Charles Fleming

R Notes

Contents

Chapter 1

Introduction

With the rise in popularity of Linux, there has occurred a concurrent rise in popularity of freeware. This popularity arises not necessarily because the software is free but because of the freeware's recognized reliability and utility. Calling freeware by the name freeware can be misleading. While some software is distributed free of charge, the source code is copyrighted to keep it a secret and to keep it a proprietary product of the company. A new book, in comparison, is likewise protected by copyright; however, the owner of the book can read it, copy parts of it, modify it, sell or give it away to someone else to read without restrictions. The owner cannot sell or distribute a modified version or parts of a copyrighted book without the publishers consent. In the sense of having the freedom to modify, to copy, and to distribute at will a creative product, there is software which is "copyrighted" under the General Public License (GPL). Under this license which the Free Software Foundation has championed, software may be sold or may be distributed, but, in either case, the source code must be made freely available to any user, so that, in effect, the source code or modified version of the code will always remain in the public domain. As neither the source code nor any derivative of it will ever become proprietary, anyone can see how the software works. More importantly, one can modify the code to suit his particular needs and, if the improvement is deemed a good one, he may communicate it to the authors of the software for their consideration to implement. This practice happens routinely in the Linux community where, from the tens of thousands of beta testers, many contribute improvements to the Linux kernel. The community of R developers like the community of Linux developers is a development team of statistical software unsurpassed in size and talent.

At one time, S, a powerful language created at Bell Labs with which students of statistics and researchers at scientific institutions have come to embrace, was freely available but not anymore, since the divestiture of AT&T in 1984. A GPL implementation of S, called R is currently underway around the world in response to the proprietary restrictions placed on its successor, S-PLUS.

Free software is a nickname which is applied to computer programs which are distributed under the provisions of the General Public License (GPL). A similar name which is often used is *open source*. The two are not exactly synonymous, but they agree in the basic tenet that the source code of the computer program must be legible and made freely available to anyone. That requirement is important because

- Source code reveals how the software works.
- Anyone can modify it.
- Accessibility to the source code encourages submission of improvements.

Source code is intelligible whereas binary code which is produced from the compiling of a program for use on a specific operating system is unintelligible. For example, the source code which comprises the function of `generrandom.c` used in R is the following:

```
static void GetSeeds()
{
    SEXP seeds;
    seeds = findVar(R_SeedsSymbol, R_GlobalEnv);
    if (seeds == R_UnboundValue) {
            Randomize();
    }
    else {
      if (seeds == R_MissingArg)
      error(".Random.seed is a missing argument with no default\n");
        if (!isVector(seeds) || LENGTH(seeds) < 3)
                error("missing or invalid random number seeds\n");
        seeds = coerceVector(seeds, INTSXP);
        ix_seed = INTEGER(seeds)[0]; if(!ix_seed) ix_seed++;
        iy_seed = INTEGER(seeds)[1]; if(!iy_seed) iy_seed++;
        iz_seed = INTEGER(seeds)[2]; if(!iz_seed) iz_seed++;
        }
}
```

Because R is governed by the General Public License, the collection of uncompiled programs which constitute the source code of R must be made freely available to anyone who might want to study the logic of a certain procedure like the excerpt shown above for generating random numbers. Curiosity could lead someone to improve or correct the program and if the modification is deemed a good one by the core developers of R then it will be implemented in the official version.

The binary code; however is unintelligible. It is produced when the source code is compiled for an operating system. The compiled version of `generrandom.c` which was shown above is the following unintelligible binary code:

```
^?ELF^A^B^A^@^@^@^@^@^@^@^@^@^@^A^@^B^@^@^@^A^@^@^@^@^@^@^@^@^@^@n<88>
^@^@^@^@^@4^@^@^@^@^@(^@^L^@^A^@.shstrtab^@.text^@.data^@.rodata^@.stab
^@.stabstr^@.symtab^@.strtab^@.rela.text^@.rela.stab^@.comment^@^@^@^@
<9D><E3><BF><90><90>^P ^@@^@^@^@^A^@^@^@<92>^P^<90>^P^@@^@^@^@^A^@^@^@
!^@^@^@@^@^@^@^A^@^@^@^S^@^@^]<92>^Rb=@^@^@^@^A^@^@^@<D0>$ ^@<D0>^D ^@
<80><A2>^@^V<80>^@^E^A^@^@^@<D0>^D ^@<92> ^<D2>$ ^@!^@^@^@@^@^@^@^A^@^
@^@^S^@^@^]<92>^Rbc@^@^@^@^A^@^@^@<D0>$ ^@<D0>^D ^@<80><A2> ^@^V<80>^@
^E^A^@^@^@<D0>^D ^@<92> ^<D2>$^@!^@^@^@@^@^@^@^A^@^@^@^S^@^@^]<92>^Rbs@
^@^@^@^A^@^@^@<D0>$ ^@<D0>^D ^@<80><A2>^@^V<80>^@^E^A^@^@^@<D0>^D ^@<92>
^<D2>$^@<81><C7><E0<81><E8>^@^@<9D><E3><BF><88>
```


The GPL originated in 1984 and comes under the auspices of the Free Software Foundation. Richard Stallman created the GNU General Public License and the GNU Library General Public License.

The provisions of the GPL among others are as follows:

1. If you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

Copies of GPL and LGPL may be obtained from the Free Software Foundation:

- GPL: *ftp.gnu.org/pub/gnu/COPYING*
- LGPL: *ftp.gnu.org/pub/gnu/COPYING.LIB*
- GnuManifesto: *ftp.gnu.org/pub/gnu/GNUinfo*

- Webpage: *www.fsf.org*

In particular, the terms governing the use of R are given verbatim below:

```
This software is distributed under the terms of the GNU GENERAL
PUBLIC LICENSE Version 2, June 1991.  The terms of this license
are in a file called COPYING which you should have received with
this software.

If you have not received a copy of this file, you can obtain one
via WWW at http://www.gnu.org/copyleft/gpl.html, or by writing to:

   The Free Software Foundation, Inc.,
   59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

A small number of files (the API header files and export files,
listed in R_HOME/COPYRIGHTS) are distributed under the
LESSER GNU GENERAL PUBLIC LICENSE version 2.1.
This can be obtained via WWW at
http://www.gnu.org/copyleft/lgpl.html, or by writing to the
address above

''Share and Enjoy.''
```

1.1 Obtaining R for the Microsoft Operating System

1. Make a sub-directory in `C:` drive and call it R.
2. `http://cran.r-project.org`
3. Select Windows (95 and later)
4. Select base
5. Select mirror near you
6. Select `http://lib.stat.cmu.edu/R/CRAN` at Carnegie Mellon University University
7. The page automatically jumps back to Windows (95 and later)
8. Select base
9. Double click on `rw1070.exe`

10. Select save
11. The next menu asks where on the PC to put `rw1070.exe`. Put it in `C:\R`.
12. Wait for about 4 minutes as R is being retrieved from Carnegie Mellon.
13. A box appears. Select open.
14. The R installation starts.
15. Do not modify anything; simply click next, next, ...
16. To test if R works, click on the R icon; R should appear.

There are many libraries or what are nicknamed *add-on* packages which are collections of procedures which serve a specialized purpose. Some of the standard libraries are invoked automatically when R is initiated. Others, because there are so many of them, are not included in the installation of R and must be invoked manually. For example, the library RQuantlib is a set of routines which specializes in finance. It has routines written for options like American Option and European Option. Given the strike price of the option, continuous dividend yield, risk-free rate, time to maturity, and volatility of the underlying stock, the routine will produce the value of the option and other numbers which appear to be important for a financial analyst. In the Microsoft Windows version of R there exists a button for downloading a library. This version of R will retrieve the library and install it on the computer.

In the UNIX/Linux arena, the procedure is more involved. A package like `survey.tar.gz` is retrieved from the *Package Sources* section of *http://cran.r-project.org/*. Once the package has been brought to the computer and placed in a directory such as `/user/local/src/R/library`, the package is installed by executing the command, `R INSTALL survey.tar.gz`.

Regardless of the operating system, to incorporate a library like `survey` into the current session of R it is sufficient to execute the command `library("survey")` while at the prompt. The command `library(help=survey)` will display procedures which are contained in the `survey` library, and the command `data(package=survey)` will display the names of sets of data which came with the survey package when it was retrieved. Taking the `api` data, for example, the command `help(api)` will describe it and will show its name, `apipop`. Finally, the command `str(apipop)` will display explicit descriptive information of the variables of `apipop`.

The market for statistical computing software is dominated by SAS, SPSS, and S-PLUS. The popularity of R comes from the highly successful S language which is now proprietary and is the basis of S-PLUS. Since R is the GPL implementation of S, many users particularly those who belong to the community of contributors of S are attracted to R . Obtaining help from someone about R is easy. In general, help is easy, speedy, and usually complete.

Help is sought from the community of users of R through the use of e-mail. First, it is necessary to subscribe to the r-help mailing list by registering after having gone to the web page, `http://www.r-project.org/`→`Mailing Lists`→`r-help`. Use the `web interface` to gain access to the
`R-help -- Main R Mailing List: Primary help`
web page. At that place in the web page system of `r-project.org`, it is possible to subscribe to the `r-help` mailing list. Address messages for help to: `r-help@stat.math.ethz.ch`. Be mindful when composing the message that it will be sent to hundreds, if not thousands, of people around the world. A response from several experienced users of R will occur in a matter of minutes.

Some of the principal developers of R are:

- Friedrich Leisch and Kurt Hornik at Technische Univerität Wien (Vienna University of Technology)
- Martin Maechler at Eidgenössische Technische Hochschule Zürich (The Swiss Federal Institute of Technology)
- Peter Dalgaard at the University of Copenhagen
- Ross Ihaka at the University of Auckland, New Zealand
- Robert Gentleman at Harvard University
- Thomas Lumley at the University of Washington

Chapter 2

Basic Syntax

What is R ? R is a statistical computing package which can be employed interactively or by submitting programs in batch mode. The prompt which appears when R is invoked looks like: >. Whenever a command is enclosed in a rectangle, the command is meant to be executed by the reader as if it belongs to a tutorial. When R is used interactively, it can be used as a big calculator.

`>1+1`	Addition
`>2*2`	Multiplication
`>10-20`	Subtraction
`>105/35`	Division
`>57%%9`	Modulo, i.e. 57(mod 9)
`>2^5`	Exponentiation
`>sqrt(49)`	Square root
`>exp(10)`	Exponentiation of e
`>log(10)`	Logarithm to the base e
`>log10(10)`	Logarithm to the base 10
`>pi`	Special constants
`>complex(modulus=13,argument=pi/3)`	Complex number
`>(13+5i)*(1-2i)`	Multiplication of complex numbers
`>Mod(1+2i)`	Modulus of a complex number
`>Arg(1+2i)`	Argument of a complex number
`>Re(1+2i)`	Real part of a complex number
`>Im(1+2i)`	Imaginary part of a complex number

Parentheses may be used for making compound expressions.

```
>2+(5-1)^(2^3)
```

R follows the usual order of precedence when performing arithmetic operations. The ma-

nipulation of variables offers greater flexibility in performing numerical calculations. In R, what looks like a variable is called an *object*. Assignment of objects is done not by an = sign but by means of using the <- symbol which is composed of the less than symbol, <, and the hyphen, - and it looks like an arrow. The assignment symbol may face in either direction. Remember that > merely signifies, in these notes, the prompt, and it is not a part of a command.

```
>x<-2
```

The number 2 is assigned to x.

```
>y<-3
```

The number 3 is assigned to y.

```
>x+y->z
```

The sum of x and y is assigned to z.

The contents of an object can be displayed by typing the name of the object like, x, or by using print(x), for example:

```
>print("First Use of y")
>print(y)
```

According to the usual convention, a string of characters which will be printed is enclosed within quotation marks.

Another common command for printing the contents of an object is cat.

```
>cat(z)
```

While this command offers additional flexibility, it requires more detailed syntax. The following set of instructions illustrate a more complete application of cat, and it illustrates the appearance of the + symbol which indicates the continuation of a line and it is not a part of a command.

```
>for (n in 1:length(z)){
+cat("First Use of cat. x+y=",z[n],"\n")
}
```

Rather than display the results of using cat on the monitor, they can be directed to a file by using the file option:

```
>for (n in 1:length(z)){
+cat("First Use of cat. x+y=",z[n],"\n",file="/tmp/demo.txt",append=TRUE)
}
```

In this example, the for loop on the index n iterates through the values 1 to the length of z. For each value of n, cat will print the phrase First Use of cat. x+y, followed by the n^{th} element of z, and then since \n was specified, cat will begin a new line. More discussion of for loops will appear in Chapter 6 on Control Language.

Vectors and matrices are more complicated objects.

```
>z<-c(0,2,4,6,8)
>u<-c(1,3,5,7,9)
>u+z
```

```
[1]  1  5  9 13 17
```

The [1] is used to denote that the output begins with element one. In more extensive output, this notational device of conveniently denoting the beginning of a row with the position of an element will become more apparent later. Multiplication of vectors takes two forms: elementwise multiplication and the dot product.

```
>u*z
[1]  0  6 20 42 72
```

This use of * demonstrates that it is a command which will produce elementwise multiplication. A different symbol must be used to perform the dot product between two vectors or to perform matrix multiplication. First, the transpose of a matrix or vector is produced as follows:

```
>t(u)
```

so that the dot product of two vectors will conform and that $\mathbf{u}'\mathbf{z}$, can be computed using the matrix multiplication, %*%, command as in:

```
>t(u)%*%z
[1,]  140
```

There are two ways to perform the product: $\mathbf{u}\mathbf{z}'$

```
>u%*%t(z)
```

or

```
>outer(u,z)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    6    8
[2,]    0    6   12   18   24
[3,]    0   10   20   30   40
[4,]    0   14   28   42   56
[5,]    0   18   36   54   72
```

where $\mathbf{u} = [1, 3, 5, 7, 9]$ and $\mathbf{z} = [0, 2, 4, 6, 8]$.

Earlier the object, z, was created by assigning a column of numbers to z:

```
>z<-c(0,2,4,6,8)
```

Another way to enter data into R is by means of the scan() command. For example,

```
>z<-scan()
>1:0
>2:2
>3:4
>4:6
>5:8
>6:
```

The elements of z can be entered one-by-one when prompted. The prompt begins with the position of the vector followed by a colon, after which the value of the element is entered. The process continues until the last element is followed by a blank.

It should be emphasized that the language of R accommodates the needs of a professional statistician who depends on matrix algebra, generating random numbers, computing probabilities, and producing graphs. In these aspects, R is very efficient, and in these are its strengths. A conspicuous difference between R and other brands of statistical computing software is the absence of a graphical user interface (GUI). There is little interest in the R community to build one, although there is a project currently underway to develop a GUI for R . The sentiments which are prevalent among the users of R are the same, in general, as those found elsewhere among professional computer programmers. A GUI is regarded to be an unwanted impediment to masterful programming. For a proficient computer programmer, the lexicon of a language like R will have been memorized, and its use is honed by practice to the extent that a set of commands may be written extemporaneously more quickly than if he were to depend on a GUI. Even if a functional GUI were available in R , it probably will receive little use.

As far as the conceptual design of writing a computer program, there is a very good correspondence between R and SAS/IML except in regard to making graphs which R is vastly superior to SAS. The logic of a program written in R is similar to one written in FORTRAN, and the syntax of R bears a similarity to the syntax of C. In other words, someone who is adept at writing programs in FORTRAN, C, or SAS/IML will quickly understand R .

To see all the objects which exist in the current session, one uses the commands:

```
>objects()
>ls()
```

Suppose an object is no longer needed and, to conserve space on a computer, it should be removed, then it is deleted by: `>rm(u)`. Use `ls` to verify that `u` has been removed:

```
>ls()
```

Although R will automatically perform an operation called *garbage collection* for the purpose of returning memory to the operating system, sometimes when large objects have been used and eventually deleted, executing `gc()` will clean things up in R 's use of memory.

Vectors and matrices must conform in dimension otherwise a warning will be produced as in:

```
>z >u<-c(1,3,5)
>u+z
```

```
[1]  1  5  9  7 11
Warning message:
longer object length
        is not a multiple of shorter object length in: u + z
```

R will cycle through the elements of the shorter vector in order to complete the operation. This is convenient sometimes, for example:

```
>u<-10
>u+z
```

```
[1] 10 12 14 16 18
```

Sets of commands sometimes occur so often that they are consolidated and given a name. One such procedure is `rep` which is an abbreviation for replicate. Rather than assign 10 to `u` as was done above and exploit the provision of cycling through the shorter vector to complete the operation, `u` could have been assigned the vector consisting of $[10, 10, 10, 10, 10]$ which then is added elementwise to the vector, `z`. That is,

```
>u<-rep(10,length(z))
>u
>u+z
```

The command `>rep(10,length(z))` means: repeat 10 as many times as z is long. A similar command to `rep` is `seq` for sequence.

```
>v<-seq(1,100,2)
```

```
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
[26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

The `[26]` means that the 26^{th} element in the vector, v, begins the second row of the output to the monitor. The command, `seq(1,100,2)`, is interpreted to mean that an arithmetic sequence

begins with 1 and ends at 100 by a difference of 2 between steps.
A shorthand method for defining a sequence of consecutive numbers is:

```
>v<-1:10
```

```
[1] 1  2  3  4  5  6  7  8  9 10
```

This same sequence can be produced by:

```
>v<-seq(1,10,1)
>v<-seq(1,10)
```

The command `seq(1,10)` is an abbreviated form of `seq(1,10,1)` which is actually an abbreviated form of `seq(1,10,by=1)`. An arithmetic sequence beginning with -1 and ending at 1 with a difference of .1 can be produced by:

```
>v<-seq(-1,1,.1)
>v
```

```
 [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4
[16]  0.5  0.6  0.7  0.8  0.9  1.0
```

or by

```
> seq(-1,1,len=21)
```

```
 [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4
[16]  0.5  0.6  0.7  0.8  0.9  1.0
```

where the option `len=21` specifies that a sequence be produced beginning with -1 and ending at 1 so that the sequence consists of exactly 21 elements.

In the case of generating a geometric series, it is sufficient to use:

```
>3^(seq(1,10))
```

```
> [1]     3     9    27    81   243   729  2187  6561 19683 59049
```

There are many functions in R which appeal to a statistician like those which generate random numbers. The function `rnorm` produces random numbers from a Normal distribution, and another popular procedure of the same kind is `runif` for generating random numbers from a Uniform distribution.

```
>v<-rnorm(10)
>v
```

```
    [1] -0.2826475604  1.5474520284  0.1604545019  0.0335328292 -1.2486159628
 [6] -1.1899395442 -1.6716649564  0.0002192872 -0.4893145173  2.1375458025
```

No computer program exists which will produce purely random numbers. In fact, devising a rigorous definition of random numbers is still an unsolved problem even though we have an intuitive understanding of its meaning. What is called a random number in a computer routine is a number which is produced from a variety of algorithms taken from the theory of numbers. Random number generators include the popular linear and multiplicative congruential random number generators. Each of these algorithms depends on a large prime number from which a

remainder is obtained by means of the operation in modular arithmetic called *modulo*. These algorithms are usually given names. Each has its strengths and problems. A few of the named algorithms for generating random numbers are: Wichmann-Hill, Marsaglia-Multicarry, Super-Duper, Mersenne-Twister, and Knuth-TAOCP. Although it is possible in R to specify which algorithm to use, an ordinary user of R will probably not care and will simply accept the default generator. If none of the random number generators which are available in R is good enough, then because R is licensed under the GPL, someone can modify the source code of the random number generator, incorporate one from somewhere else, or compose his own novel random number generator. If it is a good one, then it is worthy of communicating it to the maintainers of R for consideration.

The command, `order`, will determine the ranking of each element in a vector.

```
>v<-c(-3,4,1,2,5)
>order(v)
```

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

This is useful in arranging the elements of a vector in ascending order. It produces a list of positions of v which will correspond to the right ordering of v. In ascending order, a rearrangement of v is `-3 1 2 4 5` in which -3 is seen to correspond to position 1 in v; 1 corresponds to position 3 in v; 2 corresponds to position 4 in v; 4 corresponds to position 2 in v; and, 5 corresponds to position 5 in v. The command, `order`, therefore, will produce a vector of positions which if the original vector had been arrayed according to the positions it will be arranged in ascending order. By arranging the 1^{st}, 3^{rd}, 4^{th}, 2^{nd}, and 5^{th} elements of v in that order, v will be sorted in ascending order:

```
>o<-order(v)
>v[o]
```

```
[1] -3  1  2  4  5
```

In order to produce a vector in descending order, it is sufficient to use `order(-v)`, so that,

```
>o<-order(-v)
>v[o]
```

```
[1]  5  4  2  1 -3
```

The command, `rev`, reverses the order of a vector.

```
>rev(v)
```

```
[1]  5  2  1  4 -3
```

The square brackets are used to make subsets of a vector, matrix, or data frame.

```
>v[1]
>v[2:4]
>v[c(2,5)]
```

. To identify the position of an element of a vector, a logical relation is employed. Suppose, `>h<-c(1,2,4,1,3,10)` then `>h==1` will print TRUE at every position of v at which a 1 appears.

```
[1]  TRUE  FALSE FALSE TRUE FALSE FALSE
```

The use of == invokes a logical expression in R which produces a series of TRUE and FALSE each corresponding to an entry in the vector. The output can be used to make subsets according

to some condition.

```
>h[h==1]
[1] 1 1
```

```
>h[h<=4]
[1] 1 2 4 1 3
```

One may chose specific positions like the 1^{st}, 2^{nd}, and 5^{th} positions as in:

```
>h[c(1,2,5)]
[1] 1 2 3
```

It is not uncommon to exclude elements of a vector or matrix. In order to exclude the 1^{st} and 4^{th} elements of `v`, one may use:

`>h[-c(1,4)]`

The hypen which is the prefix of `c(1,4)` is interpreted to mean exclude.

2.1 Data Frames

A data frame is simply a combination of variables. They are useful when the objects are analyzed at once rather than typing the individual objects repeatedly for each procedure. A data frame is not a matrix, even though a data frame may at first seem like a matrix, since it can be manipulated as if it were a matrix. Let

```
>u<-seq(0,8,2)
>v<-seq(1,9,2)
>dd<-data.frame(u,v)
>dd
```

```
  u v
1 0 1
2 2 3
3 4 5
4 6 7
5 8 9
```

The sequences, `u` and `v`, have been combined into an object called a data frame. The use of `str` produces a synopsis of the structure and contents of an object, for example:

```
>str(dd)
```

```
'data.frame':   5 obs. of  2 variables:
 $ u: num  0 2 4 6 8
 $ v: num  1 3 5 7 9
```

Let `cc<-data.frame(seq(1,8,1),seq(9,2,-1))` be another data frame, then

```
>dd+cc
>dd*cc
>dd/cc
>plot(dd)
```

are all successful commands but the command for matrix multiplication
`>t(dd)%*%cc`
fails, because `dd` and `cc` are not matrices; they are data frames.

Any number of variables which conform in dimensions may be put together into a data frame. For example, let

```
>u<-seq(1,8,1)
>v<-seq(9,2,-1)
>z<-seq(-8,-1,1)
```

. The vectors, `u`, `v`, and `z` may be used to form the dataframe, `ee`, which, in turn, can be used for the purpose of processing the collection of objects all at once.

```
>ee<-data.frame(u,v,z)
>plot(ee)
```

A unit of a dataframe is extracted by means of a $ symbol as in `ee$u`. Given a dataframe, it is possible to manipulate elements individually by extracting them and operating on them, so that, for example

```
t(ee$u)%*%ee$v
```

```
> [1,]  156
```

Having to write the name of the data frame becomes a nuisance after awhile. The command, `attach()`, will put the variables which constitute the data frame in the current workspace. Rather than write `t(ee$u)%*%ee$v`, one could have written:

```
>attach(ee)
>t(u)%*%v
```

without having to make a reference to the data frame, `ee`, again. When the processing of the data frame, `ee`, ends, the constituents of the data frame are taken out of the workspace by executing:

```
>detach(ee)
```

2.2 Matrices

While a dataframe bears a resemblance to a matrix in some ways, it is not a matrix, but, on the other hand, it can be converted into a matrix by means of the command, `as.matrix`.

```
>m<-as.matrix(dd)
>m
```

. The same matrix could have been created by

```
>cbind(u,v,z)
```

The command, `cbind`, is a command which will append columns of one matrix onto the columns of another. Likewise, `rbind` is used to append rows of a matrix onto the rows of another matrix as in: `>rbind(u,v,z)` which will produce one long vector in which `z` is appended to `v` which in turn is appended to `u`. A clever use of transpose with `rbind` will produce the same thing as `cbind`, for example

```
>t(rbind(t(u),t(v),t(z)))
```

The choice of using `rbind` or `cbind` is dictated by the context of the problem. Sometimes, it is appropriate to append by columns and other times by rows. The latter use of appending by rows with `rbind` will be extensively used in an example of a Monte Carlo technique.

Names can be assigned to a matrix at its creation by

```
>m<-cbind(STAT=u, ENGLISH=v, PROFITS=z)
>m
```

```
     STAT ENGLISH PROFITS
[1,]    1       9      -8
[2,]    2       8      -7
[3,]    3       7      -6
[4,]    4       6      -5
[5,]    5       5      -4
[6,]    6       4      -3
[7,]    7       3      -2
[8,]    8       2      -1
```

Names which may be assigned to columns or to rows are helpful not only to remember a significant association of an element with something but also to make them appear in the output of a stored procedure later. Names are given to columns:

```
>colnames(x)<-c("w","x","y","z")
```

or names are given to rows:

```
>rownames(x)<-letters[1:3]
>x
```

Because names are inherited by procedures to produced popular outputs like an analysis of variance table or plots give the names as specified to columns and rows of a matrix.

An internal function for producing lower case letters is called `letters`, and `LETTERS` produces upper case letters.

```
>letters
>LETTERS
>colnames(x)<-letters[23:26]
>x
```

The transpose of a matrix is

```
>t(m)
```

An element of a matrix can be directly changed by modifying an individual element

```
>m[3,2]<-9999
>m
```

Another way to create a matrix besides creating a matrix from a dataframe is to use the command, `matrix`, for example:

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

```
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

or to initiate a matrix with a sequence of numbers which will be wrapped the sequence into a matrix along columns and rows the following command can be used:

```
>x<-matrix(1:12,nrow=3,byrow=T)
>x
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

2.3 Editing and Help

Maintaining a cheatsheet of useful or tricky commands is a good practice, because there are approximately 681 commands in the basic package of R . It is impossible to remember all of them let alone to remember the details of one's favorite commands. It is especially annoying to forget the syntax of a command after it had been used in a clever application once before.

Many features of a command are described by prefixing ? before the procedure of interest:
`>?Arithmetic` or `>?Logic`

The command, `apropos`, will search the current session of R for all commands containing the procedure in their names. For example,

```
>apropos("plot")
```

If one of the many commands involving `plot` seems interesting like `termplot`, then the description of its syntax may be examined by using `?termplot` . Usually the examples which are given at the end of a description are extremely useful when learning to understand a procedure. One could by means of the cursor, highlight an example, copy and paste it to the R prompt to see what the example does. In this way, the correct syntax of a command may be quickly learned. Notwithstanding the utility of manual pages which are always present in R some people like to use a Graphical User Interface (GUI). The command, `>help.start()` will start the computer's web browser to display the manuals for all the commands in R .

Keeping a record of every command which has been successfully used in a R session in a separate file is a helpful device in constructing a program. Conversely, the set of recorded com-

mands which have been copied to a separate file can be highlighted and pasted into the current or in a new R session. Commands which had been executed are preserved anyway in a file, `.Rhistory`, and objects are preserved in the file, `.RData` which if saved when terminating an R session, will be incorporated automatically into the next R session.

Sets of data may exist in ASCII files which can be edited directly by means of an editor like VI. Sets of data which exist as SAS data files or SPSS data files may be brought into an R session by means of the `read.ssd` or `read.spss` commands. Due to the format of these types of files, a set of data must either be edited in the parent software system or they may be edited in R once they have become an object in the current R session. There exists a GUI in R for the purpose of modifying data. It is invoked by the command, `fix`

```
>fix(x)
```

It is, also, possible to make spot revisions of a data frame or matrix manually as in

```
>x[3,2]<-8888
>x
```

The command, `library()`, will produce a listing of all the packages of R which have been installed on the computer.

```
>library()
```

There are many libraries which can be retrieved from `cran.r-project.org`. One such useful package is `foreign`. After it has been installed on the computer, it is brought into an active session of R by the command

```
>library("foreign")
```

To see a listing of all routines which are contained in a library like `foreign` use

```
>help(package="foreign")
```

From the resulting listing, we see that with this library R is capable of bringing the contents of a SPSS or SAS data file into the current session of R .

After awhile, the session of R must eventually be terminated. To do so, enter

```
>q()
```

By answering in the affirmative whether or not to save the workspace, the next time R is invoked, all the objects and functions which were active in the last session will become active in the next current session. However, an answer of `no` will cause all objects to disappear.

Chapter 3

Graphics

Manipulating vectors, matrices, and data frames in order to analyze data lies at the heart of R but the analysis of data is the easiest part of a statistical or scientific endeavor. Without a question, the most difficult and expensive part of science is getting the data. The U. S. Government spends billions of dollars per year in getting data from surveys, experiments, and espionage. Many thousands of people are employed for the sole purpose of getting data for the benefit of a few analysts. Analyzing data is relatively easy especially since the advent of the digital electronic computer. In the end, a report of the experiment and the presentation of the conclusions which the analysis of the data substantiates must be lucid and well composed. The inclusion of pictures of the data and graphs of trends in the report are indispensable devices for clarifying concepts.

A simple graph to make is the one of a mathematical function like `sin(x)`:

```
>curve(sin(x),-2*pi,2*pi)
```

It is plotted from -2π to 2π. To embellish the picture with a title, labeled axes, and colored lines of various styles options of `curve` may be used.

```
>curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
>abline(h=0,col="blue",lty=2)
```

The syntax of `curve` allows for the specification of the domain of `sin(x)`. The color of the horizontal line is specified by `col="blue"` and the dotted style of the horizontal line is specified by `lty=2`. Another function to graph is the cosine function:

```
>curve(cos(x),-2*pi,2*pi,main="Cosine of Time",xlab="",ylab="",yaxt="n",
xaxt="n",col="green",lty=5)
```

As a result of setting `xlab=""` and `ylab=""`, the axes are not labeled. This was done in order to superimpose the two graphs on each other.

```
>curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
>par(new=TRUE)
>curve(cos(x),-2*pi,2*pi,xlab="",ylab="",yaxt="n",xaxt="n",
col="green",lty=5)
>abline(h=0,col="blue",lty=2)
```

While at first the commands might seems overwhelming, a more complete description of the options for use in `curve` appears in `?curve`, `?plot`, and in `?par`. The command, `par`, does not produce statistics or a graph. It sets the graphical parameters. Graphical parameters may be specified within a plotting function as was done in making a picture of the sine function with `curve`. The other way of setting a graphical parameter is by means of the command, `par`. When a graphical parameter is set by means of `par`, it is used henceforth for the duration of the current session of R unless it is superseded by another use of `par`.

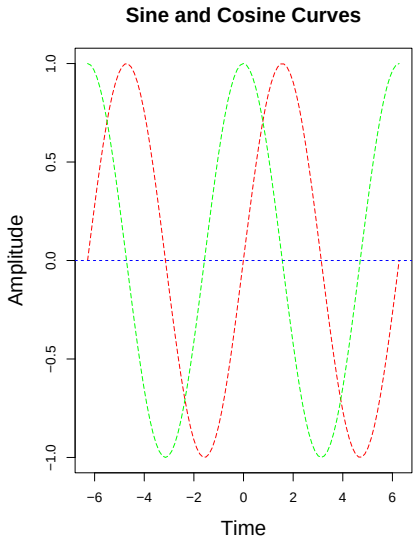
In the description of `par`, there is an option which specifies the seven styles of lines, `lty`:

Table 3.1: Styles of Lines

0	blank
1	solid
2	dashed
3	dotted
4	dot dash
5	long dash
6	two dash

The options, `xlab` and `ylab`, allows for the arbitrary use of labels for the x and y axes. The use of `xaxt="n"` specifies that the x-axis must not be plotted. In the example of drawing a picture of the cosine function, `xlab=""` and `xaxt="n"` cause no labelling of the x-axis and no use of a marked scale. Sometimes keeping the axis blank is useful at the time of superimposing two graphs. The first instance of `curve` will produce the title, labeling of the axis, and the image of the first figure, while the second graph will be superimposed on the first. The superimposition does not occur automatically. Every time a plotting function like `plot`, `curve`, and `matplot` is used, R erases any previous vestige of a plot and starts with a fresh plot. In order to superimpose two images on the same plot, the command `par(new=TRUE)` must be inserted in between the two plotting functions as was done above for superimposing a cosine plot onto a sine plot.

Another and better way to superimpose two graphs is with the use of the option, `add=T`:




```
>curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
>curve(cos(x),-2*pi,2*pi,ylab="",yaxt="n",xaxt="n",
col="green",lty=5,add=TRUE)
>abline(h=0,col="blue",lty=2)
```

In all of the examples discussed thus far, the plots have been of mathematical functions for which `curve` is used. Statisticians like to make pictures of data. By executing `apropos("plot")`, the result will prove that there are many commands with which to make pictures of data in R. Rather than make a graph of a mathematical function, the following examples will make plots of data. Even though, the set of data upon which the following plots are based is contrived, the resulting simplicity of the graphs will bring out more readily features of the commands. For example,

```
>x<-sin(-2*pi+(1:100)*pi/50)
>t<-1:100
>plot(t,x)
```

will produce a plot which will clearly shows the characteristic feature of discrete data. The plotted points of the data can be joined by a smooth line, if the `type` option is used in the `plot` command. `>plot(t,x,type="l")` The solid line interpolates the set of points so that it looks as if a mathematical function had been plotted.

Rather than superimpose two mathematical functions on one plot, we will superimpose two pictures of data on one plot. To that end, discrete values from a cosine function will be assigned to the object, `y`.

`>y<-cos(-2*pi+(1:100)*pi/50)`. When the sine and cosine derived sets of data are placed

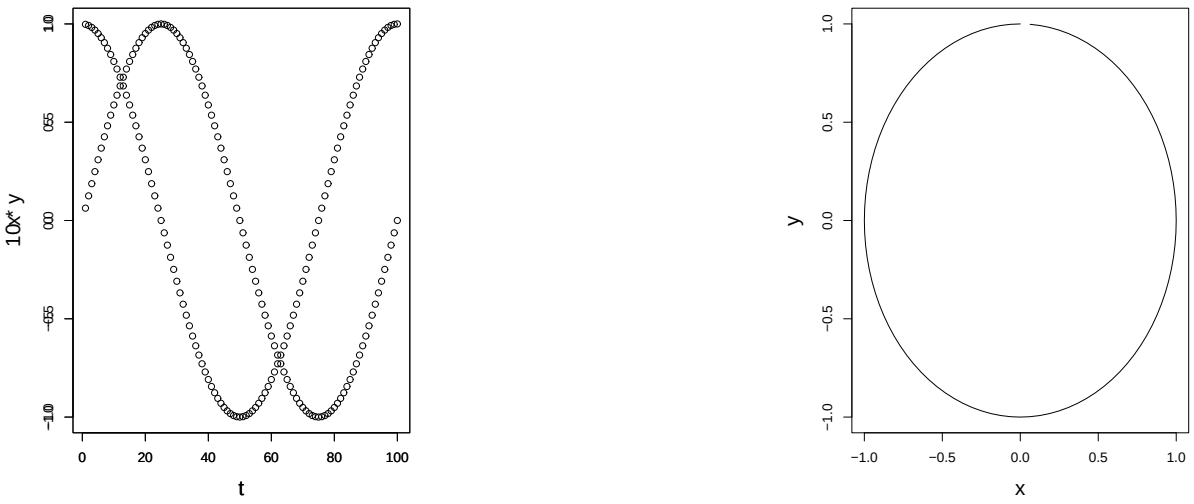


Figure 3.1:

on one plot by: `>plot(x,y)` a circle appears instead of a superimposed sine and cosine waves because the values of x and y when plotted together actually represent a parametric expression of a circle. The desired superimposition of sine and cosine waves is accomplished by creating two separate plots and then add the second one on top of the first by the use of the `par(new=TRUE)` option.

```
>plot(t,x)
>par(new=TRUE)
>plot(t,y)
```

This time, the resulting picture as shown in Figure 3.1 is what is wanted. Coincidentally, both plots are made over the identical scales of the x and y axes. Suppose the axes of the two plots are not identical as in

```
>plot(t,x)
>par(new=TRUE)
>plot(t,10*y)
```

The superimposition of the two plots, however, is a poor one. A remedy for the mismatching which is cause by this approach in superimposing two graphs can be found in another approach in which the two sets of data are combined into a matrix from which both functions are drawn in the same graph.

```
>m<-cbind(x,y)
>plot(m,type="l")
```

. Once again a circle is produced contrary to our intentions and clearly shows that this approach is flawed. Nevertheless, there is a special plotting procedure which makes plots of one column of a matrix against another column. That procedure is called `matplot`. It will produce the desired superposition of the cosine and sine functions as shown in Figure 3.2.

```
>matplot(m,type="l",main="Sine and Cosine Curves",
col=c("red","green"),xlab="Time",ylab="Amplitude")
>abline(h=0,col="blue",lty=2)
```

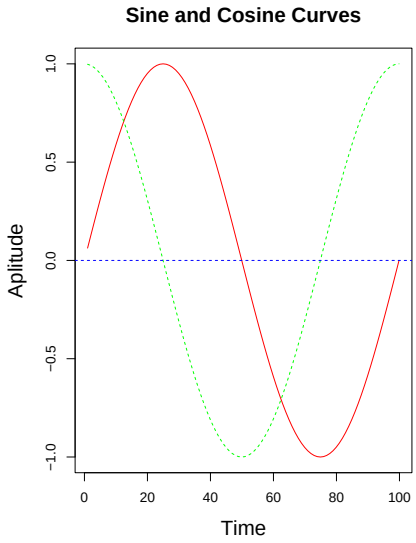


Figure 3.2:

Incorporating a legend into the plot for identifying the two curves seems appropriate, but where should it be placed? The `locator` function will produce the co-ordinates at that place on the plot where the cursor is placed and the left key of the mouse clicked. Two points will be specified in `locator`. One point will coincide with the upper left corner of the legend, and the second point will coincide with the lower right corner of the legend. By means of the cursor, these two points will be used by R to place the legend of the right size in the right place.

```
>legend(locator(n=2),legend=c("Summer","Winter"),col=c("red","green"),
lty="1")
```

. To see what `locator` produces, execute `>locator(n=2)` and click when the cursor is where the upper left and lower right corners of the legend should be placed. Explicitly, `locator` produces co-ordinates. The co-ordinates which `locator` furnishes are automatically utilized by the legend command. Even though, it is convenient to incorporate `locator` directly in the `legend` command, putting the actual co-ordinates which `locator` provides into the legend manually makes it possible to reproduce the plot with the legend in exactly the same position.

```
>legend(c(49.75750,75.37375),c(0.9460465,0.7325581),
legend=c("Sine","Cosine"),col=c("red","green"),lty="1")
```

A written report includes graphics, and unless a graph can be printed on paper it cannot be

used in a report. A graph which is created by R

```
>postscript(horizontal=F,file="/tmp/CPI.ps")
>matplot(m,type="l",main="Sine and Cosine Curves",
col=c("red","green"),xlab="Time",ylab="Amplitude")
>abline(h=0„col="blue",lty=2)
>legend(c(49.75750,75.37375),c(0.9460465,0.7325581),legend=
c("Sine","Cosine"),col=c("red","green"),lty="1")
>dev.off()
```

can be saved as was done in the preceding example to a file in a Postscript format which is recognized by modern printers. The last command, `dev.off()`, in the last set of instructions terminates the use of the graphics device and causes the image to be sent to the file, `/tmp/CPI.ps`.

One of the most popular forms of presenting data for a statistician is the histogram.

```
>w<-c(83,85,74,70,92,64,72,87,88,75)
>hist(w)
```

. There are various options in R for producing histograms with different styles. A histogram which displays the relative frequency is produced by: `>hist(w,prob=T)` ; with absolute counts by: `>hist(w,prob=F)` . The sizes of the bins may be specified by means of the `breaks` option as is done here:

```
>br<-seq(40,100,5)
>hist(w,breaks=br,prob=T,main="Exam Scores from Watching Videos",xlab="Scores")
```

It is often desired to superimpose a Normal distribution on a histogram.

```
>curve(dnorm(x,mean=mean(w),sd=sd(w)),add=T)
```

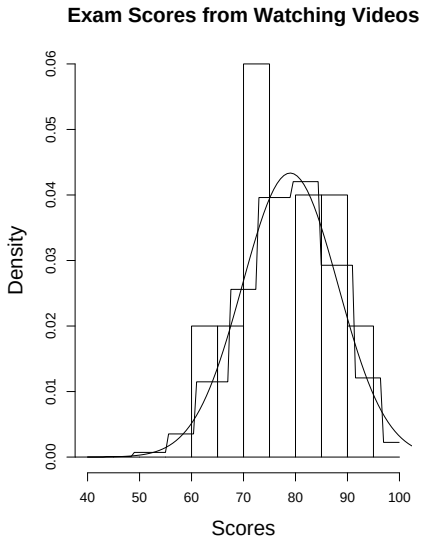
.
A Normal distribution is almost always an approximation for a histogram. Perhaps, a binomial distribution adequately describes the data.

```
>curve(dbinom(round((x-40)/60*length(w)),length(w),
mean((w-40)/60))/6,40,100,add=T)
```

In this example, a Binomial distribution was translated so that it is centered on the histogram. R does not know how to position a Binomial distribution or a Normal distribution without the help of the statistician who must employ the right mathematical formulas.

Superimposed on top of the histogram, there appear two mathematical functions. They are `dnorm(x)`, the probability density function of the standard Normal distribution and `dbinom(x)`, the probability mass function of the Binomial distribution. By entering `?dnorm` and `?dbinom`, a description of the syntax of each will be displayed. The plot of the Normal distribution is easy to make because the options are obvious. On the other hand, the task of superimposing the Binomial distribution on the histogram is difficult and tricky. It hardly comes as a surprise that the Normal distribution is everyone's favorite distribution. It is easy, versatile, and fundamental in the theory of statistics.

In an attempt to make the previous plot less complicated, it will be divided into two graphs and placed side-by-side.




```
>par(mfrow=c(1,2))
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
>hist(w,breaks=br,prob=T,main="Exam Scores from Watching Videos",
xlab="Scores",col="red")
>curve(dnorm(x,mean=mean(w),sd=sd(w)),add=T)
>hist(w,breaks=br,prob=T,main="Exam Scores from Watching Videos",
xlab="Scores",col="red")
>curve(dbinom(round((x-40)/60*length(w)),
length(w),mean((w-40)/60))/6,40,100,add=T)
```

.

The key command for putting two plots side-by-side on the same page is the parameter statement, `par(mfrow=c(1,2))`. To put four plots on the same page, `par(mfrow=c(2,2))` is used. Similarly, to put three columns in two rows of plots on the same page, `par(mfrow=c(2,3))` is used. To reset the frames so that only one plot appears on a page, use

```
>par(mfrow=c(1,1))
```

Suppose another set of data besides `w` was obtained and is assigned to the object, `x`.

```
>x<-c(95,81,59,68,74,79,72,70,81,58)
```

The set of data contained in `w` and the set of data contained in `x` are obtained in a process which makes `w` and `x` independent sets of data. The set of data in `w` are scores from an examination in understanding French from students who attend classroom lectures whereas `x` contains examination scores for proficiency in French from students who also listened to audio tapes of French. We wish to see the data of both.

```
>plot(w,x)
```

Some points lie far away from the rest of the data. The command `identify` will allow us to find which points in the data produced the points of interest in the plot.

```
>identify(w,x)
```

A more ambitious goal might be to place the names of the points on the

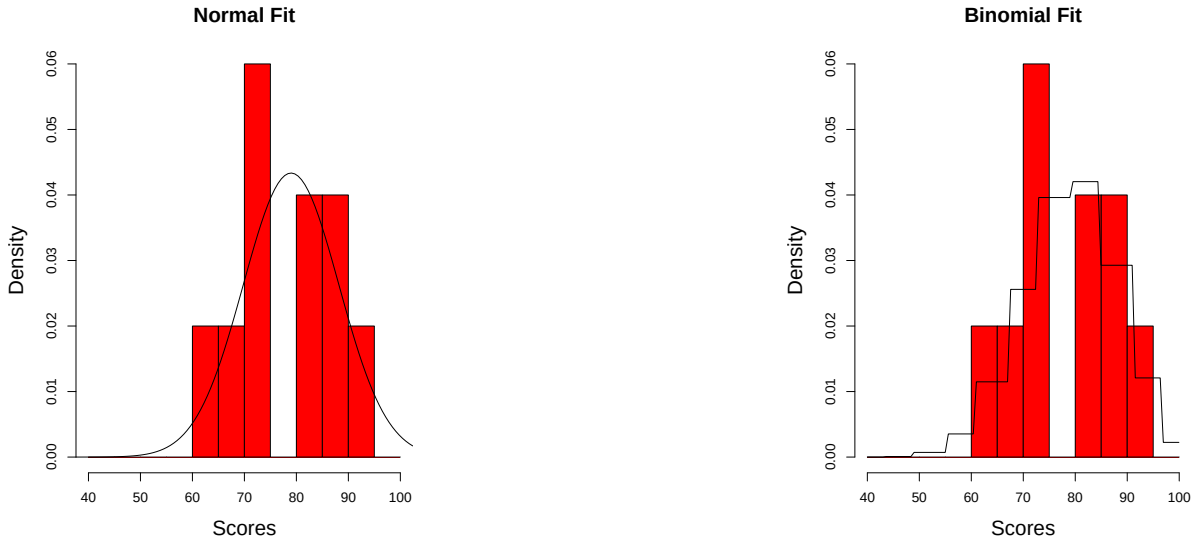


Figure 3.3:

plot as a result of identifying some of them as in the following example of identifying four points and saving the resulting image to a Postscript file, `fig8.ps`:

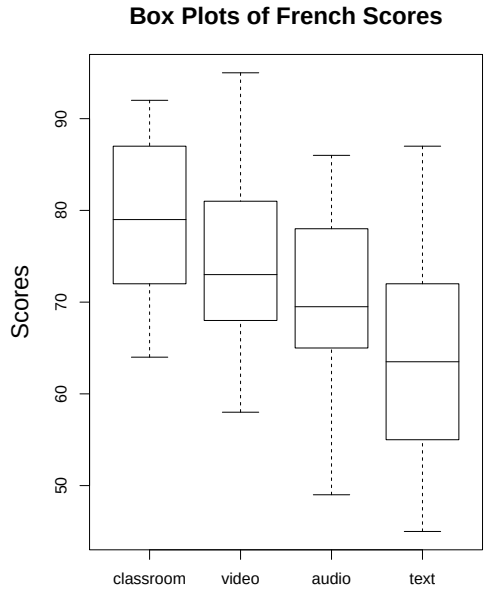
```
>w<-c(83,85,74,70,92,64,72,87,88,75)
>x<-c(95,81,59,68,74,79,72,70,81,58)
names<-c("A","B","C","D","E","F","G","H","I","J")
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
>plot(w,x,main="Scores from Lectures Alone versus Lectures and Audio Tapes",
xlab="Only Lectures", ylab="Both Lectures and Audio Tapes")
identify(w,n=4,x,labels=names,plot=T)
dev.print( postscript, horizontal=FALSE, file="fig8.ps" )
```

After the points have been identified by means of using the cursor, the plot will be saved to `fig8.ps`.

3.1 Box Plots

A single box plot is simple to make. Suppose `w<-c(83,85,74,70,92,64,72,87,88,75)`, then a box plot of this data can be made by: `boxplot(w)`.

A useful aspect of boxplots can been seen when a series of box plots are put side-by-side in the same plot. This arrangement of box plots offers a quick view of the relationship of the sets of data with each other. The following set of commands will create four box plots of the scores in French depending on classroom instruction only given in `w`, the use of only video tapes given



in `w`, the use of only audio tapes given in `y`, and the use of only a textbook given in `z`.

```
w <-c(83,85,74,70,92,64,72,87,88,75)
x <-c(95,81,59,68,74,79,72,70,81,58)
y <-c(86,71,49,63,65,72,78,68,85,65)
z <-c(87,61,45,81,72,67,66,51,55,58)
p<-list(w,x,y,z)
boxplot(p,main="Box Plots of French Scores",
ylab="Scores",xlab="",xaxt="n",horizontal=FALSE)
axis(1,at=c(1,2,3,4), labels=c("classroom","video","audio","text"))
```

The use of the `list` allows the simultaneous plotting of the four box plots in one picture, and the use of `axis` puts nice labels on the x-axis at positions 1, 2, 3, and 4, respectively.

3.2 Confidence Intervals

One of the most important concepts in statistics is the confidence interval. For a small enough population, it might be feasible to obtain all the desired information about it, like the mean and the variance. Almost always, there is limited time, and there are insufficient financial resources to examine the entire population. Instead, a sample of the population is usually drawn which, if

it is done properly, will represent the population in which case the mean of the sample will be close to the mean of the population, and the variance of the sample will be close to the variance of the population. The statistics which are derived from a sample cannot except in extremely rare events be exactly the same as the corresponding statistics of the population. A good sample, nonetheless, does contain accurate information about the population.

By means of confidence intervals, it is possible to infer some characteristics of the population based on the set of experimental data which was obtained from a sampling of the population. The length of the confidence interval will indicate the precision of the data, and its location will indicate the likely region which contains the parameter of interest of the population. The importance of the confidence interval lies in its use to substantiate an inference about the population.

If a very large number of 95 percent confidence intervals are plotted, then, on the average, 95 percent of them will cover the true population mean. We will use R to produce a picture of twenty 95 percent confidence intervals to illustrate the meaning of confidence intervals.

The example begins by defining a function, `ci`. Every command after `{` and before `}` belongs to the function. A function in R is akin to a sub-routine in FORTRAN or to a module in SAS/IML. A vector of `30*n` random numbers is generated from a standard Normal distribution. The vector, `y`, is converted into a matrix consisting of 30 rows of n columns. The lower limit of the 95% confidence interval is

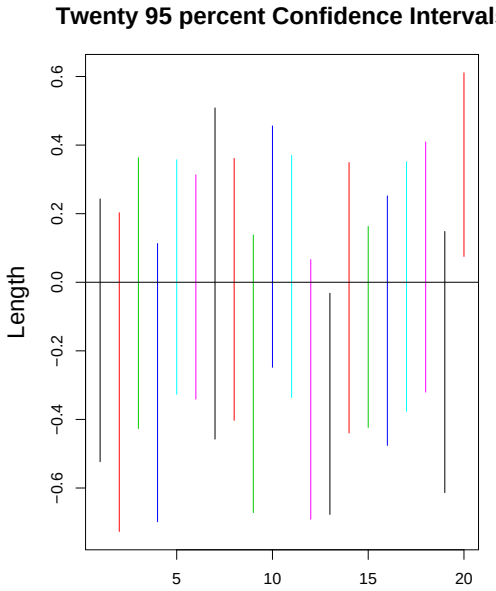
$$\bar{y} - t_{n-1,\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$$

which will be translated in the R language as:
`mean(y)-qt(.975,length(y)-1)*sqrt(var(y)/length(y))`
The upper limit is the same except that a `+` symbol is used instead of the minus sign.

```
>ci<-function(n=20){
>y<-matrix(rnorm(30*n,0,1),nrow=30)
>lower<-apply(y,2,function(y)(mean(y)-qt(.975,
length(y)-1)*sqrt(var(y)/length(y))))
>upper<-apply(y,2,function(y)(mean(y)+qt(.975,
length(y)-1)*sqrt(var(y)/length(y))))
>matplot(cbind(lower,upper),type="n",main=
"Twenty 95 percent Confidence Intervals",ylab="Length")
>z1<-cbind(1:n,1:n)
>z2<-cbind(lower,upper)
>matlines(t(z1),t(z2),lty="solid")
>abline(h=0)
>}
>ci()
```

The last command, `ci()`, will execute the function which will produce the 20 confidence intervals.

The trick which R provides is given by the command `apply`. It means that a function is to be



applied to each record of a column. That is, `apply(y,2,function(y){...})` will apply the function to every column of `y`. The command, `apply(y,1,function(y){...})`, will cause the function to be applied to every row of `y`. `apply` is a peculiar though very handy command which R inherited from S. There is no corresponding command in FORTRAN or in SAS/IML, like `apply`.

The procedure uses `matplot` to plot the end points of the twenty confidence intervals on the plot. Two vectors, `z1` and `z2`, are created which contain the end points of the twenty confidence intervals, but the end points are made invisible by the option, `type="n"`. The x co-ordinates of the lower and upper limits are contained in `z1` and the y co-ordinates for the lower and upper limits are contained in `z2`. The lower and upper limits are connected with a solid line by means of `matlines`. The true population mean is denoted by the horizontal line created by `abline(h=0)`. That 18 out of 20 confidence intervals appear to cover the population mean substantiates the theory that, on the average, 95% of the confidence intervals will contain the population mean.

Chapter 4

Statistics

One can use the basic arithmetic operations of R to calculate any statistic, but it is not necessary to re-invent the wheel for calculating elementary statistics when R contains stored procedures to perform the common computations. For the many examples of this chapter, the set of data will be maintained in a data frame. It is a peculiar structure of R which came from S. A data frame is a collection of variables which have the same length. Its structure is like an array in which the elements of a column correspond to the elements of a variable. Some permissible manipulations of a data frame are like those of an array or matrix, but they cannot be fully extended to matrix algebra. In order to apply the operations of matrix algebra on data frames, a data frame must be converted to a matrix. The name `dd` will be given to the object which will be the data frame of the following examples. The data frame will be initialized by the command:

```
>dd<-data.frame()
```

and it will consist of the four variables:

```
>w<-c(83,85,74,70,92,64,72,87,88,75)
>x<-c(95,81,59,68,74,79,72,70,81,58)
>y<-c(86,71,49,63,65,72,78,68,85,65)
>z<-c(87,61,45,81,72,67,66,51,55,58)
```

To assemble these four variables into the data frame, the following command is executed: `>dd<-data.frame(w,x,y,z)` . The resulting structure of `dd` can be displayed by `>str(dd)` . The command `str()` is like the `proc contents` procedure of SAS. Although the columns of `dd` correspond to the single lettered objects `w,x,y,z`, names may be assigned to the columns of a data frame by the `names()` command so that, for convenience, they will be inherited in the output of subsequent procedures: `>names(dd)<-c("classroom","video","audio","text")` `>str(dd)` will show the contents of the modified data frame, `dd`, and `>summary(dd)` will produce descriptive statistics for all four objects at once. Let us verify some of the statistics.

```
>mean(dd)
>var(dd)
```

Data frames and matrices are not the same even though data frames can be ma-

nipulated, in some ways, as if they were matrices. To see some of the differences between them, we will compare, in the following set of commands, data frames and matrices. The data frame, `dd`, will be converted into the matrix, `m`, by `>m<-as.matrix(dd)`.

Table 4.1: Comparison between a Data Frame and a Matrix

Data Frame	Matrix	Comments
`cor(dd)`	`cor(m)`	No Difference
`hist(dd)`	`hist(m)`	Fails for dd; histogram of union of values of m
`plot(dd)`	`plot(m)`	$\binom{n}{2}$ plots for dd; one plot for all values of m
`matplot(dd,type="l")`	`matplot(m,type="l")`	No Difference
`barplot(dd)`	`barplot(m)`	Fails for dd, four plots for m

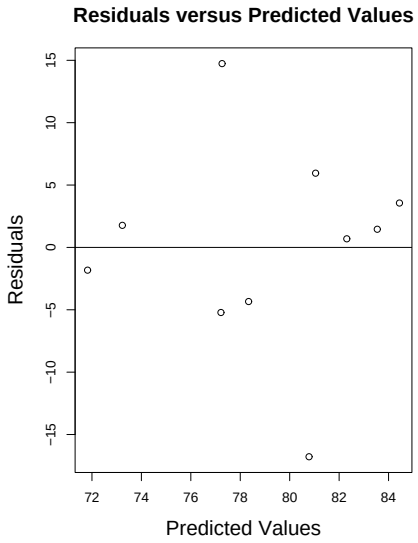
The following examples illustrate the use of producing column and row sums of a table. To take a tally by row: `>margin.table(m,1)`; and by column: `>margin.table(m,2)`. The numeral 1 specifies that the operation be performed by rows, and 2 specifies by columns. The `prop.table` command gives the proportions by row or by column according to the option 1 or 2 as in: `>prop.table(m,1)` for proportions across columns per row or `>prop.table(m,2)` for proportions across rows per column.

In degree of popularity, the method of least squares commands a preeminent role among the stored procedures in R . Special features of the `lm` command will be discussed in more detail in Chapter 5 which addresses advanced procedures. `lm` is but one procedure in R which deals with linear models. To illustrate its use, the next examples will be based on the problem of fitting a linear model $classroom = \beta_0 + \beta_1 video + \beta_2 audio + \beta_3 text + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$. The set of data already exists in the data frame, `dd`, so that the `lm()` may immediately be applied to it. `>lm(dd)` Under the heading of `Coefficients`, the estimates $\beta_0 = 64.39754$, $\beta_1 = 0.50043$, $\beta_2 = -0.05749$, and $\beta_3 = -0.28372$ appear. The same results are produced in the following equivalent formulation. `>lm(w∼x+y+z)`.

The syntax which represents the model has the form: `w∼x+y+z`. All the necessary information for performing an analysis of variance is contained in the output of the `lm` and can be passed to a subsequent procedure like `anova()`: `>anova(lm(w∼x+y+z))`. Rather than type the command, `lm(w∼x+y+z)`, many times over again, the `lm` procedure can be assigned to an object such as: `>w.lm<-lm(w∼x+y+z)` While expressed as an object, the output of the `lm` procedure can be easily analyzed by means of applying various utilities to it, like: `>anova(w.lm)`. In the case of `>fitted(w.lm)`, this procedure produces the fitted values of the linear model while the `resid` procedure will produce the residuals of the linear model: `>resid(w.lm)`. These two procedures make it easy to produce the very important

diagnostic plot of residuals versus predicted values to help determine whether or not the model is a good model.

```
>plot(fitted(w.lm),resid(w.lm),main="Residuals versus Predicted Values",
xlab="Predicted Values", ylab="Residuals")
>abline(h=0)
```



Various tests can be performed on the data like the one of testing the hypothesis that the mean equals 80 at a level of confidence of 95% `>t.test(w,mean=80)` The lower limit and upper limit of a confidence interval can be found by following the appropriate mathematical formulas for producing the lower and upper limits of a confidence interval as was done in Chapter 3 page 28.

```
>mean(w)-sqrt(var(w)/length(w))*qt(.975,length(w)-1)
>mean(w)+sqrt(var(w)/length(w))*qt(.975,length(w)-1)
```

Or the same thing can immediately be done by means of the `t.test` command:

```
>t.test(w,conf.level=.95)
```

Both methods give the same results; their use depends on the predilection of the analyst.

Data frames can also be used to calculate confidence intervals. For example, confidence intervals at a level of confidence of 97% can be produced by:

```
>t.test(dd$classroom,conf.level=.97)
>t.test(dd$video,conf.level=.97)
>t.test(dd$audio,conf.level=.97)
>t.test(dd$text,conf.level=.97)
```

However, the easy way of performing this procedure repetitively according to each variable of the data frame is to execute the `apply` command on `t.test` either by columns:

`>apply(dd,2,t.test)` or by rows: `>apply(dd,1,t.test)`. In so doing, confidence intervals for each of the four variables: `classroom, video, audio, text` will be produced. The output of the command, `apply`, will be saved to the object, `colci` as in:
`>colci<-apply(dd,2,t.test,conf.level=.97)`

Besides the data frame, R has an entity called the *list* which is a collection of objects. That `colci` is a list can be verified as follows: `>is.list(colci)` which returns an affirmative answer. The contents of `colci` is displayed by: `>str(colci)`, and it shows that `coldi` contains information about each of the four variables. Having extracted information pertaining to a specific variable like `classroom`: `>colci$classroom`,

```
colci$classroom

        One Sample t-test

data:  newX[, i]
t = 27.1501, df = 9, p-value = 6.045e-10
alternative hypothesis: true mean is not equal to 0
97 percent confidence interval:
 71.51086 86.48914
sample estimates:
mean of x
       79
```

we see that the list, `colci` contains everything that was gotten by `t.test` when it was invoked only on classroom, i.e. `>t.test(dd$classroom,conf.level=.97)`

```
        One Sample t-test

data:  dd$classroom
t = 27.1501, df = 9, p-value = 6.045e-10
alternative hypothesis: true mean is not equal to 0
97 percent confidence interval:
 71.51086 86.48914
sample estimates:
mean of x
       79
```

There are many ways to obtain the same answer, in R . Some procedures are performed so often that they are given names like `t.test`. In order to save time, the writing of data frames allows an analyst to process a collection of objects altogether, and `list` allows one to collect

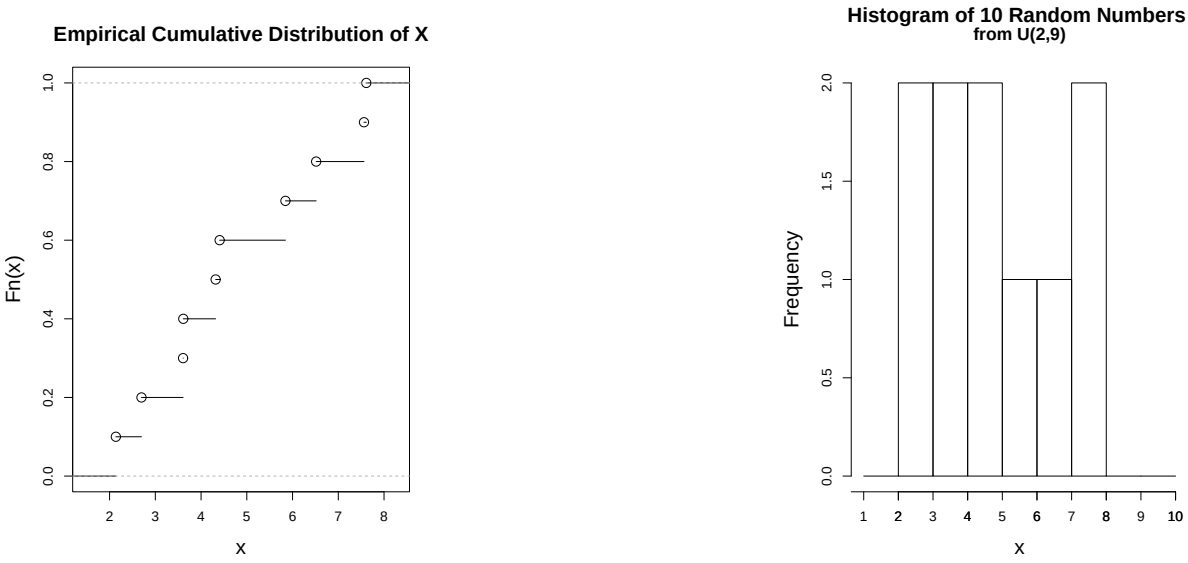


Figure 4.1: Empirical Cumulative Distributions and Histogram

the output into one name from which the pertinent information can be extracted according to the name of a column in the data frame.

The need to find quantiles and probabilities in R is easy to achieve. For example to find the t quantile, $t_{9,.025}$, one will use `>qt(.975,9)`. Conversely, to find the probability $P(t_9 < 2.262157)$, `>pt(2.262157,9)` can be used. Or if four random numbers are needed, they can be generated from a t distribution with 9 degrees of freedom by using: `>rt(4,9)`. Four random numbers from a standard Normal distribution can be obtained by `>rnorm(4)` or `>rnorm(4,mean=0,sd=1)`. The z quantile, $z_{.025}$ is obtained by `>qnorm(.975)`, and the probability $P(z < -1.959964)$ is `>pnorm(-1.959964)`.

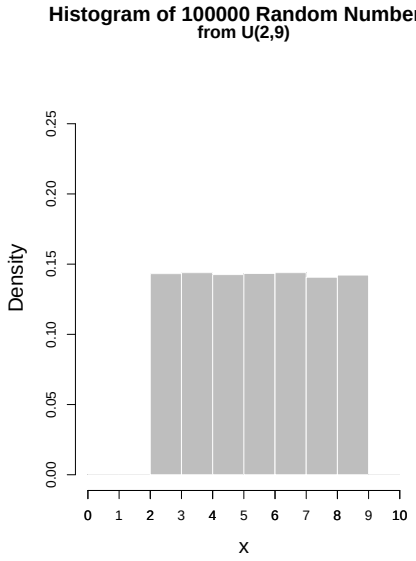
There are similar commands for other distributions, for instance, to produce 10 random numbers from a Uniform distribution, $U(2, 9)$, `runif` is used: `>x<-runif(10,2,9)`. Suppose that these generated numbers are given, one might wonder if they could actually be random and can represent a Uniform distribution in a Monte Carlo technique. The making of a picture will help in satisfying a statisticians curiosity. To that end, a picture of the empirical cumulative distribution function of these 10 random numbers, might be revealing. It is necessary, at first, to invoke the library, `stepfun`, as follows:

```
>library("stepfun")
>plot(ecdf(x))
```

The empirical cumulative probability function as displayed in Figure 4.1 looks linear enough to support the assertion that the random numbers came from a Uniform distribution. Furthermore, a histogram of the same data bears a resemblance to the density function of a Uniform distribution: `hist(x, main="Histogram of 10 Random Numbers \n from U(2,9)")`. By increasing the number of random numbers, the resemblance to a Uniform distribution becomes more apparent:

```
x<-runif(100000,2,9)
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
hist(x,ylim=c(0,.28),breaks=1:10,border="white",col="gray",prob=TRUE,main="")
title("Histogram of 100000 Random Numbers \n")
title( cex.main=1.25, "from U(2,9)")
axis(1,0:10)
```

and it serves as a good illustration of a problem with Monte Carlo techniques in deciding how

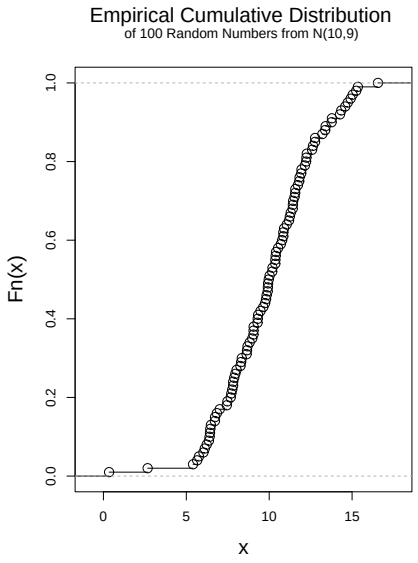


many random numbers is sufficient to create a good empirical distribution function.

Suppose that 100 random numbers are generated from a N(10,3^2)

```
>y<-rnorm(100,10,3)
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
plot(ecdf(y),main="" )
title("Empirical Cumulative Distribution \n")
title( cex.main=1.25, "of 100 Random Numbers from N(10,9) " )
```

```
plot(density(y), main="",xlab="",ylab="" )
title("Empirical Probability Function \n")
 title( cex.main=1.25, "of 100 Random Numbers from N(10,9) ")
```



The plot of the empirical distribution function shows a weak resemblance to a Normal distribution. Yet, statisticians often compare data against a Normal distribution to evaluate the claim that the set of data can be explained by a Normal distribution. Because the values of y were chosen at random from a N(10,9), a probability plot of y should be linear. In comparison, the values of x which were taken from a U(2,9) should produce a probability plot far from linear. Drawing probability plots which are also called qq plots as shown in Figure 4.2 agree with our expectations that the set of random numbers which were generated from a Normal distribution does produce a fairly straight line while those random numbers from a Uniform distribution do not produce a straight qq plot.

```
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
qqnorm(y, main="")
title("qq plot of 100 random numbers \n");title(cex.main=1.25,"from n(10,9)")
qqnorm(x, main="")
title("qq Plot of 100 Random Numbers \n");title(cex.main=1.25,"from U(2,9)")
```

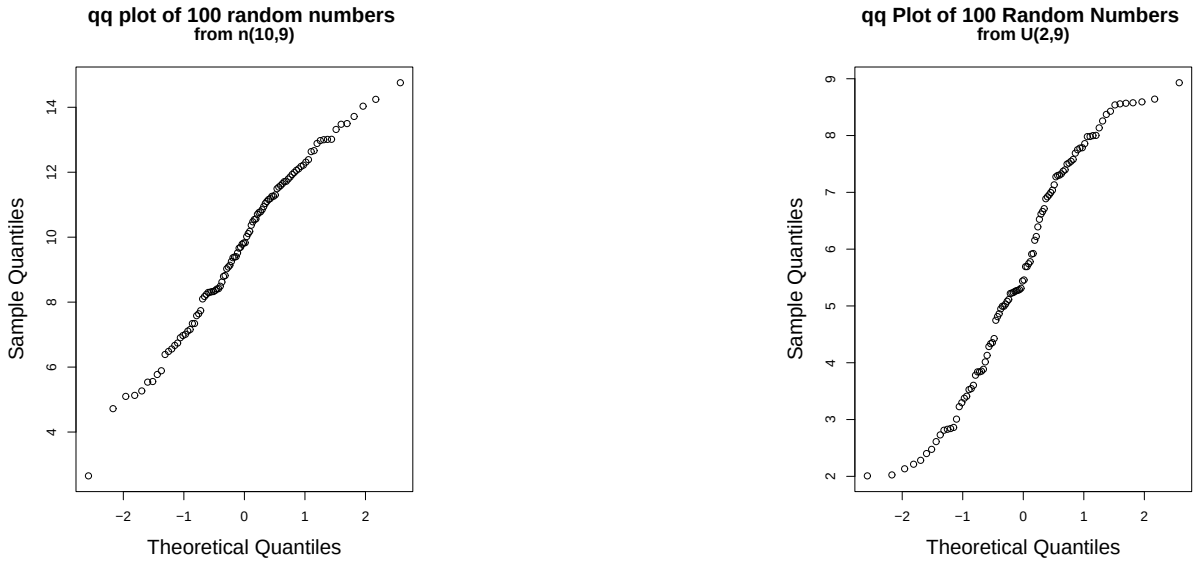


Figure 4.2: qq Plots of Simulated Normal and Uniform Distributions

Chapter 5

Advanced Procedures and Tricks

We will begin this chapter by applying the method of least squares to the problem of estimating parameters of a linear model in two different but equivalent ways. In the first case, the estimates will be determined directly from the theoretical formulation of the problem. In the second approach, estimates will be produced by stored procedures which are found in R for linear models and their accuracy will be verified by the manual computations from theory.

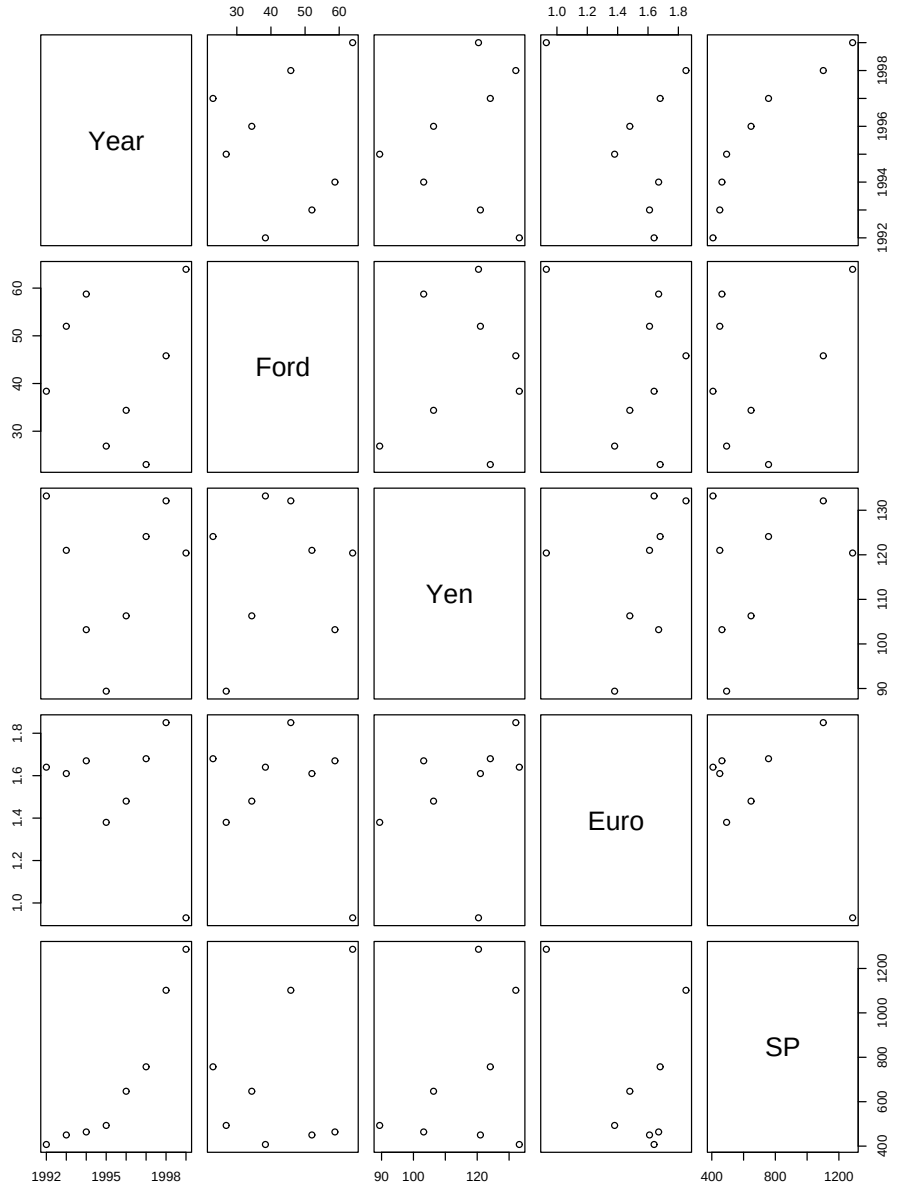
If it is the goal to compose a program which is as parsimonious as possible, in order to achieve a certain sense of elegance, then clever use of stored procedures must certainly be used. On the other hand, sometimes a less than parsimonious program, while not elegant, might be better for making the logic of the program more comprehensible.

Let us examine the price of Ford common stock per share as a function of the exchange rate for Japanese Yen, Euro, and the Standard and Poors (S&P), index as of the beginning of the year. According to theory, the higher the exchange rate of yen per dollar or euro per dollar rises, the more affordable Ford automobiles become relative to Japanese and German imported cars and therefore the greater the demand for Ford common stock.

```
>year<-c(1992,1993,1994,1995,1996,1997,1998,1999)
>ford<-c(38.38,52,58.75,26.88,34.38,23.02,45.81,63.94)
>yen<-c(133.2,121,103.2,89.4,106.3,124.1,132.1,120.4)
>eu<-c(1.64,1.61,1.67,1.38,1.48,1.68,1.85,.93)
>poors<-c(407.36,450.16,463.81,493.15,647.07,757.12,1101.75,1286.37)
>dd<-data.frame(year,ford,yen,eu,poors)
>names(dd)<-c("Year","Ford","Yen","EU","SP")
```

The first order of business in analyzing a set of data is to make a picture of the data. If the theory is correct, there should appear discernible patterns between the variables and the price of Ford stock.

```
>plot(dd)
```



No simple functional relationship is apparent between the price of Ford stock and the other variables of interest upon inspecting the plot of the data. As a result, a statistical analysis of the data will probably produce no useful information, nonetheless, let us assert a linear model like the following:

$$ford = y_i = \beta_0 + \beta_1\ yen + \beta_2\ eu + \beta_3\ sp + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$. It says that the price of Ford stock can be described by a linear combination

of the exchange rates of Yen and Euro and the level of Standard & Poors 500 Index. The linear model can be written more compactly as:

$$\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

In terms of the data, the model appears as the following:

$$\begin{bmatrix} 38.38 \\ 52 \\ 58.75 \\ 26.88 \\ 34.38 \\ 23.02 \\ 45.81 \\ 63.94 \end{bmatrix} = \begin{bmatrix} 1 & 133.2 & 1.64 & 407.36 \\ 1 & 121 & 1.61 & 450.16 \\ 1 & 103.2 & 1.67 & 463.81 \\ 1 & 89.4 & 1.38 & 493.15 \\ 1 & 106.3 & 1.48 & 647.07 \\ 1 & 124.1 & 1.68 & 757.12 \\ 1 & 132.1 & 1.85 & 1101.75 \\ 1 & 120.4 & .93 & 1286.37 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \\ \epsilon_8 \end{bmatrix} \tag{5.1}$$

In the theory of statistical linear models, $\mathbf{Y}$, is usually called the *response variable* and the variables, `yen`, `eu`, and `sp` are called the explanatory variables. Some authors might call them the predictor variables and others might call them the independent variables. The matrix, $\mathbf{X}$, is called the design matrix. We will use the design matrix and the vector of the response variable in R to calculate the estimate of the vector of parameters, $\boldsymbol{\beta}$. The design matrix must be constructed from the data.

We will use three approaches to construct the design matrix. Constructing the vector of data for the response variable is easy:

```
>y<-dd$Ford
```

1. The most logical approach to construct the design matrix would be to append a vector of all 1's to the second through fourth columns of the data frame, `dd`, as in:
   ```
   >x<-cbind(rep(1,length(y)),as.matrix(dd[,3:5]))
   ```
   The vector of all 1's is produced by `rep(1,length(y))`. The second through fourth columns of the data frame, `dd`, are converted into a matrix by means of the command, `as.matrix()`. Then both components are put together by the `cbind` command to form the design matrix.

2. A simpler approach is to exploit a trick by which R will repeatedly cycle through a short vector until the operation is done. Instead of creating a vector of 1's which is congruent in dimension with the matrix which was created from `dd`, the vector of a single element is used so that as R combines the two vectors, it will cycle through the short one until the `cbind` operation is completed. It is a feature of R which, although it is different from our accustomed way of reasoning, will make the program more parsimonious.
   ```
   >x<-cbind(1,as.matrix(dd[,3:5]))
   ```

3. A third approach utilizes a special command in R for producing the design matrix namely:
```
>x<-model.matrix(Ford~Yen+EU+SP,data=dd)
```
This command was written if full detail; however, if the data frame is attached, its variables will automatically be present so that the data frame need not be mentioned. For convenience, a data frame can be brought into the memory of the current session of R by means of the command, `attach`, as in:
```
>attach(dd)
```
Because the data frame is attached to the current workspace, it will be included in the search path of R therefore, the `model.matrix` command could have been written without reference to the data frame, `dd`:
```
>x<-model.matrix(Ford~Yen+EU+SP)
```
When the data frame is no longer needed, then `detach(dd)` will remove it from the search mechanism of the current workspace.

From the theory of numerical analysis where statisticians have taken the method of least squares, that β which minimizes the sum of squared errors, SSE, corresponds to that line which best fits the data. By imposing the condition that $SSE = \sum_{i=1}^{n} \epsilon_i^2$ be a minimum and the assumption that $\epsilon \sim N(0, \sigma^2)$, the unbiased estimator of $\boldsymbol{\beta}$ can be written in matrix form as:

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \tag{5.2}$$

This equation constitutes the most important formula in the theory of linear models, and for our purposes it gives the recipe for calculating $\widehat{\boldsymbol{\beta}}$. The key command is the one for inverting the matrix $\mathbf{X}'\mathbf{X}$ which in R is done by `solve()`:

```
>solve(t(x)%*%x)%*%t(x)%*%y
```

```
[,1]
(Intercept)   42.066649253
 Yen           0.208504528
 EU          -17.800308422
 SP            0.005467642
```

The command, `%*%`, is matrix multiplication; `t()` is the transpose operator. For reference later, the least squares estimates will be saved to the object, `betahat`.

```
>betahat<-solve(t(x)%*%x)%*%t(x)%*%y
>betahat
```

```
  (Intercept)  42.066649253
  Yen           0.208504528
  EU          -17.800308422
  SP            0.005467642
```

Although the mathematical formula for $\widehat{\beta}$ can be used as a recipe to do the computation, the use of linear models is such a popular technique in statistics, a procedure called, `lm`, already exists in R which will produce estimates of a linear model. The same estimates which have been assigned to `betahat` can also be produced by: `>lm(Ford~Yen+EU+SP,data=dd)` and the results are shown below:

```
Call:
lm(formula = Ford ~ Yen + EU + SP)

Coefficients:
(Intercept)          Yen           EU           SP
  42.066649     0.208505   -17.800308     0.005468
```

or because the data frame `dd` had been already attached by the command, `attach(dd)`, it is sufficient to write:

```
>lm(Ford~Yen+EU+SP)
```

The formula which appears in the `lm` command looks like the mathematical expression for the linear model except that reference to the β's is missing. This formulation is the typical way to write a model in R . Whether to produce the estimates by the `lm` procedure or by the mathematical formula for $\widehat{\beta}$ is a matter or personal preference. The advantage of using a stored procedure like `lm` is that it produces a package of other useful statistics.

What separates numerical analysis and statistics is the assumption which statisticians make that ϵ_i is a random variable. By virtue of that assumption, confidence intervals and the testing of hypotheses can be made to substantiate an inference which is drawn from the data. In conjunction with the `lm` command in R , there are additional procedures which address various topics of inference. For example, the composition of `summary()` with `lm()` functions will produce standard computations for linear models.

```
>summary(lm(Ford~Yen+EU+SP))
```

```
Call:
lm(formula = Ford ~ Yen + EU + SP)

Residuals:
      1       2       3       4       5       6       7       8
 -4.494  10.901  22.356 -11.959  -7.044 -19.157   3.106   6.290

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  42.066649  57.148186   0.736    0.503
Yen           0.208505   0.542683   0.384    0.720
EU          -17.800308  30.459085  -0.584    0.590
SP            0.005468   0.026440   0.207    0.846

Residual standard error: 17.66 on 4 degrees of freedom
```

```
Multiple R-Squared: 0.1845,     Adjusted R-squared: -0.427
F-statistic: 0.3017 on 3 and 4 DF,  p-value: 0.8238
```

From the `summary()` command there appears a list of the residuals of the model, estimates of each β along with sufficient information to construct confidence intervals for each of them, and the F test statistic for testing the hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3 = 0 \; vs \; H_1 : otherwise$. The results of the `lm` procedure may be preserved in an object like `stock.lm`.

```
>stock.lm<-lm(Ford~Yen+EU+SP)
>summary(stock.lm)
```

. Having applied the `summary()` command to the object, `stock.lm`, the same results are produced as was printed by means of `lm(Ford~Yen+EU+SP)` alone. Other commands like `anova` can be applied to the object `stock.lm`. In the case of `anova`, the analysis of variance (ANOVA) table will be printed.

```
>anova(stock.lm)
```

```
Analysis of Variance Table

Response: Ford
   Df  Sum Sq Mean Sq F value Pr(>F)
Yen         1   38.97   38.97  0.1249 0.7416
EU          1  230.05  230.05  0.7375 0.4389
SP          1   13.34   13.34  0.0428 0.8463
Residuals   4 1247.69  311.92
```

Entries in the ANOVA table confirm that for all practical purposes, $\beta_1 = \beta_2 = \beta_3 = 0$. The price of Ford stock cannot be predicted by the proposed model based on the available data. Not only must a statistician consider the F test statistic in evaluating the adequacy of a linear model, but he needs to examine the plot of residuals versus predicted values. The object, `stock.lm`, contains all the usual information associated with the estimation of the parameters of a linear model.

It is possible to produce predicted values by following the mathematical formula: $\mathbf{x}\widehat{\boldsymbol{\beta}}$ or by applying the stored procedure `predict()` or `fitted()` on the object, `stock.lm` as was first explained on page 32:

```
>predict(stock.lm)
```

. Similarly, the residuals can be produced by following the mathematical formula, $\mathbf{y} - \mathbf{x}\widehat{\boldsymbol{\beta}}$, which when written in R is:

```
>y-x%*%betahat
```

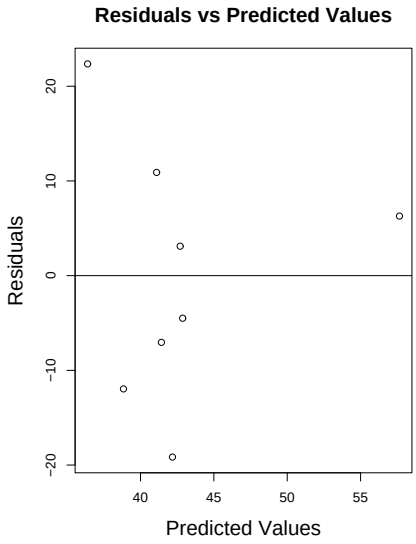
or more conveniently by applying the stored procedure, `resid()`, on the object, `stock.lm`:

```
>resid(stock.lm)
```

. Having the residual values and predicted values in hand, a plot of them will show if there is flaw in the model.

```
>plot(predict(stock.lm),resid(stock.lm),main="Residuals vs Predicted Values")
>abline(h=0)
```

By inspecting the plot of residuals versus predicted values,



there does appear to be a pattern in the residuals versus predicted values, consequently, the formulation of the model must be flawed.

Based on this exercise, we see that there is more than one way to solve a problem. One may use, on the one hand, the basic arithmetic and matrix algebra capabilities of R while following a mathematical formula or one may use stored procedures which are provided in R . The stored procedures are convenient when studying common statistical problems. Most of the time, problems are not standard ones especially when doing research, so that it usually is necessary to compose unique functions for the given problem which means that a good mathematical formulation of the problem must have already been developed. Sometimes clever use of stored programs although the original intention of designing them might not solve the problem completely can nonetheless improve the efficiency of a custom designed program.

Table 5.1: **Analysis of Variance for Fitting Regression for the General Linear Model**

Source of Variation	df	Sum of Squares	Mean Sum of Squares	F statistic
Mean	1	$SSM = n\bar{y}^2$		
Regression	r-1	$SSR_{(m)} = \widehat{\boldsymbol{\beta_m}}' \mathbf{X_m}' \mathbf{Y}$	$MSR = \frac{SSR_{(m)}}{r-1}$	$F = \frac{MSR}{MSSE}$
Residual Error	n-r	SSE=SST-SSM-$SSR_{(m)}$	$\widehat{\sigma^2} = MSSE = \frac{SSE}{n-r}$	
Total	n	$SST = \sum_{i=1}^{n} y_i^2$		

In the next few commands, we will verify the computations of the `lm` procedure by computing each entry in the ANOVA table and the confidence intervals of each β_i. The mathematical

formulas of the ANOVA table are shown in Table 5.1. By means of these formulas, the entries of the ANOVA table will be produced and compared with the respective results of the `lm` procedure. The ANOVA table is reproduced below in Table 5.2 in which the corresponding commands in R are written.

Table 5.2: **Formulas for Analysis of Variance Written in R**

Source of Variation	df	Sum of Squares	Mean Sum of Squares	F statistic
Mean	1	`SSM<-nrow(x)*mean(y)^2`		
Regression	`ncol(x)-1`	`SSRm<-t(bhatm)%*%t(tx)%*%y`	`MSSRm<-SSRm/(ncol(x)-1)`	`Fm<-MSSRm/MSSE`
Residual Error	`nrow(x)-ncol(x)`	`sum(resid(stock.lm)^2)`	`MSSE<-SSE/(nrow(x)-ncol(x))`	
Total	`nrow(x)`	`sum(y^2)`		

To test the hypothesis that $H_0 : \beta_1 = \beta_2 = \beta_3 = 0 \; vs \; H_1 : otherwise$, there is usually no interest in the significance of the intercept, β_0. Rather, the significance of the other coefficients of the linear model commands attention. To remove β_0 from consideration in the ANOVA table, the entries are corrected for the mean. To that end, we will construct a vector which contains only the means of variables, Yen, EU, and SP.

```
>mu<-rep(mean(dd[,3:5]),each=8,1)
```

Here we see a new option in using the `rep` procedure, namely the option `each`. Each mean will be repeated 8 times and each group of 8 repetitions will be repeated only once. Another but more tedious way to construct μ would be to construct a 3×8 matrix and assign the appropriate mean to each cell of the matrix. Having had constructed `mu`, the design matrix must be corrected for the mean by subtracting the mean from each element. Because the information provided by the means is now dispersed throughout the design matrix, the column of 1's which is no longer needed will be omitted.

This new design matrix which must be corrected for the mean can be constructed from the original data frame, `dd`, by subtracting `mu` from each element and by omitting the inclusion of the column of 1's as in:

```
>xm<-as.matrix(dd[,3:5])-mu
>xm
```

The revised design matrix which has been corrected for the mean now can be used to estimate the parameters of the model other than the intercept, β_0, about which we are not interested.

```
>bhatm<-solve(t(xm)%*%xm)%*%t(xm)%*%y
bhatm
```

A comparison of $\widehat{\boldsymbol{\beta}}$ with $\widehat{\boldsymbol{\beta}_m}$ verifies that the stored procedure, `lm`, produces the same esti-

mates as required.

```
>bhatm
>betahat
```

Table 5.3: Comparison of Estimates from lm and the Equations

Denomination	lm	$(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$
(Intercepts)	42.066649	—
Yen	0.208505	0.208504528
EU	-17.800308	-17.800308422
SP	0.005468	0.005467642

It is obvious that that `betahat` and `bhatm` are the same except that $\widehat{\beta}_0$ or what is called the intercept is missing from `bhatm`. The rest of the entries in the ANOVA table can be easily computed using R .

```
>SSRm<-t(bhatm)%*%t(tx)%*%y
>MSSRm<-SSRm/(ncol(x)-1)
>Fm<-MSSRm/MSSE
```

The value of the F test statistic which corresponds to the design matrix corrected for the mean is the same as the F test statistic which was produced from `summary(stock.lm)`.

```
>Fm
```

```
          [,1]
[1,] 0.3017437
```

```
>summary(stock.lm)
```

```
Call:
lm(formula = Ford ~ Yen + EU + SP)

Residuals:
      1       2       3       4       5       6       7       8
 -4.494  10.901  22.356 -11.959  -7.044 -19.157   3.106   6.290

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  42.066649  57.148186   0.736    0.503
Yen           0.208505   0.542683   0.384    0.720
EU          -17.800308  30.459085  -0.584    0.590
```

```
SP            0.005468   0.026440   0.207    0.846

Residual standard error: 17.66 on 4 degrees of freedom
Multiple R-Squared: 0.1845,     Adjusted R-squared: -0.427
F-statistic: 0.3017 on 3 and 4 DF,  p-value: 0.8238
```

Having verified that the `lm` produces the correct F test statistic, should one go through the trouble of writing an independent program to produce a statistic which a stored procedure in R can already do? No computer program should ever be trusted. It is always prudent to verify that a computer program functions correctly either by reproducing the results by an independent method as we have done here or by using a set of canned data for which the exact answer is known and compare the results of the computer program with the exact results. A highly desirable feature of R is that it is licensed under the GPL so that the logic of the of any procedure of R can be examined and studied at anytime by anyone. If an error should be discovered in the source code, it can be announced and a solution, if one was found can be submitted to the developers of R for their consideration.

Besides verifying the accuracy of the ANOVA table, the accuracy of the the confidence intervals for the β's which the `lm()` procedure produces will also be verified. Accordingly, the diagonal elements, a^{ii}, of the inverse matrix of $\mathbf{X}'\mathbf{X}$ must be computed, in order to determine the lower and upper limits of the confidence interval. The appropriate formula for a particular β_i is

$$\widehat{\beta}_i \pm \widehat{\sigma} t_{n-r,\frac{\alpha}{2}} \sqrt{a^{ii}}$$

The diagonal elements of $(\mathbf{X}'\mathbf{X})^{-1}$ are obtained by means of the command: `diag()`.

```
>a<-diag(solve(t(x)%*%x))
>lower<-betahat-rep(sqrt(MSSE)*qt(.95,6),4)*as.matrix(sqrt(a))
>upper<-betahat+rep(sqrt(MSSE)*qt(.95,6),4)*as.matrix(sqrt(a))
>ci.beta<-cbind(lower,upper)
>colnames(ci.beta)<-c("lower","upper")
>ci.beta
```

```
        lower        upper
     -68.98257581 153.11587431
Yen  -0.84602579   1.26303484
EU  -76.98780055  41.38718370
SP   -0.04590937   0.05684466
```

All the confidence intervals straddle 0 and, in so doing, confirm that the F test statistic of `0.3017437` implies that the hypothesis, $H_0 : \beta_1 = \beta_2 = \beta_3 = 0$, cannot be rejected unless the level of significance exceeds the p-value of :

```
>1-pf(.3017437,3,4)
```

```
[1] 0.8237515
```

which is exactly what was produced by `summary(stock.lm)`. A typical level of significance by tradition is .05; therefore, the current set of data strongly invalidates the model and until another set of data is obtained it appears that the proposed linear model does not account for the price of Ford stock.

The test statistic, $T = \frac{\widehat{\beta}_i}{\widehat{\sigma}\sqrt{a^{ii}}}$, is used for testing the hypothesis $H_0 : \beta_i = 0 \; vs \; H_1 : \beta_i \neq 0$ is the following:

```
>T<-betahat/(sqrt(a)*sqrt(MSSE))
>T
```

```
                [,1]
(Intercept)  0.7360977
Yen          0.3842107
EU          -0.5844006
SP           0.2067970
```

These values of T agree exactly with the values produced by the `lm` procedure under the column, `t value`.

```
>summary(stock.lm)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  42.066649  57.148186   0.736    0.503
Yen           0.208505   0.542683   0.384    0.720
EU          -17.800308  30.459085  -0.584    0.590
SP            0.005468   0.026440   0.207    0.846
```

From the same procedure, the last column represents the p-values for testing $H_0 : \beta_i = 0 \; vs \; H_1 : \beta_i \neq 0$. The p-value is for a two-sided test like the one with which we have been using. It is defined to be: $p = 2P(t_{n-1} > T)$. For example, `>p<-2*(1-pt(abs(T),4))` where 4 is the degrees of freedom. In conclusion, we reproduced the essential results of the `lm` procedure and hence verified that the `lm` procedure produced the correct statistics.

In regard to the foregoing discussion, a noteworthy characteristic of $\mathbf{X}'\mathbf{X}$ is that it is a symmetric real matrix; therefore, according to the theory of linear algebra it can be expressed by a sum of its eigenvectors and eigenvalues. The command, `eigen()`, will produce the eigenvalues and eigenvectors of a symmetric real matrix. For instance, `>eigen(t(x)%*%x)`. It produces a list of two components, values and vectors. In fact,

`>str(eigen(t(x)%*%x))` confirms that `eigen` produces a list and a component is extracted in the usual way with $:

```
>val<-eigen(t(x)%*%x)$values
>vec<-eigen(t(x)%*%x)$vectors
```

According to the theory of linear algebra,

$$X'X = \sum_i \lambda_i \mathbf{e}_i \mathbf{e}_i'$$

where λ_i is an eigenvalue and $\mathbf{e}_i$ is its associated eigenvector, or in terms of R

```
>val[1]*vec[,1]%*%t(vec[,1])+val[2]*vec[,2]%*%t(vec[,2])+
val[3]*vec[,3]%*%t(vec[,3])+val[4]*vec[,4]%*%t(vec[,4])
```

will, when executed, produce $\mathbf{X}'\mathbf{X}$ as it should. To eliminate the tedious job of typing many repetitive commands, a loop can be employed:

```
>tx<-0
>i<-1
>while(i<=4){
>tx<-tx+val[i]*vec[,i]%*%t(vec[,i])
>i<-i+1
>}
```

The first two lines set the initial values for `w`, the answer, and `i` the index in the looping mechanism. As long as `i` is less than or equal to 4, the loop will continue. With each iteration of the loop, the index, `i`, is incremented by 1. Eventually, `i` will exceed 4 and the loop will terminate. Actual computation which is saved in `tx` gotten by following the theoretical decomposition of symmetric matrix produces the original matrix, $X'X$: The displays of `>tx` and `>t(x)%*%x` show that they are identical.

The command `while` belongs to a family of commands which is known as control language. Statements for conditioning on certain criteria, loops, and functions will be discussed in the next chapter.

Chapter 6

Control Language

A function is to R what a sub-routine is to FORTRAN and a module is to SAS/IML. The form of a function looks like `function(x,y,z)` where x, y, and z are arguments and the values which are assigned to them are passed to the contents of the function. For example, the following function will convert yen to dollar:

```
yen2dol<-function(yen,exc){
dol<-exc*yen
return(dol)
}
```

The variable, `exc`, is the exchange rate in dollars per yen. After the function has been defined, then its use is simple. The function `yen2dol()` will convert ¥50000 to dollars at an exchange rate of $/¥=1/110:

```
 >yen2dol(50000,1/110)
```

```
[1] 454.5455
```

Suppose we had several amounts of yen like ¥100, ¥50000, and ¥97625800, then an application of the function `yen2dol` will do the conversion.

```
 >yen2dol(c(100,50000,97625800),1/110)
```

```
>[1] 9.090909e-01 4.545455e+02 8.875000e+04
```

If the result which is expressed as it is in scientific notation is not deemed presentable, then to make the results more attractive, we will use the command, `prettyNum`, with the option to insert a comma to separate groups of three digits.

```
 >prettyNum(yen2dol(c(100,50000,97625800),1/110),big.mark=",")
```

```
[1] "0.9090909" "454.5455"  "887,507.3"
```

We can do even better by affixing the dollar symbol to the results by means of the paste command. The paste command is a useful device to combine letters and symbols together to form names.

```
>paste("$",prettyNum(yen2dol(c(100,50000,97625800),1/110),big.mark=","),sep="")
$0.9090909 $454.5455  $887,507.3
```

An even prettier result can be produced with the `cat` command. In the event that there might be a large list of values of yen which one would like to convert to dollars, a `for` loop could be employed. It will iterate through the list of values until all the conversions have been done.

```
Y<-c(100,50000,97625800)
for (x in Y){
cat("¥",prettyNum(x,big.mark=","),," is ","$",
prettyNum(round(yen2dol(x,1/110),2),big.mark=","),".",sep="","\ n")
}
```

```
¥100 is $0.91.
¥50,000 is $454.55.
¥97,625,800 is $887,507.3.
```

(On a keyboard when working on a Linux computer, press `shift-alt \%`, to make the yen symbol, ¥).

The `for` loop uses `x` as an index which assumes at each iteration of the loop a value in the vector, `Y`, which the function, `yen2dol` converts to dollars. The result of `yen2dol` is rounded to two decimal places by the `round(,2)` command and then the result of that rounding is made pretty by the `prettyNum` command. Finally, the pretty result is concatenated together with ¥, the value of the yen, the word *is*, and then the value of the dollar in one step. Still, another improvement to the `for` loop can be made by making a single and succinct command to do everything that has already prescribed. The following function, `y2d()`, will include all the pertinent commands and can be applied to a large list of figures to convert at a given exchange rate.

```
y2d<-function(Y,exc){
for (x in Y){
cat("¥",prettyNum(x,big.mark=","),," is ","$",
prettyNum(round(yen2dol(x,1/exc),2),big.mark=","),".",sep="","\n")
}
}
>y2d(Y,110)
```

This is an example of nested functions where the index of a `for` loop is evaluated by another function until the loop terminates. It might be desirable to convert a large set of values in yen to dollars. The set might be in an text file. If it can be brought into the current workspace and the function `y2d` can be applied to it, then the conversion will be easy. If ASCII file like `yen.txt` has been created with the values: 200 300 400 500 600 700 800, then the contents of `yen.txt` is brought into `R` by means of the `read.table` command as follows:

```
>Y<-read.table(file="yen.txt")
>str(Y)
>Y
```

The function, `y2d`, will convert all the values contained in `Y` to dollars.

```
>y2d(Y)
```

```
¥200 is $1.82.
¥300 is $2.73.
¥400 is $3.64.
¥500 is $4.55.
¥600 is $5.45.
¥700 is $6.36.
¥800 is $7.27.
```

Similar in purpose to the `for` loop is the `while` loop. This loop will continue to iterate until the condition which is specified in it is satisfied. For example, 20 values of yen in multiple values of 1000 can be converted through a `while` loop.

```
i<-1
while (i <=20){
cat("¥",prettyNum(i*1000,big.mark=","),," is ","$",
prettyNum(round(yen2dol(i*1000,1/110),2),big.mark=","),".",sep="","\n")
i<-i+1
}
```

An important command for control the progress of a program is the conditional which is initiated by the familiar `if` statement. For example,

```
x<-c(0,2,4,6,8)
y<-c(1,3,5,7,9)
i<-1
while(i<=10){
if (i%%2==0){
 cat(i, "is an even number","\n")
 }
if (i%%2==1){
 cat(i, "is an odd number","\n")
 }
i<-i+1
}
```

Note that the logical equal symbol, `==`, is used to describe the condition which the `if` statement must evaluate before the subsequent command will be executed. Recall that the `%%` is the symbol for modulo, so that `i%%2` is equal to 0 if `i` is even, and `i%%2` is equal to 1 if `i` is odd.

Chapter 7

Application to Finance

7.1 Monte Carlo Simulation

In an asset price process, the price of an asset is given by the stochastic differential equation known as a stochastic volatility model:

$$dS = \kappa(\mu - S)dt + \sigma S dW$$

where $dW(t)$ is a Weiner process, σ is the volatility, μ is the effective return of S, κ is the speed of adjustment or reversion to the mean. A Monte Carlo technique can be used to solve this differential equation by numerical techniques. The differential equation must first be written in a form which will use discrete increments in the variables. To that end, we will use the following equation in the Monte Carlo simulation:

$$\triangle S = S_i - S_{i-1} = \mu S_{i-1}\triangle t + \sigma S_{i-1}\epsilon_i\sqrt{\triangle t}$$

Suppose the price path is S_0, S_1, S_2, ... , S_{100}, for example. Values of ϵ_i will come from a random number generator for a Standard Normal distribution, `rnorm`. We will call $\triangle t$ by the name, `deltat`, $\triangle S$ will be called, `deltas`, and the names of rest of the symbols will be evident. For an initial price of s0=62, s1 can be found as follows:

Stage 1. Simple beginning.

```
mu<-.15
sigma<-.15
deltat<-1/360
s0<-62
deltas<-mu*s0*deltat+sigma*s0*rnorm(1)*sqrt(deltat)
s1<-s0+deltas
```

```
>s1
```

Stage 2. Producing a price path for 100 periods with a `for` loop.

```
s<-numeric()
mu<-.15
sigma<-.15
deltat<-1/360
s0<-62
for (i in 1:100){
deltas<-mu*s0*deltat+sigma*s0*rnorm(1)*sqrt(deltat)
s1<-s0+deltas
s2<-cbind(i,deltas,s1)
s<-rbind(s,s2)
s0<-s1
i<-i+1
}
```

`>s` will show all 100 values of the price path, and a picture of them appears in Figure

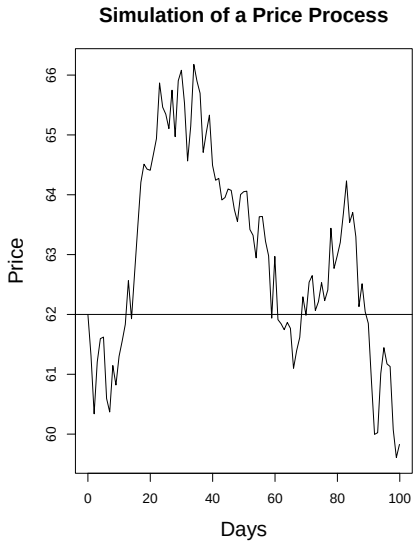


Figure 7.1:

7.1. In the program, we notice that that `rbind` command appends the new value of S onto the bottom of the previously computed values of S. The vector `s` is initialized by the command `numeric` otherwise when the program is run again `s` will contain 200 residual entries from the previous run. Nothing is more impressive that a picture of the price path. Therefore, the commands which were used to make the plot of the simulated stock price

appearing in Figure 7.1 are:

```
>plot(s[,1],s[,3],type="l")
>abline(h=62)
```

Stage 3. The function, `price`, and enhancements to the graph.

```
price<-function(mu,sigma,deltat,s0,n,graph=0){
s00<-s0
s<-cbind(0,0,s0)
i<-1
for (i in 1:n){
deltas<-mu*s0*deltat+sigma*s0*rnorm(1)*sqrt(deltat)
s1<-s0+deltas
s2<-cbind(i,deltas,s1)
s<-rbind(s,s2)
s0<-s1
i<-i+1
}
if (graph==1){
plot(s[,1],s[,3],type="l", main="Simulation of a Price Process",
xlab="Days",ylab="Price")
abline(h=s00)
}
if (graph==0) return(s)
}
```

S is a 3 dimensional vector; $S[1]$ is the time, and $S[3]$ is the simulated stock price.

```
>price(mu=.15,sigma=.15,deltat=1/360,s0=62,n=100,graph=0)
>price(mu=.15,sigma=.15,deltat=1/360,s0=62,n=100,graph=1)
```

If `graph` is set to 0, then the price of 100 periods will be produced via the `return` statement in the function, or if `graph` is set to 1, then the price of 100 periods will be displayed in a graph. Each path is a random walk. Suppose many thousands of them are plotted. In what region will they, on the average, lie? The upper and lower envelope will define the region in which a price path will lie with approximately 95% confidence.

Stage 4. Calculation of the upper and lower limits of the envelope of price paths at a level of 95% based on 500 simulations.

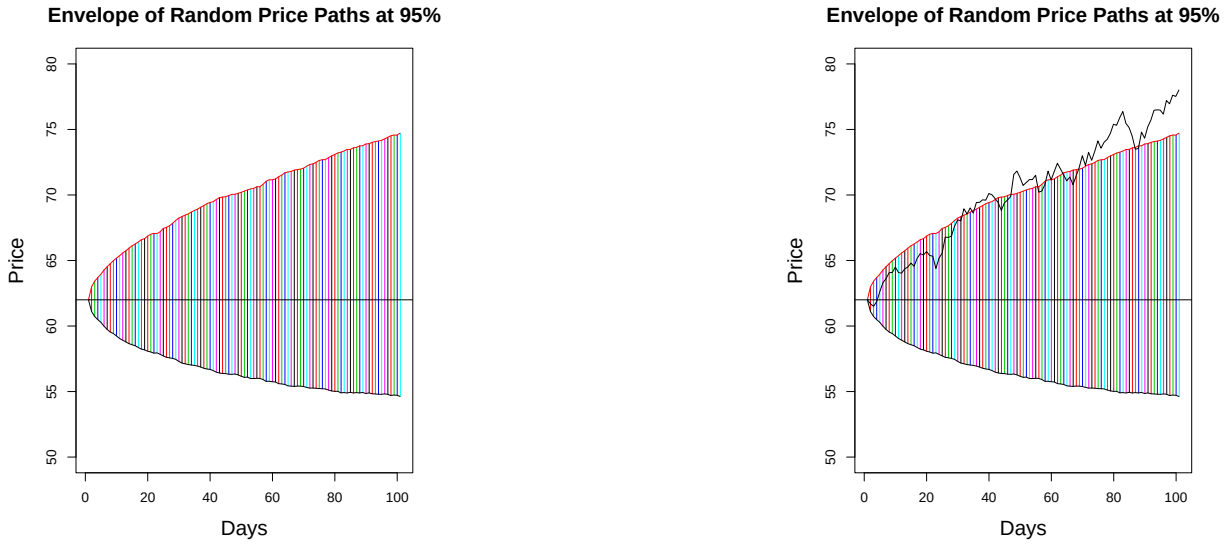


Figure 7.2: Envelope of random price paths and a simulated price path superimposed on an envelope.

```
n<-101
ss<-matrix(0,n,500)
j<-1
while (j <=500){
ss0<-price(mu=.15,sigma=.15,deltat=1/360,s0=62,n=100,graph=0)
ss[,j]<-ss0[,3]
j<-j+1
}
lower<-apply(ss,1,function(ss)(mean(ss)-
 qt(.975,length(ss)-1)*sqrt(var(ss)/1)))
upper<-apply(ss,1,function(ss)(mean(ss)+
 qt(.975,length(ss)-1)*sqrt(var(ss)/1)))
matplot(1:n,apply(ss,1,mean),ylim=c(50,80),type="l",
main="Envelope of Random Price Paths at 95%",xlab="Days",ylab="Price")
z1<-cbind(1:n,1:n)
z2<-cbind(lower,upper)
matlines(t(z1),t(z2),lty="solid")
matlines(z1,z2,lty="solid")
abline(h=62)
```

Plots of 100 confidence intervals where each one which was produced by the set of 500 simulations of the price paths are plotted and they illustrate an example of nested simulations. The average of all paths is reflected by the single price path depicted in black while the confidence intervals form an envelope within which a price path will probably lie if one is produced.

Stage 5. Check the envelopes.

```
plotprice<-function(){
sss<-price(mu=.15,sigma=.15,deltat=1/360,s0=62,n=100,graph=0)
matplot(1:101,sss[,3],type="l",add=T)
}
```

```
>plotprice()
```

A superimposed simulated random price path is shown on the right in Figure 7.2. The envelope of confidence is not quite right because each of the individual confidence intervals are presumed to be independent of the others. But they are in fact dependent because the process of calculating the price is a Markov chain process. They therefore should be wider; however, as a first approximation, the one which we produced is good enough.

7.2 Yield to Maturity

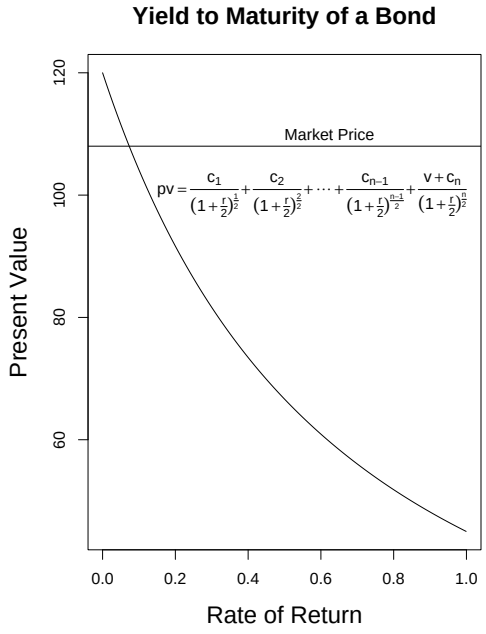
In this problem,

$$pv = \frac{c_1}{(1+r)^{\frac{1}{2}}} + \frac{c_2}{(1+r)^{\frac{2}{2}}} + \cdots \frac{v+c_n}{(1+r)^{\frac{n}{2}}}$$

where `pv` is the present value, c_i is the coupon, `r` is the interest rate, and `v` is the face value of the bond. The interest rate, `r`, must be determined given the market price of the bond, the face value of the bond, the coupon rate, the number of years and periods per year. A picture of the yield to maturity curve will illustrate the nature of the problem. It will be called `ytm` for yield to maturity. It needs to account for the possibility of paying the coupon annually or semi-annually. The function, too, must account for the possibility of whether or not the face value of the bond will be paid with the last coupon. In the present case of defining `ytm`, it is assumed that all coupons are equal. Since the unknown quantity is `r`, we will name it, `x` in `ytm` to emphasize that it is the unknown quantity.

```
ytm<-function(ind=0,p=1,cr,v,n,x){
c<-cr*v/p
w0<-0
for (i in 1:(p*n-1)){
w<-w0+c/(1+x/p)^(i/p)
w0<-w
}
i<-i+1
w<-w0+(v*ind+c)/(1+x/p)^(i/p)
return(w)
}
```

When the option `p` is set to 1 then it is assumed that coupons are issued one period per year.



When the option `p` is set to 2, it is assumed that there are two periods per year. If the face value of the bond is paid with the last coupon, then the option `ind` will be set to 1 otherwise `ind` will be set to 0.

```
ytm(ind=0,p=1,cr=.6,v=100,n=2,x=.131)
curve(ytm(ind=0,p=1,cr=.6,n=2,v=100,x=x),0,1)
abline(h=100)
curve(ytm(ind=1,p=2,cr=.05,n=7,v=100,x=x),0,1)
abline(h=120)
```

The graph can be embellished with a title and a formula for describing the present value curve.

```
ytm(ind=0,p=1,cr=.6,v=100,n=2,x=.131)
curve(ytm(ind=0,p=1,cr=.6,n=2,v=100,x=x),0,1,
main="Yield to Maturity of a Bond \n Semi-annual Payments",
ylab="Market Price", xlab="Rate of Return")
abline(h=100)
text(.5,100,c("Market Price"), adj=c(0,0))
text(.6,85, expression(pv==frac(c[1],(1+frac(r,2))^frac(1,2))+frac(c[2],
(1+frac(r,2))^frac(2,2))+cdots+frac(c[n-1],(1+frac(r,2))^frac(n-1,2))+
frac(v+c[n],(1+frac(r,2))^frac(n,2)) ) )
```

To incorporate the mathematical formula of the present value curve for the yield to maturity problem, the syntax of the mathematical annotation in R was followed. A description of it is

found in the Appendix and can be found in R by entering `?plotmath`. Such symbols as `==` for the equals, `c[n]` for making subscripts, and `frac` for writing fractions are not obvious. A reference guide like the one in the Appendix is essential.

The graph of the yield to maturity curve is hyperbolic in shape, and where it intersects the market price of the bond is the value of r which solves the yield to maturity problem. A more precise estimate of `r` than the one which can be obtained by inspecting a graph can be determined by numerical techniques. That utility in R which will find the point of intersection of the yield to present value curve with a specified market price is the `uniroot` procedure. This procedure will find `r` such that $f(r) = 0$. Let us define a new function, `f`, so that it is essentially the same as `ytm` except the market price denoted by `a` is subtracted from `ytm`. In the yield to maturity problem, we want to find that `r` such that the present value, `pv=market price`. To that end, we create the new function, $f = ytm - marketprice$ and find that value of `r` which makes f(r)=0 by means of the `uniroot` procedure.

```
f<-function(ind,p,cr,v,n,a,x=x){
c<-cr*v/p
w0<-0
for (i in 1:(p*n-1)){
w<-w0+c/(1+x/p)^(i/p)
w0<-w
}
i<-i+1
w<-w0+(v*ind+c)/(1+x/p)^(i/p) -a
return(w)
}
```

We will save some writing later, if the solution is assigned to an object like `r.sol`:

```
>r.sol<-uniroot(f,lower=0,upper=1,tol=.00001,ind=1,p=2,cr=.05,v=100,n=30,a=108)
>r.sol
```

```
List of 4
 $ root      : num 0.0912
 $ f.root    : num 0.00189
 $ iter      : int 8
 $ estim.prec: num 5e-06
```

The required interest rate, `r`, for a market price of $108 with face value of $100, current interest rate of .05 where coupons are issued twice a year for 30 thirty years is 9.12%. The `uniroot` produced a list and, in order to extract the root of `f`, we use

```
>r.sol$root
```

```
[1] 0.09123141
```

Of course, it is always prudent to check the answer:

```
>ytm(ind=1,p=2,cr=.05,n=30,v=100,x=.09123141)
```

`[1] 108.0019` and the agreement with the market price of $108 confirms that `r.sol` is correct.

7.3 Application of Cubic Splines

Given only a few points, the problem is to estimate a yield to the maturity curve and then solve for r for a given market price. In Section 7.1 on page 57, the function `price` was defined. It produces the price path of a stock for `n` days given μ, σ, $S0$, $\triangle T$. For the sake of argument, let `sss` be the actual market price of a particular bond with $\mu = 15$, $\sigma = .15$, $\triangle T = \frac{1}{360}$, $S = 108$, and $n = 100$. Recall that `sss` is obtained by:

```
>sss<-price(mu=.15,sigma=.15,deltat=1/360,s0=120,n=100,graph=0)
>plot(sss[,1],sss[,3], main="Simulated Price of a Bond",
xlab="Days into the Future", ylab="Price")
```

Suppose that five points corresponding to five random times are generated by `sss` and that they constitute all the available information about the market price of a bond. We will pretend that these five points are real so that by means of the method of splines, we will demonstrate how to estimate the time at which the present value of the bond will equal the market price of $108 and compare our answer with the known one. That is, because prices at five random times which constitute this contrived set of data were produced by a known function, `sss`, in which the number of intervals to maturity was set to 30, we should expect our answer to be 30. The price of the bond at five random times will be stored in the object, `jj`.

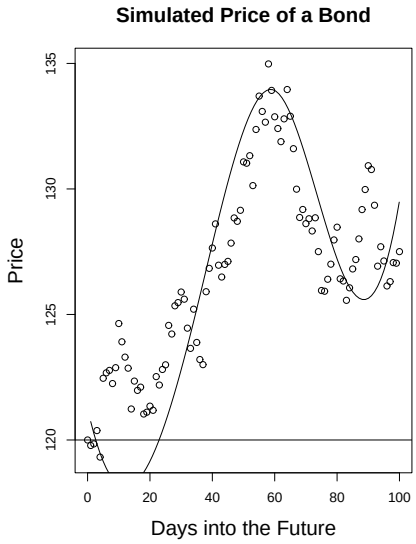
```
>jj<-sss[c(3,35,60,83,96),3]
>jj
```

`[1] 118.2990 116.1878 118.7017 124.8170 128.5055` Given these five simulated prices of a bond as if they were actual data, we want to estimate a maturity curve from them. The method of interpolation for accomplishing that goal will be the `splinefun` procedure.

```
>ff<-splinefun(c(3,35,60,83,96),jj)
```

The `splinefun` will interpolate by means of cubic splines the five points which we extracted from `sss`. Let us see how well the `splinefun` procedure did in estimating `sss`.

```
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
sss<-price(mu=.15,sigma=.15,deltat=1/360,s0=120,n=100,graph=0)
plot(sss[,1],sss[,3], main="Simulated Price of a Bond",
xlab="Days into the Future", ylab="Price")
jj<-sss[c(3,35,60,83,96),3]
ff<-splinefun(c(3,35,60,83,96),jj)
curve(ff(x),1,100,add=TRUE)
>abline(h=120)
```



Choosing a value of `r` within the range of 20 to 40 will provide a first approximation to that `r` for the market price of $120 as that is the interval in which the present value curve crosses the line: `y=120`. Recall that `uniroot` will find the root of a function, i,e. it will find that `r` such that f(r)=0. We need to redefine the function `ff` so that, instead of crossing the line at $120, it will cross the line h=0. To do that, we will subtract 120 from the original function.

```
>ff<-splinefun(c(3,35,60,83,96),jj-120)
>uniroot(ff,lower=20,upper=40,tol=.00001)$root
[1] 22.94355
```

The graph of `sss` confirms that uniroot produced the right number. Although the answer is less than the 30 which we had expected, such is the consequence of using a meager set of data.

7.4 Black and Scholes Option Pricing

The Black and Scholes option pricing formula is:

$$C = S(0)\Phi(\omega) - Ke^{-rt}\Phi(\omega - \sigma\sqrt{t})$$

where

$$\omega = \frac{rt + \sigma^2 t/2 - log(K/S(0))}{\sigma\sqrt{t}}$$

and $\Phi(x)$ is the cumulative distribution function for the Standard Normal distribution. Given the initial stock price, S0, the strike price, K, the time, t, in years to maturity, and the risk free interest rate, r, the problem is to find the implied volatility, σ. To that end, we need to define two

functions: the call price, C, and the price of a put, P. In order to draw a graph of C and P as a function of σ, we will give the unknown variable, σ, the name, x, in both functions. Therefore,

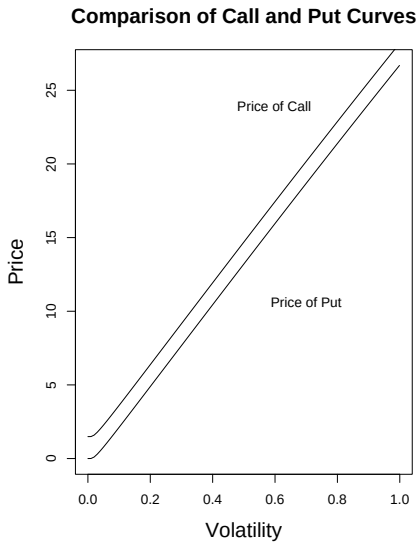
```
C<-function(x=x,s0=100,r=.03,t=.5,K=100){
omega<-(r*t+x^2*t/2-log(K/s0))/(x*sqrt(t))
c<-s0*pnorm(omega)-K*exp(-r*t)*pnorm(omega-x*sqrt(t))
return(c)
}

P<-function(x=x,s0=100,r=.03,t=.5,K=100){
omega<- (r*t+x^2*t/2-log(K/s0))/(x*sqrt(t))
p<-K*exp(-r*t)*pnorm(-(omega-x*sqrt(t)))-s0*pnorm(-omega)
return(p)
}
```

Both curves are superimposed in Figure 7.4.

```
>curve(P(x=x,s0=100,r=.03,t=.5,K=100),0,1)
>curve(C(x=x,s0=100,r=.03,t=.5,K=100),0,1,add=T)
>abline(h=11.1)
```

Because it is difficult to distinguish the two curves, we will label them.

```
>curve(P(x=x,s0=100,r=.03,t=.5,K=100,xlab="Volatility",ylab="Price"),0,1)
>A.text<-c("Price of Put")
>text(locator(n=1),A.text)
>curve(C(x=x,s0=100,r=.03,t=.5,K=100),0,1,add=T)
>B.text<-c("Price of Call")
>text(locator(n=1),B.text)
```



A specific price of a call given that the initial stock price=100, risk free interest rate=.03,

time to maturity=.5 years, selling price=100, volatility=.3806032.

```
>C(x=.3806032,s0=100,r=.03,t=.5,sp=100)
```

```
[1] 11.38542
```

Or for a volatility =.2, selling price=30, risk free interest rate=.08, time to maturity=.08 years, and the strike price=34 the call price is:

```
>C(x=.2,s0=30,r=.08,t=.25,sp=34)
```

```
[1] 0.238349
```

Conversely, to find the implied volatility given the strike price, selling price, risk free interest rate, time to maturity, we need to modify the function for finding the call price by subtracting the prescribed call price denoted by a.

```
CC<-function(x=x,s0,r,t,K,a){
omega<- (r*t+x2̂*t/2-log(K/s0))/(x*sqrt(t))
c<-s0*pnorm(omega)-K*exp(-r*t)*pnorm(omega-x*sqrt(t))-a
return(c)
}
```

Let us confirm the earlier computation of the call price by finding the volatility.

```
>uniroot(CC,lower=0,upper=1,tol=.00001,s0=100,r=.03,t=.5,K=100,a=11.38542)$root
```

```
[1] 0.3806032
```

is the volatility in the first example.

```
>uniroot(CC,lower=0,upper=1,tol=.00001,s0=30,r=.08,t=.25,K=34,a=.238349)$root
```

```
[1] 0.1999991
```

which is the value we used for the volatility in the second example.

Let us do the same thing for the price of a put. We will change the function of the price of a put by subtracting off the prescribed price of a put denoted by a.

```
PP<-function(x=x,s0,r,t,K,a){
omega<- (r*t+x2̂*t/2-log(K/s0))/(x*sqrt(t))
p<-K*exp(-r*t)*pnorm(-(omega-x*sqrt(t)))-s0*pnorm(-omega)-a
return(p)
}
```

```
>P(x=.2,s0=30,r=.08,t=.25,K=34)
```

```
[1] 3.565104
```

```
>uniroot(PP,lower=0,upper=1,tol=.00001,s0=30,r=.08,t=.25,K=34,a=3.565104)$root
```

```
[1] 0.1999991
```

which is the volatility which was used in the first example.

```
>P(x=.3806032,s0=100,r=.03,t=.5,K=100)
```

```
[1] 9.896616
```

```
>uniroot(PP,lower=0,upper=1,tol=.001,s0=100,r=.03,t=.5,K=100,a=9.896616)$root
```

```
[1] 0.3805973
```

which is the volatility which was used the second example of the call price.

Whether we use the function for the price of a call or the function for the price of a put, we obtain the same estimate for the implied volatility as the following illustrates:

Suppose that
```
>S<-30
```
then

```
>S+P(x=.2^2,s0=30,r=.08,t=.25,K=34)-
C(x=.2^2,s0=30,r=.08,t=.25,K=34)-34*exp(-.08*.25)
```
```
[1] 0
```

as it should because $P(s,t,K) - C(s,t,K) - Ke^{-rt} - S = 0$ according to the theory of no-arbitrage cost of a European option.

Chapter 8

Exercises

Exercise 1. *In this exercise, write the following program to a text file and call the file: newpasswd.fun. The suffix, fun, indicates that the file contains a function of* R . *It has no other meaning. In fact, a suffix is not required to be a part of the name of the file. The program will produce a password in which alternating letters of vowels and consonants are taken at random by means of the stored procedure,* `sample()` *with a random digit inserted the middle of the password.*

```
vow<-c("a","e","i","o","u","y")
con<-setdiff(letters,vow)
VOW<-c("A","E","I","O","U","Y")
CON<-setdiff(LETTERS,VOW)
vowels<-union(vow,VOW)
consonants<-union(con,CON)
passcon<-sample(consonants,4,replace=T)
passvow<-sample(vowels,4,replace=T)
num<-sample(0:9,1)
password<-paste(passcon[1],passvow[1],passcon[2],num,
passvow[2],passcon[3],passvow[3],passcon[4])
print(password)
```

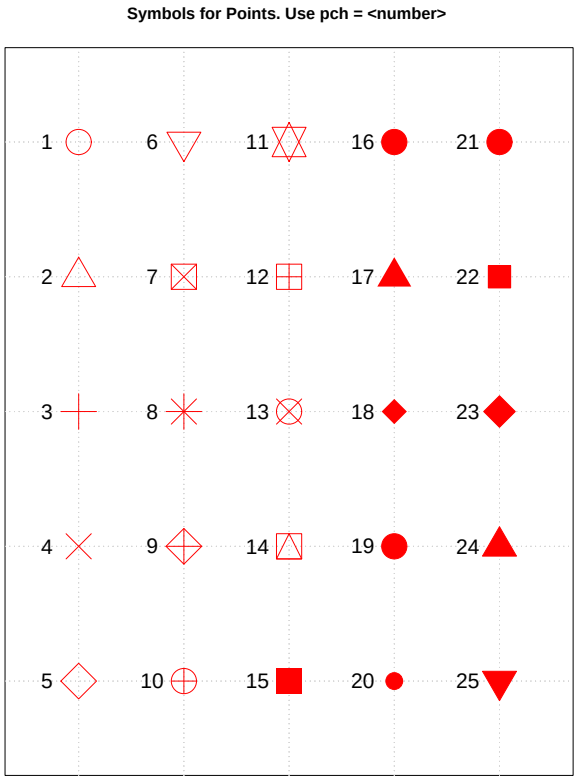
By alternating consonants and vowels, the password resembles a word so that it is easier to memorize, but it will remain very difficult to crack by being one out of 345,600,000 possibilities. It would be a nuisance to type this program into an active session of R every time a new password is sought. Because the program has been saved to a text file, it can be loaded into R by means of the `source()` command:

```
>source("newpasswd.fun")
```

Exercise 2. *This second exercise will produce the picture of all the different symbols which are available in* R *for use in making graphs. The semi-colon is used to separate distinct commands*

on the same line when it is deemed convenient to write the two commands on the same lines.

```
ipch <- 1:25; dd <- c(-1,1)/2
rx <- dd+ range(ix <- (ipch-1) %/% 5)
ry <- dd+ range(iy <- 3 + 4-(ipch-1) %% 5)
plot(rx, ry, type="n", axes = F, xlab = "", ylab = "",xaxt="n",yaxt="n",
 main = "Symbols for Points. Use pch = <number> ")
abline(v=ix, h=iy, col = "lightgray", lty = "dotted")
for(i in ipch) { # red symbols with a yellow interior (where available)
 points(ix[i], iy[i], pch=i, col="red", bg="red", cex = 4)
 text (ix[i] - .3, iy[i], i, col="black", cex = 1.5)
 }
```



Chapter 9

Appendix 1

`plotmath` *Mathematical Annotation in R*

Description

If the `text` argument to one of the text-drawing functions (`text`, `mtext`, `axis`) in R is an expression, the argument is interpreted as a mathematical expression and the output will be formatted according to TeX-like rules. Expressions can also be used for titles, subtitles and x- and y-axis labels (but not for axis labels on `persp` plots).

Details

A mathematical expression must obey the normal rules of syntax for any R expression, but it is interpreted according to very different rules than for normal R expressions.

It is possible to produce many different mathematical symbols, generate sub- or superscripts, produce fractions, etc.

The output from `demo(plotmath)` includes several tables which show the available features. In these tables, the columns of grey text show sample R expressions, and the columns of black text show the resulting output.

The available features are also described in the tables below:

Syntax	Meaning

`x + y`	x plus y
`x - y`	x minus y
`x*y`	juxtapose x and y
`x/y`	x forwardslash y
`x %+-% y`	x plus or minus y
`x %/% y`	x divided by y
`x %*% y`	x times y
`x[i]`	x subscript i
`x^2`	x superscript 2
`paste(x, y, z)`	juxtapose x, y, and z
`sqrt(x)`	square root of x
`sqrt(x, y)`	yth root of x
`x == y`	x equals y
`x != y`	x is not equal to y
`x < y`	x is less than y
`x <= y`	x is less than or equal to y
`x > y`	x is greater than y
`x >= y`	x is greater than or equal to y
`x %~~% y`	x is approximately equal to y
`x %=~% y`	x and y are congruent
`x %==% y`	x is defined as y
`x %prop% y`	x is proportional to y
`plain(x)`	draw x in normal font
`bold(x)`	draw x in bold font
`italic(x)`	draw x in italic font
`bolditalic(x)`	draw x in bolditalic font
`list(x, y, z)`	comma-separated list
`...`	ellipsis (height varies)
`cdots`	ellipsis (vertically centred)
`ldots`	ellipsis (at baseline)
`x %subset% y`	x is a proper subset of y
`x %subseteq% y`	x is a subset of y
`x %notsubset% y`	x is not a subset of y
`x %supset% y`	x is a proper superset of y
`x %supseteq% y`	x is a superset of y
`x %in% y`	x is an element of y
`x %notin% y`	x is not an element of y
`hat(x)`	x with a circumflex
`tilde(x)`	x with a tilde
`dot(x)`	x with a dot

`ring(x)`	x with a ring
`bar(xy)`	xy with bar
`widehat(xy)`	xy with a wide circumflex
`widetilde(xy)`	xy with a wide tilde
`x %<->% y`	x double-arrow y
`x %->% y`	x right-arrow y
`x %<-% y`	x left-arrow y
`x %up% y`	x up-arrow y
`x %down% y`	x down-arrow y
`x %<=>% y`	x is equivalent to y
`x %=>% y`	x implies y
`x %<=% y`	y implies x
`x %dblup% y`	x double-up-arrow y
`x %dbldown% y`	x double-down-arrow y
`alpha` – `omega`	Greek symbols
`Alpha` – `Omega`	uppercase Greek symbols
`infinity`	infinity symbol
`partialdiff`	partial differential symbol
`32*degree`	32 degrees
`60*minute`	60 minutes of angle
`30*second`	30 seconds of angle
`displaystyle(x)`	draw x in normal size (extra spacing)
`textstyle(x)`	draw x in normal size
`scriptstyle(x)`	draw x in small size
`scriptscriptstyle(x)`	draw x in very small size
`x ~~ y`	put extra space between x and y
`x + phantom(0) + y`	leave gap for "0", but don’t draw it
`x + over(1, phantom(0))`	leave vertical gap for "0" (don’t draw)
`frac(x, y)`	x over y
`over(x, y)`	x over y
`atop(x, y)`	x over y (no horizontal bar)
`sum(x[i], i==1, n)`	sum x[i] for i equals 1 to n
`prod(plain(P)(X==x), x)`	product of P(X=x) for all values of x
`integral(f(x)*dx, a, b)`	definite integral of f(x) wrt x
`union(A[i], i==1, n)`	union of A[i] for i equals 1 to n
`intersect(A[i], i==1, n)`	intersection of A[i]
`lim(f(x), x %->% 0)`	limit of f(x) as x tends to 0
`min(g(x), x > 0)`	minimum of g(x) for x greater than 0
`inf(S)`	infimum of S
`sup(S)`	supremum of S

`x^y + z`	normal operator precedence
`x^(y + z)`	visible grouping of operands
`x^{y + z}`	invisible grouping of operands
`group("(",list(a, b),"]")`	specify left and right delimiters
`bgroup("(",atop(x,y),")")`	use scalable delimiters
`group(lceil, x, rceil)`	special delimiters

References

Murrell, P. and Ihaka, R. (2000) An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics*, **9**, 582–599.

See Also

`demo(plotmath)`, `axis`, `mtext`, `text`, `title`

Examples

```
x <- seq(-4, 4, len = 101)
y <- cbind(sin(x), cos(x))
matplot(x, y, type = "l", xaxt = "n",
        main = expression(paste(plain(sin) * phi, "  and  ",
                                plain(cos) * phi)),
        ylab = expression("sin" * phi, "cos" * phi), # only 1st is taken
        xlab = expression(paste("Phase Angle ", phi)),
        col.main = "blue")
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
     lab = expression(-pi, -pi/2, 0, pi/2, pi))

## How to combine "math" and numeric variables :
plot(1:10, type="n", xlab="", ylab="", main = "plot math & numbers")
tt <- 1.23 ; mtext(substitute(hat(theta) == that, list(that= tt)))
for(i in 2:9)
    text(i,i+1, substitute(list(xi,eta) == group("(",list(x,y),")"),
                           list(x=i, y=i+1)))

plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
     cex = .8)
```

```
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n))",
     cex = .8)
text(8, 5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
                            plain(e)^{frac(-(x-mu)^2, 2*sigma^2)})),
     cex = 1.2)
```

Bibliography

John M. Chambers. *Programming with Data A Guide to the S Language*. Springer-Verlag, New York, New York, 1998.

Peter Dalgaard. *Introductory Statistics with R*. Springer-Verlag, New York, New York, 2002.

Darrel Duffie. *Dynamic Asset Pricing Theory*. Princeton University Press, New Jersey, 1996.

Sheldon Ross. *An Introduction to Mathematical Finance*. Cambridge University Press, Cambridge, U.K., 1999.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-PLUS*. Springer-Verlag, New York, New York, 1999.