Structured Graphics Using the Lattice Package

John Maindonald

Australian National University

June 18, 2008
# Lattice Graphics

| **Lattice** | Lattice implements trellis style graphics  
( the S-PLUS flavour was the original) |
| **Grid** | *grid* is a low-level graphics system, used to build *lattice*.  
For *grid*, see Part II of Paul Murrell’s *R Graphics* |
| **Lattice vs base** | Lattice is more structured, automated and stylized.  
Much is done automatically, without user intervention.  
Changes to the default style are harder than for base. |
| **‘Printing’** | Lattice functions do not “print” the graph. Conceptually:  
*graphs*  
gph <- xyplot(ACT ~ Year, data=austpop)  
print(gph) |
| **Updating** | update(gph, par.settings=simpleTheme(pch=16, cex=1) |
| **Lattice syntax** | Lattice syntax is consistent and tightly regulated  
For lattice, graphics formulae are mandatory. |
Topics (some covered cursorily)

- Printing and updating issues.
- Customization
  - Point, line and related settings.
  - Axes, tick marks & labels, scales, etc.
  - Mathematical, etc., expressions: Section 3.3.
  - Regression lines &/or smooth curves: Section 3.3.
- Panels and other “viewports”: Finer control
  - Panel functions: Section 5.1
  - Interaction with lattice plots: Section 5.2.
- Other lattice functions (there are many).

Definitive reference
A Dataset that is Ideally Made for Lattice

Australian & NZ apparent per person annual alcohol consumption of the pure alcohol content (in liters) of liquor products, 1998 to 2006.

<table>
<thead>
<tr>
<th></th>
<th>Beer</th>
<th>Wine</th>
<th>Spirit</th>
<th>Country</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.24</td>
<td>2.86</td>
<td>1.81</td>
<td>Australia</td>
<td>1998</td>
</tr>
<tr>
<td>2</td>
<td>5.15</td>
<td>2.87</td>
<td>1.77</td>
<td>Australia</td>
<td>1999</td>
</tr>
<tr>
<td>3</td>
<td>5.06</td>
<td>2.94</td>
<td>1.88</td>
<td>Australia</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>5.07</td>
<td>2.95</td>
<td>2.07</td>
<td>Australia</td>
<td>2001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4.57</td>
<td>3.11</td>
<td>2.15</td>
<td>Australia</td>
<td>2006</td>
</tr>
<tr>
<td>10</td>
<td>4.50</td>
<td>2.59</td>
<td>1.77</td>
<td>NewZealand</td>
<td>1998</td>
</tr>
<tr>
<td>11</td>
<td>4.28</td>
<td>2.65</td>
<td>1.64</td>
<td>NewZealand</td>
<td>1999</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3.96</td>
<td>3.09</td>
<td>2.20</td>
<td>NewZealand</td>
<td>2006</td>
</tr>
</tbody>
</table>

These data are in the *DAAGxtras* package:

```r
groovy
library(DAAGxtras)
```
Beer+Wine+Spirit, conditioning on Country

xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog, outer=FALSE, auto.key=list(columns=3))

NB: Code has been simplified; next slide has full details.
Beer+Wine+Spirit, conditioning on Country, with frills

grogplot <-
  xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog,
          outer=FALSE, auto.key=list(columns=3))

Send output from update() to command line, causing ‘printing’

update(grogplot, ylim=c(0,5.5),
       xlab="", ylab="Amount consumed (per person)",
       par.settings=simpleTheme(pch=c(1,3,4)))

Alternatively, spell out the details – ‘print’ explicitly

frillyplot <-
  update(grogplot, ylim=c(0,5.5),
         xlab="", ylab="Amount consumed (per person)",
         par.settings=simpleTheme(pch=c(1,3,4)))
print(frillyplot)
Grouping of Points, and Columns in Parallel

Levels of a factor

Overplot (a single panel)

Separate panels (conditioning)

Beer ~ Year, groups=Country

Beer ~ Year | Country

Columns in parallel

Beer+Wine+Spirit ~ Year,

outer=FALSE

outer=TRUE
xyplot(Beer + Wine + Spirit ~ Year, 
groups=Country, outer=TRUE, 
data=grog, 
auto.key=list(columns=2))

xyplot(Beer ~ Year | Country, data=grog)
<table>
<thead>
<tr>
<th>Year</th>
<th>Beer + Wine + Spirit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
</tr>
</tbody>
</table>

```
xyplot(Beer+Wine+Spirit ~ Year, 
groups=Country, outer=TRUE, 
data=grog, 
auto.key=list(columns=2))
```
```
# xyplot(Beer ~ Year | Country, data=grog)
```
xyplot(Beer+Wine+Spirit ~ Year | Country, outer=TRUE, data=grog)
library(DAAG)  # Make the tinting datset available

Separate the factor names with *, e.g.

```r
## 2 conditioning factors
xyplot(csoa ~ it | sex * agegp, data=tinting)
```

3 conditioning factors

```r
xyplot(csoa ~ it | sex * agegp * target, data=tinting)
```

3 conditioning factors; all panels on one page

```r
xyplot(csoa ~ it | sex * agegp * target, data=tinting, 
       layout=c(4,2), aspect=1)
```

Use `layout` to specify the columns × rows × pages layout.

Use `aspect=1` for a square plotting region (c.f. also `aspect="xy"`)
Lattice parameter settings

1. Changes to points and line settings (a change of ‘theme’) are readily made using the function `simpleTheme()` (in recent versions of `lattice`).

2. Axis, axis tick, tick label and axis label settings are readily made using the argument `scales` in the function call.

3. Lattice objects can be created, then updated – use `update()`.

4. Note also the arguments `aspect` (aspect ratio) and `layout` (# rows × # columns × # pages).

5. The `type` argument can specify any combination of `p` (points), `l` (lines), `b` (points & lines), `r` (regression lines) and `smooth` (a smooth curve). Set `span` to control the smoothness of any curve.
First use `simpleTheme()` to create a “theme” with the new settings:

```r
miscSettings <- simpleTheme(pch = 16, cex=1.25)
```

Alternatives are then:

(i) Supply the “theme” to `par.settings` in the function call.  
[This stores the settings with the object. These stored settings over-ride the global settings at the time of printing.]

```r
xyplot(Beer ~ Year | Country, data=grog, 
       par.settings=miscSettings)
```

(ii) Supply the “theme” to `trellis.par.set()`, prior to plotting:  
[Makes the change globally, until a new trellis device is opened]

```r
trellis.par.set(miscSettings)
xyplot(Beer ~ Year | Country, data=grog)
```
Axis, tick, tick label and axis label settings

```r
jobplot <- xyplot(Ontario+BC ~ Date, data=jobs)
## Half-length ticks, each quarter, Label years, Add key
tpos <- seq(from=95, by=0.25, to=97)
tlabs <- rep(c("Jan95", ",", "Jan96", ",", "Jan97"),
c(1,3,1,3,1))
update(jobplot, auto.key=list(columns=2), xlab="",
scales=list(tck=0.5, x=list(at=tpos, labels=tlabs)))
```

![Graph](image.png)
Now use logarithmic y-scale

```
logplot <-
  xyplot(Ontario+BC ~ Date, data=jobs, outer=TRUE, xlab="", scales=list(y=list(log="e")))
```

Natural log scale

```
update(logplot, scales=list(y=list(relation="sliced")))
```

Natural log scale, "sliced"
Now use logarithmic y-scale

```
logplot <- xyplot(Ontario+BC ~ Date, data=jobs, outer=TRUE, xlab="", scales=list(y=list(log="e")))
```
Now use logarithmic y-scale

```
update(logplot, scales=list(y=list(relation="sliced")))
```
Now use logarithmic y-scale
Adding regression lines (take a subset of the data)

Use `type = c("p", "r")` to get points & regression lines. Panels set apart `sex`, with `sport` set apart within panels.

First, take a subset

```r
aisBS <- subset(ais, sport %in% c("B_ball", "Swim"))
## ais$sport <- factor(ais$sport) # drop now or later!

## Code for axis labeling is omitted
xyplot(hg ~ rcc | sex, groups=sport[drop=TRUE],
       data=aisBS, type=c("p","r"))
```

Subsetting & plotting, all in one

```r
xyplot(hg ~ rcc | sex, groups=sport[drop=TRUE],
       data=ais, type=c("p","r"),
       subset = sport %in% c("B_ball", "Swim"))
```
# Code for axis labeling is omitted

```
xyplot(hg ~ rcc | sex, groups=sport[drop=TRUE],
   data=aisBS, type=c("p","r"))
```
Customized Panel & Strip Functions

Requires a customized panel function, plus strip function

Add regression lines; customize strip labels:

Blood cell to plasma ratio (%)

Red cell count ($10^{12} L^{-1}$)
Customized Panel & Strip Functions

Red cell count ($10^{12} \text{L}^{-1}$)

<table>
<thead>
<tr>
<th>Blood cell to plasma ratio (%)</th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>4.5</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>5.0</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

![Graph showing blood cell to plasma ratio and red cell count relationship]
Customized Panel & Strip Functions – Code

Add parallel regression lines; customize strip labels:

```r
xyplot(hg ~ rcc | sex, groups=sport[drop=TRUE], data=aisBS,
    auto.key=list(lines=TRUE, columns=2), aspect=1,
    strip=strip.custom(factor.levels=c("Female","Male")),
    panel=function(x, y, groups, subscripts, ...){
        panel.superpose(x,y, groups=groups,
            subscripts=subscripts, ...)
        b <- coef(lm(y ~ groups[subscripts] + x))
        lcol <- trellis.par.get()$superpose.line$col
        lty <- trellis.par.get()$superpose.line$lty
        panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
        panel.abline(b[1]+b[2], b[3], col=lcol[2],
            lty=lty[2])
    })
```
strip=strip.custom(factor.levels=c("Female","Male")),
"Female" replaces the first level name ("f"), & "Male" replaces "m"

panel=function(x, y, groups, subscripts, ...){
  panel.superpose(x,y, groups=groups,
                   subscripts=subscripts, ...)
  . . . .
  panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
  panel.abline(b[1]+b[2], b[3], col=lcol[2],
               lty=lty[2])
}

Inside panel functions, use functions such as panel.points(), panel.lines(), etc.

If there are groups, panel.xyplot() calls panel.superpose() Here, call panel.superpose() directly.
```r
# Calculate the regression estimates
b <- coef(lm(y ~ groups[subscripts] + x))
```

`x` and `y` are already subscripted. Use `groups[subscripts]`, however.

The user needs to get the point & line type

```r
lcol <- trellis.par.get()$superpose.line$col
lty <- trellis.par.get()$superpose.line$lty
```

Get default settings for colour and line type. The first two line types and colors will be required, one for each of the two calls to `abline()`.

Plotting expressions

```r
# This is extra to the code on the previous slide
xlab=expression("Red cell count \(10^{12}.\text{L}^{-1}\)")
ylab="Blood cell to plasma ratio (%)"
```
Interaction with Lattice Plots

- Following the plot, call `trellis.focus()`.
- In a multi-panel display, click on a panel to select it.
- Use functions such as `panel.points()`, `panel.text()`, `panel.abline()`, `panel.identify()`.
- Call `trellis.focus()`, as needed, to switch panels.
- When finished, call `trellis.unfocus()`.

Example

```r
xyplot(log(Time) ~ log(Distance), groups=roadORtrack, 
data=worldRecords)
trellis.focus()
trellis.focus()
## Now click (maybe twice) on a panel
panel.identify(labels=worldRecords$Place)
## Click near to points that should be labeled
## Right click to terminate
trellis.unfocus()
```
A lattice plot is made up of a number of “viewports”:

In the call to `trellis.focus()`, the default is (name="panel").

Other choices of name include "panel", "strip", name="legend" and "toplevel". For name="legend"; side should be indicated.

Use the call `trellis.panelArgs()` to identify the arguments that are available to panel functions following a call to `trellis.focus()`.

To highlight, or not to highlight:

For non-interactive use, turn off highlighting:

```r
trellis.focus(highlight=FALSE)
```

Further Information:

See the help pages for `trellis.focus()` and `trellis.vpname()`.
Lattice – further possibilities

- Axes and labels – some further customizations
  - Generation of tick labels in a date format: Section 3.2.
- More flexible keys: Section 3.4.
- Further lattice functions (there are many): 3.4.2 and 3.6.
- Much else: