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' graphs: Lattice functions do not "print" the graph. Conceptually:

```
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.
- Automatic key generation. More complex keys: See Section 3.4.
- 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

Sarkar, D. 2008. Lattice. Multivariate Data Visialization with R. Springer.

A Dataset that is Ideally Made for Lattice

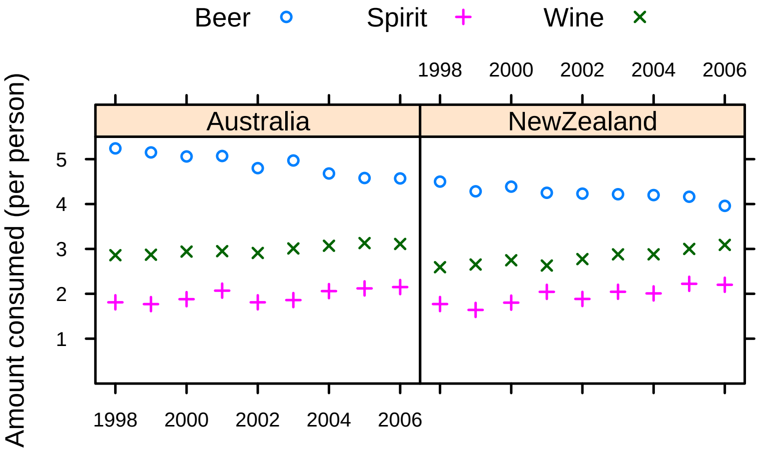
Australian & NZ apparent per person annual alcohol consumption of the pure alcohol content (in liters) of liquor products, 1998 to 2006.

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

These data are in the *DAAGxtras* package:

```
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'

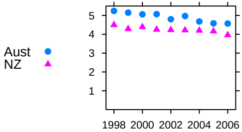
```
update(grogplot, ylim=c(0,5.5),
       xlab="", ylab="Amount consumed (per person)",
       par.settings=simpleTheme(pch=c(1,3,4)))
```

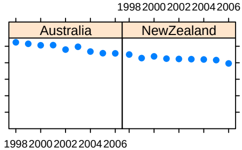
Alternatively, spell out the details – 'print' explicitly

```
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

	Overplot (a single panel)	Separate panels (conditioning)
Levels of a factor	`Beer ~ Year, groups=Country`	`Beer ~ Year | Country`



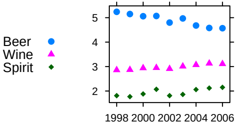


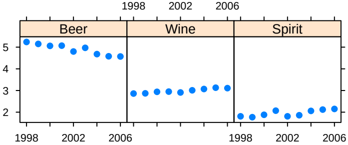
Columns in parallel

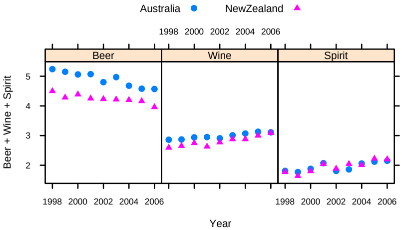
`Beer+Wine+Spirit ~ Year,`

`outer=FALSE`

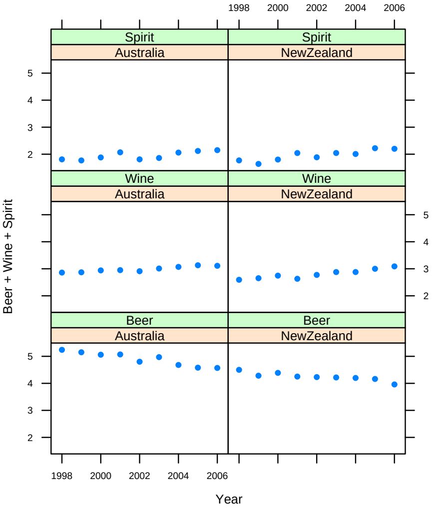
`outer=TRUE`



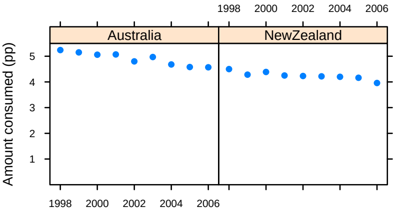




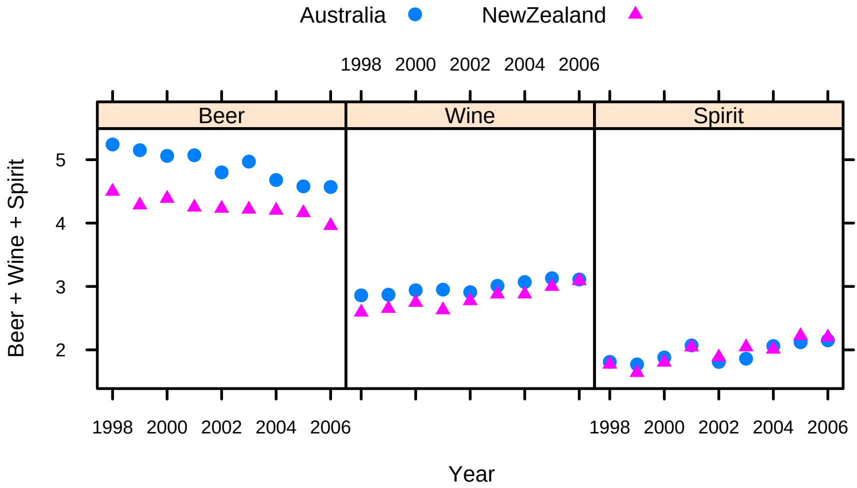

```
xyplot(Beer+Wine+Spirit ~ Year,
       groups=Country, outer=TRUE,
       data=grog,
       auto.key=list(columns=2))
```



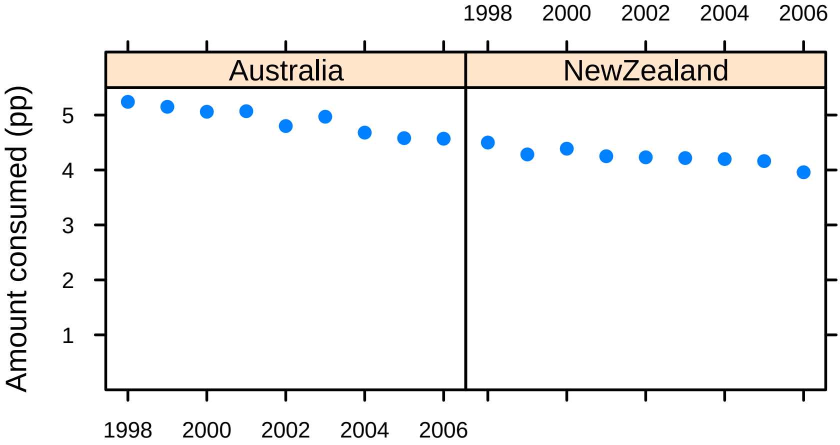

```
xyplot(Beer+Wine+Spirit ~ Year | Country,
       outer=TRUE, data=grog)
```



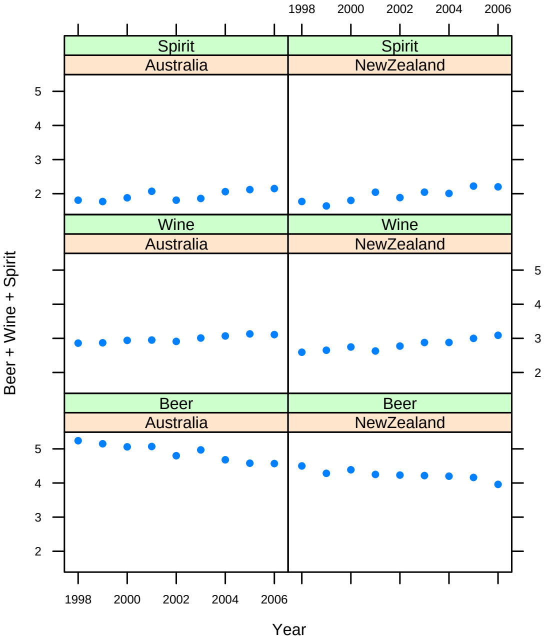

```
xyplot(Beer ~ Year | Country, data=grog)
```




```
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)
```

Conditioning on multiple factors

```
library(DAAG)    # Make the tinting datset available
```

Separate the factor names with `*`, e.g.

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

3 conditioning factors

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

3 conditioning factors; all panels on one page

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

Use `layout` to specify the columns $\times$ rows $\times$ 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 $\times$ # columns $\times$ # 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.

Use of simpleTheme() for Point & Line Settings

First use simpleTheme() to create a "theme" with the new settings:

```
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.]

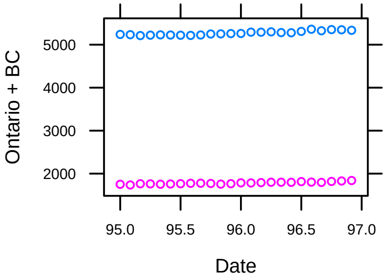
```
xyplot(Beer ~ Year | Country, data=grog,
       par.settings=miscSettings)
```

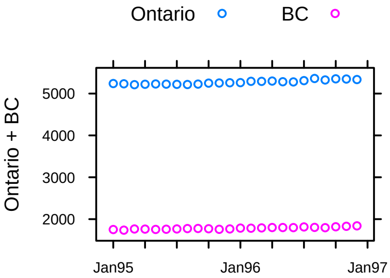
(ii) Supply the "theme" to trellis.par.set(), prior to plotting:
[Makes the change globally, until a new trellis device is opened]

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

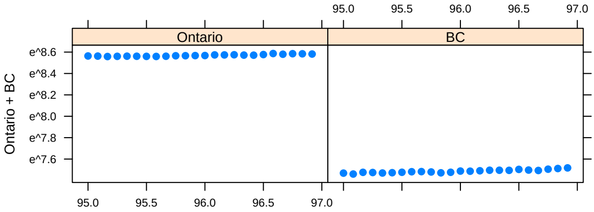
Axis, tick, tick label and axis label settings

```
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)))
```



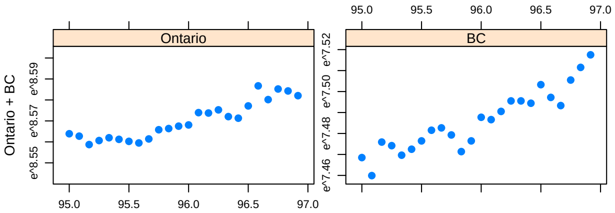


Now use logarithmic y-scale



Natural
log scale

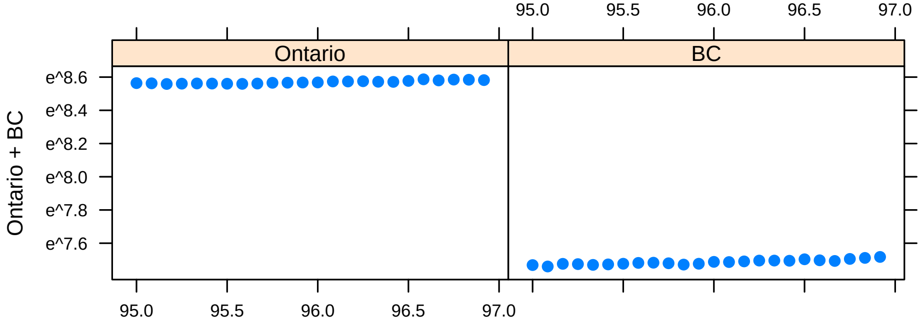
```
logplot <-
  xyplot(Ontario+BC ~ Date, data=jobs, outer=TRUE,
         xlab="", scales=list(y=list(log="e")))
```



Natural
log scale,
`"sliced"`

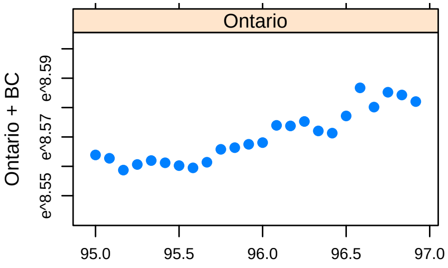
```
update(logplot, scales=list(y=list(relation="sliced")))
```

Now use logarithmic *y*-scale



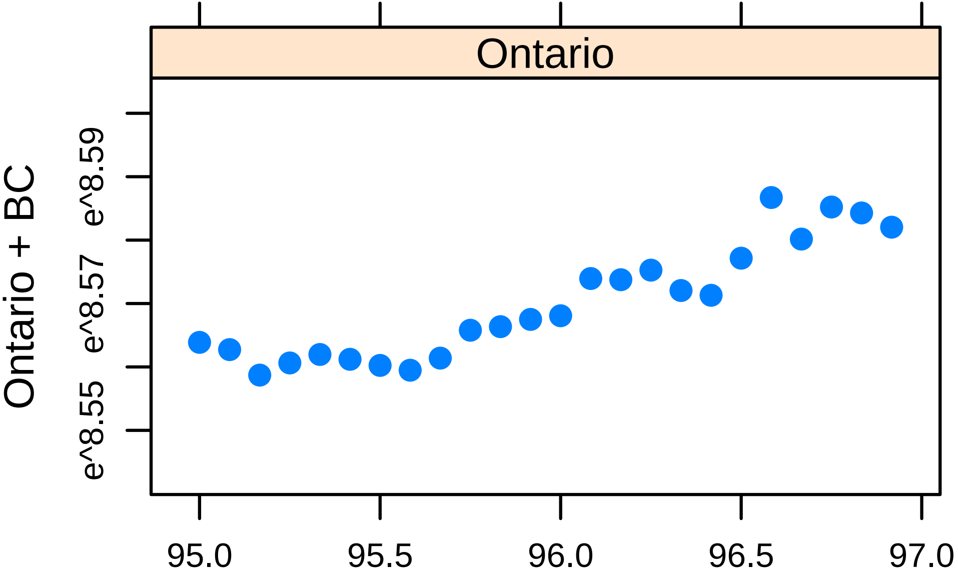

```r
logplot <-
  xyplot(Ontario+BC ~ Date, data=jobs, outer=TRUE,
         xlab="", scales=list(y=list(log="e")))
```

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.

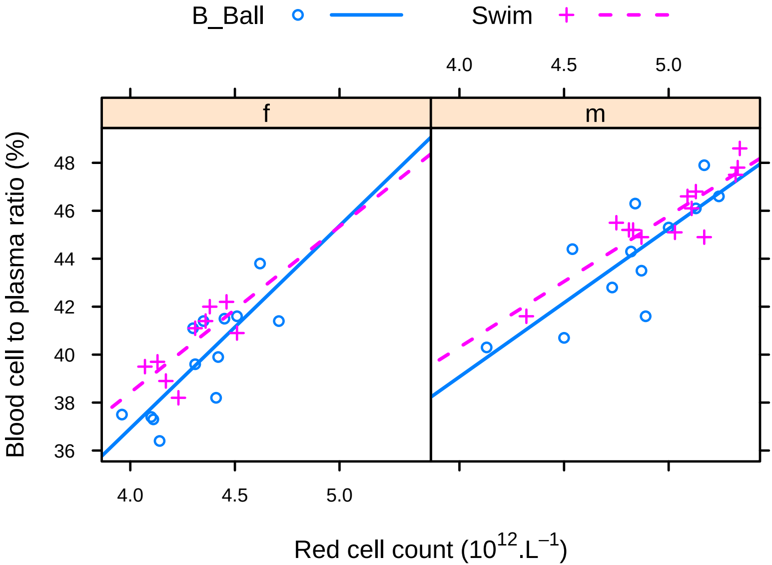
First, take a subset

```
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"))
```

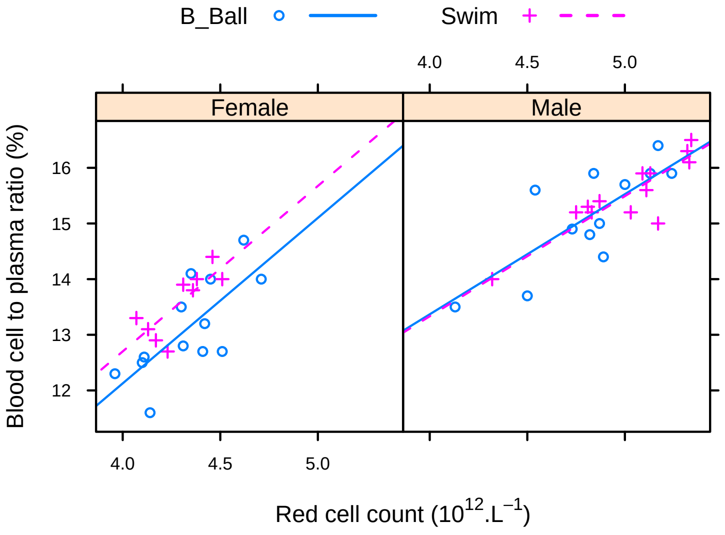
Subsetting & plotting, all in one

```
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"))
```

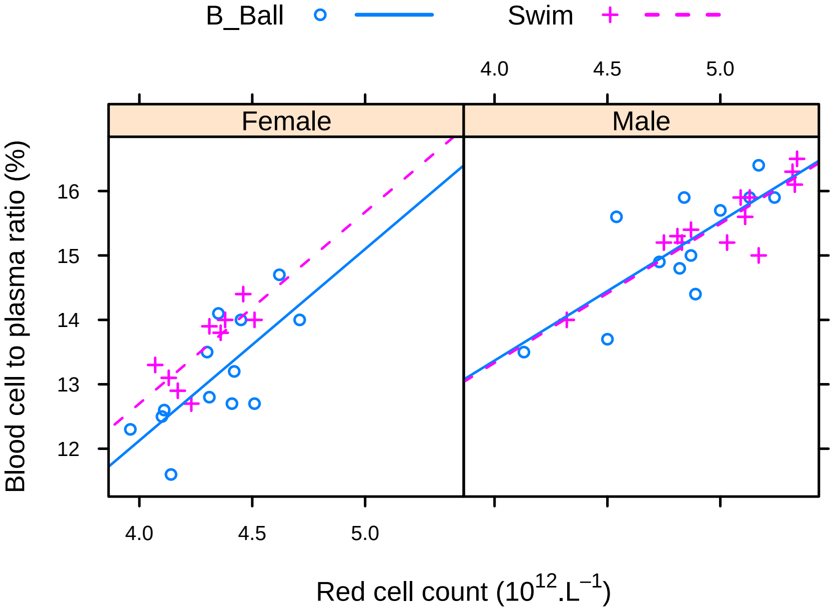
Customized Panel & Strip Functions



Add ∥ regression lines; customize strip labels:

Requires a customized panel function, plus strip function

Customized Panel & Strip Functions



Customized Panel & Strip Functions – Code

Add parallel regression lines; customize strip labels:

```
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])
         })
```

Customized Panel & Strip Functions – Notes on Code I

```
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.

Customized Panel & Strip Functions – Notes on Code II

```
## 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

```
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

```
## This is extra to the code on the previous slide
xlab=expression("Red cell count (10"^{12}*"."*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

```
xyplot(log(Time) ~ log(Distance), groups=roadORtrack,
       data=worldRecords)
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()
```

Focusing and Unfocusing – Further Notes

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:

```
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:
  Sarkar, D. 2008. Lattice. Multivariate Data Visialization with R. Springer.