

Programming in R

Jane M. Horgan*

R programming is now a major resource for statistical analysis, research, and teaching and has an impressive suite of applications and packages. In this article, we introduce the basic features of the language including data entry, data description, graphical procedures, writing simple programs, and simulation. It is not the intention to provide an exhaustive list of features of the language, just enough to give a flavor of the structure of R, and how to get help to proceed further.

© 2011 Wiley Periodicals, Inc.

How to cite this article:

WIREs Comp Stat 2012, 4:75–84. doi: 10.1002/wics.183

Keywords: data entry analysis; graphical displays; simulation

WHAT IS R?

R is a data-analysis system that provides an environment for statistical analysis and graphics. It can be used just as a calculator, all the way up to producing elaborate graphics, performing simulations, and statistical modeling. It is in fact a complete object-oriented programming language, is open source, and available from the web under the General Public License (GPL) which allows free use. It exists for Microsoft Windows, Linux, and Unix platforms, and for Apple Macintosh (OS versions newer than 8.6). Unlike standard statistical packages such as SPSS and Minitab, which use point-and-click graphical-user interfaces, R is command driven; user type commands at a prompt, and R responds. The purpose of this article is to describe enough of the main features of R to enable the new user to get started. We first, in the next section, show how to download R and describe some of its basic operations, editing, and help procedures. The methods used to read and edit statistical data are discussed in Section ‘Data Entry’, and an introduction to data analysis is given in Section ‘Data Analysis’. Some of the graphical features of R are examined in Section ‘Graphical Displays’, and Section ‘Simulation’ deals with an example in queuing, to illustrate the powerful simulation tools available in R. We conclude with some suggestions for further reading.

BASICS

As a start we look at how to download R, and get it to perform simple calculations.

*Correspondence to: janemhorgan@gmail.com

School of Computing, Dublin City University, Dublin, Ireland

Installing R

R is obtained from the website called CRAN (Comprehensive R Archive Network), and is downloaded by proceeding as follows:

- Go to the Cran website at <http://cran.r-project.org/>
- Click ‘Download and Install R’
- Choose an operating system
- Choose the ‘base’ package
- Click on ‘Download R’
- Press the option ‘Run’

R is now installed.

To start, click on the R icon, or go to ‘Programs’, select R, and then click on the R icon. When the R program is started, and after it prints an introductory message on the screen, the interpreter prompts for input with ‘>’.

R as a Calculator

Expressions that are typed at the command prompt (>) are executed by the interpreter. For example:

```
6+7*3/2
```

returns

```
[1] 16.5
```

```
x <- 1:4
```

Here the integers 1, 2, 3, 4 are assigned to the vector x. To check the contents of x, type

x

which returns

```
[1] 1 2 3 4
```

```
xx <- x**2
```

causes each element in the vector x to be squared and stored in the vector xx. To examine the contents of xx, type

```
xx
```

which gives

```
[1] 1 4 9 16
```

To multiply a vector by a constant, type

```
X <- 10
prod1 <- X*x
prod1
[1] 10 20 30 40
```

Here the integer 10 is stored in X, and X*x causes each element of the vector x to be multiplied by 10.

Some points to note:

- <- is the *assignment operator*; in the illustration 'x <- 1:4', the vector (1, 2, 3, 4) is assigned to x. An alternative assignment operator is just '=';
- R is case sensitive; x and X represent different variables;
- Variable names can consist of any combination of lower and upper case letters, numerals, periods, and underscores, but cannot begin with a numeral or an underscore;
- All of the above examples of variables are numeric, but R supports many other types of data, such as nonnumeric strings and matrices.

The entities that R creates and manipulates are called *objects*. These include variables, arrays of numbers, strings, or functions. All objects are stored in what is known as the *workspace*.

Getting Help

The easiest way of getting help when working in the R environment is to click the *Help* button on the toolbar.

Alternatively you can type

```
help()
```

for on-line help, or

```
help.start()
```

for an HTML browser interface.

It could be helpful to look at some demonstrations of R by typing

```
demo()
```

which gives a list of available demonstrations. For example,

```
demo(graphics)
```

returns some examples of graphical procedures, along with the code used to implement them.

A more specific way of getting help is to type the name of the function you require. For example:

```
?read.table
```

This will provide details on the exact syntactic structure of the instruction 'read.table'.

If you do not know the name of the command, type some words from the topic as follows:

```
help.search("data.entry")
```

Check yourself what this will give.

DATA ENTRY

Before carrying out a statistical analysis, it is necessary to get the data into the computer. How this is done varies depending on the amount of data involved. We illustrate the various options of data entry with the *Anscombe Quartet*¹ given in Table 1. It consists of four data sets (x1, y1), (x2, y2), (x3, y3), (x4, y4), each consisting of two variables in each of which there are 11 observations.

Reading and Displaying Data on Screen

A small data set may be entered directly from the screen. It is usually stored as a vector, which is essentially a list of numbers. To input the x1 values given in Table 1 from the screen, type

```
x1 <- c(10, 8, 13, 9, 11, 14, 6, 4, 12,
        7, 5)
```

The construct c(...) is used to define a vector containing the data points. These are then assigned to a vector called x1. Similarly for y1, type

```
y1 <- c(8.04, 6.95, 7.58, 8.81, 8.33,
        9.96, 7.24, 4.26, 10.84,
        4.82, 5.68)
```

TABLE 1 | The Anscombe Quartet

x1	y1	x2	y2	x3	y3	x4	y4
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.10	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.10	4	5.39	19	12.50
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

Entering Data from a File

When the data set is large, it is better to set up a text file and access the data from this, rather than to enter it directly from the screen. For example, if the data in Table 1 are stored in a file called *anscombe.txt* in the G directory and data subdirectory, the file can be read into R using

```
anscombe <- read.table(
  ('G:/data/anscombe.txt', header = T)
```

Here `header = T` specifies that the first line is a header, in this case containing the names of variables. Notice that the forward slash (/) is used in the filename, not backslash (\) which would be expected in the windows environment. The backslash has itself a meaning within R, and cannot be used in this context.

In R, this type of data set is stored in what is referred to as a *data frame*, which is an object with rows and columns. Equivalently it is a list of vectors of the same length; the columns denote the variables, while the rows are the observations on the variables. The `read.table` instruction above assigns the data to a data frame called *anscombe*.

The convention for accessing the column variables is to use the name of the data frame followed by the name of the relevant column. For example:

```
anscombe$x1[5]
```

returns

```
[1] 11
```

which is the 5th observation in the column labeled x1.

An easier way of doing this is to type

```
attach(anscombe)
```

After `attach` there is no need to use the data frame name. Now

```
x1[5]
```

returns

```
[1] 11
```

`read.table` assumes that the data in the text file are separated by spaces, as in Table 1. Other forms include:

`read.csv`, used when the data points are separated by commas;

`read.csv2`, used when the data are separated by semicolons.

Spread Sheets

It is also possible to enter data into a spreadsheet and store it in a data frame as follows:

```
anscombe <- data.frame()
fix(anscombe)
```

This brings up a blank spread sheet called *anscombe*, and the user may then enter the variable labels and the variable values. When finished entering the data, right click and close creates a data frame *anscombe* in which the new information is stored.

Editing

If you subsequently need to amend *anscombe*, type and enter

```
fix(anscombe)
```

This brings up the spreadsheet with the data, which can be changed as you wish. Alternatively click on *Edit* on the tool bar to get access to the *Data Editor*.

Missing Values

R allows vectors to contain a special NA value to indicate that the data point is not available. The absent values are referred to as *missing values*, and are not included at the analysis stage.

Saving and Retrieving the Workspace

To save the entire workspace use

```
save.image()
```

When exiting from R, you are given the opportunity to save the workspace: at *File* on the toolbar, clicking on *Exit* gets a response

```
{Save workspace image?}
```

to which you answer yes or no.

Alternatively, the command

```
q()
```

instructs R to quit; it also responds with

```
{Save workspace image?}
```

A saved workspace may be retrieved at *File* on the toolbar, clicking on *Load Workspace*, and specifying the location of a workspace that you have previously saved.

Summarizing Statistical Data

First we implement some of the most commonly used descriptive statistical measures.

For the mean of *x1* write

```
mean(x1)
```

which gives

```
[1] 9
```

For the standard deviation of *x1*

```
sd(x1)
```

gives

```
[1] 3.316625
```

Similarly for *median*, *range*, *quartiles*, and *deciles*; the functions in R are usually similar to their actual names in statistics.

To get an overall summary of *x1*, write

```
summary(x1)
```

which gives the minimum, the first quartile, the median, the mean, the third quartile, and the maximum:

```
Min. 1st Qu. Median Mean
4.0   6.5    9.0    9.0
3rd Qu. Max.
11.5   14.0
```

To get an overall summary of all the variables in the *anscombe* data frame, type

```
summary(anscombe)
```

which gives

```

      x1          y1          x2          y2          x3
Min.   : 4.0   Min.   : 4.260   Min.   : 4.0   Min.   :3.100   Min.   : 4.0
1st Qu.: 6.5   1st Qu.: 6.315   1st Qu.: 6.5   1st Qu.:6.695   1st Qu.: 6.5
Median : 9.0   Median : 7.580   Median : 9.0   Median :8.140   Median : 9.0
Mean   : 9.0   Mean   : 7.501   Mean   : 9.0   Mean   :7.501   Mean   : 9.0
3rd Qu.:11.5   3rd Qu.: 8.570   3rd Qu.:11.5   3rd Qu.:8.950   3rd Qu.:11.5
Max.   :14.0   Max.   :10.840   Max.   :14.0   Max.   :9.260   Max.   :14.0

      y3          x4          y4
Min.   : 5.39   Min.   : 8   Min.   : 5.520
1st Qu.: 6.25   1st Qu.: 8   1st Qu.: 6.170
Median : 7.11   Median : 8   Median : 7.040
Mean   : 7.50   Mean   : 9   Mean   : 7.525
3rd Qu.: 7.98   3rd Qu.: 8   3rd Qu.: 8.190
Max.   :12.74   Max.   :19   Max.   :12.500
```

DATA ANALYSIS

We now look at how to perform some data analysis in R, by using some of its built-in functions and by writing simple programs and functions.

One can get quite far using R to execute simple expressions from the command line; some users may never need to go beyond this. However, at a more advanced level, it is possible to write programs and functions in R when what is required is not among what has been built in. This facility is one of the great benefits of R, and we look now at how it is done.

Writing R Programs

Programming is not too difficult in R because vectors and data frames are treated as single objects, and calculations can be done on these as if they were ordinary numbers. To illustrate we write a program to calculate the *mean*:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (1)$$

In standard programming languages, the calculation of Eq. (1) would involve ‘initialization’ and ‘loops’; with R however the conventions of vector calculations make it very easy to calculate statistical functions.

Example 1: A program to calculate the mean

```

sx1 <- sum(x1) #sum the elements in x1
n <- length(x1) #count the number of elements in x1
averx1 <- sx1/n

```

Here we see that `sum` adds all the elements in the vector, and `length` counts the number of elements in the vector. Notice also the symbol `#`: what follows is ignored by R, and is just a comment for the benefit of the reader. To print the value obtained, type

```

averx1
[1] 9

```

Alternatively, we could calculate the mean in just one statement:

```
averx1 <- sum(x1)/length(x1)
```

Let us look at how to calculate the *standard deviation*, which is defined as:

$$sd = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}} \quad (2)$$

We illustrate step by step how to calculate Eq. (2) for the data in `x1`.

Example 2: A program to calculate the standard deviation

```

diffx1 <- x1-mean(x1) #subtract the mean from each data point
diffsq <- (diffx1)^2 # obtain the squares of these differences
sumdiffsq <- sum(diffsq) #sum the squared differences
std <- sqrt(sumdiffsq)/(length(x1)-1) #divide this sum by (length(x1)-1), take the square root

```

Writing

```

std
gives
[1] 3.316625

```

Of course, the standard deviation may be calculated in a single statement, as follows:

```
std <- sqrt(sum((x1-mean(x1))^2)/
            (length(x1)-1))
```

We have seen that R has functions built in which calculate the most commonly used statistical measures. You will recall that the mean and standard deviation can be obtained directly with

```

mean(x1)
[1] 9
sd(x1)
[1] 3.316625

```

We took you through the calculations just to illustrate how easy it is to program in R.

Program Development

There are various ways of developing programs in R. If the program is short, it may be developed while working interactively at the workstation.

Alternatively, programs may also be developed in a text editor like Notepad, saved and retrieved using the *source statement*:

```
source("C:/test")
```

which retrieves the program named *test.R* from the C directory. An easy way of doing this while working in R is to click on *File* on the tool bar where you will be given the option to *Source R code*, and to browse and retrieve the program you require.

Scripts

The most useful way of writing programs is by means of an editor built in to R, called *Scripts*. From *File* at the toolbar click on *New Script (File/New Script)*. You are then presented with a blank screen to develop your program. When done, you may save and retrieve

this program as you wish. *File/Save* causes the file to be saved; you may designate what name you want to call it, and it will be given a *.R* extension. In subsequent sessions, *File/Open Script* brings up all the *.R* files you have saved, and you can select the one

you wish to use. When you want to execute a line or group of lines, highlight them and press *Ctrl-R*, that is, *Ctrl* and the letter *R* simultaneously. The commands are then transferred to the control window and executed.

Creating Functions

Users can write function of their own when what they need is not available as a built-in function in R. We take as an example the skewness coefficient, which measures how much the data differ from symmetry and is defined as

$$\text{skew} = \frac{\sqrt{n} \sum_{i=1}^n (x_i - \bar{x})^3}{(\sum_{i=1}^n (x_i - \bar{x})^2)^{3/2}}. \quad (3)$$

A perfectly symmetrical set of data will have a skewness of zero; when the skewness coefficient is substantially greater than zero, the data are assymetric with a long tail to the right, and a negative skewness coefficient means that data have a long tail to the left.

The following syntax calculates the skewness coefficient, and assigns it to a function called *skew* which has one argument (*x*).

Example 3: Function which calculates the skewness coefficient

```
skew <- function(x)
{
  sum2 <- sum((x-mean(x))^2)
  sum3 <- sum((x-mean(x))^3)
  skew <- (sqrt(length(x)) * sum3) / (sum2^(1.5))
  return(skew)
}
```

The function *skew* can be applied to any data set. For example

```
skew(y1)
```

gives

```
1] -0.05580807
```

which indicates that the *y1* data is slightly negatively skewed.

GRAPHICAL DISPLAYS

As well as numerical summaries, there are various pictorial representations and graphical displays available which have a more dramatic impact on the user and make for a better understanding of the data. The ease and speed which graphical displays can be produced is one of the important features of R. We look at some of the most commonly used.

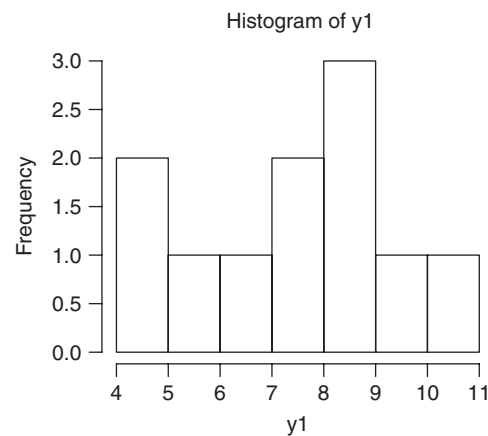


FIGURE 1 | A histogram.

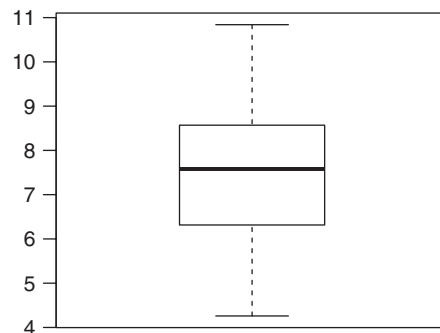


FIGURE 2 | A simple boxplot.

Histogram

The traditional way of examining the 'shape' of a set of data is a histogram.

```
hist(y1)
```

yields Figure 1.

Boxplots

A boxplot is a graphical summary based on the median, quartiles, and extreme values. To display the *y1* data using a boxplot, type

```
boxplot(y1)
```

which gives Figure 2.

Often called the *Box and Whiskers Plot*, the box represents the interquartile range which contains 50% of cases. The whiskers are the lines that extend from the box to the highest and lowest values. The line across the box indicates the median.

Multiple boxplots can be displayed on the same axis, by adding extra arguments to the boxplot

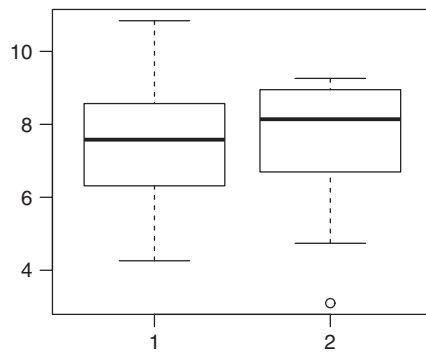


FIGURE 3 | Boxplot of y1 and y2.

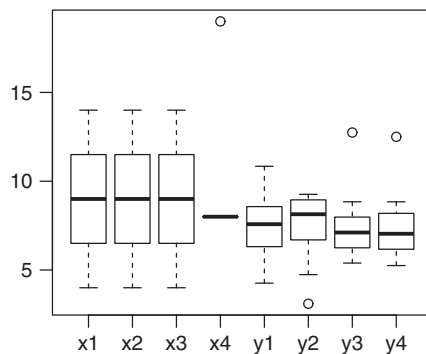


FIGURE 4 | Boxplots of all the variables in the data frame.

function or by using the complete data frame. For example

```
boxplot(y1, y2)
```

yields Figure 3.

Notice the point below the whiskers of the boxplot in y2. This data point is called an outlier and represents a case more than 1.5 box lengths from the upper or lower end of the box. This point is considered atypical of the data in general, being extremely low compared to the rest of the data.

Boxplots of all the variables in the data frame *anscombe* are obtained with

```
boxplot(anscombe)
```

which gives Figure 4.

Scatter Plots

Scatter plots are useful to investigate relationships between variables. To examine, for example, the relationship between x1 and y1, we could write:

```
plot(x1, y1)
```

to obtain Figure 5.

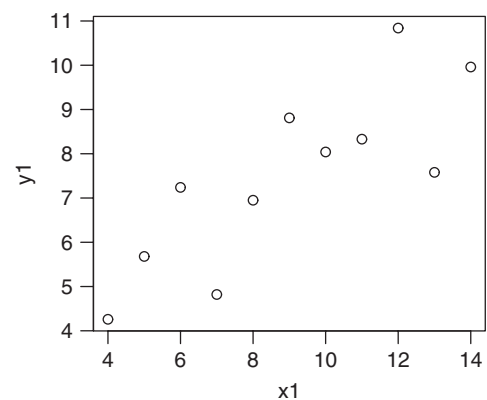


FIGURE 5 | A scatter plot.

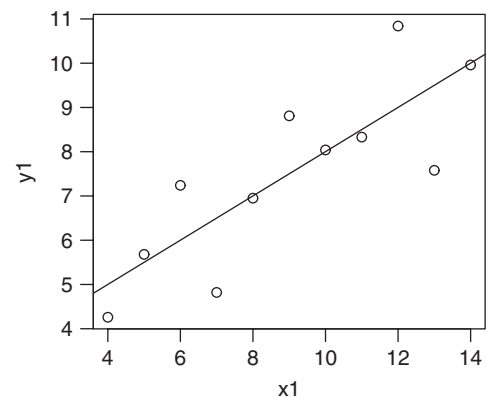


FIGURE 6 | The line of best fit.

Here you can see that there is what is called a *linear* trend in these data. The line that ‘best fits’ these data is obtained and displayed with

```
abline(lm(y1~x1))
```

This gives Figure 6.

When more than two variables are involved, R provides a facility for producing scatter plots of all possible pairs. Writing

```
pairs(anscombe)
```

will generate Figure 7.

Graphical Display versus Summary Statistics

Looking again at the Anscombe data set given in Table 1, we calculate the means (rounded to one decimal place) as follows:

```
round(mean(anscombe), 1)
```

gives

x1	x2	x3	x4	y1	y2	y3	y4
9.0	9.0	9.0	9.0	7.5	7.5	7.5	7.5

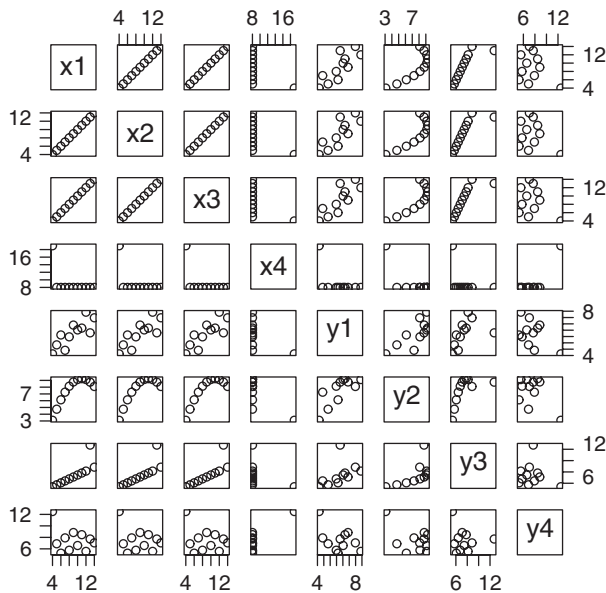


FIGURE 7 | Use of the *pairs* function.

The standard deviations (rounded to two decimal places) are calculated with

```
round(sd(anscombe), 2)
```

which gives

```
x1  x2  x3  x4  y1  y2
3.32 3.32 3.32 3.32 2.03 2.03
y3  y4
2.03 2.03
```

Notice that the four sets of data (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) have the same mean and standard deviation, which might lead to the conclusion that the four data sets are essentially the same.

Investigating further using graphical displays gives a different picture. Scatter plots is the obvious exploratory technique to use with paired data:

```
par(mfrow = c(2, 2)) #gives a two by two display
plot(x1,y1, xlim=c(0, 20), ylim=c(0, 13))
plot(x2,y2, xlim=c(0, 20), ylim=c(0, 13))
plot(x3,y3, xlim=c(0, 20), ylim=c(0, 13))
plot(x4,y4, xlim=c(0, 20), ylim=c(0, 13))
```

generates Figure 8. We use $xlim = c(0, 20)$ and $ylim = c(0, 13)$ to make the scales on the axes the same in the four plots, to allow for a valid comparison.

Examining Figure 8, we see that there are very great differences in the data sets:

1. Data set 1 is linear with some scatter;
2. Data set 2 is quadratic;

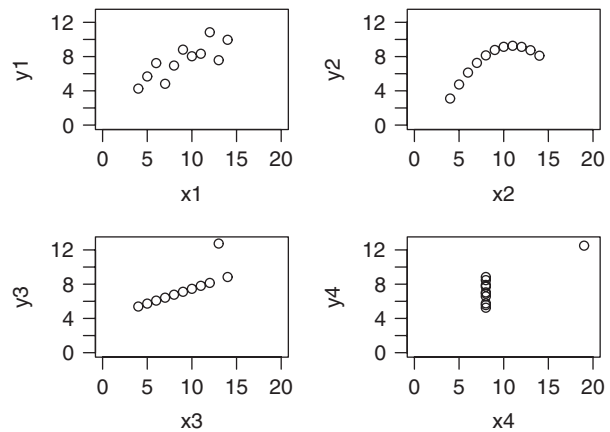


FIGURE 8 | Plots of four data sets with same means and standard deviations.

3. Data set 3 has an outlier. If the outlier were removed the data would be linear;
4. Data set 4 contains x values which are equal except for one outlier. If the outlier were removed, the data would be vertical.

Graphical displays are the core of getting ‘insight/feel’ for the data. Such ‘insight/feel’ does not come from the quantitative statistics; on the contrary, calculations of quantitative statistics should be done after the exploratory data analysis using graphical displays. The powerful graphical procedures of R facilitate this approach.

SIMULATION

With the computational power of R it is easy to *simulate* problems that might otherwise be difficult to understand. We illustrate with an example from the theory of *queues*.

Queues

There is an extensive literature on queuing theory; R enables us to sidestep the theory, and to concentrate instead on experimentation. We use as an example the M/M/1 queue, where there is one server dealing with customers on a first-in first-out basis. Customers are usually assumed to arrive in accordance with a *Poisson* distribution, and are served immediately if the queue is empty, otherwise they join the end of the queue. The service rates are also assumed to be *Poisson*.

Traffic intensity (I) is the ratio of that arrival rate to the service rate. When the arrival rate is greater than the service rate $I > 1$, when it is equal $I = 1$, and

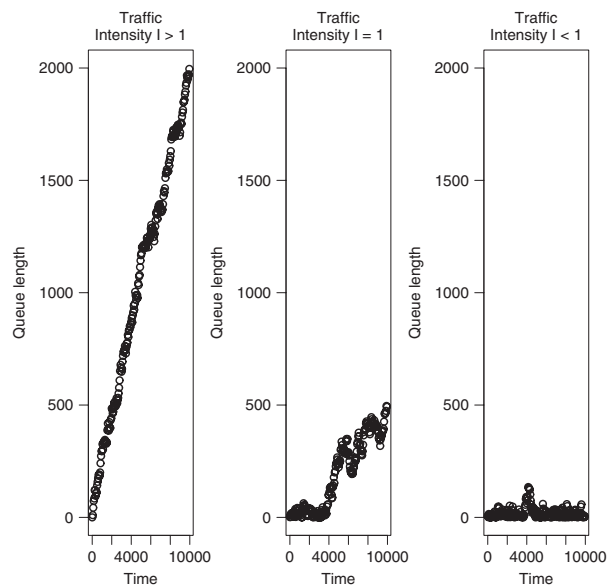


FIGURE 9 | Queue lengths.

when it is less than $I < 1$. We investigate each of these three scenarios in turn.

The following code simulates a queue in which customers arrive at the rate of 4 per minute, and are serviced at 3.8 per minute ($I > 1$). It generates 10,000 random Poisson arrivals (`rpois(10000, 4)`), and 10,000 Poisson services (`rpois(10000, 3.8)`), and calculates the queue length at each time interval.

Example 4: A program to simulate a simple queue

```
arrivals <- rpois(10000, 4) #generates 10,000 values from a Poisson dist with mean =4
service <- rpois(10000, 3.8) #generates 10,000 values from a Poisson dist with mean =3.8
queue[1] <- max(arrivals[1] - service[1], 0)
for (t in 2:10000) queue[t] = max(queue[t-1]+arrivals[t]-service[t], 0) #length of queue
plot(queue, xlab = "Time", ylab = "Queue length")
```

The service rates can be changed to `rpois(10000, 4)` for $I = 1$ and `rpois(10000, 4.2)` for $I < 1$. Figure 9 gives the output with the three different traffic intensities.

Figure 9 illustrates the severe problem that develops when the arrival rate is greater than the service rate ($I > 1$), the length of the queue is increasing steeply. With arrival and service rates equal ($I = 1$), the problem is not as severe, but it does exist, and we see that in the long run it will become serious. The only tenable solution to the queuing problem is to keep $I < 1$.

SUMMARY

We have tried to set before you some of the features of R which make it such a flexible and accessible language within which to tackle your statistical problems; we hope you have been convinced. For further and deeper information, there are many books and manuals both on and off line which, between them, deal with most statistical applications. Venables et al.² provide a manual which gives an introduction to the language and how to use R for doing statistical analysis and graphics; it is downloadable from the CRAN website (<http://cran.r-project.org/>). Chambers³ guides the reader in programming with R, from interactive use and writing simple functions to the design of packages and intersystem interfaces. Horgan⁴ deals with probability problems. Statistical inference examples are tackled in Dalgaard⁵. The book of Maindonald and Braun⁶ has extensive examples that illustrate practical data analysis using R. Fox and Weisberg⁷

give an introduction to the use of R in the context of applied regression analysis. Gentleman⁸ uses R to address bioinformatics problems.

ACKNOWLEDGMENT

Thanks to the referees whose observations and suggestions greatly improved this article.

REFERENCES

1. Anscombe FJ. Graphs in statistical analysis. *Am Stat* 1973, 27:17–21.
2. Venables WN, Smith DM, the R Development Core Team. An Introduction to R: A Programming Environment for Data Analysis and Graphics, 2004.
3. Chambers JM. *Software for Data Analysis: Programming with R*. New York: Springer; 2008.

Available at: <http://www.r-project.org/>. (Accessed July 04, 2011).

4. Horgan JM. *Probability with R: An Introduction with Computer Science Applications*. Hoboken, NJ: John Wiley & Sons; 2008.
5. Dalgaard P. *Introductory Statistics with R*. 2nd ed. Heidelberg: Springer-Verlag; 2008.
6. Maindonald J, Braun J. *Data Analysis and Graphics Using R*. 2nd ed. Cambridge: Cambridge University Press; 2007.
7. Fox J, Weisberg S. *An R Companion to Applied Regression*. 2nd ed. Thousand Oaks LA: Sage Publications; 2011.
8. Gentleman R. *R programming for Bioinformatics*. London: Chapman and Hall/CRC; 2009.