R scripting for time series*

Attila Máté
Brooklyn College of the City University of New York

August 3, 2017

Contents

1 Introduction to scripting

1.1 A short script calculating factorials

A short R program calculating factorials is given in [2, p. 3] of the textbook. We slightly modified this program, and presenting it as a script. That is, rather then typing individual commands on the command line, we put the commands into a file and run them together. The script is as follows:

```
1 #!/usr/bin/env Rscript
2 Fact <- function(n) if (n == 1) 1 else n * Fact(n - 1)
3 Fact(5)
4 factorial <- rep(0, 11)
5 for (i in 1:11) { factorial[i] <- Fact(i) }
6 factorial
```

*Written for the course Mathematics 4506 (Time Series) at Brooklyn College of CUNY.

These lines need to be put in a file. Note that the numbers on the left, called *line numbers*, are not part of the file, so they do not need to be typed. Your can choose have your text editor display or not dispay line numbers. Here, it is useful to show the line numbers so we can discuss what each line is about. In Linux, files are created by a text editor, which is different from a word processor.[1.1] In Linux, the basic text editor is vi or vim[1.2] There are many other text editors in Linux, but what is special about `vi` is that it is present even in very limited Unix/Linux distributions. If `vi` is not present, then one of its less friendly predecessors, ed or ex will be there. A basic introduction to Unix, the `vi` text editor, and basic file manipulation commands is given in [3, pp. 6–19].

The file name of the above listing can be of your choosing; for example, you can give the name `p3_factorial.r`; this indicates that it corresponds to the program is given in [2, p. 3] of the textbook, is an R script. In order to make the script executable, type

```
$ chmod u+x p3_factorial.r
```

The dollar symbol `$` on he left is the command line prompt,and it is not to be typed.[1.3] After this, you can run the script simply by running its name

```
$ p3_factorial.r
```

perhaps preceded by the string `./`:

```
$ ./p3_factorial.r
```

if the former version does not work.[1.4] We will talk about the results of running this program, but first we will give a line-by-line explanation of what it does. The first line tells that the file is an R script. The rest of the lines are much the same as the lines described in the textbook [2, pp. 7–9], except those lines are typed on the command line, and not put in a script. First, one brings up the R program by typing

```
$ R
```

to enter the R commandline environment. The prompt will change from `$` to `>`; then one may enter R commands. To exit the R environment, one needs to type

```
> quit()
```

The computer will reply with

```
> Save workspace image? [y/n/c]:
```

[1.1] In Microsoft Windows, Microsoft Word is a word processor, and not a text editor. Many windows users do not note the difference, and if you send them a text file as an attachment, they do often not know how to read it. You can use Microsoft Word, but you should not use Word to write text files, since it may add invisible control characters, which is a no-no in a text file.

[1.2] The latter stands for `vi` improved; `vi` and `vim` work much the same way on a basic level, but the latter has many additional useful features.

[1.3] Your actual prompt may be different; Unix and Linux is highly customizable, and you can define your own prompt. More precisely, this prompt is called the *shell prompt*, since, as we mentioned above, when you typing on the command line, you are interpreting with the shell.

[1.4] The string `.` indicates your present working directory; so prefixing the string `./` to the file name indicates that the file is in your present working directory. The first version only works if the your present working directory is in the execution search path, which lists the directories for the name of the command to be executed. Until some time ago, the user's working directory was usually in the execution search path, but it turned out that this can cause a security problems

In the brackets, the possible replies are listed; the reply `y` with save the workspace image (so next time you can come back and continue your work), the reply `n` will discard it, and the reply `c` will continue in R and gives back the R prompt `>`. Incidentally, if you by mistake you type

```
> quit:
```

the computer will give the confusing reply

```
> +
```

At this point, you will want to get back to the R prompt `>` in order continue to interact with R to correct your mistake. You can do this by typing control-`c` (i.e., by holding down the control key, and then pressing `c`).[1.5]

We will next discuss the commands in the script given on p. 1. Line 1 indicates that the script is an R program. Line 2 defines the factorial function, using the equations $n! = n \cdot (n-1)!$ if $n > 1$ is an integer and $1! = 1$. This is a recursive definition; R is a programming language that understands recursive definitions. the symbol `<-` is the *assignment symbol*: it means that the variable on the left gets (is assigned) the value described on the right. The *reserved word* `function` is used for defineing new functions. The asterisk `*` in the line denotes multiplication; it is common in computer languages to use asterisk as the multiplication symbol. Line 3 produces on output. The output of the above script is

```
[1] 120
 [1]         1         2         6        24       120       720      5040     40320
 [9]    362880   3628800  39916800
```

(The script is slightly indented for highlighting; the begining of the line is flush with the left bracket `[` in the first line.) In line 3 of the script, the value of the just defined function `Fact` is printed out for the argument 5 as line 1 of the output. R usually prints lists; the string `[1]` on the left indicates that the line starts with the first element of the list printed (which, in fact, is its only element). Line 4 creates a list called `factorial`; the function `rep` is used to replicate 0 eleven times. In line 5, the ith element of the list is assigned the value $i!$ for i with $1 \leq i \leq 11$. The symbol `1:11` denotes the list `1, 2,..., 11` of eleven numbers. In line 11 the list `factorial` is printed. The list is too long to fit on a single line, so it is broken up. Line 2 of the output gives the factorials of the first eight numbers; the symbol `[1]` at the beginning of the line indicates that the output starts with the first element of the list. Line 3 gives the factorials of the next three numbers; the symbol `[9]` at the beginning of the line indicates that the output starts with the nineth element of the list. R often operates with lists; this has an advantage when dealing with statistics; nevertheless, it is better to describe R as a general purpose programming language especially suitable for statistics, rather than a statistical programming language.

1.2 Maine unemployment

The next script unemployment in Maine, described in the program in [2, pp. 7–9] of the textbook:

```
1  #!/usr/bin/env Rscript
2  www <-
3    "http://www.maths.adelaide.edu.au/andrew.metcalfe/Data/Maine.dat"
4  Maine.month <- read.table(www, header = TRUE)
```

[1.5] See [3, pp. 80, 125] for the use of control-`c` in Unix/Linux. See the index on [3, p. 133] for the use of the various control keys in Unix/Linux.

```
 5
 6  attach(Maine.month)
 7  class(Maine.month)
 8
 9  Maine.month.ts <- ts(unemploy, start = c(1996, 1), freq = 12)
10  Maine.annual.ts <- aggregate(Maine.month.ts)/12
11
12  layout(1:2)
13  plot(Maine.month.ts, ylab = "unemployed (%)")
14  plot(Maine.annual.ts, ylab = "unemployed (%)")
15
16  Maine.Feb <- window(Maine.month.ts, start = c(1996,2), freq = TRUE)
17  Maine.Aug <- window(Maine.month.ts, start = c(1996,8), freq = TRUE)
18  Feb.ratio <- mean(Maine.Feb) / mean(Maine.month.ts)
19  Aug.ratio <- mean(Maine.Aug) / mean(Maine.month.ts)
20
21  Feb.ratio
22  Aug.ratio
```

Again, the numbers at the beginning of the line are line numbers, and are not part of the file. The file name of the above listing can be of your choosing; for example, you can give the name `p7_unemp_me_s.r`; this indicates that it corresponds to the program starting on [2, p. 7] of the textbook; it concerns unemployment in Maine (abbreviated as ME); `s` indicates that it is a short version of the script (there will be a longer version that is almost the same). As above, before running the script, you need to make it executable.

We will discuss the commands in this script line by line. Line 1 indicates that script is an R program. Lines 2–4 download the data from the Internet; the symbol `<-` is the *assignment symbol*: it means that the variable on the left gets (is assigned) the value described on the right.The downloading happens in two separate steps; lines 2 and 3 of the script assings the name of the location of the data (the website in this case) to the variable `www`, and line 4 reads the data at this location. Note that the website changed since the textbook [2] was published; the new website is here. The data is given at the site is the file `Maine.dat`, which is just a text file; it contains 129 lines, starting as

```
unemploy
6.700000000e+000
6.700000000e+000
6.400000000e+000
.................
```

The remaining lines in the file each contain a number. Lines 2 and 3 may more naturally have been written on a single line, but we wanted to avoid lines that are too long.[1.6] Line 4 of the script reads the data at the given location into what is called a *data frame*; this will assign an individual name to each item of data that has been read. The argument `header = TRUE` in this line indicates that the main part of the name of the variables so created will be the first line of the file, `unemploy` in the present case, as seen from the listing of the first few lines of the file `Maine.dat` above. A more detailed description of what a data frame is is of no importance at the moment, but below we will list resources where one can read about the R programming language to find out the details. Lines 5, 8, 11, 15, and 20 are blank; this has no particular significance. It perhaps makes the script easier

[1.6] The long name of the website is mainly responsible for the lenght of the line. The name website cannot be directly be broken up, since the *new line* character breaking up the line cannot be inserted in website; yet there is a way to break it up, as will be explained in the improved version of the script below.

to read by breaking it up into segments, but it has no influence on what the script does. Line 6 will be explained later, on account of line 9. Line 7 will produce an output; it fact running the script produces the following output

```
[1] "data.frame"
[1] 1.222529
[1] 0.8163732
```

This output is not especially enlightening, because it is not clear which line in the program produced which line of the output; later we will modify the script to correct this problem. In any case, line 1 of the output is produced is produced by line 7 of the script, indicating that the class of the *object* `Maine.month` is data frame. R is an object oriented language; groups of data are objects. Individual data in the object are accessible with *helper functions*. This changes an earlier paradigm in which groups of data were organized in *data structures*; the programmer knew how data structures were organized and he or she had direct access to the data. Nowadays, this is a security issue, and users of the program are not supposed to have direct access to the data; data can only be accessed if the helper functions allow one to do this. Object are organized into *classes*, and each class has its own helper functions. In fact there are functions that are common to several classes, but they behave differently depending on the class of an object; think of PostScript and PDF files. They both represent documents that can be printed, but the printing program has to behave differently whether it is printing PostScript or PDF.

Line 9 will produce a time series from the unemployment data given in the data frame `Maine.month`. Without the command `attach` on line 6, this the line will produce an error, Deleting line 6 (or replacing it with a blank line so as not to change line numbers), line 9 will produce an error message, since the name `unemploy` is not understood; the output will be as follows:

```
[1] "data.frame"
Error in is.data.frame(data) : object 'unemploy' not found
Calls: ts -> is.data.frame
Execution halted
```

This error can be corrected if the name of the data `unemploy`, which is a part of the data fram `Maine.month` is referred to by its fullname `Maine.month$unemploy`. The command

```
attach(Maine.month)
```

allows one to use the shortened name; the name `unemploy` comes from the first line of the file `Maine.dat` (on p. 4 we listed the first few lines of this file, and then, in describing the content of line 4 of the script we mentioned how this name gets associated with the data. When running R on the command line, one can ask R to expain the meaning of a command. For example, by typing

```
> ?attach
```

(recall that the symbol `>` is the prompt, and not to be typed), one gets a screenful of explanation, some of which you might want to read, but a lot of it is too complicated to understand at the beginning. If the screen ends with the single colon `:` on the line, it means there is more. To continue reading, you can press *Enter* to get one new line, or the space bar to get another screenful; to quit reading, press `q`; you will get back the prompt `>`. You can also use the Unix navigation keys `k` to go up or `j` to go down in the output; the advantage of this is that you can re-read the earlier part

again.[1.7] When you reach the end of the output, you will find a line saying `(END)`; then you can type `q` to get back the prompt.

To explain more of line 9, `freq` means that there are 12 seasons of one main unit of time (obviously meaning at present that the main unit is year, and the seasons are month). The string `c(1996, 1)` combines the two numbers into a vector or a list, and it means that the unemplyment data given on page 4 (and put into a data frame) is to be interpreted as a time series starting in January 1996. Line 10 aggregates the data, and gives the yearly unemployment data. Line 12 gives the layout of the plot (figure) given in lines 13 and 14: it says that one graph goes on a line, and the next graph goes underneath. In lines 13 and 14, `ylab` describes the label to be put on the y-axis. The figure printed by these lines is put in a file Rplots.pdf, which can be displayed by using a PDF viewer; the content of this file is shown on p. 6 as Figure 1.1.[1.8]

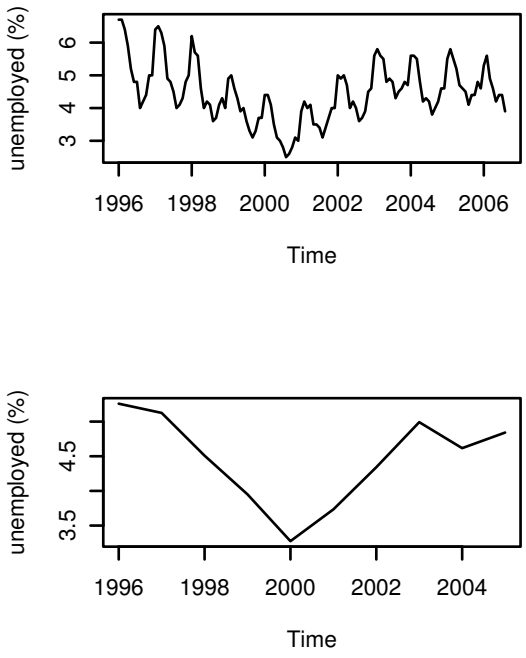

Figure 1.1: Maine uneployment

In lines 16 and 17, the function `window` extracts the data for February and August, respectively. In lines 18 and 17, the ratios of the means (averages) for these months are taken to the mean of the whole time series, and in lines 21 and 22, the commands result in printing out these these ratios, giving lines 2 and 3 of the output on page 5.

Later, if you want to learn more about R, there is plenty of material online. You can start

[1.7] The up or down arrow keys also work. The advantage of the Unix navigation keys (left: `h`, down: `j`, up `k`, right: `l` for touch typists is that one does not have to move one's hand away from the main part of the keyboard. The up and down keys will probably also work in the Microsoft Windows version of R.

[1.8] Not quite. The figure shown in p. 6 was produced by the improved version of the script given below.

with the Wikipedia description of the R programming language on R. Alternatively, one can go to the official website of R. An especially useful document to start learning to program in R is R for Beginners. One can also start with the document Introduction to R; this gives some more details. For a more precise description of R one can read the R Language Definition. For a very short yet useful introduction to R can be found here. Another document that also explains scripting for Unix or Windows is Kickstarting R. The R programming language runs on several computer at Brooklyn College in the Labrary and the Library Cafe (on Windows computers). The list of computers running R or other software can be found at the website listing Brooklyn College Computers running public software. If there is an interest, R can probably be installed also on some computers running Unix or Linux in the I.T.S. Public Computing Labs at Brooklyn College.

We mentioned that the problem with this output on page 5 is that it is hard to tell where each line of the output comes from. To make the output easier to read, we will give a more refined version of the script:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
3  www <- paste("/home/mate/courses/timeser/r_f17.dir/",
4               "website.dir/data.dir/Maine.dat",
5                sep="")
6  # www <- paste("http://www.maths.adelaide.edu.au/",
7  #             "andrew.metcalfe/Data/Maine.dat",
8  #             sep="")
9  Maine.month <- read.table(www, header = TRUE)
10
11 attach(Maine.month)
12 cat("class(Maine.month) = ")
13 cat(class(Maine.month))
14 cat("\n")
15
16 Maine.month.ts <- ts(unemploy, start = c(1996, 1), freq = 12)
17 Maine.annual.ts <- aggregate(Maine.month.ts)/12
18
19 CairoPS(width=3, height=4, pointsize=8, file = "maine")
20 # CairoPDF(width=2, height=3, pointsize=8, file = "maine")
21 layout(1:2)
22 plot(Maine.month.ts, ylab = "unemployed (%)")
23 plot(Maine.annual.ts, ylab = "unemployed (%)")
24
25 Maine.Feb <- window(Maine.month.ts, start = c(1996,2), freq = TRUE)
26 Maine.Aug <- window(Maine.month.ts, start = c(1996,8), freq = TRUE)
27 Feb.ratio <- mean(Maine.Feb) / mean(Maine.month.ts)
28 Aug.ratio <- mean(Maine.Aug) / mean(Maine.month.ts)
29
30 cat("Feb.ratio = ")
31 cat(Feb.ratio)
32 cat("\n")
33 cat("Aug.ratio = ")
34 cat(Aug.ratio)
35 cat("\n")
```

We will go through this script line by line to explain how this works; we will only discuss lines where we made changes. Line 2 loads the library Cairo for improved graphics printing. The Cairo

package is probably not present on your system, but it is easy to install it with the command when running R:[1.9]

```
> install.packages('Cairo')
```

Lines 3–5 assigns the location of the data (in this case, the name of the data file `Maine.dat` along with the full path to the location of the file) to the variable `www`. We first downloaded the file `Maine.dat` to our computer as the file

```
/home/mate/courses/timeser/r_f17.dir/website.dir/data.dir/Maine.dat
```

and we loaded the data into the script from this location rather than the website.[1.10] To avoid excessively long lines in the script, this location was broken up into two parts, the first part given in line 3 as the first argument of the function `paste`), and the second part is in line 4, as the second argument of `paste`. The third argument, `sep=""`, indicates that the separator is the empty string; The function `paste` joins the first and the second arguments together while putting the separator in between; of course, the separator being the empty string, nothing is put in between. So the result of the function `paste` is the location of the data. Lines 6–8 are commented out, so they do not have any effect. The meaning of the symbol octothorpe `#` in a line indicates that the rest of the line should be ignored by the script.[1.11] Had they not been commented out, they would have downloaded the data from the website as in the earlier version of the script. The only difference here is that the lines have been broken up with the use of `paste`, as in lines 3–5. Line 9 reads the data from the given location. Lines 12–14 produce an output.

The output of the whole script is as follows:

```
class(Maine.month) = data.frame
Feb.ratio = 1.222529
Aug.ratio = 0.8163732
```

This is much clearer than the output on p. 5 of the earlier version of the script. Lines 12–14 produce the first line of the output, using the the no-frills output function `cat`. The first line of the output is produced by partly line 12, which prints out the quoted text, without adding a newline character, so the printout of produced line 13 is added on the same line of the output. Line 13 prints the class of the variable `Maine.month`, again without producing a new line character. The new line character `\n` breaking the first line of the output is produced by line 14. Line 19 describes the format of the plot printed in lines 21–23. It indicates that the plot is printed in a PostScript file called `maine.ps`, and it gives the size of the output gives the size of the text in the figure in points.[1.12] The commented out line 20 could have been used instead; this line indicates how to print the plot into a PDF file called `maine.pdf`. Lines 30–32 produce the second line of the output, line 30 printing the string

[1.9] When you install packages in R, you need to first consider whether you are the only user of the system or whether there are also other users. In case there are other users, the packages need to be installed by a superuser (a user with administrative privileges), because the package needs to be installed at a location that only the superuser has access to. In Linux, the packages installed by R will probably be located in the directory

`/usr/local/lib/R/site-library`

If you want to install it at some other location, you need to read the online help for the command `install.packages`.

[1.10] Since the original location of the data has disappeared since the publication, this is a natural precaution. Note, however that the data is also available at the website of the publisher of the textbook [2]

[1.11] See the Wikipedia article for other names of the symbol `#`.

[1.12] Point is a typographic measure: 72 points make an inch. This document is printed with letters of size 10 points, with some letters in the document being smaller.

"`Feb.ratio =`",[1.13] line 31 prints the value `1.222529` of the February ratio, and line 32 prints the line feed character `\n` marking the end of the line. Line 33–35 produce the third line of the output in a similar way.

2 Decomposition of time series

It is often useful to decompose a time series X_t into the long term progression, called *trend* T_t, of the time series, the *seasonal component* S_t, and an *irregular* (random) component I_t. Such a decomposition can be additive:

$$X_t = T_t + S_t + I_t,$$

or multiplicative:

$$X_t = T_t \cdot S_t \cdot I_t.$$

The following script will do this for the Australian electricity supply data in GWh (gigawatt hours, i.e., 10^9 watt hours) from January 1958 to December 1990, described in [2, p. 10] of the textbook; the script is a modification of the script on [2, p. 23]:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
3  www <- paste("http://www.maths.adelaide.edu.au/",
4             "andrew.metcalfe/Data/cbe.dat",
5             sep="")
6  CBE <- read.table(www, header=T)
7
8  cat("CBE[1:4, ] :\n")
9  CBE[1:4, ]
10 cat("\n")
11 cat("CBE[3, ] :\n")
12 CBE[3, ]
13 cat("\n")
14 cat('CBE[3,2] :\n')
15 CBE[3,2]
16 cat("\n")
17 cat('CBE[3,"beer"] :\n')
18 CBE[3,"beer"]
19
20 Elec.ts <- ts(CBE[, 3], start = 1958, freq = 12)
21 CairoPS(width=3, height=5, pointsize=8, file = "elect_add")
22 plot(decompose(Elec.ts))
23
24 Elec.decom <- decompose(Elec.ts, type = "mult")
25 CairoPS(width=3, height=5, pointsize=8, file = "elect_mult")
26 plot(Elec.decom)
27
28 Trend <- Elec.decom$trend
29 Seasonal <- Elec.decom$seasonal
30 CairoPS(width=3, height=2.5, pointsize=8, file =
31               "elect_mult_recomb")
```

[1.13] American manuals of style would place the comma before the closing quotation marks, but this would be confusing, so we used the *logical placement* of quotation marks. We will do so also below when clarity requires this.

```
32  ts.plot(cbind(Trend, Trend * Seasonal), lty = 1:2)
33
34  cat("\nclass(Elec.decom) = ")
35  cat(class(Elec.decom))
```

As before, the lines on the left are line numbers, and are not part of the script. We will explain the new features in this script. The the file `cbe.dat` that is being read in lines 3–6 of the script has 397 lines, each except the first one containing three numbers. It contains the monthly chocolate-based production (tonnes, i.e., metric tons), the beer production (millions of liters), and electricity supply from January 1958 to December 1990. The file starts out as follows:

```
choc    beer    elec
1451    96.3    1497
2037    84.4    1463
2477    91.2    1648
2785    81.9    1595
.....................
```

Line 6 reads this file into a data frame `CBE` that arrange this data into a 397×3 matrix where the columns of the data will have the names `choc`, `beer`, and `elec`; the columns can be referred by their names or by their column numbers. In lines 8–19, various parts of this data frame are printed out, resulting in the following output of the script:

```
1  CBE[1:4, ] :
2    choc beer elec
3  1 1451 96.3 1497
4  2 2037 84.4 1463
5  3 2477 91.2 1648
6  4 2785 81.9 1595
7
8  CBE[3, ] :
9    choc beer elec
10 3 2477 91.2 1648
11
12 CBE[3,2] :
13 [1] 91.2
14
15 CBE[3,"beer"] :
16 [1] 91.2
17
18 class(Elec.decom) = decomposed.ts
```

The numbers on the left are line numbers to make our discussion simpler, and they are not part of the file.

Line 8 of the script prints out the quoted text, producing line 1 of the output. In the quoted text, `\n` is the new line character, ending the line of the output, so what produced by line 9 of the script is printed in a new line. The symbol `1:4` is an abbreviation for the list `1, 2, 3, 4`. The function `cat` does not by itself produces a new line character. The data frame `CBE` is essentiallu a matrix, and line 9 of the script produces lines 2–6 of the output, in line 2 giving the column names (the column names come from the first line of the file `cbe.dat` given above), and the numbers `1`, `2`, `3`, and `4`, at the beginning of lines 3–6 are the row numbers of the matrix `CBE`. Line 10 adds a new line character

to the output; this appears as the blank line 7 of the output.[2.1] Line 11 of the script prints out the quoted text, including a new line character, resulting in line 9 of the output. Line 12 prints out the 3rd row of the matrix `CBE`, resulting in lines 9–10 of the output: line 9 lists the column names, and line 10 lists the content of row 3 preceded by the row number 3. Line 13 of the script produces the blank line 12 of the output. Line 14 of the script prints out the quoted text, resulting in line 12 output. Line 15 of the script prints out the entry in row 3 and column 2 of the matrix `CBE` in line 13 on of the output. Line 16 of the script produces the blank line 14 of the output. Line 17 of the script prints out the test between the single quotes `'`, resulting in line 15 on the output. We needed to use single quotes to enclose the quoted text to make sure that the double quotes `"` are part of the quoted text. Line 18 of the scropt again script prints out the entry in row 3 and column 2 of the matrix `CBE`, but the column 2 is indicated by its name `beer` (in quotes) rather than by its column number. This results in line 16 of the output, which is, of course, identical to line 13 of the output. Line 20 creates the time series `Elect.ts` of electricity production from the third column of the dataframe `Elect`. The argument `decompose(Elec.ts)` of `plot` in line 22 is the additive decomposition of the time series `Elec.ts`; lines 21 and 22 plot this decomposition in the PostScript file `elect_add.ps`, the content of which is displayed in Figure 2.1 in p. 12. Line 24 creates the multiplicative decomposition (indicated by the argument `type = "mult"`) of the time series `Elec.ts`, and lines 25 and 26 plot this in the PostScript file `elect_mult.ps`. The content of this file is displayed in Figure 2.2 on p. 13.[2.2] Lines 28 and 29 put the multiplicative trend and seasonal component of the time series `Elec.ts` is separate files, and Figure 2.3 in p. 14 displays the trend and the product of the trend and seasonal components (but omitting the random component) in a single figure. Note that the dollar symbol `$` on lines 28 and 29 is an R operator used to extract values stored in a data frame. This figure is printed by the commands in lines 30–32 of the script. In line 32, `lty` refers to the line types in the figure. The string `1:2` denotes the list `1, 2`, meaning that the trend uses line type 1 (solid line), and the component trend times seasonal uses line type 2 (dashed line). More about line types in R can be found at the website An Easy Guide to Line Types in R. Finally, on lines 34–35 of the script, we print out the class of the object `Elec.decom`; this is shown on line 18 of the output.

It is clear from Figures 2.1 and 2.2 that the multiplicative decomposition, and not the additive decomposition is the appropriate one. This can be seen from the graph of the observed time series (which is the same in both figures) showing as the time series increases, the seasonal fluctuations also increase. The additive decomposition would be appropriate if the seasonal fluctuations remained about the same. Another clue that the addtive decomposition is not the right one is that the random fluctuations in Figure 2.1 are much smaller in the middle rather than toward the ends.

3 Holt–Winters forecasting

Exponential smoothing and Holt–Winters forecasting is discussed in the textbook [2, Section 3.4 starting on p. 55] and also in [4, p. 41 in Section 12]. The program applying Holt–Winters forecasting to international passenger bookings in thousands on Pan American airline in the United States is discussed in [2, pp. 62–63]. This program is implemented in the script `p62_ap_holt_winters.r`:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
```

[2.1] Line 9 of the script produces a new line character at the end of each of lines 3–6; that is, new new line character ending line 6 of the output is followed by another new line character produced by line 10 of the script. Two new line characters next to each other appear as a blank line in the output.

[2.2] The figure headings "Decomposition of additive time series" and "Decomposition of multiplicative time series" in Figures 2.1 and 2.2 are generated by R. Perhaps the headings "Additive decomposition of time series" and "Multiplicative decomposition of time series" would be more appropriate.

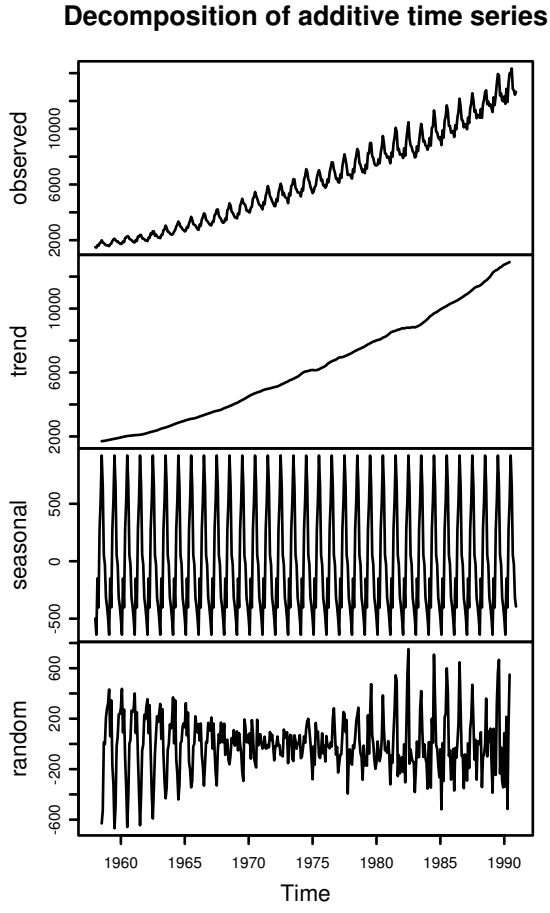

Figure 2.1: Electricity additive decomposition

```
 3  data(AirPassengers)
 4  AP <- AirPassengers
 5  cat("AP :\n")
 6  AP
 7  cat("\nclass(AP) = "); cat(class(AP))
 8  AP.hw <- HoltWinters(AP, seasonal = "mult")
 9  cat("\nclass(AP.hw) = "); cat(class(AP.hw))
10  AP.predict <- predict(AP.hw, n.ahead = 4 * 12)
11  cat("\nclass(AP.predict) = "); cat(class(AP.predict)); cat("\n")
12  CairoPS(width=3, height=2.5, pointsize=8, file =
13                  "AP_HW_pred")
14  ts.plot(AP, AP.predict, lty = 1:2)
```

A line-by-line explanation of the new features of this program follows. The data file `AirPassengers` contains the airline bookings data; this file is part of the R distribution, and is loaded by the command on line 3 of the script; line 4 assings these data to the object `AP`; the only purpose of this

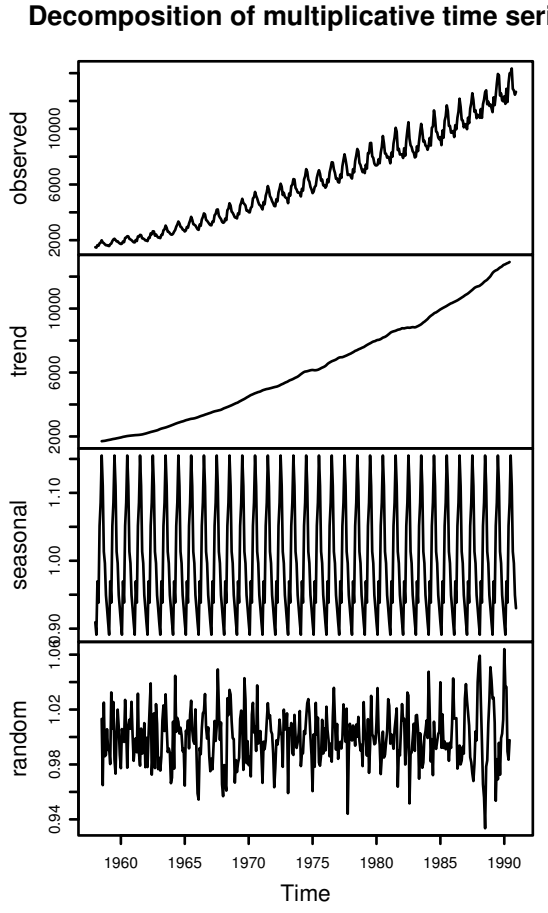

Figure 2.2: Electricity multiplicative decomposition

command is to use a shorter name; we could use the name `AirPassengers` in the script instead. Line 5 produces an output; the output of the whole script is as follows:

```
1  AP :
2        Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
3  1949 112 118 132 129 121 135 148 148 136 119 104 118
4  1950 115 126 141 135 125 149 170 170 158 133 114 140
5  1951 145 150 178 163 172 178 199 199 184 162 146 166
6  1952 171 180 193 181 183 218 230 242 209 191 172 194
7  1953 196 196 236 235 229 243 264 272 237 211 180 201
8  1954 204 188 235 227 234 264 302 293 259 229 203 229
9  1955 242 233 267 269 270 315 364 347 312 274 237 278
10 1956 284 277 317 313 318 374 413 405 355 306 271 306
11 1957 315 301 356 348 355 422 465 467 404 347 305 336
12 1958 340 318 362 348 363 435 491 505 404 359 310 337
13 1959 360 342 406 396 420 472 548 559 463 407 362 405
```

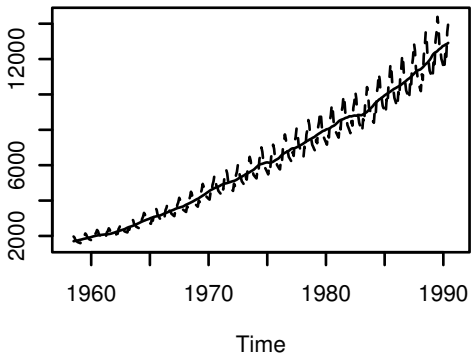

Figure 2.3: Electricity with trendline

```
14  1960 417 391 419 461 472 535 622 606 508 461 390 432
15
16  class(AP) = ts
17  class(AP.hw) = HoltWinters
18  class(AP.predict) = ts
```

Line 5 of the script prints line 1 of the output, and line 6 of the script prints out the content of the whole file `AP` in lines 2–14 of the output. Line 7 of the scripts puts two R commands on the same line; this is allowed, but the commands must be separated by semicolons (`;`). The first command prints out the quoted text; the first character of the quoted text is the new line character `\n`, resulting in the blank line 5 of the output. It is important to note that there should be no space after the new line character `\n` unless you want this space character to appear as the first character of the next line in the output. The rest of the quoted text appears on line 16 of the output. The second command on line 7 prints the class of the object `AP`, which is `ts` (time series). Line 8 apples Holt–Winters smoothing to the time series `AP`, creating the smoothed model for the time series. As the command on this line indicates, multiplicative seasonal commands are used; the reason for this is that the seasonal fluctuations increase with time, as shown in Figure 3.1 (the solid line in the figure indicates the air passenger data; the broken line the predictions, as will be discussed below). This issue whether seasonal effects are additive or multiplicative was discussed above on p. 11. Line 9 of the script prints out the class of the object `AP.hw` on line 17 of the output, indicating that the class is `HoltWinters`. Line 10 creates a four year prediction based of the air passenger data; that is the prediction is for $4 \cdot 12 = 48$ time steps (i.e., months) ahead. The command `predict` is a generic R function that behaves differently depending on the class of its argument; this is important, since different prediction methods are needed depending on the way the time series was modeled (so far, we discussed only Holt–Winter smoothing; later, we will discuss linear models and ARIMA models). This shows the importance of the class `HoltWinters`. Line 11 of the script prints out the class of the object `AP.predict` on line 18 of the output indicating that this class is `ts`, i.e., time series. The first new line character `\n` on line 11 of the scripts creates the new line character at the end of line 17 of the output, ensuring that the items printed out on line 11 of the script start on a new line of the output. The second and last new line character `\n` on line 11 prints a new line character at the

end of line 18 of the output file, so that the last character of the output is the new line character `\n`.[3.1] Finally, lines 12–14 of the script prints the graph in Figure 3.1 the air passenger data and the four year prediction. The air passenger data is represented by a solid line, the prediction is, by a broken line; this is specifyed by the argument `lty = 1:2` on line 14 of the script (line types and the meaning of the argument `lty` was discussed above on p. 11).

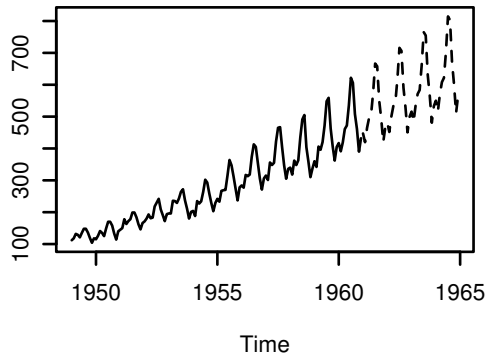

Figure 3.1: Air Passengers: Holt–Winters prediction

4 Modeling time series

A linear model for global temperatures is discussed in [2, pp. 95-96] of the textbook. Here we discuss linear model with explanatory variables t (time), t^2, and t^3 for this time series. One could say that the trend of this time series is approximated by a cubic polynomial of time; it will turn out, however, that a cubic approximation to the trend is not really better than a linear approximation. This will be discussed below. The script producing the cubic approximation to the trend is as follows:

```
 1  #!/usr/bin/env Rscript
 2  library('Cairo')
 3  www <- paste("http://www.maths.adelaide.edu.au/",
 4             "andrew.metcalfe/Data/global.dat",
 5             sep="")
 6  Global <- scan(www)
 7  cat("Global[1:5] :\n"); Global[1:5];
 8
 9  Global.ts <- ts(Global, s = c(1856, 1), end = c(2005, 12),
10      fr = 12)
11  temp <- window(Global.ts, start = 1970)
12  cat("\ntime(temp)[1:5] : (time vector)\n"); time(temp)[1:5];
13
```

[3.1] This is important, since unless the output of the program redirected to a file, the output will appear on the command line, and the next command you will want to type will appear in the middle of the line rather than on a new line.

```
14 TIME <- (time(temp)-mean(time(temp)))/sd(time(temp))
15 cat("\nTIME[1:5] : (TIME vector starts) :\n"); TIME[1:5]
16 cat("TIME[(length(TIME)-5):length(TIME)] ")
17 cat("time vector ends) :\n")
18 TIME[(length(TIME)-5):length(TIME)]
19
20 temp.lm <- lm(temp ~ TIME+I(TIME^2)+I(TIME^3))
21 cat("\ncoef(temp.lm) :\n"); coef(temp.lm)
22 cat("\nconfint(temp.lm) :\n"); confint(temp.lm)
23 CairoPS(width=2.5, height=2.5, pointsize=8,
24          file = "temp_lm_resid_acf")
25 acf(resid(lm(temp.lm)))
26
27 CairoPS(width=3, height=2.5, pointsize=8, file =
28              "temp")
29 plot(temp)
30
31 temp_lm <-
32      temp.lm$coef[4]*TIME^3+temp.lm$coef[3]*TIME^2+
33      temp.lm$coef[2]*TIME+temp.lm$coef[1]
34 CairoPS(width=3, height=2.5, pointsize=8, file =
35               "temp_lm")
36 plot(temp_lm)
37
38 cat("\nclass(temp.lm) = "); cat(class(temp.lm))
39 cat("\nclass(temp) = "); cat(class(temp))
40 cat("\nclass(temp_lm) = "); cat(class(temp_lm))
41 cat("\nclass(TIME^2) = "); cat(class(TIME^2))
42 cat("\nclass(I(TIME^2)) = "); cat(class(I(TIME^2)))
43 cat("\nAIC(temp.lm) = "); cat(AIC(temp.lm)); cat("\n")
```

The line numbers are of course not part of the file. We will give a line by line description of the new features in this script. Line 3–6 download the data from the website. The first few lines of the data given in the file `global.dat` describing the global monthly temperatures for the years 1856–2005 is as follows:

```
-0.384 -0.457 -0.673 -0.344 -0.311 -0.071 -0.246 -0.235 -0.380 -0.418 -0.670 -0.386
-0.437 -0.150 -0.528 -0.692 -0.629 -0.363 -0.375 -0.328 -0.495 -0.646 -0.754 -0.137
-0.452 -1.031 -0.643 -0.328 -0.311 -0.263 -0.248 -0.274 -0.203 -0.121 -0.913 -0.197
.....................................................................................
```

There are altogether 150 lines of data. We cannot use `read.table` in line 6 as in earlier scripts, since that would interpret the data as a matrix, whereas it is just a simple list of temperatures. Therefore we use `scan`, which creates a list (in the sense of R) of the data. On lines 7 we printed out the first five element of the list `Global`; these are of course the same as the first five numbers in the file `global.dat` given above. Note that we put two R commands in line 7, since those commands really belong together. As we mentioned before, one can put more than one R commands on a line, but, if so, these commands must be separated by a semicolon (;). Line 7 of the scripts produces lines 1 and 2 of the output. The whole output of the above script is as follows:

```
1 Global[1:5] :
2 [1] -0.384 -0.457 -0.673 -0.344 -0.311
3
4 time(temp)[1:5] : (time vector)
```

```
 5  [1] 1970.000 1970.083 1970.167 1970.250 1970.333
 6
 7  TIME[1:5] : (TIME vector starts) :
 8  [1] -1.726045 -1.718035 -1.710026 -1.702016 -1.694007
 9  TIME[(length(TIME)-5):length(TIME)] time vector ends) :
10  [1] 1.685997 1.694007 1.702016 1.710026 1.718035 1.726045
11
12  coef(temp.lm) :
13   (Intercept)          TIME    I(TIME^2)    I(TIME^3)
14   0.166317861  0.196728675  0.008713884 -0.007267253
15
16  confint(temp.lm) :
17                       2.5 %      97.5 %
18  (Intercept)  0.148374607 0.184261115
19  TIME         0.166788151 0.226669199
20  I(TIME^2)   -0.004691306 0.022119074
21  I(TIME^3)   -0.022547756 0.008013251
22
23  class(temp.lm) = lm
24  class(temp) = ts
25  class(temp_lm) = ts
26  class(TIME^2) = ts
27  class(I(TIME^2)) = AsIs ts
28  AIC(temp.lm) = -554.4239
```

Lines 9–10 uses the data to make the time series `Global.ts. fr = 12` on this line indicates that there are 12 data per period (i.e., 12 data constitue a whole year, `fr` being short for freqency). Line 11 takes a window of this time series; that is, it creates the time series starting in 1970 from the same data. The object `time(temp)` is the list of times used in this time series; the first five members of this list are printed out in line 12 of the script as lines 4–5 out the output. Line 14 creates the object `TIME` by a linear transformation of the time `time(temp)` of the temperature time series under consideration. What is accomplished by this transformation is that the values of `TIME` are small numbers, so its square and cube are not going to be of excessively large (and so, not much different in size that `TIME` itself); this will improve the accuracy of numerical calculations. As an illustration, lines 15 and 16 print out the first five and the last five values of `TIME` in lines 7–10 of the output; these are to be compared with the values of `time(temp)` in line 5.

Line 20 creates the model `temp.lm` for the time series `temp` of tempetarues. The function `lm` on the right-hand side of the assignment symbol `<-` stands for linear model. The formula `lm(x~y)` fits a regression model `y` on the object `x`; in the present case, `x` is the time series `temp`, and the coefficients in the formula

$$c_3T^3 + c_2T^2 + c_1T + c_0 \tag{4.1}$$

are sought with T standing for the time series object `TIME` that best approximates the time series `temp`; the meaning of "best" can be specified in the formula – without specifying this, least squares approximation is used. The fact that an approximation of the form given in formula (4.1) is desired is indicated by the model formula in line 20 of the script. There is some special language to writing model formulas; as you can see the constant term and the coefficients are not indicated. See pp. 56–58 of the document R for beginners for an introduction how to write model formulas. The reason for writing `I(TIME^2)` instead of `TIME^2` on line 20 of the script is to prevent data type conversion

that often happens in arithmetical conversion.[4.1] For some clarification, at the end of the script, in lines 38–42, we printed out the classes of some objects; these appear in the output on on lines 23–27. These show that the class of temp.lm is `lm` (linear model, and, for example, the class of `TIME^2` is `ts` (time series), whereas the class of `I(TIME^2)` is `AsIs ts`; *as is* classes cannot be converted; so the class of `I(TIME^2)` always stays time series; otherwise it is no different from `TIME^2`. Line 21 of the script prints out the coefficients of the model `temp.lm` created on line 20 as lines 12-14 of the output; the coefficient named `Intercept` corresponds to c_0 in formula (4.1). The 95% confidence intervals are printed out on line 22. Lines 23–25 of the script print the graph of the autocorrelations of the *residuals* of the model in Figure 4.1; the residual time series is the time series formed by the differences between the values of the modeled time series and the values given by the model. The autocorrelations of the residuals between the two horizontal broken lines are statistically indistiguishable from zero at the 95% confidence level. For the model to be perfect, all autocorrelations with lag > 1 should be zero; the 0-lag autocorrelation is of course 1 (since it is the correlation of a variable with itself).

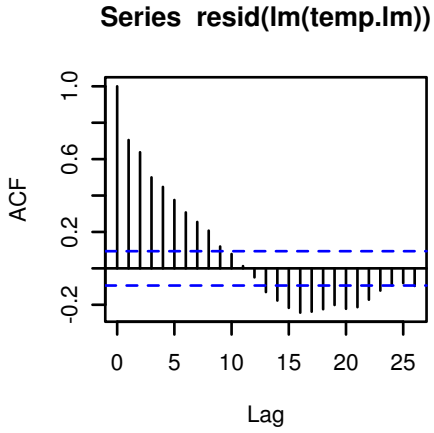

Figure 4.1: The autocorrelation functions of the residuals of the cubic trend model

Lines 27–29 print the graph of the time series `temp` of global temperatures between 1970 and 2005. Lines 31–33 create the time series `temp_lm` described by the model `temp.lm` by substituting the coefficients calculated in the model into formula (4.1).[4.2] The coeffcient c_{k-1} is named `temp.lm$coef[k]` in the model for k with $1 \leq k \leq 4$. This time series is pictured in Figure 4.3, as described in lines 34–36 of the script. Line 38–42 of the script print the classes of various object occurring in the script producing lines 23–27 of the output; this was commented on above. Line 43 of the script prints the *Akaike information criterion* (AIC) of the model `temp.lm`; this is a measure of the quality of the model. The formula for AIC is

$$\mathrm{AIC} = 2k - 2\log(\hat{L}),$$

where k is the number of parameters in the model, log is the natural logarithm. [4.3] and $\hat{L}$ is the

[4.1] The reason for the conversion usually is that without conversion the operation could not be performed. In model formulas, however, there is no operation to be performed, the formulas only indicate the type of model sought.

[4.2] There is a simpler way to do this where we do not have to repeat the model formula, as we will see in the next script.

[4.3] In mathematical writing, the symbol for natural logarithm is traditionally log and not ln.

maximum value of the likelihood function for the model.

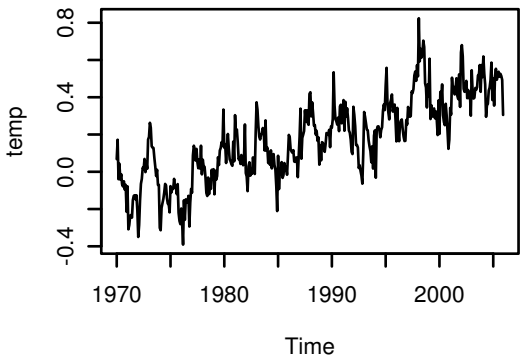

Figure 4.2: Temperatures from 1970 to 2005

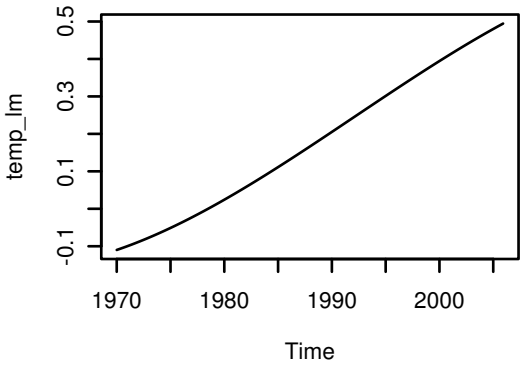

Figure 4.3: Cubic trend of temperatures from 1970 to 2005

The likelyhood function needs to assume some theoretical distribution is of the system modeled, and this theoretical distribution depends on the parameter values in the model (in the present case, the parameters are the coefficients in the model formula). The likelihood function is the same as the joint density function of the values to be measured, except that when considering this function as a density function, the parameter values are given, and when considering it as a likelihood values, the measured values of the random variables are given and the variables are the model parameters. A least square approximation usually assume a multivariate normal distribution for the random variables.

The theory of the Akaike information criterion is too deep to discuss it here; suffice it to say that a model with the least possible value of the Akaike information criterion is preferred. The issue is

overparametrization of the model: picking too many parameters may provide a better fit to a time series, but it may not improve the predictive value of the model, since the model parameters may only reflect accidental variations in the run time series under consideration, and a longer run would change these parameter values.

From the confidence intervals for the coeffcient printed out in lines 16–21 it appears that 0 is in the confidence intervals for the coefficients of `TIME^2` and `TIME^3`; so perhaps these terms are not needed in the model. The next script builds a model without these terms:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
3  www <- paste("http://www.maths.adelaide.edu.au/",
4             "andrew.metcalfe/Data/global.dat",
5             sep="")
6  Global <- scan(www)
7  Global.ts <- ts(Global, s = c(1856, 1), end = c(2005, 12),
8      fr = 12)
9  temp <- window(Global.ts, start = 1970)
10 TIME <- (time(temp)-mean(time(temp)))/sd(time(temp))
11
12 temp.lm1 <- lm(temp ~ TIME)
13 cat("coef(temp.lm1) :\n"); coef(temp.lm1)
14 cat("\nconfint(temp.lm1) :\n"); confint(temp.lm1)
15
16 CairoPS(width=2.5, height=2.5, pointsize=8,
17          file = "temp_lm1_resid_acf")
18 acf(resid(temp.lm1))
19
20 temp_lm1 <-
21      ts(fitted(temp.lm1))
22 CairoPS(width=3, height=2.5, pointsize=8, file =
23               "temp_lm1")
24 plot(temp_lm1)
25
26 resid_lm1 <- ts(resid(temp.lm1))
27 CairoPS(width=3, height=2.5, pointsize=8, file =
28               "resid_lm1")
29 plot(resid_lm1)
30
31 cat("\nAIC(temp.lm1) = "); cat(AIC(temp.lm1)); cat("\n")
```

There is very little that nees to be explained about this script, because most of the commands were already explained in connection with the earlier script. On line 12 the linear model `temp.lm1` is build; the model formula in this line is simply `TIME`; this means that the coefficients in the formula

$$c_1 T + c_0 \tag{4.2}$$

are sought with T standing for `TIME` that best approximate the time series in question. In lines 20–21, the time series of the fitted model is created: we used the `ts` and `fitted` functions instead of rewriting the model formula, as we did in lines 31-33 of the preceding script. The function `fitted` is a generic function that extracts fitted values from a model; the function `ts` uses these values to create a time series. On line 26 we create the time series of the residuals of the model. The output of this script is as follows:

```
1  coef(temp.lm1) :
2  (Intercept)         TIME
3    0.1750116    0.1836780
4
5  confint(temp.lm1) :
6                    2.5 %     97.5 %
7  (Intercept) 0.1630426 0.1869806
8  TIME        0.1716951 0.1956609
9
10 AIC(temp.lm1) = -555.9016
```

In lines 1–3 of the output the coefficients are printed out, with `Intercept` standing for the constant term c_0. The confidence intervals for these coefficients are printed in lines 5–8 of the output. The value of the Akaike information criterion printed on line 10 of the output is slightly smaller that the value printed for the cubic model earlier. The autocorrelation function of the residuals of the model is printed in Figure 4.4. The trend, i.e., time series of the fitted model, is printed in Figure 4.5. Finally, the time series of the residuals is printed in Figure 4.6

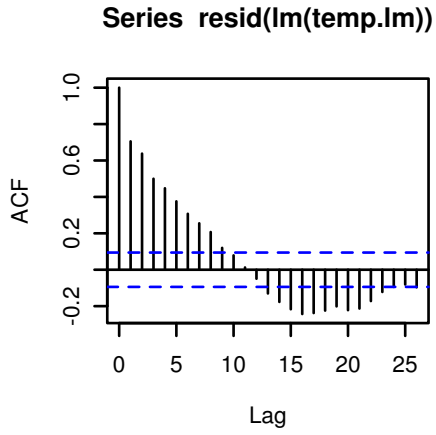

Figure 4.4: The autocorrelation functions of the residuals of the linear trend model

5 Simulating an autoregressive series

Autoregressivei, or AR, processes are described in [4, p. 27 in Section 8]. Based on the discussions starting on [2, p. 82, Subsection 4.6.1] and the program given there, we present a script that simulates an AR(1) process, and then builds an AR model for it:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
3  set.seed(1)
4  x <- w <- rnorm(100)
5  for (t in 2:100) x[t] <- 0.7 * x[t-1] + w[t]
6  cat("class(x) = "); cat(class(x))
```

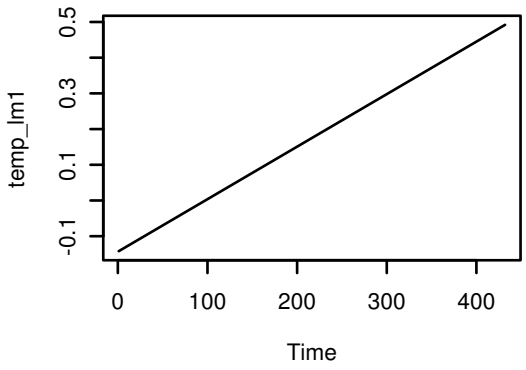

Figure 4.5: Linear trend of temperatures from 1970 to 2005

```
 7  CairoPS(width=3, height=2.5, pointsize=8, file =
 8                  "ar.simul")
 9  plot(x, type = "l")
10  CairoPS(width=3, height=2.5, pointsize=8, file =
11                  "ar.simul.acf")
12  acf(x)
13  CairoPS(width=3, height=2.5, pointsize=8, file =
14                  "ar.simul.pacf")
15  pacf(x)
16
17  x.ar <- ar(x, method = "mle")
18  cat("\nclass(x.ar) = "); cat(class(x.ar))
19  cat("\nx.ar$order = "); cat(x.ar$order)
20  cat("\n\nAR coefficient list: \n")
21  cat(x.ar$ar)
22
23  confint <- x.ar$ar + c(-2, 2) * sqrt(x.ar$asy.var.coef)
24  cat("\n\nx.ar 95% confidence interval:\n")
25  confint_frame <- data.frame(rbind(confint), row.names = "")
26  colnames(confint_frame) <- c("2.5%","97.5%")
27  confint_frame
```

We will give the new features in this script. The simulation uses *pseudo-random numbers*. There are many mathematical algorithms that rely on random input; however, it is inadvisable to use really random input in a computation for several reasons. One reason is that you may want to repeat the calculation with the same data, perhaps with some improvements; this could not be done if the input used were really random. There are a number of other reasons; the main one is, perhaps, that it is difficult to use truly random processes in a computer calculation. Customary computer hardware does not provide for this, and it would be difficult to ensure that the processes used to generate random imputs truly behave according to the high requirements of the mathematical calculation. That is, the distribution of the random numbers generated may be different from what is required.

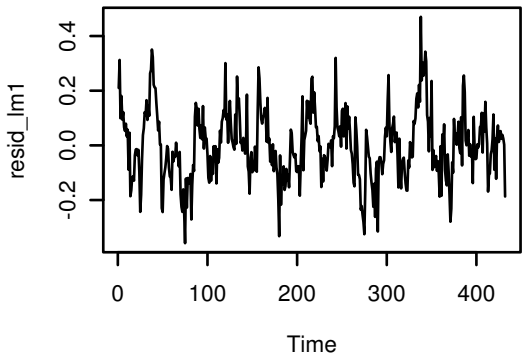

Figure 4.6: Residuals of the linear trend of temperatures from 1970 to 2005

For this reason, computers use numbers that are not really random, but, from a mathematical point of view they behave as if they were random; the numbers are called pseudo-random numbers, though one often just says *random numbers*, since the practice of using pseudo-random numbers instead of truly random ones is nearly universal, so no misunderstanding would result.

The random numbers that are generated depend on a *seed*; if you want to repeat the calculation, you need to set this seed before generating random numbers; if you do not set the seed, the computer uses some unpredictable internal parameters to set the seed.[5.1] Line 3 of the script sets the seed as 1. On line 4 the function `rnorm` creates 100 normally distributed random numbers, and these are then assigned to the lists `w` and `x`; the two assignment symbols `<-` indicate that the same values are assigned to both variables. Since on line 5 all values of the list `x` other than `x[1]` will be discarded, no purpose is served by assigning values to members of the list `x` other than `x[1]`, but the command on line 4 is a concise way of saying that `x` should be a list of numbers of length 100.[5.2] Line 5 applies the formula

$$x_t = 0.7x_{t-1} + w_t$$

for $t = 2, 3, \ldots, 100$ in turn, creating the AR(1) time series `x`. Line 6 prints out the class of the object `x`, resulting in the first line of the output:

```
1  class(x) = numeric
2  class(x.ar) = ar
3  x.ar$order = 1
4
5  AR coefficient list:
6  0.6009459
7
```

[5.1] In some cases it is important to keep the seed secret. An example where the seed of the first Linux ransomware program was ineffective, since the seed generating the random encryption key was recoverable from unencrypted files. However, after several failures, then linux ransomware creators have unfortunately perfected their product, so Linux is not immune to ransomware.

[5.2] It may be inefficient programming to assign values that are not used, but it simplifies writing the program: human time is much more expensive than computer time. Besides, some optimizing compilers may eliminate such programming inefficiencies without human intervention.

```
 8  x.ar 95% confidence interval:
 9         2.5%      97.5%
10   0.4404031 0.7614886
```

Thus, the class of `x` is `numeric`, i.e., a list of numbers, and not time series, as one might have expected. If it were important to make sure that the class of `x` is time series, one could insert the lines

```
library(methods)
x <- as(x, "ts")
```

after line 5 of the script to convert the class of `x` to time series, but this is of no importance for the present script. The first line here loads the library `methods`, which contains the function `as` used on the second line.

Lines 7–9 of the script prints out a diagram of the simulated "times series" `x` (quotes because the class of `x` is not `ts`); this appears in Figure 5.1. The argument `type = "l"` on line 9 indicates the type of the plot, `l` standing for lines connecting the data points. Lines 10–12 prints graph of the autocorrelation function of `x`; this appears as Figure 5.2, and lines 13–15 prints the *partial autocorrelation function* of `x`; what this is is briefly described in [2, Subsection 4.5.6, p. 81]; a more detailed desription is given in Wikipedia. On line 17 of the script, the autoregressive model `ar.x` is built using the method `mle`, which stands for *maximum likelihood estimate*. The Akaike information criterion (see p. 18 in Section 4) is used to determine the order of the model. On line 18, the class of the model `ar.x` is printed out, resultint in line 2 of the output. This shows that the class of the model is `ar`, which stands for autoregressive; this is important for example for the `predict` command, described in Section 3 on p. 14, since different classes of models are used for prediction in different ways. On line 19 of the script, the order of the autoregressive model `x.ar` is printed out, resulting in line 3 of the script, showing that this order is 1; this is of course what was expected since we are modeling a simulated AR(1) process. In lines 20–21 of the script, the coefficient list of the model `x.ar` is printed out. There is naturally only one coefficient on the list, since the model has order 1; the variable `x.ar$ar` mentioned on line 21 of the script contains the list of coefficient values. The coefficient 0.6009459 of the model is reasonably close to the coeffient 0.7 of the simulated process.

On line 23, the 95% confidence interval for the AR coefficient is determined. In this line, the variable `x.ar$asy.var.coef` contains the estimate the variance of the coefficient.[5.3] Writing ϕ_1 for the estimated value of this coefficient and σ_1 for its standard deviation (the square root of its variance), the values

$$\phi_1 - 2\sigma_1 \quad \text{and} \quad \phi_1 + 2\sigma_1$$

are calculated in this line; note that here `c(-2, 2)` is a list with the members -2 and 2, while everything else in this line is just a single number (or a list of lenght one). In this case, R replicates the lists of one member, making them into lists of two identical members, and then does the arithmetic operations on each member of the lists. The result is calculating the values of the two expressions

[5.3] In the name, `asy` stands for "asymptotic," meaning that the coefficient is estimated from the aymptotic theory for the covariance matrix of autoregressive models. The R online help says *variance matrix*, which is a less frequently used name for the covariance matrix. If there are more than one coefficients, then instead of the individual variances of the coefficients, the whole covariance matrix of the coefficient list is given. The program given in the textbook, uses the shorter name `x.ar$asy.var` for this variable; the even shorter name `x.ar$asy` can also be used. This is because the code completion feature is usually enabled and works without human intervention in R, so if only an initial part of the name of a variable is given, R will complete the name if this can be done in a unique way.

displayed above. The results of these two expressions are the endpoints of the confidence interval.[5.4]

The rest of the script in lines 24–27 is concerned with printing On line 24 of the script, lines 7-8 of the output are created; the two starting new line characters `\n` at the beginning of the quoted text of line 24 of the script produce the new line at the end of line 6 of the output, and it is the only character "printed" on line 7 of the output. The content of line 8 of the output and the line break at its end is created by the rest of the quoted text. In line 25, the confidence interval is put into a data frame, essentially a 1×2 matrix; the command `rbind(confint)` creates the single row of this matrix, and assings the empty string (a string of characters with a zero number of characters in it) as the name of this row. Line 26 assigns column names to this matrix. Finally, line 27 prints out this data frame, called `confint_frame`, while also printing the row and column names, resulting in lines 9–10 of the output; of course, there is nothing to be printe as row name, since the row name is the empty string.

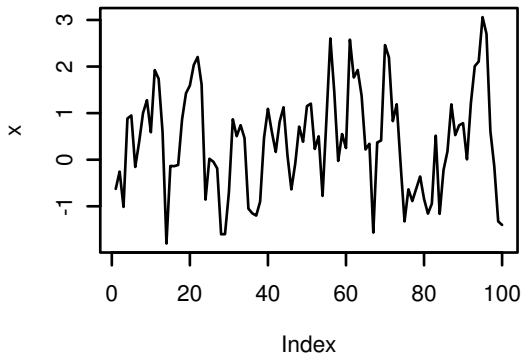

Figure 5.1: A simulated AR(1) process

6 Seasonal ARIMA models

Based on the discussion on [2, pp. 144–145] of the textbook, the following script builds a multiplicative seasonal ARIMA(0,1,1)(2,0,2) model for the time series of Australian electricity supply from January 1958 to December 1990 discussed on p. 9, and use the model for prediction for the twelve months of 1991. The script is a simplification of the program found in [2, pp. 144–145] in that the textbook describes a program that selects the best model among various possibilities; here we just describe how to construct the best model found by that program:

```
1  #!/usr/bin/env Rscript
2  library('Cairo')
3  www <- paste("http://www.maths.adelaide.edu.au/",
4               "andrew.metcalfe/Data/cbe.dat",
```

[5.4] For the sake of simplicity, the interval $(-2, 2)$ is used to give the 95% confidence interval for the standard normal distribution. More precisely, this interval would be $(-1.95996, 1.95996)$. The interval $(-2, 2)$ gives a 95.45% confidence level.

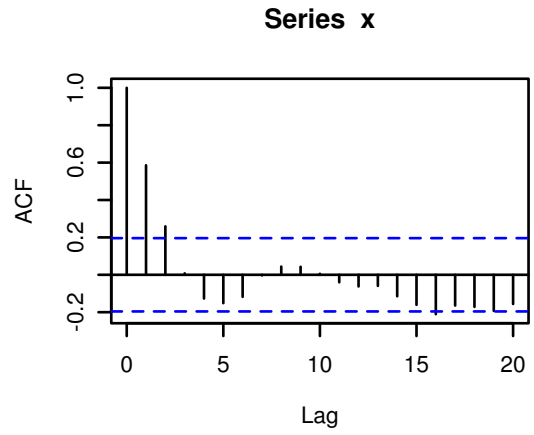

Figure 5.2: Autocorrelation of the simulated AR(1) process

```
5             sep="")
6  CBE <- read.table(www, header=T)
7
8  Elec.ts <- ts(CBE[, 3], start = 1958, freq = 12)
9  logelec.ts <- log(Elec.ts)
10 arima.elec <- arima(logelec.ts, order = c(0,1,1),
11                  seas = list(order = c(2, 0, 2),
12                  frequency(logelec.ts)), method = "CSS")
13
14 cat("coef(arima.elec) :\n")
15 coef(arima.elec)
16 cat("\nconfint(arima.elec) :\n")
17 confint(arima.elec)
18
19 CairoPS(width=3, height=2.5, pointsize=8, file =
20               "elect_arima_pred")
21 ts.plot(cbind(window(Elec.ts, start = 1981),
22               exp(predict(arima.elec,12)$pred)), lty = 1:2)
```

What follows is a line-by-line explanation of what is new in this program. The first eight lines of the script are based on the first 20 lines of the script on 9. On line 9, we take the logarithm of the time series.[6.1] The reason for this is explained by the top graph displayed in Figure 2.2, which shows that the electricity production shows an exponential increase, so its logarithm can be modeled more accurately. In lines 10–12, an ARIMA$(0,1,1)(2,0,2)$ model is built,[6.2].

We will describe a multiplicative ARIMA$(p,d,q)(P,D,Q)$ model with seasonal frequency s; that is we have s equally timed observations per period. The form suggested for such a model in [1, Subsection 9.1,3, formula (9.1.7) on p. 332] is

$$\phi(B)\Phi(B^s)(I-B)^d(I-B^s)^D X_t = \theta(B)\Theta(B^s)e_t,$$

[6.1] That is, we take the time series formed by the logarithms of the values.
[6.2] Some authors use the notation ARIMA$(0,1,1)i \times (2,0,2)$, and call the model multiplicative seasonal ARIMA

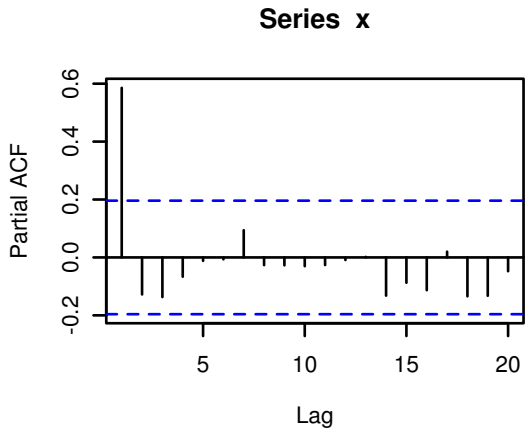

Figure 5.3: Partial autocorrelation of the simulated AR(1) process

where B is the backshift operator, the polynomial $\phi(x)$ has degree p, the polynomial $\theta(x)$ has degree q, the polynomial $\Phi(x)$ has degree P, and the polynomial $\Theta(x)$ has degree Q; these polynomials are assumed to have all their zeros outside the unit disk; the constant term in all these polynomials is assumed to be 1. That is, an ARIMA$(0,1,1)(2,0,2)$ model with $s = 12$ has form

$$(I - \Phi_1 B^{12} - \Phi_2 B^{24})(I - B)X_t = (I - \theta_1 B)(I - \Theta_1 B^{12} - \Theta_2 B^{24}))e_t \tag{6.1}$$

As seen in line 12 of the script, the method used in building the model is `CSS`, which stands for conditional sum of squares. This method minimizes the sum of squares of the residuals, or *errors* e_t, of the model. The method is *conditional* on the assumption that the first few errors, which are unobservable, are taken to be zero. These errors are unobservable since, for example, equation (6.1) cannot be used to calculate e_t for $t < t_{0+24}$ where t_0 is the starting time of the time series.

The coefficents for this model are printed out on lines 14–15 os the script, and appear as lines 1–3 of the output:

```
1  coef(arima.elec) :
2         ma1        sar1        sar2        sma1        sma2
3  -0.6565796   0.7314782   0.2556715  -0.3323874  -0.3051055
4
5  confint(arima.elec) :
6               2.5 %       97.5 %
7  ma1   -0.738815479  -0.57434365
8  sar1   0.482524762   0.98043168
9  sar2   0.008973946   0.50236897
10 sma1  -0.583101252  -0.08167347
11 sma2  -0.492888892  -0.11732220
```

In the output `ma1` $= \theta_1$, `sar1` $= \Phi_1$, `sar2` $= \Phi_2$, `sma1` $= \Theta_1$, and `sma2` $= \Theta_2$. The confidence intervals for these coefficients are printed out on lines 16–17 of the script; these appear in lines 5–11 of the output. Lines 19–21 of the script prints the graph of the the electricity supply time series starting in January 1981 and ending in December 1990; this appears in a solid line. In broken line, the graph of predictions for the twelve months of 1991 are printed. The predictions simply assume

that the new (yet unobserved) errors are 0, and uses the formula (6.1) to calculate the new values of X_t. In line 22 of the script, `lty` stands for line types. This has been explained on p. 11 on account of line 32 of the script starting on p. 9. The graph is shown in Figure 6.1.

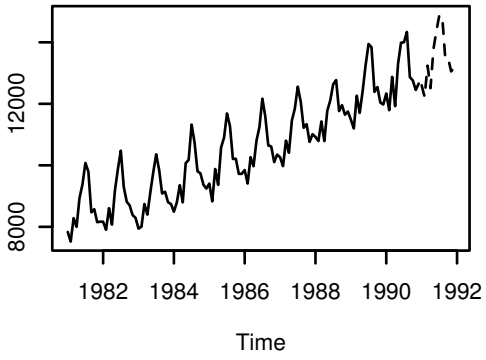

Figure 6.1: Australian electicity supply and prediction

References

[1] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis. Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ 07632, third edition, 1994.

[2] Paul S. P. Cowpertwait and Andrew V. Metcalfe. *Introductory Time Series with R*. Springer, New York, 2009.

[3] Attila Máté. Introduction to open computing. `http://www.sci.brooklyn.cuny.edu/ mate/open_computing/open.pdf`, 2003.

[4] Attila Máté. Aspects of time series, December 2016. `http://www.sci.brooklyn.cuny.edu/~mate/misc/time_series.pdf`.