

Introduction to R – Part I

ST810

February 9, 2009

Description of R

- ▶ An environment for statistical computing and graphics
- ▶ R was originally created by Ross Ihaka and Robert Gentleman (hence the name R) at the University of Auckland, New Zealand.
- ▶ Very similar to Splus.
- ▶ Operates in Windows, Mac, or Unix.
- ▶ We'll focus on Windows.

Advantages of R

- ▶ It's free!
- ▶ Very conducive to matrix manipulation
- ▶ Easier graphical interface than SAS
- ▶ Easy to load other user's code (contributed packages)

Disadvantages of R

- ▶ It's slow relative to Fortran or C
- ▶ Other user's code can have errors, be careful!
- ▶ Not as easy to clean and manipulate data as SAS

Downloading R

- ▶ The main website is <http://www.r-project.org/>
- ▶ Click “R version 2.8.1 has been released”
- ▶ Download R-2.8.1.tar.gz
- ▶ Unzip this file and open the “read me” file for further instruction.

Getting help

- ▶ If you need more information about a function, say “lm”, enter

`>?lm`

- ▶ To search for help on a specific topic use R's search page www.r-project.org/search.html.
- ▶ For more general information use R's manuals, www.cran.r-project.org/manuals.html.

Running R in Windows

- ▶ Code can be entered one-line-at-a-time at the command prompt.
- ▶ Batches of code can also be saved in a file, say `code.R`, and run all at once.
- ▶ Click File/Source R Code to browse for the code
- ▶ Or, if you know the code's location, at the command prompt enter the code

```
> source("K : /code.R")
```

Saving output

- ▶ To view a list of objects in the current workspace, enter

```
> ls()
```

- ▶ To save these objects, click File/Save workspace, and enter a name for the workspace.

- ▶ Or at the command prompt enter

```
> save.image("K : /mywork.RData")
```

- ▶ Workspaces can be large. It's often wise to delete individual objects that are no longer needed (e.g., the matrix X) before saving the workspace, i.e.,

```
> rm(X)
```

Data forms - Scalars

- ▶ Scalar variables are created like

$$a < -6$$

- ▶ Usual calculator commands apply: $+$, $-$, $\sin()$, $\log()$, $\exp()$, $\text{sqrt}()$, etc.
- ▶ To assign a missing value, enter

$$a < -NA$$

- ▶ To check if an observation is missing, enter

$$\text{is.na}(a)$$

- ▶ Rounding (2 is the number of decimal places, “;” separates commands)

$$> a < -pi; \text{round}(a, 2)$$
$$[1] 3.14$$

Data forms - vectors

Here are some vector commands and R output (“;” separates two commands).

```
> b <- -c(1, 6, 3); b
```

```
[1] 1 6 3
```

```
> b <- -1 : 7; b
```

```
[1] 1 2 3 4 5 6 7
```

```
> b <- -rep(1, 4); b
```

```
[1] 1 1 1 1
```

```
> b <- -seq(0, 1, length = 4); b
```

```
[1] 0.0000000 0.3333333 0.6666667 1.0000000
```

```
> b[2]; b[c(2, 4)]
```

```
[1] 0.3333333
```

```
[1] 0.3333333 1.0000000
```

```
> c <- matrix(1 : 6, 2, 3); c
     [, 1] [, 2] [, 3]
[1,]  1  3  5
[2,]  2  4  6
> c[2, 2]; c[2, 3]; dim(c)
[1] 4
[1] 6
[1] 2 3
```

- ▶ `> diag(3)` creates a 3×3 identity matrix.
- ▶ `> diag(1 : 3)` creates a 3×3 diagonal matrix with diagonal elements 1, 2, and 3.
- ▶ `> cbind(a, b)` creates a two-column matrix with vector *a* in

Data forms - arrays

An array is a matrix with more than two dimensions, e.g.,

```
> d <- array(1 : 12, c(2, 2, 3))
```

```
> d[2, 1, 2]
```

```
[1] 6
```

```
> dim(d)
```

```
[1] 2 2 3
```

A list is a collection of scalars, matrices, or arrays.

```
> a <- 1 : 3
> b <- "hi, my name is joe"
> l <- list(part1 = a, textpart = b)
> l$part1
[1] 1 2 3
> l$part1[3]
[1] 3
> l$textpart
[1] "hi, my name is joe"
```

```
d <- -matrix(1 : 4, 2, 2); d
```

```
[1,] 1 3
```

```
[2,] 2 4
```

- ▶ Scalar multiplication, $> 2 * d$

```
[1,] 2 6
```

```
[2,] 4 8
```

- ▶ Operations are applied to each element, $> exp(d)$

```
[1,] 2.718282 20.08554
```

```
[2,] 7.389056 54.59815
```

Matrix manipulations

- ▶ Matrix addition (assuming $\dim(X1) = \dim(X2)$): `> X1 + X2`
- ▶ Matrix multiplication (assuming it's well-defined):
`> X1% * %X2`
- ▶ Element-by-element multiplication (assuming $\dim(X1) = \dim(X2)$): `> X1 * X2`
- ▶ Matrix inversion (assuming X is square): `> solve(X)`.
- ▶ Transpose: `> t(X)`.
- ▶ Spectral decomposition: `> eig(X)`.

Probability distributions

- ▶ To generate the scalar $X \sim N(6, 2)$,

> $X < -rnorm(1, 6, 2)$

- ▶ To generate the vectors $X_1, \dots, X_{20} \stackrel{iid}{\sim} N(6, 2)$,

> $X < -rnorm(20, 6, 2)$

- ▶ To generate $X_{ij} \stackrel{iid}{\sim} N(6, 2)$, $i = 1, \dots, 10$, $j = 1, \dots, 20$,

> $X < -matrix(rnorm(200, 6, 2), 10, 20)$

- ▶ There are many other distributions: $rt()$, $rgamma()$, $rexp()$, etc.

- ▶ $dnorm$, $pnorm$, $qnorm$ give the density, distribution, and quantile function, respectively, for Gaussian variables.

Linear regression in R

First let's generate some data:

- ▶ `int <- rep(1, 100)`
- ▶ `> x1 <- rnorm(100, 0, 1)`
- ▶ `> x2 <- rnorm(100, 0, 1)`
- ▶ `> y <- rnorm(100, int + 2 * x1 - 2 * x2, 2)`
- ▶ Make the design matrix, `> X <- cbind(int, x1, x2)`.

First, let's do linear regression by hand,

```
> beta.hat <- solve(t(X)% * %X)% * %t(X)% * %y
> beta.hat
int    0.9212442
x1     1.9646612
x2    -1.8708764
```

Linear regression using the “lm” function

- ▶ R also has a function to perform linear regression,

$$fit <- lm(y \sim x1 + x2)$$

- ▶ *fit* is a list with several outputs from the linear regression.
- ▶ `> plot(fit)` gives some diagnostics
- ▶ `fit$residuals` is a vector of residuals
- ▶ `> summary(fit)` gives the results

	Estimate	Std Error	t value	Pr(> t)
Intercept	0.9212	0.2216	4.157	6.96e-05
x1	1.9647	0.2113	9.299	4.38e-15
x2	-1.8709	0.2131	-8.779	5.78e-14

Importing/exporting data

- ▶ The function `read.table()` loads tab delimited files.
- ▶ I prefer `read.csv()` which loads comma delimited files.
- ▶ For example, if “datafile.csv” has 101 rows and 10 columns, and the first row consists of variable names,

```
> data <- read.csv("K : /datafile.csv", header = TRUE)
> dim(data)
[1] 100 10
```

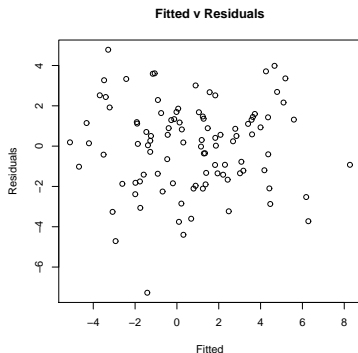
- ▶ Export data using `write.table()` or `write.csv()`.

Summarizing data

- ▶ `> mean(x)` gives the mean of the object (vector or matrix) x .
- ▶ There are many other functions: `median()`, `sd()`, `quantile()`, etc.
- ▶ `> table(x)` gives the frequency of each unique value of x , similar to `proc freq` in SAS.
- ▶ If x is a matrix, `mean(x[, 1])` gives the mean of the first column and `mean(x[1,])` gives the mean of the first row.
- ▶ If x is a matrix, `apply(x, 2, mean)` produces a vector of column means and `apply(x, 1, mean)` produces a vector of row means (mean can be replaced with `sd`, `median`,...).

Graphics - scatter plot

- ▶ $plot(x, y)$ makes a scatter plot of vectors x and y .
- ▶ For example, here's a plot of fitted values versus residuals for the linear regression,
 - > `plot(fit$fitted, fit$residuals, xlab = "Fitted", ylab = "Residuals", main = "fitted v residuals")`

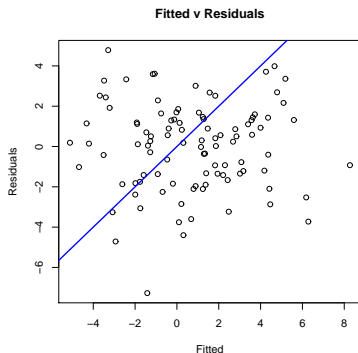


Graphics - scatter plot

- ▶ Many options are available. For example, `plot(x, y, col = 2)` makes the points red and `plot(x, y, type = "l")` connects the points.
- ▶ For a complete list of options, enter `>?par`.
- ▶ To save the plot as a postscript, you can right click on the plot and scroll to “save as postscript”.
- ▶ Or you can enter the commands:
 1. Specify plot location: `> postscript("K : /plot1.ps")`
 2. Make the plot: `> plot(x, y)`
 3. Tell R to finish the plot: `> dev.off()`

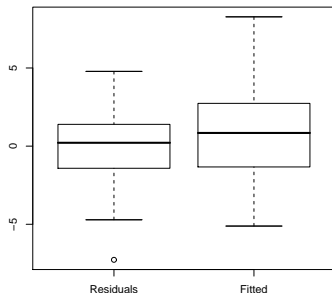
Graphics - scatter plot

- ▶ `points()`, `lines()`, and `polygon()` add points, lines, and polygons respectively, to the current plot.
- ▶ For example,
`lines(c(-100, 100), c(-100, 100), col = 4, lwd = 2)`.



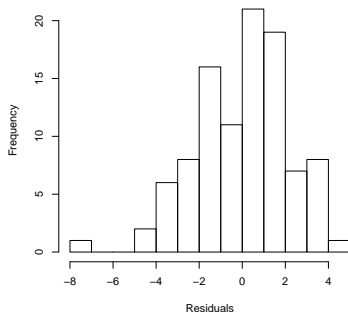
Graphics - boxplot

- ▶ `> boxplot(fit$residuals)` makes a boxplot of the residuals.
- ▶ `> boxplot(fit$residuals, fit$fitted, names = c("Residuals", "Fitted"))` makes a boxplot of the residuals and fitted values.



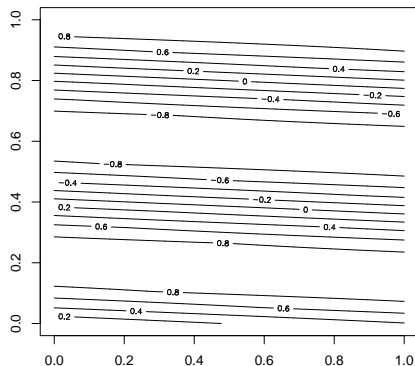
Graphics - histogram

- ▶ `hist()` makes a histogram.
- ▶ For example `hist(fit$residuals, breaks = 10, xlab = "Residuals", main = "")`



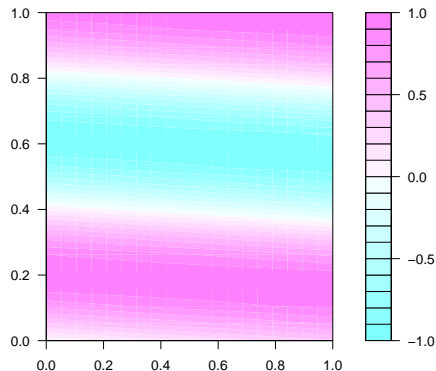
Graphics - contour plot

```
> x <- -matrix(sin(1 : 400/50), 20, 20)  
> contour(x)
```



Graphics - filled contour plot

```
> x <- matrix(sin(1 : 400/50), 20, 20)  
> filled.contour(x)
```



Next time

- ▶ Programming: loops, ifelse statements, functions
- ▶ Packages
- ▶ Making maps
- ▶ Optimization
- ▶ Calling other software: Fortran or WinBUGS
- ▶ Statistical methods