Graphics in R

Arni Magnusson

11 Nov 2010

1 Devices

1.1 Generic

New

```
dev.new()       # new
```

Info

```
dev.cur()       # integer (1 if no device is open)
dev.list()      # integer vector, named
dev.next()      # next
dev.prev()      # previous
dev.interactive(orNone=TRUE)  # interactive device, or none is open
```

Select

```
dev.set(n)      # next device by default
```

Close

```
dev.off()       # close current
graphics.off()  # close all, better than while(!is.null(dev.list()))
```

1.2 Screen

windows()
x11()
dev.new()

Data analysis and quick export to any format.

trellis.device(col=F)

Multivariate data analysis and quick export to any format.

record=T

Record graphs for PgUp and PgDn animation, works both in windows() and trellis.device().

1.3 Vector files

postscript("d:/out.eps", horizontal=F, onefile=F, width=8, height=6, paper="special")

For publication quality on any platform, full control of graph size and arrangement, as well as font. The onefile and paper arguments create an EPS compatible file.

pdf("d:/auto.pdf", version="1.1", width=6, height=6, family="Helvetica")

The pdf device is not reliable, e.g. xyplot(1~1, cex=0) is supported by all other devices. Color defaults do not apply, files can be huge, and so on. Better distill EPS. The rgb(..., alpha) support can be mimicked with plot order.

win.metafile("d:/out.wmf", width=8, height=10.5)

For publication quality in Word. Can be converted into Drawing inside Word to change fonts, but then delete masked axis lines. File size of .doc can be much smaller than importing .eps,

but .pdf file size is not going to change much. Viewing quality in Word is perfect when .wmf is converted into Drawing, ok for unconverted .wmf, bad for .eps.

trellis.device("postscript", file="d:/trellis.eps", horizontal=F, onefile=F, paper="special")

Postscript and metafile device supporting Trellis plotting functions.

1.4 Raster files (gs)

TIFF

```
bitmap("d:/fig.tif", type="tiff24nc", height=6, width=6, res=72)
```

PNG

```
bitmap("d:/fig.png", type="png16m",   height=6, width=6, res=72)
```

Counterintuitively, png16m creates equal or smaller files than png256, because the larger palette will eliminate the need for dithered colors.

JPG

```
bitmap("d:/fig.jpg", type="jpeggray", height=6, width=6, res=72)
bitmap("d:/fig.jpg", type="jpeg",     height=6, width=6, res=72)
```

1.5 Raster files (native)

TIFF

```
tiff("d:/fig/tif", width=480, height=480, res=72)
```

PNG

```
png("d:/fig.png", width=480, height=480,
```

2 Plot functions

2.1 Univariate distribution

One measurement for each factor

```
barplot/barchart(~num)
pie(num)
```

One measurement for each factor:factor combination

```
dotplot(y~x1|x2)
dotplot(y~x1, groups=x2)
```

Many measurements

```
stripplot/stripchart(~num)
boxplot/bwplot/bpplot/vioplot/vioplot2(~num)
hist/histogram(~num)
plot(density(num))/densityplot(~num)
plot(quantile(x,seq(0,1,0.01)),seq(0,1,0.01))/plot(ecdf(x))
qqnorm/qqline/qqmath(~num)
rug(num)  # add to existing plot
```

By factor

```
dotplot/stripplot/stripchart(num~fac)
boxplot/bwplot/bpplot/parviol/vioplot/vioplot2(num~fac)
dotchart(matrix)               # one num per fac
histogram(~num|fac)
densityplot(~num|fac)
qqmath(~num|fac)
cut(num, breaks)               # num->fac
shingle(num, intervals=rbind)  # num->fac
# rug
```

2.2 Bivariate relationship

Scatter

```
plot/xyplot(y~x)
bagplot(x,y)      # with density and outliers
```

Scatter by factor

```
plot(type="n"); points(x,y); points(x,y)
matplot(mat)
coplot(y~x|fac)
xyplot(y~x|fac)
xyplot(y~x, groups=fac)  # implicit panel=panel.superpose
xyplot(y1+y2+y3~x)
```

Filled area

```
areaplot(x, y)
```

Average Y by X

```
plotmeans(y~x,bars=F,n.label=F)  # ... can pass gap=0 to plotCI()
interaction.plot(x1,x2,y)
plot.design(y~x1+x2)
```

Error bars

```
plotCI(x,y,ui,li)            # err="x" for horizontal
xYplot(Cbind(y,lo,hi)~x)     # Hmisc, vertical
segplot(x~lo+hi, centers=y)  # latticeExtra, horizontal
```

Least squares line

```
plot; abline(lm)
xyplot(y~x, panel=function(...){panel.xyplot(...);panel.lmline(...)})
scatterplot(x,y); scatterplot(y~x)
```

Least squares line by factor

```
plot; points(subset,pch); abline(lm(subset),lty)
coplot(y~x|fac,
  panel=function(x,y,pch,col,...)
  {panel.xyplot(x,y,pch=pch,col=col,...); abline(lm(y~x,...))}
xyplot(y~x|fac,
  panel=function(x,y) {panel.xyplot(x,y); panel.lmline(x,y)})
```

Loess line

Use 'loess' (Cleveland et al. 1992), since the older 'lowess' (Cleveland 1979, 1981) produces spurious residual trends.

```
loess(y~x)        # 1 works for multiple dimensions, weights etc.
loess.smooth(x,y) # 2 restricted but fast, used by Trellis functions
scatter.smooth(x,y)
xyplot(y~x, panel=function(x,y){panel.xyplot(x,y);panel.loess(x,y)})
```

```
scatterplot(x,y); scatterplot(y~x)
```

Loess line by factor

```
plot; points(subset,pch); lines(fitted(loess)[order(x)],lty)
coplot(y~x|fac,
  panel=function(x,y,pch,col,...)
  {panel.xyplot(x,y,pch=pch,col=col,...); lines(loess(y~x,...))}
xyplot(y~x|fac,
  panel=function(x,y) {panel.xyplot(x,y); panel.loess(x,y)})
scatterplot(x,y); scatterplot(y~x)
```

2.3 Multivariate I: Not surface

Quick overview of dataframe

```
datadensity(data.frame)
```

Means by factors

```
plot.design(y~x)          # or median(), min(), max(), sd(), etc.
```

Frequency plots

```
fourfoldplot(table(x,y))  # y and x are binomial
cdplot(y~x)               # y is factor, x is continous
spineplot(y~x)            # y and x are factors
sunflowerplot(x,y)        #        "
balloonplot(table(x,y))   #        "
mosaicplot(table(x,y))    #        "
assocplot(table(x,y))     #        "
smoothScatter(x,y)        # y and x are continuous
hist2d(x,y)               #        "               (nbins=30,col=...)
plot(hexbin(x,y))         #        "               (repeat if error)
```

Trivariate frequency plot

The rgbDemo() function maps three variables on a triangle.
Given that A+B+C = 1, the calculations only refer to A and B:

```
x <- 1 - B - A/2
y <- A * sin(pi/3)
plot(c(0,0.5,1,0), c(0,sin(pi/3),0,0), type="l", axes=F, ann=F)
points(x, y)
```

Array of special markers

```
plot(x,y,cex=n)       # n is the marker size
bubbleplot(z~x+y)     # different color for pos/neg residuals
plotBubbles(mat)      # different color for pos/neg residuals
symbols(x,y,...)      # circle, square, rectangle, star, thermo, box
stars(data.frame(num,fac,num,num), draw.segments=T) # misleading area
faces(data.frame(num,fac,num,num))
```

Pairwise scatter

```
pairs(data.frame(x1,x2))
pairs(~x1+x2, data=X)
splom(~data.frame(x1,x2))
scatterplotMatrix(data.frame(x1,x2))
scatterplotMatrix(~x1+x2, data=X)
```

Pairwise scatter by factor

```
splom(~data.frame(x1,x2)|fac)
scatterplotMatrix(~x1+x2|fac, data=X)
```

Correlation

```
plotcorr(m)
plotcorr.bw(m)
plotcorr.col(m)
```

Parallel coordinates of every datapoint

```
parviol(data.frame(x1,x2))
parcoord(data.frame(x1,x2))
parallel(~data.frame(x1,x2))
parallel(~data.frame(x1,x2)|fac))
mvtsplot(data.frame(x1,x2))
```

3D cloud

```
cloud(z~x+y)
cloud(z~x+y|fac)
scatterplot3d(x,y,z)
spin3R(data.frame(x,y,z))
plot3d(x,y,z)
```

2.4 Multivariate II: Surface

Surface precalculations

x, y and z are data vectors of equal length

```
x<-rnorm(1000); y<-rnorm(1000); z<-sin(x)+cos(y)
xcoords<-pretty(x,10); ycoords<-pretty(y,10)
model <- loess(z~x+y)
grid <- expand.grid(x=xcoords,y=ycoords)
surface.vector <- predict(model, grid)
surface.matrix <- matrix(surface.vector,nrow=length(xcoords))
```

2D contour

```
contour(xcoords,ycoords,surface.matrix)           # elevation lines
filled.contour(xcoords, ycoords, surface.matrix)  # elevation regions
image(xcoords, ycoords, surface.matrix)           # elevation squares
contourplot(surface.vector~grid$x+grid$y) # trellis elevation lines
levelplot(surface.vector~grid$x+grid$y)   # trellis elevation squares
```

3D surface

```
persp(xcoords,ycoords,surface.matrix)
```

Try persp(xcoords,ycoords,surface.matrix,theta=45, phi=30, expand=0.5, shade=0.5, ticktype="detailed")

```
wireframe(surface.vector~grid$x+grid$y)
persp3d(xcoords, ycoords, surface.vector)
```

Overlay points on surface

```
points(trans3d(x, y, z, persp(xcoords,ycoords,surface.matrix)))
```

2.5 Multivariate III: Surface smoothers

Least squares polynomial linear model

```
lm(z~x+y+I(x^2)+I(y^2)+x*y)   # six coefficients
glm(z~x+y+I(x^2)+I(y^2)+x*y)  # β0, βX, βX2, βY, βY2, βXY
```

Least squares polynomial trend surface

```
surf.ls(np=2, x, y, z)
surf.ls(np=2, data.frame)
```

```
trmat(surf.ls.object)  # pass directly to contour() and persp()
```
The lm() and surf.ls models above are identical.
For np = 1 to 6, the number of estimated coefficients is 3, 6, 10, 15, 21, and 28.

Loess
```
loess(z~x+y)
```

Kriging
```
surf.gls(np=2, cov.function, x, y ,z, nx=1000)
prmat(surf.gls.object)
semat(surf.gls.object)
correlogram(surf.gls.object)
variogram(surf.gls.object)
```
The supported covariance functions are: expcov, gaucov, sphercov.

2.6 MCMC chains

package:mcmcplots
```
mcmcplot()
traplot()
denplot()
caterplot()
corplot()
```

2.7 Add plot elements

Points, lines, arrows
```
points()
lines()
segments()
arrows()   # open arrowheads
```

Circles

Radius in x-axis units:
```
symbols(x, y, circles=r, inches=F, add=TRUE)
```
Ellipse:
```
circle <- function(x, y, r, ...)
{
  theta <- seq(0, 2*pi, length.out=200)
  xcoords <- x + r*cos(theta)
  ycoords <- y + r*sin(theta)
  lines(xcoords, ycoords, ...)
}
```

Polygons
```
polygon()  # draw solid arrowheads from scratch
rect()
```

Text
```
text()
mtext()
```

Grid lines
```
grid()     # see also tcl
```

Axis

```
axis()
Axis()     # supports POSIXt and other classes
box()
```

Labels

```
title()
```

2.8 Identify

Interactively identify datapoints

```
identify(x,y,labels)
```

2.9 Boxplot details

Stats

Boxplot stats are not simple quantiles:

```
x <- 1:100
fivenum(x)                              # 1  25.5   50.5  75.5  100
boxplot.stats(x)                        # 1  25.5   50.5  75.5  100
quantile(x, c(0, 0.25, 0.50, 0.75, 1))  # 1  25.75  50.5  75.25 100
```

Tukey's boxplot (4 Jul 2005 email)

When you type ?boxplot, the documentation leads to ?boxplot.stats for details about the computation, but even there the explanations are a little short on the details. When Tukey (1977) invented the boxplot, he defined it like this:

```
---  max(x[x < quantile(x,0.75)+1.5*IQR])
 |
|-|  quantile(x,0.75) \
| |                    |
|=|  median(x)         | IQR
| |                    |
|-|  quantile(x,0.25) /
 |
---  min(x[x > quantile(x,0.25)-1.5*IQR])
```

where IQR is the interquartile range, IQR<-abs(quantile(x,0.75)-quantile(x,0.25)), i.e. the height of the solid box.

The whiskers are the complicated part here, so let's focus on the upper one and imagine you're drawing it without a computer. After drawing the box and median, you calculate the maximum length of the whisker, 1.5*IQR, and you put your pencil down this distance from the box (not the median). Tukey specified that the whisker must be drawn where a datapoint exists, so you slowly move your pencil down until you find an actual datapoint, and you make a horizontal line there. Datapoints above the whisker are outliers.

Quantile whiskers

The boxplot function in R doesn't make it very easy to draw the whiskers somewhere else than Tukey (1977) intended. The shortest way to do this would be creating a list similar to the one returned by boxplot, and then draw it using bxp:

```
x <- rnorm(1000)
custom <- boxplot(x, plot=FALSE)
custom$stats[c(1,5),] <- quantile(x, c(0.1,0.9))
```

```
bxp(custom[c(1,6)])  # outliers not drawn
```

3 Pre-plotting parameters

3.1 Essential: fig, mfrow, new, plt

fig = c(x=0,1, y=0,1)

Figure area and location, relative to the device area. This is where the plot and its labels are placed. Use mfrow for even layout, but par(fig,new) for uneven layout.

mfrow = c(nrow=1, ncol=1)

Arrange figure areas of even size that fill the device area. Use mfrow for even layout, but par(fig,new) for uneven layout.

new = FALSE

Pretend the device is new and thus overlay more than one high-level plots.

plt = c(x=0.1,1.0, y=0.1,1.0)

Plot area and location, relative to the figure area. This is the area bounded by the plot box.

3.2 Also used: bg, din

bg = "transparent"

Background color of device area. See also plotting parameter below.

din = c(7,7)

Device area in inches (read only).

3.3 Redundant: fin, mai, mar, oma, pin

fin = c(width=8.0, height=10.5)

Use fig.

mai = c(bottom=1.0, left=0.8, top=0.8 right=0.4)

Use plt.

mar = c(bottom=5, left=4, top=4, right=2)

Use plt.

oma = c(bottom=0, left=0, top=0, right=0)

Use fig.

pin = c(width=7, height=9)

Use plt.

4 Plotting parameters

4.1 Essential: ann, axes, cex, col, las, lty, lwd, mgp, pch, tcl, type

ann = TRUE

Display axis labels.

axes = TRUE

Display axes.

cex = 1.0

Size of plotting symbols relative to the default. For text size: cex.axis, cex.lab, cex.main, cex.sub.

col = "black"

Color of plot symbols. For text color use col.axis, col.lab, col.main, col.sub. For tick color use axis(fg).

las = 0

Alignment of axis label: 0=like axis, 1=horizontal, 2=not like axis, 3=vertical.

lty = 1

Line type.

lwd = 1

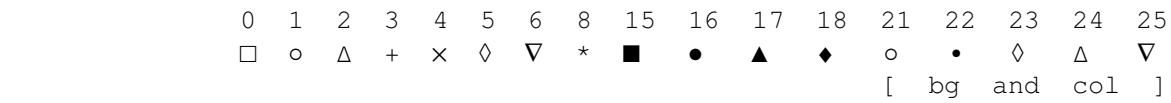
Line width.

mgp = c(title=3, label=1, line=0)

Axis margins, in mex units.

pch = 1

Plot symbol.



Use NA or "" to omit symbols. Other negative-sounding values are not useful:

	Symbol	File
1,-1,NULL	circles	big
0	squares	big
" "	empty	big
NA,""	empty	small

tcl = -0.5

Tick length, in mex units.

type = "p"

b points and lines (spaced)
c lines (spaced)
h histogram (vertical lines)

```
l lines
n no points or lines
o points and lines (lines on top)
p points
S steps (unusual, line anticipates next point)
s steps (normal, lines is flat until next point)
```

4.2 Less used: bg, bty, family, fg, font, srt, usr, xaxp, xaxs, xaxt, xpd

bg = "transparent"

Fill color for pch 21-25. See also 'pre-plotting parameters'.

bty = "O"

Box types are O, L, U, C,], 7, and "n" for no box.

family = "sans"

The options are: serif, sans, mono, symbol.

New family: ComputerModern (see ?postscript, didn't work for me)

fg = "black"

Color of box and axes/ticks.

font = 1

Font style of axis labels and main title: 1=plain, 2=bold, 3=italic, 4=bold italic, 5-13=symbol/times/courier (see subsection 'Font Style' below). Specific font.axis, font.lab, font.main, font.sub.

srt = 0

String rotation in degrees. Multiples of 90 are easy, but free rotation works best with vfont.

usr = c(x=0,1, y=0,1)

Show plot box coordinates. This is useful for adding to composite plots where labels do not show the actual range of values, usually slightly outside the xlim/ylim arguments.

xaxp = yaxp = c(0, 1, 5)

Tick mark locations, works a bit like axis(1, at=seq(0, 1, length=5)), but pretty() works best.

xaxs = yaxs = "r"

The default R style "r" puts the outer tick marks within box limits, but internal style "i" is like Excel, putting tick marks on the edges.

xaxt = yaxt = "s"

Can be used to suppress one of the axes, by passing xaxt="n".

xpd = FALSE

Clipping boundary, used to plot in the margin: FALSE=plot, TRUE=figure, NA=device.

4.3 Line ends: lend, ljoin, lmitre

lend

round, butt (no tail), square

ljoin
round, bevel (corner hacked off), mitre (sharp)

lmitre
10 by default. If this value is too low, ljoin="mitre" will be ignored and "bevel" used...

4.4 Not par: xlim, ylim, asp

asp = NA
Sets xlim or ylim, to control vertical/horizontal aspect ratio: 0.1 means flat, 10 means steep.
Tweaks xlim or ylim, whereas trellis 'aspect' tweaks plot area.

5 Smoothers

5.1 Step function

```
stepfun(x, c(0,y))        # function, 0 is value before min(x)
stepfun(x, c(0,y))(xout)  # evaluate at xout
lines(x, y, type="s")     # line
```

5.2 Linear interpolation

```
approxfun(x, y)     # function
approx(x, y, xout)  # evaluate at xout
lines(x, y)         # line
```

5.3 Spline interpolation

```
splinefun(x, y)        # function
splinefun(x, y)(xout)  # evaluate at xout
lines(spline(x, y))    # line
```

Not a smoother, but a wavy alternative to linear interpolation.

5.4 Moving average

```
moving(y, span=5)    # evaluate at x
lines(x, moving(y))  # line
```

Mainly useful if the ends have flat trends, since a diagonal line will have bent ends...

5.5 Loess smoother

```
loess(y~x, span=0.75)      # model, set span (0.25-Inf)
predict(loess(y~x), xout)  # evaluate at xout
lines(loess(y~x))          # line
```

Local polynomial, 2nd order (quadratic) by default.

5.6 Spline smoother

```
smooth.spline(x, y, spar=NULL)     # model, set spar (0-1)
predict(smooth.spline(x,y), xout)  # evaluate at xout
lines(smooth.spline(x,y))          # line
```

5.7 Kernel smoother

```
ksmooth(x, y, "normal", bandwidth=length(x)/10)  # list of x and y
ksmooth(x, y, "normal", x.points=xout)           # evaluate at xout
lines(ksmooth(x,y,"normal"))                     # line
```

Pass "normal" and bandwidth to override defaults. Package KernSmooth has better kernels.

6 Formatting text

6.1 font (1 to 13)

The plotting argument 'font' can take thirteen values (improvement since S-Plus workshop):

	plain	italic	bold	bold-italic
helvetica	1	2	3	4
symbol	5			
times	6	7	8	9
courier	10	11	12	13

The cex and font arguments only refer to symbols and text inside the graph. To format text outside of the graph, use cex.axis/font.axis, cex.lab/font.lab, cex.main/font.main, and cex.sub/font.sub.

6.2 family (windows)

Standard

```
serif
sans
mono
```

Closer look

```
windowsFonts()  # list of 3 strings:
                #   serif="TT Times New Roman"
                #   sans="TT Arial"
                #   mono="TT Courier New"
```

Add font from c:/windows/fonts

```
windowsFonts(Comic=windowsFont("TT Comic Sans MS"))
plot(1, type="n", family="comic")
text(1, "Text", family="serif")
```

6.3 family (hershey)

family	**font = 1**	**2**	**3**	**4**
	plain	italic	bold	bold-italic

```
HersheySerif              X       X     X           X
HersheySans               X       X     X           X
HersheySymbol             X       X     X           X
HersheySansSymbol         X       X
HersheyScript             X       X     X
HersheyGothicEnglish      X
HersheyGothicGerman       X
HersheyGothicItalian      X
```

6.4 family (ps)

Standard

```
postscript(family="Helvetica", pointsize=12)
```

There are many redundant font names, but eight font families. Look out for vertical placement of mathematical symbols. Sorted from abstract to ornamented:

```
AvantGarde         URWGothic
Helvetica          NimbusSan
Helvetica-Narrow   NimbusSanCond
Courier            NimbusMon
Palatino           URWPalladio
Times              NimbusRom
Bookman            URWBookman
CenturySch         NewCenturySchoolbook
```

Helvetica (default) for clarity, Courier for digital look, Times to be same as text, CenturySch for a respectable book look.

Closer look

```
postscriptFonts()  # list of 31 lists
postscriptFonts("Helvetica")  # family="Helvetica"
                              # metrics=c("Helvetica.afm", ...)
                              # encoding="default"
```

Add font from c:/gnu/texmf/fonts

Add postscript family in R:

```
postscriptFonts(CMSuper=Type1Font("CMSuper",paste("c:/gnu/texmf/fonts
/afm/public/cm-super/sf", c("rm","bx","ti","bi"), "1000.afm.gz",
sep="")))
```

Write EPS file:

```
eps("f:/test/cmsuper.eps",family="CMSuper"); plot(1); dev.off()
```

Or declare it on the fly:

```
eps("f:/test/cmsuper.eps",family=paste("c:/gnu/texmf/fonts/afm/public
/cm-super/sf", c("rm","bx","ti","bi"), "1000.afm.gz", sep="")));
plot(1); dev.off()
```

Embed font in PDF:

```
embedFonts("f:/test/cmsuper.eps", "pdfwrite",
"f:/test/cmsuper_embed.pdf", "c:/gnu/texmf/fonts/type1/public/cm-
super", "-dEPSCrop")
```

Or directly in Ghostscript with -sFONTPATH:

```
gs -sDEVICE=pdfwrite -sFONTPATH=c:/gnu/texmf/fonts/type1/public/cm-
super -dEPSCrop -dPDFSETTINGS=/prepress -dCompatibilityLevel=1.4 -
dSAFER -q -o f:/test/cmsuper_embed.pdf f:/test/cmsuper.eps
```

Finally, convert back to EPS:

```
2eps f:/test/comic_embed.pdf
```

Add font from c:/windows/fonts

Extract metrics in Dos shell:

```
ttf2afm -o f:/test/comic.afm c:/windows/fonts/comic.ttf
```

Add postscript family in R:

```
postscriptFonts(Comic=Type1Font("Comic",rep("f:/test/comic.afm",4)))
```

Write EPS file:

```
eps("f:/test/comic.eps",family="Comic"); plot(1); dev.off()
```

Convert TTF to PFB in Dos shell:

```
ttf2pt1 -b -Ga c:/windows/fonts/comic.ttf f:/test/comic
```

Embed font in PDF:

```
embedFonts("f:/test/comic.eps", "pdfwrite",
"f:/test/comic_embed.pdf", "f:/test", "-dEPSCrop")
```

Or directly in Ghostscript with -sFONTPATH:

```
gs -sDEVICE=pdfwrite -sFONTPATH=f:/test -dEPSCrop -
dPDFSETTINGS=/prepress -dCompatibilityLevel=1.4 -dSAFER -q -o
f:/test/comic_embed.pdf f:/test/comic.eps
```

Finally, convert back to EPS:

```
2eps f:/test/comic_embed.pdf
```

6.5 family (pdf)

Standard

Same as family (ps).

Closer look

Same as family (ps).

Add font from c:/gnu/texmf/fonts

Add PDF family in R:

```
pdfFonts(CMSuper=Type1Font("CMSuper",paste("c:/gnu/texmf/fonts/afm/pu
blic/cm-super/sf", c("rm","bx","ti","bi"), "1000.afm.gz", sep="")))
```

Write PDF file:

```
pdf("f:/test/cmsuper.pdf",family="CMSuper"); plot(1); dev.off()
```

Embed font:

```
embedFonts("f:/test/cmsuper.pdf", "pdfwrite",
"f:/test/cmsuper_embed.pdf", "c:/gnu/texmf/fonts/type1/public/cm-
super")
```

This calls Ghostscript with -sFONTPATH:

```
gs -sDEVICE=pdfwrite -sFONTPATH=c:/gnu/texmf/fonts/type1/public/cm-
super -dPDFSETTINGS=/prepress -dCompatibilityLevel=1.4 -dSAFER -q -o
f:/test/cmsuper_embed.pdf f:/test/cmsuper.pdf
```

Add font from c:/windows/fonts

Extract metrics in Dos shell:

```
ttf2afm -o f:/test/comic.afm c:/windows/fonts/comic.ttf
```

Add PDF family in R:

```
pdfFonts(Comic=Type1Font("Comic",rep("f:/test/comic.afm",4)))
```

Write PDF file:

```
pdf("f:/test/comic.pdf",family="Comic"); plot(1); dev.off()
```

Convert TTF to PFB in Dos shell:

```
ttf2pt1 -b -Ga c:/windows/fonts/comic.ttf f:/test/comic
```

Embed font:

```
embedFonts("f:/test/comic.pdf", "pdfwrite",
"f:/test/comic_embed.pdf", "f:/test")
```

This calls Ghostscript with -sFONTPATH:

```
gs -sDEVICE=pdfwrite -sFONTPATH=f:/test -dPDFSETTINGS=/prepress -
dCompatibilityLevel=1.4 -dSAFER -q -o f:/test/comic_embed.pdf
f:/test/comic.pdf
```

6.6 expression (math)

Maths

Format part of a text string or typeset mathematics in TeX-like syntax:

```
plot(1, xlab=expression(paste("a", italic(" b "))), ylab=expression(
  paste("Just like so: ", beta[0]+beta[1]*x+beta[2]*x^2)))
```

See help(plotmath) for details.

```
bolditalic(x)
beta is equivalent to symbol(b)
```

bquote

Protect everything, except evaluate what's .() blocks:

```
bquote(pi + .(pi))  # pi + 3.14159265358979
```

Returns a 'call', but plot labels interpret it as expression:

```
plot(pi, main=bquote(pi+.(pi)))
```

Advanced example

Philippe wanted to display a part of the plot title in boldface:

```
A <- 400
plot(1, main=paste("A is",A))
```

Dave Middleton used substitute to create the combo call:

```
plot(1, main=substitute(paste("A is ",bold(x)),
list(x=as.character(A))))
```

But Allan demonstrated a more compact way using bquote,

```
plot(1, main=bquote(paste("A is ", bold(.(as.character(A))))))
```

which boils down to:

```
plot(1, main=substitute(sqrt(x), list(x=i)))
plot(1, main=bquote(sqrt(.(i))))
```

6.7 Line height

lheight = 1

7 Color

7.1 Color argument

The 'col' argument in many plotting functions can take different forms:
- Keyword from colors()
- RGB-code between "#000000" and "#FFFFFF" (16.8M combinations)
- Integer referring to the current palette() definition

7.2 Creating RGB codes

gray(level)
Between 0 and 1, from black to white.

rgb(r,g,b,max=255)
Red, green, blue, between 0 and 255.

hsv(h,s,v)
Hue, saturation, value, between 0 and 1.

adjustcolor(x, alpha.f=1, red.f=1, green.f=1, blue.f=1)
Adjust existing color.

pickCol()
Pick color from RGB palette.

Custom ramp

```
rgb(colorRamp(c("red","white","blue"))(x),max=255) # 0 <= x <= 1
colorRampPalette(c("red","white","blue"))(n)       # number of colors
```

Pass bias=0.1 to make most colors like lower, or bias=10 to make most colors like upper.
The 'colorpanel' function in gplots is similar but less flexible.

grDevices

```
default 8        black red green blue cyan magenta yellow grey
gray.colors      black-grey-white
rainbow          red-yellow-green-blue-red (hue 0-1, or start-end)
heat.colors      red-yellow-white
terrain.colors   green-yellow-khaki-white
topo.colors      blue-green-yellow-white
cm.colors        cyan-white-magenta
```

hexbin

```
heat.ob          black-brown-white
plinrain         black-blue-green-orange-white
```

RColorBrewer::brewer.pal

```
brewer.pal.info  complete list
Blues    9       white-blue, also PuBu and PuBuGn
YlOrBr   9       white-brown, also Oranges and YlOrRd
Dark2    8       green orange purple magenta green yellow brown grey
Paired  12       light and dark: blue green red orange purple brown
Pastel2  8       green orange purple magenta green yellow brown grey
```

package:colorRamps

```
blue2red         blue-red
blue2yellow      blue-yellow
ygobb            yellow-green-olive-blue-black
```

8 Legend

8.1 Basic

```
matplot(matrix(1:4,2), type="l", lty=1, lwd=3, col=c("blue","red"))
legend("topleft", c("A","B"), lwd=3, col=c("blue","red"), bty="n")
```

8.2 Location

```
x, y           or keyword like "topleft"
xjust      0   x coordinate is left (0) or right (1) edge
yjust      1   y coordinate is bottom (0) or top (1) edge
x.instersp 1   horizontal space between label and symbol
y.intersp  1   vertical line space
adj        0   text is left (0) or right (1) aligned
ncol       1   arrange in several columns
horiz      F   arrange in one row
inset      0   padding if keyword like "topleft" is used, c(0.1,0.3)
xpd            TRUE to draw legend in margin
```

Margin example

```
par(plt=c(x=0.2,0.7, y=0.2,0.9))
matplot(matrix(1:4,2), type="l", lty=1, lwd=3, col=c("blue","red"))
legend("right", c("A","B"), lwd=3, col=c("blue","red"), bty="n",
       inset=-0.35, xpd=TRUE)
```

9 Composite plots

9.1 Traditional

Device area is the overall graph size
Figure area is the compartment
Plot area is inside the plot box

Decide how the device area should be divided into compartments, for example:

```
0.94 +----+-----+-----+
     |    |  1  |  2  |
0.20 +----+-----+-----+
     |    |     |     |
0.00 +----+-----+-----+
   0.00 0.12  0.54  0.96
```

```
figExample <- function(filename="out.eps", ...)
{
  eachPlot <- function(i, ...)
  {
    par(plt=c(0,1,0,1))
    plot(NA, ann=FALSE, axes=FALSE, xlim=c(0,4.5), ylim=c(0,4.5),
         xaxs="i", yaxs="i", type="n", ...)
    text(2.25, 2.25, LETTERS[i], cex=cex.text)
    box()
  }
  cex.text <- 1.4
  cex.axis <- 0.9
  cex.lab <- 1.1
  tcl <- -0.3
  x.mgp <- c(2,0.4,0)
  y.mgp <- c(2,0.5,0)

  if(!is.null(filename)) eps(filename, height=3, width=5)

  par(fig=c(0.12,0.54, 0.2,0.94))
  eachPlot(1)
  axis(1, cex.axis=cex.axis, mgp=x.mgp, tcl=tcl)
  axis(2, cex.axis=cex.axis, mgp=y.mgp, tcl=tcl, las=1)
  title(ylab="ylab", cex.lab=cex.lab, mgp=y.mgp, xpd=NA)

  par(fig=c(0.54,0.96, 0.2,0.94), new=TRUE)
  eachPlot(2)
  axis(1, cex.axis=cex.axis, mgp=x.mgp, tcl=tcl)

  par(fig=c(0.12,0.96, 0.2,0.94), new=TRUE); frame()
  title(xlab="xlab", cex.lab=cex.lab, mgp=x.mgp, xpd=NA)

  if(!is.null(filename)) dev.off()
  invisible(NULL)
}
```

9.2 Trellis

```
xyplot(y~x|z, layout=c(2,3))
print(trellis.object, position=c(x1,y1,x2,y2), more=T)
print(trellis.object, split=c(x,y,nx,ny), more=T)
```

9.3 ggplot2

10 Grid

10.1 Viewports

A viewport is the active canvas, in terms of coordinates and format.

Container
`grid.newpage`

Create, navigate
`pushViewport, popViewport, plotViewport`
`upViewport, downViewport`
`current.viewport, childNames, current.vpTree`

Collections
`vpList, vpStack, vbTree`

10.2 Grobs

Supplying a name enables us to activate and modify the grob later.

Elements
`grid.points`
`grid.lines, grid.segments  (use grid.lines to draw arrows)`
`grid.rect, grid.circle, grid.polygon`
`grid.text`

Fancy
`grid.strip`
`grid.grill`
`grid.xaxis, grid.yaxis, grid.pretty`
`grid.legend`

10.3 Parameters

Container
`gpar`

Units
`ncp, lines, inches`

10.4 Interactive graphics

`grid.locator`

11 Trellis

11.1 History

```
2001  0.2
      0.3   key, log, levelplot
2002  0.4
      0.5   horizontal
      0.6   plotmath, POSIXt
2003  0.7   auto.key, simpleKey
      0.8   fontface, fontfamily
2004  0.9   reverse limits
      0.10  trellis.last.object, summary, strip.custom
      0.11  current.panel.limits
2005  0.12  custom stats for bwplot
      0.13
2006  0.14
2007  0.15
      0.16
      0.17  box.width in bwplot, horizontal in parallel
2010  0.18  xyplot(ts), layout=c(NA,3), ylim=c(0,NA), improved POSIXt
      0.19  grid=TRUE
```

Relevant to scape: ylim=c(0,NA).

11.2 Points, lines, and text

```
lpoints
llines, lsegments, larrows, lrect
ltext
```

Inside a panel function, panel.xyplot() is actually a low-level function, so lpoints() = panel.xyplot(type="p"), and llines() = panel.xyplot(type="l").

11.3 Superpose

Overlay with 'groups' or '+' on right-hand side.

```
xyplot(y~x, groups=fac)  # long data frame
xyplot(y1+y2~x)          # wide data frame
```

11.4 Boxplot

```
bwplot(Speed~Expt, data=morley, col="grey", horiz=FALSE, pch="|",
       fill="grey", par.settings=list(box.umbrella=list(lty=1)))
```

11.5 Page layout

`print.trellis`

help("print.trellis")

```
position=c(x1,y1, x2,y2)            plot area
split   =c(x,y, x.layout,y.layout)  mfrow-like layout
more    =TRUE                       whether more plots will follow
```

trellis.currentLayout

11.6 Arguments

help(xyplot)

layout

```
c(columns, rows)
```

aspect

```
height  vertical/horizontal, e.g. 1.5
"fill"  fill device (default)
"xy"    45 degree banking
"iso"   same unit distance on x and y
```

Tweaks plot area, whereas traditional 'asp' tweaks xlim or ylim.

skip

```
c(T,F,F,F)                          which panels will be empty
```

between

```
list(x=hspace, y=vspace)            x and y names are required
```

scales

For splom, the argument pscales serves a similar purpose.

```
draw=TRUE                           whether to draw axis
log=FALSE                           whether to log-transform
relation=c("same","free","sliced")  axis limits vary between panels
alternating                         label sides vary between panels
limits                              axis limits, like xlim and ylim
at, tick.number, tck                tick marks (default tck=1)
labels, font, cex, rot              tick labels
abbreviate, minlength               tick label abbreviation
col                                 color of tick marks and labels
```

main xlab ylab sub

```
list(label, cex, col, font)         label, cex, col, font
```

strip

```
strip.custom(style, bg, fg)         style, bg, fg
```

It is easy to set the background color of all conditioning variables,

```
xyplot(yield~year|variety+site, data=barley,
       strip=strip.custom(bg="gold"))
```

but harder to set different colors for different conditioning variables on the fly:

```
xyplot(yield~year|variety+site, data=barley,
       strip=function(...,which.given,bg)
               strip.default(...,which.given,
                             bg=c("plum","gold")[which.given]))
```

Or adjust the default values beforehand:

```
trellis.par.set(strip.background=list(col=c("plum","gold")))
xyplot(yield~year|variety+site, data=barley)
```

par.strip.text

```
list(lines, cex, col, font)         lines, cex, col, font
```

key

```
list(space, text, lines, lwd)       space, text, lines, lwd
list(space, text, pch, cex)         space, text, pch, cex
```

```
  space  "top", "bottom", "left", "right"
```

11.7 Legend

See 'key' above.

```
splom(~iris[1:4], groups=Species, data=iris, pscales=FALSE, col=1:3,
      key=list(points=list(pch=1,col=1:3),
               text=list(levels(iris$Species))))
```

11.8 Parameters

```
show.settings               Show trellis settings
trellis.par.get             Get trellis parameters
trellis.par.set             Set trellis parameters
xyplot(par.settings=list()) Set local parameters on the fly
```

11.9 Interactive graphics

```
panel.identify
```

11.10 Custom panel function

```
xyplot(y~x, panel=function(x,y,...)
{panel.xyplot(x,y,...); panel.lmline(x,y,...)})
```

12 ggplot2

12.1 History

```
2007  0.5  Wickham at Iowa State Univ (later Rice Univ, Houston TX)
2008  0.6
      0.7
      0.8
```

12.2 qplot

12.3 ggplot

```
d <- ggplot(diamonds, aes(x=carat,y=price))
```

12.4 Low-level functions

geom

Geometric type of plot.

```
geom_abline      abline
geom_area        areaplot
geom_bar         barplot
geom_bin2d       hist2d
geom_blank       frame
geom_boxplot     boxplot
geom_contour     contour
geom_crossbar    boxplot
geom_density     density
geom_density2d   contour
geom_errorbar    plotCI
geom_errorbarh   plotCI
geom_freqpoly    plot
geom_hex         hexbin
geom_histogram   hist
geom_hline       abline
geom_jitter      jitter
geom_line        plot
geom_linerange   plot
geom_path        plot
geom_point       plot
geom_pointrange  plotCI
geom_polygon     polygon
geom_quantile    quantile
geom_rect        rect
geom_ribbon      polygon
geom_rug         rug
geom_segment     segment
geom_smooth      gam
geom_step        stepfun
geom_text        text
geom_tile        rect
geom_vline       abline
```

stat

Statistical computations.

```
stat_abline     abline
stat_bin        hist
stat_bin2d      hist2d
stat_binhex     hexbin
stat_boxplot    boxplot
stat_contour    contour
stat_density    density
stat_density2d  contour
stat_function   function
stat_hline      abline
stat_identity   identity
stat_qq         qqplot
stat_quantile   quantile
stat_smooth     gam
stat_spoke      arrows
stat_sum        sum
stat_summary    apply
stat_unique     unique
stat_vline      abline
```

scale

Scale axes, symbols, lines, and colors.

```
scale_alpha       rgb
scale_brewer      brewer.pal
```

```
scale_continuous  axis
save_date         axis
scale_datetime    axis
scale_discrete    axis
scale_gradient    rainbow
scale_gradient2   hsv
scale_gradientn   hsv
scale_grey        grey.colors
scale_hue         rainbow
scale_identity    identity
scale_linetype    lty
scale_manual      function
scale_shape       pch
scale_size        cex
```

coord

Transformed coordinates.

```
coord_cartesian   plot
coord_equal       asp
coord_flip        plot
coord_map         map
coord_polar       atan2
coord_trans       function
```

facet

Grid and multipanel facet.

```
facet_grid        grid
facet_wrap        mfrow
```

position

Stack columns or jitter position.

```
position_dodge    barplot
position_fill     barplot
position_identity barplot
position_jitter   jitter
position_stack    barplot
```

13 rgl

13.1

14 Maps

14.1 maps, mapdata, mapproj

map

```
map()  # map(database="world", region=".", projection="")
map("worldHires", "iceland", projection="lambert", par=c(63,67))
map("worldHires", "iceland", projection="mercator")
map("worldHires", "iceland", projection="orthographic")
map("worldHires", "iceland", projection="rectangular", par=65)
```

mapproject

```
c(range(x), range(y))                 # -24.5   -13.5    63.4   66.5
mapproject(x, y, "lambert", c(63,67)) # -0.0362  0.0368 -0.448 -0.398
mapproject(x, y, "mercator")          # -0.0964  0.0964  1.44   1.57
mapproject(x, y, "orthographic")      # -0.0399  0.0406 -0.448 -0.398
mapproject(x, y, "rectangular", 65)   # -0.0407  0.0407  1.11   1.16
```

Axes and grid

```
map.axes()   # box, ticks, and labels
map.scale()  # scale
grid()       # grid, if map is not projected
map.grid()   # grid, if map is projected
```

Object

```
m <- map("world", "iceland")  # x, y, range, names
```

Iceland

```
map("world", "iceland")       # 115 points
map("worldHires", "iceland")  # 1198 points
```

14.2 PBSmapping

14.3 geo

See geoplot.doc.

14.4 gmt

15 Bitmaps

15.1 [as.raster]

The 'as.raster' function (package:grDevices, 'raster' class) is a simple matrix of "#rrggbb".
The 'rasterImage' function (package:graphics) adds a 'raster' object to an existing plot.

Create

```
col.mat <- attributes(rich.colors(50, rgb=TRUE))$rgb
col.c <- array(round(col.mat,2), c(5,10,3))
col <- as.raster(col.c)
plot(NA, xlim=c(1,10), ylim=c(1,10), ann=FALSE, axes=FALSE)
rasterImage(col, 1, 2.5, 10, 7.5, interpolate=FALSE)
```

Grayscale:

```
bw.c <- matrix(round(seq(1,0,length=50),2), 5)
bw <- as.raster(bw.c)
plot(NA, xlim=c(1,10), ylim=c(1,10), ann=FALSE, axes=FALSE)
rasterImage(bw, 1, 2.5, 10, 7.5, interpolate=FALSE)
```

15.2 pixmap

Bivand, Leisch, and Maechler.
S4 object for PNM images. Stores R, G, B matrices as separate attributes. 24 bytes per pixel.

Import

```
ppm <- read.pnm("file.ppm")  # class "pixmapRGB"
attributes(ppm)$size         # c(5,10)
attributes(ppm)$channels     # c("red","green","blue")
```

Grayscale:

```
pgm <- read.pnm("file.pgm")  # class "pixmapGrey"
attributes(pgm)$size         # c(5,10)
attributes(pgm)$channels     # "grey"
```

Extract

```
ppm.c <- getChannels(ppm)    # array 5x10x3
ppm.c[,,"red"]               # matrix, same as attributes(ppm)$red
```

Grayscale:

```
pgm.c <- getChannels(pgm)    # matrix 5x10
pgm.c                        # matrix, same as attributes(pgm)$gray
```

Plot

```
plot(ppm)
```

Create

```
col.mat <- attributes(rich.colors(50, rgb=TRUE))$rgb
col.c <- array(round(col.mat,2), c(5,10,3))
col <- pixmapRGB(col.c)
plot(col)
```

Grayscale:

```
bw.c <- matrix(round(seq(1,0,length=50),2), 5)
bw <- pixmapGrey(bw.c)
plot(bw)
```

List of functions

```
getChannels
pixmapGrey
pixmapIndexed
pixmapRGB
plot
read.pnm
write.pnm                    # export
```

15.3 png

Import/export PNG images as arrays.

Import

```
p <- readPNG("file.png")  # array
```

Extract

```
p[,,1]                    # matrix 5x10 (1=red, 2=green, 3=blue)
```

Create

```
a <- array(runif(150), c(5,10,3))
```

List of functions

```
writePNG                  # export
```

15.4 ReadImages

S3 object for JPEG images. Simple RGB array with attribute 'type'. 24 bytes per pixel. Subset of 'rimage'.

Import

```
jpg <- read.jpeg("file.jpg")  # class c("imagematrix","array")
dim(jpg)                      # c(5,10,3)
attributes(jpg)$type          # "rgb"
```

Grayscale:

```
j <- read.jpeg("file.jpg")    # class c("imagematrix","matrix")
dim(jpg)                      # c(5,10)
attributes(j)$type            # "gray"
```

Extract

```
jpg[,,1]                      # matrix 5x10 (1=red, 2=green, 3=blue)
```

Grayscale:

```
j[,]                          # matrix 5x10
```

Plot

```
plot(jpg)
```

Create

```
col.c <- array(runif(150), c(5,10,3))
imagematrix(col.c)
```

Grayscale:

```
bw.c <- matrix(runif(50), nrow=5)
bw <- imagematrix(bw.c)
```

List of functions

```
imageType
normalize                    # filter
plot
read.jpeg
rgb2grey                      # convert to grayscale
```

15.5 rimage

S3 object for JPEG images. Simple RGB array with attribute 'type'. 24 bytes per pixel. Superset of 'ReadImages'.

Import

```
jpg <- read.jpeg("file.jpg")  # class c("imagematrix","array")
dim(jpg)                      # c(5,10,3)
attributes(jpg)$type          # "rgb"
```

Grayscale:

```
j <- read.jpeg("file.jpg")    # class c("imagematrix","matrix")
dim(jpg)                      # c(5,10)
attributes(j)$type            # "gray"
```

Extract

```
jpg[,,1]                      # matrix 5x10 (1=red, 2=green, 3=blue)
```

Grayscale:

```
j[,]                          # matrix 5x10
```

Plot

```
plot(jpg)
```

Create

```
col.c <- array(runif(150), c(5,10,3))
imagematrix(col.c)
```

Grayscale:

```
bw.c <- matrix(runif(50), nrow=5)
bw <- imagematrix(bw.c)
```

List of functions

```
imageType
plot
read.jpeg
rgb2grey                      # convert to grayscale
```

15.6 Summary

	Author	**Maintainer**	**Publ**	**Import**	**Exp**	**Class**
grDevices	Core	Core	2010	-	-	raster
pixmap	Bivand	Leisch	2009	pnm	pnm	pixmapRGB (S4)
png	Urbanek	Urbanek	2010	png	png	array
ReadImages	Loecher	Loecher	2009	jpeg	-	imagematrix
rimage	Nikon	[orphaned]	2010	jpeg	-	imagematrix