Convenience Functions – One Set per Chapter

Each of chapters 1-9 has a set of accompanying convenience functions that can be used to reproduce graphs, or in a few cases tables, in that chapter. The package *modregR* includes source files that can be used to generate the functions. The text box has an example.

Loading and Using the Convenience Functions: Assuming that the package *modregR* has been installed, use the command `library()` to attach it. As an example, the following places the Chapter 1 functions in the workspace:

```
library(modregR)
 figs1path <- system.file('doc/figs1.R', package='modregR')
 source(figs1path)
```

A good way to proceed, for making sense of the R code, is to first run the function, thus:

```
 ## Run the code for Figure 1.1
 fig1.1()
```

Then look inside the function to see what code is used.

```
 ## Examine the code used to give Figure 1.1
 print(fig1.1)     # Or, just type fig1.1
```

Readers who are not totally at home with R may be wise to take as given figures where the code has some modest degree of complication.

Executing the Functions: There is a device that allows functions, once they have been loaded, to be executed, with any necessary intervening code. To set this up, enter at the command line:

```
 OpenSesame <- 7
```

Be warned that the functions for Chapter 6 take a modest time to execute – around $2\frac{1}{2}$ minutes on my well-configured machine. Remove the object `OpenSesame` to enable loading of functions without execution.

The discussion that follows will use the data on record times for Northern Ireland hill races, in the dataset `nihills` in the `DAAG` package. We will need, also, access to the `lattice` and `latticeExtra` packages. The following attaches these packages:

```
library(DAAG)
library(latticeExtra)
```

Base graphics versus lattice graphics: The R system has several styles of graphics. Here, base graphics (the older more traditional style of graphics) and lattice graphics will be used. There are important differences in the conventions that they follow. Base graphics generates plots directly. Lattice graphics functions (and ggplot2 graphics functions) create graphics objects. A graph is displayed when the object is supplied as argument to a `print()` or `plot()` command. Compare:

```
## Base graphics
plot(timef~time, log="xy", data=nihills,
      xlab="Time for males (h)", ylab="Time for females (h)")
```

```
## Lattice graphics
gph <- xyplot(timef~time, scales=list(log=2), data=nihills,
              type=c("p", "r"))
print(gph)
```

Base graphics versus lattice graphics – more extensive code

The following base graphics code plots record times for females against record times for males, based on data from the *nihills* package. It adda a regression line. It adds also the line $y = x$, now plotted as a dashed line:

```
plot(timef~time, log="xy",
     data=nihills,
     xlab="Time for males (h)",
     ylab="Time for females (h)")
form <- log10(timef)~log10(time)
logline <- lm(form, data=nihills)
abline(logline)
abline(a=0, b=1, lty=2)
```

Plot points, using a $\log_{10}$ scale on both x- and y-axes, and with meaningful x- and y- axis labels.

Set up regression formula (`form`).
Fit line ($\log_{10}$ scales).
Add fitted line to plot.
Add the line $y = x$.

Here is code that uses lattice graphics to give a roughly similar plot:

```
gph <- xyplot(timef~time,
              scales=list(log=2),
              data=nihills,
              type=c("p", "r"))
gph <- update(gph,
       xlab="Time for males (h)",
       ylab="Time for females (h)")
add <- layer(panel.abline(a=0, b=1,
                          lty=2))
gph <- gph + add
plot(gph)
```

Create an initial graphics object, with $\log_2$ axis scales. In `type=c("p","r")`, `"p"`, adds points and `"r"` adds a regression line.

Update the graphics object to have meaningful x- and y- axis labels.

Add the line $y = x$.

Plot the graph.

Here are alternative ways to create a function that does the same task. Options will be demonstrated for the use of lattice functions.

Place the code inside a wrapper function, thus:

```
graph1 <- function(){
    gph <- xyplot(timef~time, scales=list(log=2),
                  data=nihills, type=c("p", "r"))
    gph <- update(gph, xlab="Time for males (h)",
                  ylab="Time for females (h)")
    gph <- gph + layer(panel.abline(a=0, b=1, lty=2))
    plot(gph)
}
```

Then typing `graph1()` at the command line has the same effect as entering the code inside the function. The function has no arguments (in `function()`, there is nothing between the parentheses). It has no return value. The final statement in the function is a a plot statement, which does not return a value.

A function with arguments and return value:

```
graph2 <- function(form=timef~time, data=nihills){
    gph <- xyplot(form, scales=list(log=2),
                  data=data, type=c("p", "r"))
    gph <- update(gph, xlab="Time for males (h)",
                  ylab="Time for females (h)")
    gph <- gph + layer(panel.abline(a=0, b=1, lty=2))
    gph
}
```

Inside the function the graphics formula `timef` ∼ `time` has been replaced by `form`. The name dataset name `nihills` has been replaced by data. The default arguments set `form=timef time`, and `data=nihills`. These can be changed when the function is called, as the user wishes.

Typing `graph2()` at the command line causes the function to be executed, with the default arguments. The relevant object(s) (here `nihills`) must be in the search path. The final line has the value that is returned. If returned to the command line, the graph is plotted. Typing:
`gph <- graph2()`

stores the graphics object with the name `gph`, and no graph is plotted.

In the functions that accompany the chapters of this text, functions that create a single lattice graphics object will have the return object on the final line. If two lattice objects are created, for exmample `ghpA` and `ghpB`, the final line will be:

```
invisible(list(gphA, gphB))
```

Such functions have an argument `plotit`. If called with `plotit=TRUE`, both graphs will be plotted in some suitable layout.

Note well. Base graphics functions plot graphs. Lattice graphics functions create graphics objects. In order to obtain a graph, use `plot()` or `print()` with the lattice graphics object as argument. This invokes the `plot` (or `print`) method for a lattice graphics object.