Introduction to R: Computer Lab 1 (Based on the R help manual)

R is a language and environment for statistical computing and graphics. R is similar to the award-winning S system, which was developed at Bell Laboratories by John Chambers et al. It provides a wide variety of statistical and graphical techniques.

Vectors and assignment

To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

Notice that the assignment operator

```
<-
```

is not the usual = operator, which is reserved for another purpose. It consists of the two characters $<$ ("less than") and $-$ ("minus") occurring strictly side-by-side and it 'points' to the object receiving the value of the expression.

The further assignment

```
> y <- c(x, 0, x)
```

would create a vector y with 11 entries consisting of two copies of x with a zero in the middle place.

Operating with vectors:

```
> x <- c(1, 3, 5)
> y <- c(2, 4, 6)
> v <- 2*x + y + 1
```

The elementary arithmetic operators are the usual $+, -, *, /$and $\hat{}$for raising to a power. In addition all of the common arithmetic functions are available. log, exp, sin, cos, tan, sqrt, and so on, all have their usual meaning. max and min select the largest and smallest elements of a vector respectively. range is a function whose value is a vector of length two, namely c(min(x), max(x)). length(x) is the number of elements in x, sum(x) gives the total of the elements in x, and prod(x) their product.

Two statistical functions are mean(x) which calculates the sample mean, which is the same as sum(x)/length(x), and var(x) which gives

```
sum((x-mean(x))^2)/(length(x)-1)
```

or sample variance. If the argument to var() is an n-by-p matrix the value is a p-by-p sample covariance matrix got by regarding the rows as independent p-variate sample vectors.

sort(x) returns a vector of the same size as x with the elements arranged in increasing order; however there are other more flexible sorting facilities available (see order() or sort.list() which produce a permutation to do the sorting).

Note that max and min select the largest and smallest values in their arguments, even if they are given several vectors.

unique(x) gives the unique elements of x.

Matrix

To create a 3×4 matrix with all elements 0,

```
> x <- matrix(0, nrow=3, ncol=4)
```

Matrices can be built up from other vectors and matrices by the functions cbind() and rbind(). Roughly cbind() forms matrices by binding together matrices horizontally, or column-wise, and rbind() vertically, or row-wise.

In the assignment

```
> x <- cbind(arg1, arg2, arg3, ...)
```

For example,

```
> x1 <- c(1,2,3)
> y1 <- c(7,8,9)
> xy1 <- cbind(x1,y1)
```

To take a look at X1:

```
> xy1
     x1 y1
[1,]  1  7
[2,]  2  8
```

```
[3,]  3  9
```

The arguments to cbind() must be either vectors of any length, or matrices with the same column size, that is the same number of rows. The result is a matrix with the concatenated arguments arg1, arg2, ... forming the columns.

If some of the arguments to cbind() are vectors they may be shorter than the column size of any matrices present, in which case they are cyclically extended to match the matrix column size (or the length of the longest vector if no matrices are given).

The function rbind() does the corresponding operation for rows. In this case any vector argument, possibly cyclically extended, are of course taken as row vectors.

Suppose **x1** and x2 have the same number of rows. To combine these by columns into a matrix X, together with an initial column of 1s we can use

```
> x1 <- c(2,3,5)
> x2 <- c(-1,0,-4)
> x <- cbind(1, x1, x2)
```

The result of rbind() or cbind() always has matrix status. Hence cbind(x) and rbind(x) are possibly the simplest ways explicitly to allow the vector x to be treated as a column or row matrix respectively.

To get the second column of the matrix:

```
> xc2 <- x[,2]
```

To get the 3rd row of the matrix:

```
> xr3 <- x[3,]
```

Some simple examples

```
x <- rnorm(50)
y <- rnorm(50)
```

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, ..., 20).

```
w <- 1 + sqrt(x)/2
```

A 'weight' vector of standard deviations.

```
foo <- data.frame(x=x, y= x + rnorm(x)*w)
foo
```

Make a data frame of two columns, x and y, and look at it.

```
fm <- lm(y ~ x, data=foo)
summary(fm)
```

Fit a simple linear regression of y on x and look at the analysis.

```
fm1 <- lm(y ~ x, data=foo, weight=1/w^2)
summary(fm1)
```

Since we know the standard deviations, we can do a weighted regression.

```
attach(foo)
```

Make the columns in the data frame visible as variables.

```
lrf <- lowess(x, y)
```

Make a nonparametric local regression function.

```
plot(x, y)
```

Standard scatter 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(fm1), col = "red")
```

Weighted regression line.

```
abline(coef(fm))
```

Unweighted 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, foo)
```

Clean up again.