knitr Graphics Manual

Yihui Xie

April 6, 2012

This manual shows features of graphics in the **knitr** package (version 0.4.4) in detail, including the graphical devices, plot recording, plot rearrangement, control of plot sizes, the tikz device, figure captions, animations and other types of plots such as **rgl** or GGobi plots.

Before reading this specific manual[1], you must have finished the main manual[2].

[1] `http://bit.ly/knitr-graphics-src` (Rnw source)

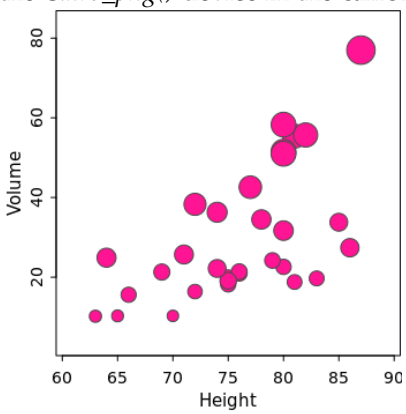
[2] `http://bit.ly/knitr-main-pdf`

Graphical Devices

The **knitr** package comes with more than 20 built-in graphical devices, and you can specify them through the `dev` option. This document uses the global option `dev='tikz'`, i.e., the plots are recorded by the tikz device by default, but we can change the device locally. Since tikz will be used extensively throughout this manual and you will see plenty of tikz graphics later, now we first show a few other devices.

```
with(trees, symbols(Height, Volume, circles = Girth/16,
    inches = FALSE, bg = "deeppink", fg = "gray30"))
```

Figure 1 and 2 show two standard devices in the **grDevices** package. We can also use devices in the **Cairo** or **cairoDevice** package, e.g., the chunk below uses the *Cairo_png()* device in the **cairoDevice** package.



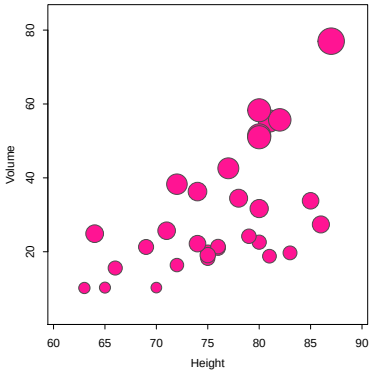


Figure 1: The default PDF device.

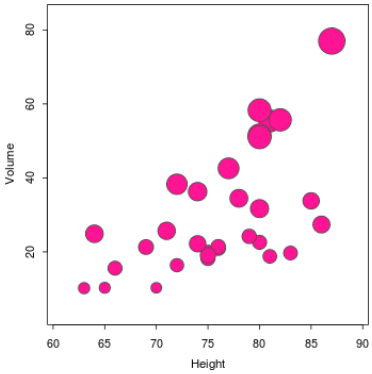


Figure 2: The PNG device.

Plot Recording

As mentioned in the main manual, **knitr** uses the **evaluate** package to record plots. There are two sources of plots: first, whenever *plot.new()* or *grid.newpage()* is called, **evaluate** will try to save a snapshot of the

current plot[3]; second, after each complete expression is evaluated, a snapshot is also saved. To speed up recording, the null graphical device `pdf(file = NULL)` is used. Figure 3 shows two expressions producing two high-level plots.

[3] For technical details, see `?setHook` and `?recordPlot`

```
plot(cars)
boxplot(cars$dist, xlab = "dist")
```

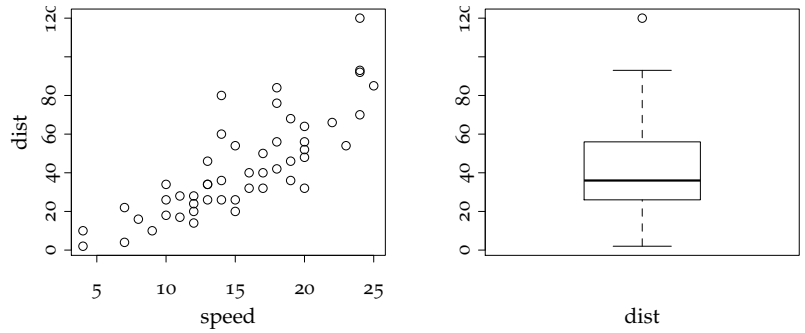


Figure 3: Two high-level plots are captured. The key to arrange two plots side by side is to specify the `out.width` option so that each plot takes less than half of the line width. We do not have to use the `par(mfrow)` trick, and it may not work in some cases (e.g. to put base graphics and **ggplot2** side by side; recall Figure 1 in the main manual).

Figure 4 shows another example of two R expressions, but the second expression only involves with low-level plotting changes. By default, low-level plot changes are discarded, but you can retain them with the option `fig.keep='all'`.

```
plot(0, 0, type = "n", ann = FALSE)
for (i in seq(0, 2 * pi, length = 20)) {
    points(cos(i), sin(i))
}
```

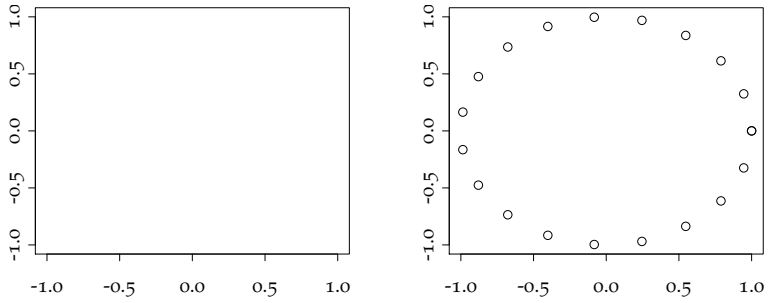


Figure 4: Two complete R expressions will produce at most two plots, as long as there are not multiple high-level plotting calls in each expression; option `fig.keep='all'` here.

Together with `fig.show='asis'`, we can show the process of plotting step by step like Figure 5.

A further note on plot recording: **knitr** examines all recorded plots (as R objects) and compares them sequentially; if the previous plot is a "subset" of the next plot (= previous plot + low-level changes), the previous plot will be removed when `fig.keep='high'`. If two succes-

```
plot(cars)
```

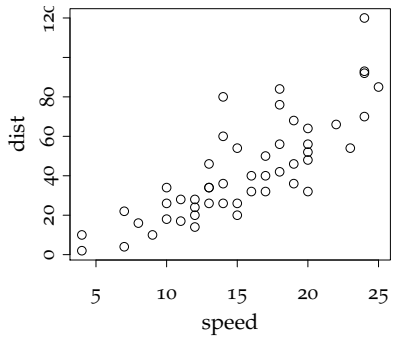
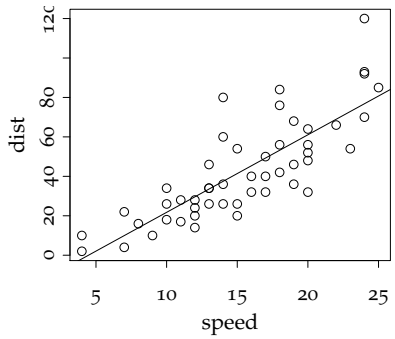


Figure 5: Low-level plot changes in base graphics can be recorded separately, and plots can be put in the places where they were produced.

```
abline(lm(dist ~ speed, data = cars))  # a regression line
```



sive plots are identical, the second one will be removed by default, so it might be a little bit surprising that the following chunk will only produce one plot by default[4]:

[4] adapted from https://github.com/yihui/knitr/issues/41

```
m <- matrix(1:100, ncol = 10)
image(m)
image(m * 2)  # exactly the same as previous plot
```

Plot Rearrangement

We can rearrange the plots in chunks in several ways. They can be inserted right after the line(s) of R code which produced them, or accumulated till the end of the chunk. There is an example in the main manual demonstrating `fig.show='asis'` for two high-level plots, and Figure 5 in this manual also demonstrates this option for a high-level plot followed by a low-level change.

Here is an example demonstrating the option `fig.keep='last'` (only the last plot is kept):

```
library(ggplot2)
pie <- ggplot(diamonds, aes(x = factor(1), fill = cut)) +
    xlab("cut") + geom_bar(width = 1)
pie + coord_polar(theta = "y")  # a pie chart
pie + coord_polar()  # the bullseye chart
```

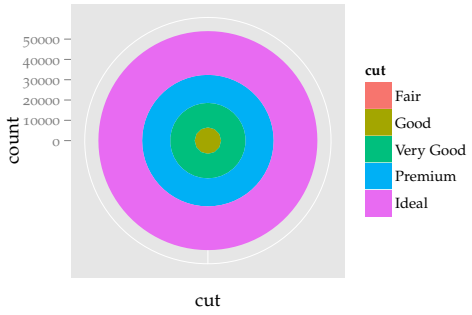


Figure 6: Two plots were produced in this chunk, but only the last one is kept. This can be useful when we experiment with many plots, but only want the last result. (Adapted from the **ggplot2** website)

When multiple plots are produced by a code chunk, we may want to show them as an animation with the option `fig.show='animate'`. Figure 7 shows a simple clock animation; you may compare the code to Figure 5 to understand that high-level plots are always recorded, regardless of where they appeared.

```
par(mar = rep(3, 4))
for (i in seq(pi/2, -4/3 * pi, length = 12)) {
    plot(0, 0, pch = 20, ann = FALSE, axes = FALSE)
    arrows(0, 0, cos(i), sin(i))
    axis(1, 0, "VI"); axis(2, 0, "IX")
    axis(3, 0, "XII"); axis(4, 0, "III"); box()
}
```

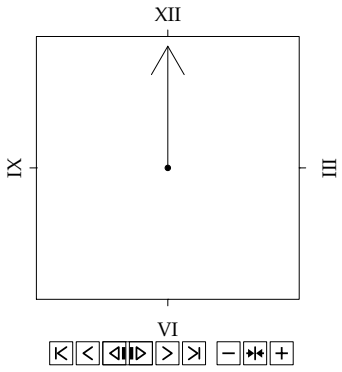


Figure 7: A clock animation. You have to view it in Adobe Reader: click to play/pause; there are also buttons to speed up or slow down the animation.

We can also set the alignment of plots easily with the `fig.align` option; this document uses `fig.align='center'` as a global option, and we can also set plots to be left/right-aligned. Figure 8 is an example

of a left-aligned plot.

```
stars(cbind(1:16, 10 * (16:1)), draw.segments = TRUE)
```

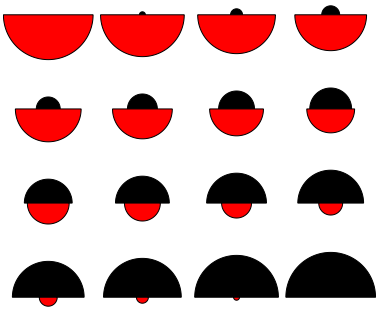


Figure 8: A left-aligned plot adapted from `?stars` (I call this the "Maruko" plot, and it is one of my favorites; see `http://en.wikipedia.org/wiki/Chibi_Maruko-chan`).

Plot Size

We have seen several examples in which two or more plots can be put side by side, and this is because the plots were resized in the output document; with the chunk option `out.width` less than half of the line width, LaTeX will arrange two plots in one line; if it is less than 1/3 of the line width, three plots can be put in one line. Of course we can also set it to be an absolute width like `3in` (3 inches). This option is used extensively in this document to control the size of plots in the output document.

The tikz Device

The main advantage of using tikz graphics is the consistency of styles between texts in plots and those in the main document. Since we can use native LaTeX commands in plots, the styles of texts in plots can be very sophisticated (see Figure 9 for an example).

When using XeLaTeX instead of PDFLaTeX to compile the document, we need to tell the **tikzDevice** package by setting the `tikzDefaultEngine` option before all plot chunks (preferably in the first chunk):

```
options(tikzDefaultEngine = "xetex")
```

This is useful and often necessary to compile tikz plots which contain (UTF8) multi-byte characters.

Figure Caption

If the chunk option `fig.cap` is not `NULL` or `NA`, the plots will be put in a `figure` environment when the output format is LaTeX, and this

```
plot(0:1, 0:1, type = "n", ylab = "origin of statistics",
    xlab = "statistical presentation rocks with \\LaTeX{}")
text(0.5, c(0.8, 0.5, 0.2), c("\\texttt{lm(y
\\textasciitilde{} x)}",
    "$\\hat{\\beta}=(X^{\\prime}X)^{-1}X^{\\prime}y$",
"$(\\Omega,\\mathcal{F},\\mu)$"))
```

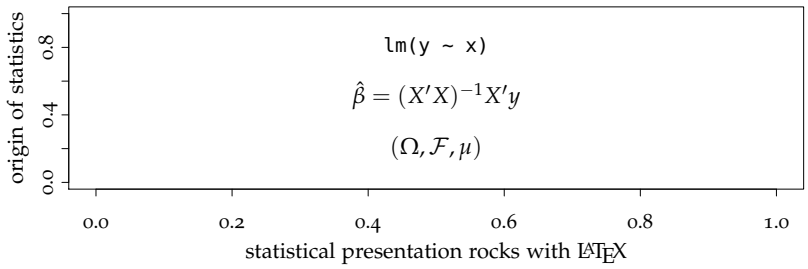


Figure 9: A plot created by **tikzDevice** with math expressions and typewriter fonts. Note the font style consistency – we write the same expressions in LaTeX here: $\hat{\beta} = (X'X)^{-1}X'y$ and $(\Omega, \mathcal{F}, \mu)$; also `lm(y ~ x)`.

option is used to write a caption in this environment using `\caption{}`. The other two related options are `fig.scap` and `fig.lp` which set the short caption and a prefix string for the figure label. The default short caption is extracted from the caption by truncating it at the first period or colon or semi-colon. The label is a combination of `fig.lp` and the chunk label. Because `figure` is a float environment, it can float away from the chunk output to other places such as the top or bottom of a page when the TeX document is compiled.

Other Features

The **knitr** package can be extended with hook functions, and here I give a few examples illustrating the flexibility.

Cropping PDF Graphics

Some R users may have been suffering from the extra margins in R plots, especially in base graphics (**ggplot2** is much better in this aspect). The default graphical option `mar` is about `c(5, 4, 4, 2)` (see `?par`), which is often too big. Instead of endlessly setting `par(mar)`, you may consider the program `pdfcrop`, which can crop the white margin automatically[5]. In **knitr**, we can set up the hook *hook_pdfcrop()* to work with a chunk option, say, `crop`.

[5] `http://www.ctan.org/pkg/pdfcrop`

```
knit_hooks$set(crop = hook_pdfcrop)
```

Now we compare two plots below. The first one is not cropped (Figure 10); then the same plot is produced but with a chunk option `crop=TRUE` which will call the cropping hook (Figure 11).

```
par(mar = c(5, 4, 4, 2), bg = "white")  # large margin
plot(lat ~ long, data = quakes, pch = 20, col = rgb(0,
    0, 0, 0.2))
```

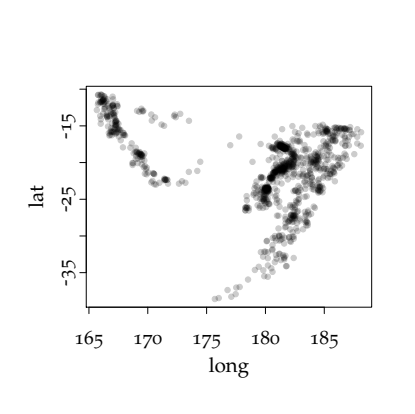


Figure 10: The original plot produced in R, with a large margin.

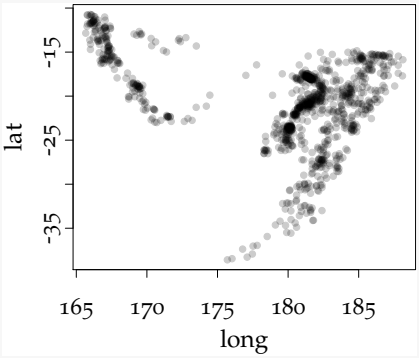


Figure 11: The cropped plot; it fits better in the document.

As we can see, the white margins are gone. If we use *par()*, it might be hard and tedious to figure out a reasonable amount of margin in order that neither is any label cropped nor do we get a too large margin. My experience is that `pdfcrop` works well with base graphics, but barely works with **grid** graphics (therefore **lattice** and **ggplot2** are ruled out).

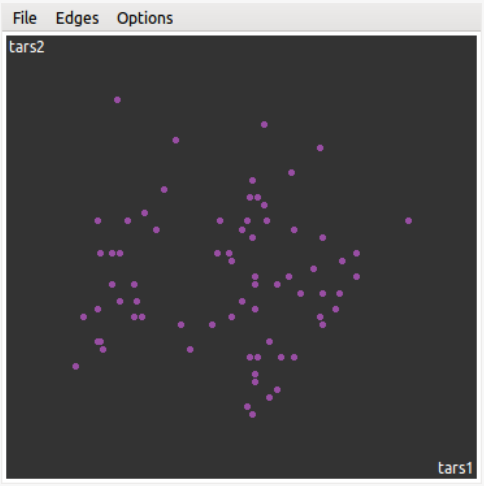
Manually Saved Plots

We have explained how R plots are recorded before. In some cases, it is not possible to capture plots by *recordPlot()* (such as **rgl** plots), but we can save them using other functions. To insert these plots into the output, we need to set up a hook first like this (see `?hook_plot_custom` for details):

```
knit_hooks$set(custom.plot = hook_plot_custom)
```

Then we set the chunk option `custom.plot=TRUE`, and manually write plot files in the chunk. Here we show an example of capturing GGobi plots using the function *ggobi_display_save_picture()* in the **rggobi** package:

```
library(rggobi)
ggobi(ggobi_find_file("data", "flea.csv"))
Sys.sleep(1)  # wait for snapshot
ggobi_display_save_picture(path = fig_path(".png"))
```

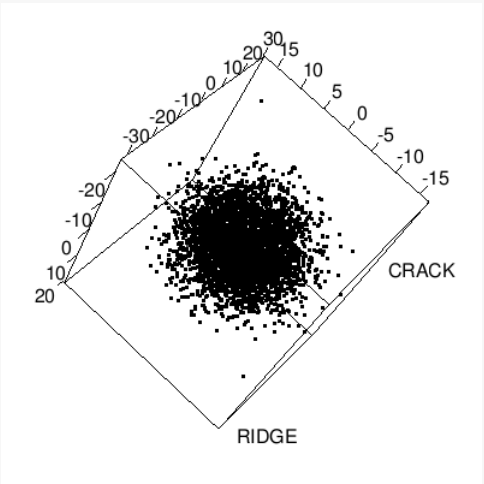


One thing to note here is we have to make sure the plot filename is from *fig_path()*, which is a convenience function to return the figure path for the current chunk.

We can do whatever normal R plots can do with this hook, and we give another example below to show how to work with animations.

```
library(animation)  # adapted from demo('rgl_animation')
data(pollen)
uM <- matrix(c(-0.37, -0.51, -0.77, 0, -0.73, 0.67,
    -0.1, 0, 0.57, 0.53, -0.63, 0, 0, 0, 0, 1), 4, 4)
library(rgl)
open3d(userMatrix = uM, windowRect = c(0, 0, 400,
    400))
plot3d(pollen[, 1:3])
zm <- seq(1, 0.05, length = 20)
par3d(zoom = 1)  # change the zoom factor gradually later
for (i in 1:length(zm)) {
    par3d(zoom = zm[i])
    Sys.sleep(0.05)
```

```
    rgl.snapshot(paste(fig_path(i), "png", sep = "."))
}
```



rgl Plots

With the hook *hook_rgl()*, we can easily save snapshots from the **rgl** package. We have shown an example in the main manual, and here we add some details. The rgl hook is a good example of taking care of details by carefully using the `options` argument in the hook; for example, we cannot directly set the width and height of rgl plots in *rgl.snapshot()* or *rgl.postscript()*, so we make use of the options `fig.width`, `fig.height` and `dpi` to calculate the expected size of the window, then resize the current window by *par3d()*, and finally save the plot.

This hook is actually built upon *hook_plot_custom()* – first it saves the **rgl** snapshot, then it calls *hook_plot_custom()* to write the output code.