Appendix A A sample session

The following session is intended to introduce to you some features of the R environment by using them. Many features of the system will be unfamiliar and puzzling at first, but this puzzlement will soon disappear.

Start R appropriately for your platform (see Appendix B [Invoking R], page 85).
: The R program begins, with a banner.

  (Within R code, the prompt on the left hand side will not be shown to avoid confusion.)

`help.start()`
: Start the HTML interface to on-line help (using a web browser available at your machine). You should briefly explore the features of this facility with the mouse.

  Iconify the help window and move on to the next part.

```
x <- rnorm(50)
y <- rnorm(x)
```
: Generate two pseudo-random normal vectors of x- and y-coordinates.

`plot(x, y)`
: Plot the points in the plane. A graphics window will appear automatically.

`ls()` See which R objects are now in the R workspace.

`rm(x, y)` Remove objects no longer needed. (Clean up).

`x <- 1:20` Make $x = (1, 2, \ldots, 20)$.

`w <- 1 + sqrt(x)/2`
: A 'weight' vector of standard deviations.

```
dummy <- data.frame(x=x, y= x + rnorm(x)*w)
dummy
```
: Make a *data frame* of two columns, x and y, and look at it.

```
fm <- lm(y ~ x, data=dummy)
summary(fm)
```
: Fit a simple linear regression and look at the analysis. With `y` to the left of the tilde, we are modelling y dependent on x.

```
fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)
summary(fm1)
```
: Since we know the standard deviations, we can do a weighted regression.

`attach(dummy)`
: Make the columns in the data frame visible as variables.

`lrf <- lowess(x, y)`
: Make a nonparametric local regression function.

`plot(x, y)`
: Standard point plot.

`lines(x, lrf$y)`
: Add in the local regression.

`abline(0, 1, lty=3)`
: The true regression line: (intercept 0, slope 1).

`abline(coef(fm))`
: Unweighted regression line.

`abline(coef(fm1), col = "red")`
: Weighted regression line.

`detach()`
: Remove data frame from the search path.

```
plot(fitted(fm), resid(fm),
     xlab="Fitted values",
     ylab="Residuals",
     main="Residuals vs Fitted")
```
: A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?

`qqnorm(resid(fm), main="Residuals Rankit Plot")`
: A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)

`rm(fm, fm1, lrf, x, dummy)`
: Clean up again.

The next section will look at data from the classical experiment of Michelson to measure the speed of light. This dataset is available in the `morley` object, but we will read it to illustrate the `read.table` function.

```
filepath <- system.file("data", "morley.tab" , package="datasets")
filepath
```
: Get the path to the data file.

`file.show(filepath)`
: Optional. Look at the file.

```
mm <- read.table(filepath)
mm
```
: Read in the Michelson data as a data frame, and look at it. There are five experiments (column `Expt`) and each has 20 runs (column `Run`) and `sl` is the recorded speed of light, suitably coded.

```
mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
```
: Change `Expt` and `Run` into factors.

`attach(mm)`
: Make the data frame visible at position 3 (the default).

`plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")`
: Compare the five experiments with simple boxplots.

```
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
```
: Analyze as a randomized block, with 'runs' and 'experiments' as factors.

```
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
```
: Fit the sub-model omitting 'runs', and compare using a formal analysis of variance.

```
detach()
rm(fm, fm0)
```
: Clean up before moving on.

We now look at some more graphical features: contour and image plots.

```
x <- seq(-pi, pi, len=50)
y <- x
```
: x is a vector of 50 equally spaced values in $-\pi \leq x \leq \pi$. y is the same.

`f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))`
: f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function $\cos(y)/(1+x^2)$.

```
oldpar <- par(no.readonly = TRUE)
par(pty="s")
```
Save the plotting parameters and set the plotting region to "square".

```
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
```
Make a contour map of f; add in more lines for more detail.

`fa <- (f-t(f))/2`
: `fa` is the "asymmetric part" of f. (`t()` is transpose).

`contour(x, y, fa, nlevels=15)`
: Make a contour plot, . . .

`par(oldpar)`
: . . . and restore the old graphics parameters.

```
image(x, y, f)
image(x, y, fa)
```
Make some high density image plots, (of which you can get hardcopies if you wish), . . .

`objects(); rm(x, y, f, fa)`
: . . . and clean up before moving on.

R can do complex arithmetic, also.

```
th <- seq(-pi, pi, len=100)
z <- exp(1i*th)
```
`1i` is used for the complex number i.

```
par(pty="s")
plot(z, type="l")
```
Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

`w <- rnorm(100) + rnorm(100)*1i`
: Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts . . .

`w <- ifelse(Mod(w) > 1, 1/w, w)`
: . . . and to map any outside the circle onto their reciprocal.

```
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+",xlab="x", ylab="y")
lines(z)
```
All points are inside the unit circle, but the distribution is not uniform.

```
w <- sqrt(runif(100))*exp(2*pi*runif(100)*1i)
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(z)
```
The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

`rm(th, w, z)`
: Clean up again.

`q()`
: Quit the R program. You will be asked if you want to save the R workspace, and for an exploratory session like this, you probably do not want to save it.