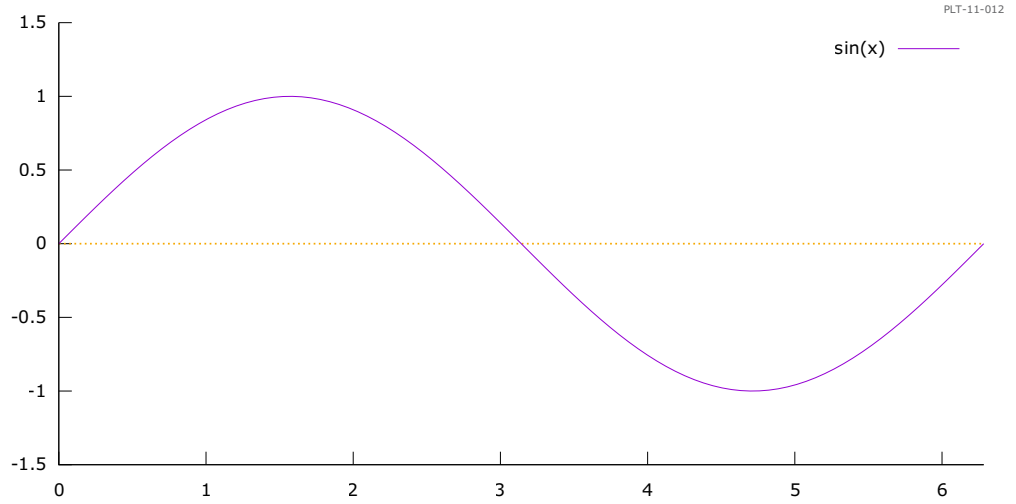
`unset border`	Suppress the surrounding frame, without removing the zero axes.

Style the horizontal zero reference

Compare with PLT-11-001 · Keep only left and bottom borders.

Add a dashed orange reference at y=0 without changing the frame. The x zero axis is the horizontal line through zero on the y scale.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set border 3
set tics nomirror
set xzeroaxis linewidth 2 dashtype 2 linecolor rgb "#F6A800"
plot sin(x)
```



PLT-11-012 · Generated from the complete Bash block

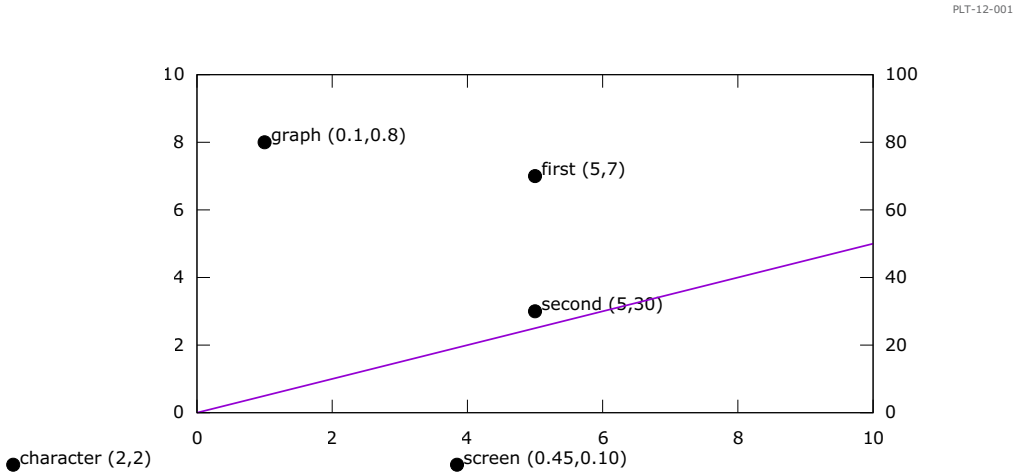
`xzeroaxis`	Draw the horizontal reference line at y=0.
`linewidth 2; dashtype 2`	Use a thicker dashed reference stroke.
`linecolor rgb "#F6A800"`	Select orange for the reference line.

Compare five coordinate systems

New baseline: the commands below define this example.

First and second refer to axis values. Graph follows the plot rectangle; screen follows the whole canvas. Character positions use nominal character width and height from the canvas origin.

```
set xrange [0:10]
set yrange [0:10]
set y2range [0:100]
set ytics nomirror
set y2tics
set lmargin at screen 0.20
set rmargin at screen 0.85
set bmargin at screen 0.20
set tmargin at screen 0.85
set label 1 "first (5,7)" at first 5,7 point pt 7
set label 2 "second (5,30)" at first 5,second 30 point pt 7
set label 3 "graph (0.1,0.8)" at graph 0.1,0.8 point pt 7
set label 4 "screen (0.45,0.10)" at screen 0.45,0.10 point pt 7
set label 5 "character (2,2)" at character 2,2 point pt 7
plot x/2 with lines lw 2 notitle
```



PLT-12-001 · Generated from the complete Bash block

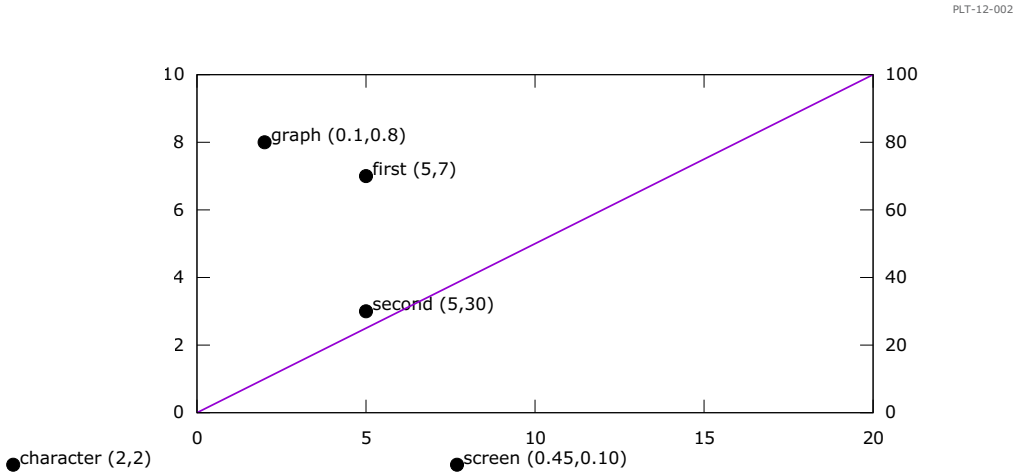
`first`	Primary data-axis coordinates; the default.
`second`	Secondary-axis coordinates; this label mixes first x and second y.
`graph`	Fractions of the plot rectangle: bottom-left (0,0), top-right (1,1).
`screen`	Fractions of the complete canvas, including margins.
`character`	Positions measured in nominal character widths and heights.

Change ranges while retaining annotation anchors

Compare with PLT-12-001 · Compare five coordinate systems.

Doubling the x range moves data-anchored labels left. Graph and screen positions stay fixed here because the explicit frame placement is unchanged. Font changes would also affect character positions.

```
set xrange [0:20]
set yrange [0:10]
set y2range [0:100]
set ytics nomirror
set y2tics
set lmargin at screen 0.20
set rmargin at screen 0.85
set bmargin at screen 0.20
set tmargin at screen 0.85
set label 1 "first (5,7)" at first 5,7 point pt 7
set label 2 "second (5,30)" at first 5,second 30 point pt 7
set label 3 "graph (0.1,0.8)" at graph 0.1,0.8 point pt 7
set label 4 "screen (0.45,0.10)" at screen 0.45,0.10 point pt 7
set label 5 "character (2,2)" at character 2,2 point pt 7
plot x/2 with lines lw 2 notitle
```



PLT-12-002 · Generated from the complete Bash block

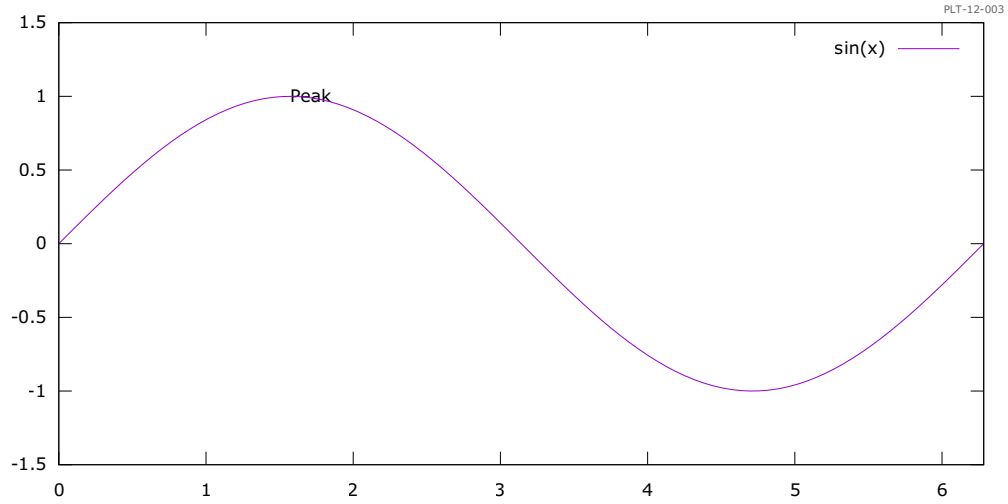
`xrange [0:20]`	Change the mapping from data x values to the plot rectangle.
`graph; screen`	Their fixed fractional anchors do not depend on data ranges.
`explicit screen margins`	Keep the rectangle fixed for this comparison.

Place text at a data point

Compare with PLT-01-007 · Leave room above and below.

A label can use the same coordinate system as the curve.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set label 1 "Peak" at first pi/2,1
plot sin(x)
```



PLT-12-003 · Generated from the complete Bash block

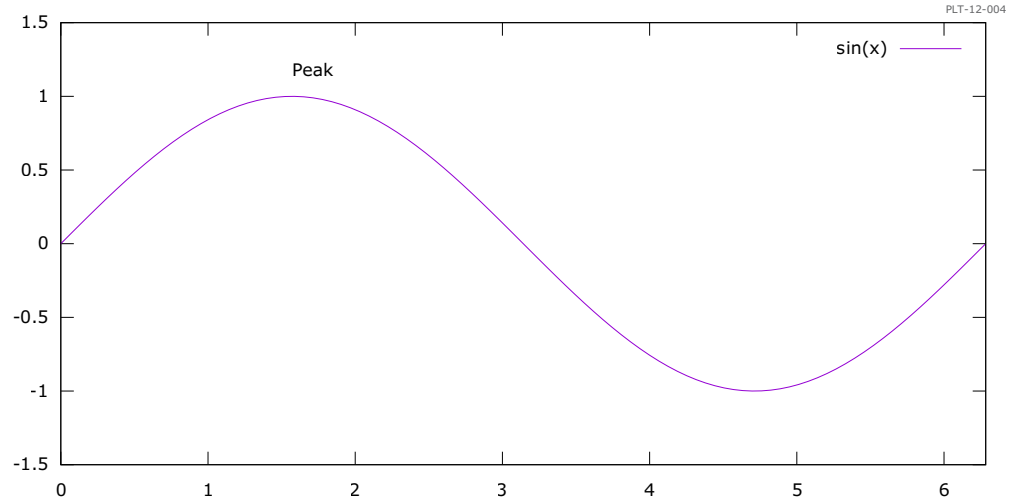
`label 1`	Create or update label number 1.
`at first`	Use primary-axis data coordinates.
`pi/2,1`	The x and y position of the anchor.

Offset a label from its point

Compare with PLT-12-003 · Place text at a data point.

Keep its data anchor, but move the text slightly for legibility.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set label 1 "Peak" at first pi/2,1 offset 0,1
plot sin(x)
```



PLT-12-004 · Generated from the complete Bash block

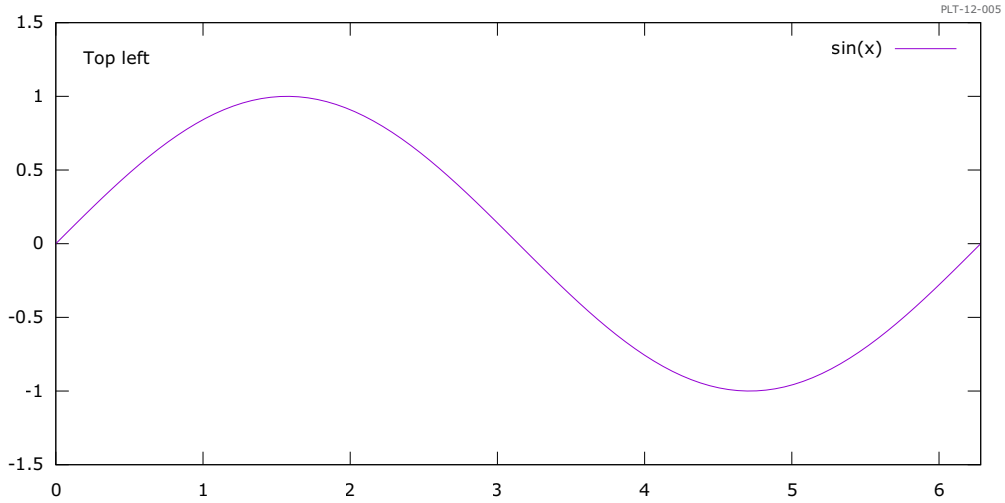
`offset 0,1` — Move the text one character height upward; keep the data anchor fixed.

Anchor text to the plot region

Compare with PLT-01-007 · Leave room above and below.

Graph coordinates keep a note in a fixed relative position when ranges change.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set label 1 "Top left" at graph 0.03,0.92
plot sin(x)
```



PLT-12-005 · Generated from the complete Bash block

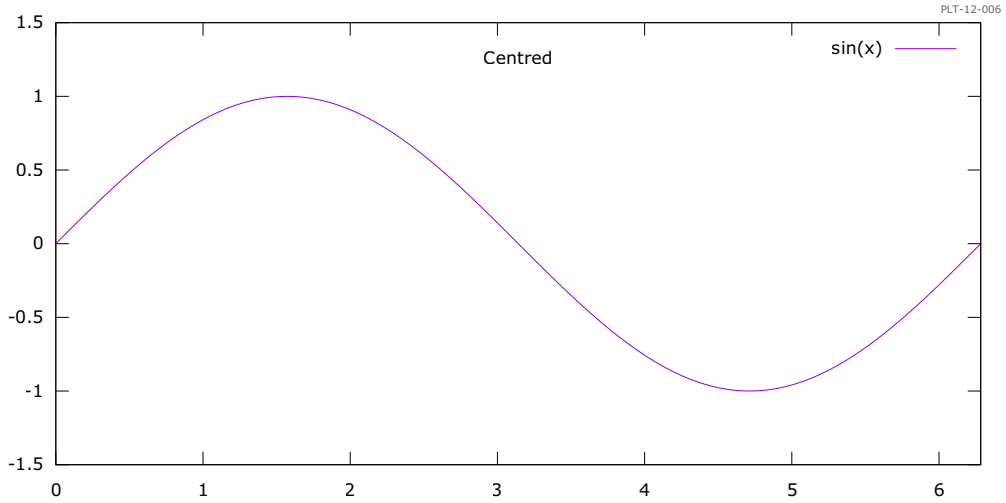
`graph`	Normalised plot-region coordinates: (0,0) is bottom-left, (1,1) is top-right.
`0.03,0.92`	3% across and 92% up the plot region.

Centre text on its anchor

Compare with PLT-12-005 · Anchor text to the plot region.

Alignment determines which part of the text sits at the chosen point.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set label 1 "Centred" at graph 0.5,0.92 center
plot sin(x)
```



PLT-12-006 · Generated from the complete Bash block

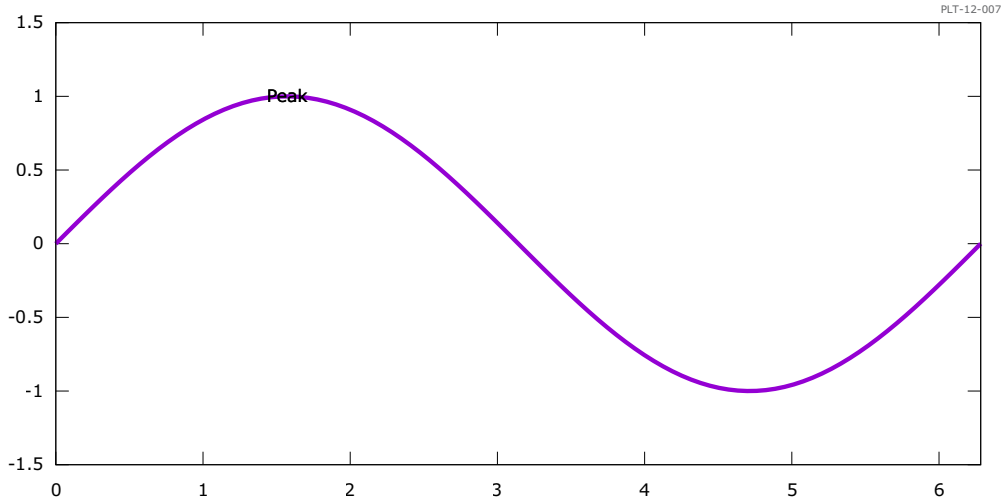
`0.5,0.92`	Horizontal centre of the plot, near its top.
`center`	Centre the text on that anchor.

Give a label an opaque text box

Compare with PLT-12-006 · Centre text on its anchor.

A text box makes an annotation readable where curves pass behind it. Its fill, border and padding are configured separately from the label text.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set style textbox opaque fc rgb "white" \
    border lc rgb "#646567" margins 1,0.5
set label 1 "Peak" at pi/2,1 center boxed front
plot sin(x) with lines lw 5 notitle
```



PLT-12-007 · Generated from the complete Bash block

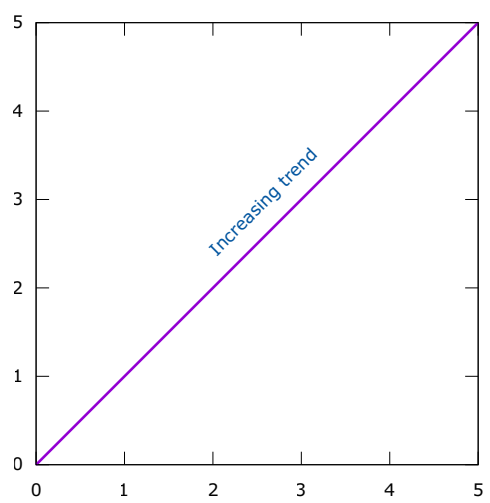
`style textbox`	Set the appearance of boxed text.
`boxed front`	Use that box and draw the label over the curve.
`margins 1,0.5`	Horizontal and vertical padding in character units.

Rotate an annotation along a trend

New baseline: the commands below define this example.

Label rotation is a display angle in degrees. It is not computed from the numeric slope, because changing the plot aspect ratio changes the visible slope.

```
set xrange [0:5]
set yrange [0:5]
set size ratio 1
set label 1 "Increasing trend" at 2,2.4 \
    rotate by 45 textcolor rgb "#00549F"
plot x with lines lw 3 notitle
```



PLT-12-008 · Generated from the complete Bash block

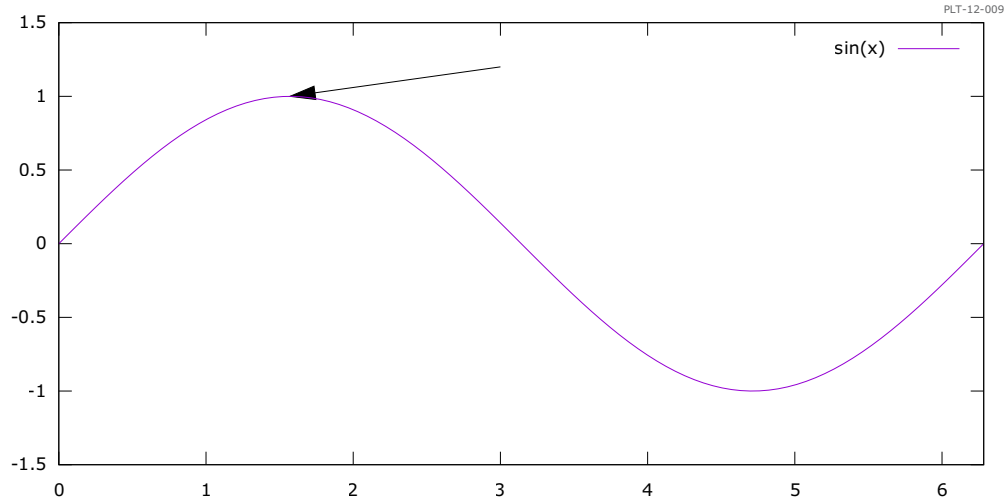
`rotate by 45`	Rotate the displayed text by 45 degrees.
`ratio 1`	Make the plot region square so equal x/y ranges have equal physical units.

Draw an arrow between points

Compare with PLT-01-007 · Leave room above and below.

Arrows can connect two data coordinates.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set arrow 1 from 3,1.2 to pi/2,1 head filled
plot sin(x)
```



PLT-12-009 · Generated from the complete Bash block

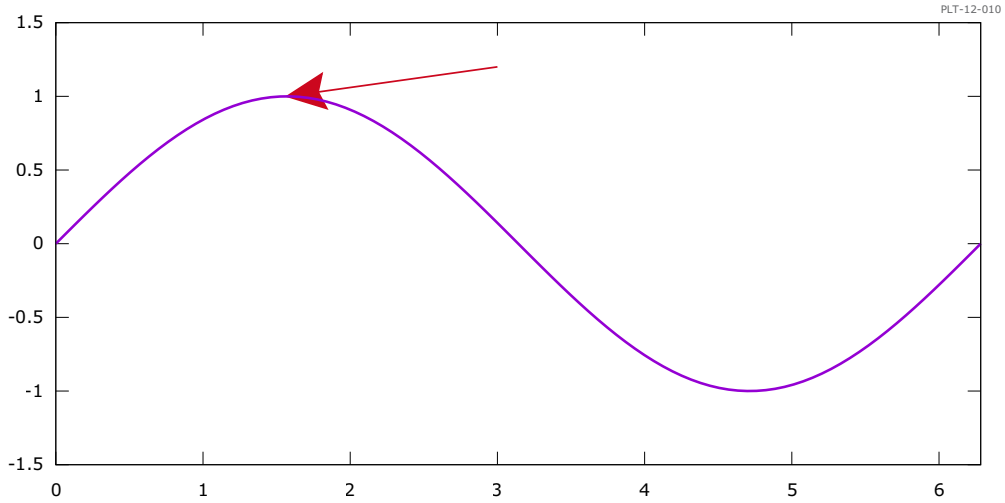
`arrow 1`	Create or update arrow number 1.
`from 3,1.2; to pi/2,1`	Start and end, in primary data coordinates by default.
`head filled`	Put a filled arrowhead at the endpoint.

Control arrowhead size and angles

Compare with PLT-12-009 · Draw an arrow between points.

Screen units keep head length independent of the data ranges. The two angles control the head opening and its back edge.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set arrow 1 from 3,1.2 to pi/2,1 head filled \
    size screen 0.04,25,70 lw 2 lc rgb "#CC071E"
plot sin(x) with lines lw 3 notitle
```



PLT-12-010 · Generated from the complete Bash block

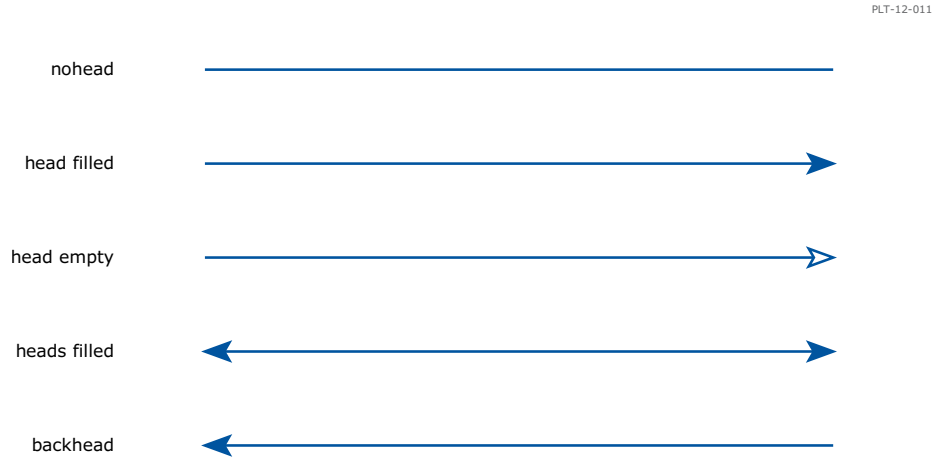
`size screen 0.04,25,70`	Head length as a canvas-width fraction, opening angle, and back angle in degrees.
`filled`	Close and fill the head polygon.

Quick reference: arrowheads

New baseline: the commands below define this example.

Keep the line geometry fixed while comparing head count and head fill. Arrowheads can be styled independently of the curve symbols.

```
unset key
unset border
unset xtics
set xrange [0:1]
set yrange [0.5:5.5]
set lmargin 18
set ytics ("nohead" 5, "head filled" 4, "head empty" 3, \
    "heads filled" 2, "backhead" 1) nomirror scale 0
set style arrow 1 size screen 0.025,20,60 lw 3 \
    lc rgb "#00549F"
set arrow 1 from 0.1,5 to 0.9,5 as 1 nohead
set arrow 2 from 0.1,4 to 0.9,4 as 1 head filled
set arrow 3 from 0.1,3 to 0.9,3 as 1 head empty
set arrow 4 from 0.1,2 to 0.9,2 as 1 heads filled
set arrow 5 from 0.1,1 to 0.9,1 as 1 backhead filled
plot NaN notitle
```



PLT-12-011 · Generated from the complete Bash block

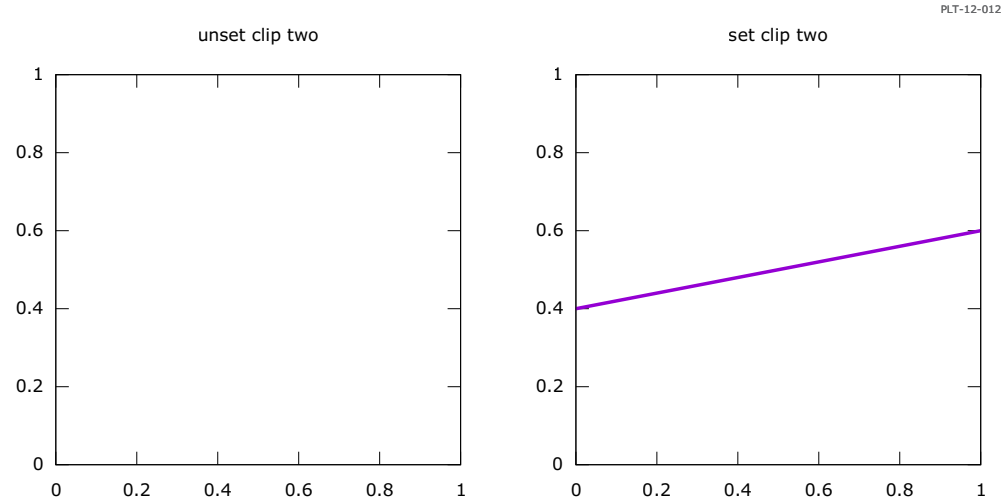
`head; backhead; heads`	Put heads at the end, start, or both ends.
`filled; empty`	Fill the arrowhead or show its outline.
`as 1`	Apply numbered arrow style 1.
`plot NaN`	Draw the frame and annotations without a data curve.

Clip a segment whose endpoints are both outside

New baseline: the commands below define this example.

A line may cross the visible region even when both endpoints lie outside it. Clip two controls whether this crossing portion is drawn. It does not control clipping of filled objects.

```
$D << EOD
-1 0.2
2 0.8
EOD
set multiplot layout 1,2
set xrange [0:1]
set yrange [0:1]
unset key
unset clip two
set title "unset clip two"
plot $D with lines lw 4
set clip two
set title "set clip two"
plot $D with lines lw 4
unset multiplot
```



PLT-12-012 · Generated from the complete Bash block

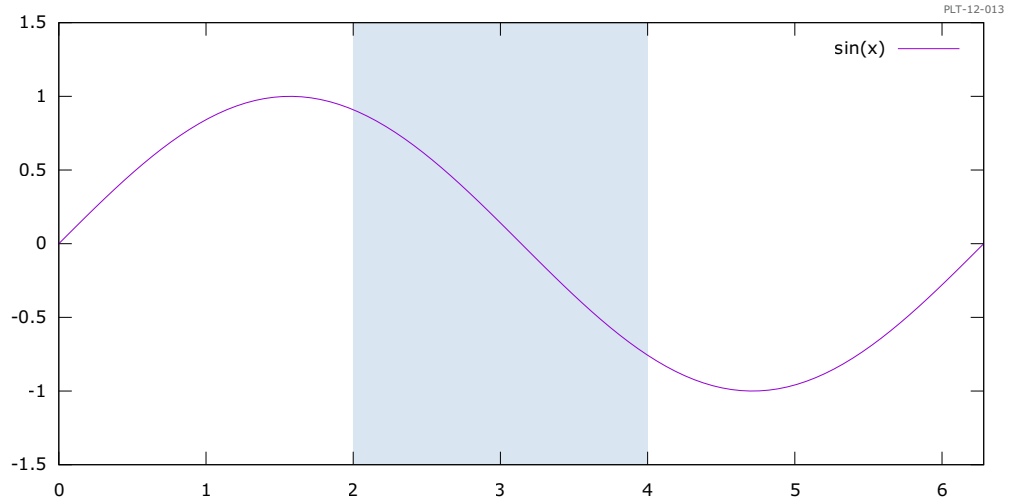
`clip two`	Draw the in-range portion even if both segment endpoints are outside.
`clip one`	Controls segments with one endpoint inside the visible range.
`clip points`	Suppress symbols that would extend beyond the frame; it does not trim them.
`objects`	Use their own clip/noclip property.

Highlight an x interval

Compare with PLT-01-007 · Leave room above and below.

Mix data x coordinates with graph y coordinates to shade the full plot height.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set object 1 rectangle from first 2, graph 0 \
    to first 4, graph 1 behind \
    fillcolor rgb "#00549F" fillstyle transparent solid 0.15 noborder
plot sin(x)
```



PLT-12-013 · Generated from the complete Bash block

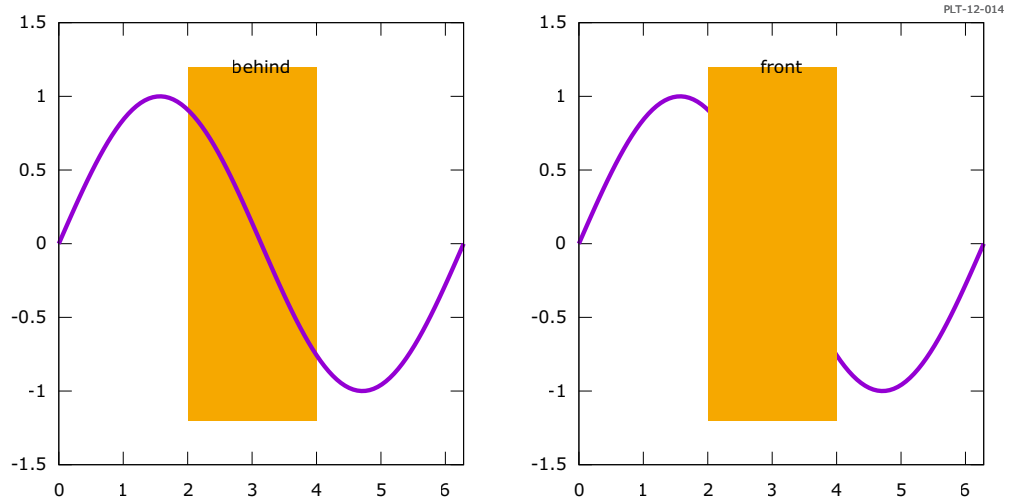
`object 1 rectangle`	Create a numbered rectangle.
`first 2 ... first 4`	The highlighted x interval in data coordinates.
`graph 0 ... graph 1`	Extend from the plot’s bottom to its top.
`fillcolor; fillstyle`	Set the colour, then use a transparent solid fill at opacity 0.15.
`behind; noborder`	Draw behind the curve and suppress the rectangle outline.

Compare object layers behind and in front

New baseline: the commands below define this example.

An opaque rectangle reveals which elements it covers. The separate front label stays readable in both panels.

```
set multiplot layout 1,2
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set object 1 rect from 2,-1.2 to 4,1.2 behind \
    fc rgb "#F6A800" fs solid 1.0 noborder
set label 1 "behind" at graph 0.5,0.9 center front
plot sin(x) with lines lw 5 notitle
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set object 1 rect from 2,-1.2 to 4,1.2 front \
    fc rgb "#F6A800" fs solid 1.0 noborder
set label 1 "front" at graph 0.5,0.9 center front
plot sin(x) with lines lw 5 notitle
unset multiplot
```



PLT-12-014 · Generated from the complete Bash block

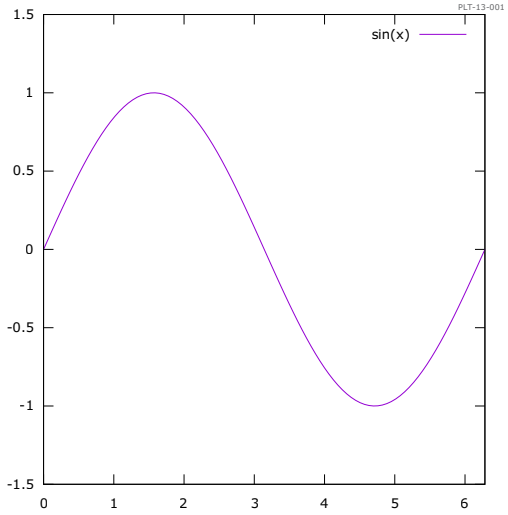
`object ... behind`	Draw the rectangle behind plot elements.
`object ... front`	Draw it over the curves.
`label ... front`	Place annotation text in the foreground.

Change the canvas shape

Compare with PLT-01-007 · Leave room above and below.

This example changes only the output dimensions. The same plot adapts to a square canvas.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
plot sin(x)
```



PLT-13-001 · Generated from the complete Bash block

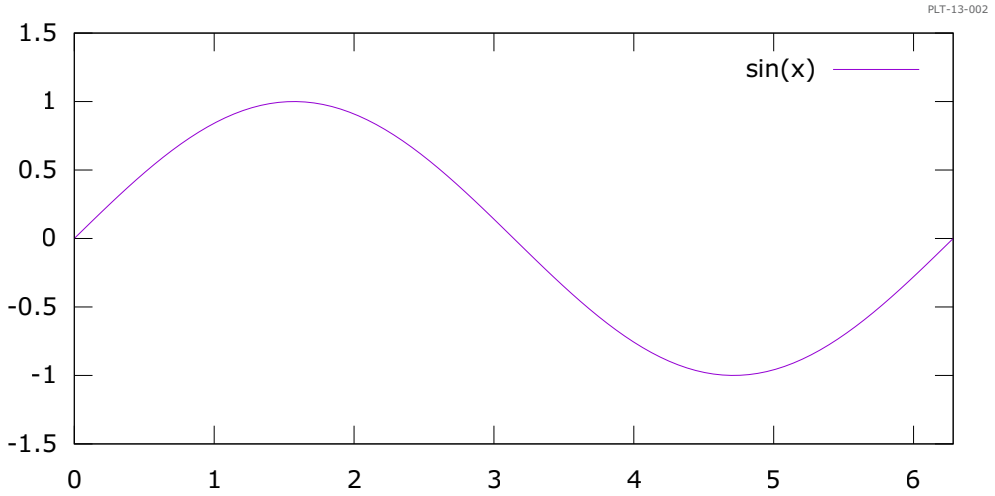
`size 800,800`	SVG canvas width and height in pixels; this is an output setting, not set size square.

Change the overall font size

Compare with PLT-01-007 · Leave room above and below.

A terminal font setting changes the default text size throughout the figure. Margins adapt automatically.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
plot sin(x)
```



PLT-13-002 · Generated from the complete Bash block

`font "sans,28"`

Use the sans family with nominal size 28 instead of 20. Individual font settings can override it.

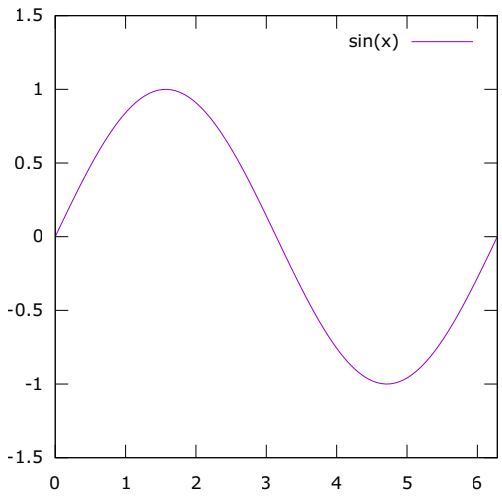
Make the plot region square

Compare with PLT-01-007 · Leave room above and below.

This changes the aspect ratio of the plotting region, not the SVG canvas.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set size square
plot sin(x)
```

PLT-13-003



PLT-13-003 · Generated from the complete Bash block

`size square` Use a square plot region within the existing 2:1 canvas.

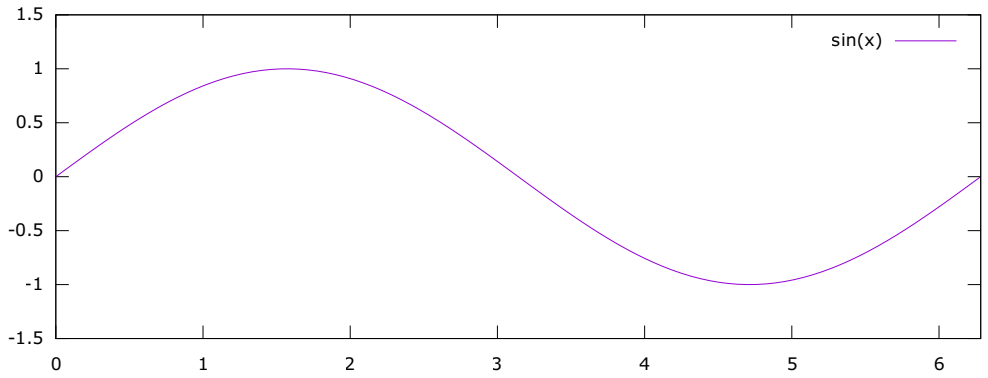
Set an explicit plot aspect ratio

Compare with PLT-13-003 · Make the plot region square.

Use a shallow plotting region while keeping the SVG canvas unchanged. A positive ratio sets physical plot height divided by plot width.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set size ratio 0.35
plot sin(x)
```

PLT-13-004



PLT-13-004 · Generated from the complete Bash block

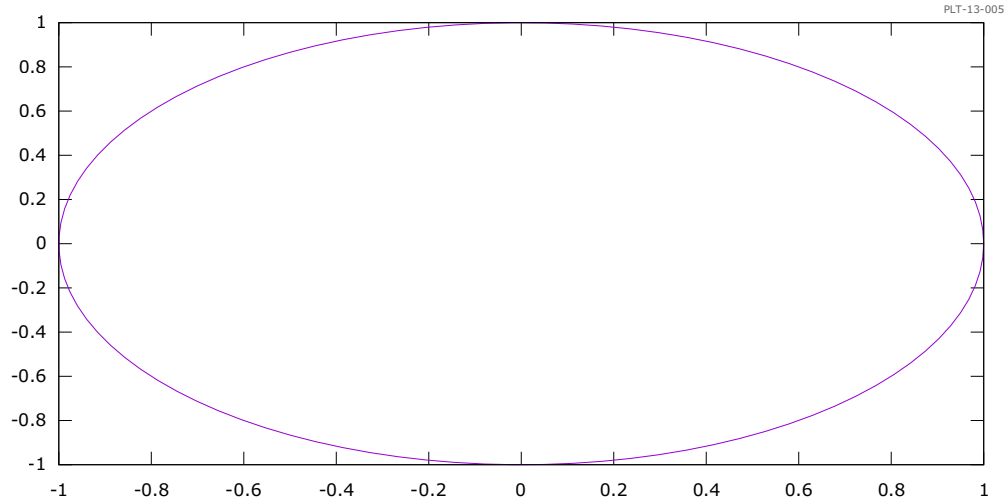
`ratio 0.35` Make the plot region height 35 percent of its width.

Compare geometry on automatic axes

New baseline: the commands below define this example.

A circle written parametrically makes the distinction between aspect ratio and equal units visible.

```
set parametric
set trange [0:2*pi]
plot cos(t),sin(t) notitle
```



PLT-13-005 · Generated from the complete Bash block

`parametric`	Interpret the next pair of expressions as x(t), y(t).
`trange [0:2*pi]`	The parameter t runs through one turn.
`cos(t),sin(t)`	Coordinates of a unit circle.
`notitle`	Omit this curve from the legend.

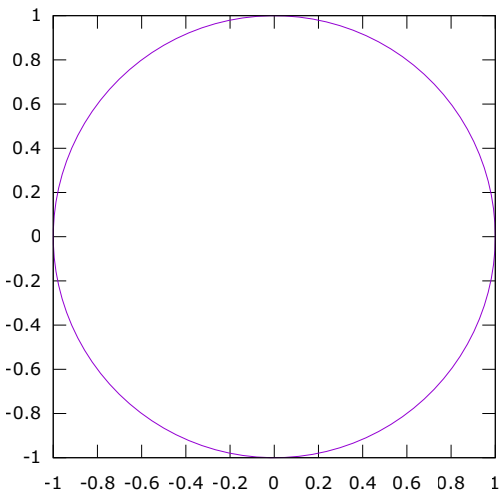
Give x and y equal physical units

Compare with PLT-13-005 · Compare geometry on automatic axes.

Equal units preserve the circle even when the canvas is wide.

```
set parametric
set trange [0:2*pi]
set size ratio -1
plot cos(t),sin(t) notitle
```

PLT-13-006



PLT-13-006 · Generated from the complete Bash block

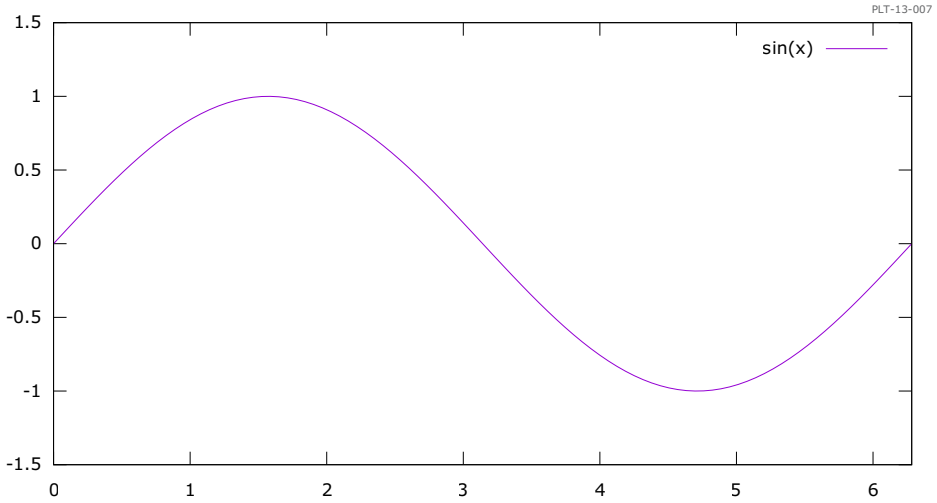
`ratio -1` Use equal physical lengths for one x unit and one y unit; this is not an ordinary negative aspect ratio.

Reserve more left margin

Compare with PLT-01-007 · Leave room above and below.

Character-based margins allocate room for labels.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set lmargin 12
plot sin(x)
```



PLT-13-007 · Generated from the complete Bash block

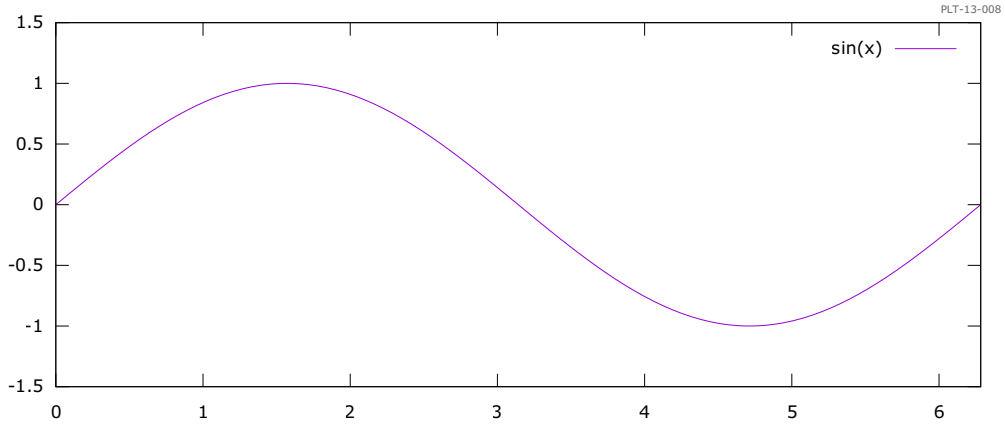
`lmargin`	Distance between the canvas's left edge and the plot border.
`12`	Margin in character widths.

Reserve more bottom margin

Compare with PLT-01-007 · Leave room above and below.

The bottom margin gives long or rotated labels more room.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set bmargin 5
plot sin(x)
```



PLT-13-008 · Generated from the complete Bash block

`bmargin`	Distance below the plot border.
`5`	Margin in character heights.

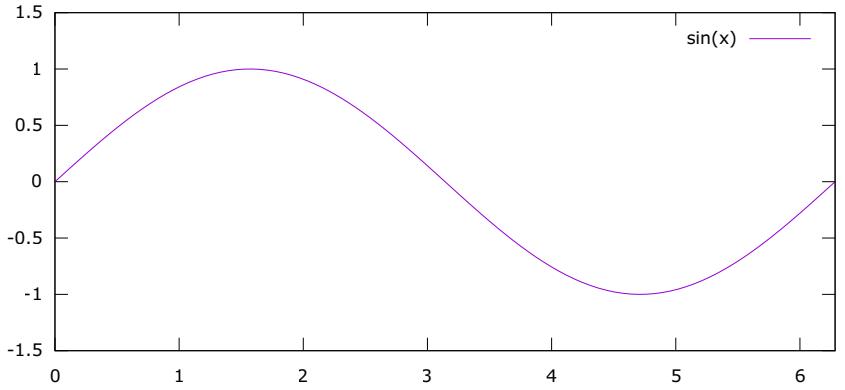
Place the plot borders explicitly

Compare with PLT-01-007 · Leave room above and below.

Screen coordinates describe fractions of the entire canvas.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set lmargin at screen 0.15
set rmargin at screen 0.90
set bmargin at screen 0.20
set tmargin at screen 0.85
plot sin(x)
```

PLT-13-009



PLT-13-009 · Generated from the complete Bash block

`at screen`	Position the border by a normalised canvas coordinate, from 0 to 1.
`0.15; 0.90`	Left and right border positions, measured from the left.
`0.20; 0.85`	Bottom and top positions, measured from the bottom.

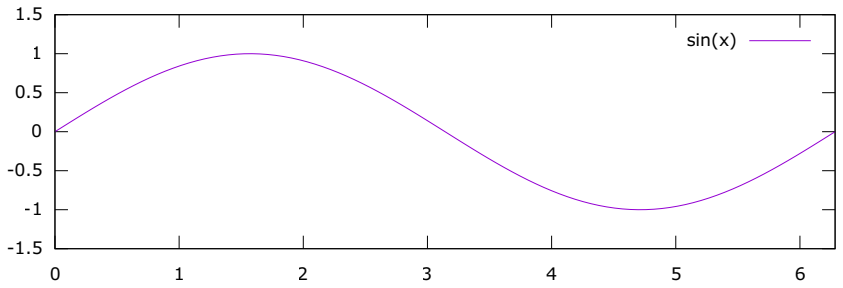
Reserve space above the plot

Compare with PLT-13-009 · Place the plot borders explicitly.

Move the top plot boundary downward while retaining the other three boundaries. The empty strip above can hold a title or annotations.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set lmargin at screen 0.15
set rmargin at screen 0.90
set bmargin at screen 0.20
set tmargin at screen 0.65
plot sin(x)
```

PLT-13-010



PLT-13-010 · Generated from the complete Bash block

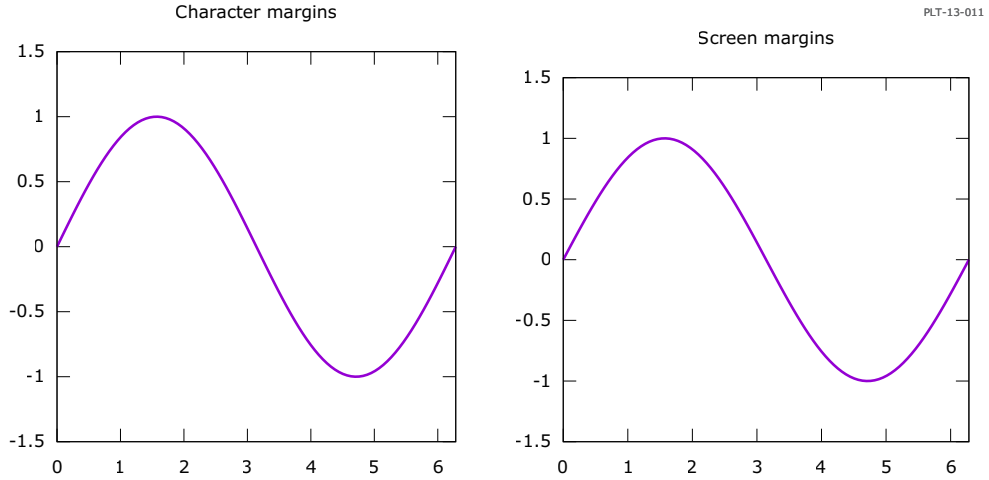
`tmargin at screen 0.65` — Place the top border at 65 percent of the canvas height, measured from the bottom.

Compare character margins with screen margins

New baseline: the commands below define this example.

Ordinary margins reserve nominal character units for labels. Screen margins instead place the frame at exact canvas fractions. The canvas itself stays 1200 by 600 in both panels.

```
set multiplot layout 1,2
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set lmargin 8
set rmargin 2
set bmargin 3
set tmargin 2
set title "Character margins"
plot sin(x) with lines lw 3 notitle
set lmargin at screen 0.58
set rmargin at screen 0.97
set bmargin at screen 0.15
set tmargin at screen 0.85
set title "Screen margins"
plot sin(x) with lines lw 3 notitle
unset multiplot
```



PLT-13-011 · Generated from the complete Bash block

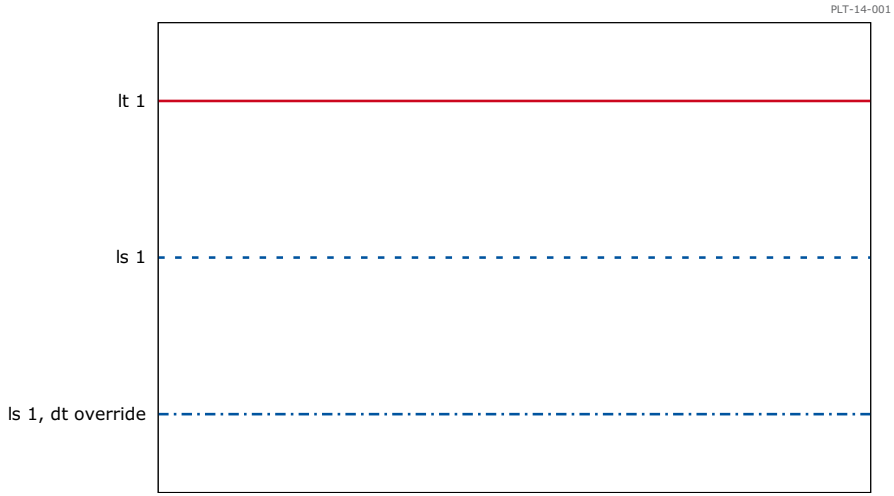
`lmargin 8`	Reserve eight nominal character widths from the current panel edge.
`at screen 0.58`	Place the left border at 58% of the entire canvas width.
`canvas versus frame`	Terminal size defines the canvas; margins determine the available plot region.

Distinguish line types from user line styles

New baseline: the commands below define this example.

Line-type numbers and user line-style numbers are separate collections. Redefining linetype 1 changes its indexed defaults; defining style line 1 does not replace that linetype.

```
unset key
unset xtics
set xrange [0:1]
set yrange [0.5:3.5]
set ytics ("lt 1" 3, "ls 1" 2, "ls 1, dt override" 1) \
    nomirror scale 0
set lmargin 24
set linetype 1 lc rgb "#CC071E" lw 3 dt 1
set style line 1 lc rgb "#00549F" lw 3 dt 2
plot 3 with lines lt 1, 2 with lines ls 1, \
    1 with lines ls 1 dt 4
```



PLT-14-001 · Generated from the complete Bash block

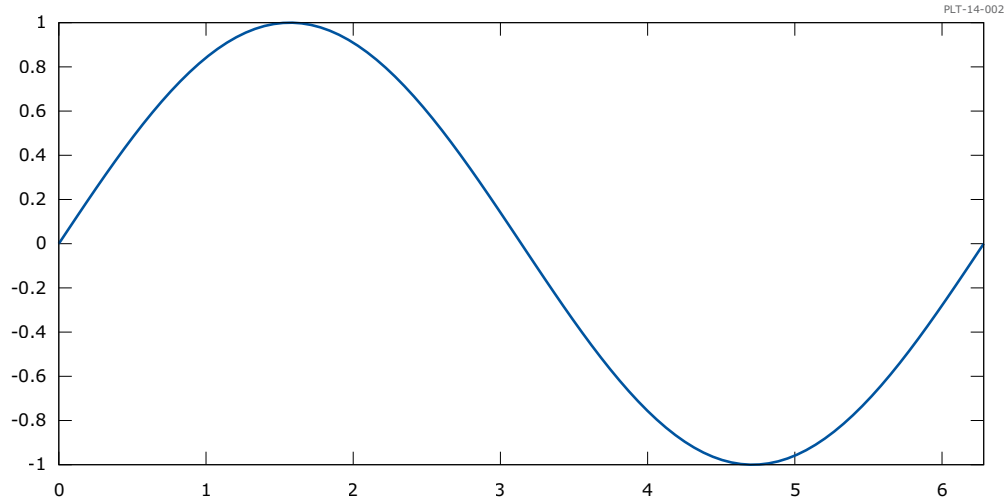
`lt 1`	Select the indexed linetype defaults.
`ls 1`	Select separately defined user style 1.
`dt 4`	Override only the dash pattern.
`reset`	Clears user line styles; linetype definitions survive a normal reset.

Replace a style fragment with a macro

New baseline: the commands below define this example.

A macro inserts command tokens. The style string becomes part of the plot command before that line executes.

```
set macros
STYLE = "lines linewidth 3 linecolor rgb '#00549F'"
set xrange [0:2*pi]
plot sin(x) with @STYLE notitle
```



PLT-14-002 · Generated from the complete Bash block

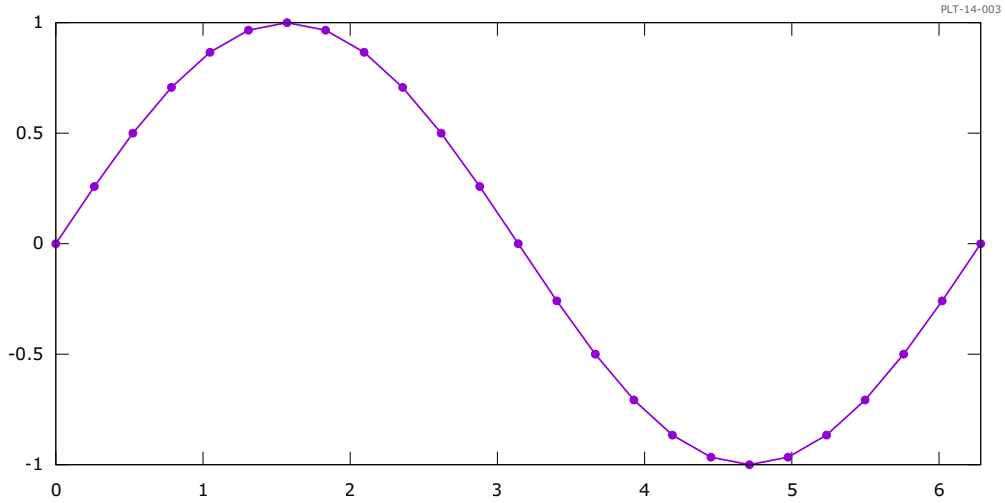
`set macros`	Explicitly enable string-variable macro substitution; accepted in gnuplot 6.
`@STYLE`	Replace the token with the contents of the string variable STYLE.
`Outer double / inner single quotes`	Keep the colour string quoted after expansion: linecolor rgb '#00549F'.
`Separate lines`	Define STYLE on an earlier line. Expansion happens before execution of the line that contains @STYLE.

Change the macro in one place

Compare with PLT-14-002 · Replace a style fragment with a macro.

The plot command stays the same; changing the string changes the inserted style.

```
set macros
STYLE = "linespoints linewidth 2 pointtype 7 pointsize 0.6"
set xrange [0:2*pi]
set samples 25
plot sin(x) with @STYLE notitle
```



PLT-14-003 · Generated from the complete Bash block

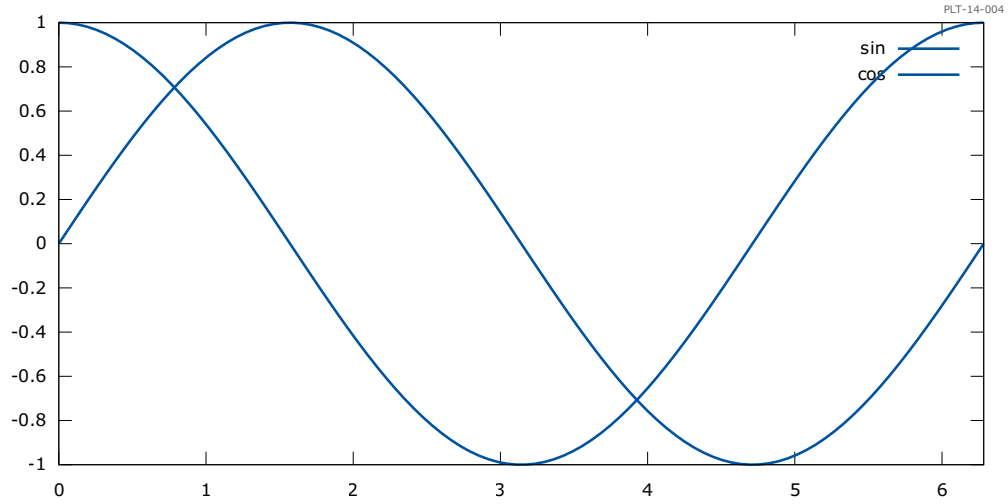
`linespoints ...`	Replace one style fragment with another; markers now appear on the curve.
`samples 25`	Reduce sampling so the markers stay distinct. This is a separate explicit change.

Reuse one style for two curves

Compare with PLT-14-002 · Replace a style fragment with a macro.

A single definition controls both occurrences. This deliberately gives both curves the same appearance.

```
set macros
STYLE = "lines linewidth 3 linecolor rgb '#00549F'"
set xrange [0:2*pi]
plot sin(x) with @STYLE title "sin", \
     cos(x) with @STYLE title "cos"
```



PLT-14-004 · Generated from the complete Bash block

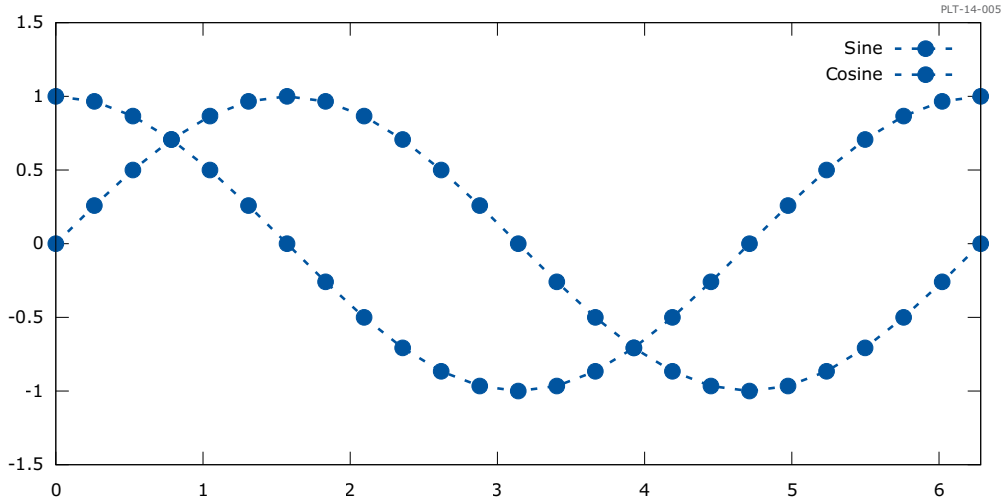
`Two @STYLE occurrences`	Both expand to the same token sequence; edit STYLE once to change both.

Define and reuse a numbered line style

Compare with PLT-14-004 · Reuse one style for two curves.

A native line style stores line and point properties under one number. It provides reusable styling without constructing macro text.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 25
set style line 1 lc rgb "#00549F" lw 3 dt 2 pt 7 ps 1.2
plot sin(x) with linespoints ls 1 title "Sine", \
    cos(x) with linespoints ls 1 title "Cosine"
```



PLT-14-005 · Generated from the complete Bash block

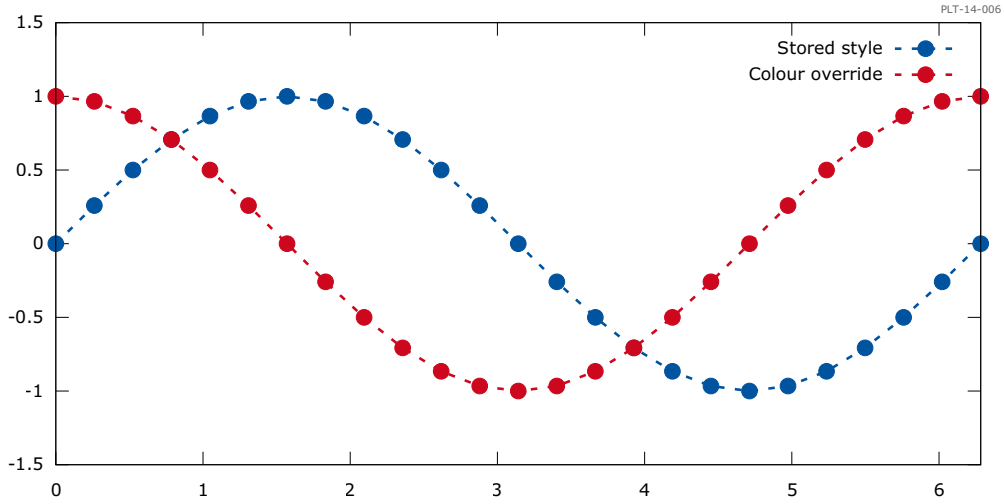
`set style line 1`	Define the properties of user line style 1.
`ls 1`	Apply that stored style to a plot element.
`lw; dt; pt; ps`	Store width, dash pattern, point symbol and point size together.

Override one property of a reusable style

Compare with PLT-14-005 · Define and reuse a numbered line style.

Place explicit overrides after the style reference. The second curve keeps the stored dash and point properties but gets a different colour.

```
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set samples 25
set style line 1 lc rgb "#00549F" lw 3 dt 2 pt 7 ps 1.2
plot sin(x) with linespoints ls 1 title "Stored style", \
    cos(x) with linespoints ls 1 lc rgb "#CC071E" \
    title "Colour override"
```



PLT-14-006 · Generated from the complete Bash block

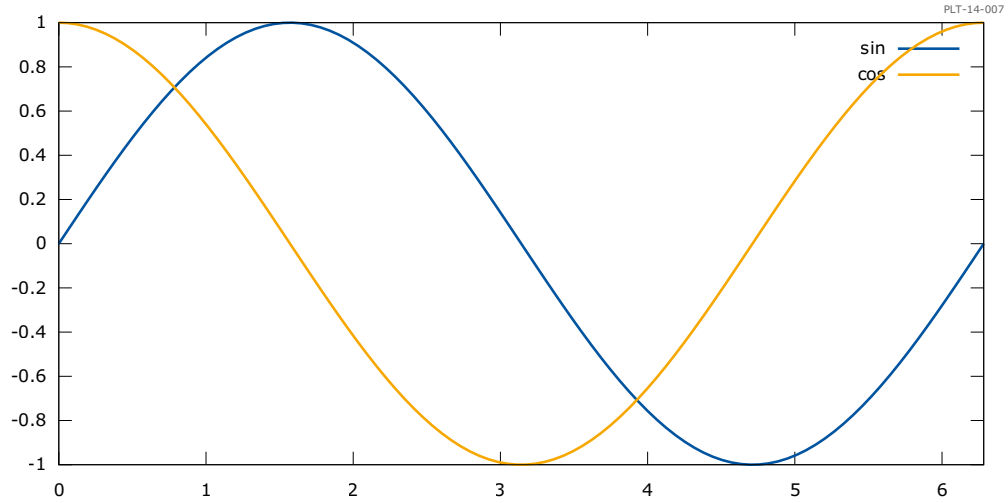
`ls 1 lc rgb ...`	Load the style, then replace its colour for this element.
`style remains unchanged`	The override does not modify style 1 for later plots.

Share geometry but vary colour

Compare with PLT-14-004 · Reuse one style for two curves.

Keep repeated line settings in the macro and place series-specific colours outside it.

```
set macros
STYLE = "lines linewidth 3"
set xrange [0:2*pi]
plot sin(x) with @STYLE linecolor rgb "#00549F" title "sin", \
     cos(x) with @STYLE linecolor rgb "#F6A800" title "cos"
```



PLT-14-007 · Generated from the complete Bash block

`@STYLE linecolor ...`

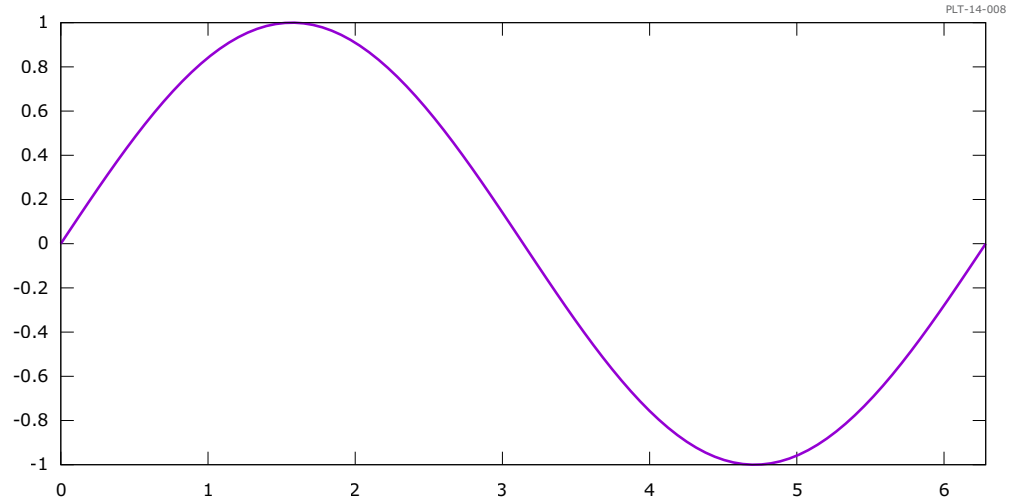
Expanded tokens and ordinary command tokens form one command. Put only genuinely shared settings in the macro.

Put a range into a macro

Compare with PLT-14-002 · Replace a style fragment with a macro.

Macros can supply syntax such as brackets, not just a numeric value.

```
set macros
XRANGE = "[0:2*pi]"
STYLE = "lines linewidth 3"
set xrange @XRANGE
plot sin(x) with @STYLE notitle
```



PLT-14-008 · Generated from the complete Bash block

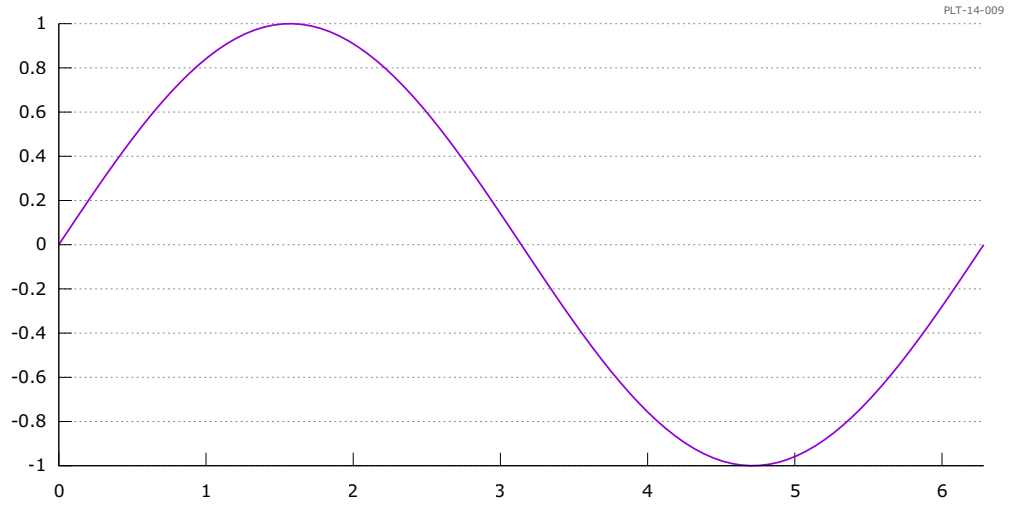
`XRANGE = "[0:2*pi]"`	The string includes the complete bracketed range syntax.
`set xrange @XRANGE`	After expansion, gnuplot reads set xrange [0:2*pi].

Bundle several setup commands

New baseline: the commands below define this example.

A macro can expand to complete commands separated by semicolons. Keep the definition on an earlier line.

```
set macros
AXES = "set border 3; set tics nomirror; set grid ytics"
@AXES
set xrange [0:2*pi]
plot sin(x) with lines linewidth 2 notitle
```



PLT-14-009 · Generated from the complete Bash block

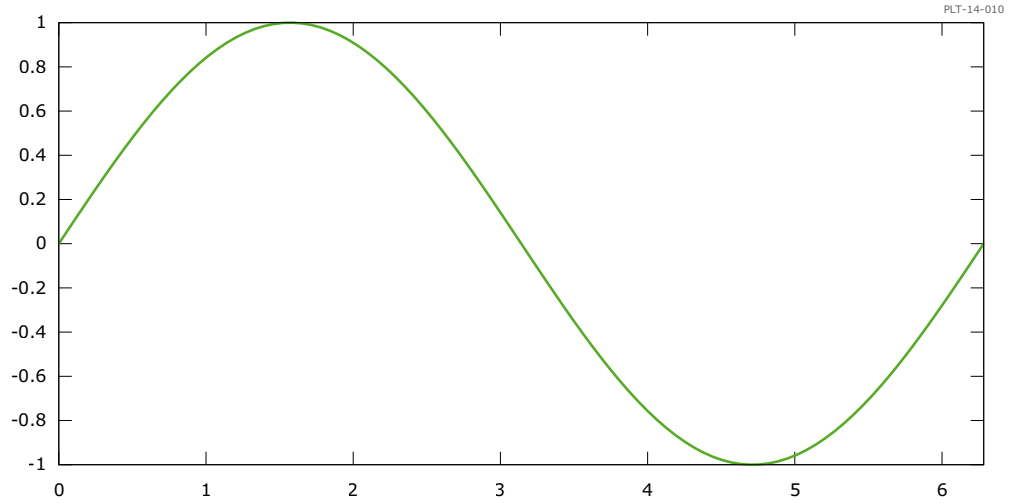
`Semicolons in AXES`	Store three commands in one string.
`@AXES on its own line`	Expand and execute the setup commands: left/bottom border, unmirrored ticks, y grid.
`Quoting boundary`	@AXES is outside quotes when invoked; the string that defines it remains quoted.

Build a macro with string concatenation

Compare with PLT-14-002 · Replace a style fragment with a macro.

The dot operator joins strings. It allows a small reusable base plus one selected colour.

```
set macros
BASE = "lines linewidth 3"
COLOUR = "#57AB27"
STYLE = BASE . " linecolor rgb '" . COLOUR . "'"
set xrange [0:2*pi]
plot sin(x) with @STYLE notitle
```



PLT-14-010 · Generated from the complete Bash block

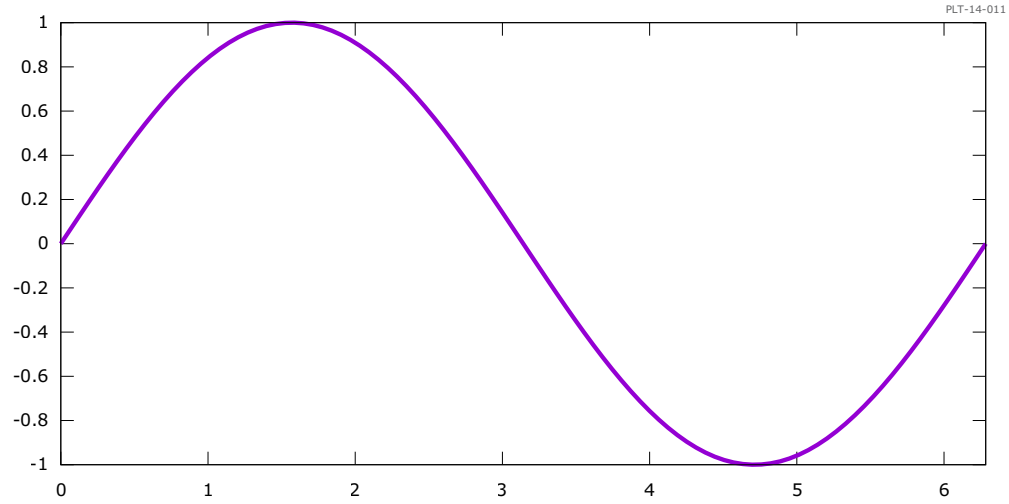
`String operator .`	Concatenate text; this is not multiplication.
`Inserted single quotes`	The resulting text is lines linewidth 3 linecolor rgb '#57AB27'.
`STYLE before @STYLE`	The entire string must exist before the invocation line is parsed.

Format a numeric macro parameter

Compare with PLT-14-002 · Replace a style fragment with a macro.

sprintf turns a number into command text. Ordinary numeric variables would also work here; this illustrates text construction.

```
set macros
W = 5
STYLE = sprintf("lines linewidth %g",W)
set xrange [0:2*pi]
plot sin(x) with @STYLE notitle
```



PLT-14-011 · Generated from the complete Bash block

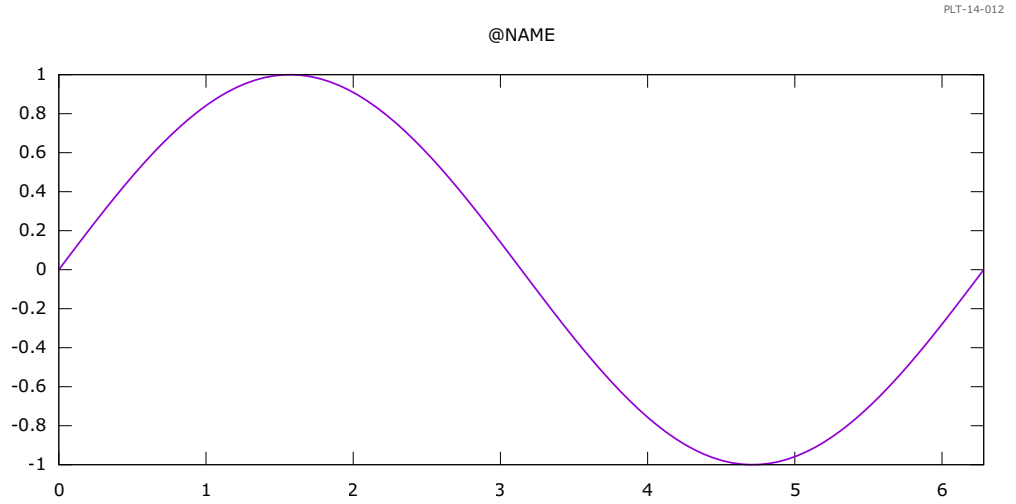
`sprintf("...%g",W)`	Substitute the number into a formatted string; %g uses compact numeric notation.
`Resulting STYLE`	The string becomes lines linewidth 5 before it is expanded.

See where a macro does not expand

New baseline: the commands below define this example.

Macro names inside quoted strings remain literal text. The plot title visibly demonstrates the boundary.

```
set macros
NAME = "A readable title"
set title "@NAME" noenhanced
set xrange [0:2*pi]
plot sin(x) with lines linewidth 2 notitle
```



PLT-14-012 · Generated from the complete Bash block

`"@NAME"`	Quoted text is not macro-expanded; the title is literally @NAME.
noenhanced	Treat @ as an ordinary character instead of an enhanced-text control character.

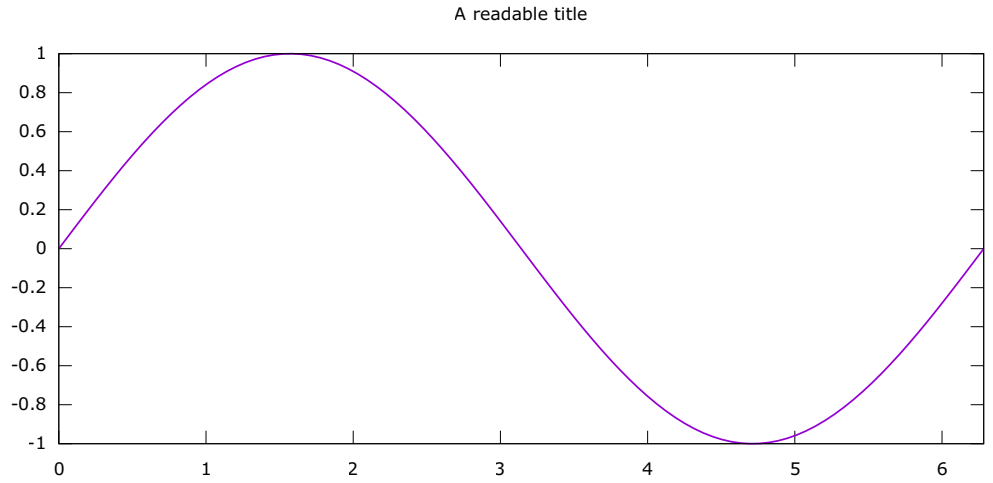
Pass the title variable correctly

Compare with PLT-14-012 · See where a macro does not expand.

Use the value directly when the command already expects a string.

```
set macros
NAME = "A readable title"
set title NAME noenhanced
set xrange [0:2*pi]
plot sin(x) with lines linewidth 2 notitle
```

PLT-14-013



PLT-14-013 · Generated from the complete Bash block

`set title NAME`

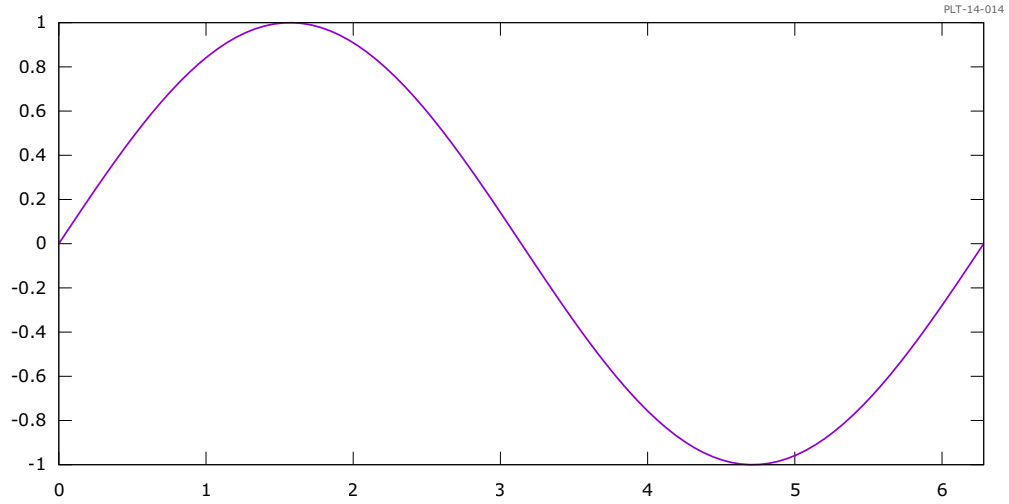
The displayed title is A readable title. A string expression is the natural solution here.

Execute constructed text with eval

Compare with PLT-14-008 · Put a range into a macro.

eval explicitly executes a command string. It is useful when you are constructing an entire command.

```
LOW = 0
HIGH = 2*pi
CMD = sprintf("set xrange [%g:%g]",LOW,HIGH)
eval CMD
plot sin(x) with lines linewidth 2 notitle
```



PLT-14-014 · Generated from the complete Bash block

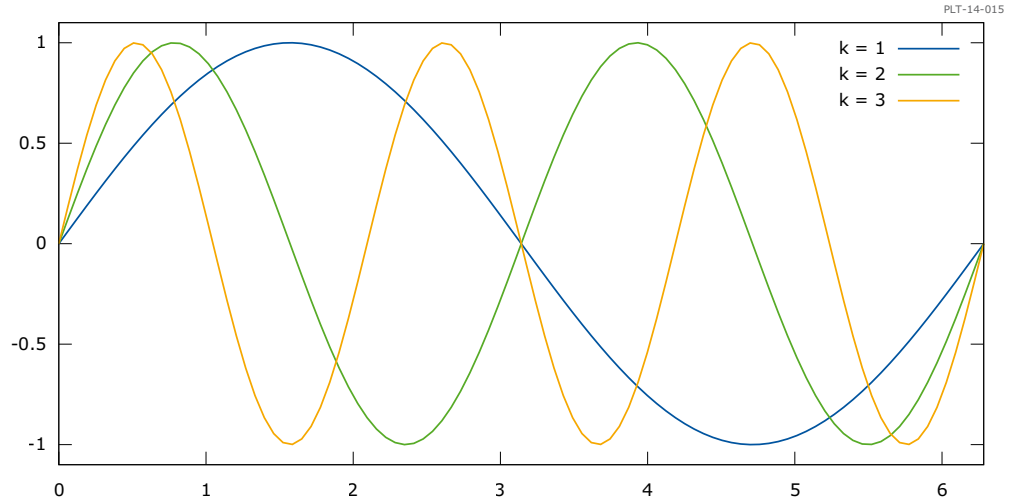
`CMD`	A string containing one complete set xrange command.
`eval CMD`	Parse and execute the string value as gnuplot commands.
`Macro versus eval`	@NAME substitutes text while a line is parsed; eval executes a string when the command runs.

Combine a plot loop with a style macro

Compare with PLT-14-002 · Replace a style fragment with a macro.

Keep the macro fixed, and let the loop variable control the numeric function and colour. The visible string lists three RWTH colour values; word selects one for each curve.

```
set macros
STYLE = "lines linewidth 2"
COLOURS = "#00549F #57AB27 #F6A800"
set xrange [0:2*pi]
set yrange [-1.1:1.1]
plot for [k=1:3] sin(k*x) with @STYLE \
     linecolor rgb word(COLOURS,k) title sprintf("k = %d",k)
```



PLT-14-015 · Generated from the complete Bash block

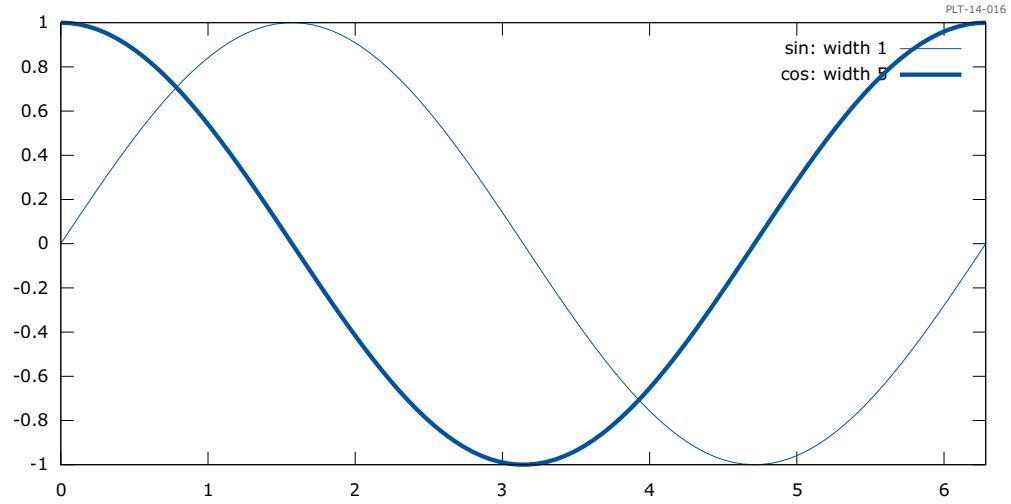
`STYLE before the loop`	The macro text is available when the plot line is parsed.
`k in sin(k*x)`	The loop variable changes the evaluated frequency for each series.
`word(COLOURS,k)`	Select the kth hexadecimal RWTH colour from the space-separated string.
`linecolor rgb`	Interpret the selected string as an RGB colour.
`sprintf("k = %d",k)`	Print the loop index as an integer in the legend.

Build macro text with a function parameter

Compare with PLT-14-004 · Reuse one style for two curves.

A string-valued function accepts the width parameter and returns a style fragment. Call it twice to give the curves different widths. The @ macros themselves take no arguments: store each function result in a string variable before expanding it. Keep assignments on earlier lines because macro expansion happens before a line executes.

```
set macros
line_style(width) = sprintf( \
    "lines lw %g lc rgb '#00549F'", width)
THIN = line_style(1)
THICK = line_style(5)
set xrange [0:2*pi]
plot sin(x) with @THIN title "sin: width 1", \
     cos(x) with @THICK title "cos: width 5"
```



PLT-14-016 · Generated from the complete Bash block

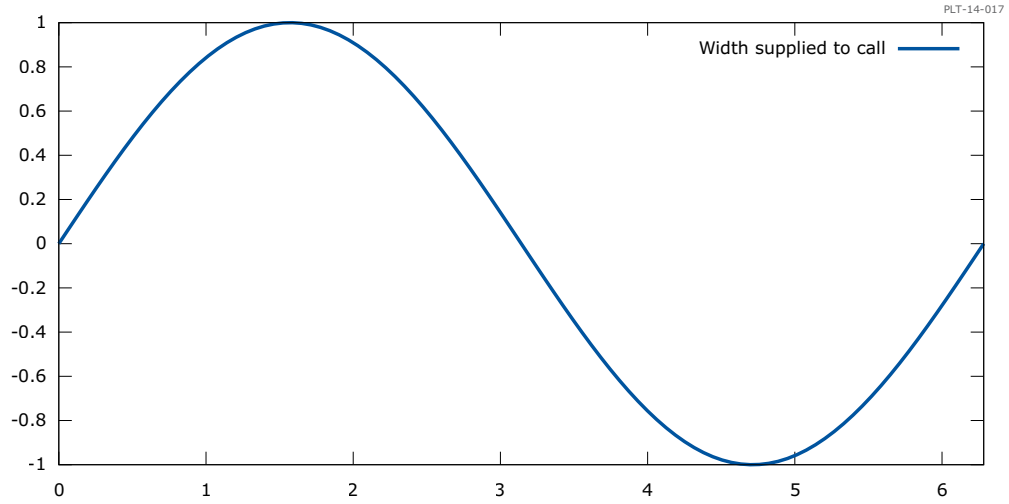
`line_style(width)`	Define a function that accepts one argument and returns a string.
`sprintf("...%g...", width)`	Insert the argument into the style text; lw and lc abbreviate linewidth and linecolor.
`line_style(1); line_style(5)`	Generate two style strings from the same function, with widths 1 and 5.
`@THIN; @THICK`	Expand the stored strings. Calling @line_style(3) directly is not supported.

Pass parameters to a reusable command block

Compare with PLT-14-016 · Build macro text with a function parameter.

Call accepts arguments for a reusable script or command datablock. Numeric arguments keep their types in ARGV; ARG1 and ARG2 provide string representations. Unlike an @ macro, the called block executes commands.

```
$Setup << ENDSETUP
set style line 1 lw ARGV[1] lc rgb ARG2
ENDSETUP
call $Setup 4 "#00549F"
set xrange [0:2*pi]
plot sin(x) with lines ls 1 title "Width supplied to call"
```



PLT-14-017 · Generated from the complete Bash block

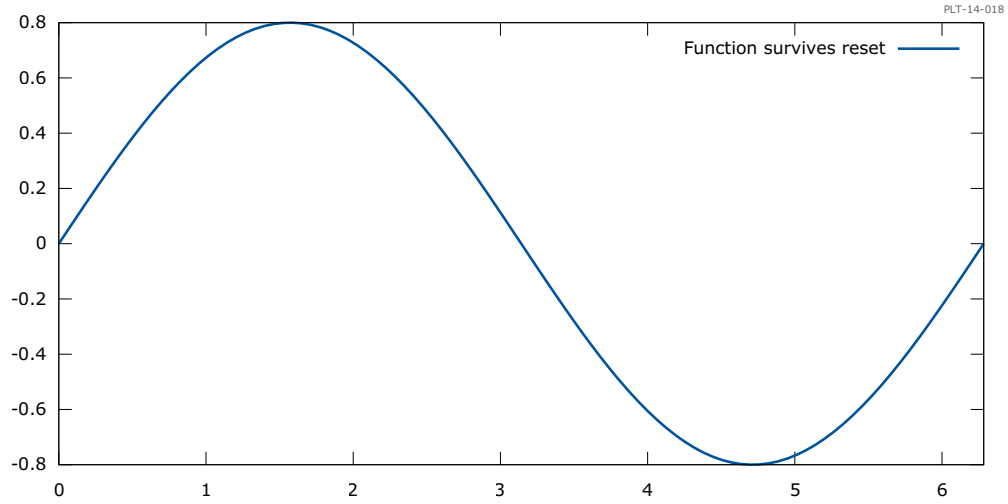
`call $Setup 4 "#00549F"`	Execute the command block with two arguments.
`ARGV[1]`	Use the first argument as its original numeric value.
`ARG2`	Use the second argument as a string colour name.
`call "setup.gp" ...`	The same mechanism can call commands stored in a file.

Reset plot settings within a session

New baseline: the commands below define this example.

Reset clears most plot settings and user line styles. Variables and functions, terminal/output selection, and redefined linetypes survive a normal reset. The reserved tour identifier is restored after its label is cleared.

```
amplitude = 0.8
f(x) = amplitude*sin(x)
set linetype 1 lc rgb "#00549F" lw 3
set grid
set xrange [0:2*pi]
set style line 1 lc rgb "#CC071E" dt 2
reset
set encoding utf8
set label 999 'PLT-14-018' at screen 0.985,0.975 \
    right front font ',12' textcolor rgb '#646567'
set xrange [0:2*pi]
plot f(x) with lines lt 1 title "Function survives reset"
```



PLT-14-018 · Generated from the complete Bash block

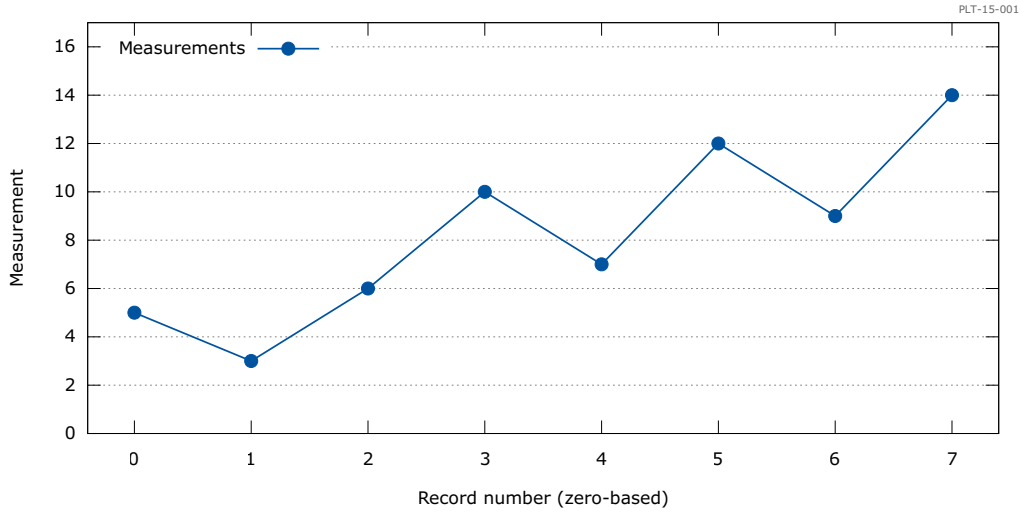
`reset`	Restore most plot defaults, remove labels and user line styles.
`amplitude; f(x)`	Ordinary variables and functions remain available.
`set linetype`	Its redefined defaults persist through a normal reset.
`terminal; output`	Retain their current selections.

Plot measurements by record number

New baseline: the commands below define this example.

Begin with eight measurements at unevenly spaced positions. Plot the second column against pseudo-column 0, the zero-based row number. The first column is deliberately unused in this first view. Later steps repeat these data in the complete Bash block.

```
$D << EOD
1 5
2 3
4 6
7 10
11 7
16 12
22 9
29 14
EOD
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "Record number (zero-based)"
set yrange [0:17]
set xrange [-0.4:7.4]
set xtics 1
plot $D using 0:2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements"
```



PLT-15-001 · Generated from the complete Bash block

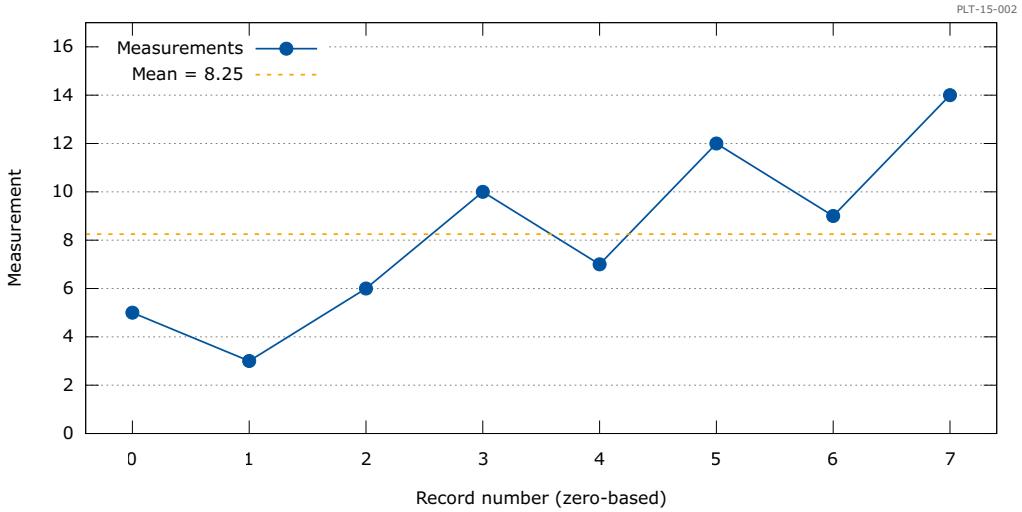
`using 0:2`	Use the row number as x and the second column as y.
`$D`	An inline datablock keeps these eight synthetic observations inside the example.

Calculate and draw the mean

Compare with PLT-15-001 · Plot measurements by record number.

Add one-column statistics before plotting. The named variables describe column 2: eight records with mean 8.25. Draw that mean as a constant function; nooutput suppresses the default statistics report.

```
stats $D using 2 name "A" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "Record number (zero-based)"
set yrange [0:17]
set xrange [-0.4:7.4]
set xtics 1
plot $D using 0:2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     A_mean with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",A_mean)
print A_records,A_mean,A_min,A_max, \
      A_index_min,A_index_max
```



PLT-15-002 · Generated from the complete Bash block

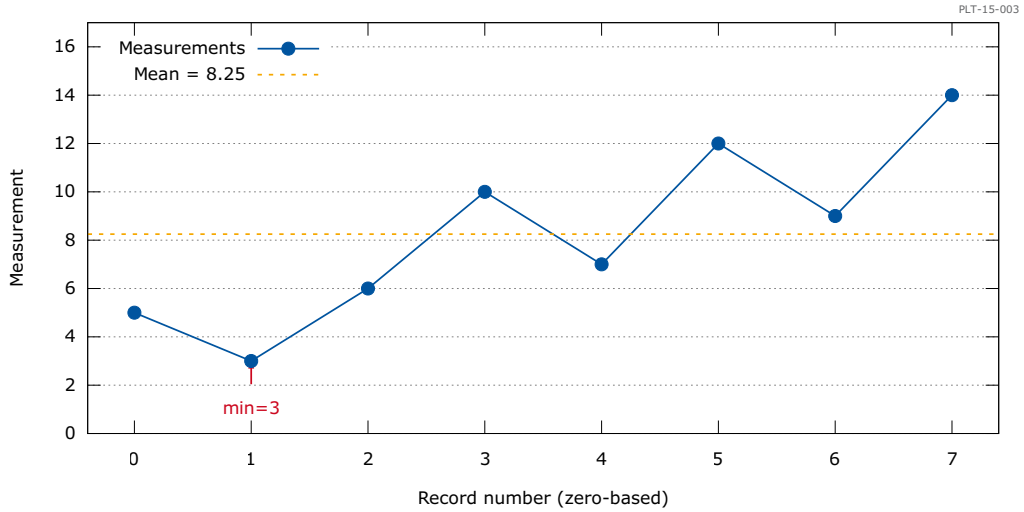
`stats $D using 2 name "A" nooutput`	Calculate one-column statistics with the A_ variable prefix.
`A_records; A_mean`	The observation count and arithmetic mean.
`A_mean with lines`	A constant expression draws a horizontal reference line.

Locate the minimum by its record index

Compare with PLT-15-002 · Calculate and draw the mean.

Add an arrow and a value label at the lowest observation. A_index_min is 1 because the minimum is the second record. It matches pseudo-column 0 in the plotted data; it is not the first-column position 2.

```
stats $D using 2 name "A" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "Record number (zero-based)"
set yrange [0:17]
set xrange [-0.4:7.4]
set xtics 1
set arrow 1 from first A_index_min, graph 0.12 \
    to first A_index_min, first A_min filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",A_min) \
    at first A_index_min, graph 0.06 center tc rgb "#CC071E"
plot $D using 0:2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     A_mean with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",A_mean)
print A_records,A_mean,A_min,A_max, \
      A_index_min,A_index_max
```



PLT-15-003 · Generated from the complete Bash block

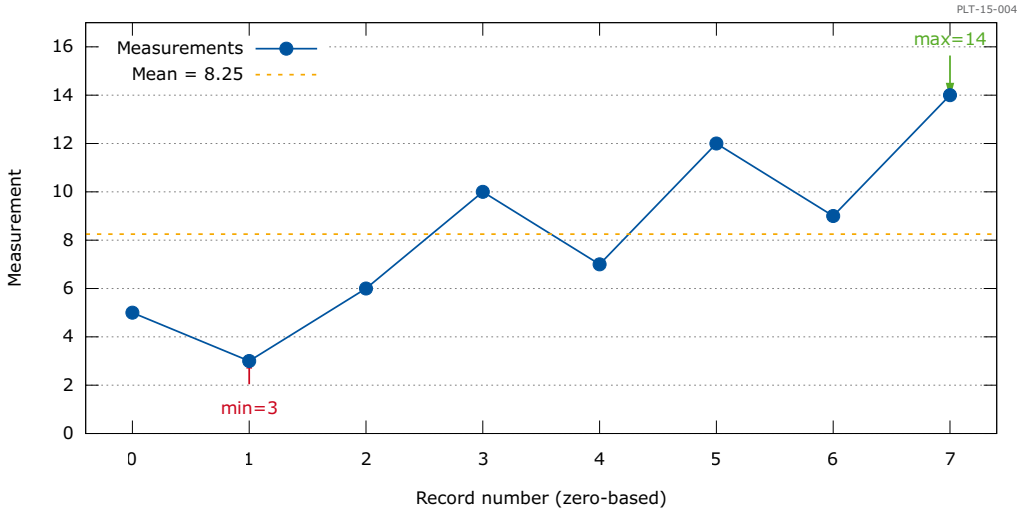
`A_index_min; A_min`	Zero-based record index of the minimum, and its y value 3.
`first ..., graph ...`	Mix data coordinates for x with a fraction of the plotting area for y.
`filled head`	Give the annotation arrow a solid arrowhead.

Locate the maximum as well

Compare with PLT-15-003 · Locate the minimum by its record index.

Keep the minimum annotation and add the maximum independently. The largest observation is 14 at record index 7. Separate arrow and label numbers let both annotations coexist.

```
stats $D using 2 name "A" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "Record number (zero-based)"
set yrange [0:17]
set xrange [-0.4:7.4]
set xtics 1
set arrow 1 from first A_index_min, graph 0.12 \
    to first A_index_min, first A_min filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",A_min) \
    at first A_index_min, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first A_index_max, graph 0.92 \
    to first A_index_max, first A_max filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",A_max) \
    at first A_index_max, graph 0.96 center tc rgb "#57AB27"
plot $D using 0:2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     A_mean with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",A_mean)
print A_records,A_mean,A_min,A_max, \
      A_index_min,A_index_max
```



PLT-15-004 · Generated from the complete Bash block

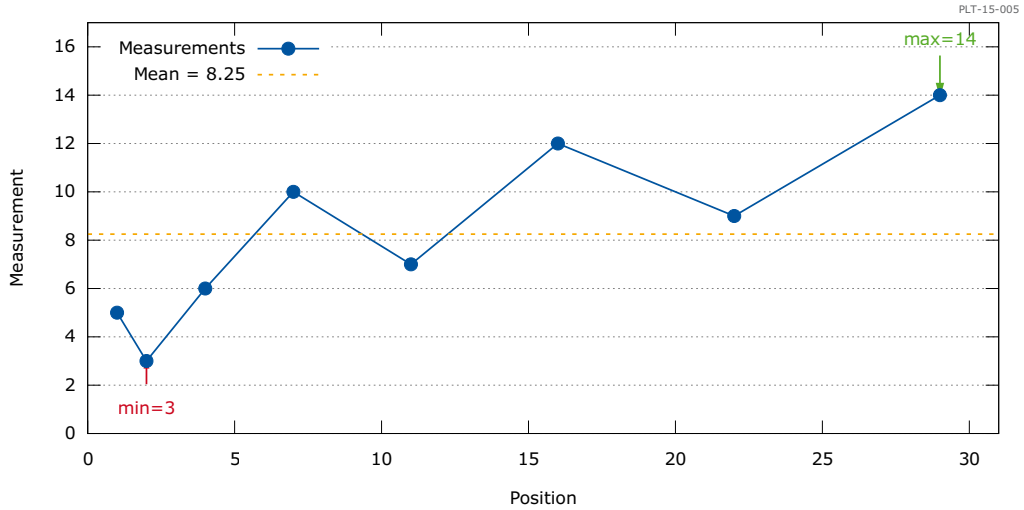
`A_index_max; A_max`	Zero-based record index 7 and maximum measurement 14.
`arrow 2; label 2`	A second numbered annotation leaves the minimum annotation intact.
`graph 0.92`	Start near the top of the plotting area, independently of the y scale.

Use the actual x positions

Compare with PLT-15-004 · Locate the maximum as well.

Switch from record numbers to the first-column positions. Analyze both columns together and use B_pos_min_y and B_pos_max_y for the annotation coordinates. The minimum now sits at x=2 and the maximum at x=29; their row indices would put the arrows in the wrong places.

```
stats $D using 1:2 name "B" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "Position"
set yrange [0:17]
set xrange [0:31]
set arrow 1 from first B_pos_min_y, graph 0.12 \
    to first B_pos_min_y, first B_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",B_min_y) \
    at first B_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first B_pos_max_y, graph 0.92 \
    to first B_pos_max_y, first B_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",B_max_y) \
    at first B_pos_max_y, graph 0.96 center tc rgb "#57AB27"
plot $D using 1:2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     B_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",B_mean_y)
print B_records,B_mean_y,B_min_y,B_max_y, \
      B_pos_min_y,B_pos_max_y
```



PLT-15-005 · Generated from the complete Bash block

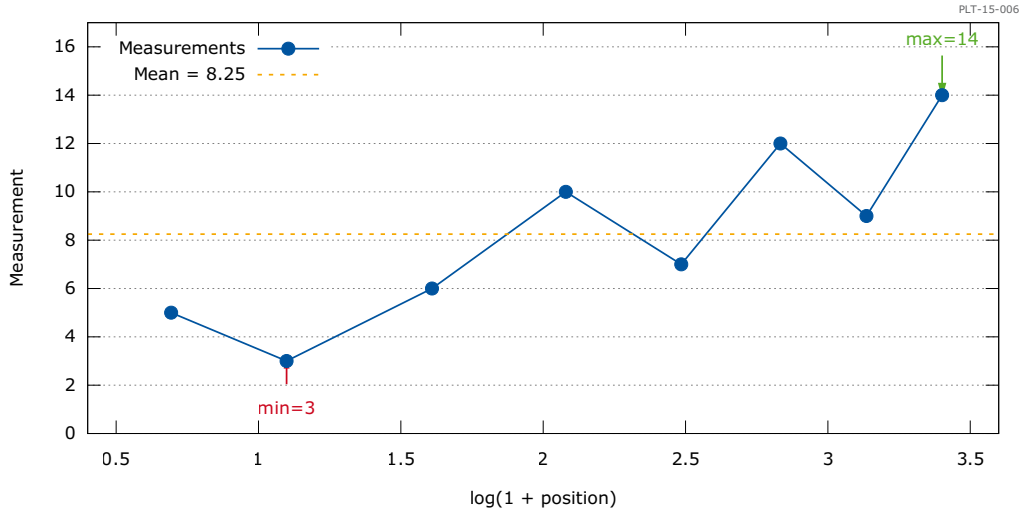
`stats $D using 1:2 name "B" nooutput`	Analyze the same x and y coordinates that will be plotted.
`B_mean_y`	Two-column statistics add _x and _y suffixes to column summaries.
`B_pos_min_y; B_pos_max_y`	Actual x coordinates associated with the lowest and highest y values.

Transform x consistently for statistics and plotting

Compare with PLT-15-005 · Use the actual x positions.

Define f(x)=log(1+x) and apply it to column 1 in both commands. The extrema remain y=3 and y=14, but their x coordinates become log(3) and log(30). This changes the coordinate values themselves; the displayed x axis stays linear.

```
f(x) = log(1+x)
stats $D using (f($1)):2 name "C" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "log(1 + position)"
set yrange [0:17]
set xrange [0.4:3.6]
set arrow 1 from first C_pos_min_y, graph 0.12 \
    to first C_pos_min_y, first C_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",C_min_y) \
    at first C_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first C_pos_max_y, graph 0.92 \
    to first C_pos_max_y, first C_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",C_max_y) \
    at first C_pos_max_y, graph 0.96 center tc rgb "#57AB27"
plot $D using (f($1)):2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     C_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",C_mean_y)
print C_records,C_mean_y,C_min_y,C_max_y, \
      C_pos_min_y,C_pos_max_y
```



PLT-15-006 · Generated from the complete Bash block

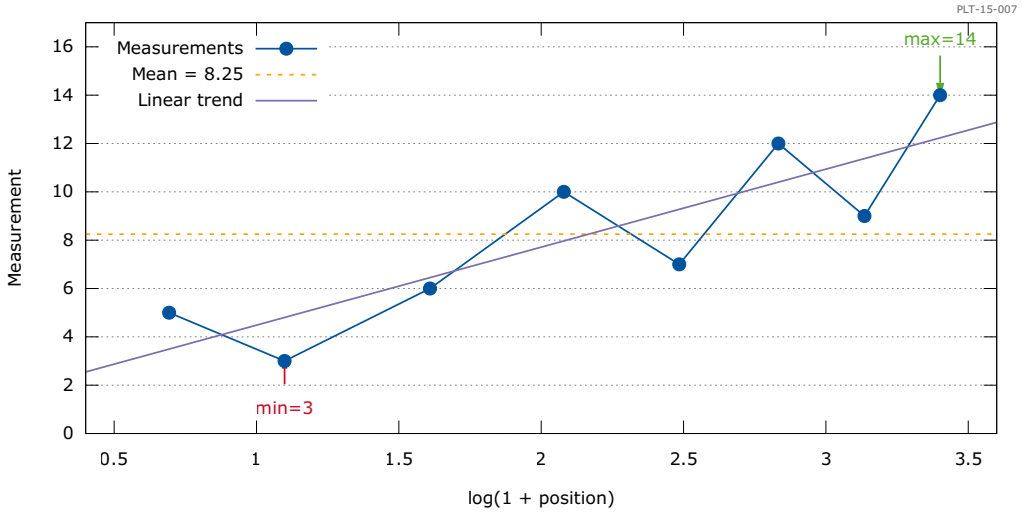
`f(x) = log(1+x)`	Natural logarithm of a shifted, positive position.
`using (f($1)):2`	Use the same transformed x expression for statistics and the plot.
`C_pos_min_y; C_pos_max_y`	Extremum positions in the transformed coordinate system.

Draw the linear trend from two-column statistics

Compare with PLT-15-006 · Transform x consistently for statistics and plotting.

Add the least-squares line already summarized by the two-column stats command. Its slope and intercept describe measurement versus log(1+position). Changing back to the original position coordinates would require recomputing the statistics.

```
f(x) = log(1+x)
stats $D using (f($1)):2 name "C" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "log(1 + position)"
set yrange [0:17]
set xrange [0.4:3.6]
set arrow 1 from first C_pos_min_y, graph 0.12 \
    to first C_pos_min_y, first C_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",C_min_y) \
    at first C_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first C_pos_max_y, graph 0.92 \
    to first C_pos_max_y, first C_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",C_max_y) \
    at first C_pos_max_y, graph 0.96 center tc rgb "#57AB27"
trend(x) = C_slope*x + C_intercept
plot $D using (f($1)):2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     C_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",C_mean_y), \
     trend(x) with lines lw 2 lc rgb "#7A6FAC" title "Linear trend"
print C_records,C_mean_y,C_min_y,C_max_y, \
      C_pos_min_y,C_pos_max_y
print C_slope,C_intercept,C_correlation
```



PLT-15-007 · Generated from the complete Bash block

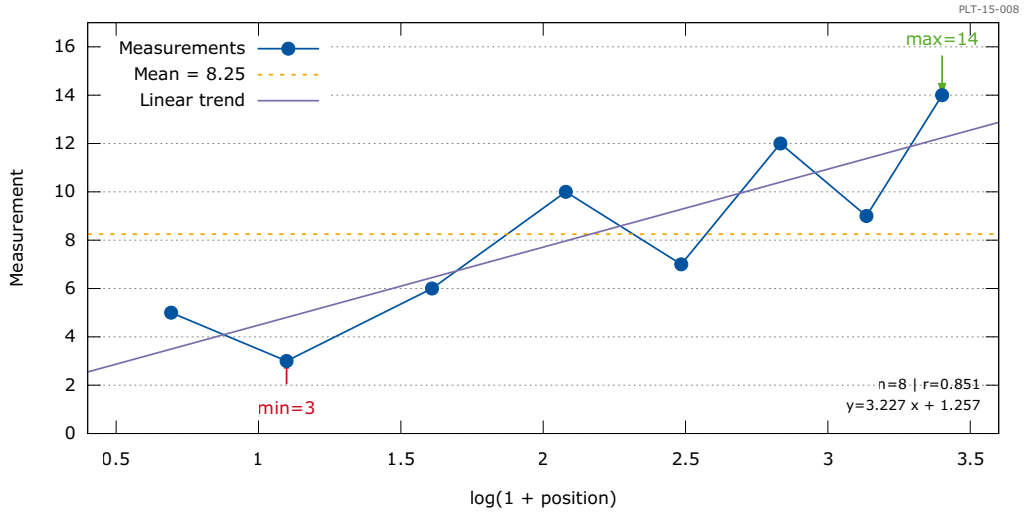
`C_slope; C_intercept`	Coefficients of the straight line for the analyzed x and y values.
`trend(x) = C_slope*x + C_intercept`	Define the line from those two coefficients.
`trend(x) with lines`	Overlay the trend without a separate nonlinear fit command.

Show the computed summary beside the plot

Compare with PLT-15-007 · Draw the linear trend from two-column statistics.

Finish with a compact annotation showing the record count, correlation and trend equation. sprintf formats the computed values rather than hard-coding them. Correlation describes the association in the transformed coordinates; these synthetic data do not establish a causal relationship.

```
f(x) = log(1+x)
stats $D using (f($1)):2 name "C" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "log(1 + position)"
set yrange [0:17]
set xrange [0.4:3.6]
set arrow 1 from first C_pos_min_y, graph 0.12 \
    to first C_pos_min_y, first C_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",C_min_y) \
    at first C_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first C_pos_max_y, graph 0.92 \
    to first C_pos_max_y, first C_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",C_max_y) \
    at first C_pos_max_y, graph 0.96 center tc rgb "#57AB27"
trend(x) = C_slope*x + C_intercept
set label 3 sprintf("n=%d | r=%.3f\ny=%.3f x + %.3f", \
    C_records,C_correlation,C_slope,C_intercept) \
    at graph 0.98,0.12 right front font ",16"
plot $D using (f($1)):2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     C_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",C_mean_y), \
     trend(x) with lines lw 2 lc rgb "#7A6FAC" title "Linear trend"
print C_records,C_mean_y,C_min_y,C_max_y, \
      C_pos_min_y,C_pos_max_y
print C_slope,C_intercept,C_correlation
```



PLT-15-008 · Generated from the complete Bash block

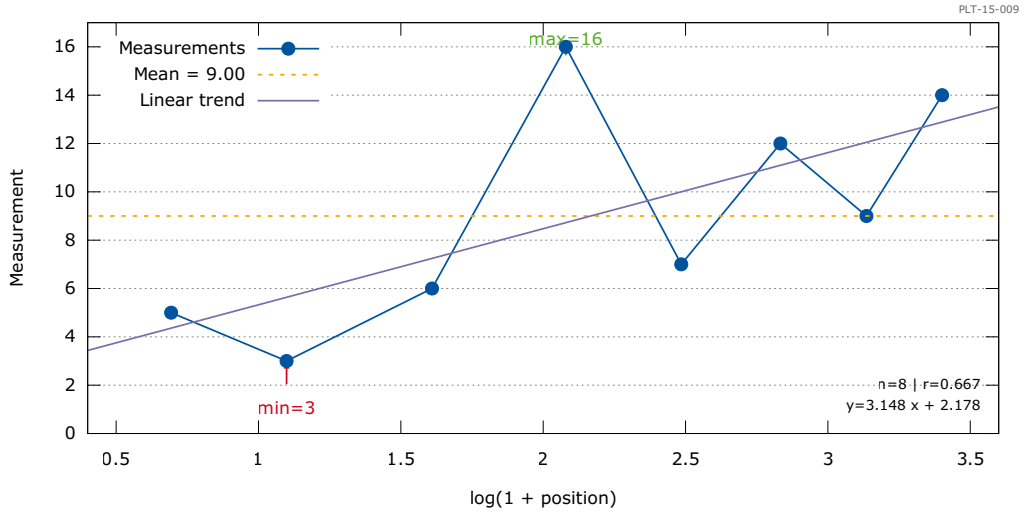
`C_correlation`	Correlation between the analyzed transformed x values and y values.
`sprintf("...", ...)`	Insert the computed values with controlled decimal precision.
`at graph 0.98,0.12 right`	Place the summary inside the plotting area near the lower-right corner.

Move the maximum by changing only the data

Compare with PLT-15-008 · Show the computed summary beside the plot.

Change just the input row from 7 10 to 7 16. The maximum moves from measurement 14 at position 29 to measurement 16 at position 7. All statistics, arrows and labels are recalculated from the data; no annotation coordinates are edited. Expand the complete Bash block to see the changed row.

```
# Change only the data row: 7 10 -> 7 16.
f(x) = log(1+x)
stats $D using (f($1)):2 name "C" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "log(1 + position)"
set yrange [0:17]
set xrange [0.4:3.6]
set arrow 1 from first C_pos_min_y, graph 0.12 \
    to first C_pos_min_y, first C_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",C_min_y) \
    at first C_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first C_pos_max_y, graph 0.92 \
    to first C_pos_max_y, first C_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",C_max_y) \
    at first C_pos_max_y, graph 0.96 center tc rgb "#57AB27"
trend(x) = C_slope*x + C_intercept
set label 3 sprintf("n=%d | r=%.3f\ny=%.3f x + %.3f", \
    C_records,C_correlation,C_slope,C_intercept) \
    at graph 0.98,0.12 right front font ",16"
plot $D using (f($1)):2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     C_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",C_mean_y), \
     trend(x) with lines lw 2 lc rgb "#7A6FAC" title "Linear trend"
print C_records,C_mean_y,C_min_y,C_max_y, \
      C_pos_min_y,C_pos_max_y
print C_slope,C_intercept,C_correlation
```



PLT-15-009 · Generated from the complete Bash block

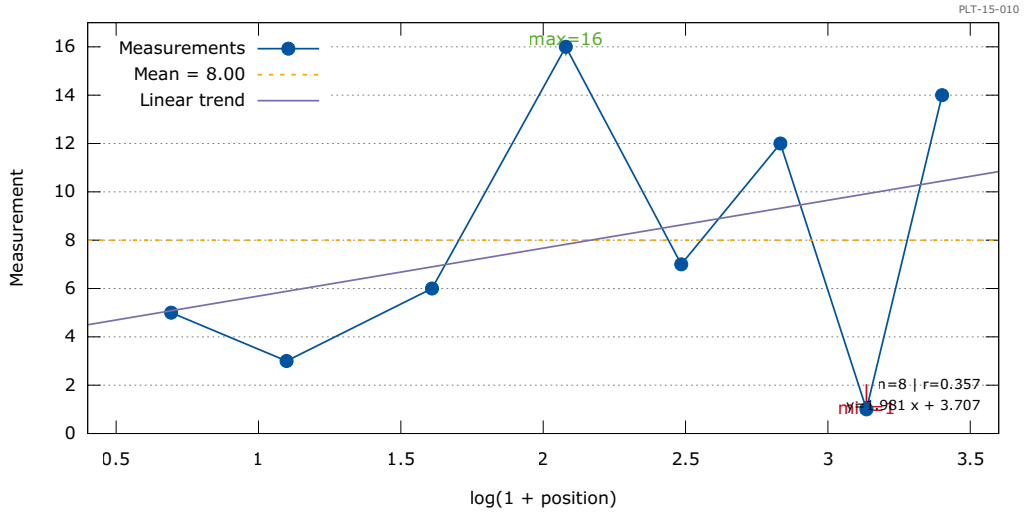
`7 10 -> 7 16`	Only this input row changes from the comparison example.
`stats $D using (f($1)):2`	Recompute summaries from the actual plotted coordinates.
`C_pos_min_y; C_pos_max_y`	Find the x positions of the new extrema automatically.
`C_min_y; C_max_y`	Both arrow endpoints and label values come from statistics.

Move the minimum without editing annotations

Compare with PLT-15-009 · Move the maximum by changing only the data.

Keep the new maximum and change just the input row from 22 9 to 22 1. The minimum now moves from measurement 3 at position 2 to measurement 1 at position 22. The same stats command finds both extrema and their transformed x positions automatically. The mean, trend and summary update too.

```
# Change only the data row: 22 9 -> 22 1.
f(x) = log(1+x)
stats $D using (f($1)):2 name "C" nooutput
set grid ytics
set key left top
set ylabel "Measurement"
set xlabel "log(1 + position)"
set yrange [0:17]
set xrange [0.4:3.6]
set arrow 1 from first C_pos_min_y, graph 0.12 \
    to first C_pos_min_y, first C_min_y filled head \
    lw 2 lc rgb "#CC071E"
set label 1 sprintf("min=%.0f",C_min_y) \
    at first C_pos_min_y, graph 0.06 center tc rgb "#CC071E"
set arrow 2 from first C_pos_max_y, graph 0.92 \
    to first C_pos_max_y, first C_max_y filled head \
    lw 2 lc rgb "#57AB27"
set label 2 sprintf("max=%.0f",C_max_y) \
    at first C_pos_max_y, graph 0.96 center tc rgb "#57AB27"
trend(x) = C_slope*x + C_intercept
set label 3 sprintf("n=%d | r=%.3f\ny=%.3f x + %.3f", \
    C_records,C_correlation,C_slope,C_intercept) \
    at graph 0.98,0.12 right front font ",16"
plot $D using (f($1)):2 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Measurements", \
     C_mean_y with lines dt 2 lw 2 lc rgb "#F6A800" title sprintf("Mean =
%.2f",C_mean_y), \
     trend(x) with lines lw 2 lc rgb "#7A6FAC" title "Linear trend"
print C_records,C_mean_y,C_min_y,C_max_y, \
      C_pos_min_y,C_pos_max_y
print C_slope,C_intercept,C_correlation
```



PLT-15-010 · Generated from the complete Bash block

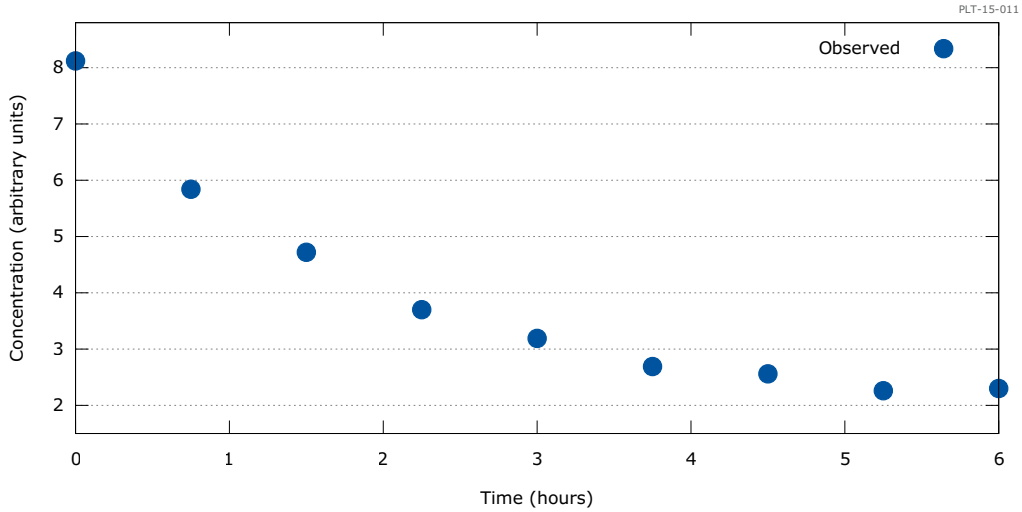
`22 9 -> 22 1`	Only this input row changes from the comparison example.
`stats $D using (f($1)):2`	Recompute summaries from the actual plotted coordinates.
`C_pos_min_y; C_pos_max_y`	Find the x positions of the new extrema automatically.
`C_min_y; C_max_y`	Both arrow endpoints and label values come from statistics.

Begin with observations for an exponential fit

New baseline: the commands below define this example.

Nine original measurements suggest a rapid decline towards a nonzero floor. Inspect the observations before choosing a model; every later step repeats this fixture in the complete Bash script.

```
$Decay << EOD
0 8.12
0.75 5.84
1.5 4.72
2.25 3.70
3 3.19
3.75 2.69
4.5 2.56
5.25 2.26
6 2.30
EOD
set xrange [0:6]
set yrange [1.5:8.8]
set xlabel "Time (hours)"
set ylabel "Concentration (arbitrary units)"
set grid ytics
set key top right
plot $Decay using 1:2 with points pt 7 ps 1.4 lc rgb "#00549F" title
"Observed"
```



PLT-15-011 · Generated from the complete Bash block

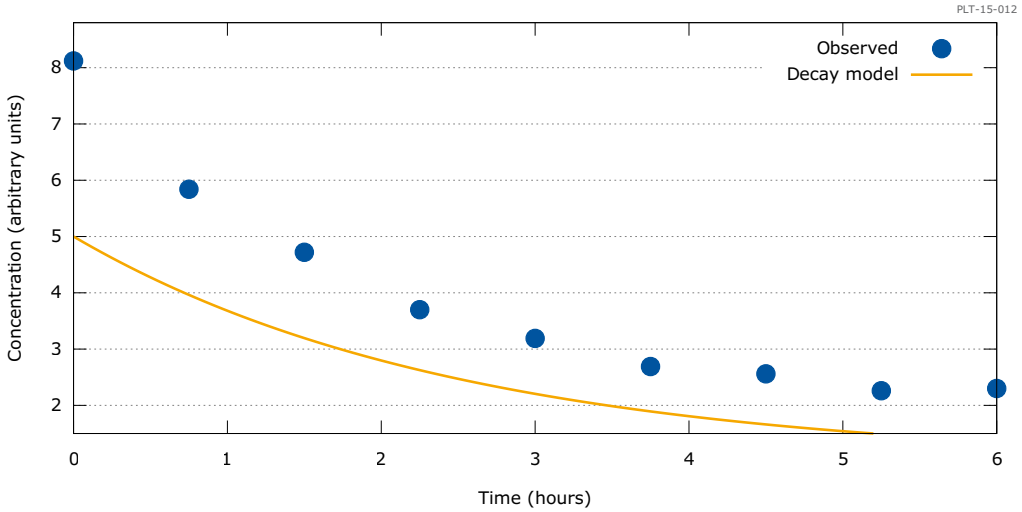
`$Decay`	An inline two-column table of elapsed time and concentration.
`using 1:2`	Read elapsed time and the measured response.

Overlay an initial decay model

Compare with PLT-15-011 · Begin with observations for an exponential fit.

Add an exponential decay with an offset. The orange curve uses deliberately imperfect starting values; these are parameters to estimate, not measured facts.

```
set xrange [0:6]
set yrange [1.5:8.8]
set xlabel "Time (hours)"
set ylabel "Concentration (arbitrary units)"
set grid ytics
set key top right
a=4; b=0.4; c=1
f(x)=a*exp(-b*x)+c
plot $Decay using 1:2 with points pt 7 ps 1.4 lc rgb "#00549F" title
"Observed", \
     f(x) with lines lw 3 lc rgb "#F6A800" title "Decay model"
```



PLT-15-012 · Generated from the complete Bash block

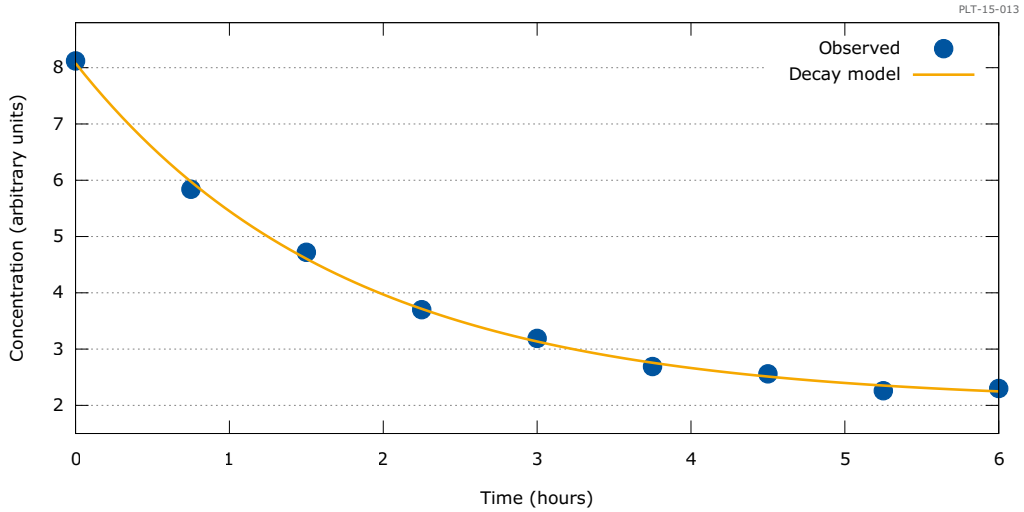
`a*exp(-b*x)+c`	Amplitude a, positive decay rate b and asymptotic floor c.
`a=4; b=0.4; c=1`	Starting guesses; sensible scales help convergence.

Let fit estimate the decay parameters

Compare with PLT-15-012 · Overlay an initial decay model.

Fit the same model to the observations with equal weights. gnuplot replaces a, b and c with least-squares estimates. Iterations are bounded, and the example reports whether optimisation converged. Convergence alone does not establish model suitability.

```
set xrange [0:6]
set yrange [1.5:8.8]
set xlabel "Time (hours)"
set ylabel "Concentration (arbitrary units)"
set grid ytics
set key top right
a=4; b=0.4; c=1
f(x)=a*exp(-b*x)+c
set fit nolog quiet maxiter 100 errorvariables
fit f(x) $Decay using 1:2 via a,b,c
print sprintf("fit: a=%.6f b=%.6f c=%.6f converged=%d", a,b,c,FIT_CONVERGED)
plot $Decay using 1:2 with points pt 7 ps 1.4 lc rgb "#00549F" title
"Observed", \
     f(x) with lines lw 3 lc rgb "#F6A800" title "Decay model"
```



PLT-15-013 · Generated from the complete Bash block

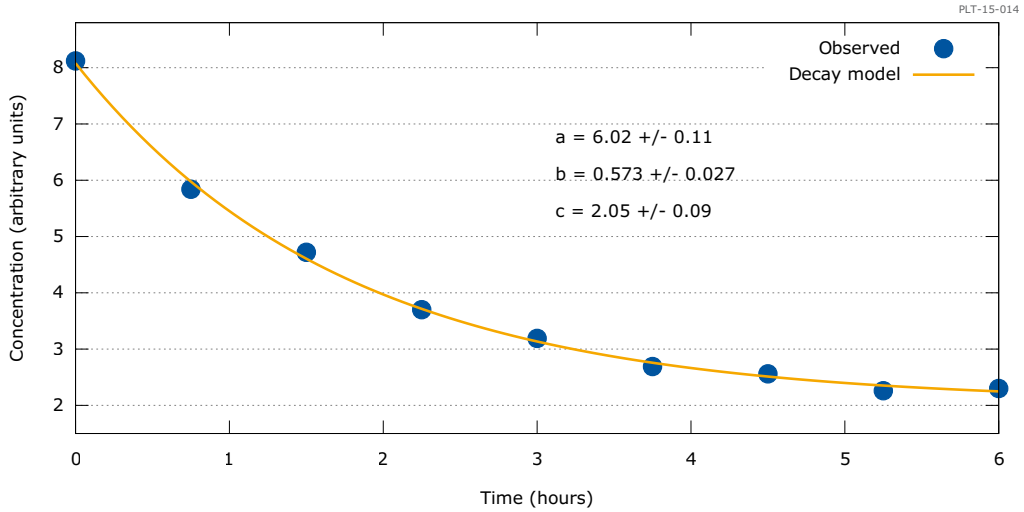
`fit ... via a,b,c`	Adjust these variables to minimise squared residuals.
`maxiter 100; nolog`	Bound iterations and avoid creating fit.log.
`FIT_CONVERGED`	Nonzero means the stopping criterion was reached.

Display fitted values and their uncertainties

Compare with PLT-15-013 · Let fit estimate the decay parameters.

Keep the fitted curve and add parameter values. The plus/minus values are estimated parameter standard errors; they are not prediction intervals or a guarantee that exponential decay is the correct model.

```
set xrange [0:6]
set yrange [1.5:8.8]
set xlabel "Time (hours)"
set ylabel "Concentration (arbitrary units)"
set grid ytics
set key top right
a=4; b=0.4; c=1
f(x)=a*exp(-b*x)+c
set fit nolog quiet maxiter 100 errorvariables
fit f(x) $Decay using 1:2 via a,b,c
print sprintf("fit: a=%.6f b=%.6f c=%.6f converged=%d", a,b,c,FIT_CONVERGED)
set label 1 sprintf("a = %.2f +/- %.2f",a,a_err) at graph 0.52,0.72
set label 2 sprintf("b = %.3f +/- %.3f",b,b_err) at graph 0.52,0.63
set label 3 sprintf("c = %.2f +/- %.2f",c,c_err) at graph 0.52,0.54
plot $Decay using 1:2 with points pt 7 ps 1.4 lc rgb "#00549F" title
"Observed", \
     f(x) with lines lw 3 lc rgb "#F6A800" title "Decay model"
```



PLT-15-014 · Generated from the complete Bash block

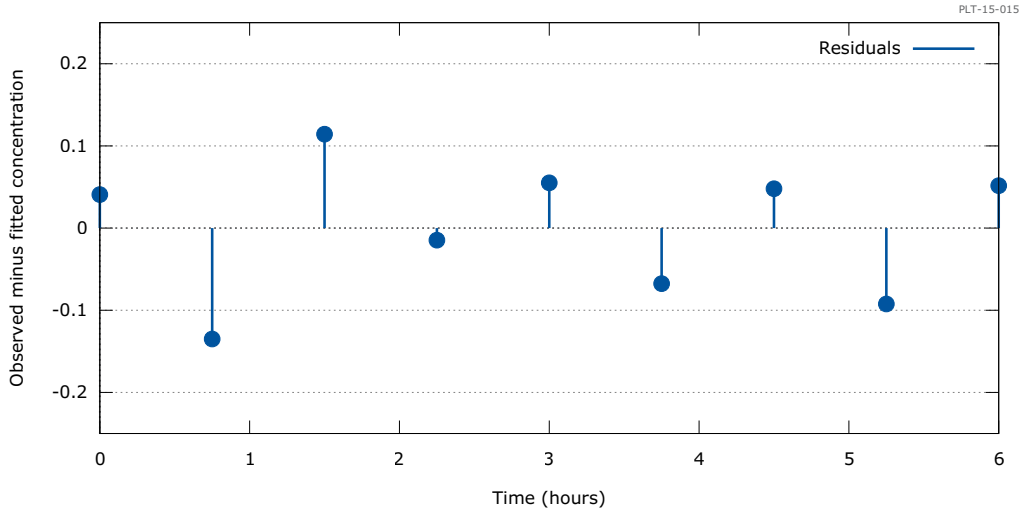
`errorvariables`	Create a_err, b_err and c_err after fitting.
`sprintf`	Format computed values rather than hard-coding labels.

Inspect residuals after fitting

Compare with PLT-15-013 · Let fit estimate the decay parameters.

Plot measured minus fitted values around a horizontal zero line. A fitted curve can hide systematic errors; residual patterns help reveal them. This small synthetic fixture demonstrates the method rather than validating a statistical model.

```
set xrange [0:6]
set yrange [1.5:8.8]
set xlabel "Time (hours)"
set ylabel "Concentration (arbitrary units)"
set grid ytics
set key top right
a=4; b=0.4; c=1
f(x)=a*exp(-b*x)+c
set fit nolog quiet maxiter 100 errorvariables
fit f(x) $Decay using 1:2 via a,b,c
print sprintf("fit: a=%.6f b=%.6f c=%.6f converged=%d", a,b,c,FIT_CONVERGED)
set yrange [-0.25:0.25]
set ylabel "Observed minus fitted concentration"
set zeroaxis lw 2 lc rgb "#646567"
plot $Decay using 1:($2-f($1)) with impulses lw 3 \
    lc rgb "#00549F" title "Residuals", \
    $Decay using 1:($2-f($1)) with points pt 7 ps 1.2 \
    lc rgb "#00549F" notitle
```



PLT-15-015 · Generated from the complete Bash block

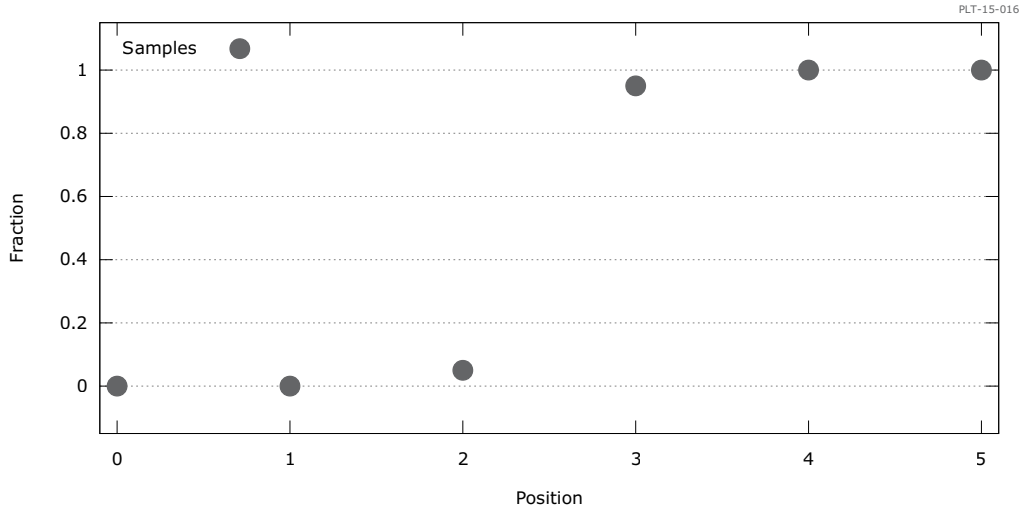
`$2-f($1)`	Compute observed minus fitted concentration.
`set zeroaxis`	Draw a reference at zero; residuals can have either sign.

Start an interpolation with six samples

New baseline: the commands below define this example.

These original fractions begin almost flat, jump between positions two and three, then level off. Begin with the observations. Interpolation connects known values; it differs from fitting a noisy response.

```
$Ramp << EOD
0 0
1 0
2 0.05
3 0.95
4 1
5 1
EOD
set xrange [-0.1:5.1]
set yrange [-0.15:1.15]
set xlabel "Position"
set ylabel "Fraction"
set grid ytics
set key top left
set samples 250
plot $Ramp using 1:2 with points pt 7 ps 1.5 lc rgb "#646567" title "Samples"
```



PLT-15-016 · Generated from the complete Bash block

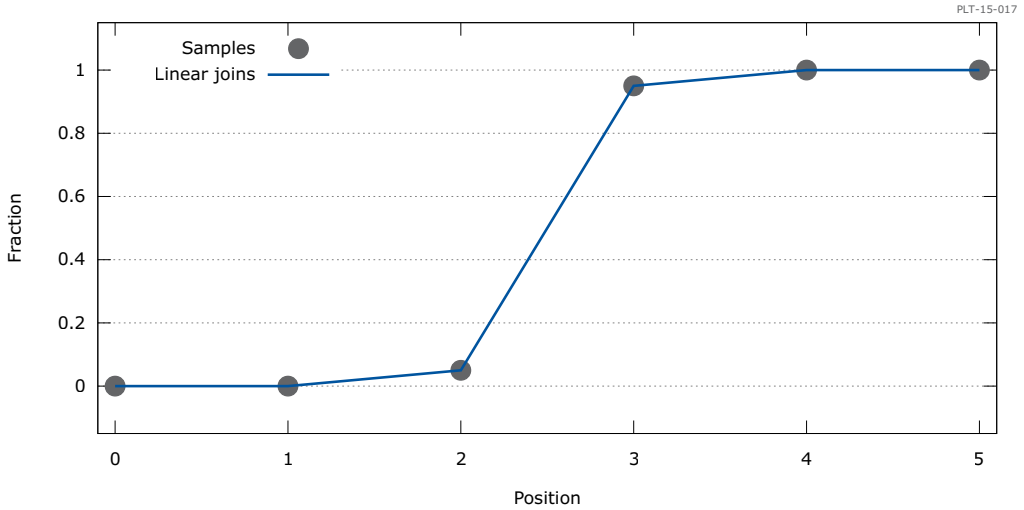
`$Ramp`	An ordered, monotone fixture with a sharp transition.
`with points`	Show only values supplied in the data.

Connect samples with straight lines

Compare with PLT-15-016 · Start an interpolation with six samples.

Add straight segments between consecutive measurements. The joins pass through every observation, but their slope changes abruptly at each sample position.

```
set xrange [-0.1:5.1]
set yrange [-0.15:1.15]
set xlabel "Position"
set ylabel "Fraction"
set grid ytics
set key top left
set samples 250
plot $Ramp using 1:2 with points pt 7 ps 1.5 lc rgb "#646567" title
"Samples", \
     $Ramp using 1:2 with lines lw 3 lc rgb "#00549F" title "Linear joins"
```



PLT-15-017 · Generated from the complete Bash block

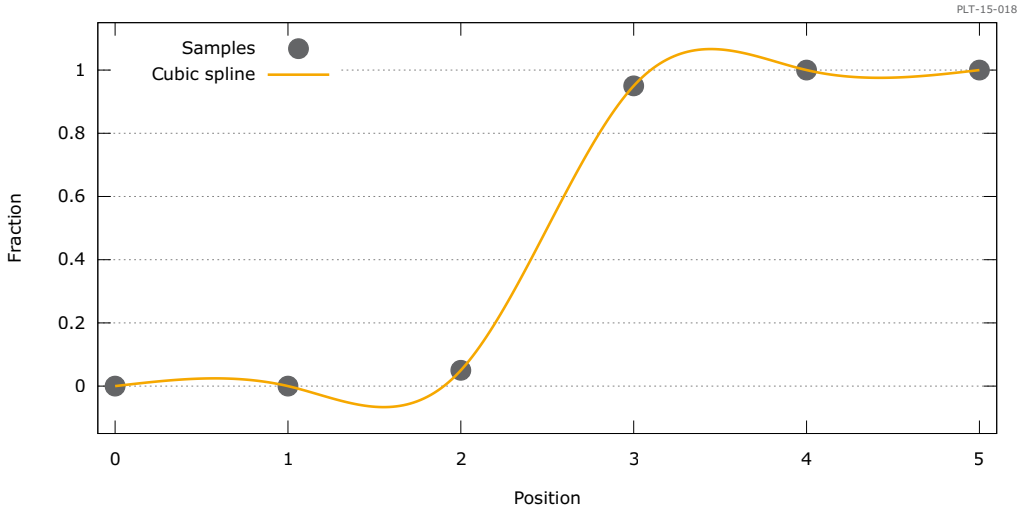
`with lines`	Connect neighbouring records without estimating coefficients.

Replace joins with a cubic spline

Compare with PLT-15-017 · Connect samples with straight lines.

A cubic spline smoothly connects the samples. Near the sharp transition it overshoots the physical fraction range: smoothness alone does not preserve the data shape.

```
set xrange [-0.1:5.1]
set yrange [-0.15:1.15]
set xlabel "Position"
set ylabel "Fraction"
set grid ytics
set key top left
set samples 250
plot $Ramp using 1:2 with points pt 7 ps 1.5 lc rgb "#646567" title
"Samples", \
     $Ramp using 1:2 smooth csplines with lines lw 3 \
     lc rgb "#F6A800" title "Cubic spline"
```



PLT-15-018 · Generated from the complete Bash block

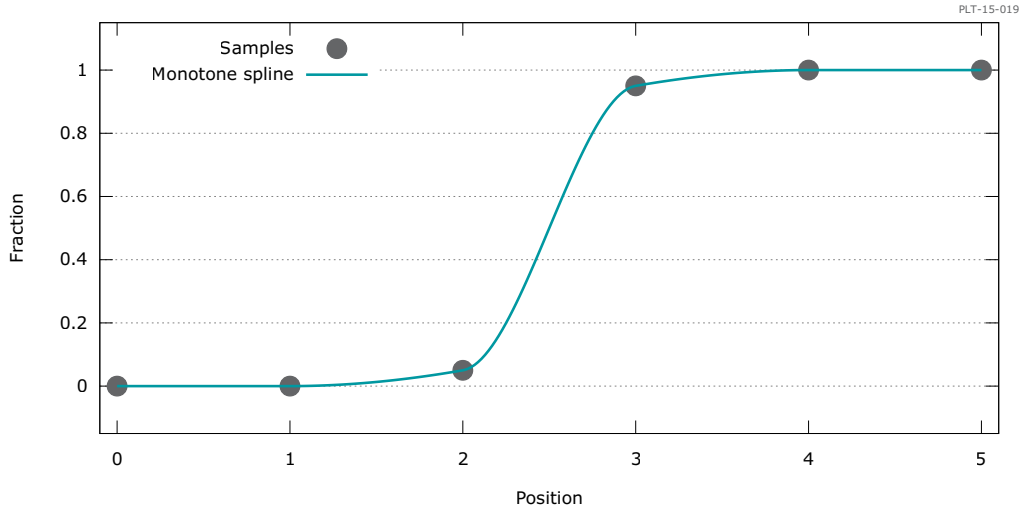
`smooth csplines`	Interpolate with cubic segments; overshoot is possible.
`samples 250`	Choose the number of points drawing the smooth curve.

Preserve monotonicity with mcsplines

Compare with PLT-15-018 · Replace joins with a cubic spline.

Change only the interpolation method. The monotone cubic spline preserves this fixture's increasing shape and stays between neighbouring values. It interpolates rather than denoising the observations.

```
set xrange [-0.1:5.1]
set yrange [-0.15:1.15]
set xlabel "Position"
set ylabel "Fraction"
set grid ytics
set key top left
set samples 250
plot $Ramp using 1:2 with points pt 7 ps 1.5 lc rgb "#646567" title
"Samples", \
     $Ramp using 1:2 smooth mcsplines with lines lw 3 \
     lc rgb "#0098A1" title "Monotone spline"
```



PLT-15-019 · Generated from the complete Bash block

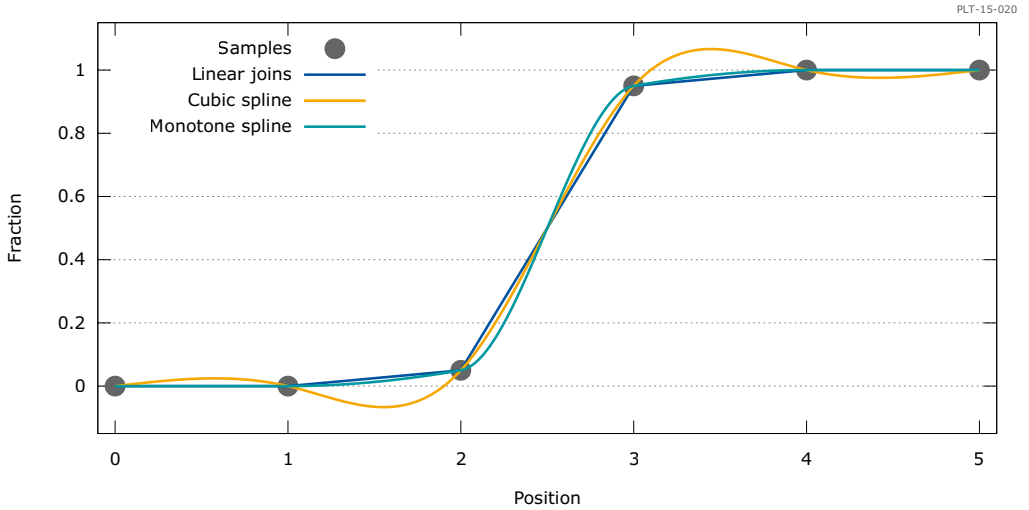
`smooth mcsplines`	Constrain the interpolated shape to preserve monotonicity.

Compare interpolation choices in one figure

Compare with PLT-15-019 · Preserve monotonicity with mcsplines.

Overlay all three methods against the same observations. Keep negative and above-one margins visible so cubic overshoot is easy to see; do not hide it with restrictive axis ranges.

```
set xrange [-0.1:5.1]
set yrange [-0.15:1.15]
set xlabel "Position"
set ylabel "Fraction"
set grid ytics
set key top left
set samples 250
plot $Ramp using 1:2 with points pt 7 ps 1.5 lc rgb "#646567" title
"Samples", \
     $Ramp using 1:2 with lines lw 3 lc rgb "#00549F" title "Linear joins", \
     $Ramp using 1:2 smooth csplines with lines lw 3 \
     lc rgb "#F6A800" title "Cubic spline", \
     $Ramp using 1:2 smooth mcsplines with lines lw 3 \
     lc rgb "#0098A1" title "Monotone spline"
```



PLT-15-020 · Generated from the complete Bash block

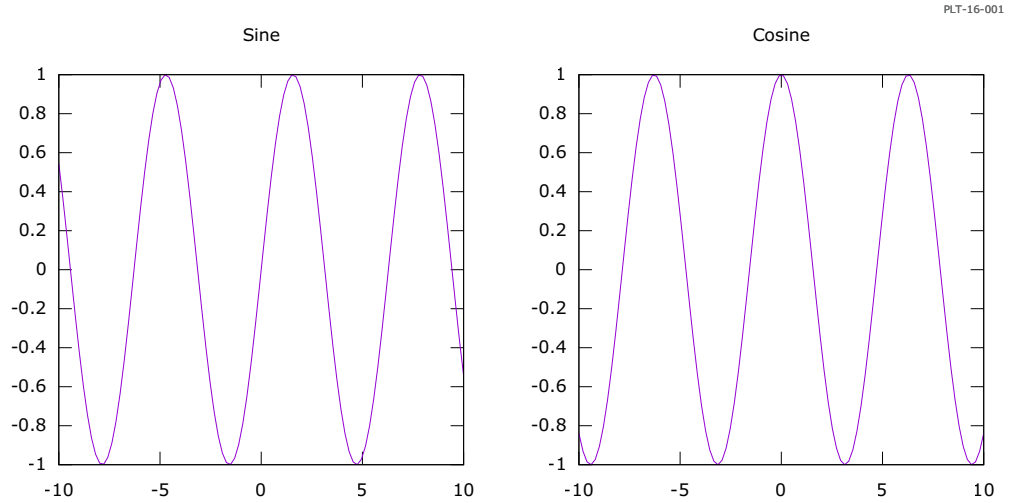
`three plot clauses`	Reuse one fixture with three interpolation choices.
`yrange [-0.15:1.15]`	Expose overshoot outside the valid fraction range.

Create two side-by-side panels

New baseline: the commands below define this example.

multiplot directs several plot commands onto one canvas.

```
set multiplot layout 1,2
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-001 · Generated from the complete Bash block

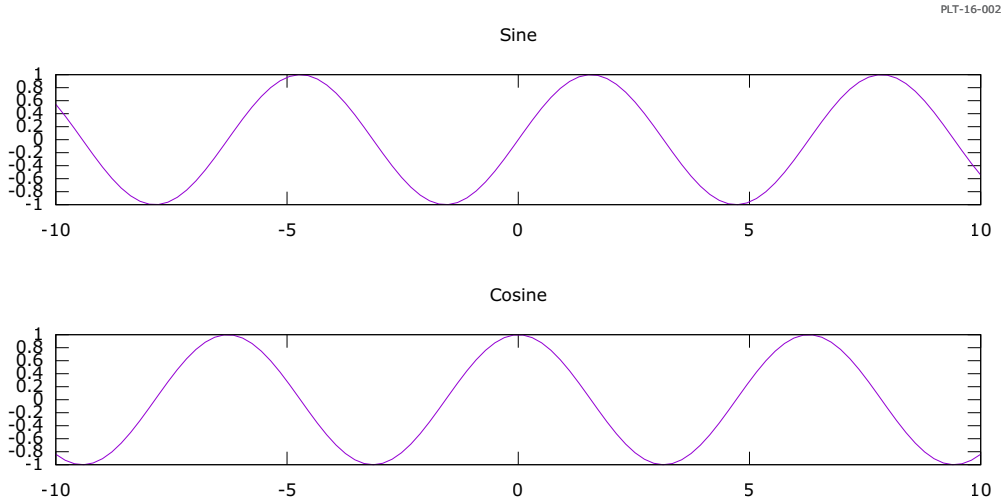
`layout 1,2`	One row and two columns.
`set title`	Set the title of the next panel.
`notitle`	Omit the curve’s legend entry.
`unset multiplot`	Finish the combined figure.

Stack the same panels vertically

Compare with PLT-16-001 · Create two side-by-side panels.

Only the row/column layout changes.

```
set multiplot layout 2,1
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-002 · Generated from the complete Bash block

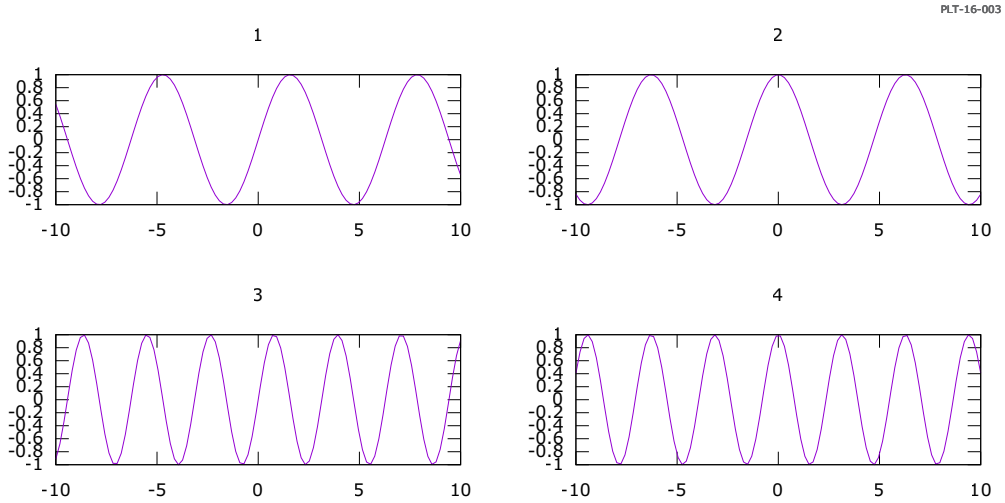
`layout 2,1`	Two rows and one column; plot commands fill the panels in order.

Use a two-by-two layout

Compare with PLT-16-001 · Create two side-by-side panels.

Each subsequent plot command occupies the next cell.

```
set multiplot layout 2,2 rowsfirst
set title "1"
plot sin(x) notitle
set title "2"
plot cos(x) notitle
set title "3"
plot sin(2*x) notitle
set title "4"
plot cos(2*x) notitle
unset multiplot
```



PLT-16-003 · Generated from the complete Bash block

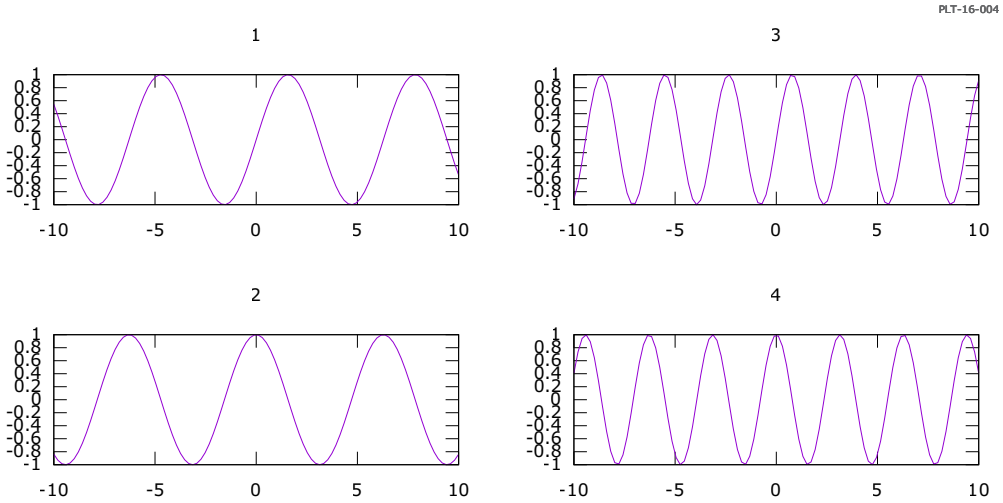
`layout 2,2`	Two rows and two columns.
`rowsfirst`	Fill left to right, then move down to the next row.

Fill panels down columns first

Compare with PLT-16-003 · Use a two-by-two layout.

The four plot commands stay in the same order; their cell positions change.

```
set multiplot layout 2,2 columnsfirst
set title "1"
plot sin(x) notitle
set title "2"
plot cos(x) notitle
set title "3"
plot sin(2*x) notitle
set title "4"
plot cos(2*x) notitle
unset multiplot
```



PLT-16-004 · Generated from the complete Bash block

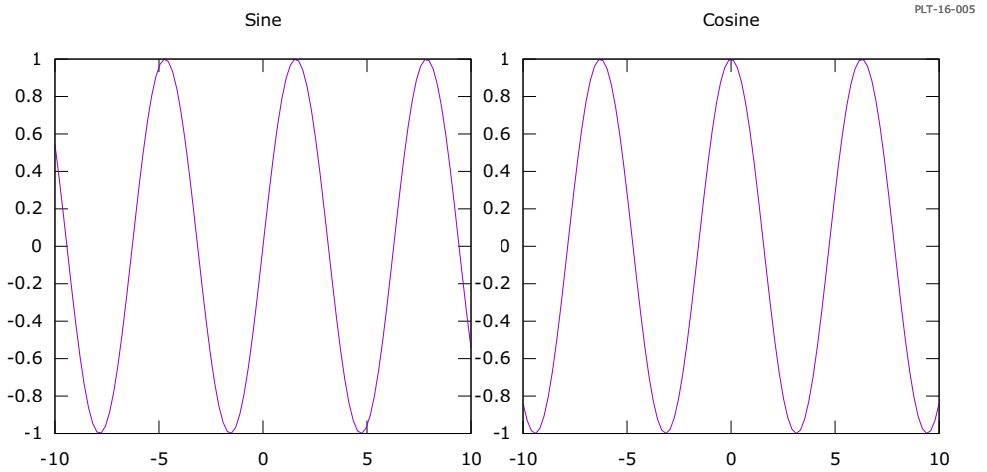
`columnsfirst`	Fill top to bottom in a column before moving right.

Specify shared outer margins

Compare with PLT-16-001 · Create two side-by-side panels.

The four margin values place the outer panel borders in screen coordinates.

```
set multiplot layout 1,2 margins 0.10,0.95,0.16,0.88
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-005 · Generated from the complete Bash block

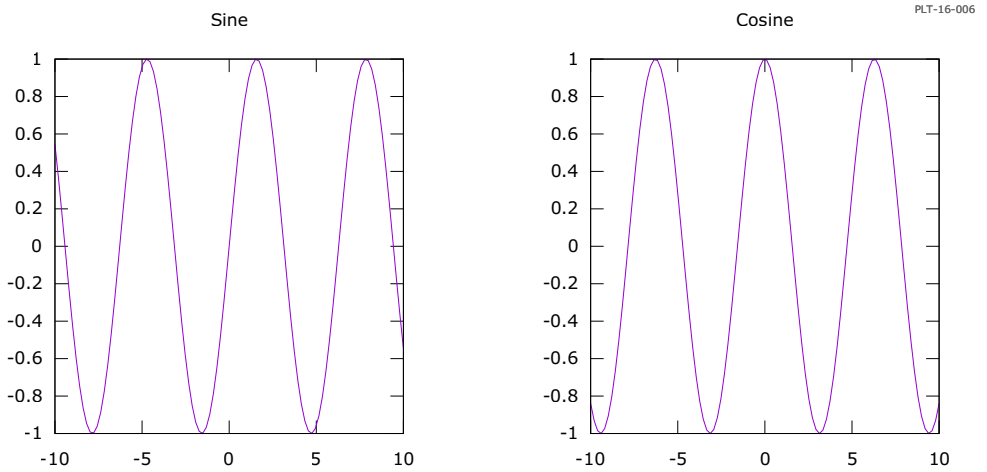
`margins 0.10,0.95,0.16,0.88`	Left, right, bottom and top positions, as canvas fractions.
`Screen fractions`	0 is the left/bottom edge; 1 is the right/top edge.

Increase the gap between columns

Compare with PLT-16-005 · Specify shared outer margins.

Spacing separates neighbouring panel borders within the chosen outer margins.

```
set multiplot layout 1,2 margins 0.10,0.95,0.16,0.88 spacing 0.18,0.08
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-006 · Generated from the complete Bash block

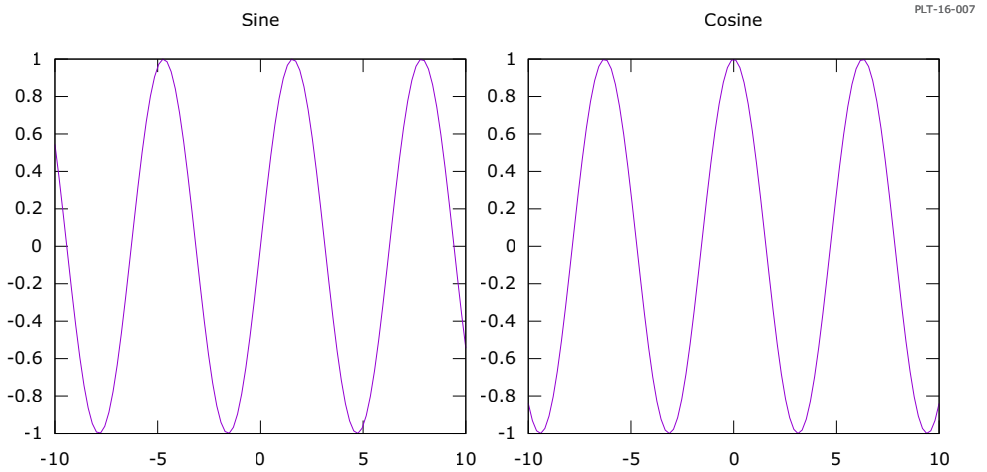
`spacing 0.18,0.08`	Horizontal and vertical gaps in screen fractions.
`0.18`	Only the horizontal gap is visible in this one-row layout.

Reduce the gap again

Compare with PLT-16-006 · Increase the gap between columns.

Change one spacing value while retaining the same canvas and outer margins.

```
set multiplot layout 1,2 margins 0.10,0.95,0.16,0.88 spacing 0.06,0.08
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-007 · Generated from the complete Bash block

`0.06`

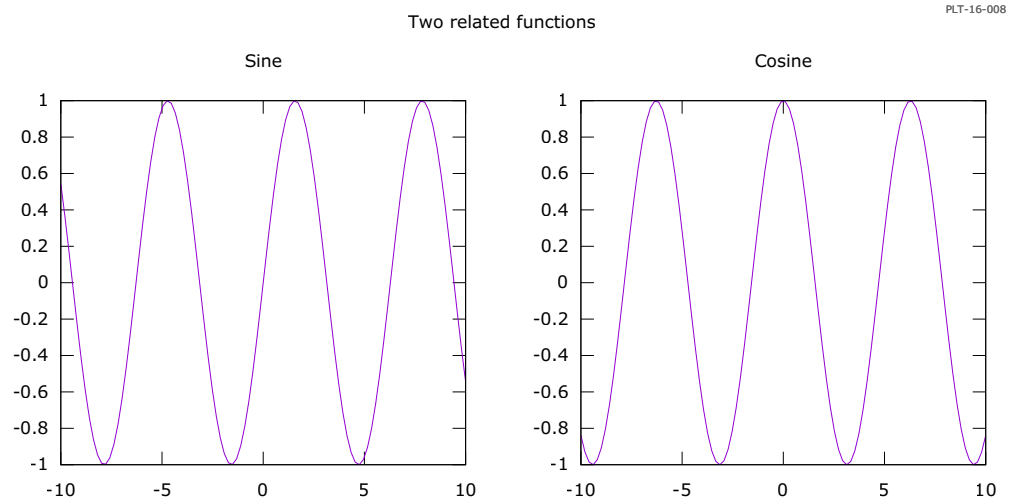
Reduce the horizontal gap from 0.18 to 0.06; vertical spacing stays 0.08.

Add a shared figure title

Compare with PLT-16-001 · Create two side-by-side panels.

A multiplot title describes the entire panel group.

```
set multiplot layout 1,2 title "Two related functions"
set title "Sine"
plot sin(x) notitle
set title "Cosine"
plot cos(x) notitle
unset multiplot
```



PLT-16-008 · Generated from the complete Bash block

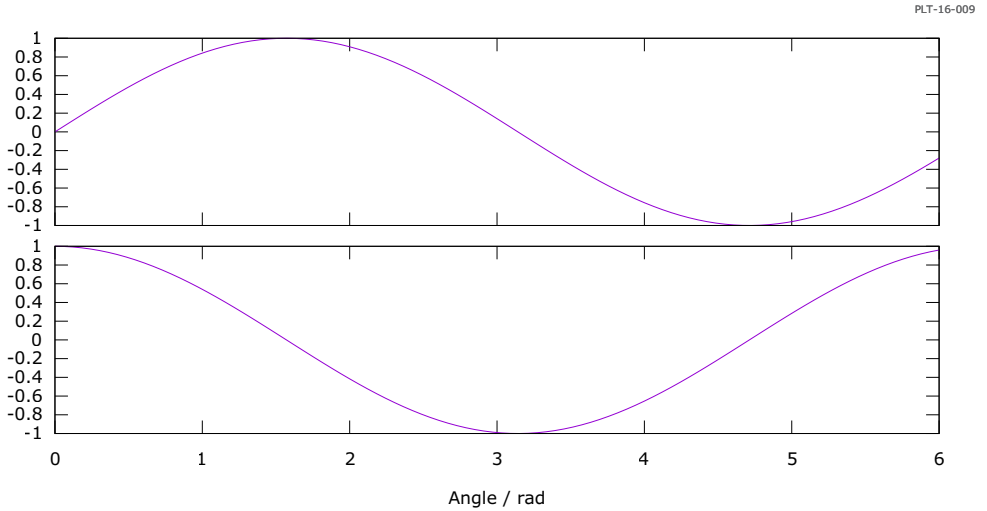
`title "Two related functions"`	The title belongs to the multiplot command, so it spans the whole figure.

Share an x scale across stacked panels

Compare with PLT-16-002 · Stack the same panels vertically.

Suppress repeated x labels in the upper panel and restore them for the lower one. Explicit outer margins align the borders.

```
set xrange [0:6]
set multiplot layout 2,1 \
    margins 0.10,0.95,0.16,0.92 spacing 0.04,0.04
set format x ""
plot sin(x) notitle
set format x "%g"
set xlabel "Angle / rad"
plot cos(x) notitle
unset multiplot
```



PLT-16-009 · Generated from the complete Bash block

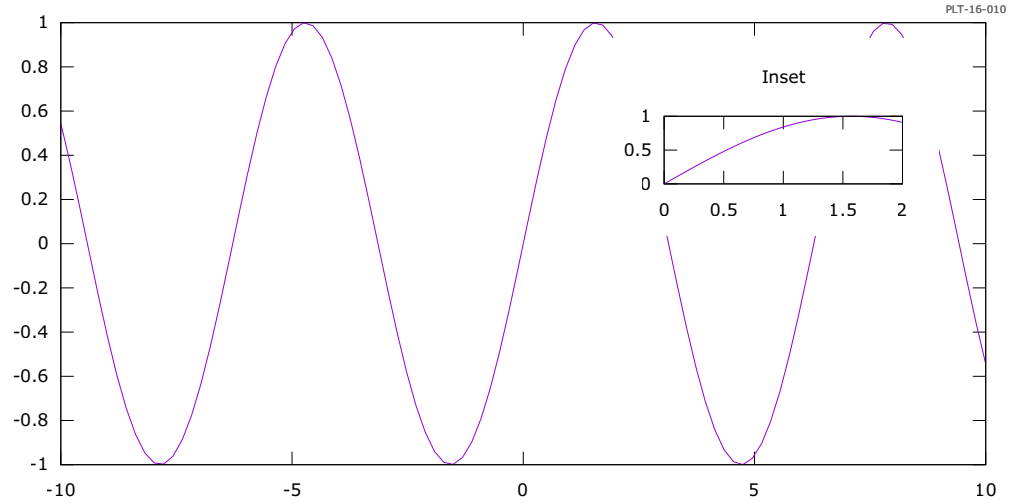
`format x ""`	Hide only the upper panel’s x tick text.
`format x "%g"`	Restore compact numeric labels for the lower panel.
`margins; spacing`	Shared outer borders and a small vertical gap.
`xlabel`	Place a descriptive x label on the bottom panel only.

Create an inset manually

Compare with PLT-16-001 · Create two side-by-side panels.

set size and set origin position each plot within a common canvas. Restore settings explicitly when adding more panels.

```
set multiplot
set size 1,1
set origin 0,0
plot sin(x) notitle
set size 0.34,0.38
set origin 0.58,0.54
set object 1 rectangle from screen 0.58,0.54 \
    to screen 0.92,0.92 behind fillcolor rgb "white" \
    fillstyle solid 1 noborder
set xrange [0:2]
set ytics 0.5
set title "Inset"
plot sin(x) notitle
unset multiplot
```



PLT-16-010 · Generated from the complete Bash block

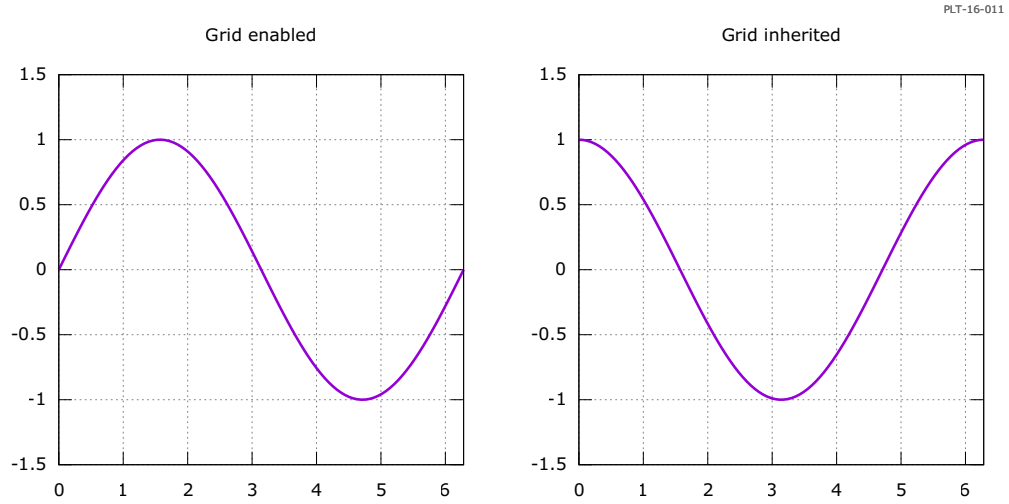
`set multiplot`	Begin manual panel placement without an automatic layout.
`size 1,1; origin 0,0`	The first plot uses the full canvas.
`size 0.34,0.38`	Inset width and height as fractions of the canvas.
`origin 0.58,0.54`	Inset's lower-left location in screen fractions.
`object ... behind; ytics 0.5`	A white background covers the main plot beneath the inset; half-unit y ticks avoid crowded labels.

See settings persist between panels

New baseline: the commands below define this example.

A new plot command does not start a new session. Multiplot advances the panel position while retaining settings such as ranges, grids, styles and annotations.

```
set multiplot layout 1,2
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set grid
set title "Grid enabled"
plot sin(x) with lines lw 3 notitle
set title "Grid inherited"
plot cos(x) with lines lw 3 notitle
unset multiplot
```



PLT-16-011 · Generated from the complete Bash block

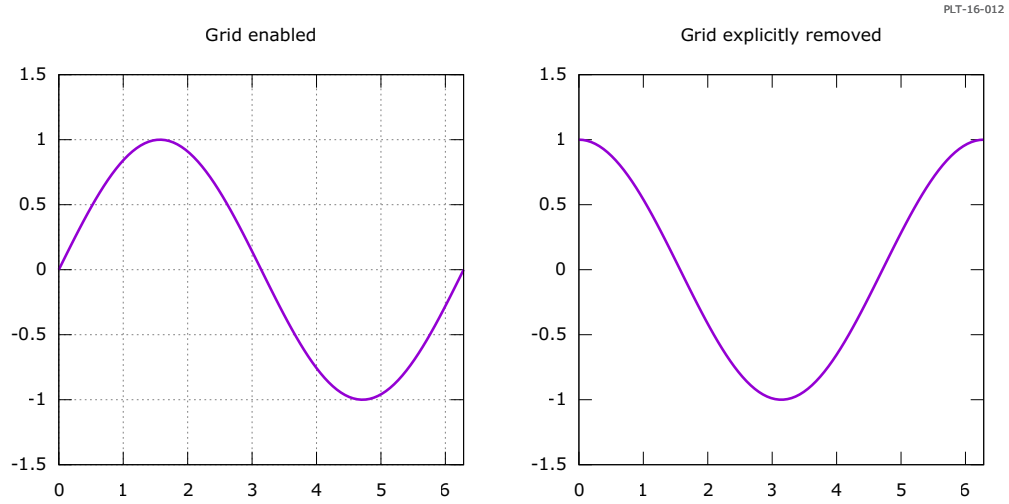
`second plot`	Inherits the grid and ranges from the first panel.
`layout`	Moves the panel; it does not reset plotting settings.

Clear a setting before the next panel

Compare with PLT-16-011 · See settings persist between panels.

Unset individual settings when only one panel needs them. This retains the shared ranges and other formatting while removing the grid from the second panel.

```
set multiplot layout 1,2
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set grid
set title "Grid enabled"
plot sin(x) with lines lw 3 notitle
unset grid
set title "Grid explicitly removed"
plot cos(x) with lines lw 3 notitle
unset multiplot
```



PLT-16-012 · Generated from the complete Bash block

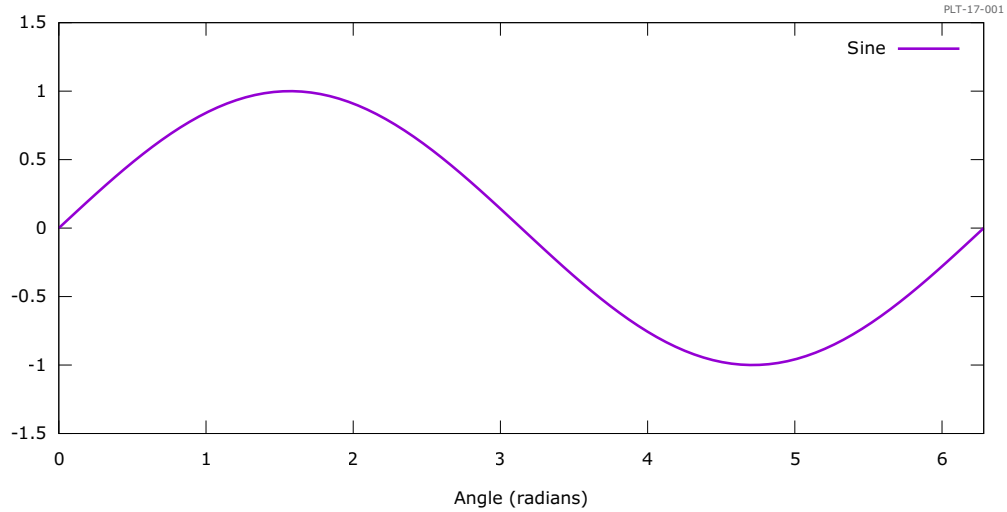
`unset grid`	Disable the inherited grid before plotting the next panel.
`fresh gnuplot process`	Each complete tour example starts independently; its panels still share one process.

Write an SVG with explicit terminal settings

New baseline: the commands below define this example.

Choose the terminal before the output filename. SVG is vector output; its nominal pixel dimensions define the canvas and its aspect ratio, while the paths remain scalable.

```
set terminal svg size 1200,600 enhanced \
    font "sans,20" background rgb "white"
set output 'figures/PLT-17-001.svg'
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle (radians)"
plot sin(x) with lines lw 3 title "Sine"
```



PLT-17-001 · Generated from the complete Bash block

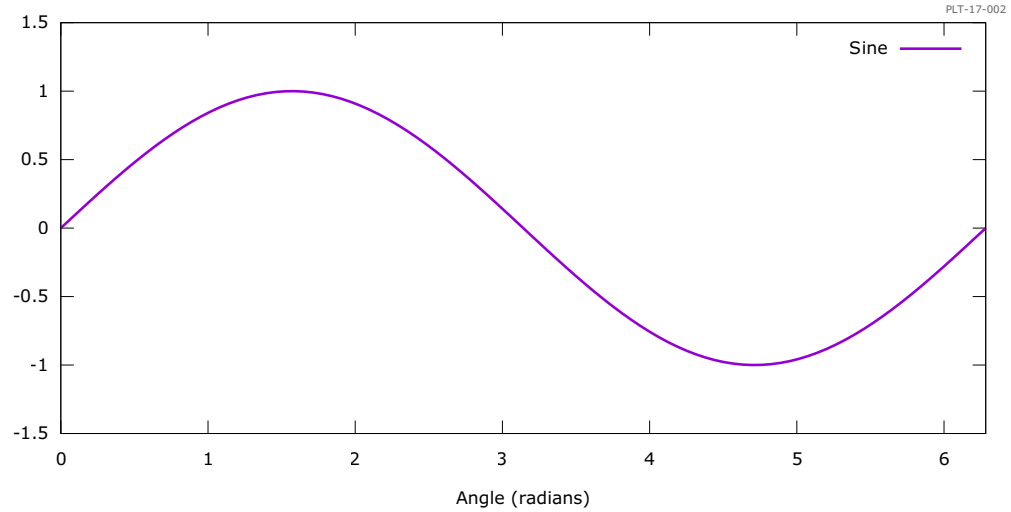
`terminal svg size 1200,600`	Choose a vector SVG canvas with a 2:1 aspect ratio.
`font "sans,20"`	Set terminal-specific font family and nominal size.
`set output`	Open the file after selecting its terminal.
`unset output`	Close and finish the file; supplied by the complete Bash wrapper.

Export the same plot as a PNG image

Compare with PLT-17-001 · Write an SVG with explicit terminal settings.

Pngcairo writes a raster image at the requested pixel resolution. Terminal push/pop saves and restores the SVG settings for the guide preview; the PNG is also written to figures/PLT-17-002.png.

```
set term push
set terminal pngcairo size 1200,600 enhanced font "sans,20"
set output 'figures/PLT-17-002.png'
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle (radians)"
plot sin(x) with lines lw 3 title "Sine"
unset output
set term pop
set output 'figures/PLT-17-002.svg'
replot
```



PLT-17-002 · Generated from the complete Bash block

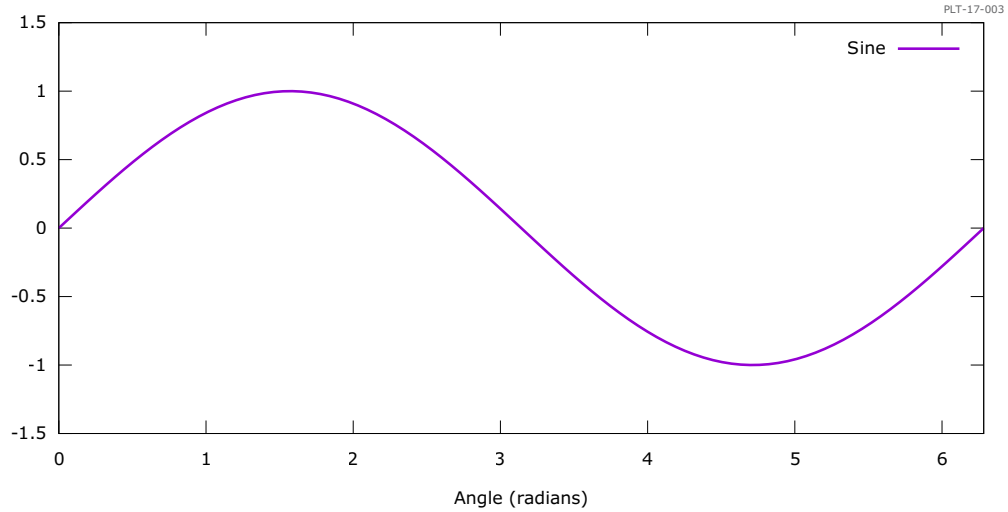
`pngcairo size 1200,600`	Write 1200 by 600 raster pixels; requires Cairo terminal support.
`term push; term pop`	Save and restore terminal settings; filenames must be selected separately.
`replot`	Render the preceding plot command again with the restored terminal.
`font sizes`	SVG and Cairo font metrics may differ even with the same nominal size.

Export a PDF at a physical page size

Compare with PLT-17-002 · Export the same plot as a PNG image.

Pdfcairo uses physical dimensions rather than raster pixels. Its font sizes are in points. Choose a suitable physical page size for a plot intended for print.

```
set term push
set terminal pdfcairo size 12cm,6cm enhanced font "sans,10"
set output 'figures/PLT-17-003.pdf'
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle (radians)"
plot sin(x) with lines lw 3 title "Sine"
unset output
set term pop
set output 'figures/PLT-17-003.svg'
replot
```



PLT-17-003 · Generated from the complete Bash block

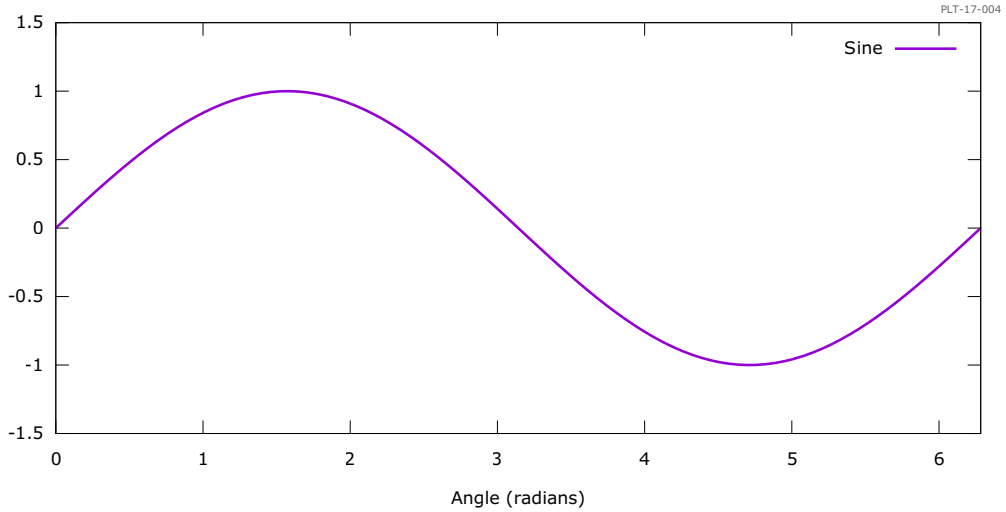
`pdfcairo size 12cm,6cm`	Write a vector PDF page twelve centimetres wide and six high.
`font "sans,10"`	Use a ten-point Cairo font for this smaller physical page.
`terminal before output`	Close the previous output before restoring the terminal and opening the next file.

Export a PNG with a transparent background

Compare with PLT-17-002 · Export the same plot as a PNG image.

Transparent output is useful when placing a plot over a coloured page or slide. It affects the canvas background; transparency of curve fills is a separate fillstyle setting.

```
set term push
set terminal pngcairo transparent size 1200,600 \
    enhanced font "sans,20"
set output 'figures/PLT-17-004.png'
set xrange [0:2*pi]
set yrange [-1.5:1.5]
set xlabel "Angle (radians)"
plot sin(x) with lines lw 3 title "Sine"
unset output
set term pop
set output 'figures/PLT-17-004.svg'
replot
```



PLT-17-004 · Generated from the complete Bash block

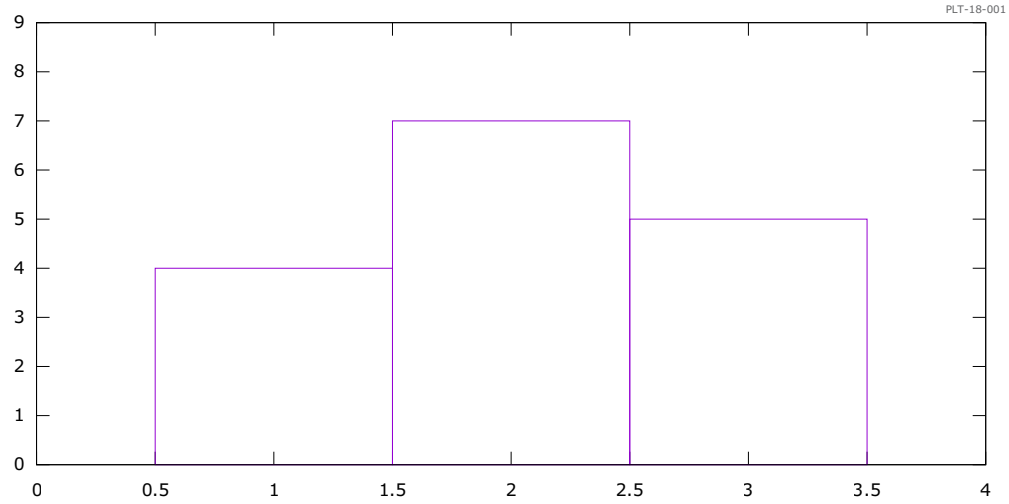
`pngcairo transparent`	Give the raster canvas an alpha channel with a transparent background.
`fillstyle transparent solid`	Controls opacity of a filled plot element independently.
`SVG preview`	The guide retains its white canvas; inspect the generated PNG for its transparent background.

Start with three bars

New baseline: the commands below define this example.

A tiny datablock keeps the data beside the drawing. Here the first column is a numeric position and the second is the bar height.

```
$D << EOD
1 4
2 7
3 5
EOD
set xrange [0:4]
set yrange [0:9]
plot $D using 1:2 with boxes notitle
```



PLT-18-001 · Generated from the complete Bash block

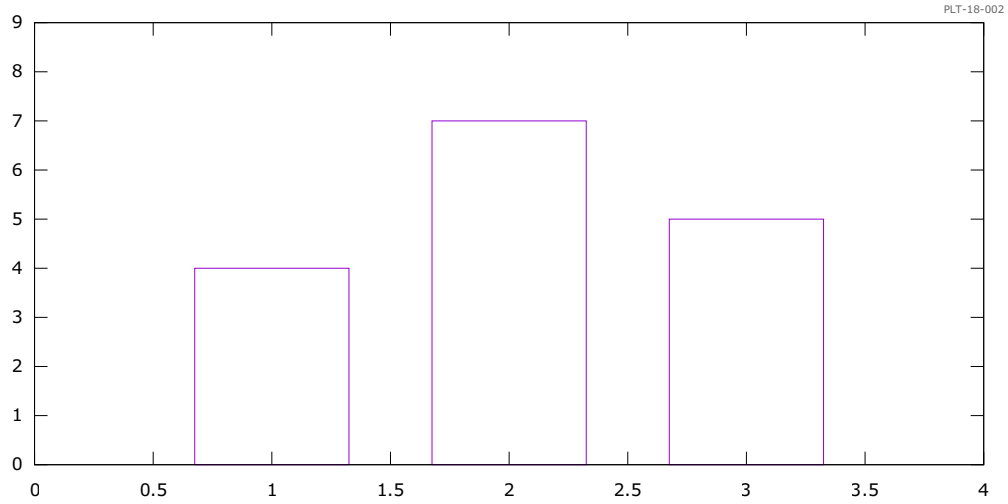
`$D << EOD`	Define an in-memory datablock; EOD alone ends it. The quoted Bash heredoc preserves $D.
`using 1:2`	Map column 1 to x and column 2 to height.
`with boxes`	Draw vertical boxes from zero to each y value; width is initially inferred.
`xrange [0:4]; yrange [0:9]`	Keep all three bars and zero visible; reserve space above the bars.

Set an exact bar width

Compare with PLT-18-001 · Start with three bars.

Make the spacing between bars deliberate.

```
$D << EOD
1 4
2 7
3 5
EOD
set xrange [0:4]
set boxwidth 0.65 absolute
set yrange [0:9]
plot $D using 1:2 with boxes notitle
```



PLT-18-002 · Generated from the complete Bash block

`0.65 absolute`

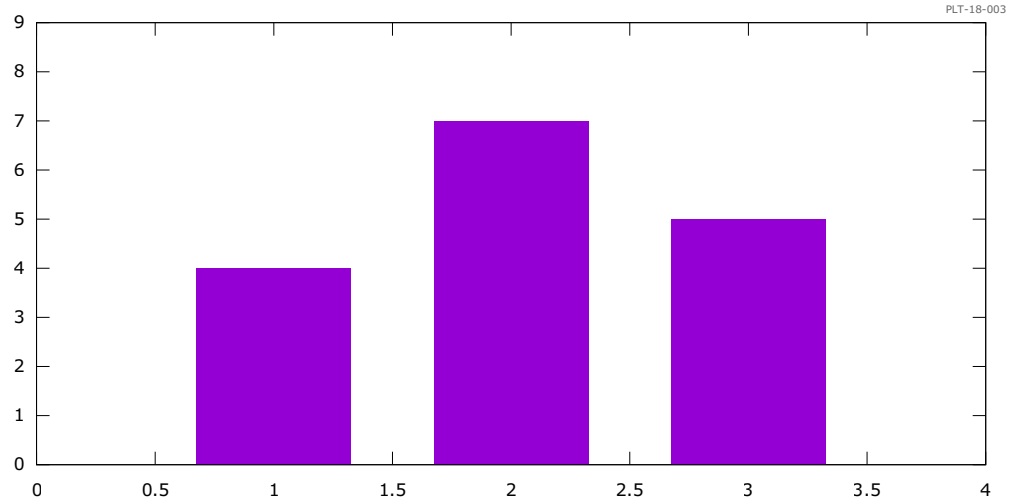
Width is 0.65 x-axis units. Positions are one unit apart, leaving a 0.35-unit gap.

Fill the bars

Compare with PLT-18-002 · Set an exact bar width.

Solid fill makes the areas readable; removing outlines changes the visual weight.

```
$D << EOD
1 4
2 7
3 5
EOD
set xrange [0:4]
set boxwidth 0.65 absolute
set yrange [0:9]
set style fill solid 1.0 noborder
plot $D using 1:2 with boxes notitle
```



PLT-18-003 · Generated from the complete Bash block

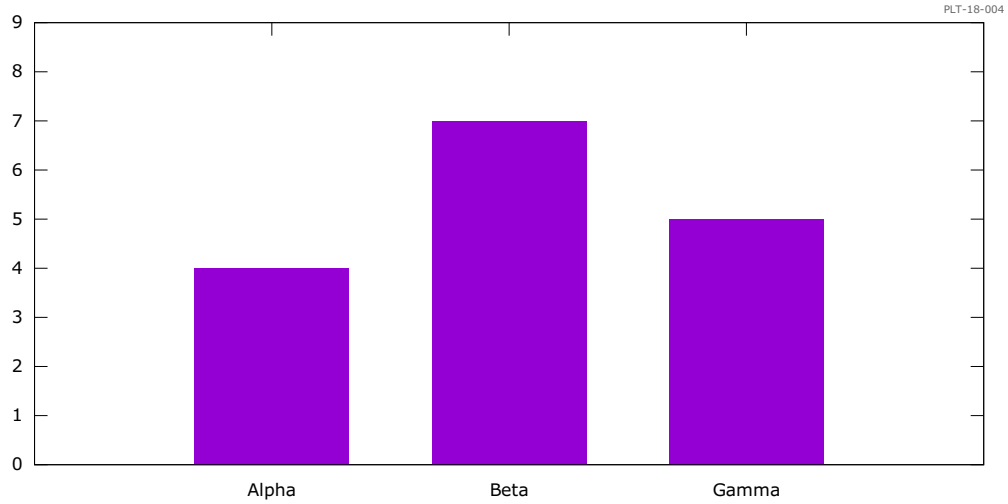
`solid 1.0`	Opaque fill; 1.0 is the fill density.
`noborder`	Suppress the outline around each box.

Use category names

Compare with PLT-18-003 · Fill the bars.

The x positions stay numeric. A third column supplies the displayed names.

```
$D << EOD
1 4 "Alpha"
2 7 "Beta"
3 5 "Gamma"
EOD
set xrange [0:4]
set boxwidth 0.65 absolute
set yrange [0:9]
set style fill solid 1.0 noborder
plot $D using 1:2:xticlabels(3) with boxes notitle
```



PLT-18-004 · Generated from the complete Bash block

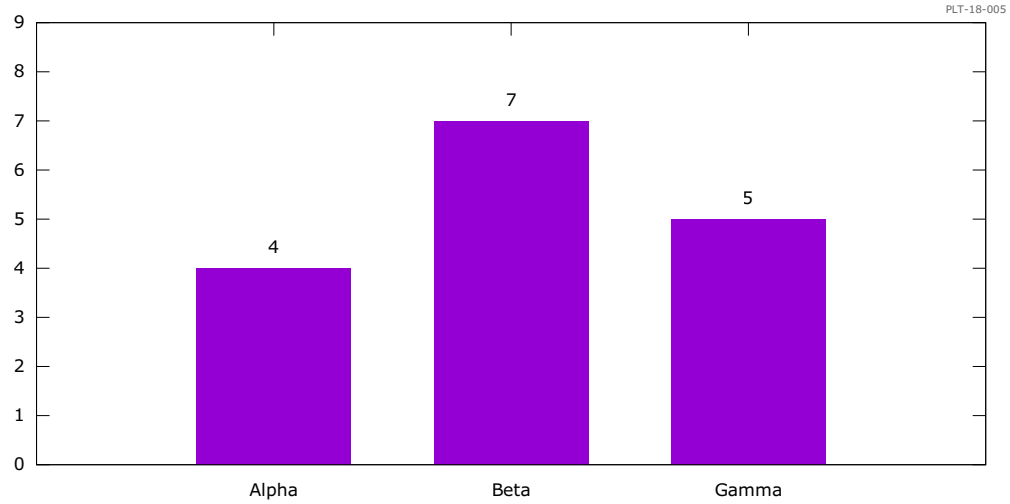
`xticlabels(3)`	Read labels from column 3 and place them at the corresponding x positions.

Write values above the bars

Compare with PLT-18-004 · Use category names.

Add a second plot layer using the same rows.

```
$D << EOD
1 4 "Alpha"
2 7 "Beta"
3 5 "Gamma"
EOD
set xrange [0:4]
set boxwidth 0.65 absolute
set yrange [0:9]
set style fill solid 1.0 noborder
plot $D using 1:2:xticlabels(3) with boxes notitle, \
     $D using 1:2:(sprintf("%g",$2)) \
     with labels offset 0,0.8 notitle
```



PLT-18-005 · Generated from the complete Bash block

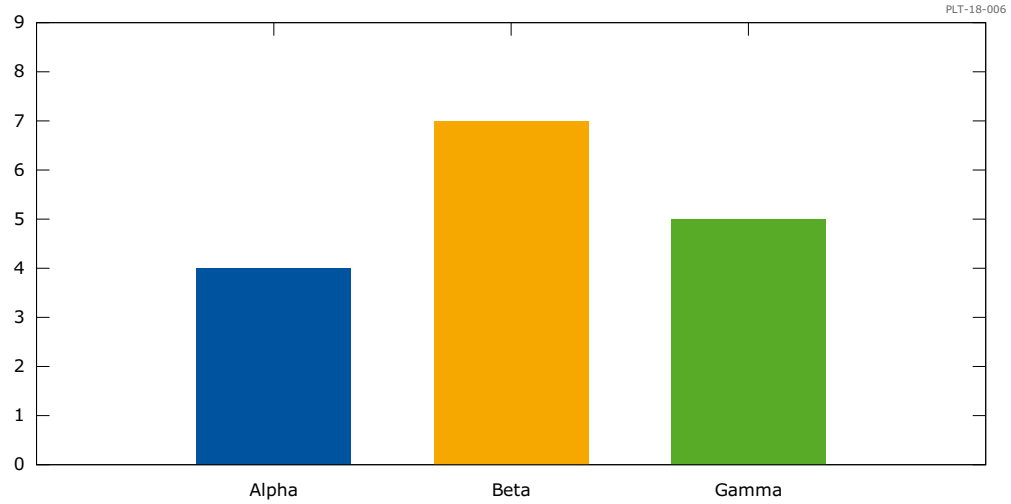
`sprintf("%g",$2)`	Format the current row's second column as compact numeric text.
`with labels`	Use x, y and a text expression as the three fields.
`offset 0,0.8`	Move text upward by 0.8 character heights; x offset is zero.

Colour each category

Compare with PLT-18-004 · Use category names.

Use one extra numeric field to choose an explicit RGB colour for each row.

```
$D << EOD
1 4 "Alpha" 0x00549F
2 7 "Beta" 0xF6A800
3 5 "Gamma" 0x57AB27
EOD
set xrange [0:4]
set boxwidth 0.65 absolute
set yrange [0:9]
set style fill solid 1.0 noborder
plot $D using 1:2:4:xticlabels(3) \
     with boxes linecolor rgb variable notitle
```



PLT-18-006 · Generated from the complete Bash block

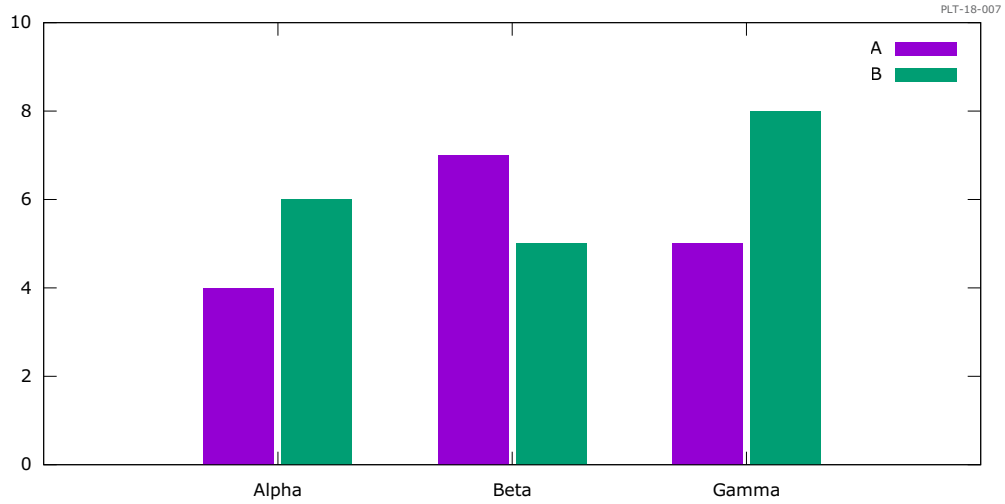
`0x00549F`	Hexadecimal integer 0xRRGGBB; its bytes encode red, green and blue.
`using 1:2:4`	The third numeric plotting field provides the colour.
`linecolor rgb variable`	Read an RGB integer from each row; the fill follows that colour.

Put two series side by side

Compare with PLT-18-004 · Use category names.

gnuplot calls this categorical layout a histogram. It consumes already summarized values; it does not bin raw observations.

```
$D << EOD
Alpha 4 6
Beta 7 5
Gamma 5 8
EOD
set style data histograms
set style histogram clustered gap 1
set style fill solid 1 noborder
set boxwidth 0.9 relative
set yrange [0:10]
plot $D using 2:xticlabels(1) title "A", \
     $D using 3 title "B"
```



PLT-18-007 · Generated from the complete Bash block

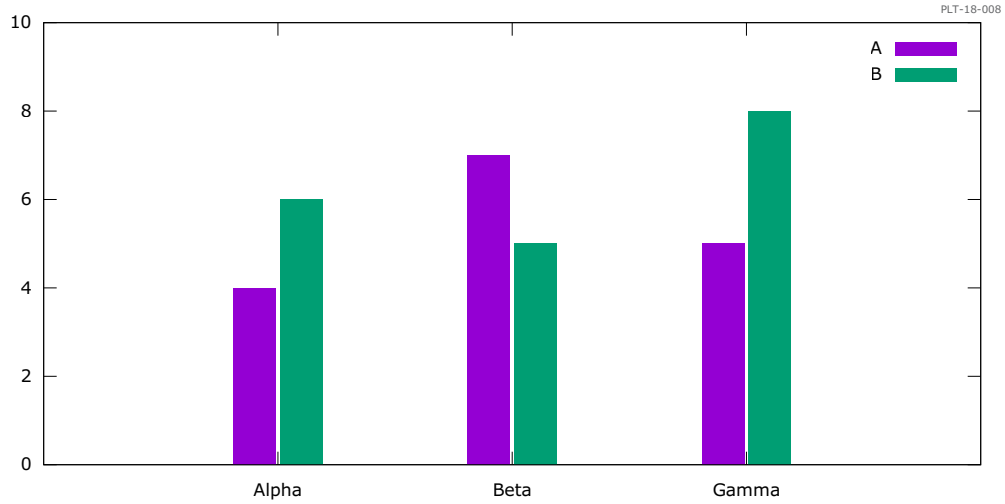
`style data histograms`	Set the default plot style for data sources.
`clustered gap 1`	Group series at each category; leave a gap of one bar-width unit between clusters.
`0.9 relative`	Use 90% of the width allocated to each bar.
`using 2 / using 3`	Each selected value column becomes one series; rows become categories.

Widen the gap between groups

Compare with PLT-18-007 · Put two series side by side.

The group spacing changes while the categories and values stay the same.

```
$D << EOD
Alpha 4 6
Beta 7 5
Gamma 5 8
EOD
set style data histograms
set style histogram clustered gap 3
set style fill solid 1 noborder
set boxwidth 0.9 relative
set yrange [0:10]
plot $D using 2:xticlabels(1) title "A", \
     $D using 3 title "B"
```



PLT-18-008 · Generated from the complete Bash block

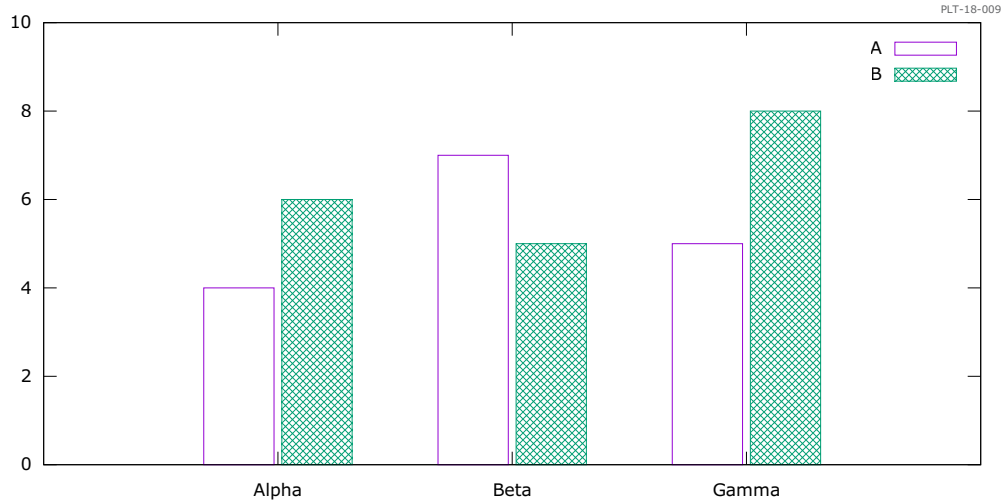
`gap 3` Reserve three bar-width units between neighbouring clusters; bars become narrower in the fixed canvas.

Use hatching for the series

Compare with PLT-18-007 · Put two series side by side.

Pattern fill provides another cue when colour is unavailable.

```
$D << EOD
Alpha 4 6
Beta 7 5
Gamma 5 8
EOD
set style data histograms
set style histogram clustered gap 1
set style fill pattern border
set boxwidth 0.9 relative
set yrange [0:10]
plot $D using 2:xticlabels(1) title "A", \
     $D using 3 title "B"
```



PLT-18-009 · Generated from the complete Bash block

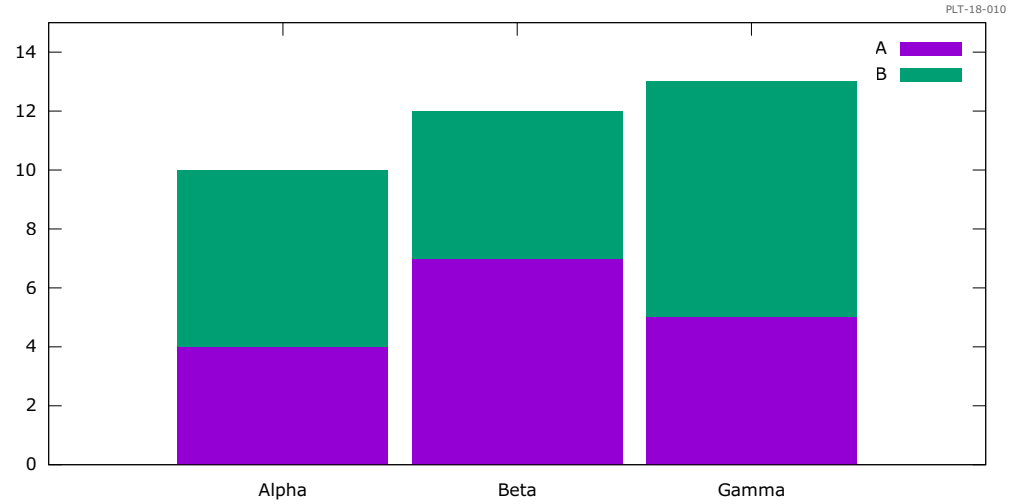
`pattern`	Cycle through the terminal's fill patterns.
`border`	Draw the outlines as well as the fill pattern.

Stack the series by row

Compare with PLT-18-007 · Put two series side by side.

Replace neighbouring bars with components of a total.

```
$D << EOD
Alpha 4 6
Beta 7 5
Gamma 5 8
EOD
set style data histograms
set style histogram rowstacked
set style fill solid 1 noborder
set boxwidth 0.9 relative
set yrange [0:15]
plot $D using 2:xticlabels(1) title "A", \
     $D using 3 title "B"
```



PLT-18-010 · Generated from the complete Bash block

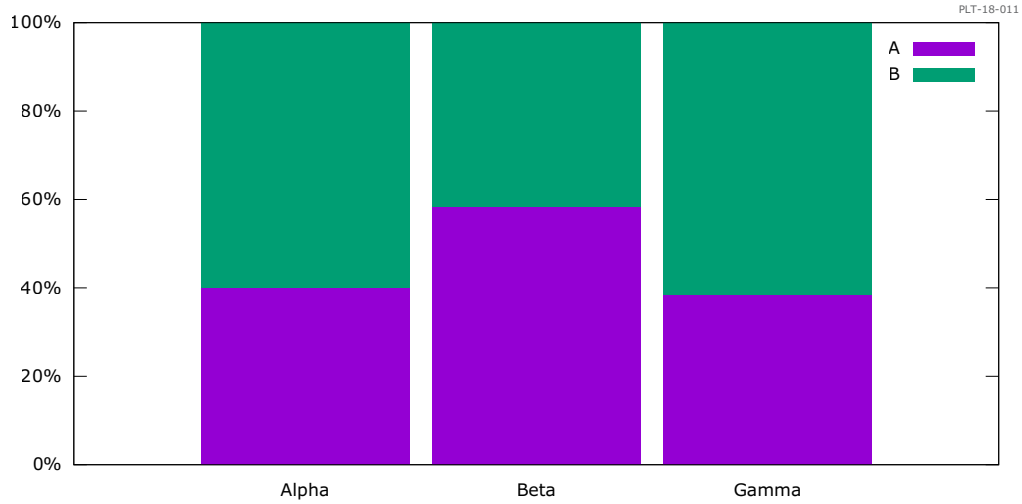
`rowstacked`	Stack the plotted columns within each row/category.
`[0:15]`	Allow room for totals 10, 12 and 13.

Normalize every stack to 100 percent

Compare with PLT-18-010 · Stack the series by row.

Each row is divided by its own sum. The chart now compares composition rather than totals.

```
$D << EOD
Alpha 4 6
Beta 7 5
Gamma 5 8
EOD
set style data histograms
set style histogram rowstacked
set style fill solid 1 noborder
set boxwidth 0.9 relative
set yrange [0:100]
set format y "%.0f%%"
plot $D using (100.0*$2/($2+$3)):xticlabels(1) title "A", \
     $D using (100.0*$3/($2+$3)) title "B"
```



PLT-18-011 · Generated from the complete Bash block

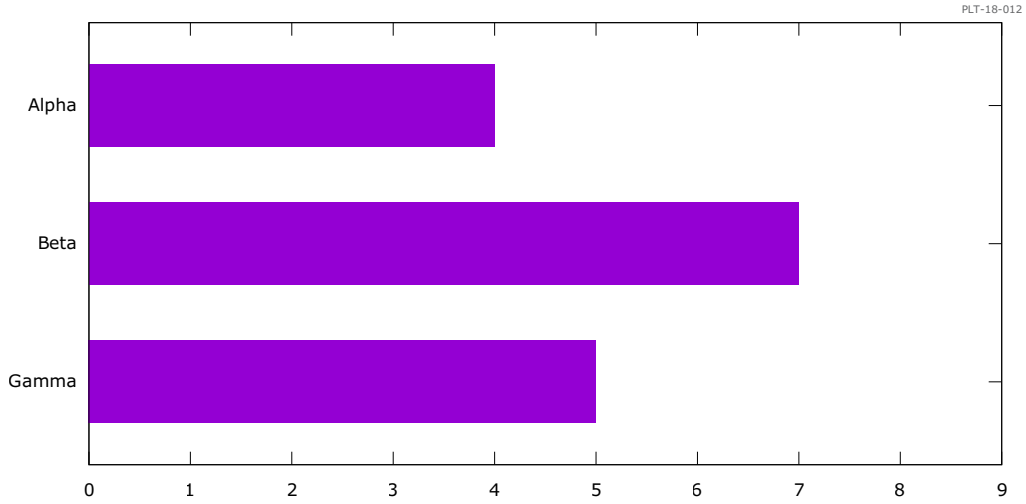
`100.0*$2/($2+$3)`	Convert the first component to a percentage; the second expression does the same for column 3.
`%.0f%%`	Display no decimals and a literal percent sign. Rows must have a nonzero total.

Turn bars horizontally

Compare with PLT-18-004 · Use category names.

Explicit rectangle bounds offer a simple horizontal bar chart and prepare the floating-bar examples.

```
$D << EOD
1 4 Alpha
2 7 Beta
3 5 Gamma
EOD
set xrange [0:9]
set yrange [3.6:0.4]
set style fill solid 1 noborder
plot $D using ($2/2):1:(0):2:($1-0.3):($1+0.3) \
     :yticlabels(3) with boxxyerrorbars notitle
```



PLT-18-012 · Generated from the complete Bash block

`boxxyerrorbars`	With six fields: x centre, y centre, x low, x high, y low, y high. These bounds describe a filled rectangle.
`($2/2):1:(0):2`	Centre each bar halfway along its value; extend x from zero to column 2.
`($1-0.3):($1+0.3)`	Give each row a vertical thickness of 0.6.
`[3.6:0.4]`	Descending explicit bounds reverse the y axis, placing the first category at the top.

Start with labelled bars

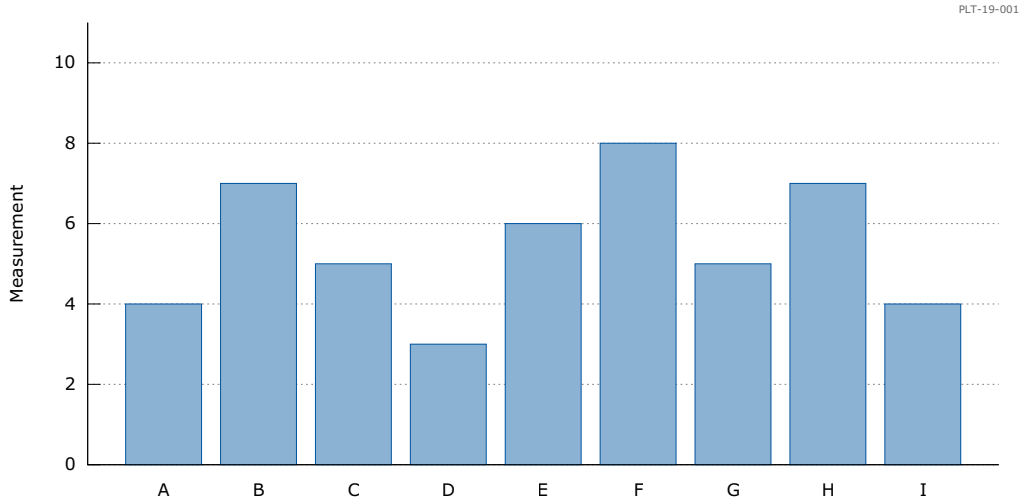
New baseline: the commands below define this example.

The single blank lines separate three groups in the inline data. Begin by drawing every observation in one colour. Pseudo-column 0 counts records across single blank lines, so the bars still appear evenly spaced.

```
$Bars << EOD
A 4
B 7
C 5

D 3
E 6

F 8
G 5
H 7
I 4
EOD
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
set yrange [0:11]
plot $Bars using 0:2:xtic(1) with boxes lc rgb "#00549F"
```



PLT-19-001 · Generated from the complete Bash block

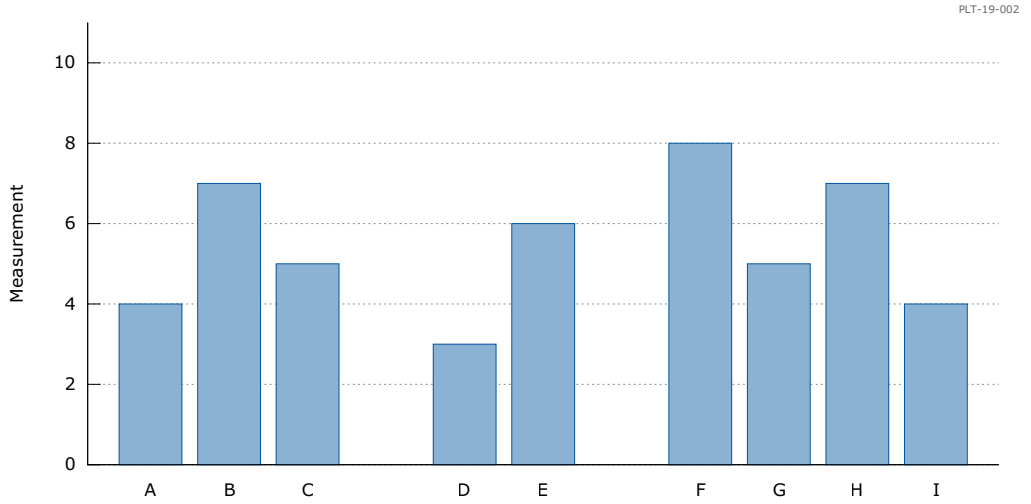
`using 0:2:xtic(1)`	Use record numbers for x, measurements for y and names for tick labels.
`single blank line`	Separate blocks without resetting the global record number; two consecutive blank lines instead start a new dataset.
`boxwidth 0.8`	Leave a small gap between neighbouring unit-spaced bars.

Leave one empty position between groups

Compare with PLT-19-001 · Start with labelled bars.

Change only the x expression. Pseudo-column -1 gives the zero-based block number. Adding one extra position for each block leaves a full empty slot between groups of unequal sizes.

```
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
set yrange [0:11]
plot $Bars using ($0+column(-1)+1):2:xtic(1) \
    with boxes lc rgb "#00549F"
```



PLT-19-002 · Generated from the complete Bash block

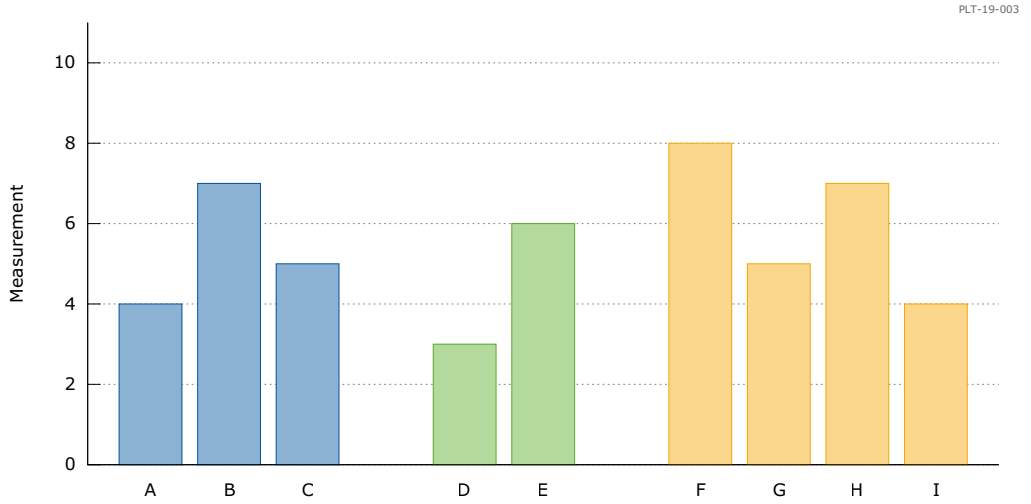
`column(-1)+1`	Convert zero-based block numbers into one-based group numbers.
`$0 + column(-1) + 1`	Advance each later group by one extra slot while keeping its bars contiguous.

Colour bars by group

Compare with PLT-19-002 · Leave one empty position between groups.

Use the same block number as a line-type selector. Each group now has its own colour; the tick labels continue to identify individual observations. A fourth colour is ready for the later data-only extension.

```
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
set yrange [0:11]
set linetype 1 lc rgb "#00549F"
set linetype 2 lc rgb "#57AB27"
set linetype 3 lc rgb "#F6A800"
set linetype 4 lc rgb "#7A6FAC"
plot $Bars using ($0+column(-1)+1):2:(column(-1)+1):xtic(1) \
    with boxes lc variable
```



PLT-19-003 · Generated from the complete Bash block

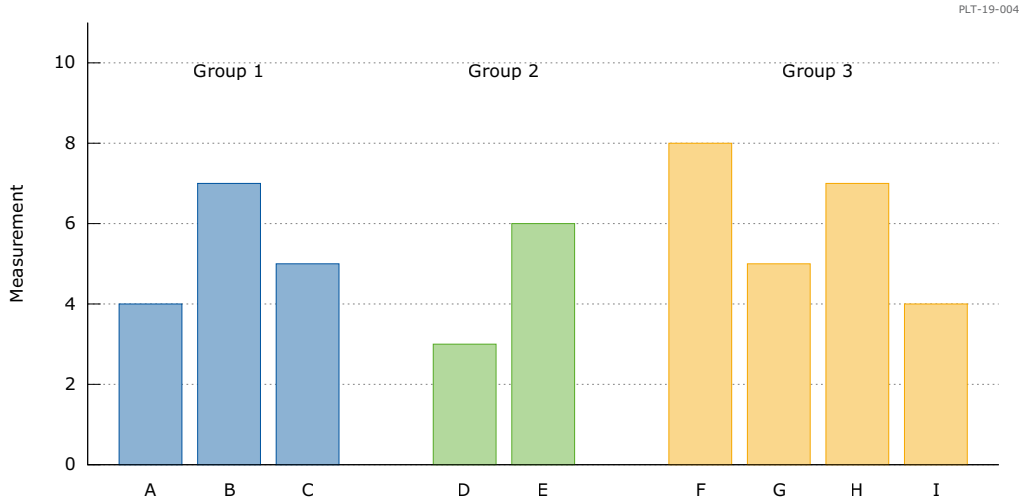
`lc variable`	Read a line-type number from the additional using column.
`set linetype 1 ... 4`	Define four colours explicitly rather than relying on terminal defaults.

Derive group labels and widths from the data

Compare with PLT-19-003 · Colour bars by group.

Count blank lines with stats, then count members while making a table with an explicit group column. A second datablock stores each generated name, left edge and width. Three groups of 3, 2 and 4 members begin at edges 0.5, 4.5 and 7.5. All loops use the detected group count.

```
stats $Bars using 2 name "B" nooutput
groups = B_blank + 1
array members[groups]
do for [g=1:groups] { members[g] = 0 }
set table $Plot
plot $Bars using (strcol(1)):2: \
    (g=column(-1)+1, members[g]=members[g]+1, g) with table
unset table
set print $Groups
left = 0.5
do for [g=1:groups] {
    print sprintf('"Group %d" %.1f %d',g,left,members[g])
    left = left + members[g] + 1
}
unset print
mark_y = B_max + 1
set yrange [0:B_max+3]
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
plot $Plot using ($0+$3):2:3:xtic(1) with boxes lc variable, \
    $Groups using ($2+$3/2):(mark_y+0.8):1 with labels center
```



PLT-19-004 · Generated from the complete Bash block

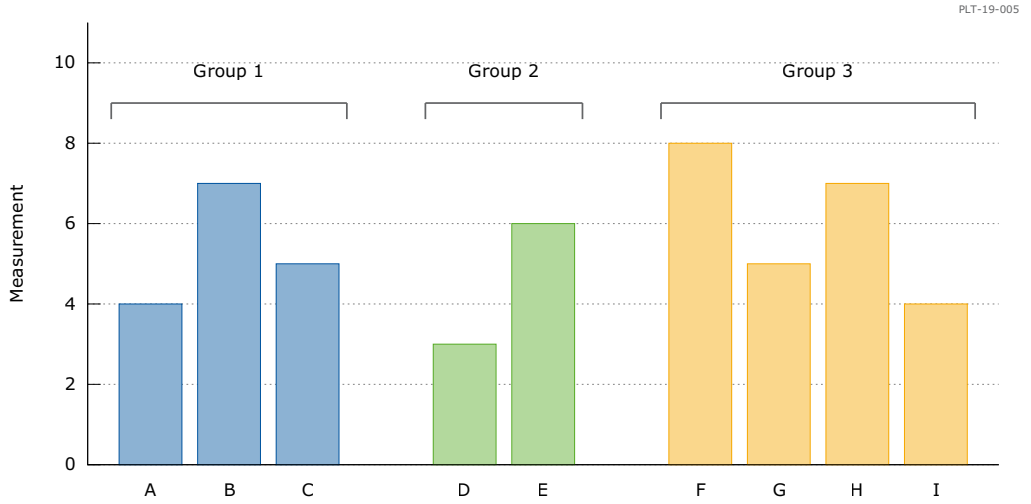
`B_blank + 1`	One single blank separator between each adjacent group gives the group count; avoid leading or trailing blank separators.
`members[g]`	Count each group while reading its rows into the plotting table.
`set table; with table`	Record names, measurements and group numbers without drawing a figure.
`left + members[g] + 1`	Place the next group after the current width and one empty slot.
`B_max + 1`	Place annotations above the detected tallest bar; reserve additional space in yrange.

Scale a bracket to each group

Compare with PLT-19-004 · Derive group labels and widths from the data.

Define one three-segment bracket in normalised coordinates. Multiply its horizontal coordinates by the detected group width and translate it to the group left edge. The three brackets span 3, 2 and 4 slots automatically.

```
stats $Bars using 2 name "B" nooutput
groups = B_blank + 1
array members[groups]
do for [g=1:groups] { members[g] = 0 }
set table $Plot
plot $Bars using (strcol(1)):2: \
    (g=column(-1)+1, members[g]=members[g]+1, g) with table
unset table
set print $Groups
left = 0.5
do for [g=1:groups] {
    print sprintf('"Group %d" %.1f %d',g,left,members[g])
    left = left + members[g] + 1
}
unset print
mark_y = B_max + 1
set yrange [0:B_max+3]
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
array sx[3] = [0,0,1]
array sy[3] = [-0.4,0,0]
array dx[3] = [0,1,0]
array dy[3] = [0.4,0,-0.4]
plot $Plot using ($0+$3):2:3:xtic(1) with boxes lc variable, \
    for [s=1:3] $Groups using \
    ($2+sx[s]*$3):(mark_y+sy[s]):(dx[s]*$3):(dy[s]) \
    with vectors nohead lw 2 lc rgb "#646567", \
    $Groups using ($2+$3/2):(mark_y+0.8):1 with labels center
```



PLT-19-005 · Generated from the complete Bash block

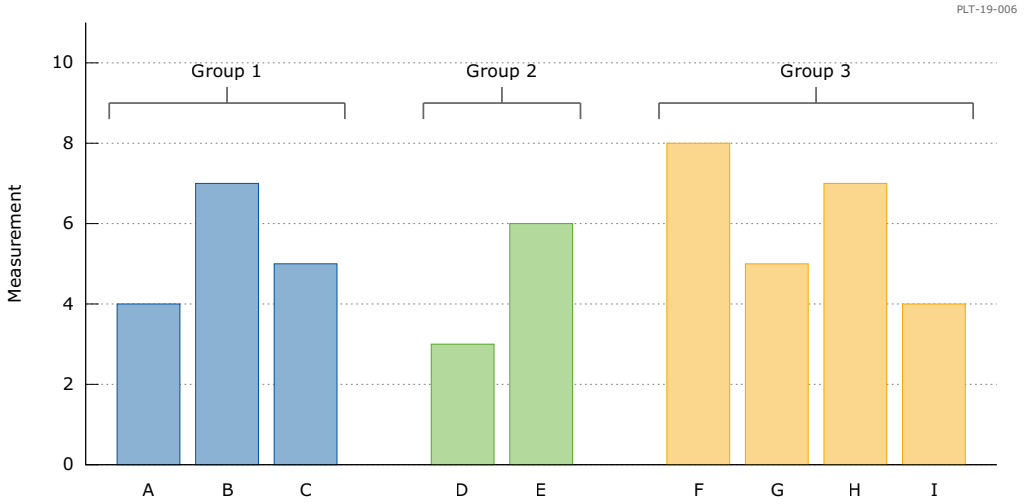
`sx, sy`	Start coordinates of each normalised segment.
`dx, dy`	Segment displacements; horizontal values are scaled by group width.
`for [s=1:3]`	Draw the same three template segments for every group.
`with vectors nohead`	Draw plain line segments rather than arrows.
`left edge and width`	The first bar centre is left+0.5; the last is left+width-0.5. The bracket spans all their unit slots.

Add a centred stem to the reusable shape

Compare with PLT-19-005 · Scale a bracket to each group.

Extend the template with one vertical segment at normalised x=0.5. Its x position follows the centre of any group width. Its height stays fixed in y-axis units. This produces the bracket-and-stem grouping shape shown in the linked demonstration using commands available in gnuplot 6.0.5.

```
stats $Bars using 2 name "B" nooutput
groups = B_blank + 1
array members[groups]
do for [g=1:groups] { members[g] = 0 }
set table $Plot
plot $Bars using (strcol(1)):2: \
    (g=column(-1)+1, members[g]=members[g]+1, g) with table
unset table
set print $Groups
left = 0.5
do for [g=1:groups] {
    print sprintf('"Group %d" %.1f %d',g,left,members[g])
    left = left + members[g] + 1
}
unset print
mark_y = B_max + 1
set yrange [0:B_max+3]
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
array sx[4] = [0,0,1,0.5]
array sy[4] = [-0.4,0,0,0]
array dx[4] = [0,1,0,0]
array dy[4] = [0.4,0,-0.4,0.4]
plot $Plot using ($0+$3):2:3:xtic(1) with boxes lc variable, \
    for [s=1:4] $Groups using \
    ($2+sx[s]*$3):(mark_y+sy[s]):(dx[s]*$3):(dy[s]) \
    with vectors nohead lw 2 lc rgb "#646567", \
    $Groups using ($2+$3/2):(mark_y+0.8):1 with labels center
```



PLT-19-006 · Generated from the complete Bash block

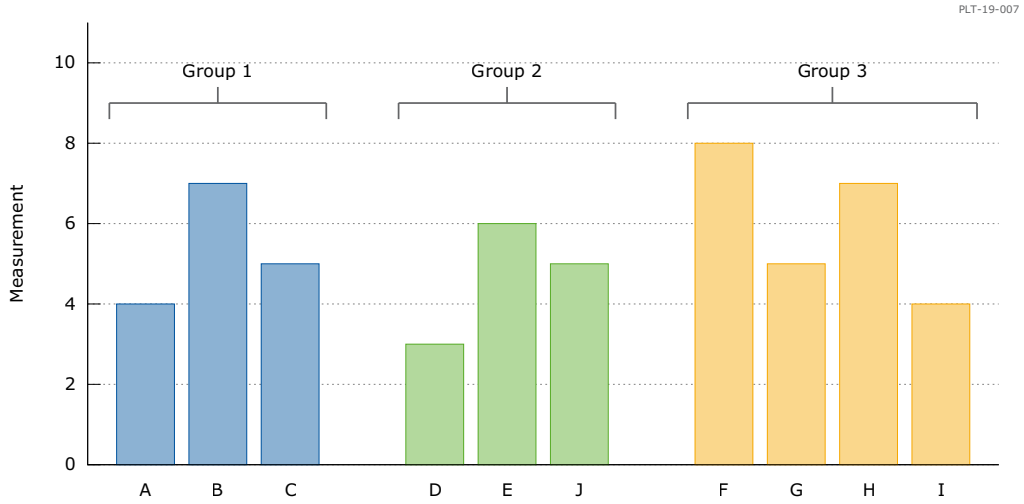
`sx[4] = 0.5`	Centre the new segment halfway across the scaled bracket.
`dy[4] = 0.4`	Raise the stem by 0.4 measurement units independently of group width.
`for [s=1:4]`	Include the added stem in the reusable template.

Widen a group by adding only a row

Compare with PLT-19-006 · Add a centred stem to the reusable shape.

Add J with value 5 to the second group. The commands are unchanged from the previous step: member counts become 3, 3 and 4, the second bracket widens, and the third group shifts right by one slot. Its centred stem and label move with it.

```
stats $Bars using 2 name "B" nooutput
groups = B_blank + 1
array members[groups]
do for [g=1:groups] { members[g] = 0 }
set table $Plot
plot $Bars using (strcol(1)):2: \
    (g=column(-1)+1, members[g]=members[g]+1, g) with table
unset table
set print $Groups
left = 0.5
do for [g=1:groups] {
    print sprintf('"Group %d" %.1f %d',g,left,members[g])
    left = left + members[g] + 1
}
unset print
mark_y = B_max + 1
set yrange [0:B_max+3]
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
array sx[4] = [0,0,1,0.5]
array sy[4] = [-0.4,0,0,0]
array dx[4] = [0,1,0,0]
array dy[4] = [0.4,0,-0.4,0.4]
plot $Plot using ($0+$3):2:3:xtic(1) with boxes lc variable, \
    for [s=1:4] $Groups using \
    ($2+sx[s]*$3):(mark_y+sy[s]):(dx[s]*$3):(dy[s]) \
    with vectors nohead lw 2 lc rgb "#646567", \
    $Groups using ($2+$3/2):(mark_y+0.8):1 with labels center
```



PLT-19-007 · Generated from the complete Bash block

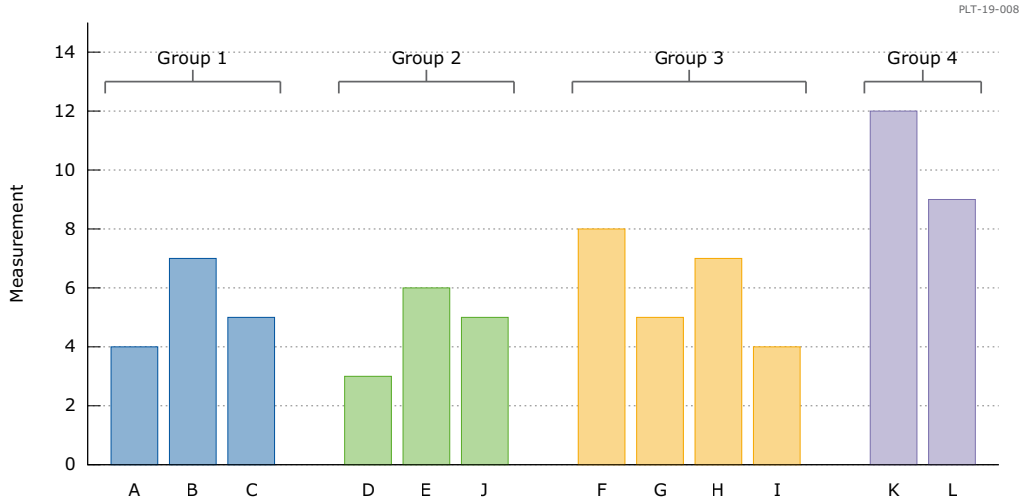
data-only change	Adding J 5 before the next blank separator changes the detected second-group width.
unchanged template and loops	Statistics, offsets, labels and bracket geometry are recomputed from the new rows.

Detect an extra group and a new maximum

Compare with PLT-19-007 · Widen a group by adding only a row.

Append a blank separator and a fourth group containing K=12 and L=9. Keep every command unchanged. The detected group count becomes four and its sizes are 3, 3, 4 and 2. The new maximum raises all brackets from y=9 to y=13 and the top of the y range from 11 to 15.

```
stats $Bars using 2 name "B" nooutput
groups = B_blank + 1
array members[groups]
do for [g=1:groups] { members[g] = 0 }
set table $Plot
plot $Bars using (strcol(1)):2: \
    (g=column(-1)+1, members[g]=members[g]+1, g) with table
unset table
set print $Groups
left = 0.5
do for [g=1:groups] {
    print sprintf('"Group %d" %.1f %d',g,left,members[g])
    left = left + members[g] + 1
}
unset print
mark_y = B_max + 1
set yrange [0:B_max+3]
set boxwidth 0.8
set style fill solid 0.45 border
set border 3
set xtics nomirror
set ytics nomirror
set grid ytics
unset key
set ylabel "Measurement"
array sx[4] = [0,0,1,0.5]
array sy[4] = [-0.4,0,0,0]
array dx[4] = [0,1,0,0]
array dy[4] = [0.4,0,-0.4,0.4]
plot $Plot using ($0+$3):2:3:xtic(1) with boxes lc variable, \
    for [s=1:4] $Groups using \
    ($2+sx[s]*$3):(mark_y+sy[s]):(dx[s]*$3):(dy[s]) \
    with vectors nohead lw 2 lc rgb "#646567", \
    $Groups using ($2+$3/2):(mark_y+0.8):1 with labels center
```



PLT-19-008 · Generated from the complete Bash block

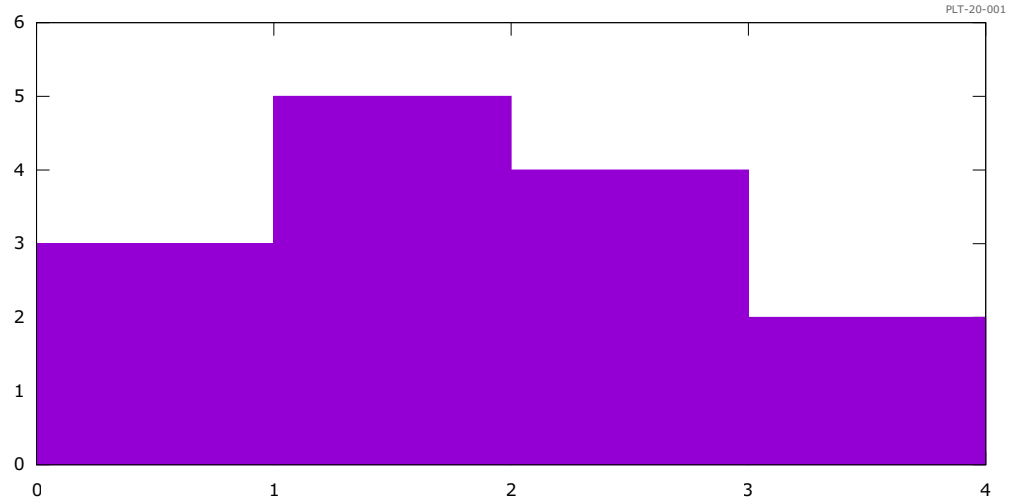
`B_blank + 1 = 4`	A new separator creates another group without editing any loop bound.
`members = 3, 3, 4, 2`	Every bracket width comes from the current row counts.
`B_max = 12`	The annotation height and plot range follow the tallest observation automatically.

Bin raw observations into counts

New baseline: the commands below define this example.

Unlike categorical histograms, this recipe first groups numerical observations. Each observation contributes one count to its bin.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
w = 1.0
bin(x) = w*(floor(x/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:6]
set xtics 1
plot $D using (bin($1)):(1) \
     smooth frequency with boxes notitle
```



PLT-20-001 · Generated from the complete Bash block

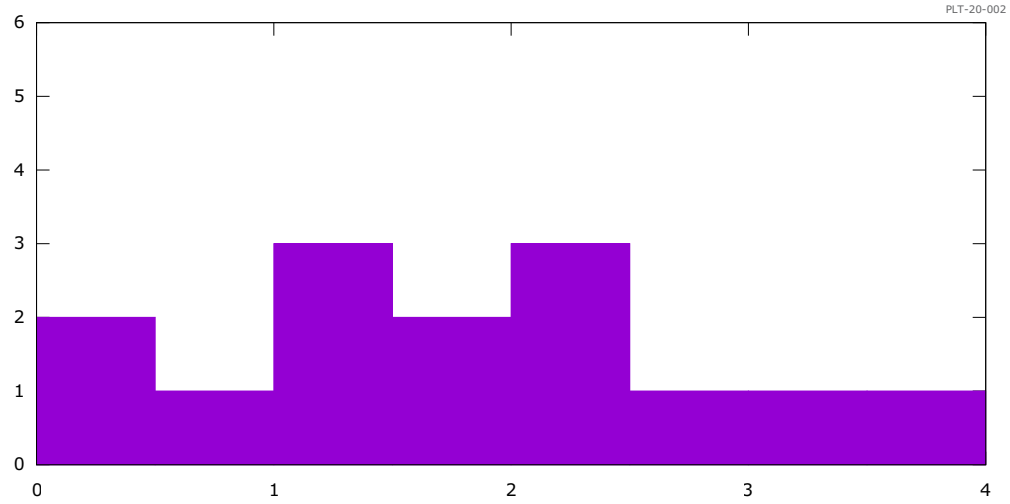
`w = 1.0`	Bin width in data units. Bins are [0,1), [1,2), and so on.
`floor(x/w)+0.5`	Round down to a bin index, then add half a bin to plot at its centre.
`smooth frequency`	Sort x values and sum the y contributions at equal x.
`(1)`	Every observation contributes a count of one. Expected counts here: 3, 5, 4, 2.

Halve the bin width

Compare with PLT-20-001 · Bin raw observations into counts.

Narrow bins reveal finer detail but make counts more variable.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
w = 0.5
bin(x) = w*(floor(x/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:6]
set xtics 1
plot $D using (bin($1)):(1) \
     smooth frequency with boxes notitle
```



PLT-20-002 · Generated from the complete Bash block

`w = 0.5`

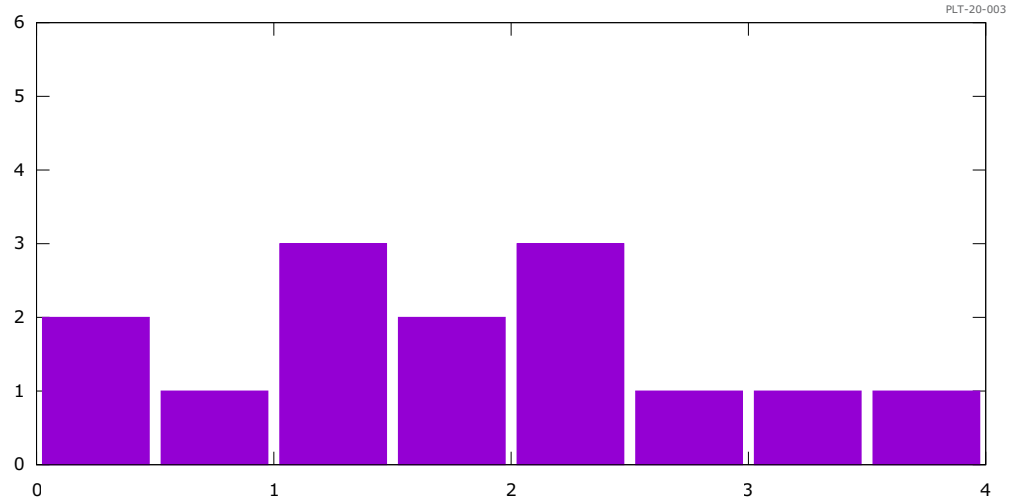
Change both the binning function and bar width through the same variable.

Leave small gaps between bins

Compare with PLT-20-002 · Halve the bin width.

A small gap makes the separate bins visible without changing their boundaries or counts.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
w = 0.5
bin(x) = w*(floor(x/w)+0.5)
set boxwidth (0.9*w) absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:6]
set xtics 1
plot $D using (bin($1)):(1) \
     smooth frequency with boxes notitle
```



PLT-20-003 · Generated from the complete Bash block

`0.9*w`

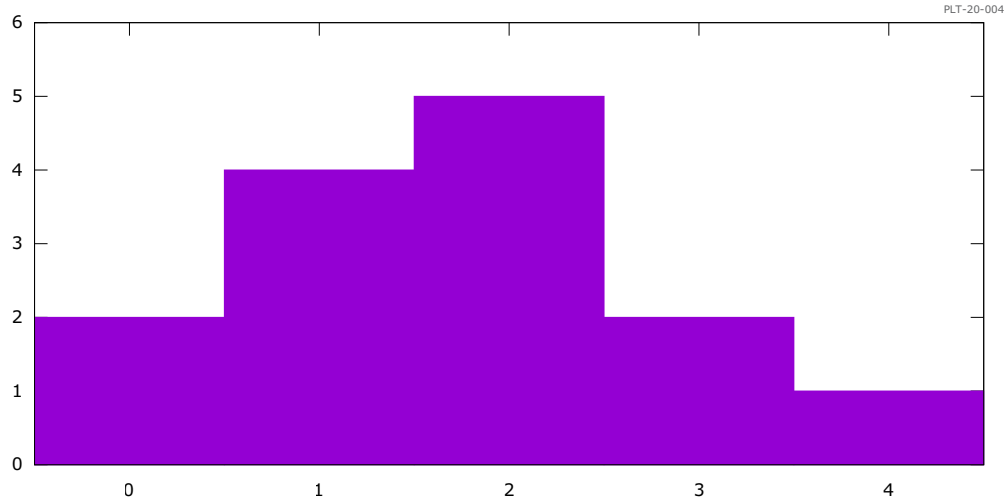
Draw each bar at 90% of the statistical bin width. The bins themselves still have width w.

Shift the bin boundaries

Compare with PLT-20-001 · Bin raw observations into counts.

Changing the origin can change the apparent pattern. The interval rule stays left-closed and right-open.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
w = 1.0
origin = -0.5
bin(x) = origin+w*(floor((x-origin)/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [-0.5:4.5]
set yrange [0:6]
set xtics 1
plot $D using (bin($1)):(1) \
     smooth frequency with boxes notitle
```



PLT-20-004 · Generated from the complete Bash block

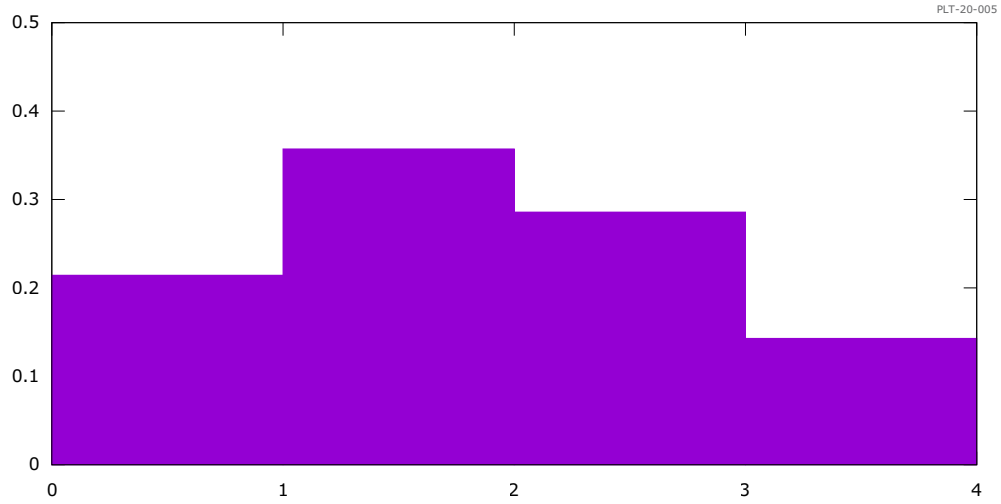
`origin = -0.5`	Start bins at -0.5, 0.5, 1.5, etc.
`(x-origin)/w`	Measure each observation relative to the chosen origin before rounding down.

Show probability per bin

Compare with PLT-20-001 · Bin raw observations into counts.

The bar heights now sum to one. This is a relative-frequency histogram.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
stats $D using 1 nooutput
N = STATS_records
w = 1.0
bin(x) = w*(floor(x/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:0.5]
set xtics 1
plot $D using (bin($1)):(1.0/N) \
     smooth frequency with boxes notitle
```



PLT-20-005 · Generated from the complete Bash block

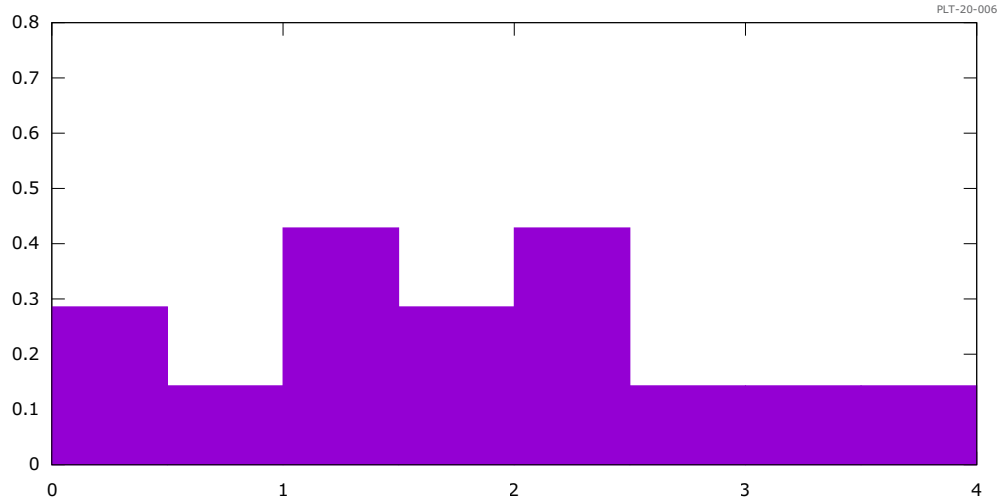
`stats ... nooutput`	Compute summaries without printing a report.
`STATS_records`	Number of valid observations; save it as N.
`1.0/N`	Each observation contributes its fraction of the total.

Make the histogram a density

Compare with PLT-20-005 · Show probability per bin.

For unequal bin widths or changed widths, compare areas rather than heights. These equal-width bars have total area one.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
stats $D using 1 nooutput
N = STATS_records
w = 0.5
bin(x) = w*(floor(x/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:0.8]
set xtics 1
plot $D using (bin($1)):(1.0/(N*w)) \
     smooth frequency with boxes notitle
```



PLT-20-006 · Generated from the complete Bash block

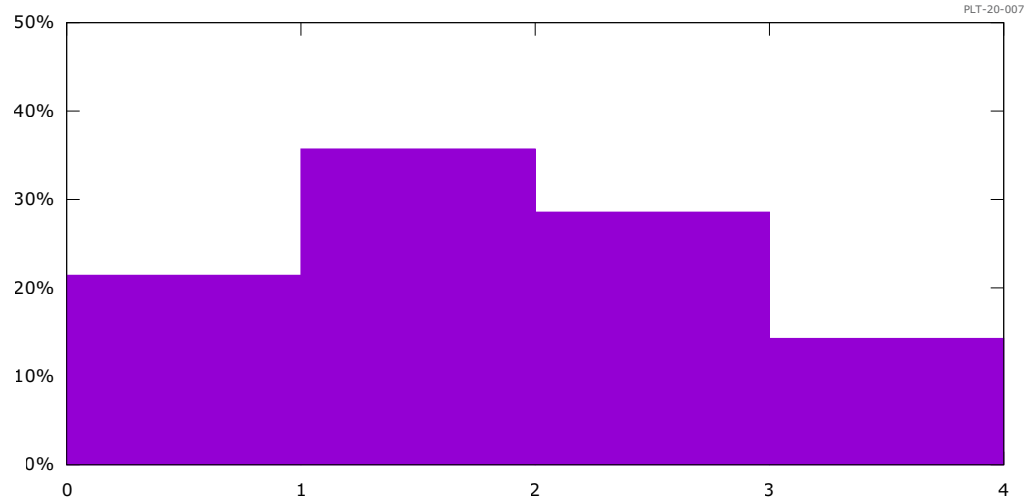
`1.0/(N*w)`	Divide by both observation count and bin width; density has units of 1/x.
`Area = height * w`	Summing the bar areas gives one. Keep full bin widths for this interpretation.

Display bin shares as percentages

Compare with PLT-20-005 · Show probability per bin.

Formatting changes the labels; multiplying by 100 changes the underlying scale.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
stats $D using 1 nooutput
N = STATS_records
w = 1.0
bin(x) = w*(floor(x/w)+0.5)
set boxwidth w absolute
set style fill solid 1 border
set xrange [0:4]
set yrange [0:50]
set format y "%.0f%%"
set xtics 1
plot $D using (bin($1)):(100.0/N) \
     smooth frequency with boxes notitle
```



PLT-20-007 · Generated from the complete Bash block

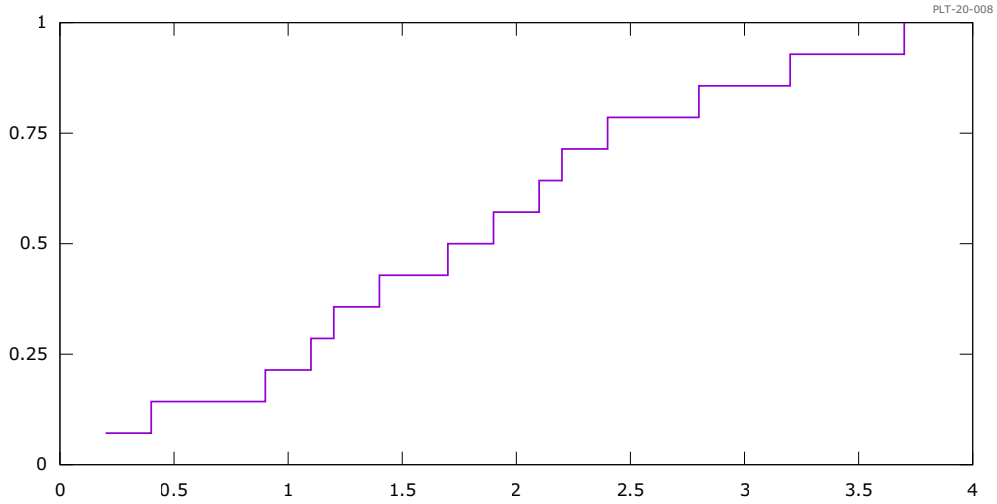
`100.0/N`	Each contribution is a percentage point share.
`%.0f%%`	Append a literal percent sign to rounded integer tick values.

Show the empirical cumulative distribution

Compare with PLT-20-005 · Show probability per bin.

A cumulative view avoids choosing a bin width. It shows the fraction of observations at or below each observed x.

```
$D << EOD
0.2
0.4
0.9
1.1
1.2
1.4
1.7
1.9
2.1
2.2
2.4
2.8
3.2
3.7
EOD
set xrange [0:4]
set yrange [0:1]
set ytics 0.25
plot $D using 1:(1) smooth cnormal \
     with steps linewidth 2 notitle
```



PLT-20-008 · Generated from the complete Bash block

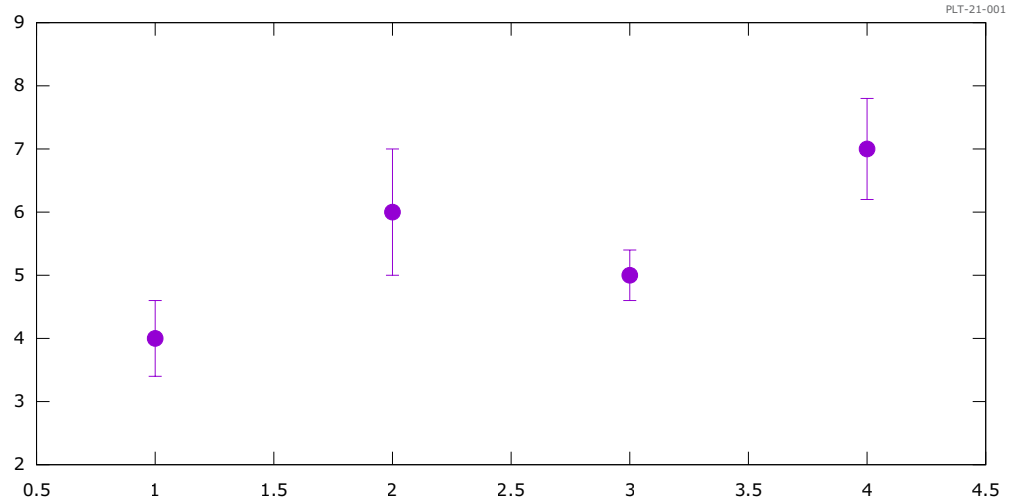
`smooth cnormal`	Sort, accumulate the y contributions, and normalize the final sum to one.
`with steps`	Keep the cumulative level constant until the next observation. The curve is drawn between observed x values.

Add symmetric error bars

New baseline: the commands below define this example.

The third column specifies a distance above and below the measured value.

```
$D << EOD
1 4 0.6
2 6 1.0
3 5 0.4
4 7 0.8
EOD
set xrange [0.5:4.5]
set yrange [2:9]
plot $D using 1:2:3 with yerrorbars \
     pointtype 7 pointsize 1.2 notitle
```



PLT-21-001 · Generated from the complete Bash block

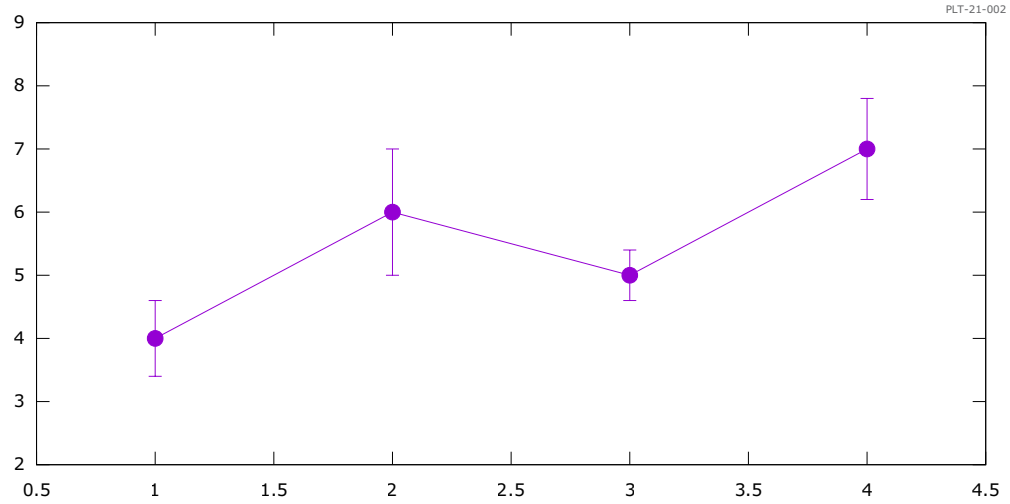
`using 1:2:3`	For yerrorbars, the fields mean x, y and symmetric y error.
`pointtype 7 / pointsize 1.2`	Filled-circle marker at 1.2 times the terminal's base marker size.
`Error interpretation`	The plot does not determine whether errors are SD, SE or confidence bounds; label the chosen meaning.

Connect the uncertain measurements

Compare with PLT-21-001 · Add symmetric error bars.

Draw connecting lines as well as the same error bars and markers.

```
$D << EOD
1 4 0.6
2 6 1.0
3 5 0.4
4 7 0.8
EOD
set xrange [0.5:4.5]
set yrange [2:9]
plot $D using 1:2:3 with yerrorlines \
     pointtype 7 pointsize 1.2 notitle
```



PLT-21-002 · Generated from the complete Bash block

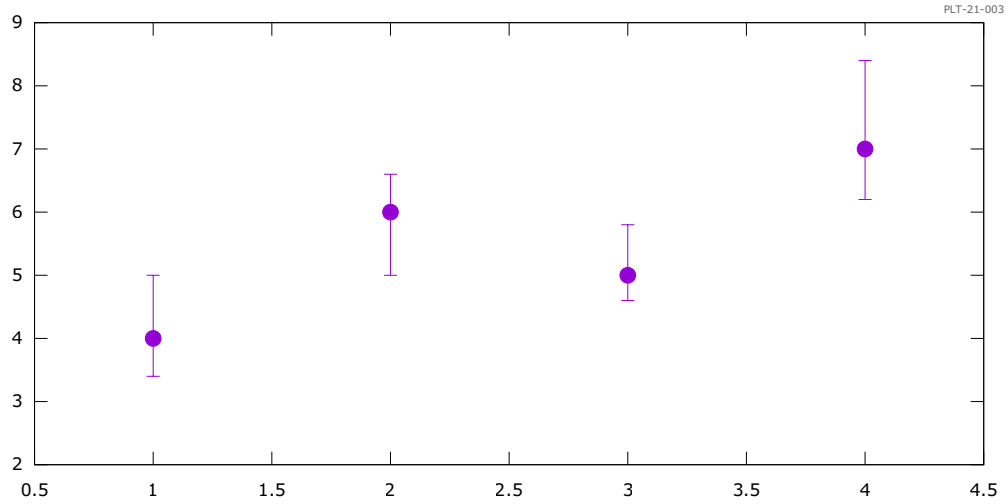
`yerrorlines`	Combine y error bars, points and connecting line segments.

Use asymmetric bounds

Compare with PLT-21-001 · Add symmetric error bars.

The low and high values are absolute endpoints, not error magnitudes.

```
$D << EOD
1 4 3.4 5.0
2 6 5.0 6.6
3 5 4.6 5.8
4 7 6.2 8.4
EOD
set xrange [0.5:4.5]
set yrange [2:9]
plot $D using 1:2:3:4 with yerrorbars \
     pointtype 7 pointsize 1.2 notitle
```



PLT-21-003 · Generated from the complete Bash block

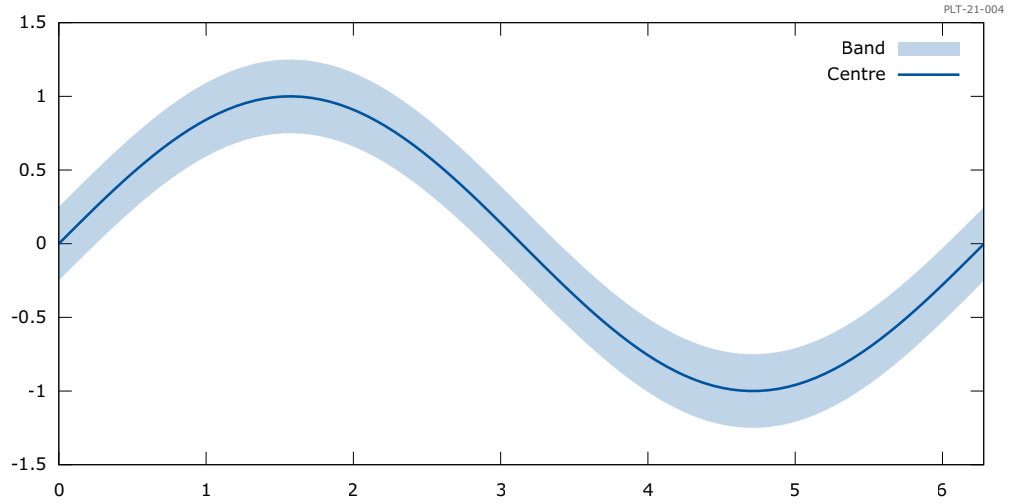
`x:y:ylow:yhigh` Four fields select asymmetric error-bar endpoints. For the first point the range is 3.4 to 5.0.

Show uncertainty as a ribbon

New baseline: the commands below define this example.

Two expressions define the lower and upper curves. Draw the translucent band first, then the central line.

```
set xrange [0:6.28]
set yrange [-1.5:1.5]
set style fill transparent solid 0.25 noborder
plot '+' using 1:(sin($1)-0.25):(sin($1)+0.25) \
     with filledcurves linecolor rgb "#00549F" \
     title "Band", \
     sin(x) with lines linewidth 3 \
     linecolor rgb "#00549F" title "Centre"
```



PLT-21-004 · Generated from the complete Bash block

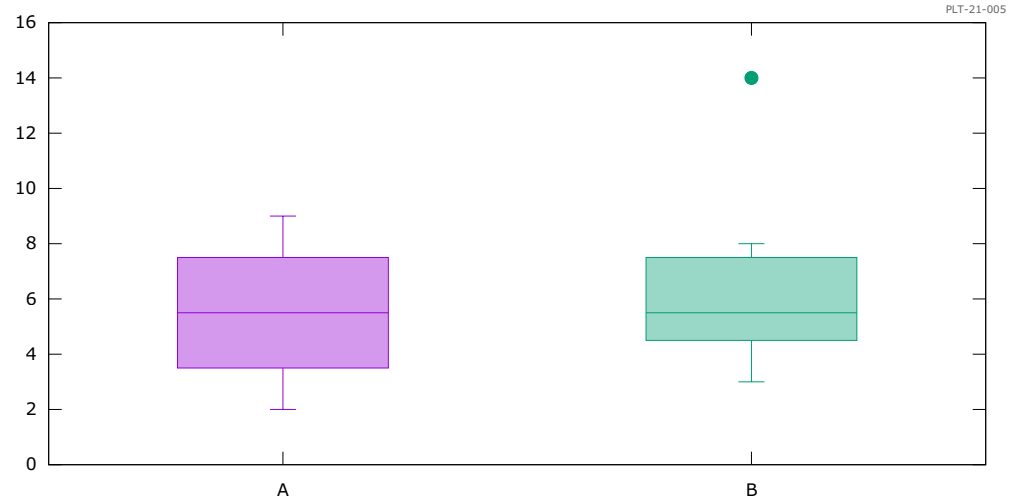
`using x:lower:upper`	Fill the region between two y values at every sampled x.
`transparent solid 0.25`	Use 25% opacity, allowing other plot layers to show through.
`Layer order`	The later central line remains visible above the filled band.

Summarize groups with boxplots

New baseline: the commands below define this example.

A boxplot summarizes each group's distribution instead of showing every value as a separate bar.

```
$D << EOD
2 3
3 4
4 5
5 5
6 6
7 7
8 8
9 14
EOD
set style fill solid 0.4 border
set boxwidth 0.45 absolute
set style boxplot outliers pointtype 7
set xrange [0.5:2.5]
set yrange [0:16]
set xtics ("A" 1,"B" 2)
plot $D using (1):1 with boxplot notitle, \
     $D using (2):2 with boxplot notitle
```



PLT-21-005 · Generated from the complete Bash block

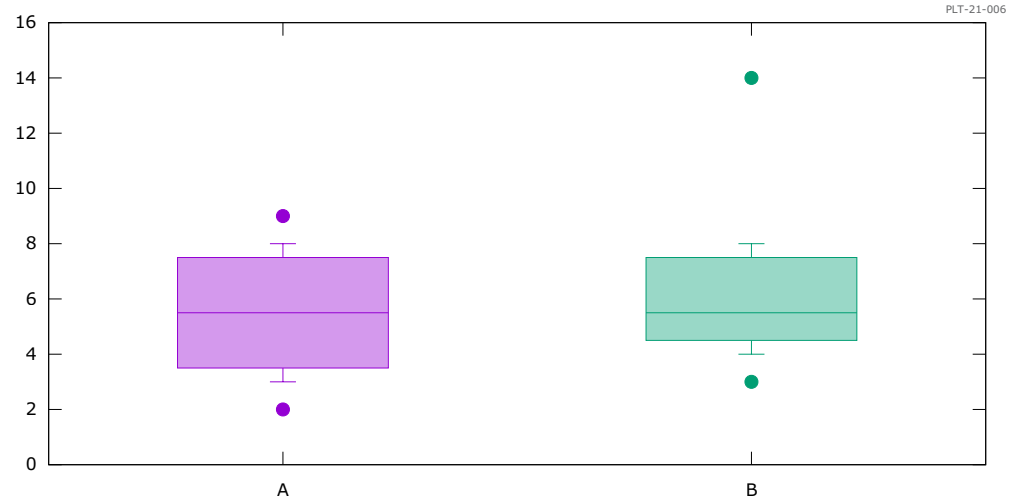
`using (1):1 / (2):2`	Place values from column 1 at x=1 and values from column 2 at x=2.
`with boxplot`	Show median, quartiles, whiskers and any selected outliers.
`Default whiskers`	Extend to observations within 1.5 interquartile ranges of the box; points beyond are outliers.

Change the boxplot whisker rule

Compare with PLT-21-005 · Summarize groups with boxplots.

A range of 0.25 makes the outlier threshold much tighter. The median and quartiles do not change.

```
$D << EOD
2 3
3 4
4 5
5 5
6 6
7 7
8 8
9 14
EOD
set style fill solid 0.4 border
set boxwidth 0.45 absolute
set style boxplot range 0.25 outliers pointtype 7
set xrange [0.5:2.5]
set yrange [0:16]
set xtics ("A" 1,"B" 2)
plot $D using (1):1 with boxplot notitle, \
     $D using (2):2 with boxplot notitle
```



PLT-21-006 · Generated from the complete Bash block

`range 0.25`

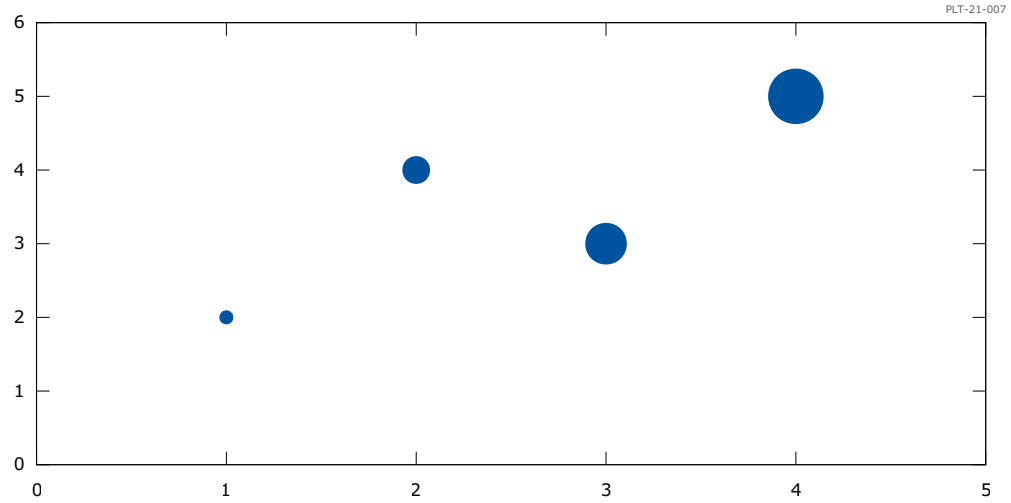
Limit whiskers to observations within a quarter interquartile range beyond the nearer quartile; more points become outliers.

Encode a third quantity as bubble area

New baseline: the commands below define this example.

Point size controls a linear marker dimension. Taking the square root makes marker area proportional to the chosen value.

```
$D << EOD
1 2 1
2 4 4
3 3 9
4 5 16
EOD
set xrange [0:5]
set yrange [0:6]
plot $D using 1:2:(sqrt($3)) with points \
     pointtype 7 pointsize variable \
     linecolor rgb "#00549F" notitle
```



PLT-21-007 · Generated from the complete Bash block

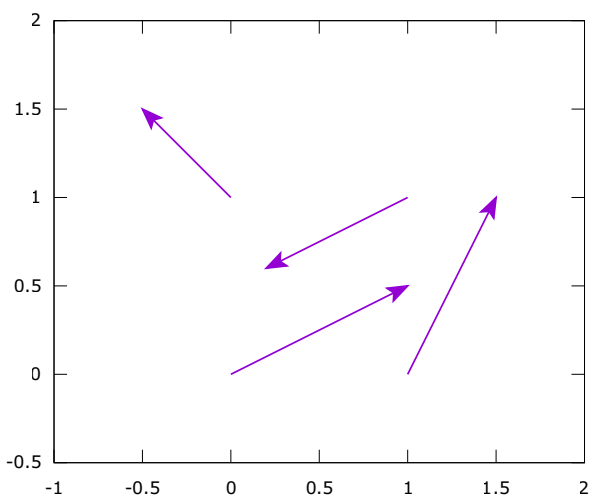
`sqrt($3)`	Convert an area quantity to a radius/diameter scale.
`pointsize variable`	Read the marker-size multiplier from the third plotting field.
`Comparability`	Use one scale factor throughout the chart; the numerical sizes remain terminal-relative.

Draw vectors from component data

New baseline: the commands below define this example.

The last two columns are displacements, so arrows can represent direction and magnitude.

```
$D << EOD
0 0 1 0.5
1 0 0.5 1
0 1 -0.5 0.5
1 1 -0.8 -0.4
EOD
set xrange [-1:2]
set yrange [-0.5:2]
set size ratio -1
plot $D using 1:2:3:4 with vectors \
     head filled size screen 0.02,20,60 \
     linewidth 2 notitle
```



PLT-21-008 · Generated from the complete Bash block

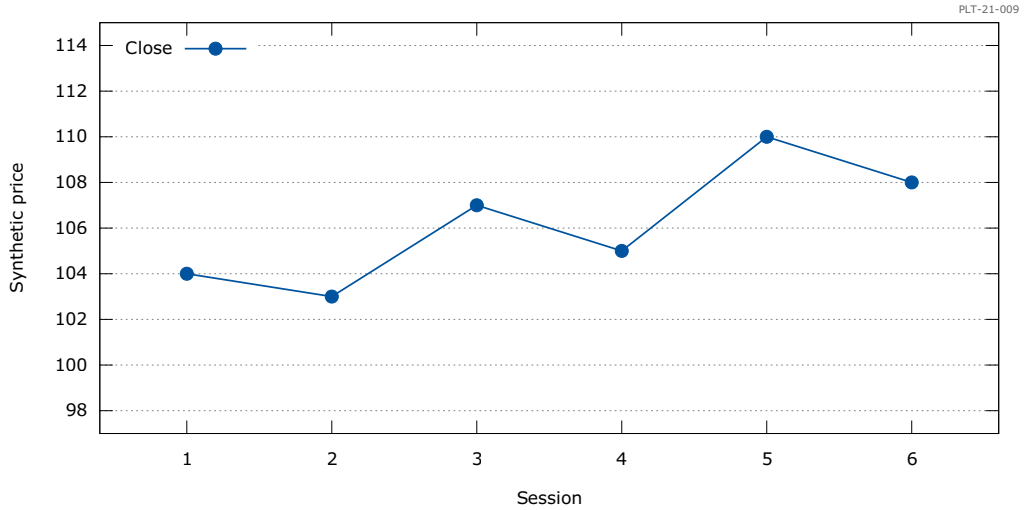
`x:y:dx:dy`	Arrow starts at (x,y) and ends at (x+dx,y+dy).
`head filled`	Add a filled arrowhead at the end.
`size screen 0.02,20,60`	Head length is 2% of canvas width; head-side and back angles are 20 and 60 degrees.

Begin a price chart with closing values

New baseline: the commands below define this example.

Six fictional sessions contain session, open, low, high and close columns. Begin with closing values; later steps reveal information this line cannot show.

```
$Prices << EOD
1 101 99 106 104
2 104 102 108 103
3 103 101 109 107
4 107 104 110 105
5 105 103 112 110
6 110 106 113 108
EOD
set xrange [0.4:6.6]
set yrange [97:115]
set xlabel "Session"
set ylabel "Synthetic price"
set xtics 1
set grid ytics
set key top left
plot $Prices using 1:5 with linespoints lw 2 pt 7 lc rgb "#00549F" title
"Close"
```



PLT-21-009 · Generated from the complete Bash block

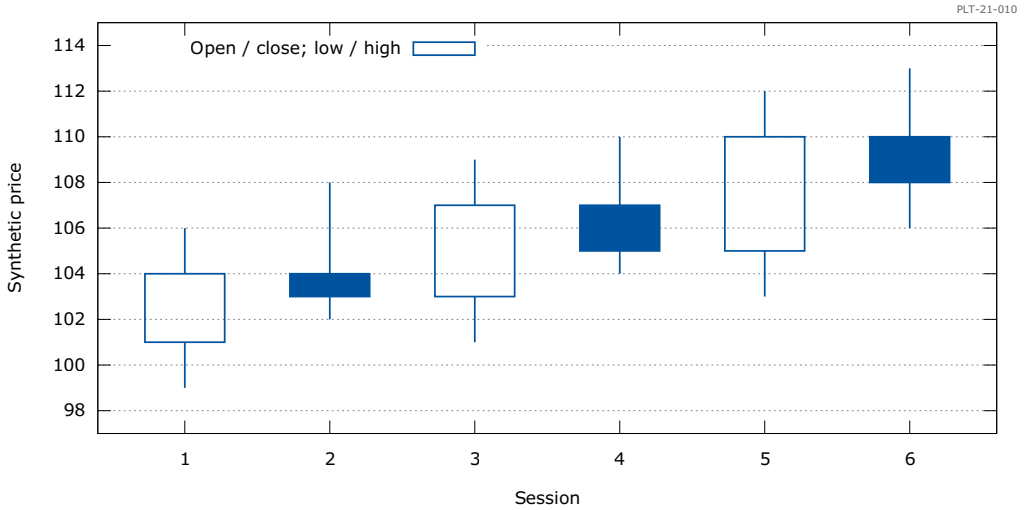
`using 1:5`	Select session and close; columns 2–4 are not yet plotted.
`$Prices`	Synthetic OHLC data: low is below both open and close.

Show open, close, low and high as candlesticks

Compare with PLT-21-009 · Begin a price chart with closing values.

Replace the closing line with candle bodies and wicks. The box spans open to close; the wick spans low to high. Empty fill gives the conventional open/close-dependent hollow or solid body.

```
set xrange [0.4:6.6]
set yrange [97:115]
set xlabel "Session"
set ylabel "Synthetic price"
set xtics 1
set grid ytics
set key top left
set boxwidth 0.55 absolute
set style fill empty
plot $Prices using 1:2:3:4:5 with candlesticks lw 2 lc rgb "#00549F" title
"Open / close; low / high"
```



PLT-21-010 · Generated from the complete Bash block

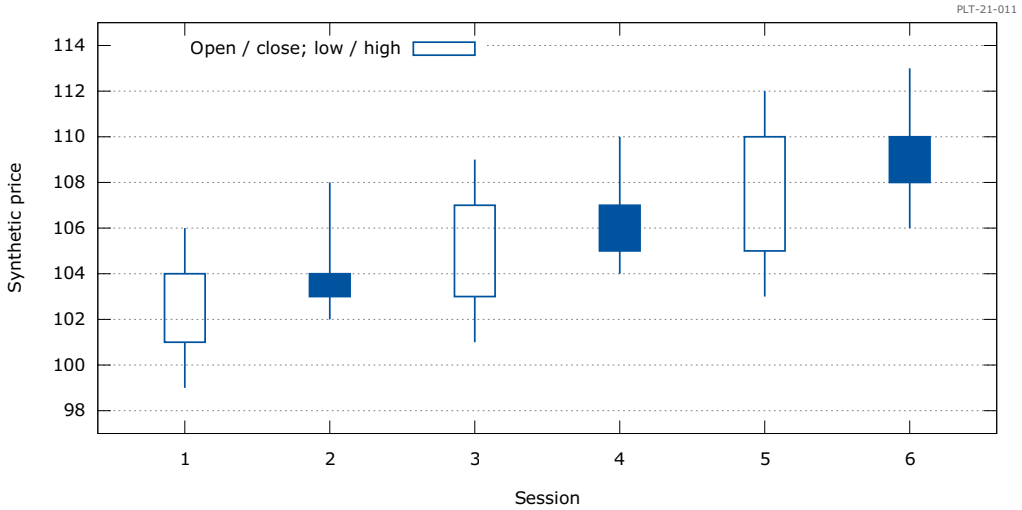
`using 1:2:3:4:5`	Order is x, open, low, high, close.
`boxwidth 0.55 absolute`	Set body width in session-axis units.

Narrow the candlestick bodies

Compare with PLT-21-010 · Show open, close, low and high as candlesticks.

Change only the body width. Narrower rectangles leave more space between sessions without changing any price or wick length.

```
set xrange [0.4:6.6]
set yrange [97:115]
set xlabel "Session"
set ylabel "Synthetic price"
set xtics 1
set grid ytics
set key top left
set boxwidth 0.28 absolute
set style fill empty
plot $Prices using 1:2:3:4:5 with candlesticks lw 2 lc rgb "#00549F" title
"Open / close; low / high"
```



PLT-21-011 · Generated from the complete Bash block

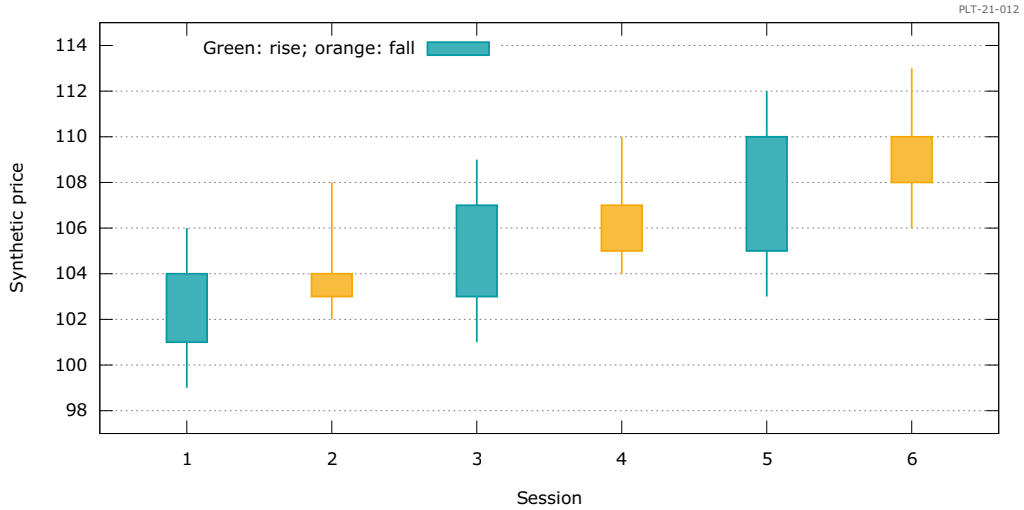
`boxwidth 0.28 absolute` Only the horizontal candle body width changes.

Colour rising and falling sessions from the data

Compare with PLT-21-011 · Narrow the candlestick bodies.

Compute an RGB colour per candle and use solid fill. RWTH turquoise means close is at least open; RWTH orange means close is below open. Colours update from the data rather than a session-number list.

```
set xrange [0.4:6.6]
set yrange [97:115]
set xlabel "Session"
set ylabel "Synthetic price"
set xtics 1
set grid ytics
set key top left
set boxwidth 0.28 absolute
set style fill solid 0.75 border
plot $Prices using 1:2:3:4:5:($5 >= $2 ? 0x0098A1 : 0xF6A800) \
    with candlesticks lw 2 lc rgb variable title "Green: rise; orange: fall"
```



PLT-21-012 · Generated from the complete Bash block

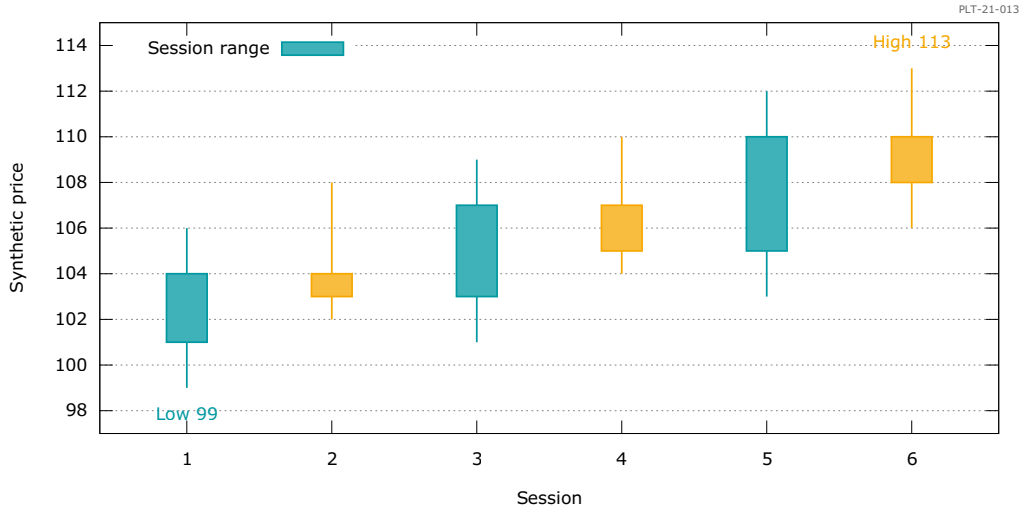
`$5 >= $2`	Compare each close with its opening value.
`lc rgb variable`	Read a packed RGB colour from the extra using column.

Annotate the automatically detected high and low

Compare with PLT-21-012 · Colour rising and falling sessions from the data.

Compute extrema before setting ranges. Use each returned record index to retrieve the correct session. Change an OHLC row and both labels move automatically.

```
stats $Prices using 3 name "Low" nooutput
stats $Prices using 4 name "High" nooutput
set table $LowPoint
plot $Prices every ::Low_index_min::Low_index_min using 1:3 with table
unset table
set table $HighPoint
plot $Prices every ::High_index_max::High_index_max using 1:4 with table
unset table
set xrange [0.4:6.6]
set yrange [97:115]
set xlabel "Session"
set ylabel "Synthetic price"
set xtics 1
set grid ytics
set key top left
set boxwidth 0.28 absolute
set style fill solid 0.75 border
plot $Prices using 1:2:3:4:5:($5 >= $2 ? 0x0098A1 : 0xF6A800) \
    with candlesticks lw 2 lc rgb variable title "Session range", \
    $LowPoint using 1:2:(sprintf("Low %.0f",Low_min)) \
    with labels offset 0,-1 tc rgb "#0098A1" notitle, \
    $HighPoint using 1:2:(sprintf("High %.0f",High_max)) \
    with labels offset 0,1 tc rgb "#F6A800" notitle
print sprintf("OHLC: low=%.0f row=%d high=%.0f
row=%d",Low_min,Low_index_min,High_max,High_index_max)
```



PLT-21-013 · Generated from the complete Bash block

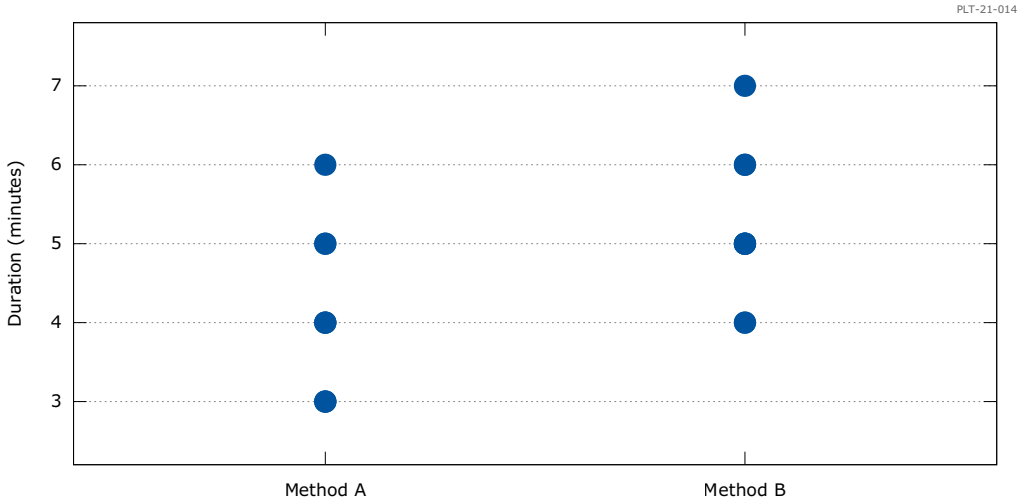
`stats ... name "Low" / "High"`	Find extrema and their zero-based record indices.
`every ::index::index`	Select the record containing an extreme.
`set table`	Capture selected coordinates for the labelled overlay.

See why repeated points hide observations

New baseline: the commands below define this example.

Two fictional methods have nine duration measurements each, including repeats. Identical points overlap exactly. All eighteen rows remain in the full Bash fixture; only eight distinct positions are visible.

```
set xrange [0.4:2.6]
set yrange [2.2:7.8]
set xtics ("Method A" 1,"Method B" 2)
set ylabel "Duration (minutes)"
set grid ytics
unset key
plot $Times using 1:2 with points pt 7 ps 1.6 lc rgb "#00549F"
```



PLT-21-014 · Generated from the complete Bash block

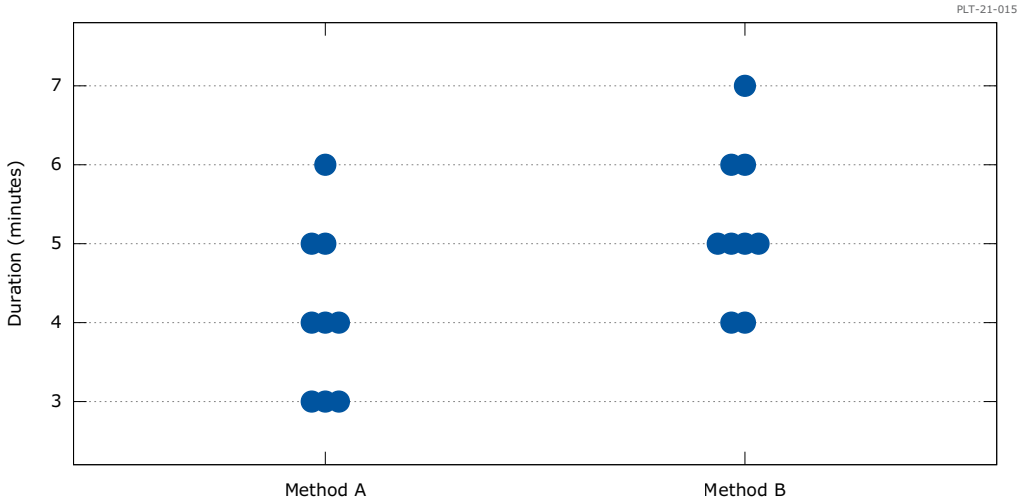
`repeated rows`	Separate observations can share the same coordinates.
`complete Bash script`	Open it to see all eighteen measurements.

Separate overlapping observations with jitter

Compare with PLT-21-014 · See why repeated points hide observations.

Enable bee-swarm arrangement. gnuplot displaces overlapping symbols horizontally while retaining their duration coordinates. Horizontal spread is a drawing aid, not measurement uncertainty.

```
set xrange [0.4:2.6]
set yrange [2.2:7.8]
set xtics ("Method A" 1,"Method B" 2)
set ylabel "Duration (minutes)"
set grid ytics
unset key
set jitter swarm
plot $Times using 1:2 with points pt 7 ps 1.6 lc rgb "#00549F"
```



PLT-21-015 · Generated from the complete Bash block

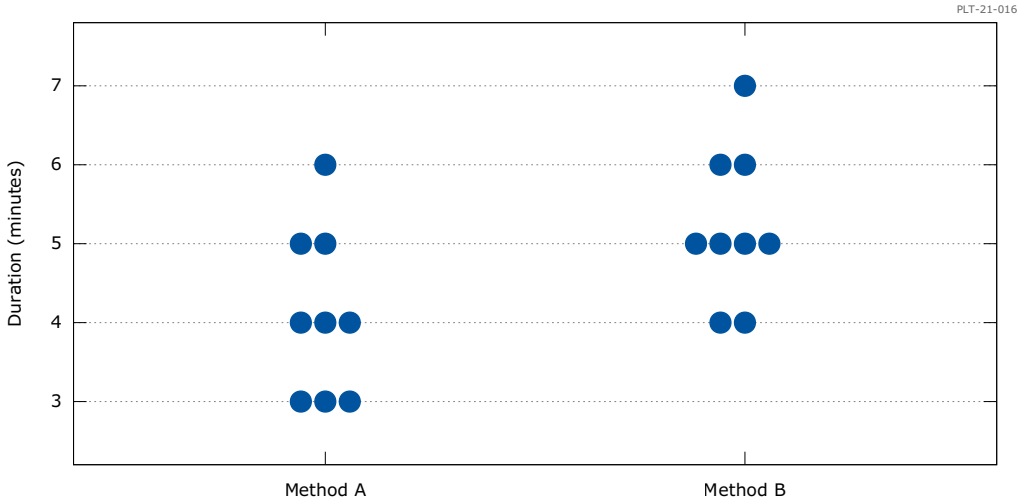
`set jitter swarm`	Separate symbols along x; preserve every y value.
`horizontal displacement`	A layout effect, not additional observed data.

Increase the swarm separation

Compare with PLT-21-015 · Separate overlapping observations with jitter.

Change only the spread multiplier to make duplicates easier to count. Default spacing uses terminal character size, so appearance depends on font and canvas dimensions.

```
set xrange [0.4:2.6]
set yrange [2.2:7.8]
set xtics ("Method A" 1,"Method B" 2)
set ylabel "Duration (minutes)"
set grid ytics
unset key
set jitter swarm spread 1.8
plot $Times using 1:2 with points pt 7 ps 1.6 lc rgb "#00549F"
```



PLT-21-016 · Generated from the complete Bash block

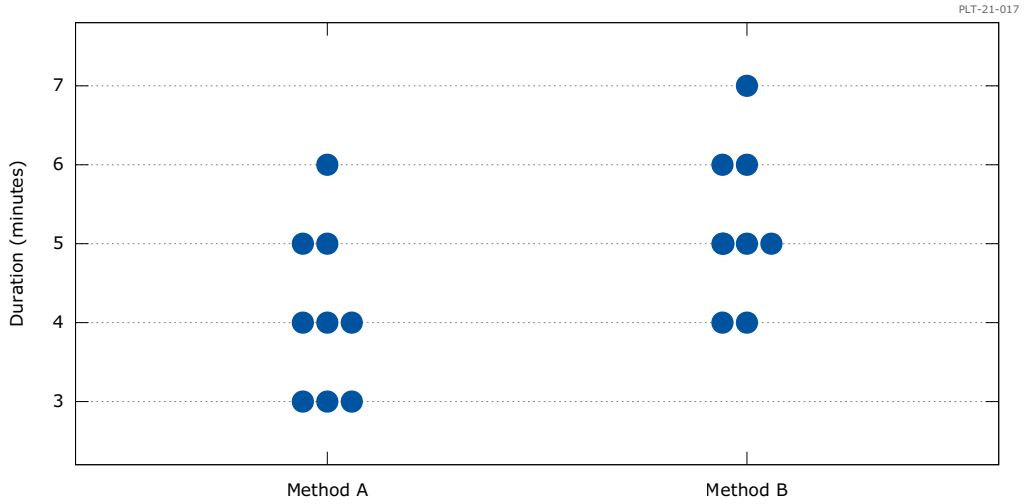
`spread 1.8`	Scale horizontal symbol displacement.
`terminal character units`	Font or canvas changes can alter swarm appearance.

Limit horizontal displacement with wrap

Compare with PLT-21-016 · Increase the swarm separation.

Keep the larger spacing but bound the swarm width. Wrapping can reintroduce crowding in dense groups; inspect the result rather than assuming every symbol stays isolated.

```
set xrange [0.4:2.6]
set yrange [2.2:7.8]
set xtics ("Method A" 1,"Method B" 2)
set ylabel "Duration (minutes)"
set grid ytics
unset key
set jitter swarm spread 1.8 wrap 3
plot $Times using 1:2 with points pt 7 ps 1.6 lc rgb "#00549F"
```



PLT-21-017 · Generated from the complete Bash block

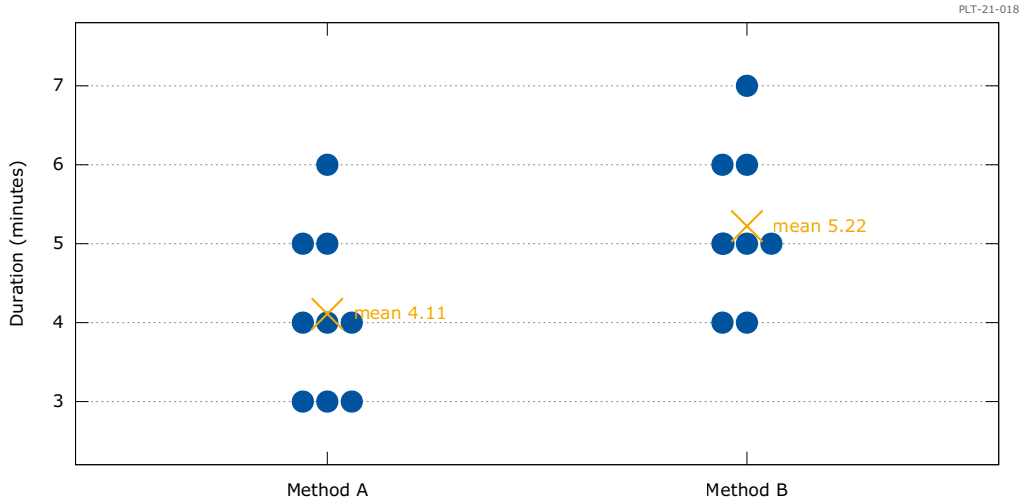
`wrap 3`	Limit horizontal displacement in character units.

Overlay automatically computed group means

Compare with PLT-21-017 · Limit horizontal displacement with wrap.

Calculate each method's mean, then overlay a labelled summary. Individual points retain their original y values. Each summary clause has only one point per group, so its mean markers have no overlapping peers to displace.

```
stats $Times using ($1==1 ? $2 : 1/0) name "A" nooutput
stats $Times using ($1==2 ? $2 : 1/0) name "B" nooutput
set print $Means
print sprintf("1 %.8f",A_mean)
print sprintf("2 %.8f",B_mean)
set print
set xrange [0.4:2.6]
set yrange [2.2:7.8]
set xtics ("Method A" 1,"Method B" 2)
set ylabel "Duration (minutes)"
set grid ytics
unset key
set jitter swarm spread 1.8 wrap 3
plot $Times using 1:2 with points pt 7 ps 1.6 lc rgb "#00549F", \
    $Means using 1:2 with points pt 2 ps 2.4 lw 3 \
    lc rgb "#F6A800" title "Mean", \
    $Means using 1:2:(sprintf("mean %.2f",$2)) with labels \
    offset 6,0 tc rgb "#F6A800" notitle
print sprintf("swarm means: A=%.8f B=%.8f",A_mean,B_mean)
```



PLT-21-018 · Generated from the complete Bash block

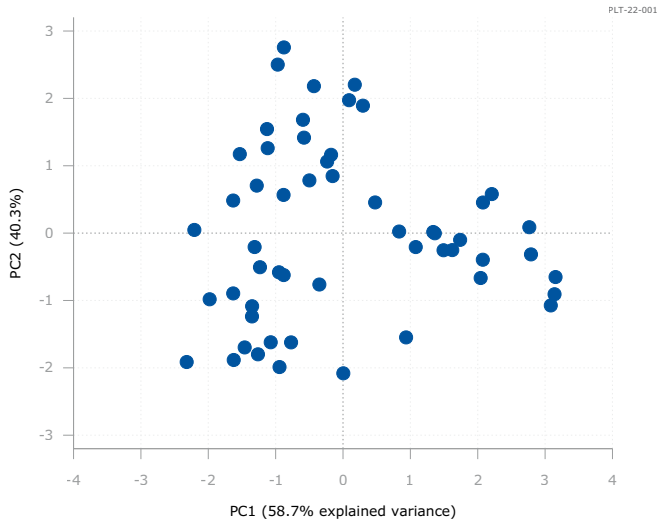
`stats ... ? $2 : 1/0`	Select each group before calculating its mean.
`one mean per group`	The summary clause has no overlapping peers, so its markers remain centred.
`$Means`	Create a computed two-row summary table.

Plot prepared PCA scores

New baseline: the commands below define this example.

Plot PC1 against PC2 for 54 synthetic samples. PCA was calculated from four centred, standardised variables before plotting; the axis percentages report its explained variance. The prepared score table is included in the complete Bash block, so this example runs with gnuplot alone.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
unset key
plot $Scores using 1:2 with points pt 7 ps 1.25 lc rgb "#00549F"
```



PLT-22-001 · Generated from the complete Bash block

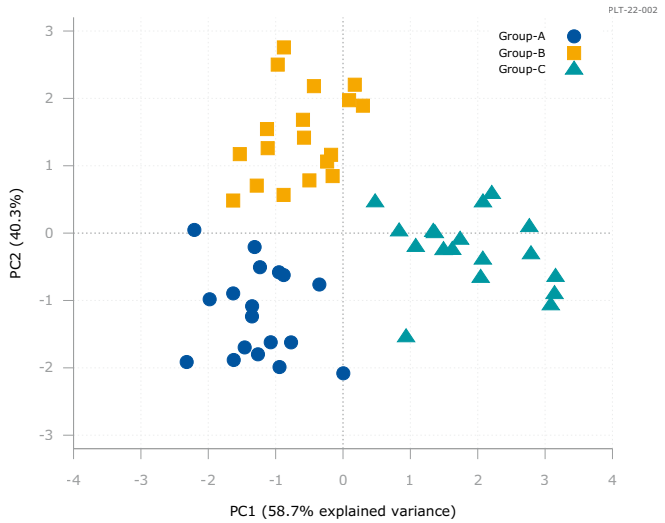
`$Scores using 1:2`	Read the first two component scores from the inline table in the complete Bash block.
`with points pt 7 ps 1.25`	Draw separate circular observations without joining them.
`58.7% / 40.3%`	Variance explained by PC1 and PC2, computed during PCA; gnuplot does not calculate these percentages.
`size ratio -1; zeroaxes`	Keep equal score units in both directions and show where each component score is zero.

Distinguish PCA groups by colour and symbol

Compare with PLT-22-001 · Plot prepared PCA scores.

Use three colours and three shapes to distinguish the sample groups. Group labels describe the observations; all samples share the same PCA transformation. The complete Bash block includes the same 54 prepared score rows as PLT-22-001.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
set style line 1 lc rgb "#00549F" pt 7 ps 1.25
set style line 2 lc rgb "#F6A800" pt 5 ps 1.25
set style line 3 lc rgb "#0098A1" pt 9 ps 1.25
set key top right font ",16" opaque
plot for [g=1:3] $Scores using ($4==g ? $1 : 1/0):2 \
    with points ls g title word("Group-A Group-B Group-C",g)
```



PLT-22-002 · Generated from the complete Bash block

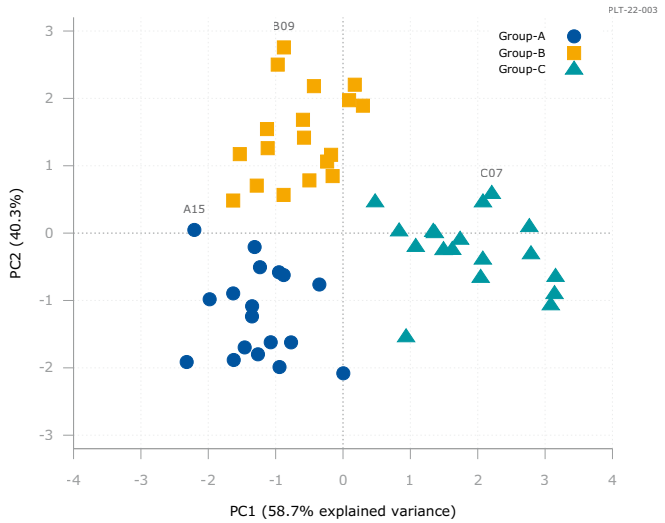
`$4==g ? $1 : 1/0`	Plot the PC1 coordinate only for rows belonging to group g; skip other rows.
`ls g`	Use the matching colour, point symbol and size from a numbered line style.
`word(...,g); key opaque`	Name each group and give its legend a white background.

Label selected PCA samples

Compare with PLT-22-002 · Distinguish PCA groups by colour and symbol.

Label one selected sample near the top of each group. Selective annotation keeps the scatter readable. The complete Bash block contains both the 54 score rows and the three matching label positions.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
set style line 1 lc rgb "#00549F" pt 7 ps 1.25
set style line 2 lc rgb "#F6A800" pt 5 ps 1.25
set style line 3 lc rgb "#0098A1" pt 9 ps 1.25
set key top right font ",16" opaque
plot for [g=1:3] $Scores using ($4==g ? $1 : 1/0):2 \
    with points ls g title word("Group-A Group-B Group-C",g), \
    $Labels using 1:2:3 with labels offset 0,1 \
    font ",16" textcolor rgb "#646567" notitle
```



PLT-22-003 · Generated from the complete Bash block

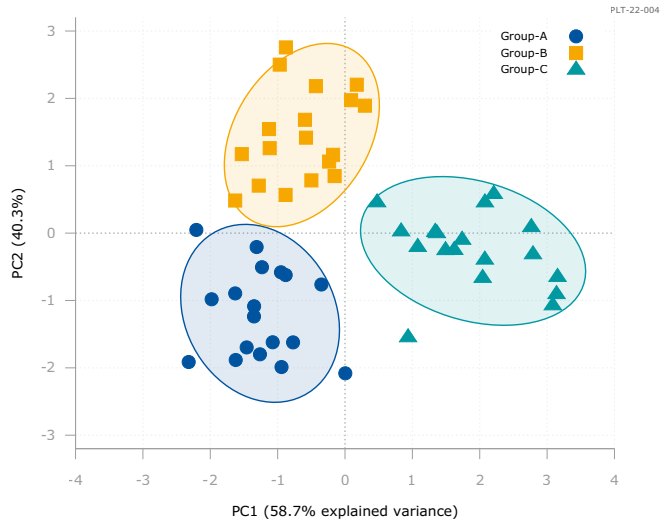
`$Labels using 1:2:3`	Read the selected PC1, PC2 and quoted sample identifier.
`with labels offset 0,1`	Place each identifier one character-height above its point.
`notitle`	Keep annotation text out of the group legend.

Draw PCA group-spread ellipses

Compare with PLT-22-002 · Distinguish PCA groups by colour and symbol.

These ellipses describe group spread, with semiaxes of two sample standard deviations along each group covariance eigenvector. They are not confidence regions for a group mean. Their centres, diameters and angles were prepared from the scores and are included in the complete Bash block.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
set style line 1 lc rgb "#00549F" pt 7 ps 1.25
set style line 2 lc rgb "#F6A800" pt 5 ps 1.25
set style line 3 lc rgb "#0098A1" pt 9 ps 1.25
set key top right font ",16" opaque
set linetype 1 lc rgb "#00549F"
set linetype 2 lc rgb "#F6A800"
set linetype 3 lc rgb "#0098A1"
set style fill transparent solid 0.12 border
plot $Ellipses using 1:2:3:4:5:6 \
    with ellipses units xy lc variable lw 2 notitle, \
    for [g=1:3] $Scores using ($4==g ? $1 : 1/0):2 \
    with points ls g title word("Group-A Group-B Group-C",g)
```



PLT-22-004 · Generated from the complete Bash block

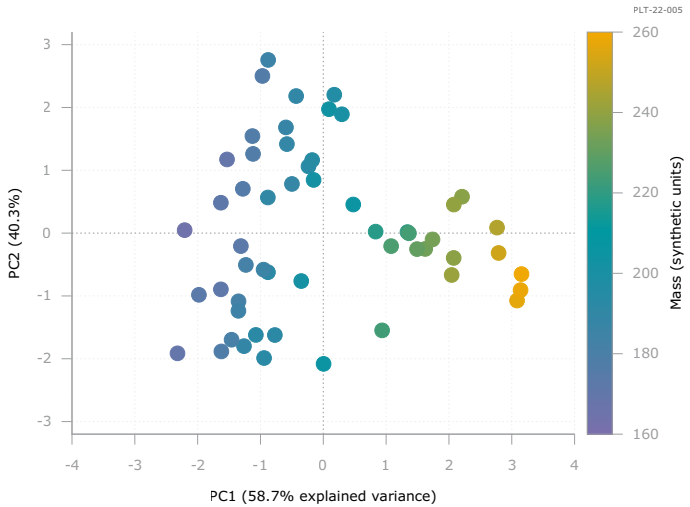
`using 1:2:3:4:5:6`	Read centre x/y, major/minor diameters, angle in degrees, and group colour index.
`diameters = 4*sqrt(eigenvalue)`	Full ellipse diameters correspond to semiaxes of two standard deviations.
`units xy; size ratio -1`	Interpret ellipse geometry in score coordinates, preserving equal physical axis scales.
`lc variable; transparent solid 0.12`	Match group colours and keep observations visible through the fill.

Colour PCA points by a continuous measurement

Compare with PLT-22-001 · Plot prepared PCA scores.

Colour the same scores by the original synthetic mass measurement instead of discrete group membership. The colourbar describes mass, while position still describes PC1 and PC2. All 54 score rows and mass values are included in the complete Bash block.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
unset key
set palette defined (0 "#7A6FAC",0.5 "#0098A1",1 "#F6A800")
set cbrange [160:260]
set cblabel "Mass (synthetic units)"
plot $Scores using 1:2:5 with points pt 7 ps 1.4 lc palette
```



PLT-22-005 · Generated from the complete Bash block

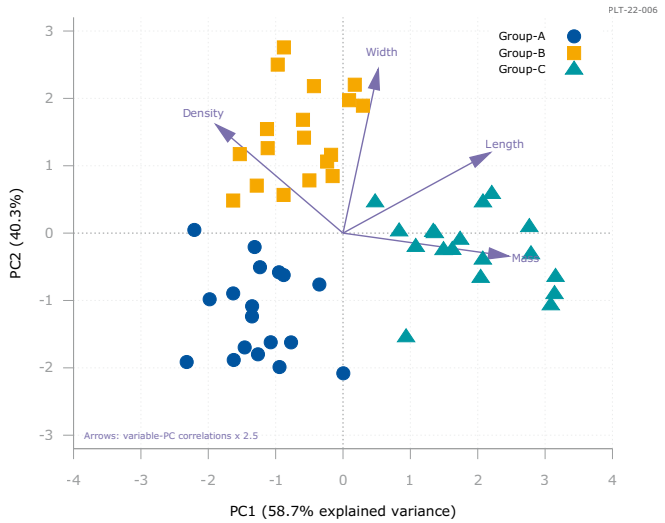
`using 1:2:5`	Read PC1 and PC2 for position and column five for the colour value.
`lc palette; cbrange [160:260]`	Map mass to a continuous colour scale over an explicit common range.
`palette defined; cblabel`	Set colour anchors and explain the separate measurement encoded by colour.

Overlay variable arrows in a PCA biplot

Compare with PLT-22-002 · Distinguish PCA groups by colour and symbol.

Overlay variable-PC correlations, magnified by 2.5 for visibility, on the group score plot. These arrows use the same component signs as the scores. Their displayed lengths are scaled correlations, not score distances; this scaling should be stated whenever the plot is shared.

```
set border 3 lw 1.2 lc rgb "#9C9E9F"
set tics nomirror out
set grid back lc rgb "#ECEDED"
set xzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set yzeroaxis lw 1 lc rgb "#9C9E9F" dt 2
set size ratio -1
set xrange [-4:4]
set yrange [-3.2:3.2]
set xlabel "PC1 (58.7% explained variance)"
set ylabel "PC2 (40.3%)"
set style line 1 lc rgb "#00549F" pt 7 ps 1.25
set style line 2 lc rgb "#F6A800" pt 5 ps 1.25
set style line 3 lc rgb "#0098A1" pt 9 ps 1.25
set key top right font ",16" opaque
arrow_scale = 2.5
set label 1 "Arrows: variable-PC correlations x 2.5" \
    at graph 0.02,0.03 front font ",13" textcolor rgb "#7A6FAC"
plot $Loadings using (0):(0):($1*arrow_scale):($2*arrow_scale) \
    with vectors head filled lw 2 lc rgb "#7A6FAC" notitle, \
    for [g=1:3] $Scores using ($4==g ? $1 : 1/0):2 \
    with points ls g title word("Group-A Group-B Group-C",g), \
    $Loadings using ($1*arrow_scale*1.1):($2*arrow_scale*1.1):3 \
    with labels center font ",16" textcolor rgb "#7A6FAC" notitle
```



PLT-22-006 · Generated from the complete Bash block

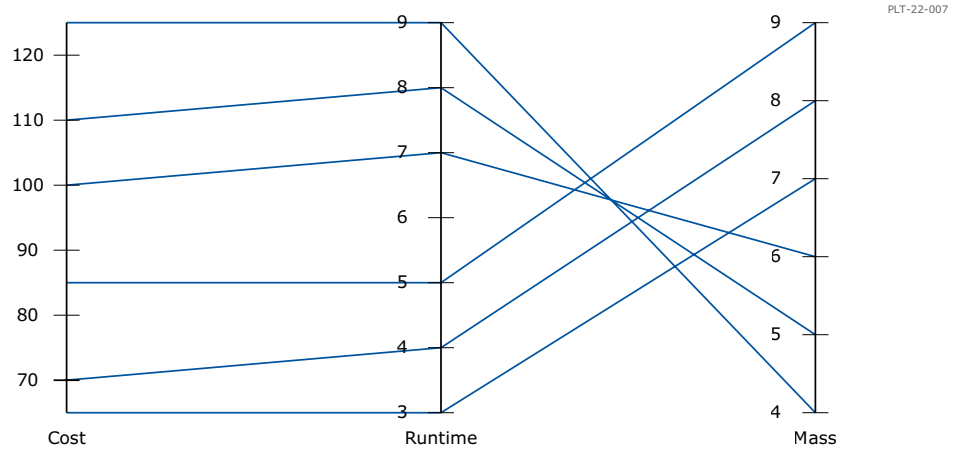
`$Loadings`	Inline columns contain each variable correlation with PC1, its correlation with PC2, and its name.
`with vectors; (0):(0):dx:dy`	Draw arrows from the origin with the specified displacements.
`arrow_scale = 2.5`	Magnify correlations for readability; this is not an additional PCA computation.
`labels at 1.1 times the endpoints`	Place variable names just beyond the arrowheads.

Connect multiple measurements with parallel axes

New baseline: the commands below define this example.

Each of six design alternatives becomes one polyline through cost, runtime and mass. Separate axes allow different units; read each scale rather than treating all positions as a shared numeric y coordinate.

```
$Design << EOD
1 70 4 8
1 85 5 9
1 65 3 7
2 110 8 5
2 125 9 4
2 100 7 6
EOD
set style data parallelaxes
unset border
unset ytics
unset key
set xrange [0.7:3.3]
set xtics ("Cost" 1,"Runtime" 2,"Mass" 3) scale 0
set bmargin 4
set for [i=1:3] paxis i tics
plot $Design using 2 lw 2 lc rgb "#00549F", \
     $Design using 3, $Design using 4
```



PLT-22-007 · Generated from the complete Bash block

`with parallelaxes`	Connect columns from the same row across separate axes.
`using 2, ... using 3, ... using 4`	Select cost, runtime and mass; column 1 is the group.
`paxis i tics`	Show each vertical axis’s numeric scale.

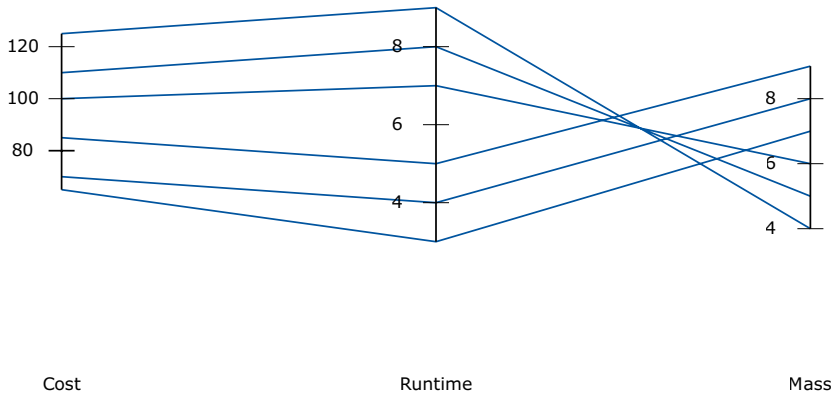
Fix meaningful ranges for each parallel axis

Compare with PLT-22-007 · Connect multiple measurements with parallel axes.

Replace automatic ranges with explicit reference intervals. This avoids exaggerating small variations and makes data updates comparable against the same scales.

```
set style data parallelaxes
unset border
unset ytics
unset key
set xrange [0.7:3.3]
set xtics ("Cost" 1,"Runtime" 2,"Mass" 3) scale 0
set bmargin 4
set for [i=1:3] paxis i tics
set paxis 1 range [0:150]
set paxis 2 range [0:10]
set paxis 3 range [0:12]
plot $Design using 2 lw 2 lc rgb "#00549F", \
     $Design using 3, $Design using 4
```

PLT-22-008



PLT-22-008 · Generated from the complete Bash block

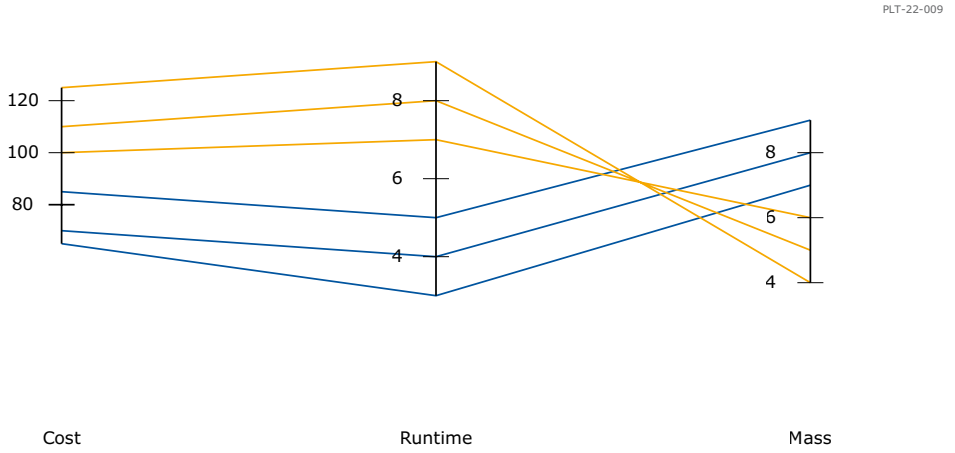
`paxis 1/2/3 range`	Set independent cost, runtime and mass ranges.

Colour parallel lines by design group

Compare with PLT-22-008 · Fix meaningful ranges for each parallel axis.

Add a colour value to the first parallel-axis clause. It follows each row across all three axes. Blue designs cost less and run for less time but have higher mass; orange designs favour runtime and lower mass.

```
set style data parallelaxes
unset border
unset ytics
unset key
set xrange [0.7:3.3]
set xtics ("Cost" 1,"Runtime" 2,"Mass" 3) scale 0
set bmargin 4
set for [i=1:3] paxis i tics
set paxis 1 range [0:150]
set paxis 2 range [0:10]
set paxis 3 range [0:12]
plot $Design using 2:($1==1 ? 0x00549F : 0xF6A800) \
    lw 2 lc rgb variable, $Design using 3, $Design using 4
```



PLT-22-009 · Generated from the complete Bash block

`using 2:(colour)`	The first clause supplies colour for all segments of a row.
`column 1`	Groups designs; it is not a plotted measurement.

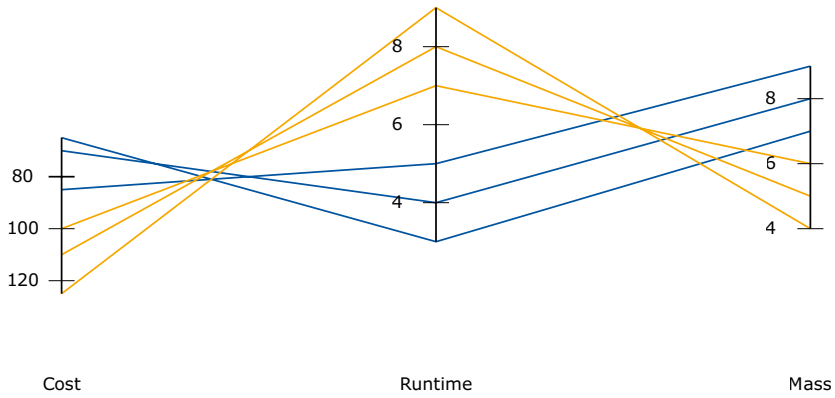
Reverse the cost axis to align preferences

Compare with PLT-22-009 · Colour parallel lines by design group.

Reverse only cost. Lower cost and longer runtime now appear higher; mass still needs reversal for lower values to appear higher. Reversing an axis changes visual direction rather than stored measurements.

```
set style data parallelaxes
unset border
unset ytics
unset key
set xrange [0.7:3.3]
set xtics ("Cost" 1,"Runtime" 2,"Mass" 3) scale 0
set bmargin 4
set for [i=1:3] paxis i tics
set paxis 1 range [0:150]
set paxis 2 range [0:10]
set paxis 3 range [0:12]
set paxis 1 range [0:150] reverse
plot $Design using 2:($1==1 ? 0x00549F : 0xF6A800) \
    lw 2 lc rgb variable, $Design using 3, $Design using 4
```

PLT-22-010



PLT-22-010 · Generated from the complete Bash block

`range [0:150] reverse` Place lower costs near the top while retaining cost units.

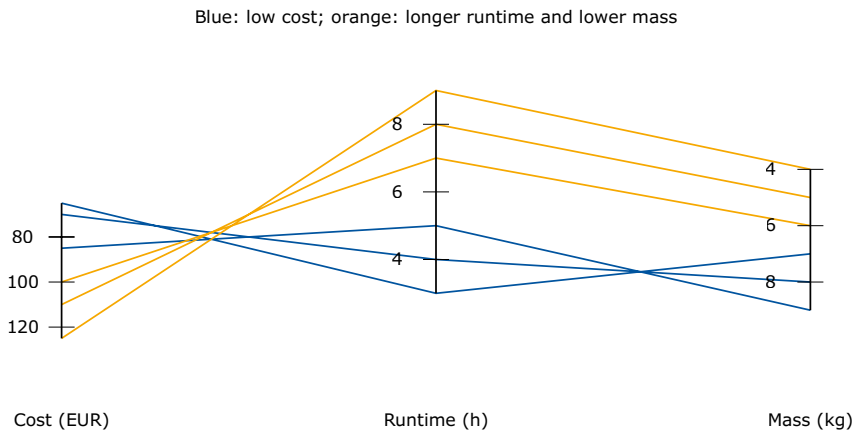
Align all axes so higher means preferable

Compare with PLT-22-010 · Reverse the cost axis to align preferences.

Reverse mass too: lower cost, longer runtime and lower mass all appear towards the top. Label the units. Crossing lines expose trade-offs; separate scales do not assign objective weights or rank alternatives.

```
set style data parallelaxes
unset border
unset ytics
unset key
set xrange [0.7:3.3]
set xtics ("Cost" 1,"Runtime" 2,"Mass" 3) scale 0
set bmargin 4
set for [i=1:3] paxis i tics
set paxis 1 range [0:150]
set paxis 2 range [0:10]
set paxis 3 range [0:12]
set paxis 1 range [0:150] reverse
set paxis 3 range [0:12] reverse
set xtics ("Cost (EUR)" 1,"Runtime (h)" 2,"Mass (kg)" 3) scale 0
set title "Blue: low cost; orange: longer runtime and lower mass"
plot $Design using 2:($1==1 ? 0x00549F : 0xF6A800) \
    lw 2 lc rgb variable, $Design using 3, $Design using 4
```

PLT-22-011



PLT-22-011 · Generated from the complete Bash block

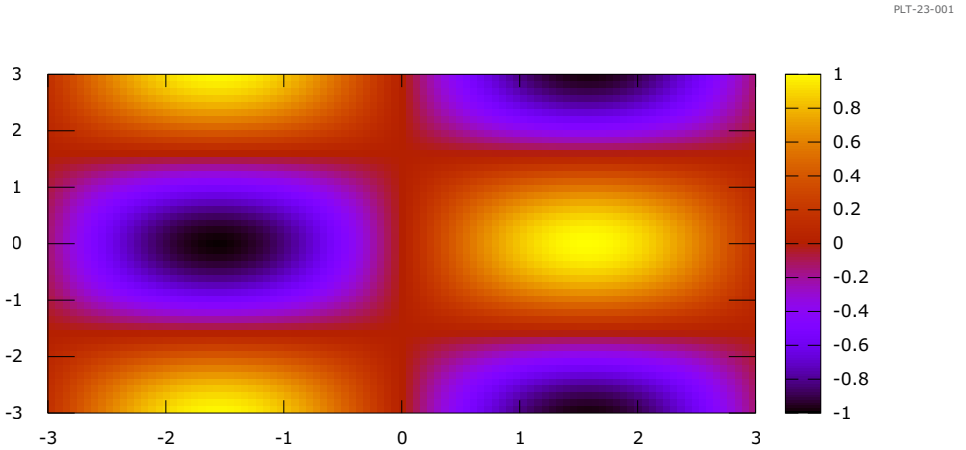
`paxis 3 ... reverse`	Place lower mass higher on its own axis.
`explicit units`	Each axis has a different meaning and range.

Start with a simple colour map

New baseline: the commands below define this example.

A map view projects a coloured surface onto the x/y plane. This is the baseline for the colourbar examples.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-001 · Generated from the complete Bash block

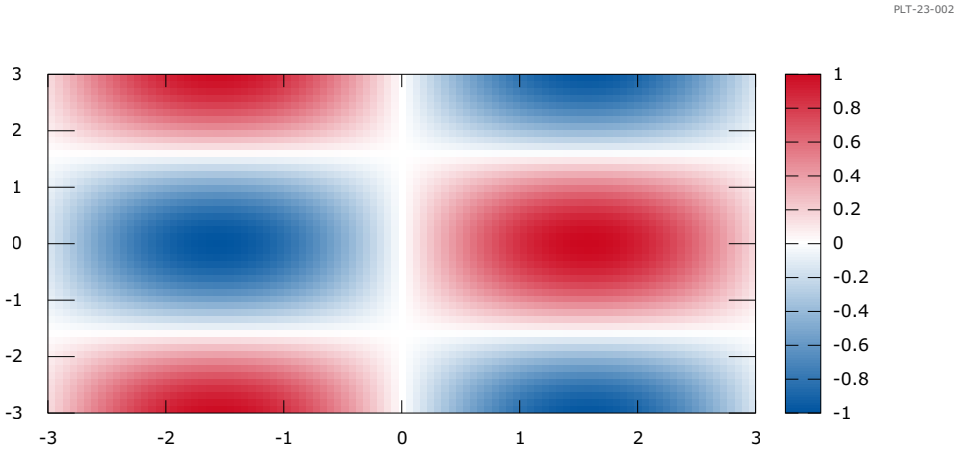
`splot`	Draw a three-dimensional function or dataset.
`sin(x)*cos(y)`	Surface height is a product of two simple functions.
`isosamples 50,50`	Sample 50 grid lines in each direction.
`view map; with pm3d`	Look down from above and colour the sampled surface.

Choose a diverging palette

Compare with PLT-23-001 · Start with a simple colour map.

Three specified colour anchors make zero a neutral midpoint.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-002 · Generated from the complete Bash block

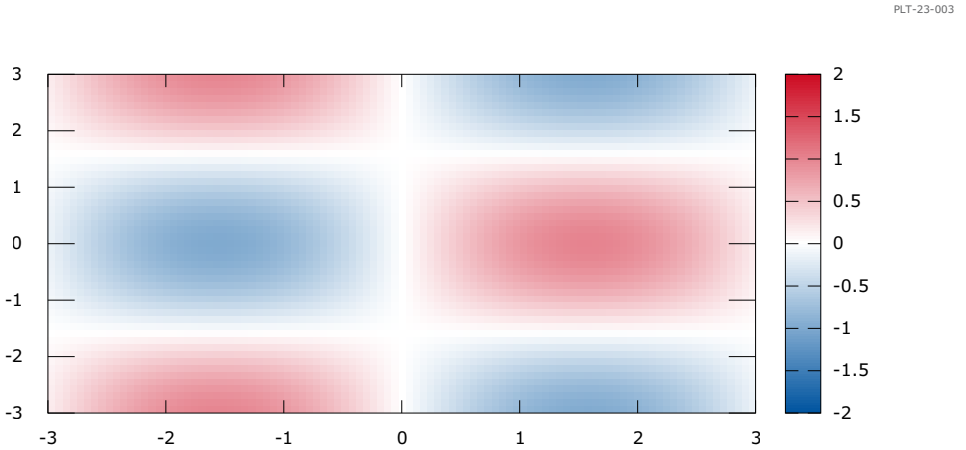
`palette defined`	Interpolate colours between specified value/colour pairs.
`-1 blue; 0 white; 1 red`	Three palette anchors; their values define the palette's internal scale.

Expand the colour range

Compare with PLT-23-002 · Choose a diverging palette.

The same values now occupy the middle half of the palette, so the colours become less saturated. Fixed limits can be reused across related figures.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
set cbrange [-2:2]
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-003 · Generated from the complete Bash block

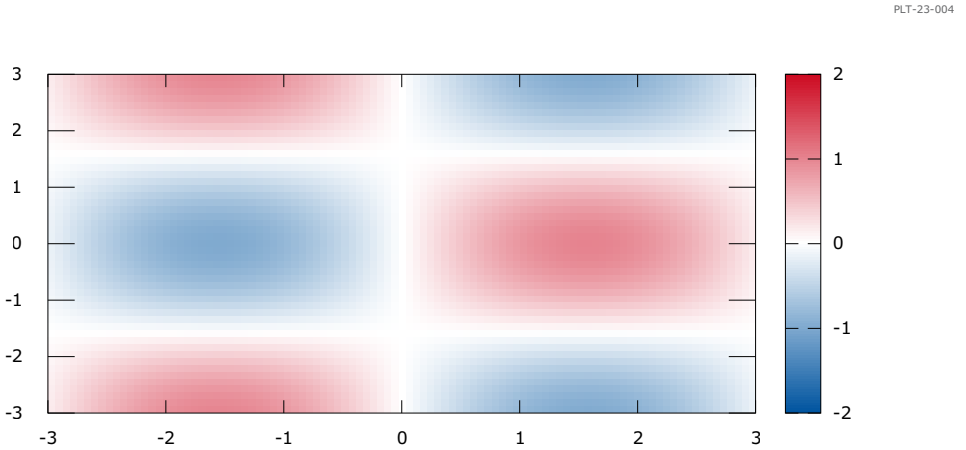
`cbrange`	Range of values mapped to the colour palette.
`[-2:2]`	Map −2 and 2 to the endpoints; the plotted values still lie near −1 to 1.

Set colourbar tick spacing

Compare with PLT-23-003 · Expand the colour range.

Colourbar ticks have their own axis-like controls.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
set cbrange [-2:2]
set cbtics 1
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-004 · Generated from the complete Bash block

`cbtics`	Major colourbar tick settings.
`1`	A labelled colourbar tick every one unit, reducing the number of labels.

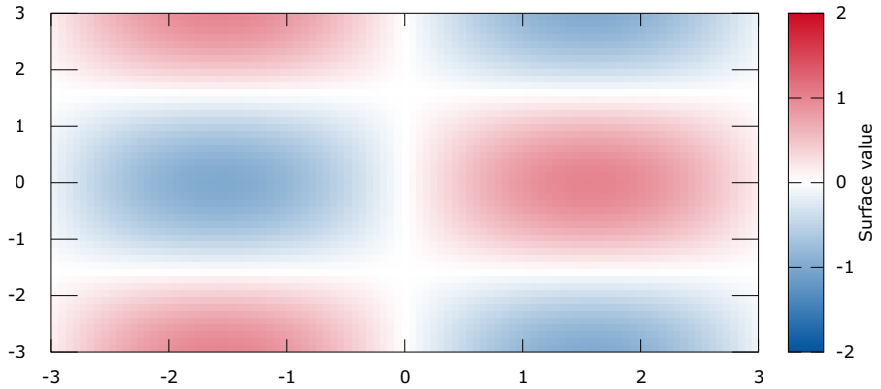
Label the colourbar

Compare with PLT-23-004 · Set colourbar tick spacing.

A colourbar label explains what the colours represent.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
set cbrange [-2:2]
set cbtics 1
set cblabel "Surface value"
splot sin(x)*cos(y) with pm3d notitle
```

PLT-23-005



PLT-23-005 · Generated from the complete Bash block

`cblabel`	Text label beside the colourbar.

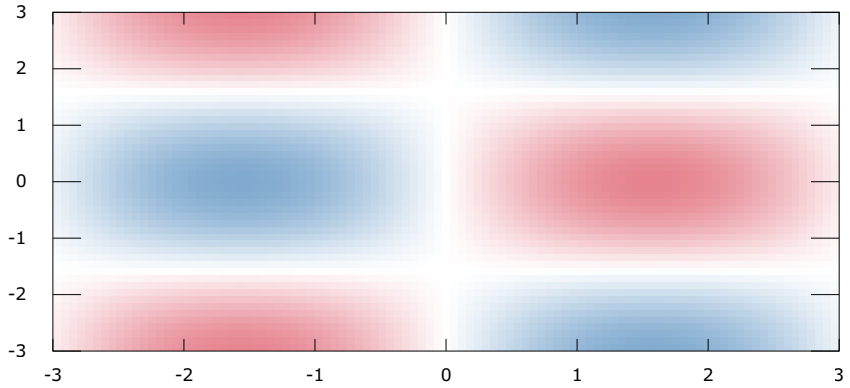
Hide the colourbar

Compare with PLT-23-005 · Label the colourbar.

The surface stays coloured, but the colour key is removed.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
set cbrange [-2:2]
set cbtics 1
set cblabel "Surface value"
unset colorbox
splot sin(x)*cos(y) with pm3d notitle
```

PLT-23-006



PLT-23-006 · Generated from the complete Bash block

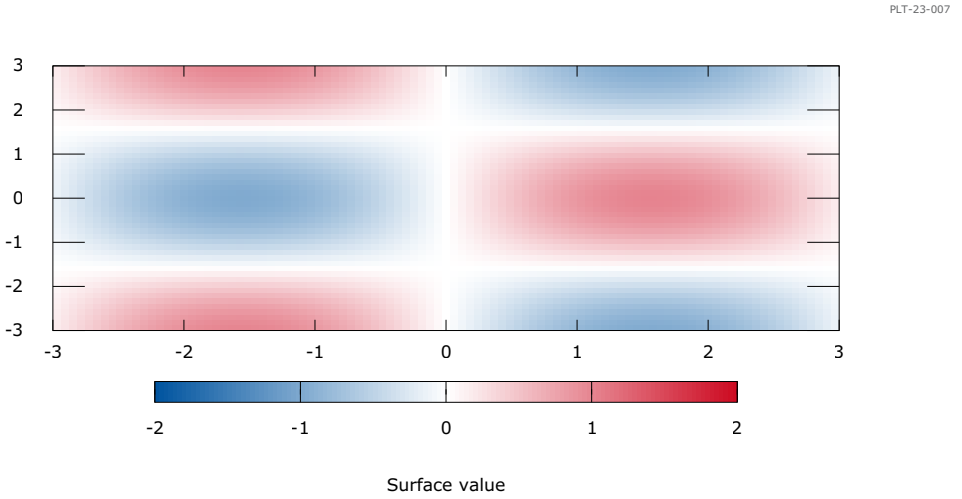
`colorbox`	The rectangular colourbar and its scale.
`unset`	Hide it; the palette still colours the plot.

Place a horizontal colourbar

Compare with PLT-23-005 · Label the colourbar.

An explicitly positioned colourbar needs reserved space below the map.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", \
    0 "#FFFFFF", 1 "#CC071E")
set cbrange [-2:2]
set cbtics 1
set bmargin at screen 0.30
set colorbox horizontal user \
    origin 0.22,0.22 size 0.56,0.04
set cblabel "Surface value" offset 0,-0.5
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-007 · Generated from the complete Bash block

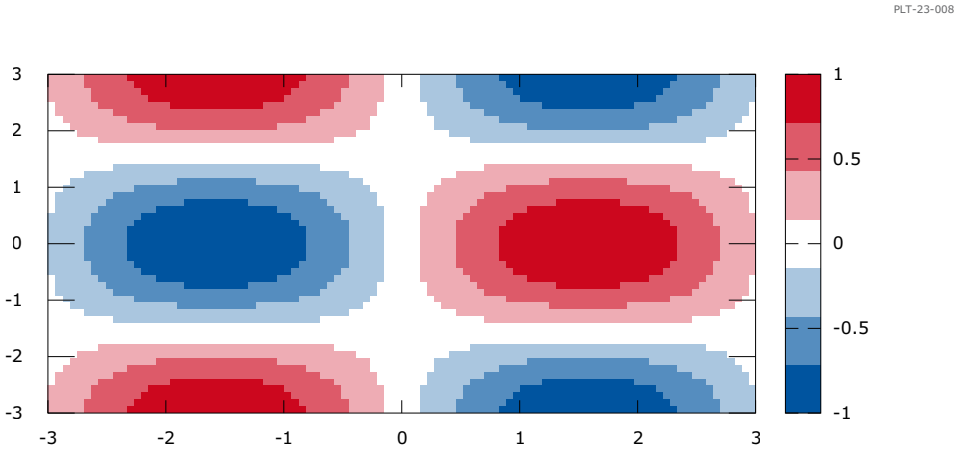
`horizontal user`	Use a horizontal bar with explicit placement.
`origin 0.22,0.22`	Lower-left corner in screen fractions.
`size 0.56,0.04`	Width and height as screen fractions.
`bmargin 0.30; offset`	Reserve lower space; move the colourbar label half a character height downward.

Reduce a palette to discrete colour bands

Compare with PLT-23-002 · Choose a diverging palette.

A small maximum colour count makes the continuous surface appear in bands. The data values remain continuous; only the displayed palette is quantized.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", 0 "white", 1 "#CC071E")
set palette maxcolors 7
set cbrange [-1:1]
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-008 · Generated from the complete Bash block

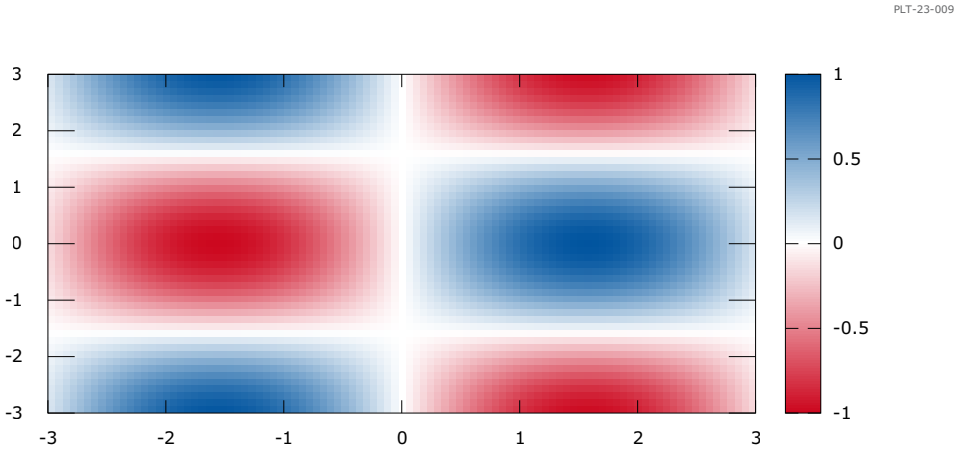
`palette maxcolors 7`	Use seven discrete colours for the palette mapping.
`maxcolors 0`	Restore an unrestricted continuous palette.

Reverse the palette direction

Compare with PLT-23-002 · Choose a diverging palette.

A negative palette reverses which end of the colour gradient represents low and high values. It leaves the data and colour range unchanged.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 50,50
set view map
set palette defined (-1 "#00549F", 0 "white", 1 "#CC071E")
set palette negative
set cbrange [-1:1]
splot sin(x)*cos(y) with pm3d notitle
```



PLT-23-009 · Generated from the complete Bash block

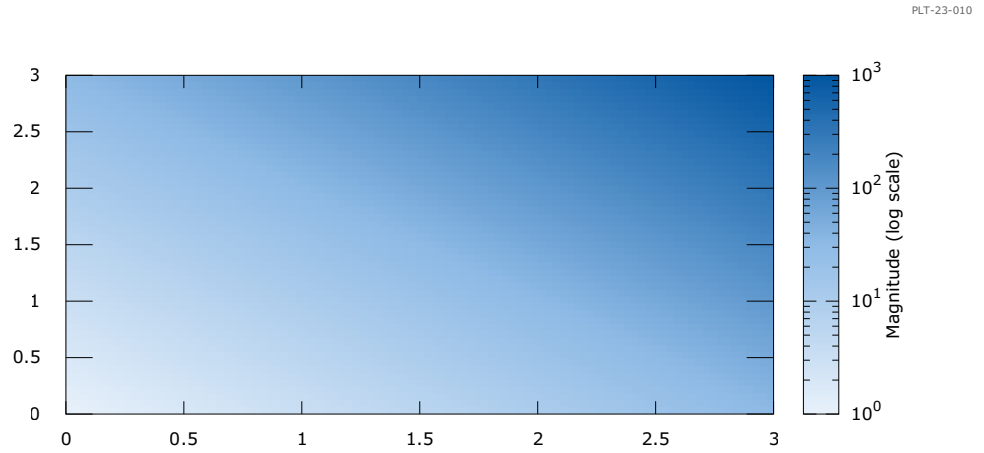
`palette negative`	Reverse the colour gradient.
`palette positive`	Restore the forward direction.

Use a logarithmic colour scale

New baseline: the commands below define this example.

Logarithmic colour mapping gives equal visual space to equal ratios. Every colour value and both colour-range limits must be positive.

```
set xrange [0:3]
set yrange [0:3]
set isosamples 50,50
set view map
set palette defined (0 "#E8F1FA", 1 "#8EBAE5", 2 "#00549F")
set cbrange [1:1000]
set logscale cb 10
set format cb "10^{%L}"
set cblabel "Magnitude (log scale)"
splot 10**((x+y)/2) with pm3d notitle
```



PLT-23-010 · Generated from the complete Bash block

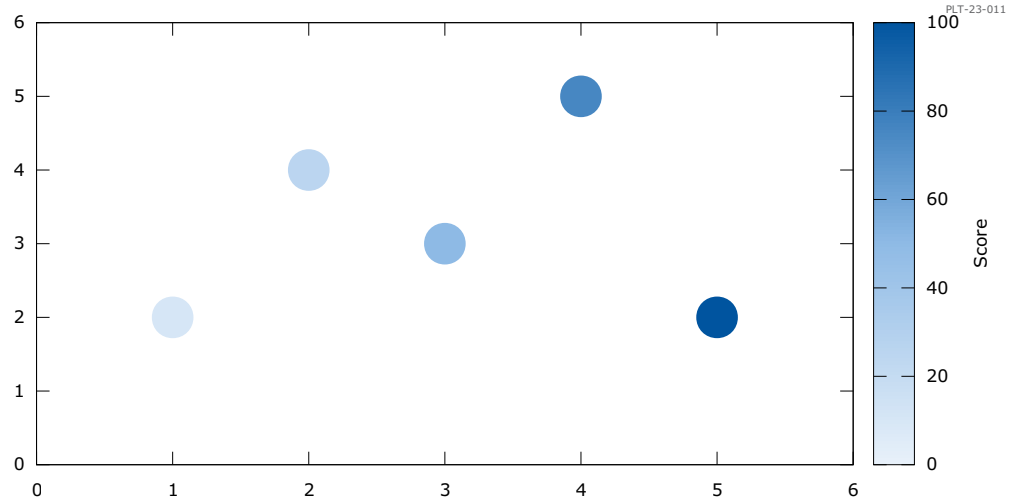
`logscale cb 10`	Apply base-10 logarithmic mapping to the colour axis.
`cbrange [1:1000]`	Cover three positive decades.
`format cb`	Format colourbar labels separately from x and y labels.

Colour points using a third data column

New baseline: the commands below define this example.

The extra using column supplies the value mapped through the palette. Point size stays constant, so colour alone represents the third quantity.

```
$D << EOD
1 2 10
2 4 25
3 3 50
4 5 75
5 2 100
EOD
set xrange [0:6]
set yrange [0:6]
set palette defined (0 "#E8F1FA", 1 "#8EBAE5", 2 "#00549F")
set cbrange [0:100]
set cblabel "Score"
plot $D using 1:2:3 with points pt 7 ps 3 \
    lc palette notitle
```



PLT-23-011 · Generated from the complete Bash block

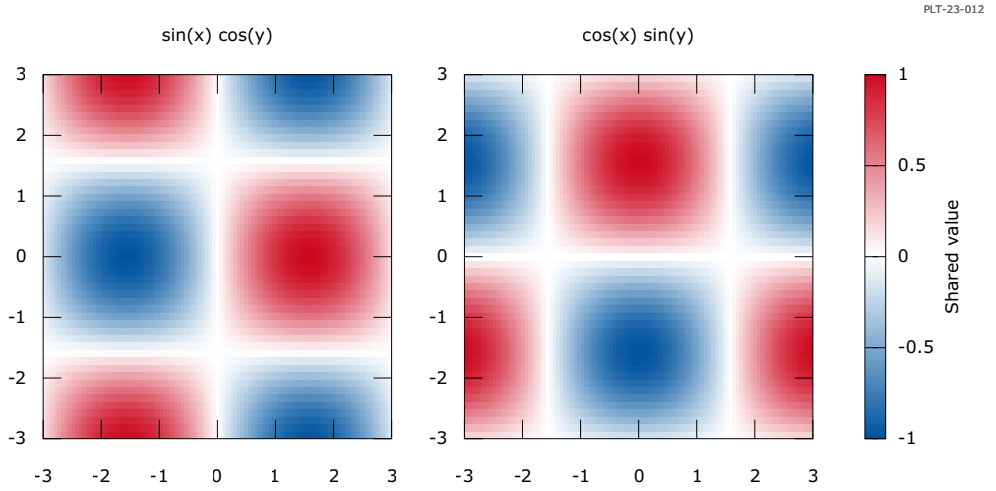
`using 1:2:3`	Read x, y, and the palette value from three columns.
`lc palette`	Map each third-column value through cbrange and the palette.
`ps 3`	Hold symbol size constant.

Share one colourbar across two panels

New baseline: the commands below define this example.

Matching palettes and colour ranges are essential for comparable panels. Reserve space once, hide the first panel colourbar, and draw a single explicitly positioned colourbar with the second.

```
set multiplot layout 1,2 \
    margins 0.08,0.82,0.15,0.85 spacing 0.07,0.05
set xrange [-3:3]
set yrange [-3:3]
set cbrange [-1:1]
set palette defined (-1 "#00549F", 0 "white", 1 "#CC071E")
set view map
set isosamples 50,50
unset colorbox
set title "sin(x) cos(y)"
splot sin(x)*cos(y) with pm3d notitle
set colorbox vertical user origin 0.87,0.15 size 0.02,0.70
set cblabel "Shared value"
set title "cos(x) sin(y)"
splot cos(x)*sin(y) with pm3d notitle
unset multiplot
```



PLT-23-012 · Generated from the complete Bash block

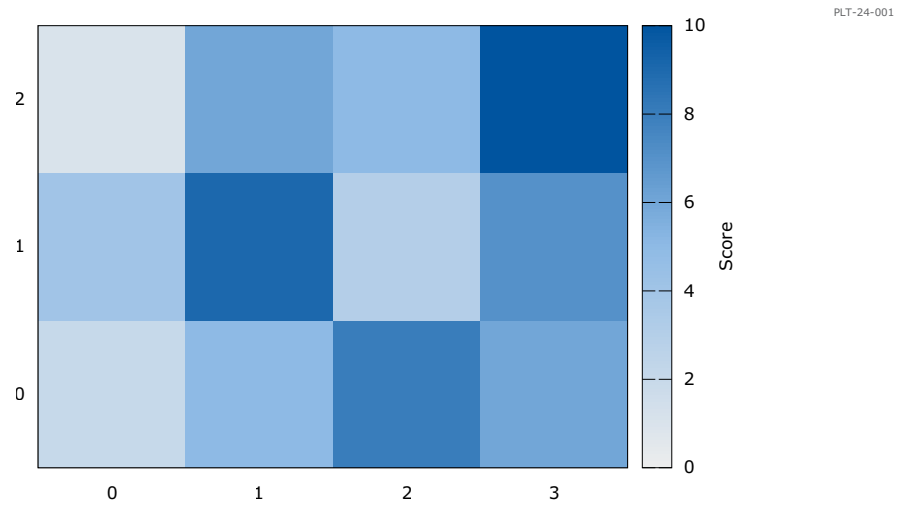
`same cbrange`	Ensure a given colour means the same value in both panels.
`margins ... 0.82`	Reserve the right part of the canvas for a shared colourbar.
`colorbox ... user`	Place that colourbar in canvas coordinates.
`unset colorbox`	Suppress duplicate colourbars.

Draw a heatmap from a matrix

New baseline: the commands below define this example.

Each matrix entry becomes one coloured cell. Columns give x and rows give y, both starting at zero. The first row appears at the bottom; these are cell values, rather than a sampled surface.

```
$M << EOD
2 5 8 6
4 9 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [-0.5:2.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
plot $M matrix with image pixels
```



PLT-24-001 · Generated from the complete Bash block

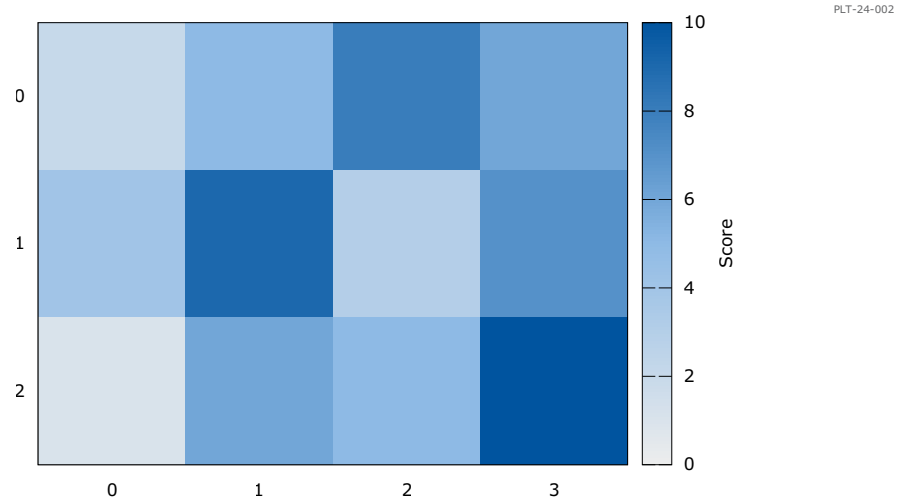
`matrix with image pixels`	Map a rectangular matrix to equally spaced cells using the palette.
`pixels`	Draw cells individually so browsers and PDF renderers preserve sharp boundaries rather than smoothing a tiny embedded bitmap.
`x/y ranges`	Half-cell padding keeps the outside cells complete.
`size ratio -1`	Give x and y equal physical units, making these cells square.
`cbrange [0:10]`	Keep the value-to-colour mapping fixed across comparisons.

Put the first matrix row at the top

Compare with PLT-24-001 · Draw a heatmap from a matrix.

Reverse the y range for a table-like display. The data and palette remain unchanged; only the vertical reading direction changes.

```
$M << EOD
2 5 8 6
4 9 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
plot $M matrix with image pixels
```



PLT-24-002 · Generated from the complete Bash block

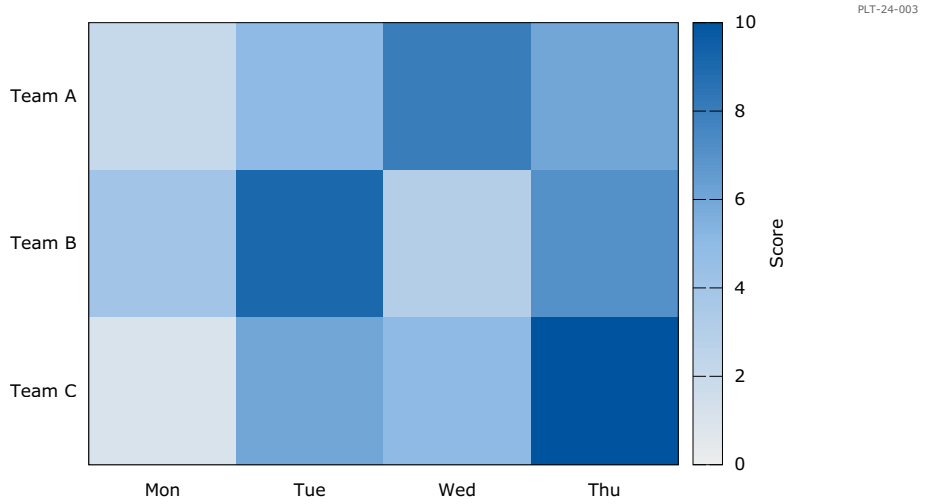
`yrange [2.5:-0.5]`	Display row zero above rows one and two.
`matrix row order`	The input still lists rows in their original order.

Label heatmap rows and columns

Compare with PLT-24-002 · Put the first matrix row at the top.

Replace numeric cell coordinates with category names. Tick positions stay at cell centres, while the colourbar still describes the numeric scores.

```
$M << EOD
2 5 8 6
4 9 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
set xtics ("Mon" 0, "Tue" 1, "Wed" 2, "Thu" 3)
set ytics ("Team A" 0, "Team B" 1, "Team C" 2)
plot $M matrix with image pixels
```



PLT-24-003 · Generated from the complete Bash block

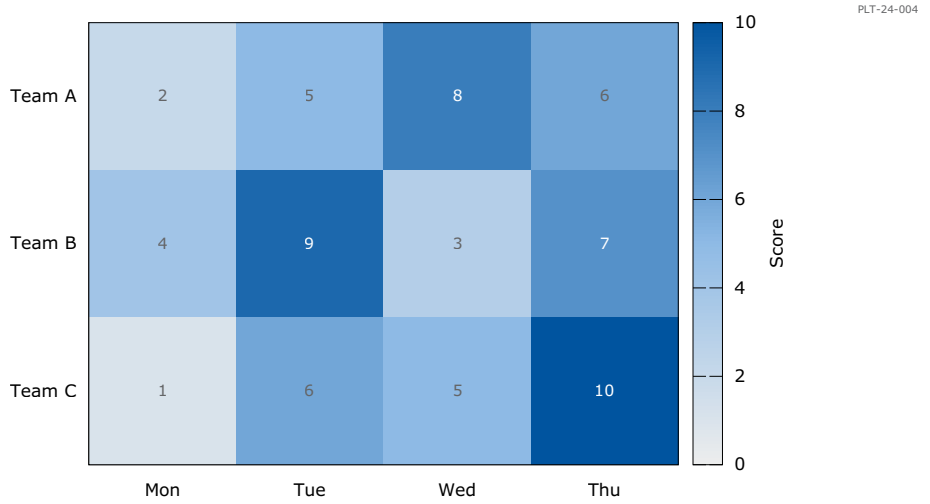
`xtics / ytics lists`	Pair each category label with its zero-based cell position.
`scale 0`	Keep category text without tick strokes.

Print the value inside every cell

Compare with PLT-24-003 · Label heatmap rows and columns.

Read the same matrix a second time with labels. Choose white text on darker cells and dark text on lighter cells so the values remain legible.

```
$M << EOD
2 5 8 6
4 9 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
set xtics ("Mon" 0, "Tue" 1, "Wed" 2, "Thu" 3)
set ytics ("Team A" 0, "Team B" 1, "Team C" 2)
plot $M matrix with image pixels,      $M matrix using 1:2:(sprintf("%.0f",
$3)):($3>=7 ? 0xFFFFFF : 0x646567)      with labels textcolor rgb variable
font ",18"
```



PLT-24-004 · Generated from the complete Bash block

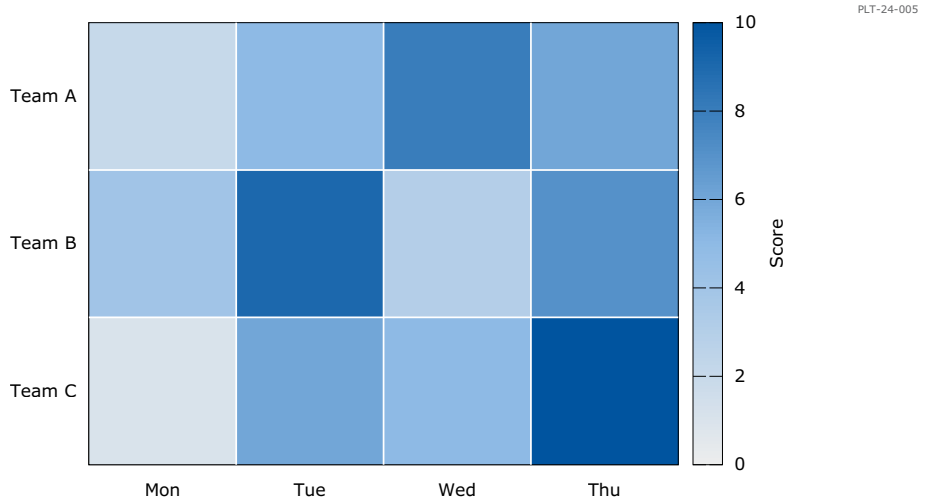
`matrix using 1:2:3`	The generated columns are x, y and the matrix value.
`sprintf("%.0f",$3)`	Format each value as an integer label.
`textcolor rgb variable`	Read the text colour from the fourth using expression.

Draw lines between heatmap cells

Compare with PLT-24-003 · Label heatmap rows and columns.

Place front-layer arrows at half-integer coordinates. The separators follow cell edges, rather than the category ticks at cell centres.

```
$M << EOD
2 5 8 6
4 9 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
set xtics ("Mon" 0, "Tue" 1, "Wed" 2, "Thu" 3)
set ytics ("Team A" 0, "Team B" 1, "Team C" 2)
set for [x=0:2] arrow (x+1) from x+0.5,-0.5 to x+0.5,2.5     nohead front lc
rgb "white" lw 2
set for [y=0:1] arrow (y+10) from -0.5,y+0.5 to 3.5,y+0.5     nohead front
lc rgb "white" lw 2
plot $M matrix with image pixels
```



PLT-24-005 · Generated from the complete Bash block

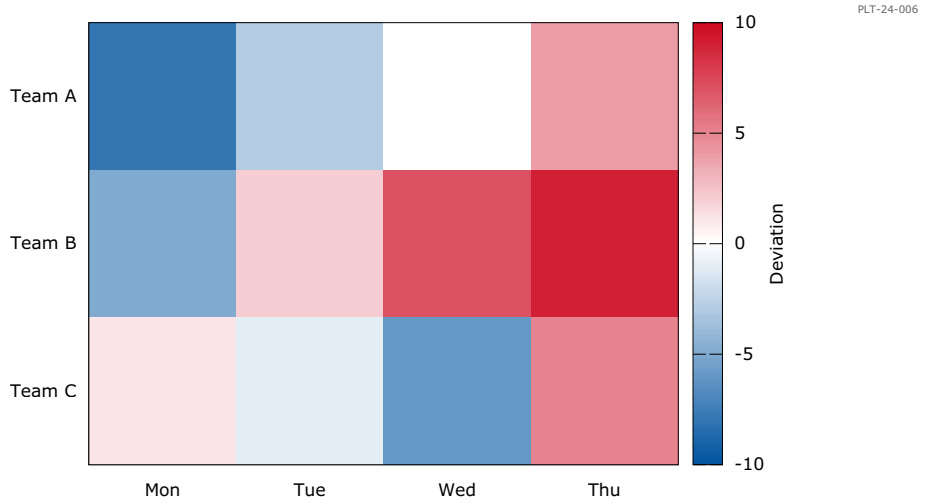
`half-integer edges`	Adjacent matrix cell centres are one unit apart.
`nohead front`	Draw separators above the image without arrowheads.

Use a symmetric palette for signed values

Compare with PLT-24-003 · Label heatmap rows and columns.

A diverging palette makes zero neutral and assigns different colours to negative and positive deviations. The symmetric colour range gives equal magnitudes comparable colour strength.

```
$M << EOD
-8 -3 0 4
-5 2 7 9
1 -1 -6 5
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
set xtics ("Mon" 0, "Tue" 1, "Wed" 2, "Thu" 3)
set ytics ("Team A" 0, "Team B" 1, "Team C" 2)
set cbrange [-10:10]
set palette defined (-10 "#00549F", 0 "#FFFFFF", 10 "#CC071E")
set cblabel "Deviation"
plot $M matrix with image pixels
```



PLT-24-006 · Generated from the complete Bash block

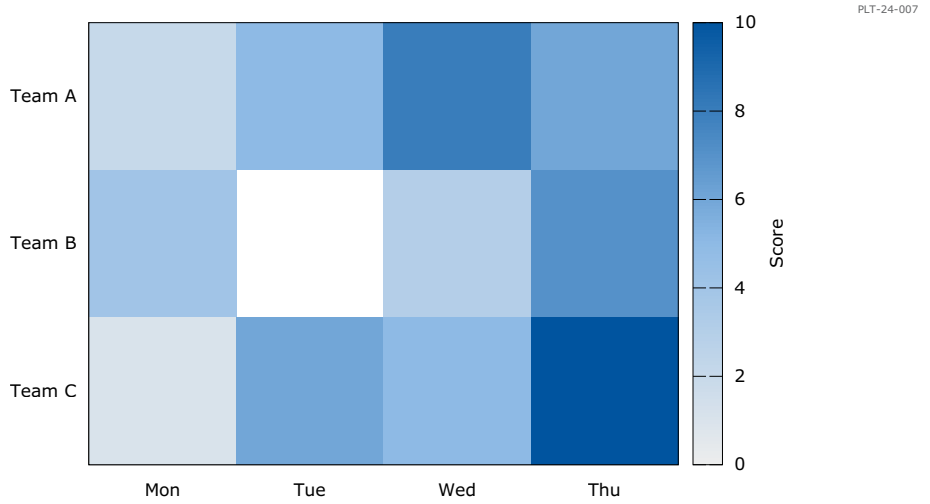
`cbrange [-10:10]`	Centre the colour scale on zero.
`palette defined`	Blue and red encode opposite signs, with white at zero.

Leave missing heatmap cells blank

Compare with PLT-24-003 · Label heatmap rows and columns.

Reserve -999 as a missing-score marker and turn it into an undefined colour value with 1/0. Zero remains a measured score. This SVG leaves the missing cell unpainted, revealing the white canvas; it must not be interpreted as a low score.

```
$M << EOD
2 5 8 6
4 -999 3 7
1 6 5 10
EOD
unset key
set size ratio -1
set xrange [-0.5:3.5]
set yrange [2.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
set cblabel "Score"
set xtics ("Mon" 0, "Tue" 1, "Wed" 2, "Thu" 3)
set ytics ("Team A" 0, "Team B" 1, "Team C" 2)
plot $M matrix using 1:2:($3 == -999 ? 1/0 : $3) with image pixels
```



PLT-24-007 · Generated from the complete Bash block

`$3 == -999 ? 1/0 : $3`	Replace only the reserved missing marker with an undefined value.
`blank cell`	No palette value is drawn for the missing observation.

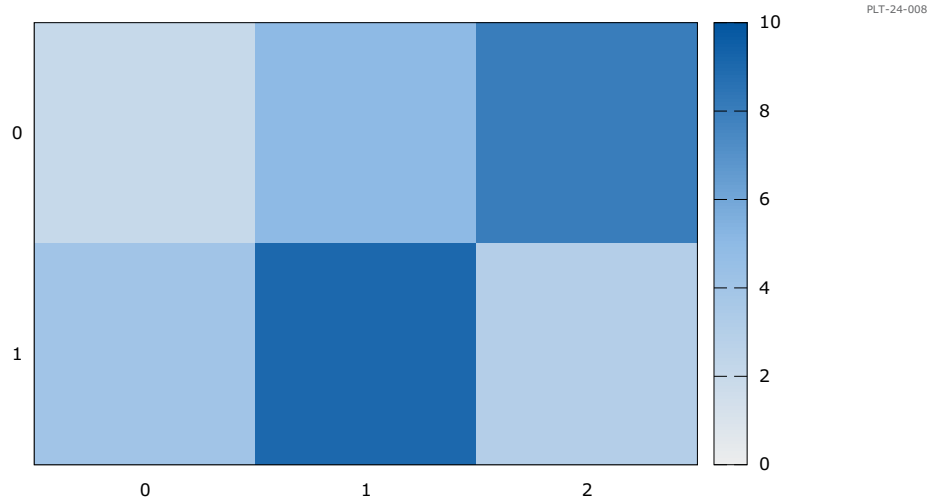
Read a heatmap as x, y and value triples

Compare with PLT-24-001 · Draw a heatmap from a matrix.

A regular grid can also be supplied as explicit coordinates. Blank lines separate scans. Use uniformly spaced x and y coordinates with image pixels; scattered observations need a different gridding or plotting method.

```
$D << EOD
0 0 2
1 0 5
2 0 8

0 1 4
1 1 9
2 1 3
EOD
unset key
set size ratio -1
set xrange [-0.5:2.5]
set yrange [1.5:-0.5]
set xtics 1 nomirror scale 0
set ytics 1 nomirror scale 0
set cbrange [0:10]
set palette defined (0 "#ECEDED", 5 "#8EBAE5", 10 "#00549F")
plot $D using 1:2:3 with image pixels
```



PLT-24-008 · Generated from the complete Bash block

`using 1:2:3`	Read x, y and the palette value directly.
`blank lines`	Separate complete grid scans; this example has two rows.

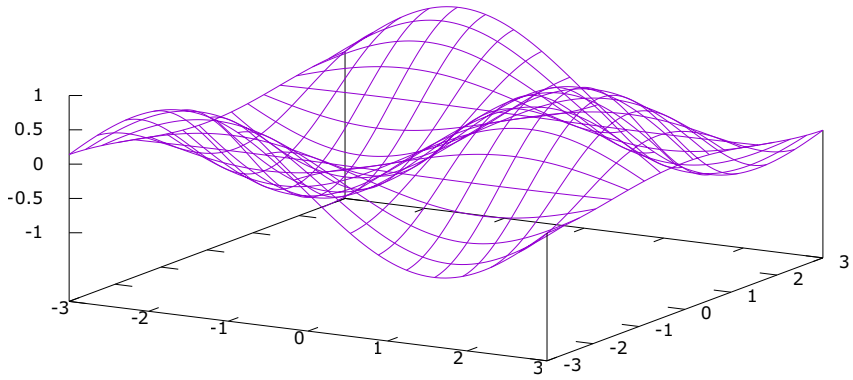
Draw a wireframe surface

New baseline: the commands below define this example.

Move from plot to splot to evaluate a function of two independent variables.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 18,18
set ztics 0.5
set view 60,30
splot sin(x)*cos(y) with lines notitle
```

PLT-25-001



PLT-25-001 · Generated from the complete Bash block

`splot`	Plot a 3D surface z=f(x,y).
`isosamples 18,18`	Use 18 isolines in each grid direction.
`view 60,30`	Set the two camera rotation angles in degrees.

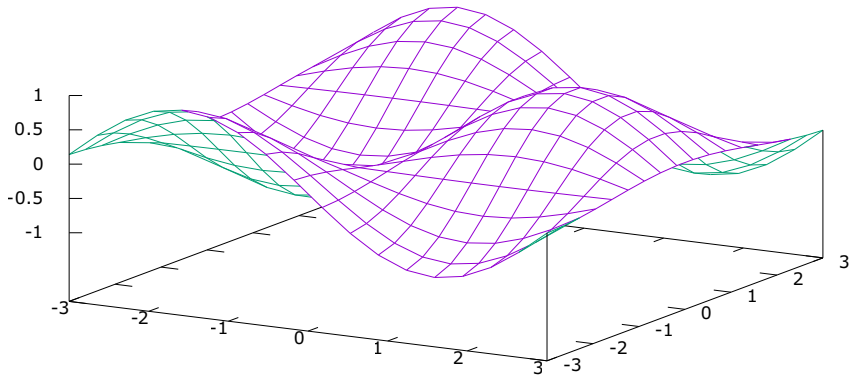
Remove hidden wireframe lines

Compare with PLT-25-001 · Draw a wireframe surface.

Occluded lines disappear behind the surface.

```
set hidden3d
set xrange [-3:3]
set yrange [-3:3]
set isosamples 18,18
set ztics 0.5
set view 60,30
splot sin(x)*cos(y) with lines notitle
```

PLT-25-002



PLT-25-002 · Generated from the complete Bash block

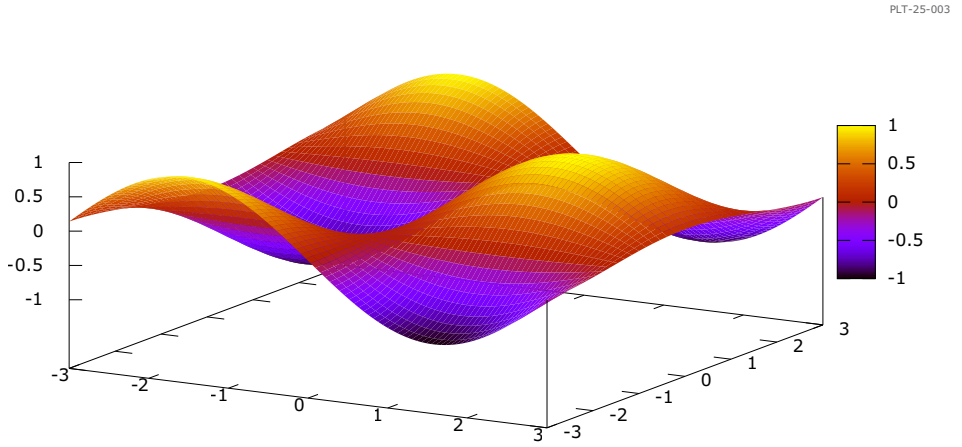
`hidden3d`	Perform hidden-line removal for line surfaces.

Colour the surface height

Compare with PLT-25-001 · Draw a wireframe surface.

Filled grid cells replace the wireframe; the colourbar explains the height-to-colour mapping.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 40,40
set ztics 0.5
set cbtics 0.5
set view 60,30
splot sin(x)*cos(y) with pm3d notitle
```



PLT-25-003 · Generated from the complete Bash block

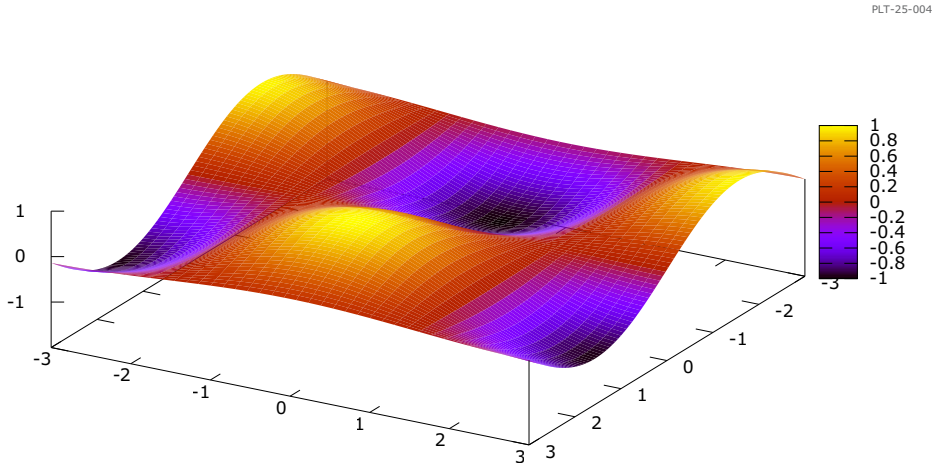
`with pm3d`	Fill grid cells with palette colours determined by z.
`40,40`	Increase mesh density for a smoother-looking surface.

Change the surface viewing angle

Compare with PLT-25-003 · Colour the surface height.

Changing the view rotates the projection without changing the surface data. A map view removes perspective and is often easier for reading contour positions.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 40,40
set view 35,120
set ztics 1
splot sin(x)*cos(y) with pm3d notitle
```



PLT-25-004 · Generated from the complete Bash block

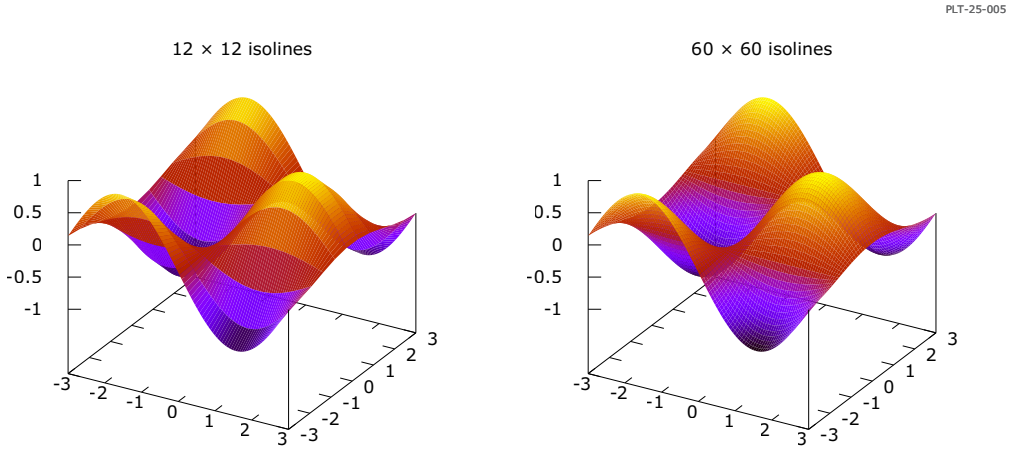
`view 35,120`	Set the x-axis rotation and z-axis rotation in degrees.
`ztics 1`	Keep the shortened projected z axis readable with only three major ticks.
`view map`	Select the top-down map projection used in the colour-map examples.

Compare coarse and fine surface sampling

New baseline: the commands below define this example.

Isosamples sets the grid density for a function surface. Higher density produces smoother colour patches at a greater rendering cost; it does not interpolate measurements from a data file.

```
set multiplot layout 1,2
set xrange [-3:3]
set yrange [-3:3]
set zrange [-1:1]
set cbrange [-1:1]
unset colorbox
set view 60,30
set title "12 × 12 isolines"
set isosamples 12,12
splot sin(x)*cos(y) with pm3d notitle
set title "60 × 60 isolines"
set isosamples 60,60
splot sin(x)*cos(y) with pm3d notitle
unset multiplot
```



PLT-25-005 · Generated from the complete Bash block

`isosamples nx,ny`	Control the number of isolines in each function-surface direction.
`same ranges and view`	Hold geometry and colour mapping fixed while comparing density.

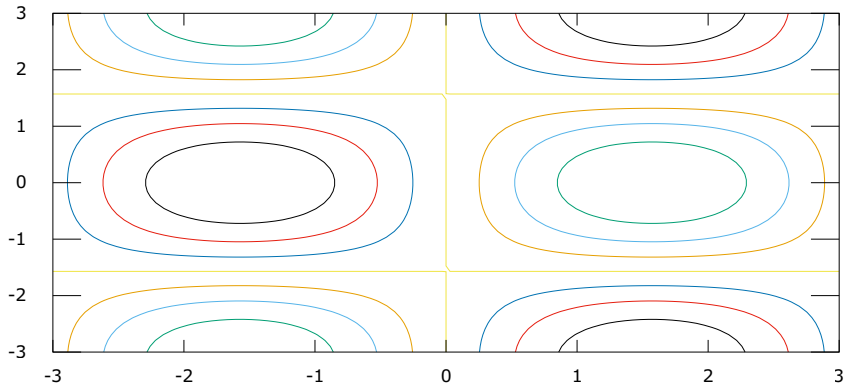
Project contour lines into a map

New baseline: the commands below define this example.

Contours join points of equal height. Suppress the surface and look straight down at its base plane.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 60,60
set contour base
set cntrparam levels incremental -0.75,0.25,0.75
unset surface
set view map
splot sin(x)*cos(y) with lines notitle
```

PLT-25-006



PLT-25-006 · Generated from the complete Bash block

`contour base`	Generate contours projected onto the base plane.
`levels incremental -0.75,0.25,0.75`	Start at -0.75, increment by 0.25 and stop at 0.75.
`unset surface / view map`	Hide the mesh and use a top-down view.

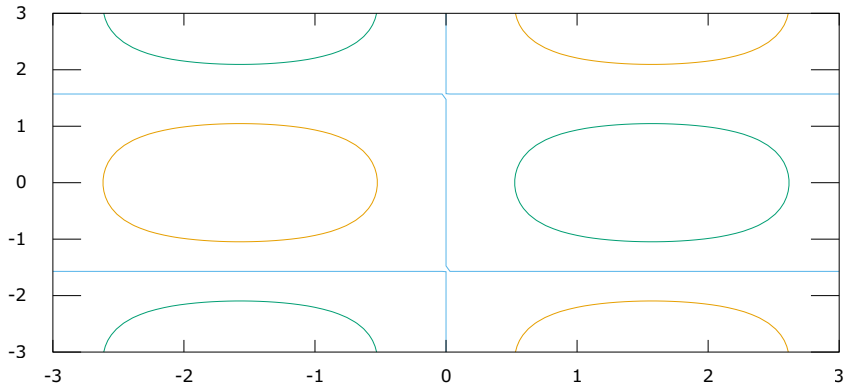
Choose only specific contour levels

Compare with PLT-25-006 · Project contour lines into a map.

A discrete list focuses on selected thresholds.

```
set xrange [-3:3]
set yrange [-3:3]
set isosamples 60,60
set contour base
set cntrparam levels discrete -0.5,0,0.5
unset surface
set view map
splot sin(x)*cos(y) with lines notitle
```

PLT-25-007



PLT-25-007 · Generated from the complete Bash block

`levels discrete`	Generate exactly the three listed contour values.

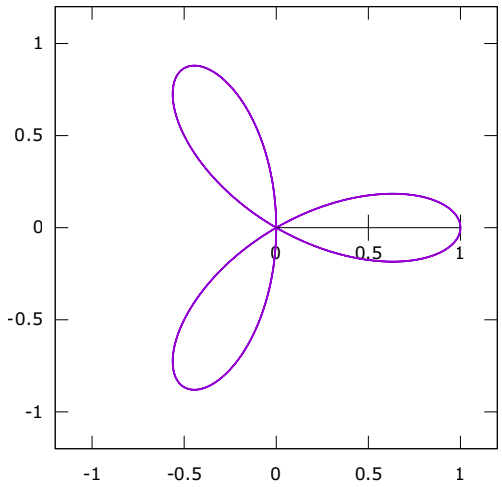
Draw a polar curve

New baseline: the commands below define this example.

In polar mode the function returns radius as a function of angle t.

```
set polar
set angles radians
set trange [0:2*pi]
set samples 400
set size ratio -1
set xrange [-1.2:1.2]
set yrange [-1.2:1.2]
plot cos(3*t) with lines linewidth 2 notitle
```

PLT-25-008



PLT-25-008 · Generated from the complete Bash block

`trange [0:2*pi]`	Sweep one full rotation in radians.
`cos(3*t)`	Radius changes with angle; negative radii are reflected through the origin, producing three petals.
`samples 400`	Sample the angular parameter densely enough for a smooth curve.

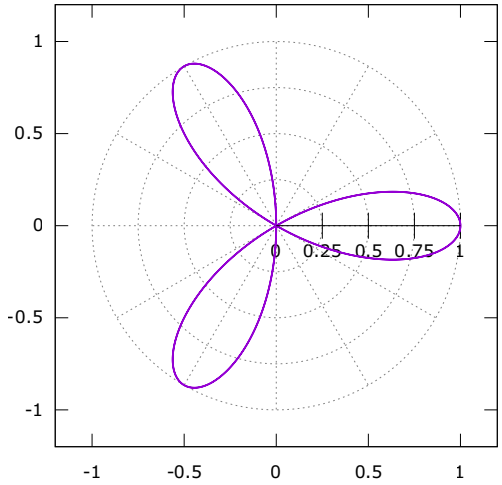
Add a polar grid

Compare with PLT-25-008 · Draw a polar curve.

Angular spokes and radial rings make positions easier to estimate.

```
set polar
set angles radians
set trange [0:2*pi]
set samples 400
set grid polar pi/6
set rtics 0.25
set rrange [0:1]
set size ratio -1
set xrange [-1.2:1.2]
set yrange [-1.2:1.2]
plot cos(3*t) with lines linewidth 2 notitle
```

PLT-25-009



PLT-25-009 · Generated from the complete Bash block

`grid polar pi/6`	Draw spokes every pi/6 radians (30 degrees).
`rtics 0.25`	Draw radial ticks at quarter-unit intervals.
`rrange [0:1]`	Set the displayed radial axis from zero to one.

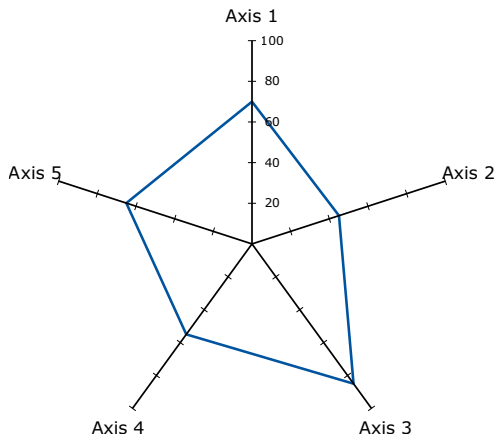
Start a native spider plot

New baseline: the commands below define this example.

Each data row makes a polygon and each selected column makes a spoke. Axis one points upward; the remaining axes proceed clockwise at equal angles. Fix the spoke ranges so polygon sizes remain comparable.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label sprintf("Axis %d",p)
set style spiderplot fs empty border lw 3
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-001



PLT-26-001 · Generated from the complete Bash block

`set spiderplot`	Enable radar coordinates, hide Cartesian ticks and borders, and make the plot square.
`paxis p range [0:100]`	Map zero to the centre and 100 to the end of each spoke.
`using i`	Select a different column for each successive spoke.
`paxis p tics format ""`	Hide repeated numeric labels, retaining tick positions.

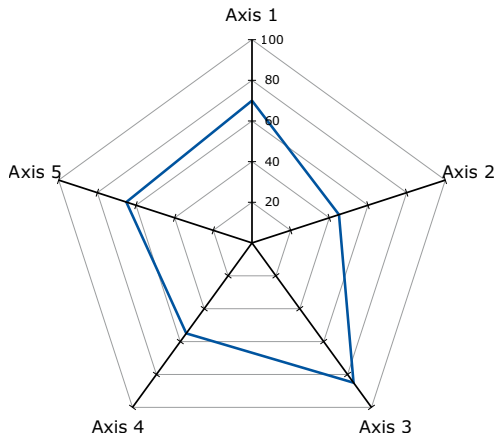
Add the polygonal spiderweb

Compare with PLT-26-001 · Start a native spider plot.

Connect matching tick levels across the spokes to form nested polygons. The grid sits behind the data. These rings use spoke tick positions rather than Cartesian x/y ticks.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label sprintf("Axis %d",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-002



PLT-26-002 · Generated from the complete Bash block

`grid spiderplot`	Draw a web connecting radial axis ticks.
`back`	Draw the grid underneath the data polygon.

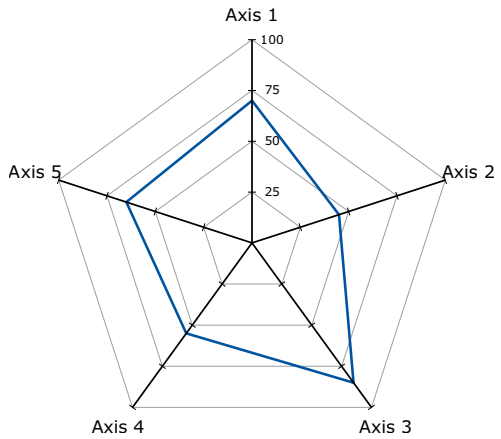
Change the number of web rings

Compare with PLT-26-002 · Add the polygonal spiderweb.

A tick interval of 25 produces fewer web levels than an interval of 20. Apply the same positions to all spokes for a coherent grid.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 25 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label sprintf("Axis %d",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-003



PLT-26-003 · Generated from the complete Bash block

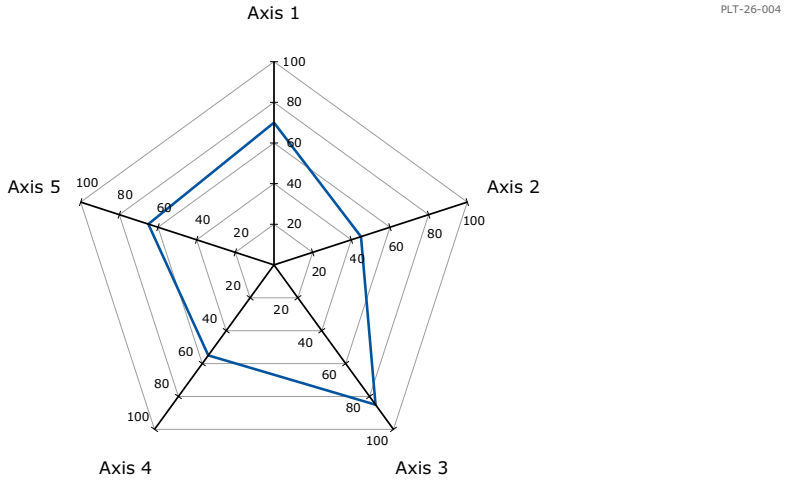
`paxis p tics 25`	Place radial ticks at multiples of 25 on each 0-to-100 axis.

Show numeric notation on every spoke

Compare with PLT-26-002 · Add the polygonal spiderweb.

Remove the empty formats to repeat numeric tick labels around the web. This is useful for separate units, but shared scales are usually clearer when labelled on just one spoke.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=1:5] paxis p label sprintf("Axis %d",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set for [p=1:5] paxis p label offset 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```



PLT-26-004 · Generated from the complete Bash block

`tics font ",14"`	Use smaller numeric text than the spoke names.
`format ""`	Omitting this suppression restores the default numeric labels.

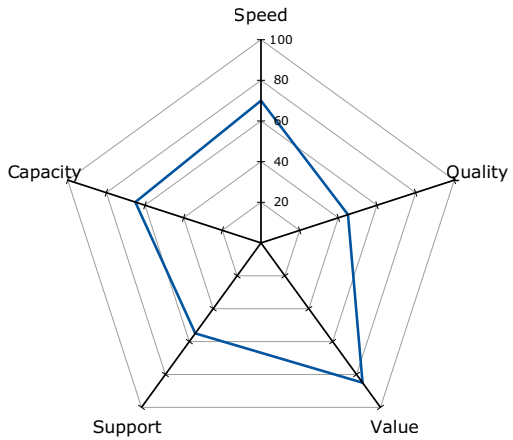
Give each spoke a meaningful name

Compare with PLT-26-002 · Add the polygonal spiderweb.

Set axis names independently of the dataset. Use notitle in the plot command so a component title does not replace the paxis label.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-005



PLT-26-005 · Generated from the complete Bash block

`paxis p label`	Name the corresponding spoke.
`word(text,p)`	Select the p-th space-separated name.
`notitle`	Preserve explicit axis names; spiderplot component titles label axes.

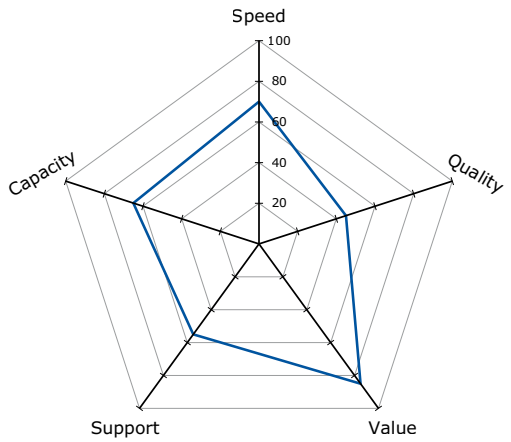
Rotate individual spoke labels

Compare with PLT-26-005 · Give each spoke a meaningful name.

Rotate the right-hand and left-hand names by different angles. This rotates the text baseline, not the spokes or polygon. The SVG terminal supports these label angles.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 2 label rotate by -30
set paxis 5 label rotate by 30
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-006



PLT-26-006 · Generated from the complete Bash block

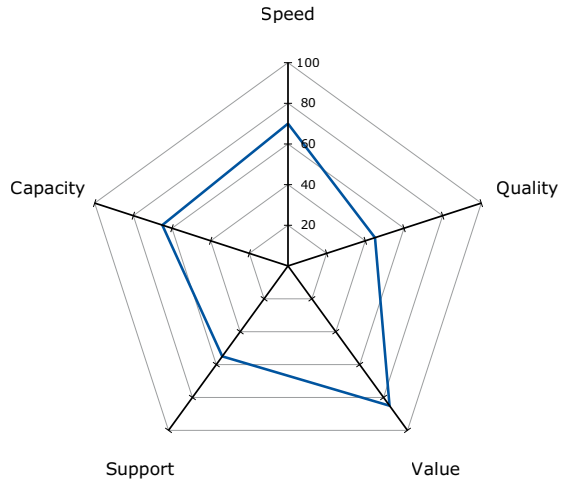
`label rotate by -30`	Turn the second spoke name clockwise by 30 degrees.
`label rotate by 30`	Turn the fifth name counterclockwise; norotate restores horizontal text.

Move spoke names radially outward

Compare with PLT-26-005 · Give each spoke a meaningful name.

The paxis label offset is a scalar radial displacement in character-sized units. It moves each name away from the centre along its spoke, rather than applying the ordinary x,y label offset.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set for [p=1:5] paxis p label offset 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```



PLT-26-007 · Generated from the complete Bash block

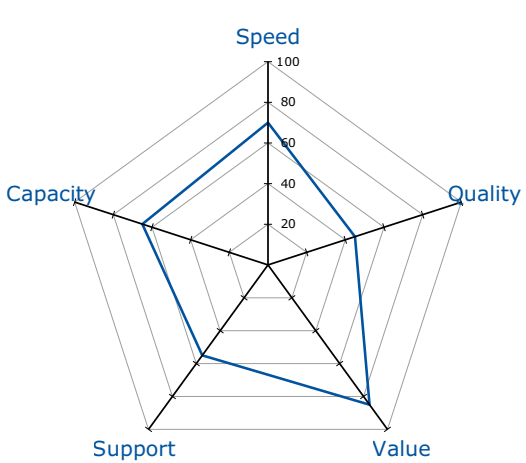
`label offset 1`	Add one character-sized step outward along the spoke; the distance depends on the terminal font metrics.
`negative offset`	Move a name inward; check clearance from the data and ticks.

Style the spoke-name font and colour

Compare with PLT-26-005 · Give each spoke a meaningful name.

Keep numeric ticks modest and make category names larger and coloured. Font metrics affect the space required around the web.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set for [p=1:5] paxis p label font ",24" textcolor rgb "#00549F"
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```



PLT-26-008 · Generated from the complete Bash block

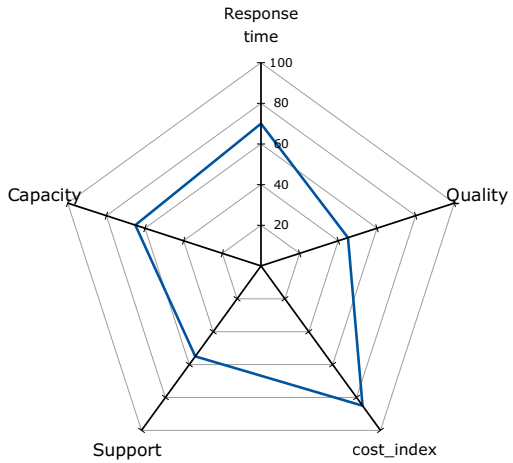
`label font ",24"`	Change spoke-name size while keeping the current font family.
`textcolor rgb`	Give names an explicit colour, independent of data lines.

Use multiline and literal spoke names

Compare with PLT-26-005 · Give each spoke a meaningful name.

A newline splits a long name. Disable enhanced text for an identifier containing an underscore so it prints literally rather than as a subscript.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 1 label "Response\ntime" font ",18" offset 1
set paxis 3 label "cost_index" noenhanced font ",18"
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```



PLT-26-009 · Generated from the complete Bash block

`\n` inside double quotes	Break the first name over two lines.
`label noenhanced`	Print the underscore literally; enhanced restores markup.

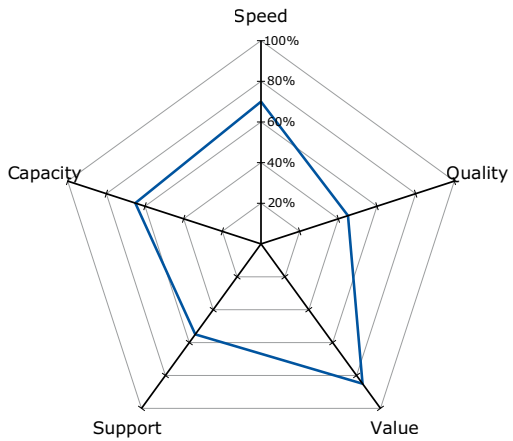
Format a spoke scale as percentages

Compare with PLT-26-005 · Give each spoke a meaningful name.

These scores already use the 0-to-100 percentage scale. Add a percent sign to their tick text. For fractional input such as 0.7, multiply by 100 in the using expression first; formatting alone does not rescale data.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 1 tics format "%.0f%%"
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-010



PLT-26-010 · Generated from the complete Bash block

`%.0f%%`	Print an integer followed by a literal percent sign.
`fractional input`	Use column(i)*100.0 with a 0-to-100 spoke range.

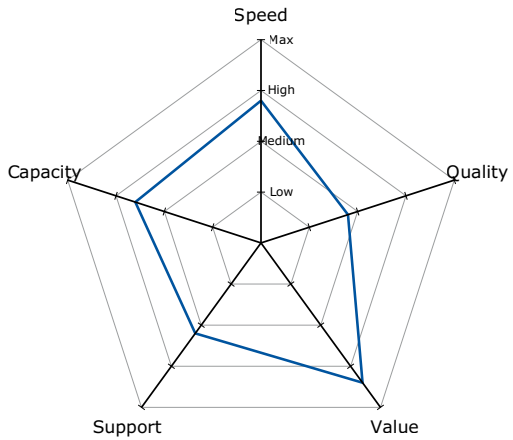
Use words for radial tick levels

Compare with PLT-26-005 · Give each spoke a meaningful name.

Replace numbers on the first spoke with named thresholds. Set the same level positions on every spoke, then hide the other text labels.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set for [p=1:5] paxis p tics ("Low" 25, "Medium" 50, "High" 75, "Max" 100)
set for [p=2:5] paxis p tics format "" ("" 25, "" 50, "" 75, "" 100)
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-011



PLT-26-011 · Generated from the complete Bash block

`label/value pairs`	Place named radial thresholds at selected numeric positions.
`explicit empty labels`	Suppress literal labels on other spokes; format alone does not replace them.

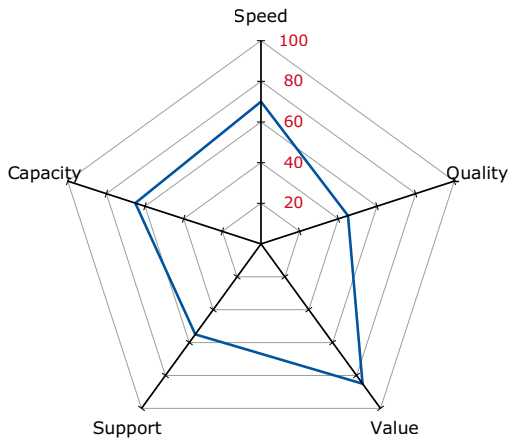
Change numeric tick text and offset

Compare with PLT-26-005 · Give each spoke a meaningful name.

Move the first spoke tick labels to the right and give them a distinct text style. In gnuplot 6.0.5, native spiderplot tick text stays horizontal even if paxis tics rotate is requested; PLT-26-013 supplies rotated labels explicitly.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 1 tics font ",18" textcolor rgb "#CC071E" offset 1,0
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-012



PLT-26-012 · Generated from the complete Bash block

`tics offset 1,0`	Move tick text one character width to the right.
`tics textcolor`	Colour the text; this does not colour the spoke line.

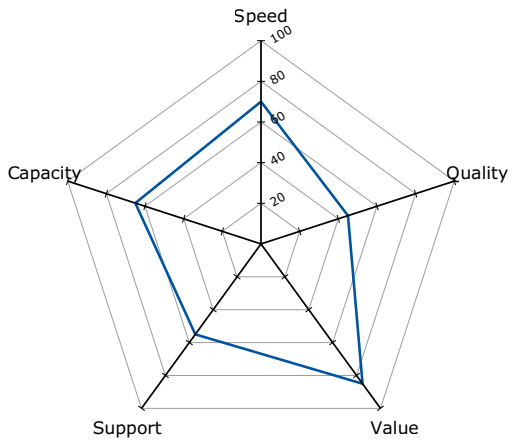
Rotate numeric tick labels explicitly

Compare with PLT-26-012 · Change numeric tick text and offset.

Native spiderplot tick rotation is ignored in gnuplot 6.0.5. Hide the first spoke numeric text and draw matching labels in normalised Cartesian coordinates instead. Keep the native tick positions so the web does not change.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 1 tics format ""
set for [v=20:100:20] label v sprintf("%d",v)     at first 0.05,v/100.0 left
rotate by 30 font ",14" front
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-013



PLT-26-013 · Generated from the complete Bash block

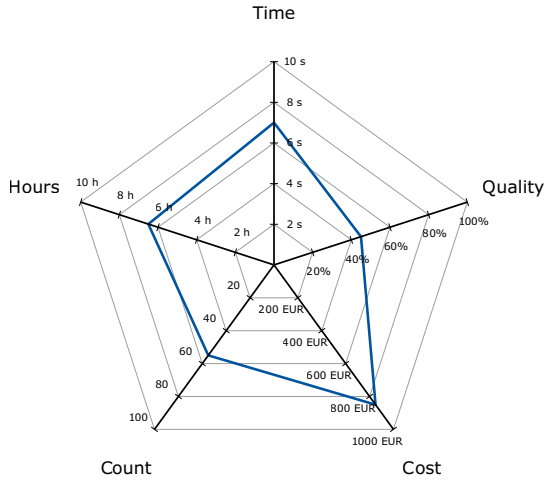
`first 0.05,v/100.0`	Place text beside the vertical spoke; its radius runs from zero to one.
`rotate by 30`	Rotate these ordinary labels counterclockwise; SVG supports the rotation.
`label tags 20..100`	Keep custom labels distinct from reserved ID label 999.

Give spokes different units and ranges

Compare with PLT-26-004 · Show numeric notation on every spoke.

Each spoke maps its own range to the same radial length. Here seconds, percentages, euros, counts and hours have separate tick units. Similar polygon radii therefore do not imply equal raw values.

```
$D << EOD
7 45 850 55 6.5
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 1 range [0:10]
set paxis 3 range [0:1000]
set paxis 5 range [0:10]
set paxis 1 label "Time"
set paxis 2 label "Quality"
set paxis 3 label "Cost"
set paxis 4 label "Count"
set paxis 5 label "Hours"
set paxis 1 tics 2 format "%g s" font ",12"
set paxis 2 tics 20 format "%.0f%%" font ",12"
set paxis 3 tics 200 format "%.0f EUR" font ",12"
set paxis 4 tics 20 format "%.0f" font ",12"
set paxis 5 tics 2 format "%g h" font ",12"
set for [p=1:5] paxis p label offset 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```



PLT-26-014 · Generated from the complete Bash block

`individual paxis ranges`	Normalise each raw variable to its own radial interval.
`tics format`	Print appropriate units; %% prints a literal percent sign.

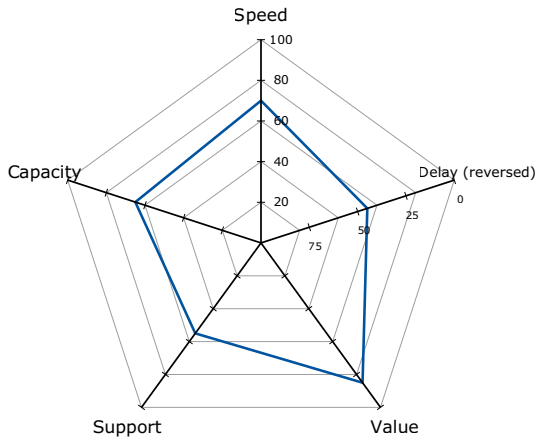
Reverse a lower-is-better spoke

Compare with PLT-26-005 · Give each spoke a meaningful name.

Reverse the second axis so a low raw value sits farther from the centre. All other axes retain their original direction. Label the reversal explicitly to keep the chart interpretable.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set paxis 2 range [100:0]
set paxis 2 label "Delay (reversed)" font ",16"
set paxis 2 tics 25 format "%.0f" font ",12"
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-015



PLT-26-015 · Generated from the complete Bash block

`range [100:0]`	Map 100 to the centre and zero to the outer end.
`value 45`	The second score lies at 55 percent of the radius.

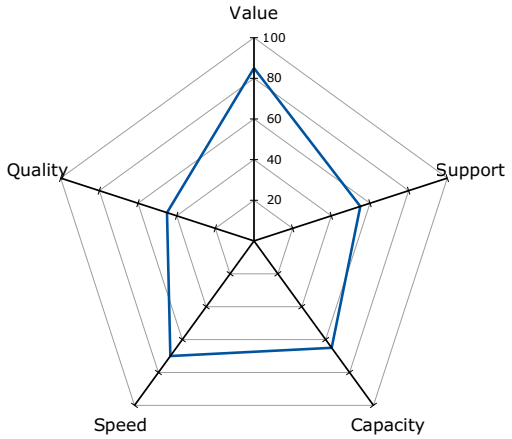
Choose which variable starts at the top

Compare with PLT-26-005 · Give each spoke a meaningful name.

The first spoke stays vertical in native spiderplot mode. Reorder the columns and names together to place a different variable at the top. This changes the polygon geometry, rather than rotating the whole chart freely.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
array Order[5] = [3,4,5,1,2]
Names = "Speed Quality Value Support Capacity"
set for [p=1:5] paxis p label word(Names,Order[p])
plot for [i=1:5] $D using (column(Order[i])) lc rgb "#00549F" notitle
```

PLT-26-016



PLT-26-016 · Generated from the complete Bash block

`Order = [3,4,5,1,2]`	Place Value at the top, then Support, Capacity, Speed and Quality.
`labels and columns`	Apply the same permutation so names match their data.

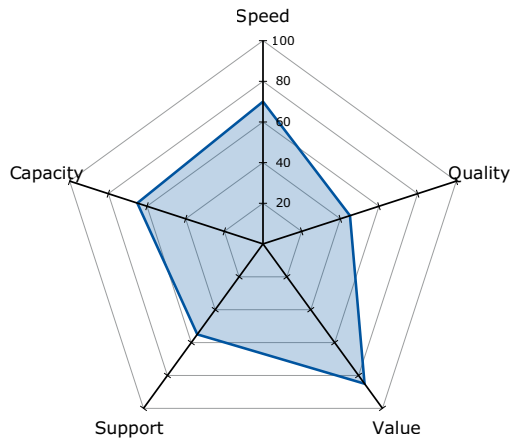
Fill the polygon transparently

Compare with PLT-26-005 · Give each spoke a meaningful name.

A translucent fill shows the polygon area while leaving the web visible. Radar area depends on axis ordering and scale choices; it is not an independent total score.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.25 border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-017



PLT-26-017 · Generated from the complete Bash block

`fs transparent solid 0.25`	Fill the polygon at 25 percent opacity.
`border lw 3`	Retain a visible polygon outline.

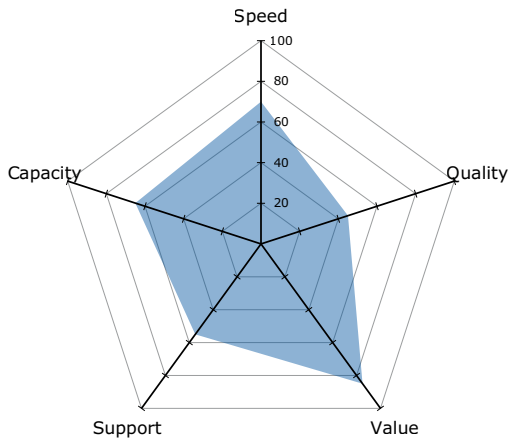
Remove the polygon border

Compare with PLT-26-017 · Fill the polygon transparently.

Use the coloured fill alone to describe the scores. The web and spoke lines have their own settings and remain visible.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.45 noborder
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-018



PLT-26-018 · Generated from the complete Bash block

`noborder`	Suppress the data polygon outline.
`solid 0.45`	Increase opacity so the borderless area remains visible.

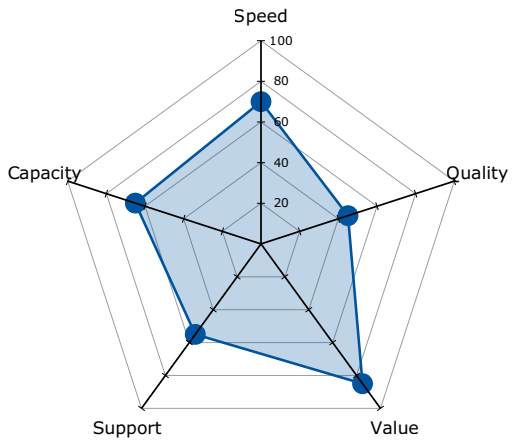
Mark every polygon vertex

Compare with PLT-26-017 · Fill the polygon transparently.

Points mark the actual score on each spoke. Set their symbol and size in the spiderplot style, alongside the outline and fill.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.25 border lw 3 pt 7 ps 1.5
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-019



PLT-26-019 · Generated from the complete Bash block

`pt 7`	Use a filled circular marker with this SVG terminal.
`ps 1.5`	Scale vertex symbols by a factor of 1.5.

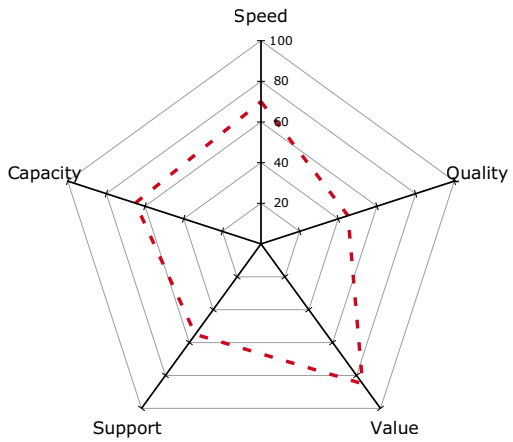
Choose a dashed polygon outline

Compare with PLT-26-005 · Give each spoke a meaningful name.

Line colour, width and dash pattern belong to the data polygon. Use an empty fill here so the outline change is easy to see.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
plot for [i=1:5] $D using i lc rgb "#CC071E" lw 4 dt 2 notitle
```

PLT-26-020



PLT-26-020 · Generated from the complete Bash block

`lc rgb; lw; dt`	Select a red, width-four, dashed data outline.
`first plot component`	Its style options establish the polygon appearance.

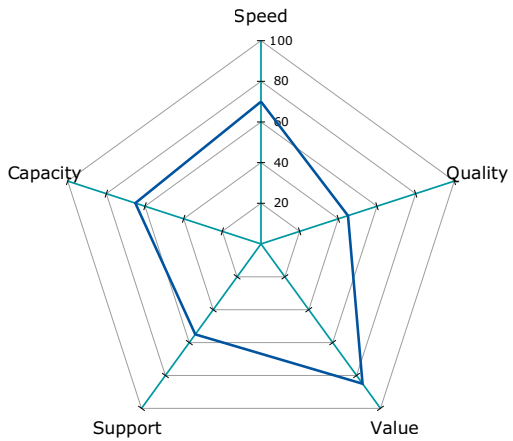
Style the radial spoke lines

Compare with PLT-26-005 · Give each spoke a meaningful name.

The parallelaxis style controls the radial axes too. Change their colour and width without changing the polygon or connecting web rings.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs empty border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set style parallelaxis back lc rgb "#0098A1" lw 2
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-021



PLT-26-021 · Generated from the complete Bash block

`style parallelaxis`	Set radial axis line properties and layer.
`back`	Keep the spokes behind the data.

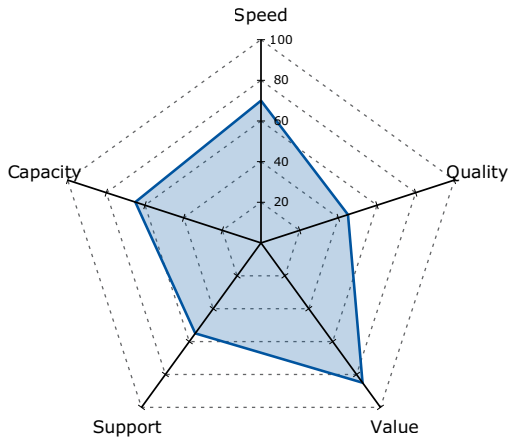
Style and layer the web grid

Compare with PLT-26-017 · Fill the polygon transparently.

Use a stronger dashed grid in front of a filled polygon. Front-layer lines remain visible above the fill; use back for a quieter background web.

```
$D << EOD
70 45 85 55 65
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.25 border lw 3
set grid spiderplot front lt 1 lc rgb "#646567" lw 1.5 dt 2
plot for [i=1:5] $D using i lc rgb "#00549F" notitle
```

PLT-26-022



PLT-26-022 · Generated from the complete Bash block

`grid lc; lw; dt`	Set web-ring colour, width and dash pattern.
`front / back`	Choose whether the web is drawn above or below the polygon.

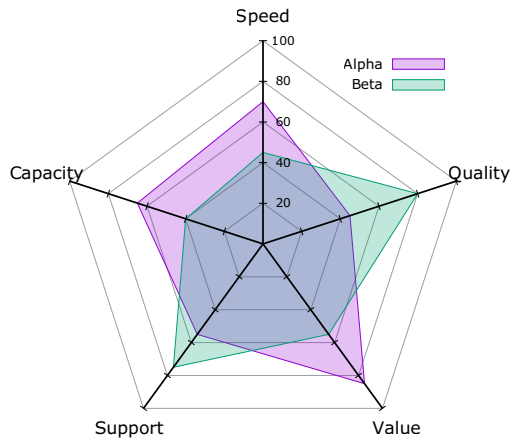
Compare several rows and name the series

Compare with PLT-26-017 · Fill the polygon transparently.

Each row becomes a separate polygon. key(1) supplies the row name for the legend; ordinary plot titles would instead name spokes. Shared ranges keep the comparison consistent.

```
$D << EOD
Alpha 70 45 85 55 65
Beta 45 80 55 75 40
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.25 border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set key top right font ",16"
plot for [i=2:6] $D using i:key(1) notitle
```

PLT-26-023



PLT-26-023 · Generated from the complete Bash block

`using i:key(1)`	Use column i for a spoke and column one for each series name.
`multiple rows`	Generate one polygon per row; default line types distinguish them.

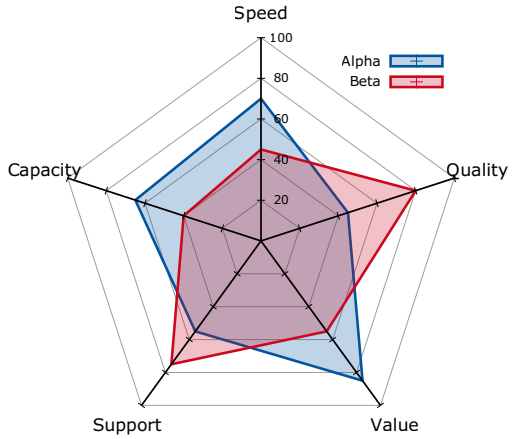
Combine separate datasets with newspiderplot

Compare with PLT-26-023 · Compare several rows and name the series.

A new polygon group needs newspiderplot between the spoke sequences. Without it, two five-column datasets would be interpreted as ten successive spokes. Use explicit legend samples for the two groups.

```
$D << EOD
70 45 85 55 65
EOD
$B << EOD
45 80 55 75 40
EOD
set spiderplot
unset key
set for [p=1:5] paxis p range [0:100]
set for [p=1:5] paxis p tics 20 font ",14"
set for [p=2:5] paxis p tics format ""
set for [p=1:5] paxis p label word("Speed Quality Value Support Capacity",p)
set style spiderplot fs transparent solid 0.25 border lw 3
set grid spiderplot back lt 1 lc rgb "#9C9E9F" lw 1
set key top right font ",16"
plot keyentry with spiderplot lc rgb "#00549F" title "Alpha",      for
[i=1:5] $D using i lc rgb "#00549F" notitle,      newspiderplot,
keyentry with spiderplot lc rgb "#CC071E" title "Beta",      for [i=1:5] $B
using i lc rgb "#CC071E" notitle
```

PLT-26-024



PLT-26-024 · Generated from the complete Bash block

`newspiderplot`	Restart spoke numbering for the next polygon group.
`keyentry`	Add a legend sample without overwriting a spoke name.

Draw a single sector

New baseline: the commands below define this example.

gnuplot 6 adds the sectors style. Begin with a quarter disk: the radius and angle are explicit.

```
$D << EOD
0 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set style fill solid 1 border
plot $D using 1:(0):2:(1) with sectors
```

PLT-27-001



PLT-27-001 · Generated from the complete Bash block

`set polar; angles degrees`	Interpret polar positions as angle and radius; angles are measured in degrees.
`theta top cw`	Put zero degrees at the top and increase clockwise.
`ratio -1`	Use equal physical scale on x and y, so circles stay circular.
`1:(0):2:(1)`	Start angle from column 1, inner radius 0, span from column 2, radial width 1. Outer radius is inner + width.
`unset raxis / tics / border`	Hide radial axis, tick marks and rectangular frame.

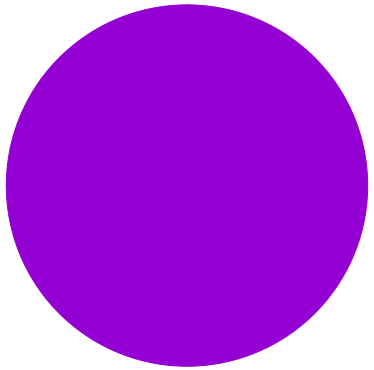
Complete a three-slice pie

Compare with PLT-27-001 · Draw a single sector.

The spans sum to 360 degrees. These example shares are 40%, 35% and 25%.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set style fill solid 1 border
plot $D using 1:(0):2:(1) with sectors
```

PLT-27-002



PLT-27-002 · Generated from the complete Bash block

`0, 144, 270`	Cumulative start angles; each slice begins where the preceding one ends.
`144, 126, 90`	Angular span = 360 * share / total. Negative values are unsuitable for pie shares.

Assign distinct slice colours

Compare with PLT-27-002 · Complete a three-slice pie.

Use the row index as the colour selector.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set style fill solid 1 border
plot $D using 1:(0):2:(1):($0+1) \
     with sectors linecolor variable
```

PLT-27-003



PLT-27-003 · Generated from the complete Bash block

`$0`	Zero-based record index; adding one selects line colours 1, 2, 3.
`linecolor variable`	Use the extra numeric field as a line-type colour index; default colours can vary by terminal.

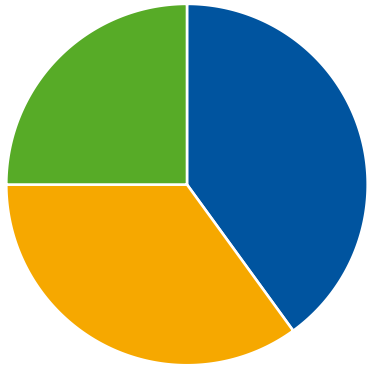
Separate slices with white borders

Compare with PLT-27-003 · Assign distinct slice colours.

Custom line colours and a white outline make the segments easier to distinguish.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
plot $D using 1:(0):2:(1):($0+1) \
     with sectors linecolor variable linewidth 3
```

PLT-27-004



PLT-27-004 · Generated from the complete Bash block

`set linetype 1/2/3`	Assign explicit RGB colours to the three indexed styles.
`border linecolor rgb "white"`	Use white borders while keeping the variable fill colour.
`linewidth 3`	Thicken the boundaries between slices.

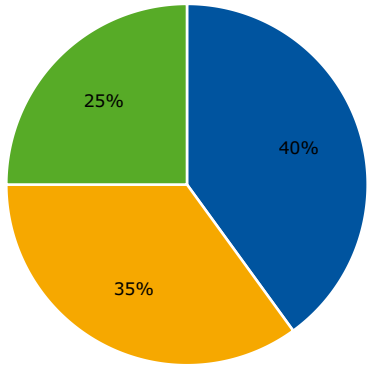
Label the percentages

Compare with PLT-27-004 · Separate slices with white borders.

Place text at each angular midpoint and at radius 0.65.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
plot $D using 1:(0):2:(1):($0+1) \
     with sectors linecolor variable linewidth 3, \
     $D using ($1+$2/2):(0.65):\
     (sprintf("%.0f%%",$2/3.6)) with labels
```

PLT-27-005



PLT-27-005 · Generated from the complete Bash block

`$1+$2/2`	Start angle plus half the span gives the slice midpoint.
`0.65`	Label radius measured from the centre.
`$2/3.6`	Convert degrees to percentage points: span / 360 * 100.

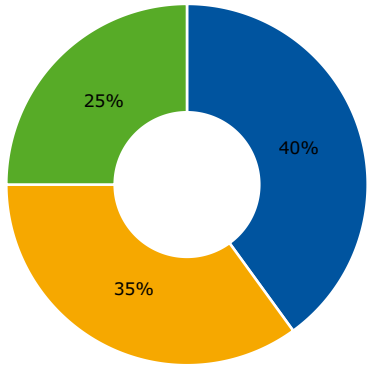
Open a donut hole

Compare with PLT-27-005 · Label the percentages.

Keep the outer radius equal to one while creating an empty centre.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
plot $D using 1:(0.4):2:(0.6):($0+1) \
     with sectors linecolor variable linewidth 3, \
     $D using ($1+$2/2):(0.65):\
     (sprintf("%.0f%%",$2/3.6)) with labels
```

PLT-27-006



PLT-27-006 · Generated from the complete Bash block

`inner radius 0.4`	Leave the central disk empty.
`radial width 0.6`	Outer radius = 0.4 + 0.6 = 1. The fourth field is a width, not an outer radius.

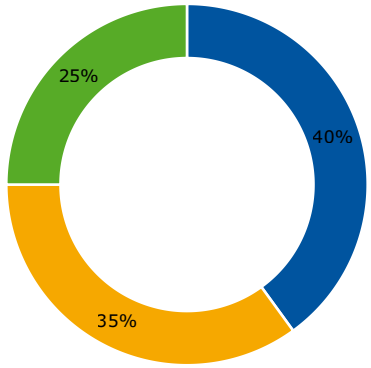
Make the donut ring thinner

Compare with PLT-27-006 · Open a donut hole.

Both radii change together so the outside size stays fixed.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
plot $D using 1:(0.7):2:(0.3):($0+1) \
     with sectors linecolor variable linewidth 3, \
     $D using ($1+$2/2):(0.85):\
     (sprintf("%.0f%%",$2/3.6)) with labels
```

PLT-27-007



PLT-27-007 · Generated from the complete Bash block

`0.7 and 0.3`	Increase the inner radius and reduce the width by the same amount.
`label radius 0.85`	Move the labels to the middle of the thinner ring.

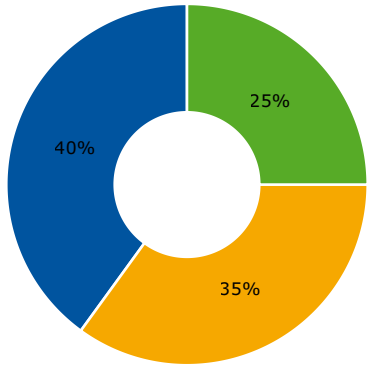
Reverse the angular direction

Compare with PLT-27-006 · Open a donut hole.

Change only the direction in which the slices are laid out.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top ccw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
plot $D using 1:(0.4):2:(0.6):($0+1) \
     with sectors linecolor variable linewidth 3, \
     $D using ($1+$2/2):(0.65):\
     (sprintf("%.0f%%",$2/3.6)) with labels
```

PLT-27-008



PLT-27-008 · Generated from the complete Bash block

`ccw`	Increase angles counterclockwise from the top; shares and colours stay unchanged.

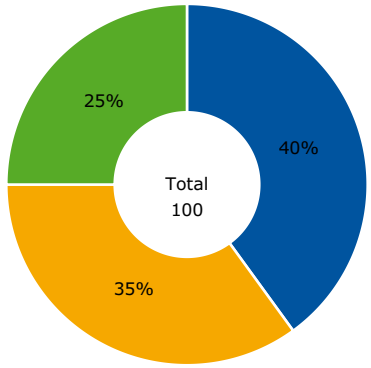
Add a central total

Compare with PLT-27-006 · Open a donut hole.

The hole can carry one short summary. Keep the surrounding labels uncluttered.

```
$D << EOD
0 144
144 126
270 90
EOD
set polar
set angles degrees
set theta top cw
set size ratio -1
set xrange [-1.3:1.3]
set yrange [-1.3:1.3]
unset border
unset raxis
unset tics
unset key
set linetype 1 linecolor rgb "#00549F"
set linetype 2 linecolor rgb "#F6A800"
set linetype 3 linecolor rgb "#57AB27"
set style fill solid 1 border linecolor rgb "white"
set label 1 "Total\n100" at 0,0 center
plot $D using 1:(0.4):2:(0.6):($0+1) \
     with sectors linecolor variable linewidth 3, \
     $D using ($1+$2/2):(0.65):\
     (sprintf("%.0f%%",$2/3.6)) with labels
```

PLT-27-009



PLT-27-009 · Generated from the complete Bash block

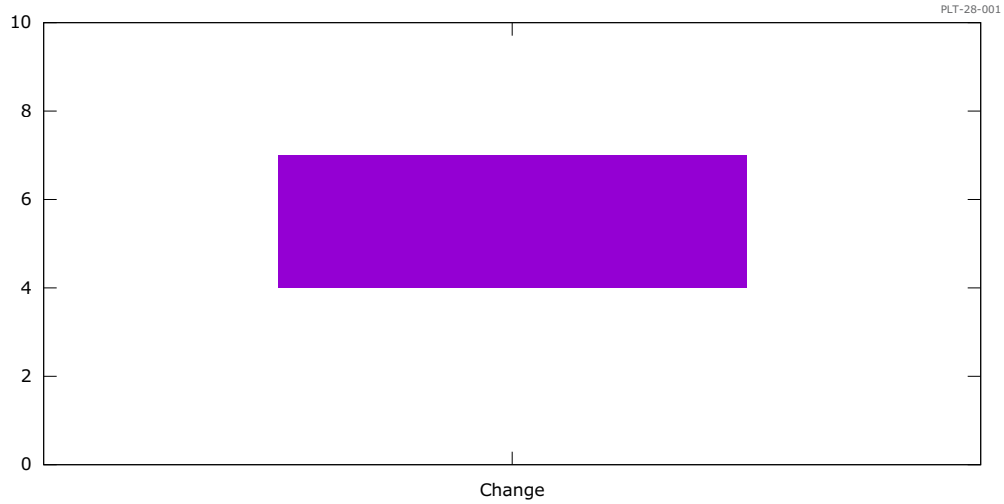
`label 1`	Assign a reusable label tag; it is separate from the reserved plot-ID tag 999.
`"Total\n100"`	A double-quoted string interprets the backslash-n as a line break.
`at 0,0 center`	Centre the text at the origin of the plot.

Start with one floating rectangle

New baseline: the commands below define this example.

A financial waterfall is a sequence of floating bars. First draw one interval without any running-total logic.

```
$D << EOD
1 4 7
EOD
set xrange [0.4:1.6]
set yrange [0:10]
set xtics ("Change" 1)
set style fill solid 1 noborder
plot $D using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3 with boxxyerrorbars notitle
```



PLT-28-001 · Generated from the complete Bash block

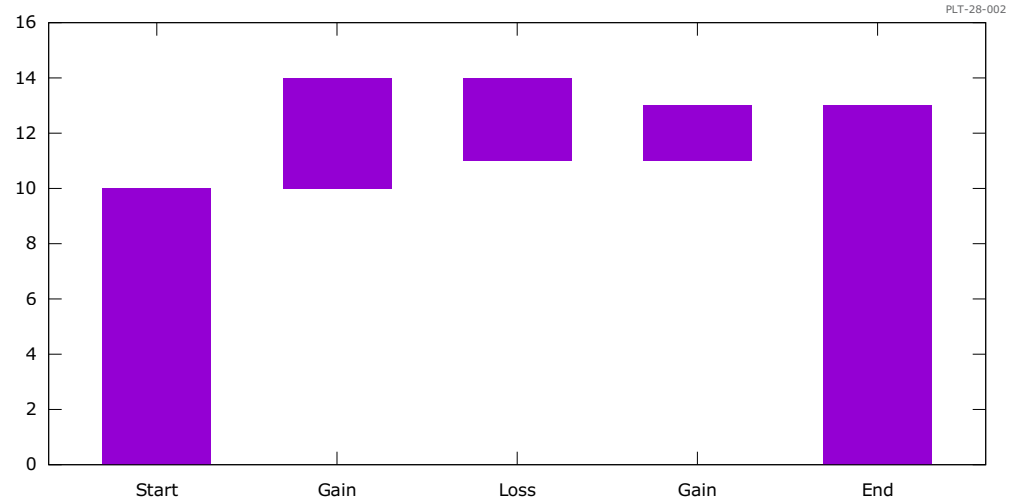
`Columns: x, low, high`	This rectangle extends from y=4 to y=7.
`($2+$3)/2`	Supply the y centre required by the six-field rectangle style.
`($1-0.3):($1+0.3):2:3`	Explicit x low, x high, y low and y high bounds; width is 0.6.

Arrange a sequence of intervals

Compare with PLT-28-001 · Start with one floating rectangle.

Start at 10, add 4, subtract 3, then add 2. The final total is 13. A decrease still uses low < high; direction is encoded separately next.

```
$D << EOD
1 0 10 Start
2 10 14 Gain
3 11 14 Loss
4 11 13 Gain
5 0 13 End
EOD
set xrange [0.4:5.6]
set yrange [0:16]
set style fill solid 1 noborder
plot $D using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3:xticlabels(4) with boxxyerrorbars notitle
```



PLT-28-002 · Generated from the complete Bash block

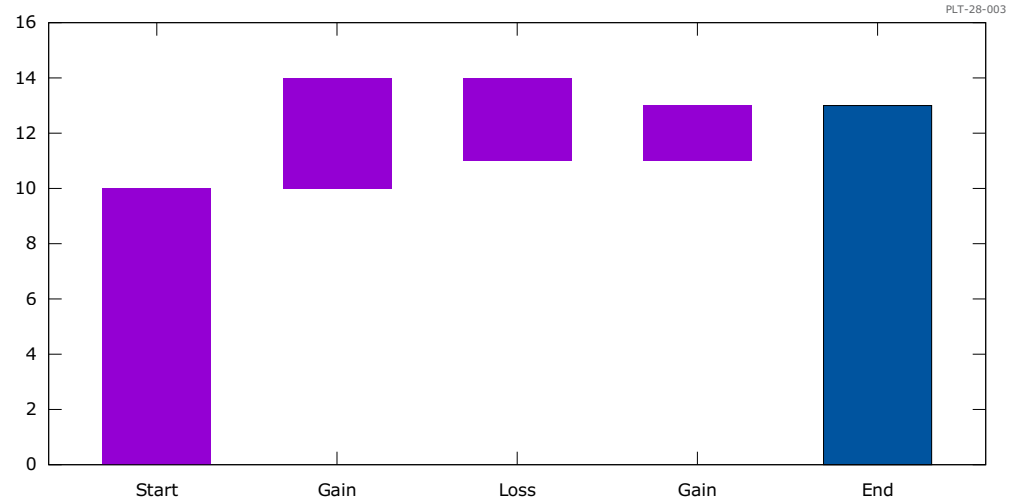
`Start and End`	Total bars begin at zero. Intermediate bars float between the old and new total.
`Loss: 11 to 14`	The decrease starts at 14 and ends at 11; the rectangle bounds must still be ordered.
`xticlabels(4)`	Use the fourth column for category labels.

Derive intervals from signed changes

Compare with PLT-28-002 · Arrange a sequence of intervals.

A preprocessing pass converts deltas into rectangle bounds. The first row starts at zero; the last row is an explicit total.

```
$D << EOD
Start 10
Gain 4
Loss -3
Gain 2
EOD
running = 0
set table $B
plot $D using ($0+1):\
 (old=running,running=running+$2,(old<running ? old : running)):\
 ((old>running ? old : running)) with table
unset table
set xrange [0.4:5.6]
set yrange [0:16]
set xtics ("Start" 1,"Gain" 2,"Loss" 3,"Gain" 4,"End" 5)
set style fill solid 1 noborder
set object 1 rect from 4.7,0 to 5.3,running \
    fillcolor rgb "#00549F"
plot $B using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3 with boxxyerrorbars notitle
```



PLT-28-003 · Generated from the complete Bash block

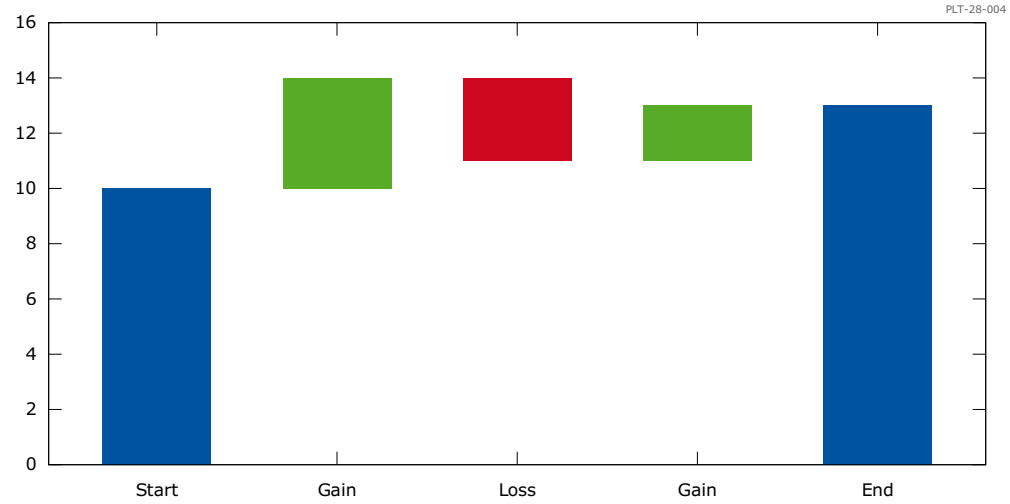
`set table $B / with table`	Write evaluated columns to a datablock instead of drawing a plot.
`old=running, running=...`	Comma expressions evaluate in order; save the old total before adding the next delta.
`condition ? a : b`	Select a when the comparison is true, otherwise b; this keeps lower and upper bounds ordered.
`set object 1 rect`	Draw the final total as a rectangle from zero to running=13.

Colour totals, gains and losses

Compare with PLT-28-002 · Arrange a sequence of intervals.

Colour encodes the role of a bar, independently of its vertical bounds.

```
$D << EOD
1 0 10 Start 0x00549F
2 10 14 Gain 0x57AB27
3 11 14 Loss 0xCC071E
4 11 13 Gain 0x57AB27
5 0 13 End 0x00549F
EOD
set xrange [0.4:5.6]
set yrange [0:16]
set style fill solid 1 noborder
plot $D using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3:5:xticlabels(4) with boxxyerrorbars linecolor rgb variable
notitle
```



PLT-28-004 · Generated from the complete Bash block

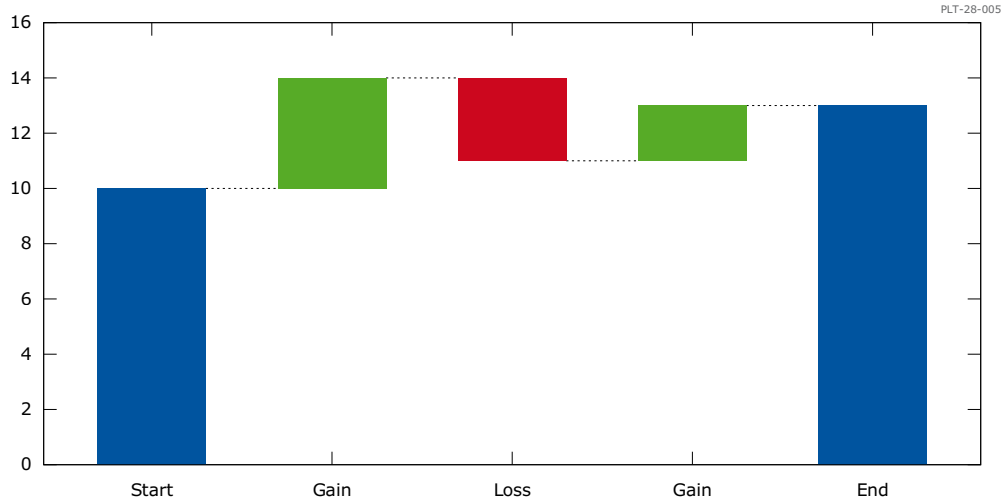
`Seventh numeric field`	After the six rectangle fields, column 5 supplies an RGB integer.
`Blue / green / red`	Totals / increases / decreases. Label the sign as well when colour alone would be ambiguous.

Connect the running levels

Compare with PLT-28-004 · Colour totals, gains and losses.

Short horizontal segments show how the end of one step becomes the start of the next.

```
$D << EOD
1 0 10 Start 0x00549F
2 10 14 Gain 0x57AB27
3 11 14 Loss 0xCC071E
4 11 13 Gain 0x57AB27
5 0 13 End 0x00549F
EOD
set xrange [0.4:5.6]
set yrange [0:16]
set style fill solid 1 noborder
set arrow 1 from 1.3,10 to 1.7,10 nohead dt 2
set arrow 2 from 2.3,14 to 2.7,14 nohead dt 2
set arrow 3 from 3.3,11 to 3.7,11 nohead dt 2
set arrow 4 from 4.3,13 to 4.7,13 nohead dt 2
plot $D using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3:5:xticlabels(4) with boxxyerrorbars linecolor rgb variable
notitle
```



PLT-28-005 · Generated from the complete Bash block

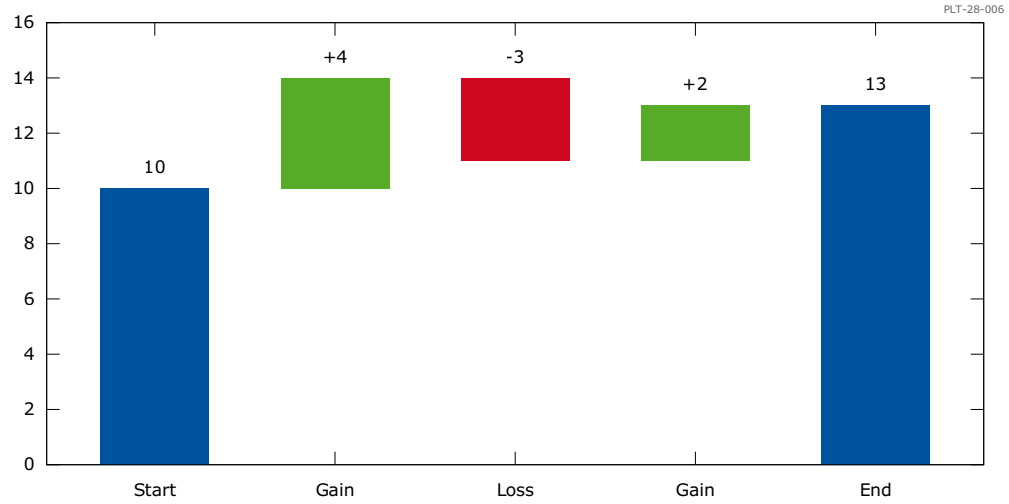
`from x1,y to x2,y`	Draw from the right edge of one bar to the left edge of the next at the running total.
`nohead dt 2`	Use a dashed line without an arrowhead.

Print the signed changes

Compare with PLT-28-004 · Colour totals, gains and losses.

Labels make increases and decreases explicit. The total bars show their absolute values.

```
$D << EOD
1 0 10 Start 0x00549F "10"
2 10 14 Gain 0x57AB27 "+4"
3 11 14 Loss 0xCC071E "-3"
4 11 13 Gain 0x57AB27 "+2"
5 0 13 End 0x00549F "13"
EOD
set xrange [0.4:5.6]
set yrange [0:16]
set style fill solid 1 noborder
plot $D using 1:(($2+$3)/2):($1-0.3):\
     ($1+0.3):2:3:5:xticlabels(4) with boxxyerrorbars linecolor rgb variable notitle, \
     $D using 1:3:6 with labels offset 0,0.8 notitle
```



PLT-28-006 · Generated from the complete Bash block

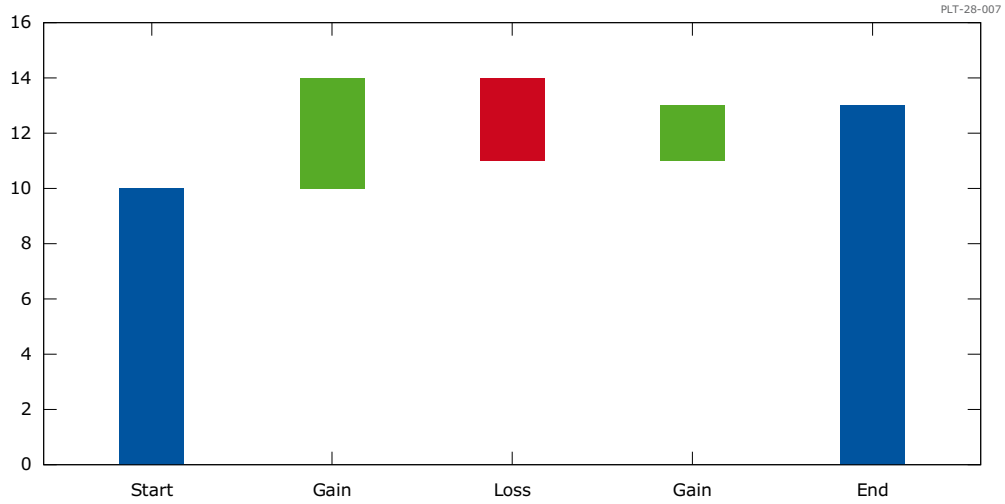
`Column 6`	Text labels are stored explicitly, including plus and minus signs.
`using 1:3:6`	Position each label at the bar's x and upper y bound.

Narrow the waterfall bars

Compare with PLT-28-004 · Colour totals, gains and losses.

The interval heights remain unchanged; only horizontal whitespace changes.

```
$D << EOD
1 0 10 Start 0x00549F
2 10 14 Gain 0x57AB27
3 11 14 Loss 0xCC071E
4 11 13 Gain 0x57AB27
5 0 13 End 0x00549F
EOD
set xrange [0.4:5.6]
set yrange [0:16]
set style fill solid 1 noborder
plot $D using 1:(($2+$3)/2):($1-0.18):\
     ($1+0.18):2:3:5:xticlabels(4) with boxxyerrorbars linecolor rgb
variable notitle
```



PLT-28-007 · Generated from the complete Bash block

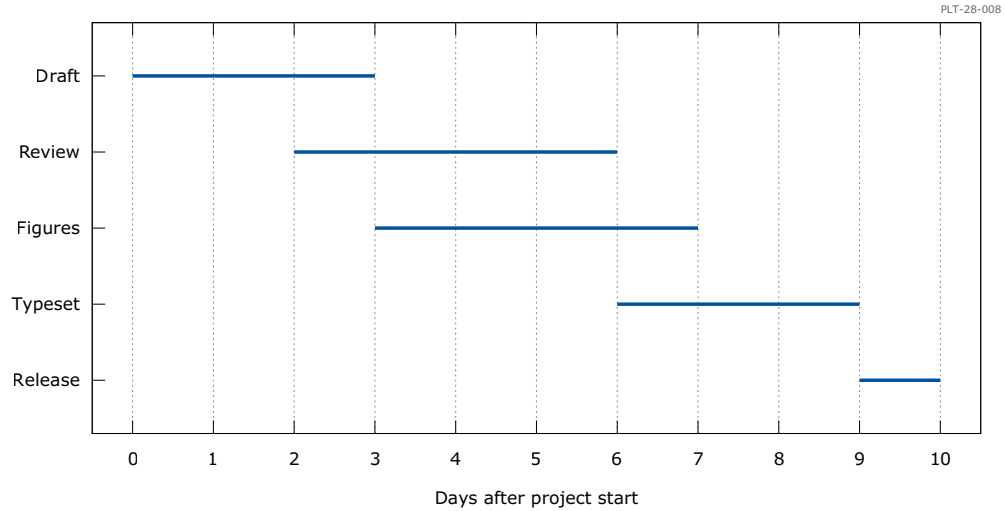
`0.18` Half-width; the complete bar is 0.36 x units wide.

Draw a small schedule from task intervals

New baseline: the commands below define this example.

A fictional publication has five tasks. Rows supply position, name, start day, end day and completed fraction. Draw a headless vector from start to end; reverse y so the first task is at the top.

```
$Tasks << EOD
1 "Draft" 0 3 1.0
2 "Review" 2 6 0.6
3 "Figures" 3 7 0.4
4 "Typeset" 6 9 0.2
5 "Release" 9 10 0.0
EOD
set xrange [-0.5:10.5]
set yrange [5.7:0.3]
set xlabel "Days after project start"
set ytics nomirror
set xtics 1
set grid xtics
unset key
plot $Tasks using 3:1:($4-$3):(0):yticlabels(2) \
    with vectors nohead lw 4 lc rgb "#00549F"
```



PLT-28-008 · Generated from the complete Bash block

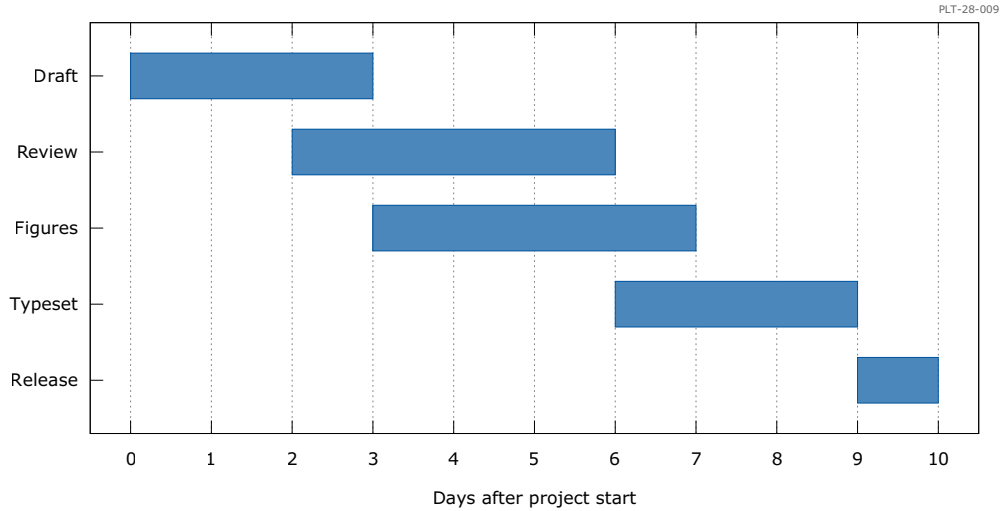
`using 3:1:($4-$3):(0)`	Start at the task's start and extend by its duration.
`yticlabels(2)`	Use task names for row labels.
`yrange [5.7:0.3]`	Order tasks from top to bottom.

Replace schedule lines with interval boxes

Compare with PLT-28-008 · Draw a small schedule from task intervals.

Turn intervals into bars with explicit start, end and height limits. The six-column boxxyerrorbars form accepts absolute bounds, which draw horizontal task rectangles.

```
set xrange [-0.5:10.5]
set yrange [5.7:0.3]
set xlabel "Days after project start"
set ytics nomirror
set xtics 1
set grid xtics
unset key
set style fill solid 0.7 border
plot $Tasks using (($3+$4)/2):1:3:4:($1-0.3):($1+0.3):yticlabels(2) \
    with boxxyerrorbars lc rgb "#00549F"
```



PLT-28-009 · Generated from the complete Bash block

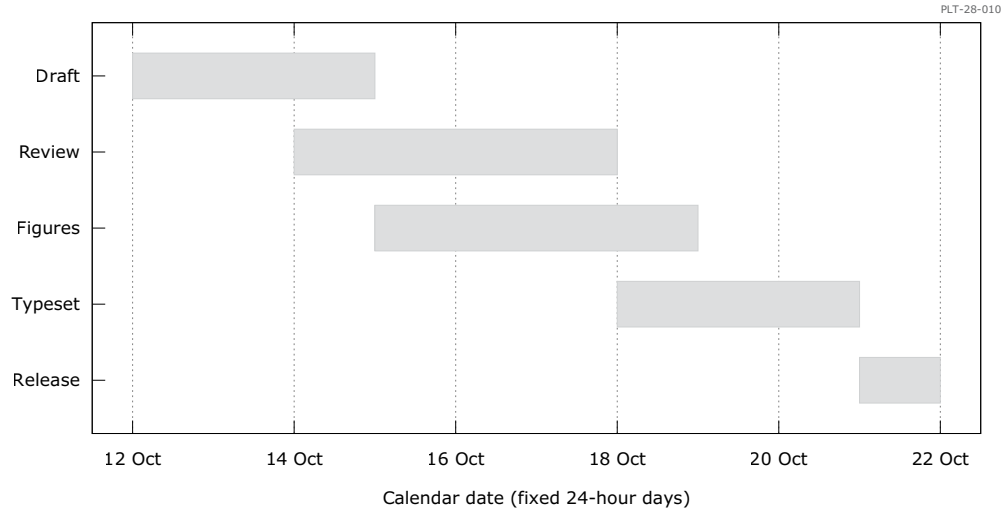
`using centre:y:xlow:xhigh:ylow:yhigh`	Supply the centre and four absolute rectangle bounds.
`$1-0.3; $1+0.3`	Set task height without changing duration.

Show calendar dates on the schedule

Compare with PLT-28-009 · Replace schedule lines with interval boxes.

Convert day offsets to timestamps from a fixed start. Convert the centre and both time bounds consistently. The example uses fixed 24-hour intervals; it does not model daylight-saving transitions or working-day calendars.

```
start=strptime("%Y-%m-%d","2026-10-12")
day=24*60*60
T(d)=start+d*day
set xrange [T(-0.5):T(10.5)]
set yrange [5.7:0.3]
set xdata time
set format x "%d %b"
set xtics day*2
set xlabel "Calendar date (fixed 24-hour days)"
set ytics nomirror
set grid xtics
set style fill solid 0.7 border
unset key
plot $Tasks using (T(($3+$4)/2)):1:(T($3)):(T($4)):($1-0.3):
($1+0.3):yticlabels(2) \
    with boxxyerrorbars lc rgb "#CFD1D2"
```



PLT-28-010 · Generated from the complete Bash block

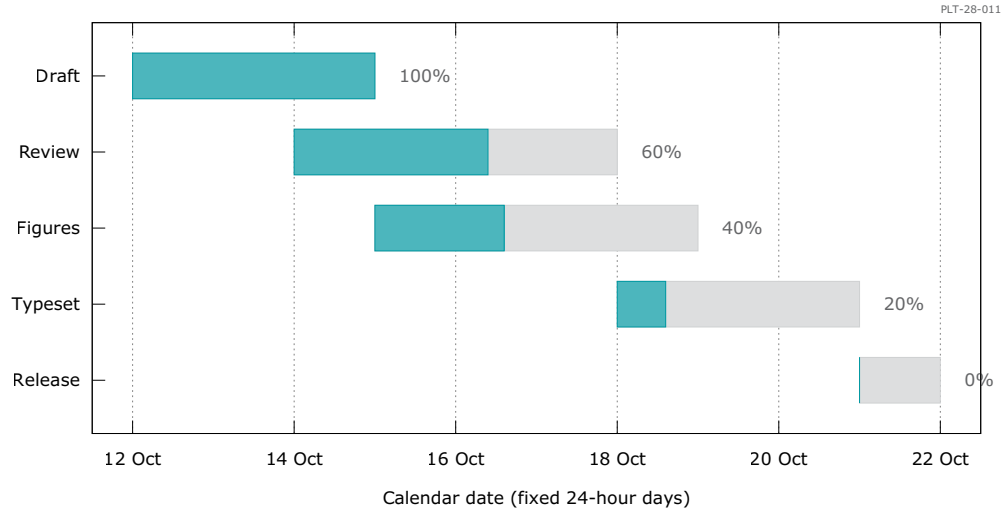
`strptime; T(d)`	Parse a start date and convert all day coordinates.
`xdata time; format x`	Render timestamps as calendar labels.
`xtics day*2`	Choose a tick interval in seconds.

Overlay the completed fraction of each task

Compare with PLT-28-010 · Show calendar dates on the schedule.

Draw planned intervals in grey and completed fractions in RWTH turquoise. A 60% task fills 60% of its own duration. The release task retains a grey bar and zero-percent label.

```
start=strptime("%Y-%m-%d","2026-10-12")
day=24*60*60
T(d)=start+d*day
set xrange [T(-0.5):T(10.5)]
set yrange [5.7:0.3]
set xdata time
set format x "%d %b"
set xtics day*2
set xlabel "Calendar date (fixed 24-hour days)"
set ytics nomirror
set grid xtics
set style fill solid 0.7 border
unset key
plot $Tasks using (T(($3+$4)/2)):1:(T($3)):(T($4)):($1-0.3):
($1+0.3):yticlabels(2) \
    with boxxyerrorbars lc rgb "#CFD1D2", \
    $Tasks using (T($3+($4-$3)*$5/2)):1:(T($3)):(T($3+($4-$3)*$5)):($1-0.3):
($1+0.3) \
    with boxxyerrorbars lc rgb "#0098A1", \
    $Tasks using (T($4)):1:(sprintf("%.0f%%",100*$5)) \
    with labels offset 2,0 left tc rgb "#646567"
```



PLT-28-011 · Generated from the complete Bash block

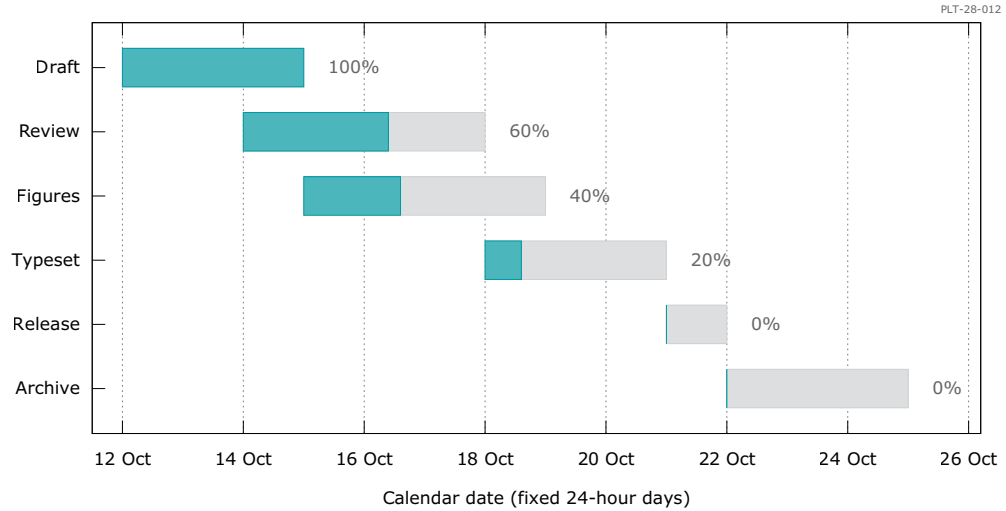
`$3+($4-$3)*$5`	Compute each completed endpoint.
`second box clause`	Overlay progress with the same row height.
`sprintf("%.0f%%",...)`	Format a literal percent sign with the computed value.

Adapt the schedule range when a task is added

Compare with PLT-28-011 · Overlay the completed fraction of each task.

Add an archive task after release, then compute earliest start, latest finish and row count. Both axis ranges follow the updated data. Reserve an extra day for percentage labels.

```
stats $Tasks using 3 name "Begin" nooutput
stats $Tasks using 4 name "End" nooutput
start=strptime("%Y-%m-%d","2026-10-12")
day=24*60*60
T(d)=start+d*day
set xrange [T(Begin_min-0.5):T(End_max+1.2)]
set yrange [Begin_records+0.7:0.3]
set xdata time
set format x "%d %b"
set xtics day*2
set xlabel "Calendar date (fixed 24-hour days)"
set ytics nomirror
set grid xtics
set style fill solid 0.7 border
unset key
plot $Tasks using (T(($3+$4)/2)):1:(T($3)):(T($4)):($1-0.3):
($1+0.3):yticlabels(2) \
    with boxxyerrorbars lc rgb "#CFD1D2", \
    $Tasks using (T($3+($4-$3)*$5/2)):1:(T($3)):(T($3+($4-$3)*$5)):($1-0.3):
($1+0.3) \
    with boxxyerrorbars lc rgb "#0098A1", \
    $Tasks using (T($4)):1:(sprintf("%.0f%%",100*$5)) \
    with labels offset 2,0 left tc rgb "#646567"
print sprintf("schedule: start=%.0f finish=%.0f
tasks=%d",Begin_min,End_max,Begin_records)
```



PLT-28-012 · Generated from the complete Bash block

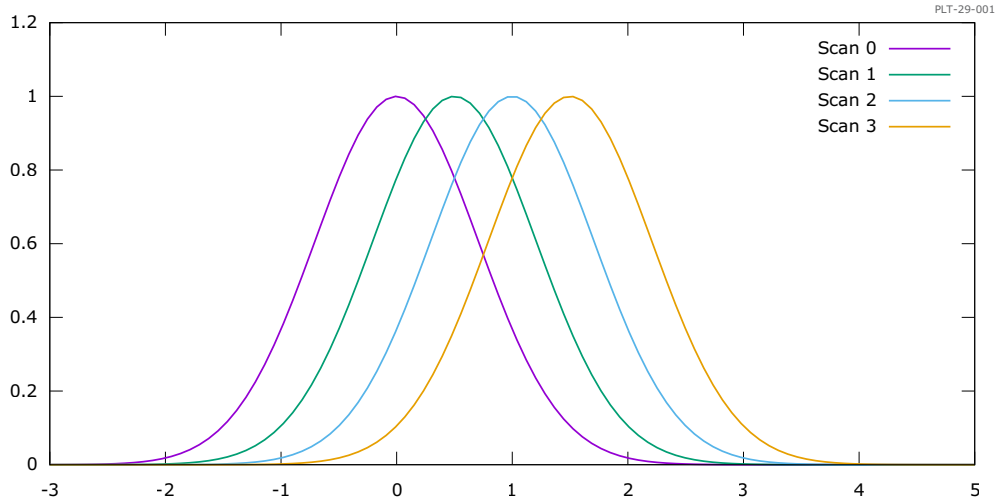
`Begin_min; End_max`	Detect earliest start and latest finish.
`Begin_records`	Count tasks with consecutive row positions starting at one.
`extra day margin`	Reserve room for completion labels.

Draw a family of related curves

New baseline: the commands below define this example.

The word waterfall also describes a sequence of signals. Begin with a loop over four shifted peaks.

```
f(x,k) = exp(-(x-0.5*k)**2)
set xrange [-3:5]
set yrange [0:1.2]
plot for [k=0:3] f(x,k) with lines linewidth 2 \
     title sprintf("Scan %d",k)
```



PLT-29-001 · Generated from the complete Bash block

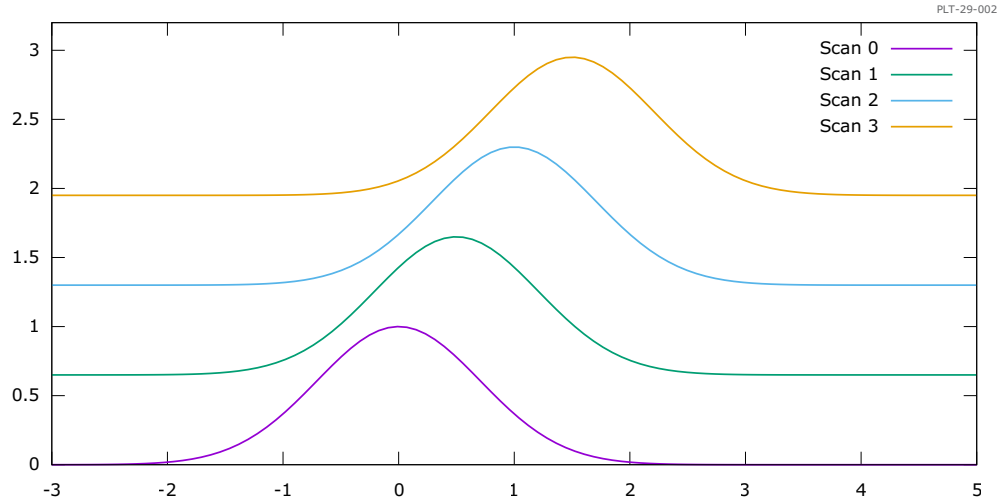
`f(x,k)`	A two-argument function; k shifts the peak by 0.5*k.
`for [k=0:3]`	Repeat the plot element for integers 0, 1, 2 and 3.
`exp / **2`	Natural exponential and exponentiation; the peak shape is a Gaussian-like function.

Separate curves with a vertical offset

Compare with PLT-29-001 · Draw a family of related curves.

Offsets improve visibility, but the vertical coordinates now include an artificial display shift.

```
f(x,k) = exp(-(x-0.5*k)**2)
set xrange [-3:5]
set yrange [0:3.2]
plot for [k=0:3] f(x,k)+0.65*k with lines linewidth 2 \
     title sprintf("Scan %d",k)
```



PLT-29-002 · Generated from the complete Bash block

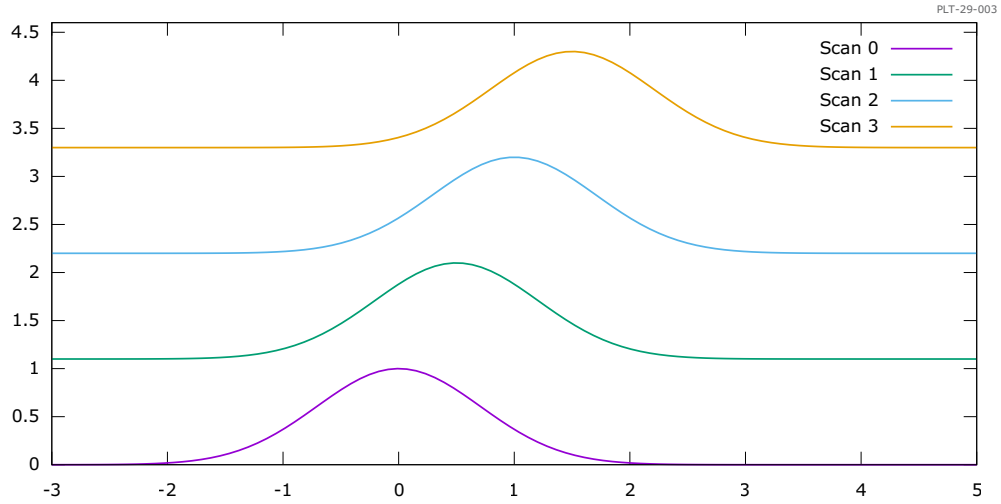
`+0.65*k` Lift each successive curve by 0.65 y units. This does not change the signal itself.

Increase the separation

Compare with PLT-29-002 · Separate curves with a vertical offset.

A single spacing constant controls the distance between baselines.

```
f(x,k) = exp(-(x-0.5*k)**2)
set xrange [-3:5]
set yrange [0:4.6]
plot for [k=0:3] f(x,k)+1.1*k with lines linewidth 2 \
     title sprintf("Scan %d",k)
```



PLT-29-003 · Generated from the complete Bash block

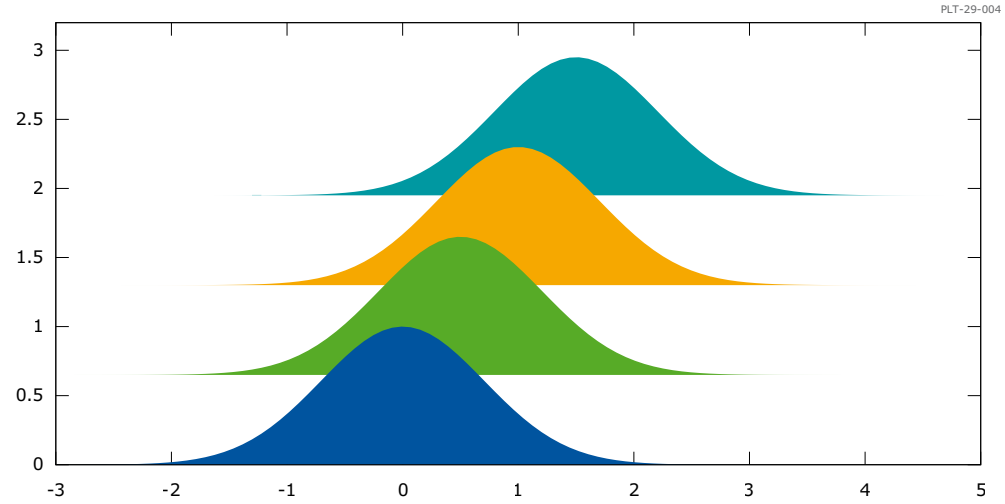
`1.1*k` The gap between neighbouring baselines becomes 1.1 units.

Fill offset curves back to front

Compare with PLT-29-002 · Separate curves with a vertical offset.

Filled curves create overlapping ridges. Draw the highest baseline first so later curves cover it cleanly. Each ridge selects a colour from the explicit RWTH list.

```
f(x,k) = exp(-(x-0.5*k)**2)
COLOURS = "#00549F #57AB27 #F6A800 #0098A1"
set xrange [-3:5]
set yrange [0:3.2]
set style fill solid 1 border
unset key
plot for [k=3:0:-1] '+' \
     using 1:(f($1,k)+0.65*k):(0.65*k) \
     with filledcurves linecolor rgb word(COLOURS,k+1)
```



PLT-29-004 · Generated from the complete Bash block

`for [k=3:0:-1]`	Visit k=3,2,1,0; the third loop argument is the negative step.
`'+'`	Generate sampled x values over the current x range.
`using x:top:bottom`	Fill between the signal plus offset and its own horizontal baseline.
`word(COLOURS,k+1)`	Select one explicit RWTH colour per ridge; word indices start at one.
`linecolor rgb`	Use the selected hexadecimal RGB string.

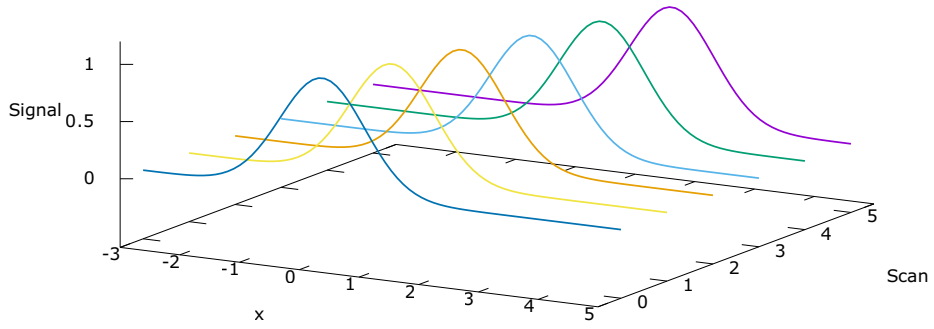
Move the signal waterfall into 3D

New baseline: the commands below define this example.

Generate six sampled curves into a datablock, then draw them as slices at different y positions.

```
set xrange [-3:5]
set samples 80
set table $S
do for [k=0:5] {
    plot exp(-(x-0.4*k)**2)
}
unset table
set yrange [-0.5:5.5]
set zrange [0:1.2]
set ztics 0.5
set view 60,30
set xlabel "x"
set ylabel "Scan"
set zlabel "Signal"
unset key
splot for [k=5:0:-1] $S index k using 1:(k):2 \
      with lines linewidth 2
```

PLT-29-005



PLT-29-005 · Generated from the complete Bash block

`set samples 80`	Generate 80 x samples per curve during the table pass.
`do for [...] { ... }`	Execute a block of commands for each integer k.
`index k`	Select one table block; indices begin at zero.
`using 1:(k):2`	Map stored x to x, scan index to y, and stored signal to z.
`set view 60,30`	Rotate 60 degrees about x and 30 degrees about z.

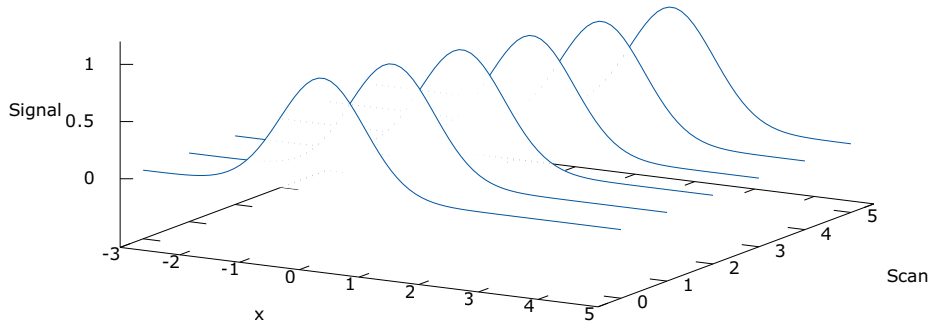
Fill the 3D waterfall slices

Compare with PLT-29-005 · Move the signal waterfall into 3D.

Opaque white slice faces emphasize the layered shape. This splot filledcurves style requires gnuplot 6.0.2 or later.

```
set xrange [-3:5]
set samples 80
set table $S
do for [k=0:5] {
    plot exp(-(x-0.4*k)**2)
}
unset table
set yrange [-0.5:5.5]
set zrange [0:1.2]
set ztics 0.5
set view 60,30
set xlabel "x"
set ylabel "Scan"
set zlabel "Signal"
unset key
set style fill solid 1
splot for [k=5:0:-1] $S index k using 1:(k):2 \
      with filledcurves linecolor rgb "#00549F" \
      fillcolor rgb "white"
```

PLT-29-006



PLT-29-006 · Generated from the complete Bash block

`solid 1; with filledcurves`	Use an opaque fill for each x-z slice down to its baseline in the y=constant plane.
`k=5:0:-1`	For this view, draw the distant scan first; reverse order when the camera moves to the opposite side.

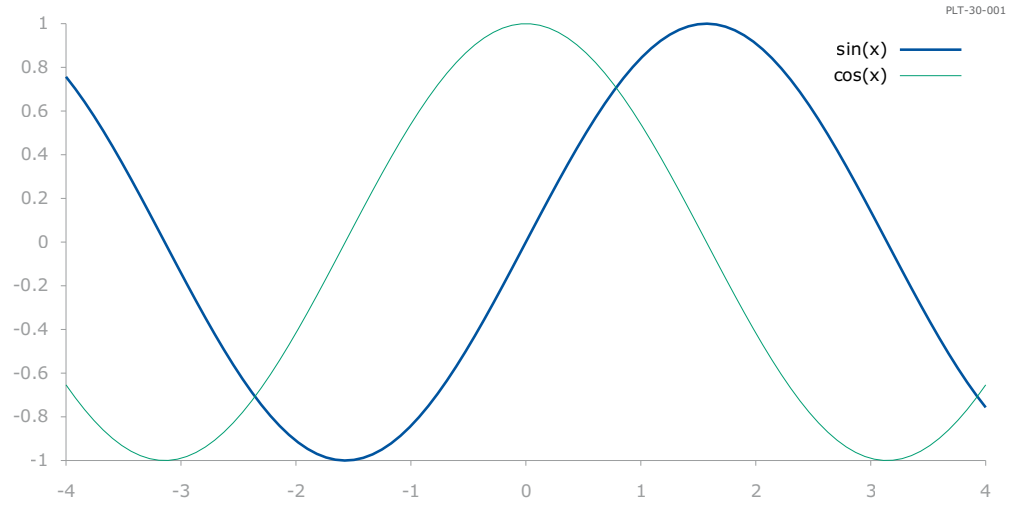
Define a reusable style file

New baseline: the commands below define this example.

Define all fourteen RWTH colour variables first, then use those names in the style commands. RWTH_BLUE selects the curve colour; RWTH_GRAY_50 selects the border colour. Save the commands through unset grid as config/my-style.gp; the complete file and download follow this example. Keep terminal, output, ranges, data and plot commands in the individual plot. Each definition is visible here and takes effect before its first use.

```
RWTH_BLUE       = '#00549F'
RWTH_BLUE_75    = '#407FB7'
RWTH_BLUE_50    = '#8EBAE5'
RWTH_BLUE_25    = '#C7DDF2'
RWTH_BLUE_10    = '#E8F1FA'
RWTH_GRAY_75    = '#646567'
RWTH_GRAY_50    = '#9C9E9F'
RWTH_GRAY_25    = '#CFD1D2'
RWTH_GRAY_10    = '#ECEDED'
RWTH_ORANGE     = '#F6A800'
RWTH_GREEN      = '#57AB27'
RWTH_RED        = '#CC071E'
RWTH_TURQUOISE  = '#0098A1'
RWTH_LILAC      = '#7A6FAC'

set linetype 1 lc rgb RWTH_BLUE lw 3
set border 3 back lw 0.6 lc rgb RWTH_GRAY_50
set tics nomirror out scale 0.4
unset grid
# End of config/my-style.gp
set xrange [-4:4]
plot sin(x) title 'sin(x)', cos(x) title 'cos(x)'
```



PLT-30-001 · Generated from the complete Bash block

`config/my-style.gp`	Create this text file in the project; copy the commands through unset grid, omitting the final comment.
`RWTH_BLUE = '#00549F'`	Define RWTH blue as a named hexadecimal RGB string before using it.
`RWTH_GRAY_50 = '#9C9E9F'`	Name the grey used by the border; all other palette colours are defined above too.
`lc rgb RWTH_BLUE`	Use the previously defined colour for the first series, with linewidth 3.
`lc rgb RWTH_GRAY_50`	Use the previously defined grey for the left and bottom borders.
`unset grid`	Make grid lines an explicit choice in each plot.

config/my-style.gp

Save this file at `config/my-style.gp` in your working directory. Download my-style.gp

```
# Minimal reusable style from chapter 30. No terminal, output or data commands.
# Define the complete RWTH palette before using any colour.
RWTH_BLUE       = '#00549F'
RWTH_BLUE_75    = '#407FB7'
RWTH_BLUE_50    = '#8EBAE5'
RWTH_BLUE_25    = '#C7DDF2'
RWTH_BLUE_10    = '#E8F1FA'
RWTH_GRAY_75    = '#646567'
RWTH_GRAY_50    = '#9C9E9F'
RWTH_GRAY_25    = '#CFD1D2'
RWTH_GRAY_10    = '#ECEDED'
RWTH_ORANGE     = '#F6A800'
RWTH_GREEN      = '#57AB27'
RWTH_RED        = '#CC071E'
RWTH_TURQUOISE  = '#0098A1'
RWTH_LILAC      = '#7A6FAC'

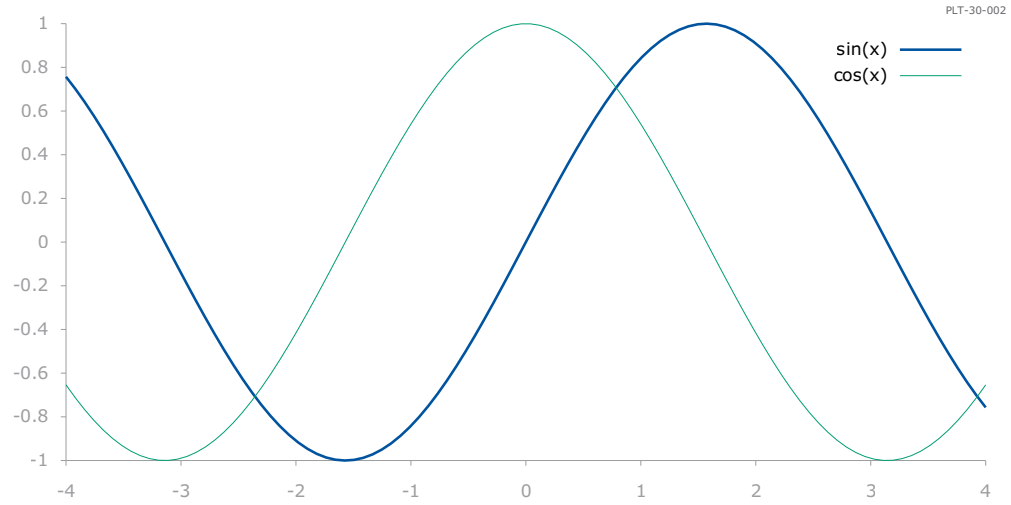
# Plot appearance
set linetype 1 lc rgb RWTH_BLUE lw 3
set border 3 back lw 0.6 lc rgb RWTH_GRAY_50
set tics nomirror out scale 0.4
unset grid
```

Load a project or personal style

Compare with PLT-30-001 · Define a reusable style file.

Replace the inline settings with load. Save the downloaded my-style.gp in a config folder in your working directory. Run from that directory: paths are relative to the process working directory, including paths inside a loaded file. This refactoring keeps the same picture. For a personal copy, create ~/.config/gnuplot, copy config/my-style.gp there and instead use load '~/.config/gnuplot/my-style.gp'. Each new gnuplot process needs its own load; -d skips startup files but still executes this explicit command.

```
load 'config/my-style.gp'
set xrange [-4:4]
plot sin(x) title 'sin(x)', cos(x) title 'cos(x)'
```



PLT-30-002 · Generated from the complete Bash block

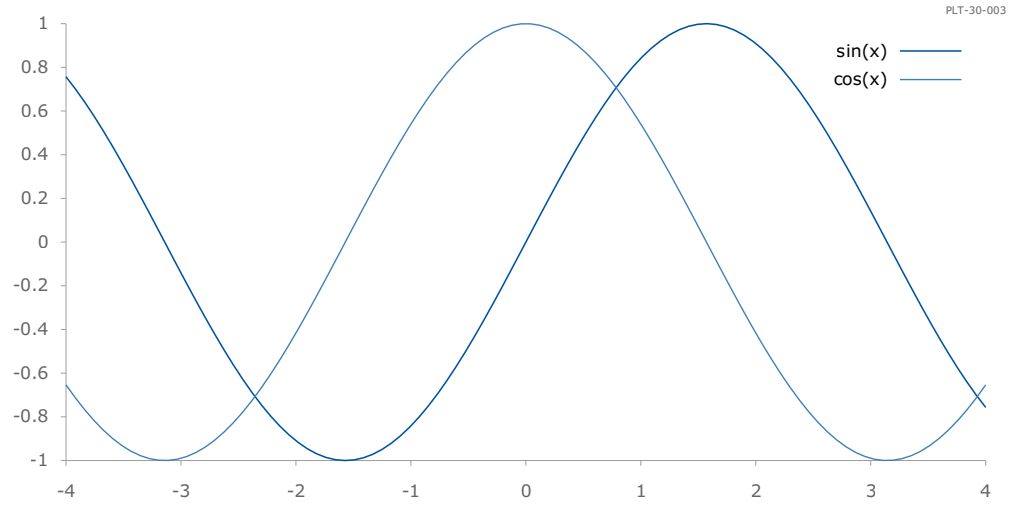
`load 'config/my-style.gp'`	Read and execute the style commands before plotting; filenames must be quoted.
`config/`	A project folder relative to the current working directory.
`load '~/.config/gnuplot/my-style.gp'`	Alternative personal location: ~ refers to your home directory on macOS/Linux.
`~/.gnuplot`	Optional startup file for ordinary gnuplot sessions; the tour uses -d and loads styles explicitly.

Load RWTH–Tufte for a line plot

Compare with PLT-30-002 · Load a project or personal style.

The complete rwth-tufte.gp file follows this example, with a download. Create a config folder in your working directory and save the downloaded file there. Load this one file, then choose the plot geometry explicitly with with lines. The file supplies named colours, series, quiet axes, bar fills and a sequential blue palette, without selecting a terminal, output file or data ranges. Enable the legend for these two named curves. The font settings adapt the style's small-figure defaults to the 1200-by-600 tour canvas.

```
load 'config/rwth-tufte.gp'
set tics font ',20'
set key top right nobox font ',20'
set xrange [-4:4]
plot sin(x) with lines title 'sin(x)', \
     cos(x) with lines title 'cos(x)'
```



PLT-30-003 · Generated from the complete Bash block

`config/rwth-tufte.gp`	The reusable base; blue series, light axes, no grid and no key until enabled.
`with lines`	Choose line geometry in the plot; the style file does not select a plot type.
`set key top right nobox`	Enable a top-right legend for the named curves.
`set tics; set key font`	Change font sizes while retaining the font family.
`style -> local settings -> plot`	Load the appearance once, then choose the legend, geometry and local overrides.
`Complete style file`	Read or download rwth-tufte.gp below and save it in config/.

config/rwth-tufte.gp

Save this file at `config/rwth-tufte.gp` in your working directory. Download rwth-tufte.gp

```
# rwth-tufte.gp -- RWTH Aachen colors / Tufte-inspired low-ink defaults
# Works with gnuplot 5.4+ (no output/terminal commands).
# ONE LOAD ONLY: load '~/.config/gnuplot/rwth-tufte.gp'
# or             load 'config/rwth-tufte.gp' (project-local)
# Plot types are selected with 'with lines', 'with boxes', 'with points', etc.
# Do not load another .gp file just to change plot geometry.
# This is not an official RWTH graphical standard.
# No reset, terminal, output, title, plot ranges, fixed margins, or
# forced data geometry. Each plot explicitly selects its own 'with' style.

# Named colors (RWTH Aachen color palette)
RWTH_BLUE       = '#00549F'
RWTH_BLUE_75    = '#407FB7'
RWTH_BLUE_50    = '#8EBAE5'
RWTH_BLUE_25    = '#C7DDF2'
RWTH_BLUE_10    = '#E8F1FA'
RWTH_GRAY_75    = '#646567'
RWTH_GRAY_50    = '#9C9E9F'
RWTH_GRAY_25    = '#CFD1D2'
RWTH_GRAY_10    = '#ECEDED'
RWTH_ORANGE     = '#F6A800'
RWTH_GREEN      = '#57AB27'
RWTH_RED        = '#CC071E'
RWTH_TURQUOISE  = '#0098A1'
RWTH_LILAC      = '#7A6FAC'

# Line and point series: color + varying point/line shapes aid print reading
set linetype 1 lc rgb RWTH_BLUE       lw 1.8 pt 7  ps 0.75
set linetype 2 lc rgb RWTH_BLUE_75    lw 1.6 pt 5  ps 0.75
set linetype 3 lc rgb RWTH_TURQUOISE  lw 1.6 pt 9  ps 0.75
set linetype 4 lc rgb RWTH_ORANGE     lw 1.6 pt 11 ps 0.75
set linetype 5 lc rgb RWTH_GREEN      lw 1.6 pt 13 ps 0.75
set linetype 6 lc rgb RWTH_RED        lw 1.6 pt 3  ps 0.75
set linetype 7 lc rgb RWTH_LILAC      lw 1.6 pt 4  ps 0.75
set linetype 8 lc rgb RWTH_GRAY_75    lw 1.3 pt 6  ps 0.75
set linetype cycle 8
```

```
# Minimal non-data ink; no enclosing box, no default grid
unset grid
set border 3 back lw 0.6 lc rgb RWTH_GRAY_50
set tics nomirror out scale 0.4
set xtics font ',10' textcolor rgb RWTH_GRAY_75
set ytics font ',10' textcolor rgb RWTH_GRAY_75
set xlabel font ',11' textcolor rgb RWTH_GRAY_75
set ylabel font ',11' textcolor rgb RWTH_GRAY_75
set title font ',12' textcolor rgb RWTH_GRAY_75

# Only show key when a plot really needs it
set key nobox font ',10' spacing 1.1
unset key

# Reusable line styles for plot-specific reference lines or faint grid
set style line 101 lc rgb RWTH_GRAY_25 lw 0.6
set style line 102 lc rgb RWTH_GRAY_50 lw 0.9

# Common bar/histogram defaults (do not force 'with histograms' / 'with boxes')
set boxwidth 0.65 relative
set style fill solid 0.85 noborder
set style histogram clustered gap 2

# Sequential palette: readable when rasterized or printed
set palette defined ( 0.00 RWTH_BLUE_10, \
                      0.25 RWTH_BLUE_25, \
                      0.50 RWTH_BLUE_50, \
                      0.75 RWTH_BLUE_75, \
                      1.00 RWTH_BLUE )

# Optional configuration for *one plot* (not global):
#   set key top right nobox
#   set grid ytics ls 101
#   set yrange [0:*]
#   set term svg size 900,360   # Babel :file chooses terminal automatically

# --- Ready for many plot geometries with this ONE file ---
# lines:       plot sin(x) with lines
# scatter:     plot 'data' using 1:2 with points
# boxes:       set yrange [0:*]; plot 'data' using 2:xticlabels(1) with boxes
# histogram:   set style histogram rowstacked  # only when stacking is intended
# boxplots:    plot 'data' using 1:2 with boxplot
# error bars:  plot 'data' using 1:2:3 with yerrorbars
# heatmaps:    set view map; set pm3d map; splot f(x,y) with pm3d
# contours:    set view map; unset surface; set contour base; splot f(x,y)
```

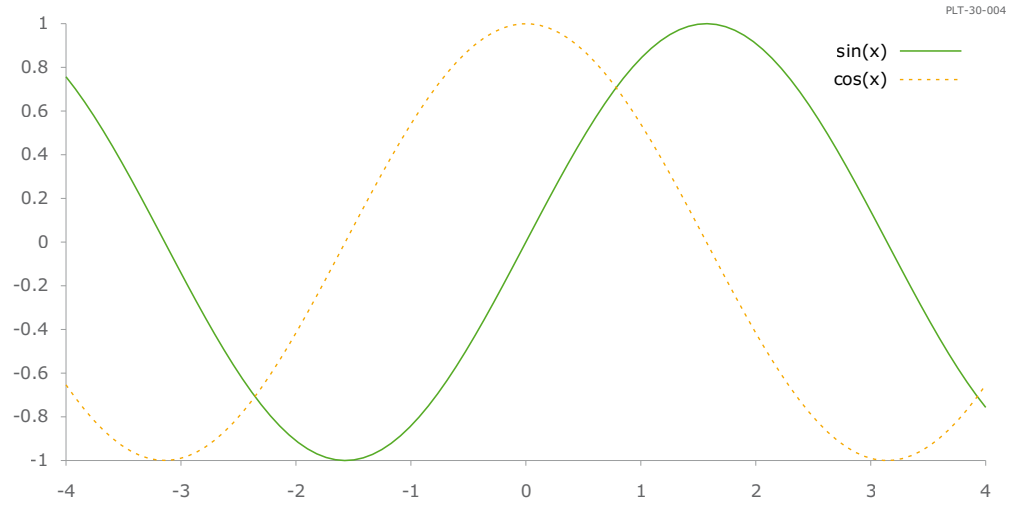
```
# 3D:         set border 4095; splot f(x,y) with lines
# polar:       set polar; set angles degrees; plot f(t) with lines
# time:        set xdata time; set timefmt '%Y-%m-%d'
# stacked:     set style histogram rowstacked
#                 plot 'data' using 2:xticlabels(1) with histograms, \
#                      '' using 3 with histograms
# Diverging palette (only if signed quantity requires centered color scale):
#    set palette defined (0 RWTH_BLUE, 0.5 '#FFFFFF', 1 RWTH_ORANGE)
#
# These commands describe the data/geometry and belong to individual plots.
# This file supplies the common *appearance*, without changing the plot type.
```

Use the library’s named colours

Compare with PLT-30-003 · Load RWTH–Tufte for a line plot.

Keep the loaded style and line geometry, then select named colours for individual series. RWTH_GREEN and RWTH_ORANGE are variables defined by the base file, so load it before using them. Explicit line colours override the corresponding series defaults without changing the axes or other series. Different dash patterns also distinguish the curves.

```
load 'config/rwth-tufte.gp'
set tics font ',20'
set key top right nobox font ',20'
set xrange [-4:4]
plot sin(x) with lines lc rgb RWTH_GREEN dt 1 title 'sin(x)', \
     cos(x) with lines lc rgb RWTH_ORANGE dt 2 title 'cos(x)'
```



PLT-30-004 · Generated from the complete Bash block

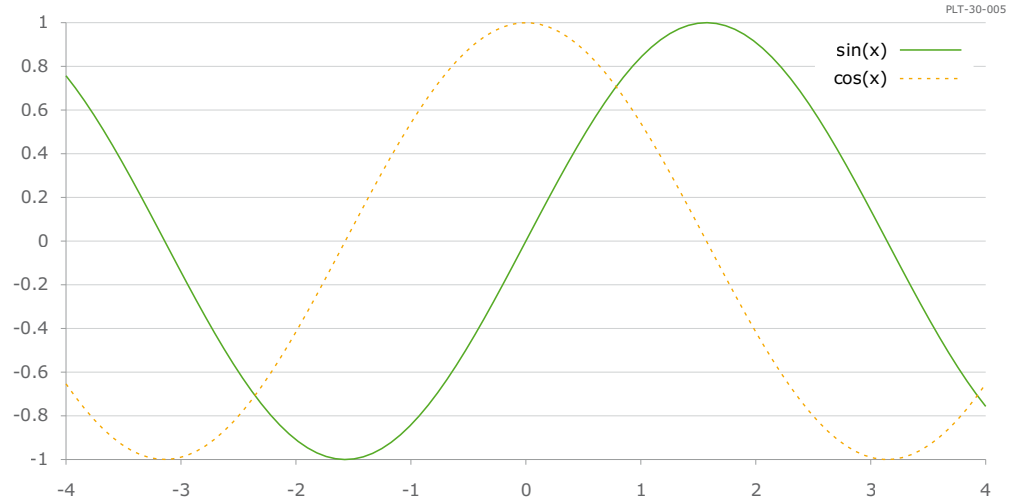
`lc rgb RWTH_GREEN`	Use the named green colour defined by the base file.
`lc rgb RWTH_ORANGE`	Use the named orange colour for this series.
`dt 1; dt 2`	Select solid and dashed lines in addition to the colour distinction.

Override a style for one plot

Compare with PLT-30-004 · Use the library’s named colours.

Add a faint horizontal grid after loading the style. The base defines line style 101 as a light grey reference line. Loading the base again after set grid would unset the grid, so put local overrides last. The same rule applies to legends, fonts, fills and palettes. Each tour plot uses a fresh process; this override does not carry into another example.

```
load 'config/rwth-tufte.gp'
set tics font ',20'
set key top right nobox font ',20'
set xrange [-4:4]
set grid ytics back ls 101
plot sin(x) with lines lc rgb RWTH_GREEN dt 1 title 'sin(x)', \
     cos(x) with lines lc rgb RWTH_ORANGE dt 2 title 'cos(x)'
```



PLT-30-005 · Generated from the complete Bash block

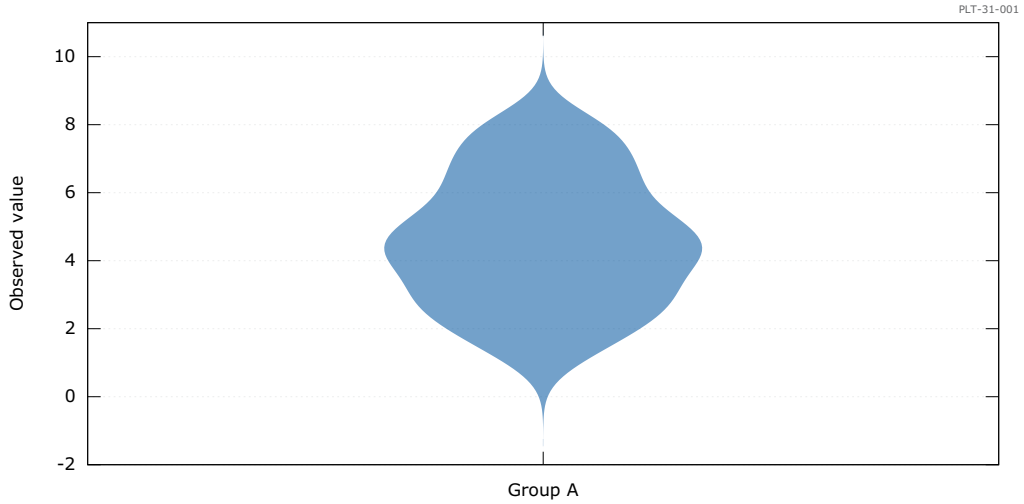
`set grid ytics`	Draw horizontal guides at the major y ticks.
`back ls 101`	Draw behind the data using the base file’s faint grey line style.
`Override order`	Apply local settings after the load command; the last setting of a property wins.

Mirror one empirical density

New baseline: the commands below define this example.

Sum Gaussian kernels centred on the actual sample values, then trace the right boundary upwards and the left boundary downwards to close the violin. No analytical bell curve stands in for the observations.

```
N=24
array
A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6.5,7,7
.2,7.8,8.1]
array
B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7.5,7.8
,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.65; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/(N*bw*sqrt(2*pi))
groups=1
set print $V
do for [g=1:groups] {
    do for [side=1:-1:-2] {
        do for [k=0:120] {
            v=-2+(side==1?k:120-k)*13.0/120
            print g+side*width*density(v,g),v
        }
    }
    print ""; print ""
}
set print
set xrange [0.3:1.7]
set xtics ("Group A" 1)
set yrange [-2:11]
set ylabel "Observed value"
set grid ytics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
plot for [g=1:groups] $V index (g-1) using 1:2 with filledcurves closed lc rgb
C[g] notitle
```



PLT-31-001 · Generated from the complete Bash block

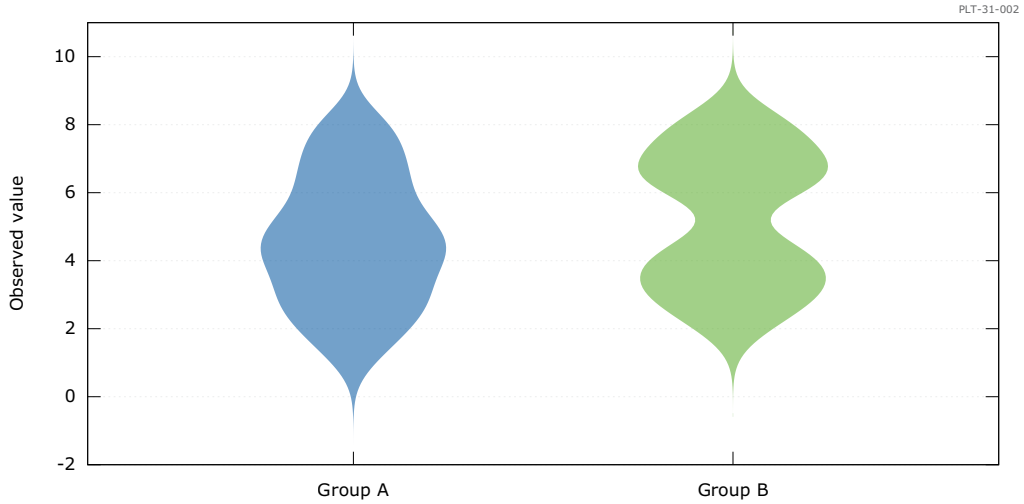
`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`closed / reversed return path`	Trace one boundary up and the other down; blank blocks separate category polygons.

Compare two groups on a common scale

Compare with PLT-31-001 · Mirror one empirical density.

Keep the kernel bandwidth and width multiplier fixed while adding a second sample. Both groups have equal sample size and unit-area densities; widths therefore have the same meaning.

```
N=24
array
A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6.5,7,7
.2,7.8,8.1]
array
B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7.5,7.8
,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.65; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/(N*bw*sqrt(2*pi))
groups=2
set print $V
do for [g=1:groups] {
    do for [side=1:-1:-2] {
        do for [k=0:120] {
            v=-2+(side==1?k:120-k)*13.0/120
            print g+side*width*density(v,g),v
        }
    }
    print ""; print ""
}
set print
set xrange [0.3:2.7]
set xtics ("Group A" 1,"Group B" 2)
set yrange [-2:11]
set ylabel "Observed value"
set grid ytics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
plot for [g=1:groups] $V index (g-1) using 1:2 with filledcurves closed lc rgb
C[g] notitle
```



PLT-31-002 · Generated from the complete Bash block

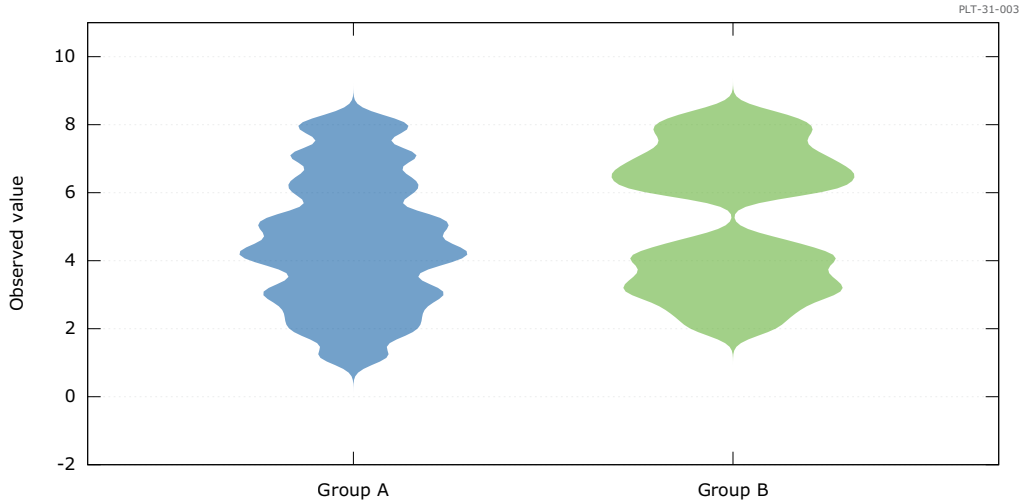
`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`closed / reversed return path`	Trace one boundary up and the other down; blank blocks separate category polygons.

Expose the effect of a smaller bandwidth

Compare with PLT-31-002 · Compare two groups on a common scale.

Change only bw from 0.65 to 0.25. Narrower kernels reveal local peaks but also magnify sampling noise; a bump in a small sample need not represent a separate population.

```
N=24
array
A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6.5,7,7
.2,7.8,8.1]
array
B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7.5,7.8
,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.25; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/(N*bw*sqrt(2*pi))
groups=2
set print $V
do for [g=1:groups] {
    do for [side=1:-1:-2] {
        do for [k=0:120] {
            v=-2+(side==1?k:120-k)*13.0/120
            print g+side*width*density(v,g),v
        }
    }
    print ""; print ""
}
set print
set xrange [0.3:2.7]
set xtics ("Group A" 1,"Group B" 2)
set yrange [-2:11]
set ylabel "Observed value"
set grid ytics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
plot for [g=1:groups] $V index (g-1) using 1:2 with filledcurves closed lc rgb
C[g] notitle
```



PLT-31-003 · Generated from the complete Bash block

`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`closed / reversed return path`	Trace one boundary up and the other down; blank blocks separate category polygons.

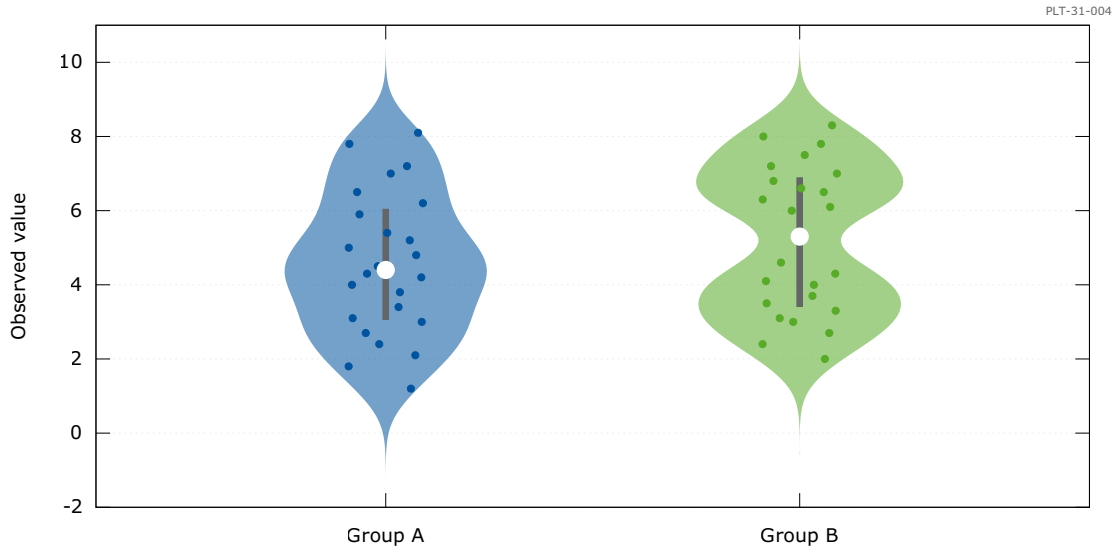
Overlay observations and quartiles

Compare with PLT-31-002 · Compare two groups on a common scale.

Keep the smoothed shape and add every observation with deterministic horizontal jitter. A thick segment marks the interquartile range and a white point marks the median; neither is a confidence interval.

```
N=24
array A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6.5,7,7.2,7.8,8.1]
array B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7.5,7.8,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.65; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/(N*bw*sqrt(2*pi))
groups=2
set print $V
do for [g=1:groups] {
    do for [side=1:-1:-2] {
        do for [k=0:120] {
            v=-2+(side==1?k:120-k)*13.0/120
            print g+side*width*density(v,g),v
        }
    }
    print ""; print ""
}
set print
set xrange [0.3:2.7]
set xtics ("Group A" 1,"Group B" 2)
set yrange [-2:11]
set ylabel "Observed value"
set grid ytics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
stats A using 2 name "A_" nooutput
stats B using 2 name "B_" nooutput
set print $M
print 1,A_median,A_lo_quartile,A_up_quartile
print 2,B_median,B_lo_quartile,B_up_quartile
set print
plot for [g=1:groups] $V index (g-1) using 1:2 with filledcurves closed lc rgb C[g] notitle, \
     $M using 1:3:(0):($4-$3) with vectors nohead lw 7 lc rgb "#646567" notitle, \
     A using (1+0.09*sin($1*2.4)):2 with points pt 7 ps 0.5 lc rgb "#00549F" notitle, \
```

```
     B using (2+0.09*sin($1*2.4)):2 with points pt 7 ps 0.5 lc rgb "#57AB27" notitle, \
     $M using 1:2 with points pt 7 ps 1.2 lc rgb "white" notitle
```



PLT-31-004 · Generated from the complete Bash block

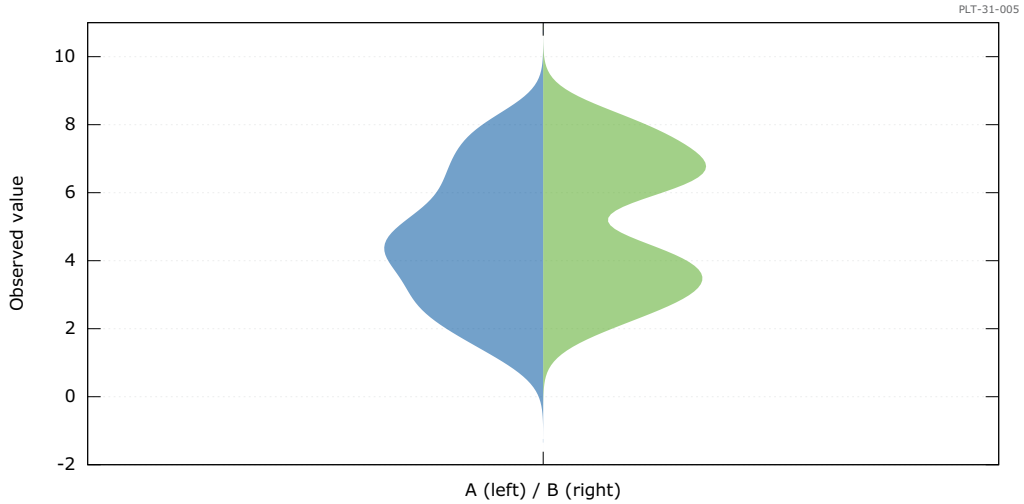
`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`closed / reversed return path`	Trace one boundary up and the other down; blank blocks separate category polygons.
`stats ... lo_quartile / median / up_quartile`	Compute summaries from the observations, independently of the smoothed density.

Split two groups across one violin

Compare with PLT-31-002 · Compare two groups on a common scale.

Place group A on the left and group B on the right of the same category centre. Shared bandwidth and width scaling make the two half-violins comparable; use this for a paired category with two conditions.

```
N=24
array
A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6.5,7,7
.2,7.8,8.1]
array
B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7.5,7.8
,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.65; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/(N*bw*sqrt(2*pi))
groups=2
set print $V
do for [g=1:groups] {
    do for [side=1:0:-1] {
        do for [k=0:120] {
            v=-2+(side==1?k:120-k)*13.0/120
            print 1+side*(g==1?-1:1)*width*density(v,g),v
        }
    }
    print ""; print ""
}
set print
set xrange [0.3:1.7]
set xtics ("A (left) / B (right)" 1)
set yrange [-2:11]
set ylabel "Observed value"
set grid ytics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
plot for [g=1:groups] $V index (g-1) using 1:2 with filledcurves closed lc rgb
C[g] notitle
```



PLT-31-005 · Generated from the complete Bash block

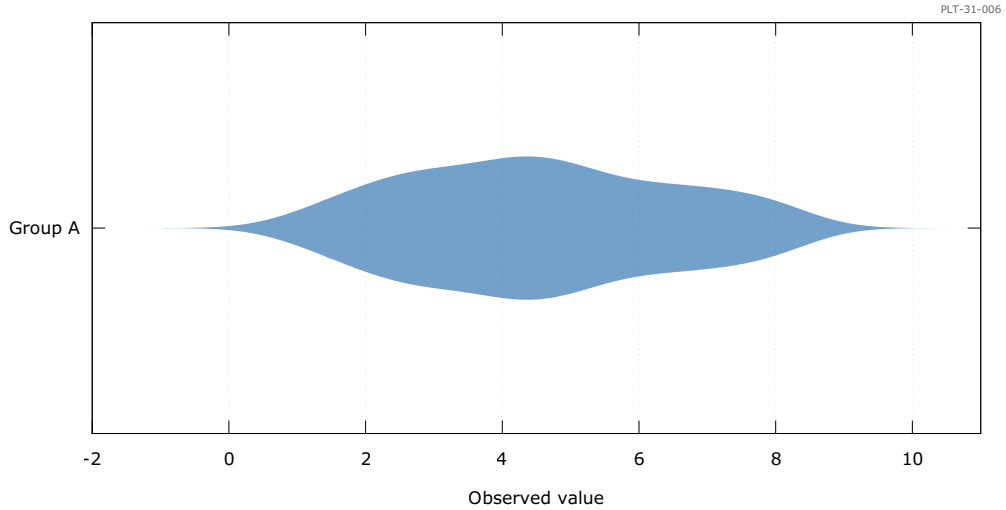
`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`closed / reversed return path`	Trace one boundary up and the other down; blank blocks separate category polygons.
`side=1:0:-1`	Trace the outer boundary and return on the category centre, giving one half per condition.

Turn the density sideways with native kdensity

Compare with PLT-31-005 · Split two groups across one violin.

Use gnuplot smooth kdensity directly, storing its sampled output before filling above and below a horizontal centre line. Each input receives weight 1/N, giving unit area; changing orientation does not change the sample.

```
N=24
array
A[N]=[1.2,1.8,2.1,2.4,2.7,3,3.1,3.4,3.8,4,4.2,4.3,4.5,4.8,5,5.2,5.4,5.9,6.2,6
.5,7,7.2,7.8,8.1]
array
B[N]=[2,2.4,2.7,3,3.1,3.3,3.5,3.7,4,4.1,4.3,4.6,6,6.1,6.3,6.5,6.6,6.8,7,7.2,7
.5,7.8,8,8.3]
array C[2]=["#00549F","#57AB27"]
bw=0.65; width=1.3
density(v,g)=sum [j=1:N] exp(-0.5*((v-(g==1?A[j]:B[j]))/bw)**2)/
(N*bw*sqrt(2*pi))
set samples 161
set xrange [-2:11]
set table $K
plot A using 2:(1.0/N) smooth kdensity bandwidth bw
unset table
set yrange [0.3:1.7]
set ytics ("Group A" 1)
set xlabel "Observed value"
set grid xtics back lc rgb "#ECEDED"
set style fill transparent solid 0.55 noborder
plot $K using 1:(1-width*$2):(1+width*$2) with filledcurves lc rgb "#00549F"
notitle
```



PLT-31-006 · Generated from the complete Bash block

`density(v,g)`	A normalized Gaussian KDE: divide the sum of kernels by N, bandwidth and sqrt(2*pi).
`bw`	Bandwidth in the same measurement units as the observations; all groups use the same value.
`width=1.3`	One common density-to-width conversion, rather than separately maximizing each violin.
`A using 2:(1.0/N)`	Array column 1 is the index; column 2 is the value. Equal weights sum to one.
`set table $K`	Capture native KDE samples; using columns 1, lower edge and upper edge then fills a horizontal violin.

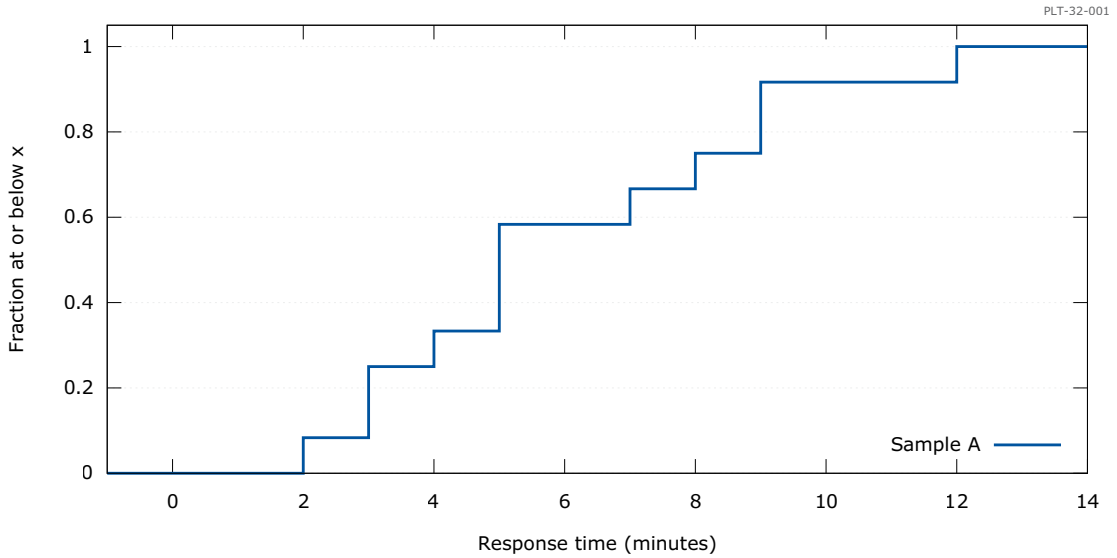
Draw a right-continuous ECDF with ties

New baseline: the commands below define this example.

Each record carries unit mass. smooth cnormal sorts values, merges ties and divides by total mass. Explicit boundary records extend F=0 before the first observation and F=1 after the last; steps uses the value immediately to the right of each jump.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:(1.0) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
set xrange [-1:14]
set yrange [0:1.05]
```

```
plot $E using 1:2 with steps lw 3 lc rgb "#00549F" title "Sample A"
```



PLT-32-001 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.

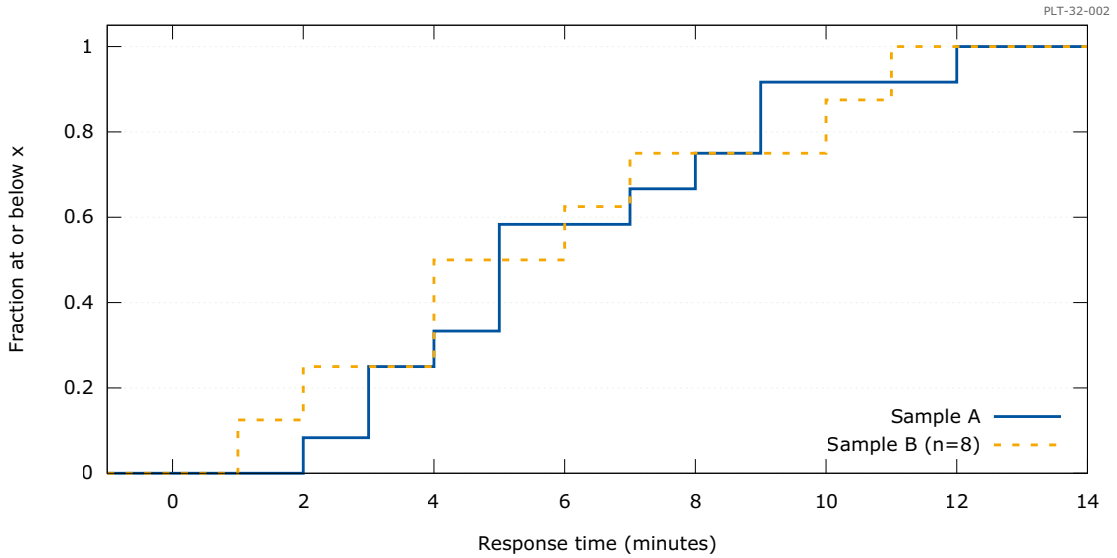
Compare samples of different sizes

Compare with PLT-32-001 · Draw a right-continuous ECDF with ties.

Add a second sample with eight observations instead of twelve. Normalize each sample independently, so both curves end at one; raw cumulative counts would confound distribution shape with sample size.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:(1.0) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
$B << EOD
1
2
```

```
4
4
6
7
10
11
EOD
set table $G
plot $B using 1:(1.0) smooth cnormal
unset table
set print $H
print -1,0
do for [j=1:|$G|] {
    if (word($G[j],3) eq "i") { print real(word($G[j],1)),real(word($G[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
plot $E using 1:2 with steps lw 3 lc rgb "#00549F" title "Sample A", \
     $H using 1:2 with steps lw 3 dt 2 lc rgb "#F6A800" title "Sample B (n=8)"
```



PLT-32-002 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.
`Independent cnormal passes`	Each group receives its own denominator; sample A has 12 records and B has 8.

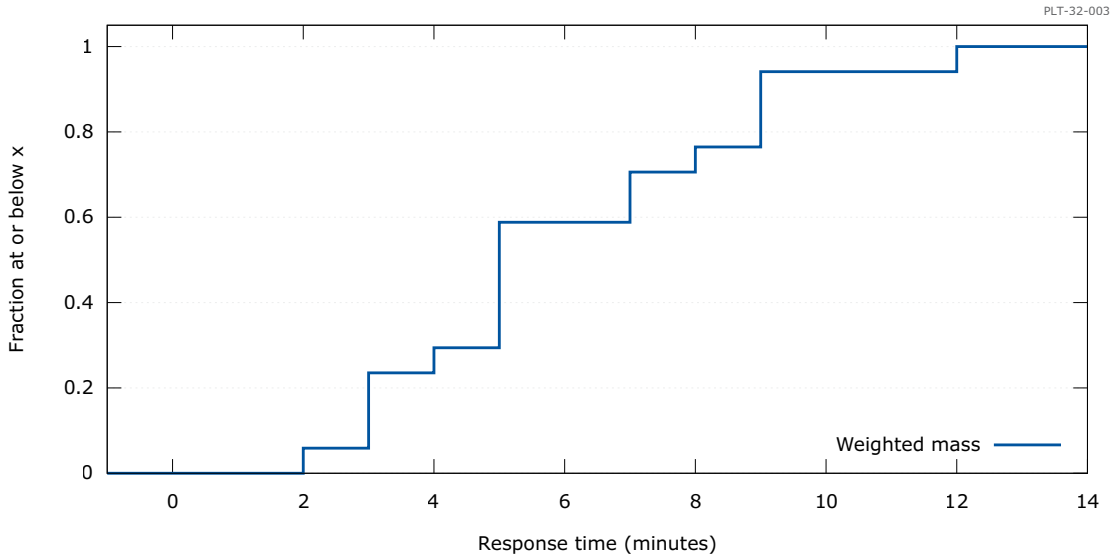
Accumulate observation weights

Compare with PLT-32-001 · Draw a right-continuous ECDF with ties.

Use the second input column as positive mass instead of counting every row equally. This is a weighted empirical distribution, not twelve independent observations; the DKW band below applies only to the unweighted sample.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:($2) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
set xrange [-1:14]
set yrange [0:1.05]
```

```
plot $E using 1:2 with steps lw 3 lc rgb "#00549F" title "Weighted mass"
```



PLT-32-003 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.
`using 1:($2)`	Positive integer weights add mass at each x; the denominator is the sum of weights, not the row count.

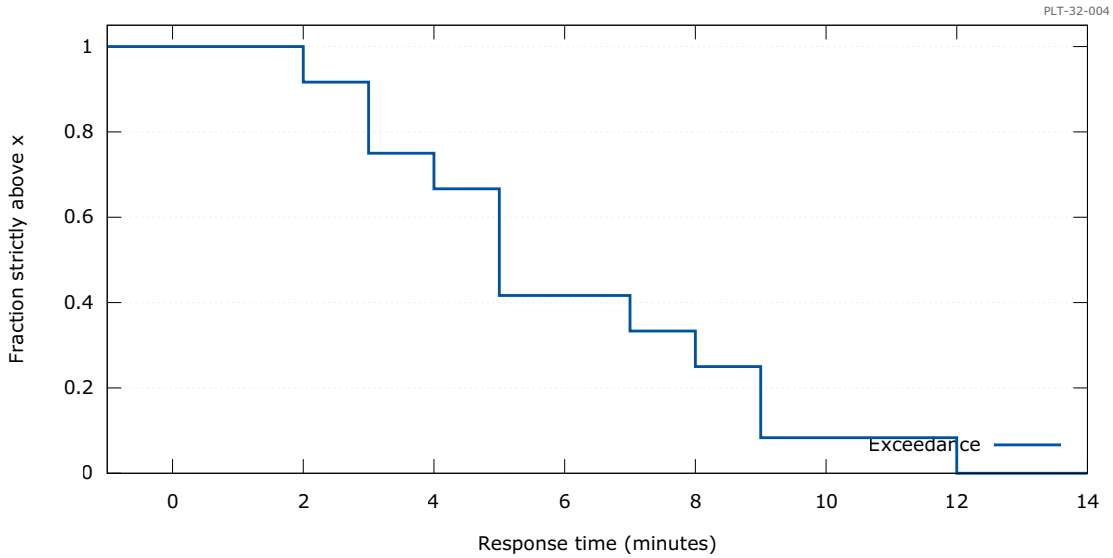
Plot the fraction strictly above a threshold

Compare with PLT-32-001 · Draw a right-continuous ECDF with ties.

Replace F(x) by 1-F(x), the observed fraction strictly greater than x. This complements the at-or-below ECDF, including its treatment of ties; the result is an empirical exceedance curve rather than a fitted survival model.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:(1.0) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
set ylabel "Fraction strictly above x"
set xrange [-1:14]
set yrange [0:1.05]
```

```
plot $E using 1:(1-$2) with steps lw 3 lc rgb "#00549F" title "Exceedance"
```



PLT-32-004 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.
`1-$2`	Complement the normalized ECDF; ties at the threshold are excluded from the exceedance fraction.

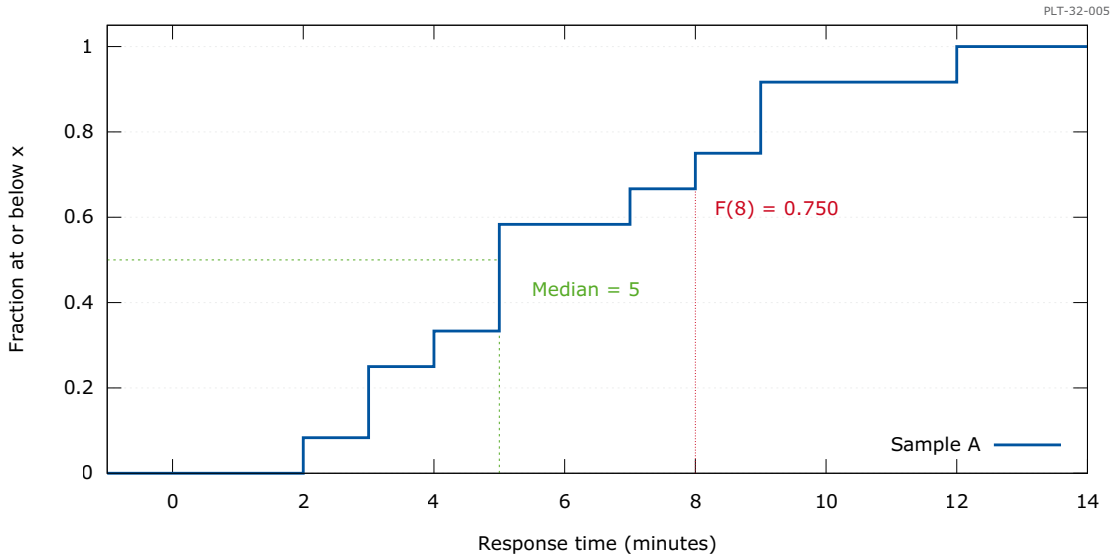
Read the empirical median and a service threshold

Compare with PLT-32-001 · Draw a right-continuous ECDF with ties.

Find the first observed value whose ECDF reaches one half, rather than interpolating between observations. Also evaluate the fraction at or below an independently chosen threshold of eight; dashed guides make both lookups explicit.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:(1.0) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
set autoscale
stats $F using ($2>=0.5?$1:1/0) name "Q_" nooutput
median=Q_min
```

```
stats $F using ($1<=8?$2:1/0) name "T_" nooutput
p=T_max
set arrow 1 from median,0 to median,0.5 nohead dt 2 lc rgb "#57AB27"
set arrow 2 from -1,0.5 to median,0.5 nohead dt 2 lc rgb "#57AB27"
set arrow 3 from 8,0 to 8,p nohead dt 3 lc rgb "#CC071E"
set label 1 sprintf("Median = %g",median) at 5.5,0.43 tc rgb "#57AB27"
set label 2 sprintf("F(8) = %.3f",p) at 8.3,0.62 tc rgb "#CC071E"
set xrange [-1:14]
set yrange [0:1.05]
plot $E using 1:2 with steps lw 3 lc rgb "#00549F" title "Sample A"
```



PLT-32-005 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.
`Q_min / T_max`	First x with F>=0.5 is 5; largest F at x<=8 is 9/12=0.75.

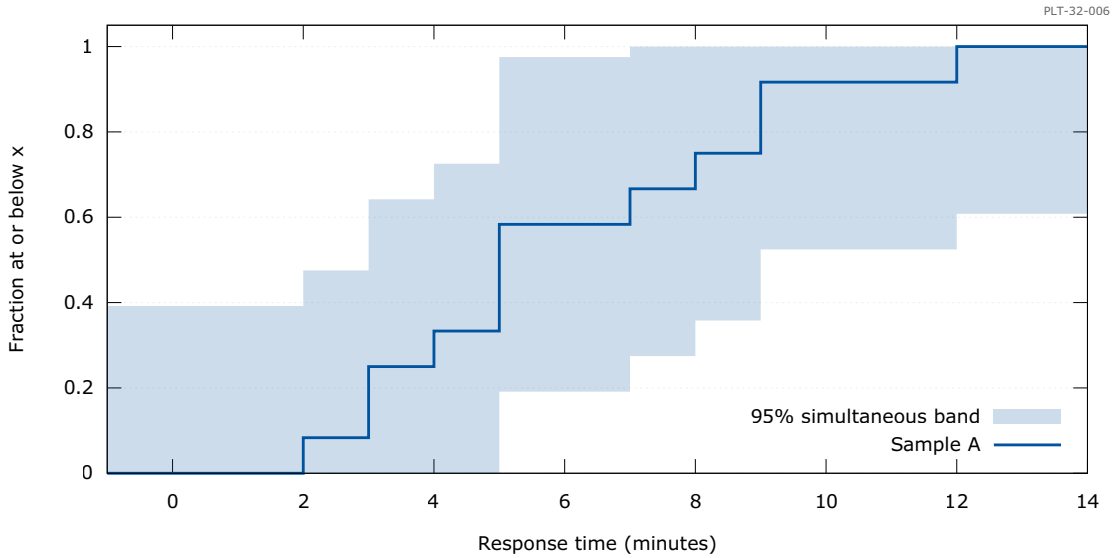
Add a simultaneous DKW confidence band

Compare with PLT-32-001 · Draw a right-continuous ECDF with ties.

For independent identically distributed observations, the Dvoretzky-Kiefer-Wolfowitz inequality gives a distribution-free simultaneous band F_n(x) plus or minus epsilon, with epsilon=sqrt(log(2/alpha)/(2N)). The band is clipped to [0,1], uses N=12 and alpha=0.05, and is intentionally wide for this small sample.

```
$A << EOD
2 1
3 2
3 1
4 1
5 3
5 1
5 1
7 2
8 1
9 1
9 2
12 1
EOD
set table $F
plot $A using 1:(1.0) smooth cnormal
unset table
set print $E
print -1,0
do for [j=1:|$F|] {
    if (word($F[j],3) eq "i") { print real(word($F[j],1)),real(word($F[j],2)) }
}
print 14,1
set print
set xrange [-1:14]
set yrange [0:1.05]
set xlabel "Response time (minutes)"
set ylabel "Fraction at or below x"
set grid ytics back lc rgb "#ECEDED"
set key bottom right
set autoscale
stats $A using 1 noutput
alpha=0.05; eps=sqrt(log(2/alpha)/(2*STATS_records))
```

```
set table $P
plot $E using 1:2 with table
unset table
lo(p)=p>eps?p-eps:0
hi(p)=p+eps<1?p+eps:1
set print $Band
previous=0
do for [j=1:|$P|] {
    if (words($P[j])>=2) {
        x=real(word($P[j],1)); p=real(word($P[j],2))
        print x,lo(previous),hi(previous)
        print x,lo(p),hi(p)
        previous=p
    }
}
set print
set style fill transparent solid 0.2 noborder
set xrange [-1:14]
set yrange [0:1.05]
plot $Band using 1:2:3 with filledcurves lc rgb "#00549F" title "95% simultaneous band", \
     $E using 1:2 with steps lw 3 lc rgb "#00549F" title "Sample A"
```



PLT-32-006 · Generated from the complete Bash block

`smooth cnormal`	Sort x, combine ties and divide accumulated positive mass by its total.
`with steps`	Right-continuous steps: hold the previous level until the next observed x.
`-1,0 / 14,1`	Explicit tails for this fixture; choose boundaries outside the data when adapting it.
`eps`	The DKW bound uses natural logarithms and the number of unweighted IID observations.
`Duplicate x coordinates`	Expand vertical jumps before filling so the confidence band has the same step geometry as the ECDF.

Count observations in true hexagonal bins

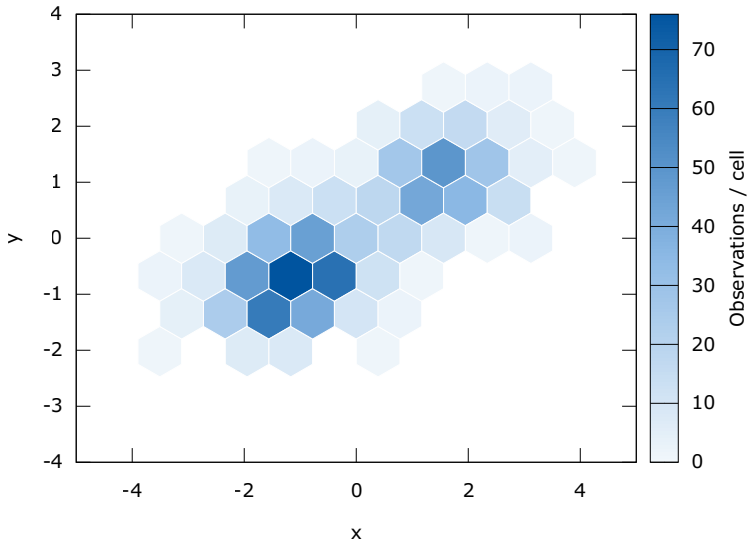
New baseline: the commands below define this example.

Convert Cartesian observations to fractional axial coordinates, round all three cube coordinates and correct the largest rounding error so q+r+z=0. Accumulate counts before drawing regular hexagons; every observation contributes to exactly one cell.

```
N=800; radius=0.45; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Count[j]
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [0:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Observations / cell"
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle
```

PLT-33-001



PLT-33-001 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.

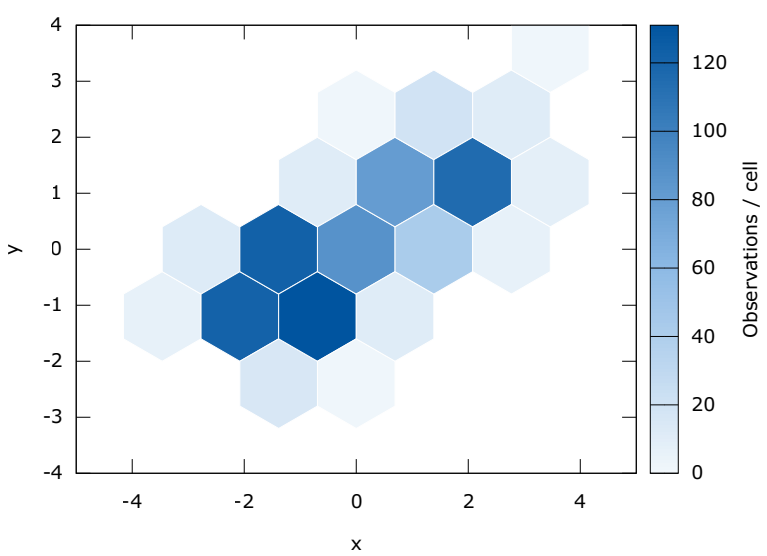
Trade detail for larger bins

Compare with PLT-33-001 · Count observations in true hexagonal bins.

Change only the circumradius from 0.45 to 0.8 while keeping the same seeded sample. Larger cells stabilize counts but merge nearby structure; the colour scale remains counts per cell, so numerical values rise when cell area grows.

```
N=800; radius=0.8; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Count[j]
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [0:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Observations / cell"
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle
```



PLT-33-002 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.

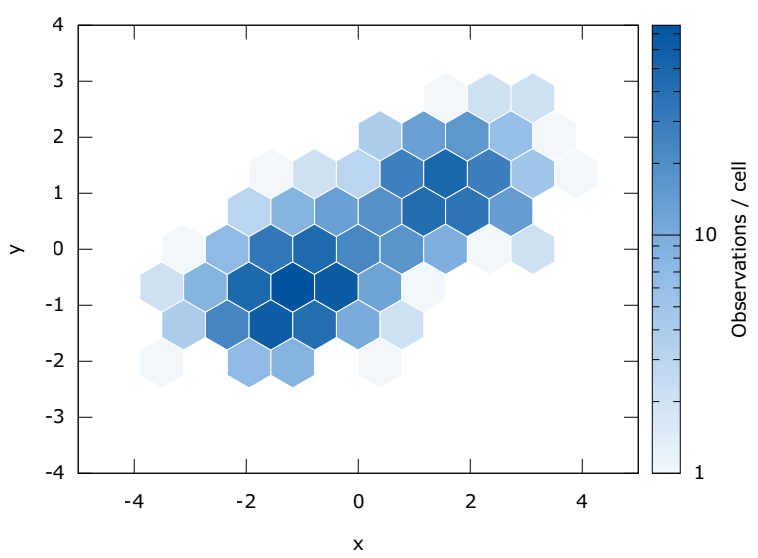
Reveal sparse cells with logarithmic colour

Compare with PLT-33-001 · Count observations in true hexagonal bins.

Return to radius 0.45 and logarithmically scale counts on the colour axis. Every occupied cell has count at least one. Empty cells remain blank and are not assigned an artificial log(0) colour.

```
N=800; radius=0.45; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Count[j]
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [1:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Observations / cell"
set logscale cb
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle
```



PLT-33-003 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.
`set logscale cb`	Apply a logarithm to positive colour values while keeping the displayed counts in their original units.

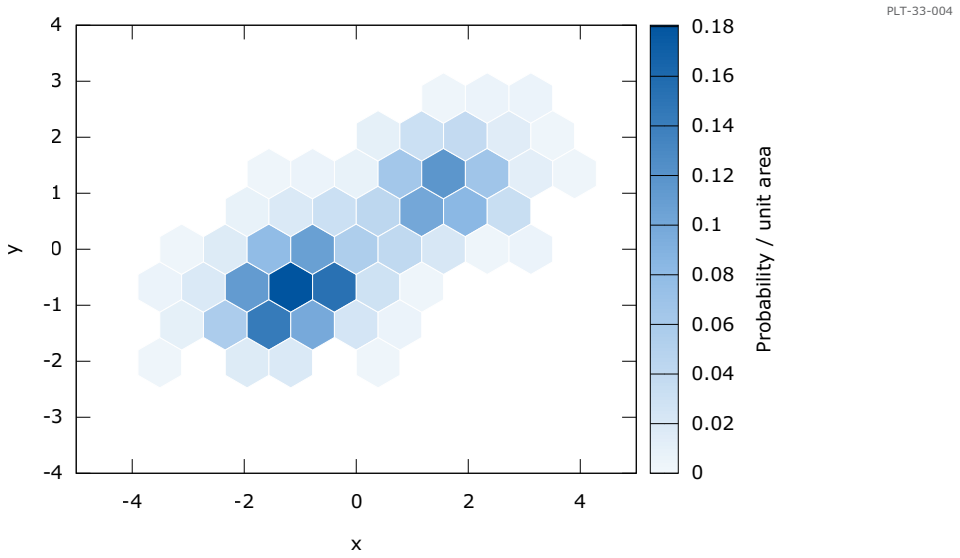
Normalize counts to probability density

Compare with PLT-33-001 · Count observations in true hexagonal bins.

Divide each cell count by the total sample size and regular-hexagon area, 3*sqrt(3)*r*r/2. The resulting units are probability per square coordinate unit; summing density times cell area over all occupied cells gives one.

```
N=800; radius=0.45; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Count[j]/(N*area)
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [0:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Probability / unit area"
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle
```



PLT-33-004 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.
`N*area`	Convert counts to an area-normalized probability density; values are not percentages per cell.

Map a measurement mean instead of frequency

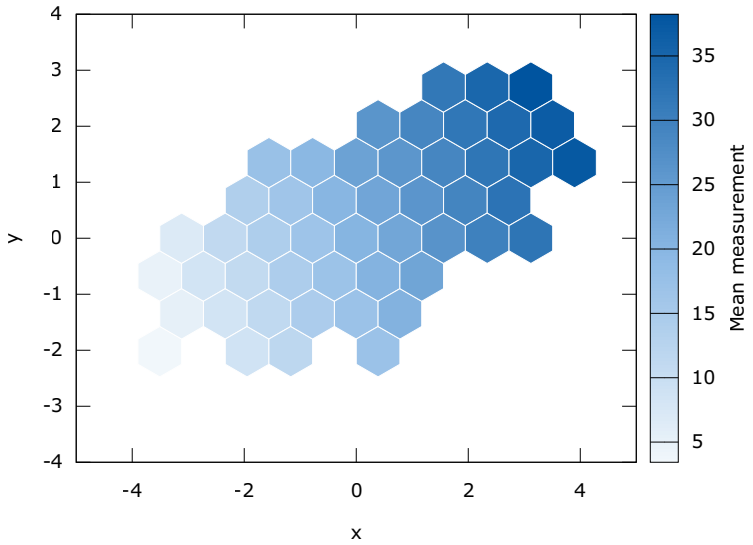
Compare with PLT-33-001 · Count observations in true hexagonal bins.

Accumulate a second quantity along with counts and colour each occupied cell by its mean measurement. This is a binned response map, not an observation-density map; no value is imputed for empty cells.

```
N=800; radius=0.45; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Total[j]/Count[j]
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [STATS_min:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Mean measurement"
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle
```

PLT-33-005



PLT-33-005 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.
`Total[j]/Count[j]`	The mean of the synthetic measurement 20+4*x+2*y among observations assigned to this cell.

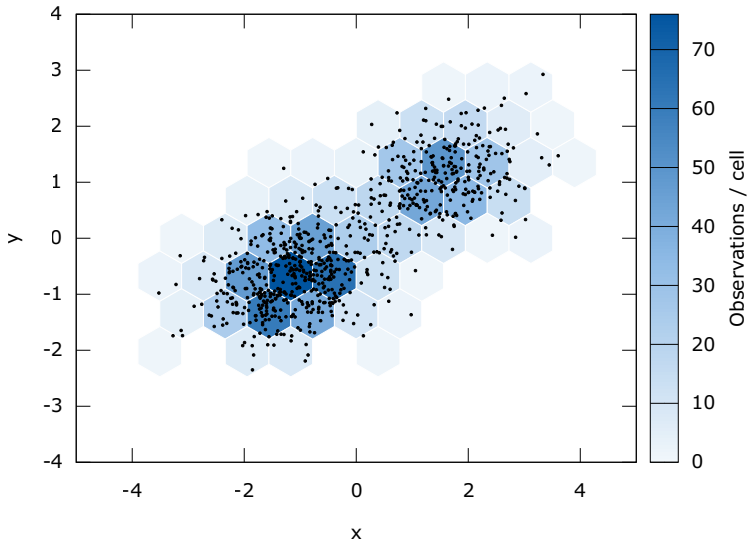
Overlay the original observations

Compare with PLT-33-001 · Count observations in true hexagonal bins.

Add the seeded source points above the count cells using small black markers. The overlay explains the aggregation visually and helps distinguish a dense cloud from a single large marker; remove it for very large datasets.

```
N=800; radius=0.45; offset=24; size=49
array Count[size*size]; array Total[size*size]; array Value[size*size]
do for [j=1:size*size] { Count[j]=0; Total[j]=0 }
seed=rand(73491)
set print $D
do for [j=1:N] {
    u=rand(0); v=rand(0)
    z=sqrt(-2*log(u+1e-12))
    x=(j<=500?-1.1:1.5)+0.75*z*cos(2*pi*v)
    y=(j<=500?-0.45:0.8)+0.55*z*sin(2*pi*v)+0.2*x
    measurement=20+4*x+2*y
    print x,y
    qf=(sqrt(3)*x/3-y/3)/radius; rf=2*y/(3*radius)
    zf=-qf-rf; q=floor(qf+0.5); r=floor(rf+0.5); zz=floor(zf+0.5)
    dq=abs(q-qf); dr=abs(r-rf); dz=abs(zz-zf)
    if (dq>dr && dq>dz) { q=-r-zz } else { if (dr>dz) { r=-q-zz } }
    cell=(q+offset)*size+r+offset+1
    if (abs(q)>offset || abs(r)>offset) { exit error "Increase hex grid extent" }
    Count[cell]=Count[cell]+1; Total[cell]=Total[cell]+measurement
}
set print
area=3*sqrt(3)*radius**2/2
bins=0; total_count=0
set print $Cells
do for [j=1:size*size] {
    total_count=total_count+Count[j]
    if (Count[j]>0) {
        bins=bins+1
        Value[bins]=Count[j]
        q=int((j-1)/size)-offset; r=(j-1)%size-offset
        cx=sqrt(3)*radius*(q+r/2.0); cy=1.5*radius*r
        do for [k=0:6] {
            angle=(30+60*k)*pi/180
```

```
            print cx+radius*cos(angle),cy+radius*sin(angle)
        }
        print ""; print ""
    }
}
set print
if (total_count!=N) { exit error "Hexbin count conservation failed" }
stats Value using 2 every ::0::(bins-1) nooutput
set cbrange [0:STATS_max]
set palette defined (0 "#F1F7FB",0.45 "#8EBAE5",1 "#00549F")
set size ratio -1
set xrange [-5:5]; set yrange [-4:4]
set xlabel "x"; set ylabel "y"
set style fill solid 1 border lc rgb "white"
set cblabel "Observations / cell"
plot for [b=1:bins] $Cells index (b-1) using 1:2 with filledcurves closed fc palette cb Value[b] notitle, \
     $D using 1:2 with points pt 7 ps 0.18 lc rgb "#000000" notitle
```



PLT-33-006 · Generated from the complete Bash block

`Axial coordinates / cube correction`	A nearest-cell assignment preserving q+r+z=0; independent rounding of q and r alone fails near corners.
`radius`	Hexagon circumradius in the same units as x and y; ratio -1 preserves regular geometry.
`Value / filledcurves closed`	One aggregation value per closed seven-vertex path; the seventh vertex repeats the first.
`total_count!=N`	An executed conservation check verifies exactly 800 assignments.

Connect two nodes with one flow

New baseline: the commands below define this example.

Start with two equal-height node bars and one ribbon of 100 units. A cubic ease function moves the lower edge smoothly between endpoints; adding the flow amount gives the upper edge without changing thickness.

```
nodes=2; edges=1; scale=1
array X[nodes]=[0,1]
array Y[nodes]=[0,0]
array Name[nodes]=["Supply","Use"]
array S[edges]=[1]; array T[edges]=[2]
array W[edges]=[100]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:1.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```



PLT-34-001 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.

Split a source into three destinations

Compare with PLT-34-001 · Connect two nodes with one flow.

Partition one 100-unit source into 40, 35 and 25 units. Independent source-outgoing and destination-incoming offsets stack each ribbon exactly once; gaps between destination bars do not consume flow.

```
nodes=4; edges=3; scale=1
array X[nodes]=[0,1,1,1]
array Y[nodes]=[0,0,50,95]
array Name[nodes]=["Supply","Homes","Industry","Loss"]
array S[edges]=[1,1,1]; array T[edges]=[2,3,4]
array W[edges]=[40,35,25]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:1.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```



PLT-34-002 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.

Combine multiple sources and destinations

Compare with PLT-34-002 · Split a source into three destinations.

Use a two-by-three flow matrix with row totals 500 and 500 and column totals 400, 300 and 300. Source colour lets a ribbon retain its identity at its destination; crossings are a consequence of the explicitly chosen ordering.

```
nodes=5; edges=6; scale=1
array X[nodes]=[0,0,1,1,1]
array Y[nodes]=[0,600,0,500,900]
array Name[nodes]=["Solar","Wind","Homes","Industry","Storage"]
array S[edges]=[1,1,1,2,2,2]; array T[edges]=[3,4,5,3,4,5]
array W[edges]=[300,120,80,100,180,220]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:1.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```



PLT-34-003 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.

Carry the same flow through a third stage

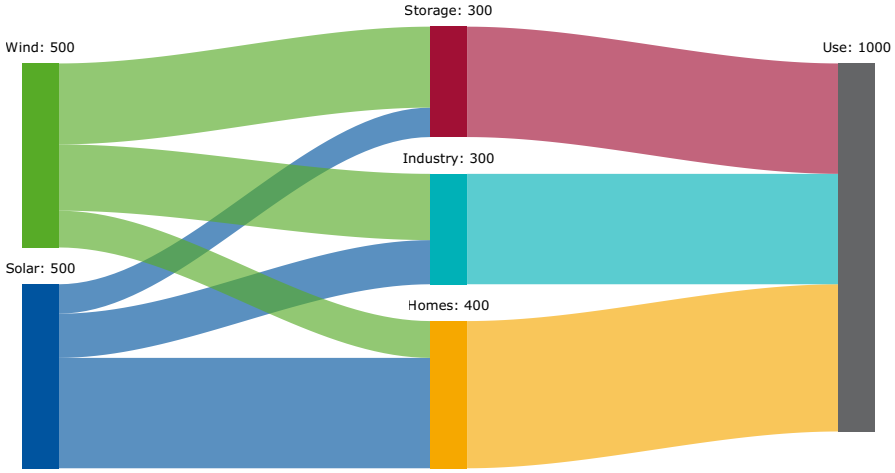
Compare with PLT-34-003 · Combine multiple sources and destinations.

Add a collection node after the three intermediate nodes. Each intermediate has equal inflow and outflow, so its height is reused on both sides. The final node receives all 1000 units; this is one flow through three stages, not 2000 distinct units.

```
nodes=6; edges=9; scale=1
array X[nodes]=[0,0,1,1,1,2]
array Y[nodes]=[0,600,0,500,900,100]
array Name[nodes]=["Solar","Wind","Homes","Industry","Storage","Use"]
array S[edges]=[1,1,1,2,2,2,3,4,5]; array T[edges]=[3,4,5,3,4,5,6,6,6]
array W[edges]=[300,120,80,100,180,220,400,300,300]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:2.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```

PLT-34-004



PLT-34-004 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.

Change node order to inspect crossings

Compare with PLT-34-003 · Combine multiple sources and destinations.

Keep all six amounts fixed while reversing the destination placement. This controlled comparison shows how a manual order changes crossings without changing any totals. It is an explicit layout choice, not an automatic crossing-minimization algorithm.

```
nodes=5; edges=6; scale=1
array X[nodes]=[0,0,1,1,1]
array Y[nodes]=[0,600,800,400,0]
array Name[nodes]=["Solar","Wind","Homes","Industry","Storage"]
array S[edges]=[1,1,1,2,2,2]; array T[edges]=[3,4,5,3,4,5]
array W[edges]=[300,120,80,100,180,220]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:1.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```

PLT-34-005



PLT-34-005 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.

Normalize the flow to a 100-percent budget

Compare with PLT-34-003 · Combine multiple sources and destinations.

Divide every flow by the source total and multiply by 100 before building the same geometry. Node and ribbon labels now show shares of the complete 1000-unit budget; percentages are not separately normalized at each destination.

```
nodes=5; edges=6; scale=0.1
array X[nodes]=[0,0,1,1,1]
array Y[nodes]=[0,600,0,500,900]
array Name[nodes]=["Solar","Wind","Homes","Industry","Storage"]
array S[edges]=[1,1,1,2,2,2]; array T[edges]=[3,4,5,3,4,5]
array W[edges]=[300,120,80,100,180,220]
array C[6]=["#00549F","#57AB27","#F6A800","#00B1B7","#A11035","#646567"]
array In[nodes]; array Out[nodes]; array H[nodes]
array UsedIn[nodes]; array UsedOut[nodes]
do for [v=1:nodes] { In[v]=0; Out[v]=0; UsedIn[v]=0; UsedOut[v]=0; Y[v]=Y[v]*scale }
do for [e=1:edges] { W[e]=W[e]*scale; Out[S[e]]=Out[S[e]]+W[e]; In[T[e]]=In[T[e]]+W[e] }
do for [v=1:nodes] {
    if (In[v]>0 && Out[v]>0 && abs(In[v]-Out[v])>1e-8) { exit error "Unbalanced node" }
    H[v]=In[v]>Out[v]?In[v]:Out[v]
}
ease(t)=t*t*(3-2*t)
bar=0.045
set print $Flow
do for [e=1:edges] {
    a=Y[S[e]]+UsedOut[S[e]]; b=Y[T[e]]+UsedIn[T[e]]
    do for [k=0:40] {
        t=k/40.0; lower=(1-ease(t))*a+ease(t)*b
        print (1-t)*(X[S[e]]+bar)+t*(X[T[e]]-bar),lower,lower+W[e]
    }
    UsedOut[S[e]]=UsedOut[S[e]]+W[e]; UsedIn[T[e]]=UsedIn[T[e]]+W[e]
    print ""; print ""
}
set print
top=0
do for [v=1:nodes] {
    top=(Y[v]+H[v]>top?Y[v]+H[v]:top)
}
do for [v=1:nodes] {
```

```
    set object v rect from X[v]-bar,Y[v] to X[v]+bar,Y[v]+H[v] fc rgb C[v] fs solid 1 noborder front
    set label v sprintf("%s: %.0f%%",Name[v],H[v]) at X[v],Y[v]+H[v]+0.035*top center font ",14" front
}
unset border; unset tics; unset colorbox
set xrange [-0.3:1.3]
set yrange [-0.03*top:1.13*top]
set style fill transparent solid 0.65 noborder
plot for [e=1:edges] $Flow index (e-1) using 1:2:3 with filledcurves lc rgb C[S[e]] notitle
```



PLT-34-006 · Generated from the complete Bash block

`S / T / W`	Source node, target node and nonnegative flow amount; arrays explicitly define the network.
`In / Out / H`	Compute node totals and reject unequal inflow and outflow at intermediate nodes.
`UsedOut / UsedIn`	Separate stacking offsets at each end keep ribbons within their node heights.
`ease(t)`	Cubic interpolation with horizontal endpoint tangents; upper edge is always lower+W[e].
`X / Y / bar`	Authored node columns, vertical offsets and node half-width; gnuplot does not compute this layout.
`scale=0.1`	100/1000 converts every amount and vertical position into percent of the complete source budget.

Compute a bridge from signed contributions

New baseline: the commands below define this example.

Build every rectangle from a signed contribution and the current running balance. The final closing bar is generated from that balance, never typed separately. Ordered low/high bounds preserve decreases as well as increases.

```
N=6
array Delta[N]=[80,25,-12,18,-30,0]
array IsTotal[N]=[1,0,0,0,0,1]
array Name[N]=["Opening","Sales","Returns","Services","Costs","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
}
set print
set style fill solid 0.9 noborder
unset key
set xrange [0.4:N+0.6]
set yrange [-5:130]
set ylabel "Balance (units)"
set grid ytics back lc rgb "#ECEDED"
set xzeroaxis lw 1.5 lc rgb "#646567"
plot $Bars using 1:(($2+$3)/2):($1-0.32):($1+0.32):2:3:xticlabels(5) with
boxxyerrorbars lc rgb "#00549F"
```



PLT-35-001 · Generated from the complete Bash block

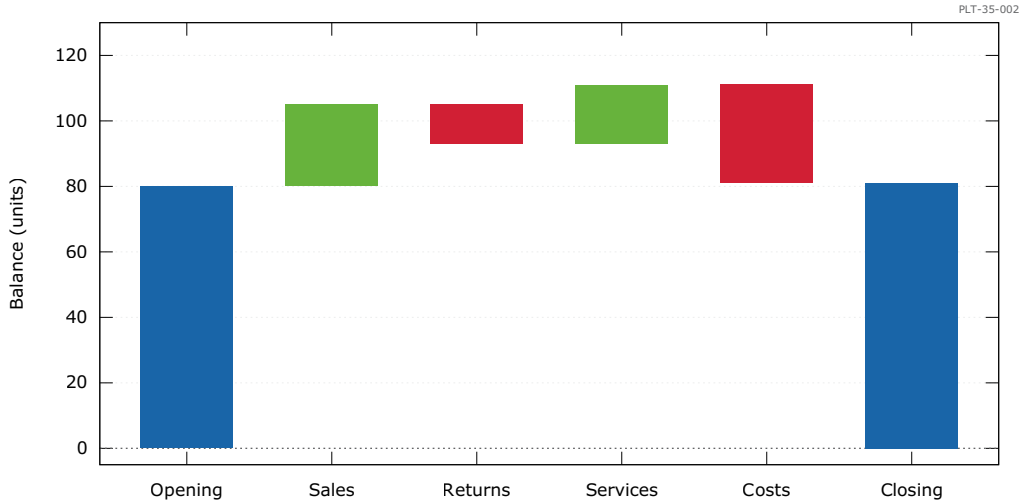
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.

Distinguish increases, decreases and totals

Compare with PLT-35-001 · Compute a bridge from signed contributions.

Keep the same geometry and assign blue to totals, green to positive contributions and red to negative contributions. The direction is read from the signed value and old/new levels, not inferred from the rectangle bounds.

```
N=6
array Delta[N]=[80,25,-12,18,-30,0]
array IsTotal[N]=[1,0,0,0,0,1]
array Name[N]=["Opening","Sales","Returns","Services","Costs","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
}
set print
set style fill solid 0.9 noborder
unset key
set xrange [0.4:N+0.6]
set yrange [-5:130]
set ylabel "Balance (units)"
set grid ytics back lc rgb "#ECEDED"
set xzeroaxis lw 1.5 lc rgb "#646567"
plot $Bars using 1:(($2+$3)/2):($1-0.32):($1+0.32):2:3:4:xticlabels(5) with
boxxyerrorbars lc rgb variable
```



PLT-35-002 · Generated from the complete Bash block

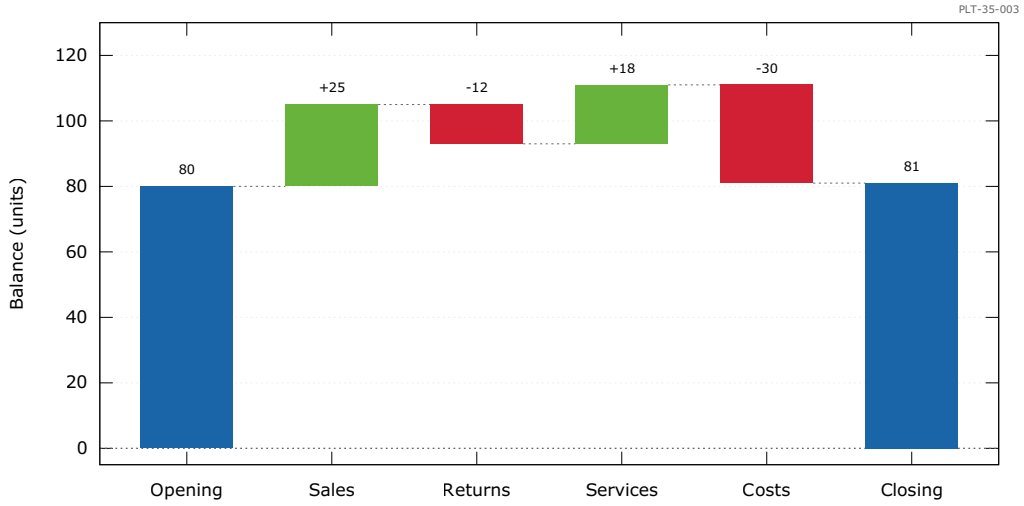
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.

Connect levels and label every contribution

Compare with PLT-35-002 · Distinguish increases, decreases and totals.

Add dashed connectors between successive balances and label changes with an explicit plus or minus. Opening and closing totals have no plus sign. Labels sit above the higher rectangle bound so decreases remain readable.

```
N=6
array Delta[N]=[80,25,-12,18,-30,0]
array IsTotal[N]=[1,0,0,0,0,1]
array Name[N]=["Opening","Sales","Returns","Services","Costs","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
    if (j<N) { set arrow j from j+0.32,running to j+0.68,running nohead dt 2 lc
rgb "#646567" }
}
set print
set style fill solid 0.9 noborder
unset key
set xrange [0.4:N+0.6]
set yrange [-5:130]
set ylabel "Balance (units)"
set grid ytics back lc rgb "#ECEDED"
set xzeroaxis lw 1.5 lc rgb "#646567"
plot $Bars using 1:(($2+$3)/2):($1-0.32):($1+0.32):2:3:4:xticlabels(5) with
boxxyerrorbars lc rgb variable, \
     $Bars using 1:($3+5):6 with labels font ",15"
```



PLT-35-003 · Generated from the complete Bash block

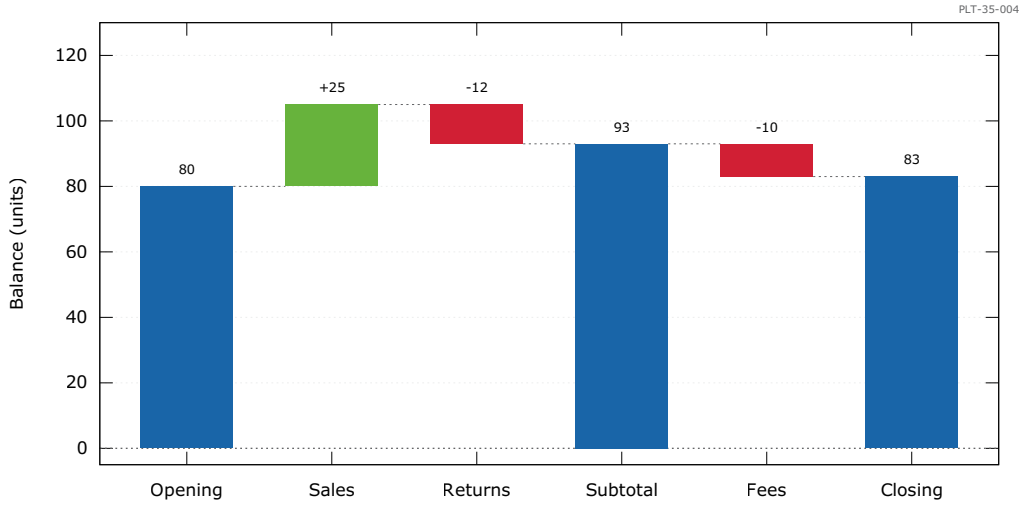
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.
`Dashed connectors`	Use the current running balance between adjacent bar edges; connectors do not add a contribution.

Insert a subtotal without double counting

Compare with PLT-35-003 · Connect levels and label every contribution.

Mark an intermediate row as a total. Its rectangle starts at zero and reaches the current balance, while the accumulator stays unchanged. The closing balance is 83, not 176: the subtotal of 93 is displayed, not added.

```
N=6
array Delta[N]=[80,25,-12,0,-10,0]
array IsTotal[N]=[1,0,0,1,0,1]
array Name[N]=["Opening","Sales","Returns","Subtotal","Fees","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
    if (j<N) { set arrow j from j+0.32,running to j+0.68,running nohead dt 2 lc
rgb "#646567" }
}
set print
set style fill solid 0.9 noborder
unset key
set xrange [0.4:N+0.6]
set yrange [-5:130]
set ylabel "Balance (units)"
set grid ytics back lc rgb "#ECEDED"
set xzeroaxis lw 1.5 lc rgb "#646567"
plot $Bars using 1:(($2+$3)/2):($1-0.32):($1+0.32):2:3:4:xticlabels(5) with
boxxyerrorbars lc rgb variable, \
     $Bars using 1:($3+5):6 with labels font ",15"
```



PLT-35-004 · Generated from the complete Bash block

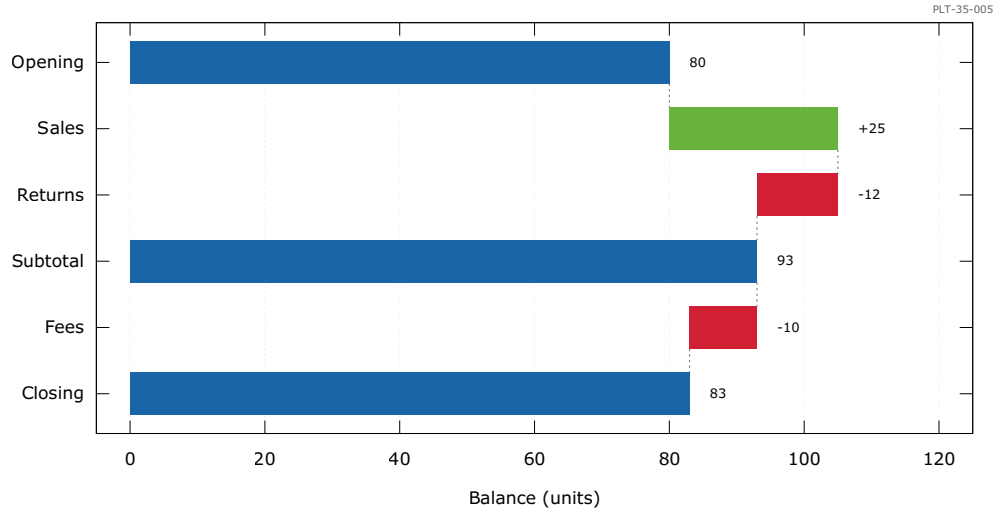
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.
`Dashed connectors`	Use the current running balance between adjacent bar edges; connectors do not add a contribution.

Turn the bridge into horizontal bars

Compare with PLT-35-004 · Insert a subtotal without double counting.

Use the same derived intervals as the subtotal example, swapping their role onto the horizontal axis. Reverse the category-axis range to keep reading order top to bottom; labels become y ticks and units belong on the x axis.

```
N=6
array Delta[N]=[80,25,-12,0,-10,0]
array IsTotal[N]=[1,0,0,1,0,1]
array Name[N]=["Opening","Sales","Returns","Subtotal","Fees","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
    if (j<N) { set arrow j from running,j+0.32 to running,j+0.68 nohead dt 2 lc
rgb "#646567" }
}
set print
set style fill solid 0.9 noborder
unset key
set yrange [N+0.6:0.4]
set xrange [-5:125]
set xlabel "Balance (units)"
set grid xtics back lc rgb "#ECEDED"
plot $Bars using (($2+$3)/2):1:2:3:($1-0.32):($1+0.32):4:yticlabels(5) with
boxxyerrorbars lc rgb variable, \
     $Bars using ($3+3):1:6 with labels left font ",15"
```



PLT-35-005 · Generated from the complete Bash block

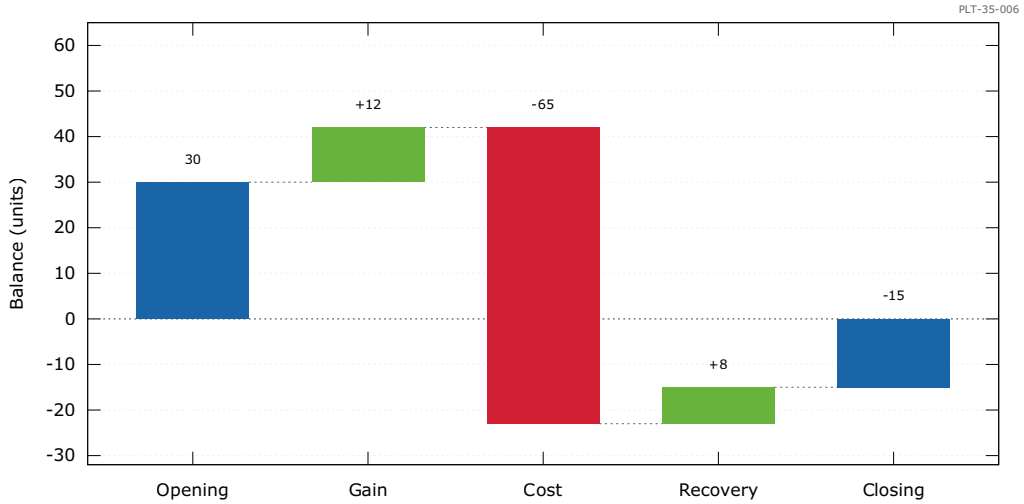
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.
`Dashed connectors`	Use the current running balance between adjacent bar edges; connectors do not add a contribution.
`Reversed yrange`	Preserve top-to-bottom reading order after exchanging the numeric and categorical axes.

Keep negative totals and zero crossings correct

Compare with PLT-35-003 · Connect levels and label every contribution.

Change the input to an opening balance of 30, a gain of 12, a cost of -65 and a recovery of 8. Both a contribution and the closing total extend below zero. Ordered rectangle bounds and a visible zero reference line keep negative balances accurate.

```
N=5
array Delta[N]=[30,12,-65,8,0]
array IsTotal[N]=[1,0,0,0,1]
array Name[N]=["Opening","Gain","Cost","Recovery","Closing"]
running=0
set print $Bars
do for [j=1:N] {
    before=running
    if (j==1 || !IsTotal[j]) { running=running+Delta[j] }
    start=IsTotal[j]?0:before
    low=start<running?start:running; high=start>running?start:running
    colour=IsTotal[j]?0x00549F:(Delta[j]>=0?0x57AB27:0xCC071E)
    text=IsTotal[j]?sprintf("%.0f",running):sprintf("%+.0f",Delta[j])
    print j,low,high,colour,sprintf('"%s"',Name[j]),sprintf('"%s"',text)
    if (j<N) { set arrow j from j+0.32,running to j+0.68,running nohead dt 2 lc
rgb "#646567" }
}
set print
set style fill solid 0.9 noborder
unset key
set xrange [0.4:N+0.6]
set yrange [-32:65]
set ylabel "Balance (units)"
set grid ytics back lc rgb "#ECEDED"
set xzeroaxis lw 1.5 lc rgb "#646567"
plot $Bars using 1:(($2+$3)/2):($1-0.32):($1+0.32):2:3:4:xticlabels(5) with
boxxyerrorbars lc rgb variable, \
     $Bars using 1:($3+5):6 with labels font ",15"
```



PLT-35-006 · Generated from the complete Bash block

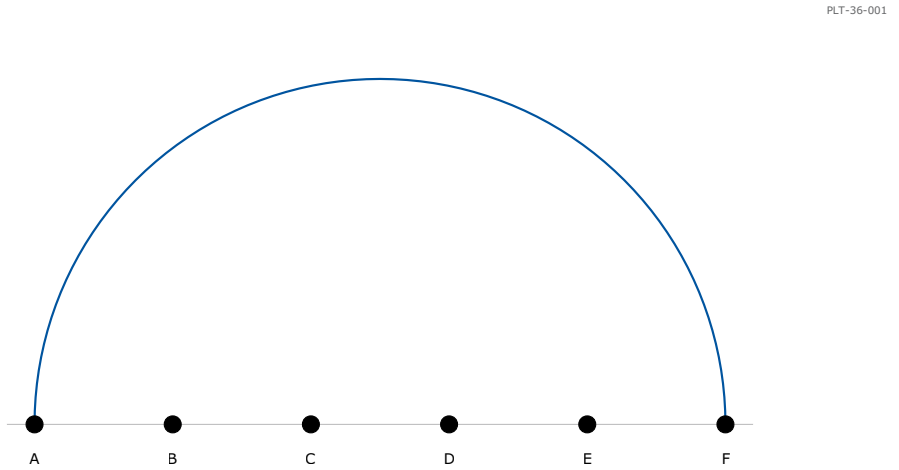
`Delta / IsTotal`	Only contributions and the opening row update the accumulator; other total rows display it.
`start / low / high`	Totals start at zero; contributions span old and new balances. Always order low and high, including negative totals.
`boxxyerrorbars`	Supply the centre and four explicit bounds for each floating rectangle.
`Dashed connectors`	Use the current running balance between adjacent bar edges; connectors do not add a contribution.

Connect two ordered nodes with a semicircle

New baseline: the commands below define this example.

For endpoints a and b, the centre is (a+b)/2 and the radius is abs(b-a)/2. Sample a half-turn from the source to the target; equal coordinate scaling makes the curve a true semicircle.

```
edges=1
array S[edges]=[1]; array T[edges]=[6]
array W[edges]=[3]
array Position[6]=[1,2,3,4,5,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-0.5:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw 2.5 lc rgb
C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```



PLT-36-001 · Generated from the complete Bash block

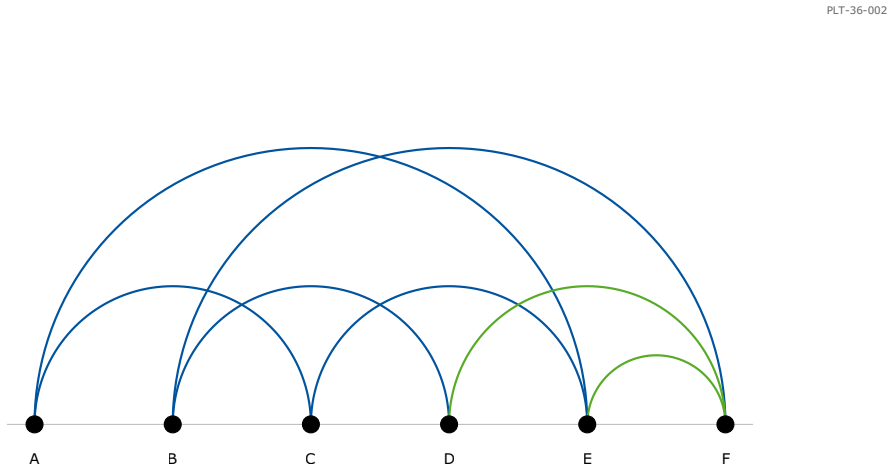
`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.

Display a network as nested arcs

Compare with PLT-36-001 · Connect two ordered nodes with a semicircle.

Add seven relationships among six nodes. Each data block contains one sampled arc; node labels show the categorical order. A crossing or long arc reflects this order and is not a measure of relationship strength.

```
edges=7
array S[edges]=[1,1,2,2,3,4,5]; array T[edges]=[3,5,4,6,5,6,6]
array W[edges]=[3,1,2,5,4,2,1]
array Position[6]=[1,2,3,4,5,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-0.5:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw 2.5 lc rgb
C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```



PLT-36-002 · Generated from the complete Bash block

`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.

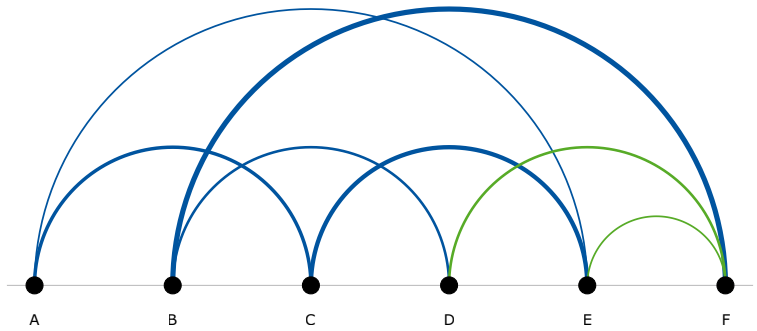
Encode relationship weight with stroke width

Compare with PLT-36-002 · Display a network as nested arcs.

Keep every arc endpoint unchanged and scale its stroke width by the supplied relationship weight. Weights range from one to five; circle radius still represents node separation, so width and radius encode different quantities.

```
edges=7
array S[edges]=[1,1,2,2,3,4,5]; array T[edges]=[3,5,4,6,5,6,6]
array W[edges]=[3,1,2,5,4,2,1]
array Position[6]=[1,2,3,4,5,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-0.5:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw (1+W[e]) lc rgb
C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```

PLT-36-003



PLT-36-003 · Generated from the complete Bash block

`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.
`lw (1+W[e])`	Map weight to stroke width, with a base width to keep the lightest edge visible.

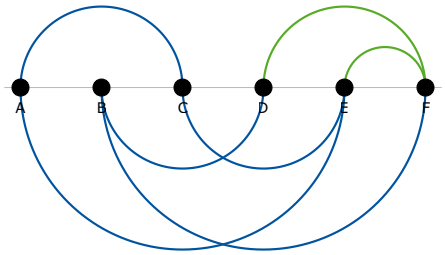
Separate within-group and cross-group links

Compare with PLT-36-002 · Display a network as nested arcs.

Put links inside a community above the baseline and links between communities below it. Colour identifies the source community; the two sides reduce some overlapping strokes while preserving the same edges.

```
edges=7
array S[edges]=[1,1,2,2,3,4,5]; array T[edges]=[3,5,4,6,5,6,6]
array W[edges]=[3,1,2,5,4,2,1]
array Position[6]=[1,2,3,4,5,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=(S[e]<=3)==(T[e]<=3)?1:-1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-2.9:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw 2.5 lc rgb
C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```

PLT-36-004



PLT-36-004 · Generated from the complete Bash block

`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.
`side`	Within-community edges use +1 and cross-community edges -1; source communities select colour.

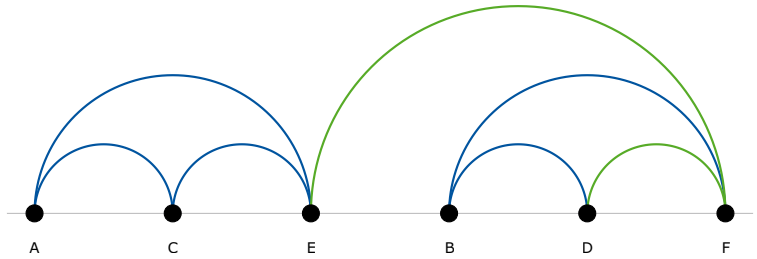
Change the node order explicitly

Compare with PLT-36-002 · Display a network as nested arcs.

Change the position array from the identity order to A,C,E,B,D,F. The data still refer to node IDs, which are converted to positions only during drawing. A supplied permutation is not an automatic layout optimization.

```
edges=7
array S[edges]=[1,1,2,2,3,4,5]; array T[edges]=[3,5,4,6,5,6,6]
array W[edges]=[3,1,2,5,4,2,1]
array Position[6]=[1,4,2,5,3,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-0.5:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw 2.5 lc rgb
C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```

PLT-36-005



PLT-36-005 · Generated from the complete Bash block

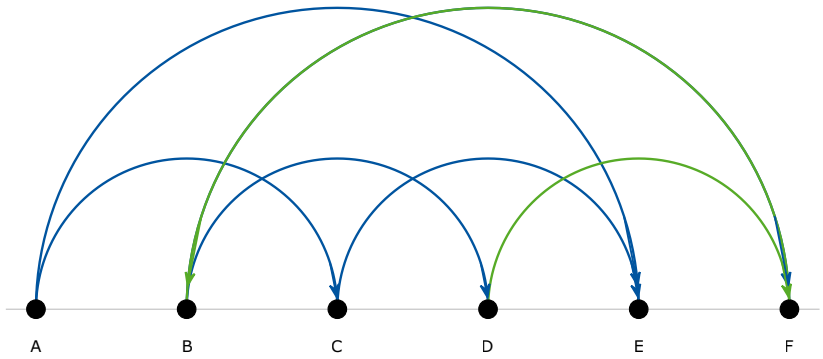
`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.

Add direction at the target endpoints

Compare with PLT-36-002 · Display a network as nested arcs.

Add a short arrow near the end of each sampled arc, using two late sample positions for an approximately tangent segment. Source and target arrays now carry direction, including F to B; the arrow tip approaches the target marker without covering it.

```
edges=7
array S[edges]=[1,1,2,2,3,4,6]; array T[edges]=[3,5,4,6,5,6,2]
array W[edges]=[3,1,2,5,4,2,1]
array Position[6]=[1,2,3,4,5,6]
array Name[6]=["A","B","C","D","E","F"]
array C[2]=["#00549F","#57AB27"]
arcx(t,a,b)=(a+b)/2.0-(b-a)*cos(t)/2
arcy(t,a,b)=abs(b-a)*sin(t)/2.0
set print $Edges
do for [e=1:edges] {
    a=Position[S[e]]; b=Position[T[e]]
    side=1
    do for [k=0:80] {
        t=pi*k/80
        print arcx(t,a,b),side*arcy(t,a,b)
    }
    print ""; print ""
    set arrow e from arcx(0.90*pi,a,b),arcy(0.90*pi,a,b) to arcx(0.975*pi,a,b),arcy(0.975*pi,a,b) head filled size screen 0.012,20,45 lw 2 lc rgb C[S[e]<=3?1:2]
front
}
set print
set print $Nodes
do for [v=1:6] { print Position[v],0,sprintf('"%s"',Name[v]) }
set print
set size ratio -1
set xrange [0.5:6.5]
set yrange [-0.5:2.9]
unset border; unset tics; unset key
set arrow 100 from 0.8,0 to 6.2,0 nohead lc rgb "#B8B8B9" back
plot for [e=1:edges] $Edges index (e-1) using 1:2 with lines lw 2.5 lc rgb C[S[e]<=3?1:2] notitle, \
     $Nodes using 1:2 with points pt 7 ps 1.3 lc rgb "#000000" notitle, \
     $Nodes using 1:($2-0.25):3 with labels font ",17" notitle
```



PLT-36-006 · Generated from the complete Bash block

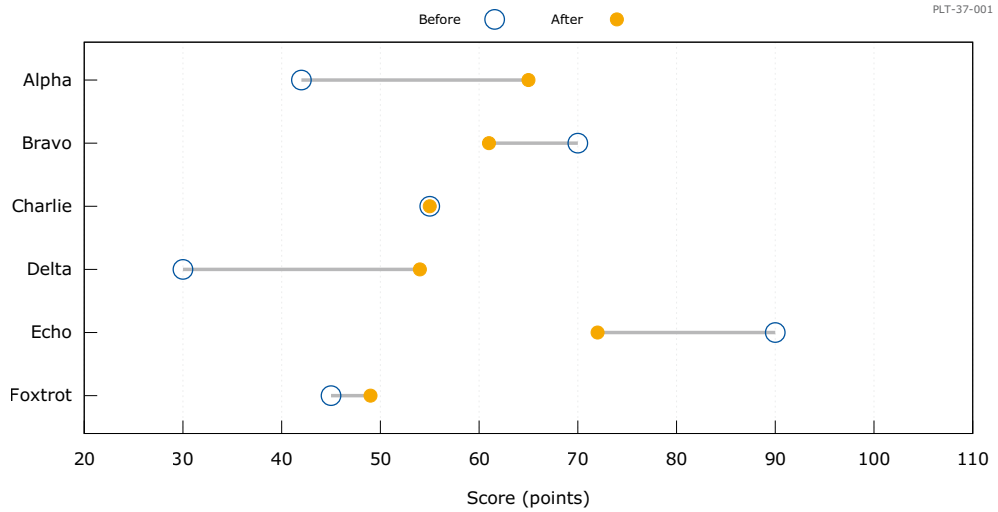
`arcx / arcy`	Sample x and y from a semicircle with the actual source and target as endpoints.
`Position[node]`	Map stable node IDs to categorical horizontal positions; edge weights do not determine radius.
`ratio -1`	Use equal x/y units so semicircles are not distorted into ellipses.
`Blank blocks / index`	Each relationship is a separate curve; node dots and labels are drawn afterwards.
`0.90*pi to 0.975*pi`	A short near-tangent arrow segment indicates the target without drawing a straight source-to-target chord.

Connect before and after values horizontally

New baseline: the commands below define this example.

Connect two measurements of the same category without an arrowhead. An open blue circle marks before and a smaller filled orange circle marks after. When the values coincide, the open ring stays visible around the filled centre; marker style identifies the observations independently of connector direction.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [20:110]; set yrange [N+0.6:0.4]
set xlabel "Score (points)"
set grid xtics back lc rgb "#ECEDED"
plot $D using 2:($0+1):($3-$2):(0):yticlabels(1) with vectors nohead lw 4 lc rgb
"#B8B8B9" notitle, \
     $D using 2:($0+1) with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title
"Before", \
     $D using 3:($0+1) with points pt 7 ps 1.0 lc rgb "#F6A800" title "After"
```



PLT-37-001 · Generated from the complete Bash block

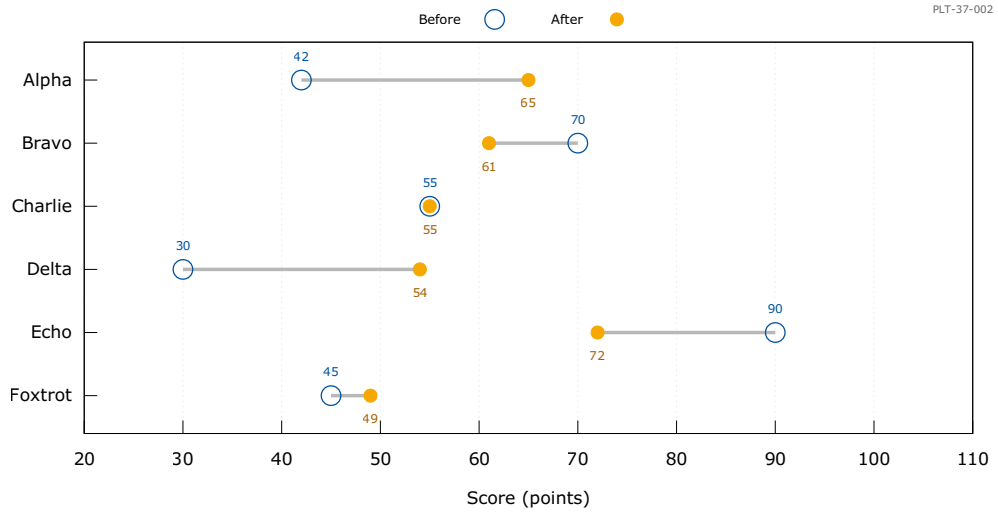
`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`Reversed yrange`	Place the first data row at the top; row positions are categories, not measurements.

Label both endpoints directly

Compare with PLT-37-001 · Connect before and after values horizontally.

Keep the pairs unchanged and write their values beside the corresponding markers. Before labels sit above their row and after labels below it, so equal or nearly equal values do not produce overlapping text. Both endpoint series retain their legend.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [20:110]; set yrange [N+0.6:0.4]
set xlabel "Score (points)"
set grid xtics back lc rgb "#ECEDED"
plot $D using 2:($0+1):($3-$2):(0):yticlabels(1) with vectors nohead lw 4 lc rgb
"#B8B8B9" notitle, \
     $D using 2:($0+1) with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title
"Before", \
     $D using 3:($0+1) with points pt 7 ps 1.0 lc rgb "#F6A800" title "After", \
     $D using 2:($0+1):(sprintf("%g",$2)) with labels offset char 0,0.9 font ",14"
tc rgb "#00549F" notitle, \
     $D using 3:($0+1):(sprintf("%g",$3)) with labels offset char 0,-0.9 font
",14" tc rgb "#A65E00" notitle
```



PLT-37-002 · Generated from the complete Bash block

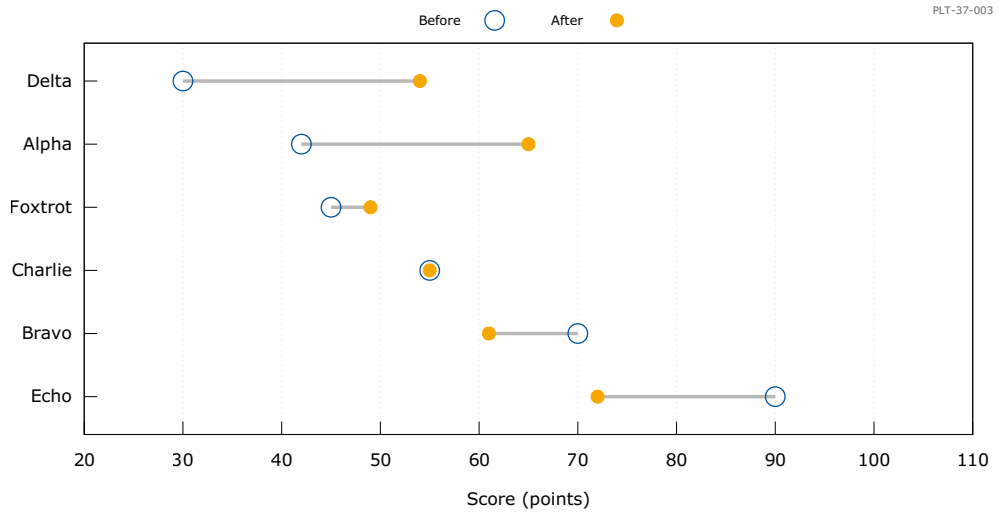
`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`Reversed yrange`	Place the first data row at the top; row positions are categories, not measurements.
`offset char 0,+/-0.9`	Place endpoint labels above and below their row without moving the points.

Sort categories by their signed change

Compare with PLT-37-001 · Connect before and after values horizontally.

Sort row indices by after-minus-before, largest increase first. The original arrays stay intact: reorder complete pairs and their category names together. A stable bubble sort suffices for six rows; equal differences retain their original order.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
do for [pass=1:N-1] {
    do for [k=1:N-pass] {
        a=Order[k]; b=Order[k+1]
        if (After[a]-Before[a]<After[b]-Before[b]) {
            Order[k]=b; Order[k+1]=a
        }
    }
}
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [20:110]; set yrange [N+0.6:0.4]
set xlabel "Score (points)"
set grid xtics back lc rgb "#ECEDED"
plot $D using 2:($0+1):($3-$2):(0):yticlabels(1) with vectors nohead lw 4 lc rgb
"#B8B8B9" notitle, \
     $D using 2:($0+1) with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title
"Before", \
     $D using 3:($0+1) with points pt 7 ps 1.0 lc rgb "#F6A800" title "After"
```



PLT-37-003 · Generated from the complete Bash block

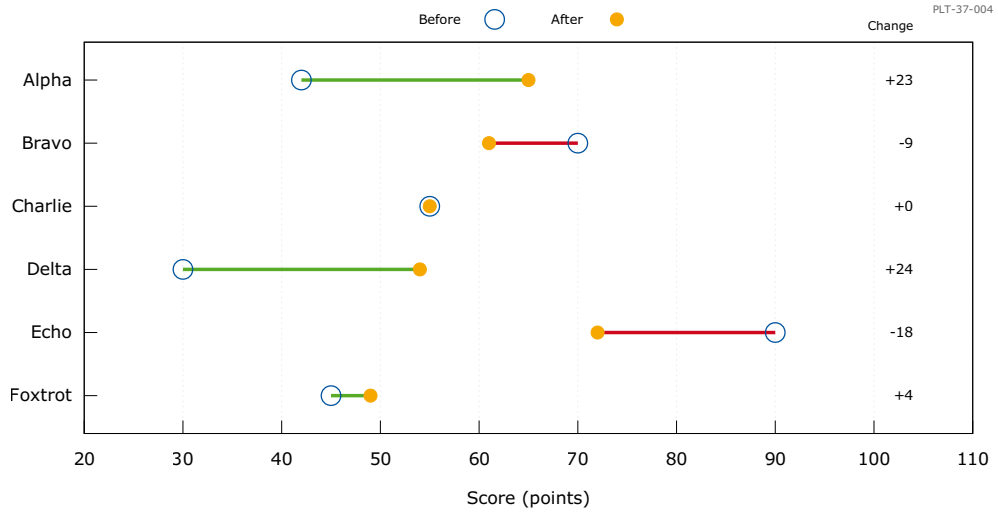
`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`Reversed yrange`	Place the first data row at the top; row positions are categories, not measurements.
`Order / signed difference`	Swap row indices only when the next change is larger; strict comparison preserves ties.

Colour increases, decreases and unchanged pairs

Compare with PLT-37-001 · Connect before and after values horizontally.

Use green connectors for increases, red for decreases and grey for no change. Endpoint markers keep their before/after meanings. A separate signed-change column gives exact differences, including zero; colour does not replace the numeric explanation.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [20:110]; set yrange [N+0.6:0.4]
set xlabel "Score (points)"
set grid xtics back lc rgb "#ECEDED"
set label 1 "Change" at first 104, graph 1.04 right font ",14"
plot $D using 2:($0+1):($3-$2):(0):5:yticlabels(1) with vectors nohead lw 4 lc rgb
variable notitle, \
     $D using 2:($0+1) with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title
"Before", \
     $D using 3:($0+1) with points pt 7 ps 1.0 lc rgb "#F6A800" title "After", \
     $D using (104):($0+1):(sprintf("%+g",$4)) with labels right font ",16" notitle
```



PLT-37-004 · Generated from the complete Bash block

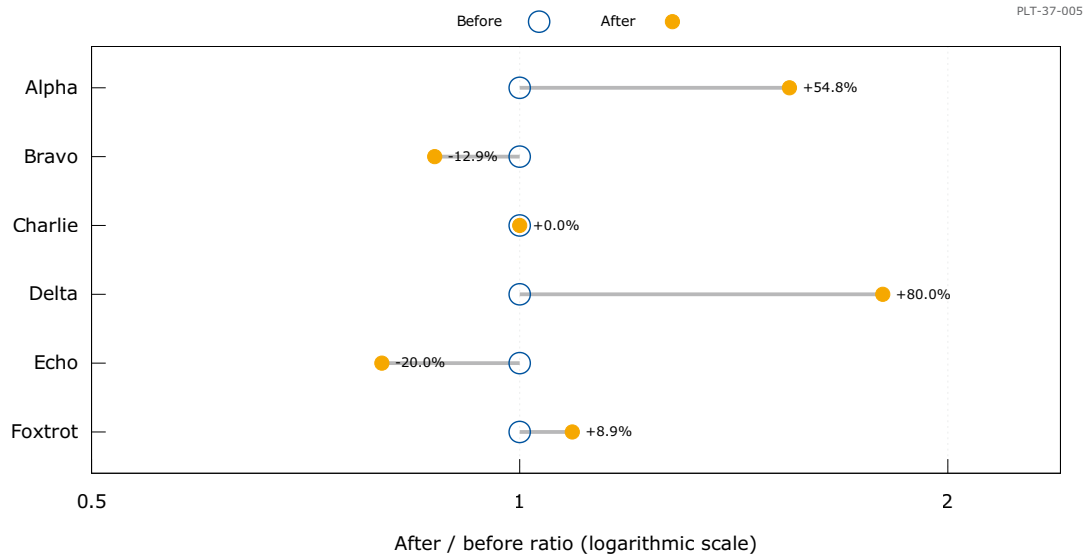
`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`Reversed yrange`	Place the first data row at the top; row positions are categories, not measurements.
`delta / rgb variable`	Connector colour follows the sign; the text column gives the exact difference in points.

Compare relative changes on a logarithmic scale

Compare with PLT-37-001 · Connect before and after values horizontally.

Normalize before to 100 and plot after as 100*after/before. Tick labels display ratios on a base-two log axis: halving and doubling are equally far from one. Labels show percentage change relative to before. Both raw endpoints must be positive. Ratios are not differences in score points.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    if (a<=0 || b<=0) { exit error "Log ratios require positive endpoints" }
    b=100.0*b/a; a=100.0
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [20:110]; set yrange [N+0.6:0.4]
set xlabel "Score (points)"
set grid xtics back lc rgb "#ECEDED"
set logscale x 2
set xrange [50:240]
set xtics ("0.5" 50,"1" 100,"2" 200)
set xlabel "After / before ratio (logarithmic scale)"
plot $D using 2:($0+1):($3-$2):(0):yticlabels(1) with vectors nohead lw 4 lc rgb "#B8B8B9" notitle, \
     $D using 2:($0+1) with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title "Before", \
     $D using 3:($0+1) with points pt 7 ps 1.0 lc rgb "#F6A800" title "After", \
     $D using 3:($0+1):(sprintf("%+.1f%%",$4)) with labels offset char 1,0 left font ",14" notitle
```



PLT-37-005 · Generated from the complete Bash block

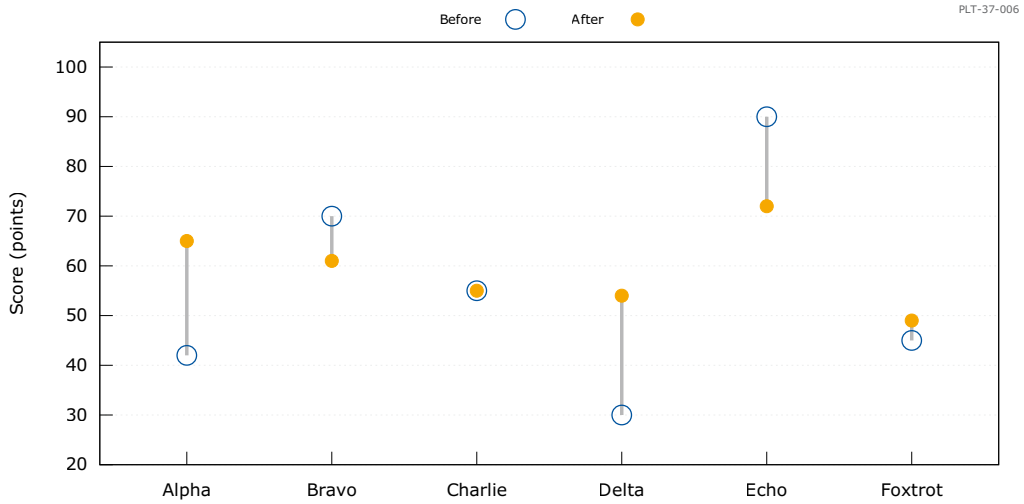
`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`Reversed yrange`	Place the first data row at the top; row positions are categories, not measurements.
`100*after/before / logscale x 2`	Normalize before to 100; percentage change is the normalized after value minus 100.

Turn paired comparisons into vertical dumbbells

Compare with PLT-37-001 · Connect before and after values horizontally.

Exchange the numeric and categorical axes while keeping the same paired data and marker meanings. Each connector is vertical, with zero horizontal displacement. No lines connect different categories, so the result is a paired comparison rather than a time series.

```
N=6
array Name[N]=["Alpha","Bravo","Charlie","Delta","Echo","Foxtrot"]
array Before[N]=[42,70,55,30,90,45]
array After[N]=[65,61,55,54,72,49]
array Order[N]
do for [j=1:N] { Order[j]=j }
set print $D
do for [row=1:N] {
    j=Order[row]; a=Before[j]; b=After[j]
    delta=b-a
    rgb=delta>0?0x57AB27:(delta<0?0xCC071E:0x646567)
    print sprintf('"%s"',Name[j]),a,b,delta,rgb
}
set print
set tics nomirror
set key outside top center horizontal font ",15"
set xrange [0.4:N+0.6]; set yrange [20:105]
set ylabel "Score (points)"
set grid ytics back lc rgb "#ECEDED"
plot $D using ($0+1):2:(0):($3-$2):xticlabels(1) with vectors nohead lw 4 lc rgb
"#B8B8B9" notitle, \
     $D using ($0+1):2 with points pt 6 ps 1.5 lw 2 lc rgb "#00549F" title
"Before", \
     $D using ($0+1):3 with points pt 7 ps 1.0 lc rgb "#F6A800" title "After"
```



PLT-37-006 · Generated from the complete Bash block

`Name / Before / After`	Inline paired scores; each row retains its category at both endpoints.
`with vectors nohead`	Draw only the within-category connector; point layers identify before and after.
`pt 6 / pt 7`	An open ring around a smaller filled centre keeps an unchanged pair identifiable.
`($0+1):2:(0):($3-$2)`	Category position, before value, zero x displacement and signed y displacement.

Keep exploring

This guide explains the commands and arguments used in its examples; it is not an exhaustive language manual. Data are deliberately small and synthetic. Figures are generated by gnuplot itself.

See the build provenance for the gnuplot version used to generate this edition. Sectors require gnuplot 6; 3D filledcurves requires 6.0.2 or later. Default styles, patterns and text metrics can vary by version, terminal and installed fonts.

Primary references

- Official gnuplot 6 manual: commands, plot styles, expressions and macros.
- Official demonstration gallery: broader feature catalogue.
- Hexagonal grids: axial coordinates and cube rounding.
- DKW inequality: simultaneous bounds for the empirical distribution.
- Gallery licence and coverage review: upstream terms and six original step-by-step additions.
- Official stats demonstration: statistics used to annotate a plot.
- Official grouping-mark demonstration: adjustable group brackets using newer marks syntax; chapter 19 uses vector segments on gnuplot 6.0.5.
- stats command reference: column summaries and named statistics.
- Official spiderplot documentation: spoke axes, row-based polygons and legends.
- Spiderplot style settings: fill, line and point options.
- Official heatmap demonstration: matrix and coordinate-grid input.
- PCA documentation: component scores and explained variance, calculated before plotting.
- Official sectors demo: annular-sector field conventions.
- gnuplot 6.0.2 release notes: minimum version for filled 3D waterfalls.
- Official signal-waterfall demo: drawing order in 3D.
- Org Babel shell documentation: running and tangling Bash blocks.

Files and reproducibility

The PDF includes the Org source and ID index as attachments. The project ZIP includes Org, generated Bash scripts, SVG figures and both editions. Each build records source hashes and gnuplot diagnostics.

The HTML embeds its figures and source downloads and works offline. Download the rendered PDF from the opening page for offline reading.

Layout inspiration: the compact code-and-plot approach in Lee Phillips's gnuplot 5 reference. This guide's examples and explanations are original.

Example and command index

- PLT-01-001 · Start with one command
- PLT-01-002 · Change the function
- PLT-01-003 · Add a second curve
- PLT-01-004 · Name the curves
- PLT-01-005 · Hide the legend
- PLT-01-006 · Choose the horizontal interval
- PLT-01-007 · Leave room above and below
- PLT-01-008 · See coarse sampling of a function
- PLT-01-009 · Increase samples without changing the function
- PLT-01-010 · Break a line at an excluded interval
- PLT-02-001 · Use variables for repeated numbers
- PLT-02-002 · Change one variable
- PLT-02-003 · Name the expression as a function
- PLT-02-004 · Use a string variable directly
- PLT-02-005 · Avoid accidental integer division
- PLT-02-006 · Calculate and combine functions
- PLT-02-007 · Scale and shift a function with arithmetic
- PLT-02-008 · Calculate the secant through two points
- PLT-02-009 · Calculate a tangent from the known derivative
- PLT-02-010 · Move the tangent by changing one variable
- PLT-02-011 · Estimate a tangent with a numerical derivative
- PLT-02-012 · Compare numerical derivative step sizes
- PLT-03-001 · Read columns and calculate plotted values
- PLT-03-002 · Read CSV headers and use them in the legend
- PLT-03-003 · Compare missing values with invalid values
- PLT-03-004 · Filter observations with a column expression
- PLT-03-005 · Select every second data row
- PLT-03-006 · Select datasets separated by blank records
- PLT-03-007 · Read dates and format their tick labels
- PLT-03-008 · Place ticks at calendar-month intervals
- PLT-03-009 · Read timestamps and show hourly ticks
- PLT-04-001 · Prepare a simple tick example
- PLT-04-002 · Place a tick every two units
- PLT-04-003 · Halve the tick interval
- PLT-04-004 · Use half-unit ticks
- PLT-04-005 · Set start, increment and end
- PLT-04-006 · Choose irregular tick positions
- PLT-04-007 · Change y ticks independently
- PLT-04-008 · Remove x ticks
- PLT-04-009 · Remove y ticks
- PLT-04-010 · Keep marks but hide their labels
- PLT-05-001 · Observe automatic range rounding
- PLT-05-002 · Keep automatic ranges at the exact extrema
- PLT-05-003 · Pad the data range with offsets
- PLT-05-004 · Let the upper x limit be automatic
- PLT-05-005 · Reverse the horizontal direction
- PLT-05-006 · Reverse the vertical direction
- PLT-05-007 · Begin with a positive function
- PLT-05-008 · Use a base-10 logarithmic y axis
- PLT-05-009 · Use a base-2 logarithmic y axis
- PLT-05-010 · Format logarithmic ticks as powers
- PLT-05-011 · Link a second axis in different units
- PLT-05-012 · Plot a separate quantity on the right axis
- PLT-06-001 · Replace numbers with words
- PLT-06-002 · Add one labelled tick
- PLT-06-003 · Rotate tick labels
- PLT-06-004 · Stand tick labels vertically
- PLT-06-005 · Left-align horizontal tick labels
- PLT-06-006 · Offset tick labels
- PLT-06-007 · Increase tick-label size

- PLT-06-008 · Colour tick-label text
- PLT-06-009 · Use a different tick-label font
- PLT-06-010 · Keep labels and hide tick strokes
- PLT-06-011 · Print one decimal place
- PLT-06-012 · Format values as percentages
- PLT-06-013 · Use scientific tick notation
- PLT-06-014 · Use engineering prefixes on ticks
- PLT-06-015 · Label an angular axis in multiples of pi
- PLT-07-001 · Remove mirrored ticks
- PLT-07-002 · Point ticks outward
- PLT-07-003 · Lengthen major ticks
- PLT-07-004 · Give the two axes different tick lengths
- PLT-07-005 · Add a minor tick halfway
- PLT-07-006 · Add four minor ticks
- PLT-07-007 · Set major and minor lengths
- PLT-07-008 · Subdivide the vertical axis
- PLT-07-009 · Show minor ticks on a logarithmic axis
- PLT-07-010 · Add a major grid
- PLT-07-011 · Show a lighter minor grid
- PLT-07-012 · Use dashed grid lines
- PLT-07-013 · Keep only horizontal grid lines
- PLT-07-014 · Draw the grid at the back
- PLT-07-015 · Draw the grid at the front
- PLT-08-001 · Name the x axis
- PLT-08-002 · Name the y axis
- PLT-08-003 · Move the x label downward
- PLT-08-004 · Make the y label horizontal
- PLT-08-005 · Use subscripts and superscripts
- PLT-08-006 · Add a figure title
- PLT-08-007 · Increase the title font
- PLT-08-008 · Write a two-line title
- PLT-09-001 · Increase line width
- PLT-09-002 · Choose an explicit line colour
- PLT-09-003 · Use a dashed line
- PLT-09-004 · Plot sampled points
- PLT-09-005 · Choose the point symbol
- PLT-09-006 · Increase point size
- PLT-09-007 · Connect the points
- PLT-09-008 · Show a symbol at every tenth sample
- PLT-09-009 · Limit the total number of point symbols
- PLT-09-010 · Fill down to zero
- PLT-09-011 · Make the fill translucent
- PLT-09-012 · Fill between two curves
- PLT-09-013 · Set different defaults for data and functions
- PLT-09-014 · Quick reference: line types
- PLT-09-015 · Quick reference: line widths
- PLT-09-016 · Quick reference: dash patterns
- PLT-09-017 · Quick reference: point symbols
- PLT-09-018 · Quick reference: point sizes
- PLT-09-019 · Quick reference: fill patterns
- PLT-09-020 · Quick reference: lines, steps and impulses
- PLT-10-001 · Move the legend to the lower left
- PLT-10-002 · Move the legend to the upper left
- PLT-10-003 · Move the legend outside
- PLT-10-004 · Place a horizontal legend below
- PLT-10-005 · Draw a legend box
- PLT-10-006 · Give the legend an opaque background
- PLT-10-007 · Give the legend a title
- PLT-10-008 · Anchor the legend with graph coordinates
- PLT-10-009 · Enlarge legend text
- PLT-10-010 · Lengthen legend samples
- PLT-10-011 · Increase spacing between entries
- PLT-10-012 · Arrange the legend in two columns
- PLT-10-013 · Reverse legend entry order
- PLT-10-014 · Reverse sample placement without reordering entries
- PLT-11-001 · Keep only left and bottom borders
- PLT-11-002 · Restore all four borders
- PLT-11-003 · Keep just the bottom border
- PLT-11-004 · Put the frame and ticks at the top and right
- PLT-11-005 · Thicken the remaining borders
- PLT-11-006 · Colour the border and tick strokes
- PLT-11-007 · Use a dashed border

- PLT-11-008 · Shorten borders to the data extent
- PLT-11-009 · Draw the zero axes
- PLT-11-010 · Put ticks on the zero axes
- PLT-11-011 · Remove the outer frame
- PLT-11-012 · Style the horizontal zero reference
- PLT-12-001 · Compare five coordinate systems
- PLT-12-002 · Change ranges while retaining annotation anchors
- PLT-12-003 · Place text at a data point
- PLT-12-004 · Offset a label from its point
- PLT-12-005 · Anchor text to the plot region
- PLT-12-006 · Centre text on its anchor
- PLT-12-007 · Give a label an opaque text box
- PLT-12-008 · Rotate an annotation along a trend
- PLT-12-009 · Draw an arrow between points
- PLT-12-010 · Control arrowhead size and angles
- PLT-12-011 · Quick reference: arrowheads
- PLT-12-012 · Clip a segment whose endpoints are both outside
- PLT-12-013 · Highlight an x interval
- PLT-12-014 · Compare object layers behind and in front
- PLT-13-001 · Change the canvas shape
- PLT-13-002 · Change the overall font size
- PLT-13-003 · Make the plot region square
- PLT-13-004 · Set an explicit plot aspect ratio
- PLT-13-005 · Compare geometry on automatic axes
- PLT-13-006 · Give x and y equal physical units
- PLT-13-007 · Reserve more left margin
- PLT-13-008 · Reserve more bottom margin
- PLT-13-009 · Place the plot borders explicitly
- PLT-13-010 · Reserve space above the plot
- PLT-13-011 · Compare character margins with screen margins
- PLT-14-001 · Distinguish line types from user line styles
- PLT-14-002 · Replace a style fragment with a macro
- PLT-14-003 · Change the macro in one place
- PLT-14-004 · Reuse one style for two curves
- PLT-14-005 · Define and reuse a numbered line style
- PLT-14-006 · Override one property of a reusable style
- PLT-14-007 · Share geometry but vary colour
- PLT-14-008 · Put a range into a macro
- PLT-14-009 · Bundle several setup commands
- PLT-14-010 · Build a macro with string concatenation
- PLT-14-011 · Format a numeric macro parameter
- PLT-14-012 · See where a macro does not expand
- PLT-14-013 · Pass the title variable correctly
- PLT-14-014 · Execute constructed text with eval
- PLT-14-015 · Combine a plot loop with a style macro
- PLT-14-016 · Build macro text with a function parameter
- PLT-14-017 · Pass parameters to a reusable command block
- PLT-14-018 · Reset plot settings within a session
- PLT-15-001 · Plot measurements by record number
- PLT-15-002 · Calculate and draw the mean
- PLT-15-003 · Locate the minimum by its record index
- PLT-15-004 · Locate the maximum as well
- PLT-15-005 · Use the actual x positions
- PLT-15-006 · Transform x consistently for statistics and plotting
- PLT-15-007 · Draw the linear trend from two-column statistics
- PLT-15-008 · Show the computed summary beside the plot
- PLT-15-009 · Move the maximum by changing only the data
- PLT-15-010 · Move the minimum without editing annotations
- PLT-15-011 · Begin with observations for an exponential fit
- PLT-15-012 · Overlay an initial decay model
- PLT-15-013 · Let fit estimate the decay parameters
- PLT-15-014 · Display fitted values and their uncertainties
- PLT-15-015 · Inspect residuals after fitting
- PLT-15-016 · Start an interpolation with six samples
- PLT-15-017 · Connect samples with straight lines
- PLT-15-018 · Replace joins with a cubic spline
- PLT-15-019 · Preserve monotonicity with mcsplines
- PLT-15-020 · Compare interpolation choices in one figure
- PLT-16-001 · Create two side-by-side panels
- PLT-16-002 · Stack the same panels vertically
- PLT-16-003 · Use a two-by-two layout
- PLT-16-004 · Fill panels down columns first

- PLT-16-005 · Specify shared outer margins
- PLT-16-006 · Increase the gap between columns
- PLT-16-007 · Reduce the gap again
- PLT-16-008 · Add a shared figure title
- PLT-16-009 · Share an x scale across stacked panels
- PLT-16-010 · Create an inset manually
- PLT-16-011 · See settings persist between panels
- PLT-16-012 · Clear a setting before the next panel
- PLT-17-001 · Write an SVG with explicit terminal settings
- PLT-17-002 · Export the same plot as a PNG image
- PLT-17-003 · Export a PDF at a physical page size
- PLT-17-004 · Export a PNG with a transparent background
- PLT-18-001 · Start with three bars
- PLT-18-002 · Set an exact bar width
- PLT-18-003 · Fill the bars
- PLT-18-004 · Use category names
- PLT-18-005 · Write values above the bars
- PLT-18-006 · Colour each category
- PLT-18-007 · Put two series side by side
- PLT-18-008 · Widen the gap between groups
- PLT-18-009 · Use hatching for the series
- PLT-18-010 · Stack the series by row
- PLT-18-011 · Normalize every stack to 100 percent
- PLT-18-012 · Turn bars horizontally
- PLT-19-001 · Start with labelled bars
- PLT-19-002 · Leave one empty position between groups
- PLT-19-003 · Colour bars by group
- PLT-19-004 · Derive group labels and widths from the data
- PLT-19-005 · Scale a bracket to each group
- PLT-19-006 · Add a centred stem to the reusable shape
- PLT-19-007 · Widen a group by adding only a row
- PLT-19-008 · Detect an extra group and a new maximum
- PLT-20-001 · Bin raw observations into counts
- PLT-20-002 · Halve the bin width
- PLT-20-003 · Leave small gaps between bins
- PLT-20-004 · Shift the bin boundaries
- PLT-20-005 · Show probability per bin
- PLT-20-006 · Make the histogram a density
- PLT-20-007 · Display bin shares as percentages
- PLT-20-008 · Show the empirical cumulative distribution
- PLT-21-001 · Add symmetric error bars
- PLT-21-002 · Connect the uncertain measurements
- PLT-21-003 · Use asymmetric bounds
- PLT-21-004 · Show uncertainty as a ribbon
- PLT-21-005 · Summarize groups with boxplots
- PLT-21-006 · Change the boxplot whisker rule
- PLT-21-007 · Encode a third quantity as bubble area
- PLT-21-008 · Draw vectors from component data
- PLT-21-009 · Begin a price chart with closing values
- PLT-21-010 · Show open, close, low and high as candlesticks
- PLT-21-011 · Narrow the candlestick bodies
- PLT-21-012 · Colour rising and falling sessions from the data
- PLT-21-013 · Annotate the automatically detected high and low
- PLT-21-014 · See why repeated points hide observations
- PLT-21-015 · Separate overlapping observations with jitter
- PLT-21-016 · Increase the swarm separation
- PLT-21-017 · Limit horizontal displacement with wrap
- PLT-21-018 · Overlay automatically computed group means
- PLT-22-001 · Plot prepared PCA scores
- PLT-22-002 · Distinguish PCA groups by colour and symbol
- PLT-22-003 · Label selected PCA samples
- PLT-22-004 · Draw PCA group-spread ellipses
- PLT-22-005 · Colour PCA points by a continuous measurement
- PLT-22-006 · Overlay variable arrows in a PCA biplot
- PLT-22-007 · Connect multiple measurements with parallel axes
- PLT-22-008 · Fix meaningful ranges for each parallel axis
- PLT-22-009 · Colour parallel lines by design group
- PLT-22-010 · Reverse the cost axis to align preferences
- PLT-22-011 · Align all axes so higher means preferable
- PLT-23-001 · Start with a simple colour map
- PLT-23-002 · Choose a diverging palette
- PLT-23-003 · Expand the colour range

- PLT-23-004 · Set colourbar tick spacing
- PLT-23-005 · Label the colourbar
- PLT-23-006 · Hide the colourbar
- PLT-23-007 · Place a horizontal colourbar
- PLT-23-008 · Reduce a palette to discrete colour bands
- PLT-23-009 · Reverse the palette direction
- PLT-23-010 · Use a logarithmic colour scale
- PLT-23-011 · Colour points using a third data column
- PLT-23-012 · Share one colourbar across two panels
- PLT-24-001 · Draw a heatmap from a matrix
- PLT-24-002 · Put the first matrix row at the top
- PLT-24-003 · Label heatmap rows and columns
- PLT-24-004 · Print the value inside every cell
- PLT-24-005 · Draw lines between heatmap cells
- PLT-24-006 · Use a symmetric palette for signed values
- PLT-24-007 · Leave missing heatmap cells blank
- PLT-24-008 · Read a heatmap as x, y and value triples
- PLT-25-001 · Draw a wireframe surface
- PLT-25-002 · Remove hidden wireframe lines
- PLT-25-003 · Colour the surface height
- PLT-25-004 · Change the surface viewing angle
- PLT-25-005 · Compare coarse and fine surface sampling
- PLT-25-006 · Project contour lines into a map
- PLT-25-007 · Choose only specific contour levels
- PLT-25-008 · Draw a polar curve
- PLT-25-009 · Add a polar grid
- PLT-26-001 · Start a native spider plot
- PLT-26-002 · Add the polygonal spiderweb
- PLT-26-003 · Change the number of web rings
- PLT-26-004 · Show numeric notation on every spoke
- PLT-26-005 · Give each spoke a meaningful name
- PLT-26-006 · Rotate individual spoke labels
- PLT-26-007 · Move spoke names radially outward
- PLT-26-008 · Style the spoke-name font and colour
- PLT-26-009 · Use multiline and literal spoke names
- PLT-26-010 · Format a spoke scale as percentages
- PLT-26-011 · Use words for radial tick levels
- PLT-26-012 · Change numeric tick text and offset
- PLT-26-013 · Rotate numeric tick labels explicitly
- PLT-26-014 · Give spokes different units and ranges
- PLT-26-015 · Reverse a lower-is-better spoke
- PLT-26-016 · Choose which variable starts at the top
- PLT-26-017 · Fill the polygon transparently
- PLT-26-018 · Remove the polygon border
- PLT-26-019 · Mark every polygon vertex
- PLT-26-020 · Choose a dashed polygon outline
- PLT-26-021 · Style the radial spoke lines
- PLT-26-022 · Style and layer the web grid
- PLT-26-023 · Compare several rows and name the series
- PLT-26-024 · Combine separate datasets with newspiderplot
- PLT-27-001 · Draw a single sector
- PLT-27-002 · Complete a three-slice pie
- PLT-27-003 · Assign distinct slice colours
- PLT-27-004 · Separate slices with white borders
- PLT-27-005 · Label the percentages
- PLT-27-006 · Open a donut hole
- PLT-27-007 · Make the donut ring thinner
- PLT-27-008 · Reverse the angular direction
- PLT-27-009 · Add a central total
- PLT-28-001 · Start with one floating rectangle
- PLT-28-002 · Arrange a sequence of intervals
- PLT-28-003 · Derive intervals from signed changes
- PLT-28-004 · Colour totals, gains and losses
- PLT-28-005 · Connect the running levels
- PLT-28-006 · Print the signed changes
- PLT-28-007 · Narrow the waterfall bars
- PLT-28-008 · Draw a small schedule from task intervals
- PLT-28-009 · Replace schedule lines with interval boxes
- PLT-28-010 · Show calendar dates on the schedule
- PLT-28-011 · Overlay the completed fraction of each task
- PLT-28-012 · Adapt the schedule range when a task is added
- PLT-29-001 · Draw a family of related curves

- PLT-29-002 · Separate curves with a vertical offset
- PLT-29-003 · Increase the separation
- PLT-29-004 · Fill offset curves back to front
- PLT-29-005 · Move the signal waterfall into 3D
- PLT-29-006 · Fill the 3D waterfall slices
- PLT-30-001 · Define a reusable style file
- PLT-30-002 · Load a project or personal style
- PLT-30-003 · Load RWTH–Tufte for a line plot
- PLT-30-004 · Use the library’s named colours
- PLT-30-005 · Override a style for one plot
- PLT-31-001 · Mirror one empirical density
- PLT-31-002 · Compare two groups on a common scale
- PLT-31-003 · Expose the effect of a smaller bandwidth
- PLT-31-004 · Overlay observations and quartiles
- PLT-31-005 · Split two groups across one violin
- PLT-31-006 · Turn the density sideways with native kdensity
- PLT-32-001 · Draw a right-continuous ECDF with ties
- PLT-32-002 · Compare samples of different sizes
- PLT-32-003 · Accumulate observation weights
- PLT-32-004 · Plot the fraction strictly above a threshold
- PLT-32-005 · Read the empirical median and a service threshold
- PLT-32-006 · Add a simultaneous DKW confidence band
- PLT-33-001 · Count observations in true hexagonal bins
- PLT-33-002 · Trade detail for larger bins
- PLT-33-003 · Reveal sparse cells with logarithmic colour
- PLT-33-004 · Normalize counts to probability density
- PLT-33-005 · Map a measurement mean instead of frequency
- PLT-33-006 · Overlay the original observations
- PLT-34-001 · Connect two nodes with one flow
- PLT-34-002 · Split a source into three destinations
- PLT-34-003 · Combine multiple sources and destinations
- PLT-34-004 · Carry the same flow through a third stage
- PLT-34-005 · Change node order to inspect crossings
- PLT-34-006 · Normalize the flow to a 100-percent budget
- PLT-35-001 · Compute a bridge from signed contributions
- PLT-35-002 · Distinguish increases, decreases and totals
- PLT-35-003 · Connect levels and label every contribution
- PLT-35-004 · Insert a subtotal without double counting
- PLT-35-005 · Turn the bridge into horizontal bars
- PLT-35-006 · Keep negative totals and zero crossings correct
- PLT-36-001 · Connect two ordered nodes with a semicircle
- PLT-36-002 · Display a network as nested arcs
- PLT-36-003 · Encode relationship weight with stroke width
- PLT-36-004 · Separate within-group and cross-group links
- PLT-36-005 · Change the node order explicitly
- PLT-36-006 · Add direction at the target endpoints
- PLT-37-001 · Connect before and after values horizontally
- PLT-37-002 · Label both endpoints directly
- PLT-37-003 · Sort categories by their signed change
- PLT-37-004 · Colour increases, decreases and unchanged pairs
- PLT-37-005 · Compare relative changes on a logarithmic scale
- PLT-37-006 · Turn paired comparisons into vertical dumbbells

Content: CC0 1.0. Structure and publishing framework: CC BY 4.0 by Lukas C. Bossert. Repository source · Publishing framework source · License scope and third-party notices.