Appendix A A Sample session

The following session is intended to introduce you to some features of the R environment. Many features will be unfamiliar and puzzling at first, but this puzzlement will soon disappear. The commands are stored in executable form in the file `samplesession.r`.

```
                    amp; Login and start R.
help.start()
                    amp; Start the HTML interface to on-line help (using a web
                    amp; browser available at your machine). You should briefly
                    amp; explore the features of this facility with the mouse.
                    amp; Iconify the help window and move on to the next part.
x <- rnorm(50)
y <- rnorm(50)
                    amp; Generate two pseudo-random normal vectors of x−
                    amp; and y−coordinates.
plot(x,y)
                    amp; Plot the points in the place. A graphics window will
                    amp; appear automatically.
ls()                amp; See which R objects are now in the R workspace.
rm(x,y)             amp; Remove objects no longer needed. (Clean up).
x <- 1:20           amp; Make x = (1, 2, ..., 20).
w <- 1 + sqrt(x)/2
                    amp; A 'weight' vector of standard deviations.
dummy <- data.frame(x=x, y=x + rnorm(x)*w)
dummy               amp; Make a data frame of two columns, x and y, and look
                    amp; at it.
fm <- lm(y ~ x, data=dummy)
summary(fm)
                    amp; Fit a simple linear regression of y on x and look at the
                    amp; analysis.
fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)
summary(fm1)
                    amp; Since we know the standard deviations, we can do a
                    amp; weighted regression.
attach(dummy)
                    amp; Make the columns in the data frame visible as variables.
lrf <- lowess(x,y)
                    amp; Make a nonparametric local regression function.
plot(x,y)
                    amp; Standard point plot.
lines(x, lrf$y)
                    amp; Add in the local regression.
abline(0, 1, lty=3)
                    amp; The true regression line; (intercept 0, slope 1).
```

```
abline(coef(fm))
                    amp; Unweighted regression line.
abline(coef(fm1), col="red")
                    amp; Weighted regression line.
detach()            amp; Remove data frame from the search path.
fitted(fm), resid(fm),
           xlab=    amp; "Fitted values",
           ylab=    amp; "Residuals",
           main=    amp; "Residuals vs Fitted",
                    amp; A standard regression diagnostic plot to check for
                    amp; heteroscedasticity. Can you see it?
qqnorm(resid(fm), main="Residuals Rankit Plot")
                    amp; A Normal scores plot to check for skewness, kurtosis
                    amp; and outliers. (Not very useful here.)
rm(fm, fm1, lrf, x, dummy)
                    amp; Clean up again.
```

The next section will look at data from the classical experiment of `Michaelson and Morley to measure the speed of light.`

```
data(morley)
mm <- morley
mm                  amp; Copy the data to a data frame mm, and look at it.
                    amp; There are five experiments (column Expt) and each
                    amp; has 20 runs (column Run) and Speed) is the
                    amp; recorded speed of light, suitably coded.
mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
                    amp; Change Expt and Run into factors.
attach(mm)
                    amp; Make the data frame visible at position 2 (the default).
plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
                    amp; Compare the five experiments with simple boxplots.
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
                    amp; Analyze as a randomized block, with 'runs' and
                    amp; 'experiments' as factors.
fm0 <- update(fm, .  ~ .  - Run)
anova(fm0,fm)
                    amp; Fit the sub-model omitting 'runs', and compare using a
                    amp; formal anlaysis of variance
detach()
rm(fm,fm0)
                    amp; 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              amp; x is a vector of 50 equally spaced values in -pi ≤ x ≤ π.
                    amp; y is the same.
```

```
f <- outer(x, y, function(x, y) cos(y)/(1 + x2))
                    amp; f is a square matrix, withn rows and columns indexed
                    amp; by x and y respectively, of values of the function
                    amp; cos(y)/(1 + x^2).
oldopar <- par(no.readonly = TRUE)
par(pty="s")
                    amp; Save the plotting parameters and set the plotting region
                    amp; to "square".
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
                    amp; Make a contour map of f; add in more lines for more
                    amp; detail.
fa <- (f-t(f))/2
                    amp; fa is the "asymmetric part" of f. (t() is transpose).
contour(x, y, fa, nint=15)
                    amp; Make a contour plot, ...
par(oldpar)
                    amp; ... and restore the old graphics parameters.
image(x, y, f)
image(x, y, fa)
                    amp; Make some high density image plots, (of which you can
                    amp; get hardcopies if you wish), ...
objects(); rm(x, y, f, fa)
                    amp; ... and clean up before moving on.
```

R can do complex arithmetic, also.

```
th <- seq(-pi, pi, len=100)
z <- exp(1i*th)
                    amp; 1i is used for the complex number i.
par(pty="s")
plot(z, type="l")
                    amp; Plotting complex arguments means plot imaginary
                    amp; versus real parts. This
                    amp; should be a circle.
w <- rnorm(100) + rnorm(100)*1i
                    amp; Suppose we want to sample points within the unit
                    amp; circle. One method would be to take complex numbers
                    amp; with standard normal real and imaginary parts ...
w <- ifelese(Mod(w) > 1, 1/w, w)
                    amp; ... and to map any outside the circle onto their
                    amp; reciprocal.
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(x)
                    amp; All points are inside the unit circle, but the
                    amp; 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)
                amp; The second method uses the uniform distribution.
                amp; The points should now look more evenly spaced over
                amp; the disc.
rm(th, w, z)
                amp; Clean up again.
q()             amp; Quit the R program. You will be asked if you want to
                amp; save the R workspace, and for an exploratory session
                amp; like this, you probably do not want to save it.
```