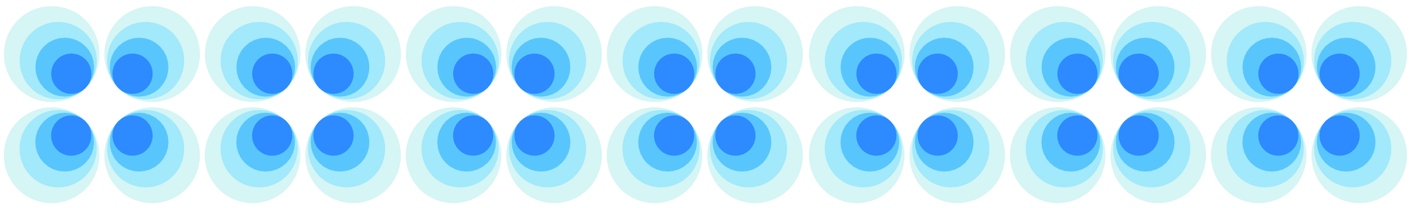
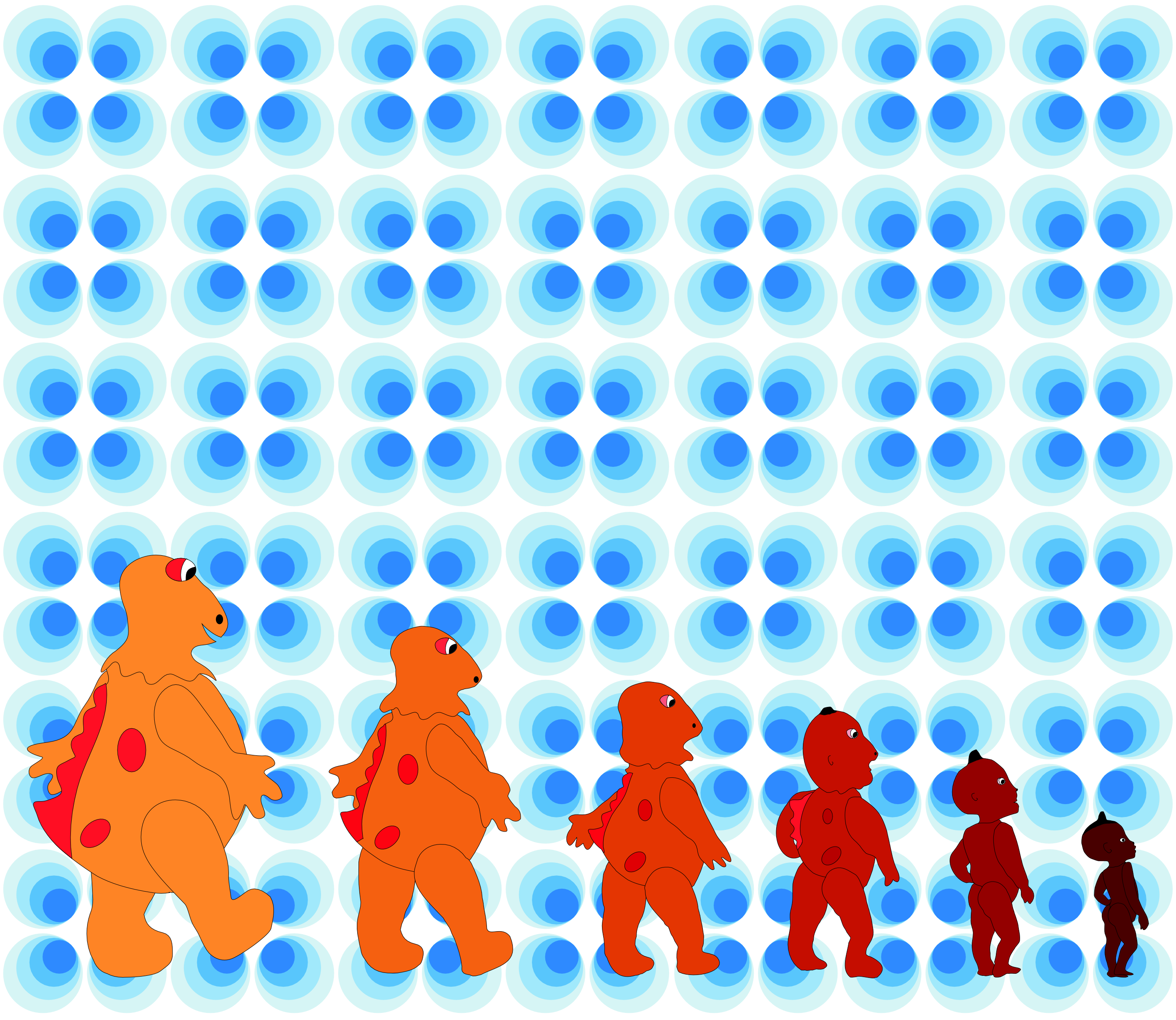
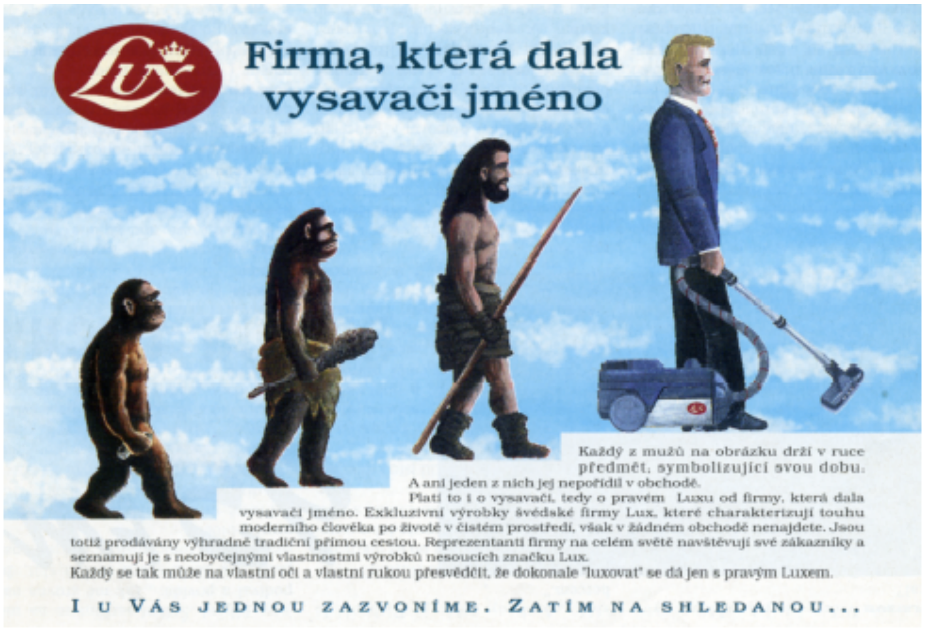


SeqinR 3.0–11




Lux
Firma, která dala vysavači jméno
I u Vás jednou zazvoníme. Zatím na shledanou...

Každý z mužů na obrázku drží v ruce předmět, symbolizující svou dobu.
A ani jeden z nich jej nepořídil v obchodě.
Platí to i o vysavači, tedy o pravém Luxu od firmy, která dala vysavači jméno. Exkluzivní výrobky švédské firmy Lux, které charakterizují touhu moderního člověka po životě v čistém prostředí, však v žádném obchodě nenajdete. Jsou totiž prodávány výhradně tradiční přímou cestou. Reprezentanti firmy na celém světě navštěvují své zákazníky a seznamují je s neobyčejnými vlastnostmi výrobků nesoucích značku Lux.
Každý se tak může na vlastní oči a vlastní rukou přesvědčit, že dokonale "luxovat" se dá jen s pravým Luxem.

Figure 1: The march of progress icon is very common in popular press. This example is from page 46 of a 1984 summer issue of the tchek edition of *Playboy*.

The march of progress icon

The cover, an artwork created[1] by Lionel Humblot, is an allusion to what Stephen J. Gould considered as a caonical icon of "[t]he most serious and pervasive of all misconceptions about evolution equates the concept with some notion of progress, usually inherent and predictable, and leading to a human pinnacle" [26]. Some examples of the so-called "march of progress icon" out of hundreds in S.J. Gould's collection from popular press are given in the begining of his famous book *Wonderful life* [25].

Note that the underlying conception predates Darwin [61]. We know now that evolution doesn not equal progress, and this is illutrated here in the cover by the unusual **decreasing** size from the initial character (on the left) to the last one (on the right).

L'île aux enfants.

The character on the left

The character on the left is called Casimir, the cult character of the french TV show *l'île aux enfants* (literally Kid's island, a french adaptation of *Sesame Street* from 1974 to 1975 and then an autonomous production until 1982 when it eventually ended). Casimir was a muppet, human-sized, with an actor playing inside, representing an orange dinosaur (the exact taxonomy has never been published) with yellow and red spots. Casimir was symbolically chosen here for two reasons. Fisrt, it's birth correspond to one of the earliest paper from our

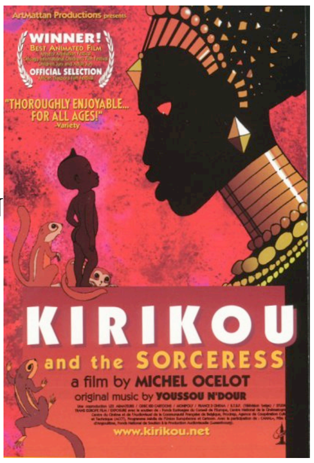
[1] with Canvas from ACD Systems.

lab about molecular evolution [31]. If you dig into **seqinR** you will find that the data from this more than 30 years old paper are still available[2]:

```
data(aaindex)
grth <- which(sapply(aaindex, function(x) length(grep("Grantham", x$A)) != 0))
lapply(aaindex[grth],"[[","D")
```

```
$GRAR740101
[1] "Composition (Grantham, 1974)"

$GRAR740102
[1] "Polarity (Grantham, 1974)"

$GRAR740103
[1] "Volume (Grantham, 1974)"
```

Second, Casimir's life span correspond more or less to the time during which the sequence analysis software called ANALSEQ[3] [38] was under development in our lab. ANALSEQ has never been published as a regular paper (although it is mentioned in one of the ACNUC paper [30]), there is only a reference manual in french [38] also available on-line at `http://biomserv.univ-lyon1.fr/doclogi/docanals/manuel.html`. ANALSEQ was entirely written in FORTRAN 77, and although you won't find any fossil code from it within **seqinR**, we wanted to credit symbolically ANALSEQ as a kind of spiritual ancestor of **seqinR** with the cover.


ArtMattan Productions presents
WINNER! Best Animated Film
OFFICIAL SELECTION
"THOROUGHLY ENJOYABLE... FOR ALL AGES!" Variety
KIRIKOU and the SORCERESS
a film by MICHEL OCELOT
original music by YOUSSOU N'DOUR
www.kirikou.net
Kirikou and the sorceress, a film by Michel Ocelot with original music by Youssou N'Dour.

The character on the right

The character on the right is called Kirikou. He is the main character of the animated film *Kirikou et la sorcière* (Kirikou and the sorceress, 1998) and *Kirikou et les bêtes sauvages* (Kirikou and the Wild Beasts, 2005). Kirikou was chosen as a symbol of **seqinR** development time. **SeqinR** started in september 2002 as part of the work of Delphine Charif's master of sciences. The first public presentation of **seqinR** was a seminar (2-JUL-2003, Lausanne University, Swiss) and the first public release on the CRAN[4] was in october 2004.

Technical details

The cover was saved from Canvas into an EPS[5] file. This file was then manually edited to remove non-ASCII characters. It was then converted into RGML[6] format with the following R code based on `grid` [78], `XML` [16] and `grImport` [64]:

```
library(grid)
library(XML)
library(grImport)
PostScriptTrace("../figs/couverture.eps", "../figs/couverture.rgml")
```

The picture was then edited to add automatically the current **seqinR** release number:

[2] thanks to `aaindex` database [43, 95, 65].
[3] not to be confused with the ANALYSEQ program by Rodger Staden [91].
[4] Comprehensive R Archive Network.
[5] Encapsulated Postscrit.
[6] RDF (Resource Description Framework) Graph Modeling Language (`http://www.cs.rpi.edu/~puninj/RGML/`).

```
cover <- readPicture("../figs/couverture.rgml")
pdf(file="../figs/cover.pdf", width = 21/2.54, height =29.7/2.54)
pushViewport(plotViewport(margins = c(0, 0, 0, 0)))
grid.picture(cover)
grid.text(paste("SeqinR", packageDescription("seqinr")$Version), gp = gpar(cex = 5),
y = unit(0.72, "npc"))
popViewport()
dev.off()
```

And finally inserted at the begining of the LaTeX file with:

```
\atxy(0cm,0cm){
  \includegraphics[width=\paperwidth,height=\paperheight]{../figs/cover}
}
```

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.0-11, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Tue May 31 17:10:53 2016
- LaTeX compilation time was: June 2, 2016

SeqinR 3.1-5: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis

Charif, D. Humblot, L. Lobry, J.R. Necşulea, A.
Palmeira, L. Penel, S.

June 2, 2016

CONTENTS

Part I

Frontmatter

CHAPTER 1

Licence of this document

Licence

Copyright © 2003-2014 J.R. Lobry. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License", that is in appendix 10 page 143.

Using and contributing

If you want to re-use or contribute to this document, some indications are given in `template.pdf` file located in the `www/src/template/` folder. All the code source is available in a `svn` repository hosted by `R-forge` at `https://r-forge.r-project.org/scm/viewvc.php/?root=seqinr`.

Part II

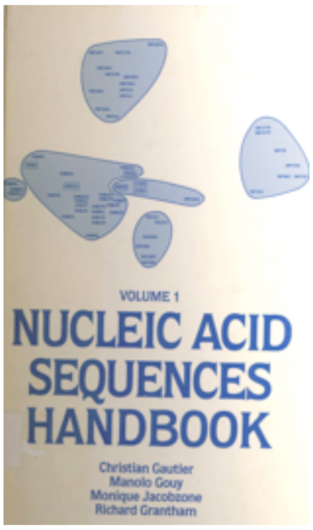
Mainmatter

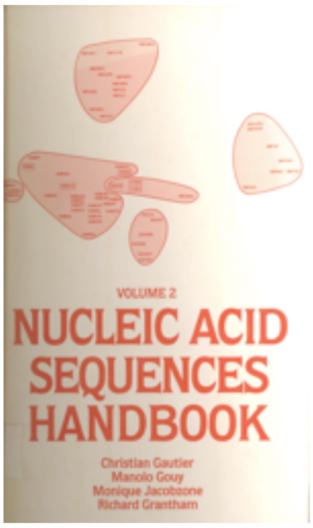
CHAPTER 2

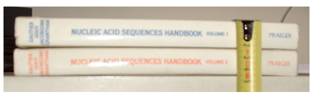
Introduction

Lobry, J.R.

2.1 About ACNUC


Cover of ACNUC book vol. 1


Cover of ACNUC book vol. 2


ACNUC books are about 4.5 cm width

ACNUC[1] was first a database of nucleic acids developed in the early 80's in the same lab (Lyon, France) that issued **seqinR**. ACNUC was first published as a printed book in two volumes [22, 23] whose covers are reproduced in margin there. At about the same time, two other databases were created, one in the USA (GenBank, at Los Alamos and now managed by the NCBI[2]), and another one in Germany (created in Köln by K. Stüber). To avoid duplication of efforts at the european level, a single repository database was initiated in Germany yielding the EMBL[3] database that moved from Köln to Heidelberg, and then to its current location at the EBI[4] near Cambridge. The DDBJ[5] started in 1986 at the NIG[6] in Mishima. These three main repository DNA databases are now collaborating to maintain the INSD[7] and are sharing data on a daily basis.

The sequences present in the ACNUC books [22, 23] were all the published nucleic acid sequences of about 150 or more continuous unambiguous nucleotides up to May or June 1981 from the journal given in table 2.1.

The total number of base pair was 526,506 in the two books. They were about 4.5 cm width. We can then compute of much place would it take to print the last GenBank release with the same format as the ACNUC book:

```
acnucbooksize <- 4.5 # cm
acnucbp <- 526506 # bp
choosebank("genbank") -> mybank
```

[1] A contraction of ACides NUCléiques, that is *NUCleic ACids* in french (http://pbil.univ-lyon1.fr/databases/acnuc/acnuc.html)

[2] National Center for Biotechnology Information

[3] European Molecular Biology Laboratory

[4] European Bioinformatic Institute

[5] DNA Data Bank of Japan

[6] National Institute of Genetics

[7] International Nucleotide Sequence Database (http://www.insdc.org/)

Journal name
Biochimie
Biochemistry (ACS)
Cell
Comptes Rendus de l'Académie des Sciences, Paris
European Journal of Biochemistry
FEBS Letters
Gene
Journal of Bacteriology
Journal of Biological Chemistry
Journal of Molecular Biology
Molecular and General Genetics
Nature
Nucleic Acids Research
Proceedings of the National Academy of Sciences of the United States of America
Science

Table 2.1: The list of journals that were manually scanned for nucleic sequences that were included in the ACNUC books [22, 23]

```
closebank()
mybank$details
```

```
[1] "              ****      ACNUC Data Base Content       ****                         "
[2] "          GenBank Release 213 (15 April 2016) Last Updated: May 22, 2016"
[3] "212,493,047,396 bases; 194,219,757 sequences; 31,530,545 subseqs; 876,736 refers."
[4] "Software M. Gouy, Lab. Biometrie et Biologie Evolutive, Universite Lyon I "
```

```
unlist(strsplit(mybank$details[3], split=" "))[1] -> bpbk
bpbk
```

```
[1] "212,493,047,396"
```

```
bpbk <- as.numeric(paste(unlist(strsplit(bpbk, split = ",")), collapse = ""))
widthcm <- acnucbooksize*bpbk/acnucbp
(widthkm <- widthcm/10^5)
```

```
[1] 18.16159
```

Our local library building in 2007 has a capacity of about 4 linear km of journals. That wouldn't be enough to store a printed version of GenBank. Picture by Lionel Clouzeau.

It would be about 18.2 kilometer long in ACNUC book format to print GenBank today (June 2, 2016). As a matter of comparison, our local universitary library buiding[8] contains about 4 km of books and journals.

2.2 About R and CRAN

R [37, 77] is a *libre* language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage at `http://www.R-project.org/` for further information.

The Comprehensive R Archive Network, CRAN, is a network of servers around the world that store identical, up-to-date, versions of code and documentation for R. At compilation time of this document, there were 95 mirrors available from 50 countries. Please use the CRAN mirror nearest to you to minimize network load, they are listed at `http://cran.r-project.org/mirrors.html`, and can be directly selected with the function `chooseCRANmirror()`.

[8]Université de Lyon, F-69000, Lyon ; Université Lyon 1 ; Bibliothèque Universitaire Sciences, 18-25-27 Avenue Claude BERNARD, F-69622, Villeurbanne, France.

2.3 About this document

In the terminology of the R project [37, 77], this document is a package *vignette*, which means that all code outputs present here were actually obtained by runing them. The examples given thereafter were run under `R version 3.2.4 (2016-03-10)` on Tue May 31 18:00:24 2016 with Sweave [51]. There is a section at the end of each chapter called **Session Informations** that gives details about packages and package versions that were involved[9]. The last compiled version of this document is available at the **seqinR** home page at `http://seqinr.r-forge.r-project.org/`.

2.4 About sequin and seqinR

Sequin is the well known sofware used to submit sequences to GenBank, **seqinR** [9] has definitively no connection with sequin. **seqinR** is just a shortcut, with no google hit, for ”Sequences in R”.

However, as a mnemotechnic tip, you may think about the **seqinR** package as the **R**eciprocal function of sequin: with sequin you can submit sequences to Genbank, with **seqinR** you can **R**etrieve sequences from Genbank (and many other sequence databases). This is a very good summary of a major functionality of the **seqinR** package: to provide an efficient access to sequence databases under R.

2.5 About getting started

You need a computer connected to the Internet. First, install R on your computer. There are distributions for Linux, Mac and Windows users on the CRAN (`http://cran.r-project.org`). Then, install the `ape`, `ade4` and `seqinr` packages. This can be done directly in an R console with for instance the command `install.packages("seqinr")`. Last, load the **seqinR** package with:

```
library(seqinr)
```

The command `lseqinr()` lists all what is defined in the package **seqinR**:

```
 lseqinr()[1:9]
```

```
[1] "a"            "aaa"          "aacost"       "aaindex"
[5] "AAstat"       "acnucclose"   "acnucopen"    "al2bp"
[9] "alllistranks"
```

We have printed here only the first 9 entries because they are too numerous. To get help on a specific function, say `aaa()`, just prefix its name with a question mark, as in `?aaa` and press enter.

2.6 About running R in batch mode

Although R is usually run in an interactive mode, some data pre-processing and analyses could be too long. You can run your R code in batch mode in a shell with a command that typically looks like :

[9] Previous versions of R and packages are available on CRAN mirrors, for instance at `http://cran.univ-lyon1.fr/src/contrib/Archive`.

```
unix$ R CMD BATCH input.R results.out &
```

where `input.R` is a text file with the R code you want to run and `results.out` a text file to store the outputs. Note that in batch mode, the graphical user interface is not active so that some graphical devices (*e.g.* `x11`, `jpeg`, `png`) are not available (see the R FAQ [35] for further details).

It's worth noting that R uses the XDR representation of binary objects in binary saved files, and these are portable across all R platforms. The `save()` and `load()` functions are very efficient (because of their binary nature) for saving and restoring any kind of R objects, in a platform independent way. To give a striking real example, at a given time on a given platform, it was about 4 minutes long to import a numeric table with 70000 lines and 64 columns with the defaults settings of the `read.table()` function. Turning it into binary format, it was then about 8 *seconds* to restore it with the `load()` function. It is therefore advisable in the `input.R` batch file to save important data or results (with something like `save(mybigdata, file = "mybigdata.RData")`) so as to be able to restore them later efficiently in the interactive mode (with something like `load("mybigdata.RData")`).

2.7 About the learning curve

Introduction

If you are used to work with a purely graphical user interface, you may feel frustrated in the beginning of the learning process because apparently simple things are not so easily obtained (*ce n'est que le premier pas qui coûte !*). In the long term, however, you are a winner for the following reasons.

2.7.1 Wheel (the)

Do not re-invent (there's a patent [44] on it anyway). At the compilation time of this document there were 8463 contributed packages available. Even if you don't want to be spoon-feed *à bouche ouverte*, it's not a bad idea to look around there just to check what's going on in your own application field. Specialists all around the world are there.

2.7.2 Hotline

There is a very reactive discussion list to help you, just make sure to read the posting guide there: `http://www.R-project.org/posting-guide.html` before posting. Because of the high traffic on this list, we strongly suggest to answer *yes* at the question *Would you like to receive list mail batched in a daily digest?* when subscribing at `https://stat.ethz.ch/mailman/listinfo/r-help`. Some *bons mots* from the list are archived in the R `fortunes` package.

2.7.3 Automation

Consider the 178 pages of figures in the additional data file 1 (`http://genomebiology.com/2002/3/10/research/0058/suppl/S1`) from [60]. They were produced in part automatically (with a proprietary software that is no more maintained) and manually, involving a lot of tedious and repetitive manipulations (such as italicising species names by hand in subtitles). In few words, a waste of time. The advantage of the R environment is that once you are happy with the outputs (including graphical outputs) of an analysis for species x, it's very easy to run the same analysis on n species.

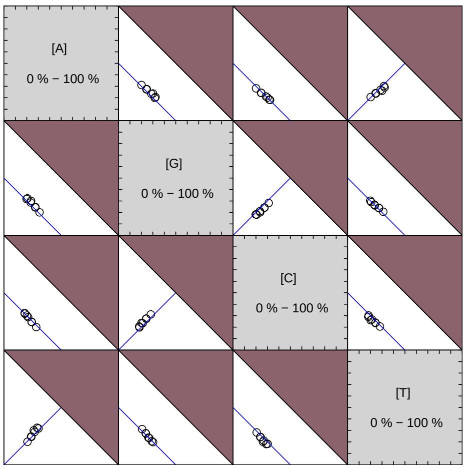
2.7.4 Reproducibility

If you do not consider the reproducibility of scientific results to be a serious problem in practice, then the paper by Jonathan Buckheit and David Donoho [7] is a must read. Molecular data are available in public databases, this is a necessary but not sufficient condition to allow for the reproducibility of results. Publishing the R source code that was used in your analyses is a simple way to greatly facilitate the reproduction of your results at the expense of no extra cost. At the expense of a little extra cost, you may consider to set up a RWeb server so that even the laziest reviewer may reproduce your results just by clicking on the "do it again" button in his web browser (*i.e.* without installing any software on his computer). For an example involving the **seqinR** pacakage, follow this link `http://pbil.univ-lyon1.fr/members/lobry/repro/bioinfo04/` to reproduce on-line the results from [10].

2.7.5 Fine tuning

You have full control on everything, even the source code for all functions is available. The following graph was specifically designed to illustrate the first experimental evidence [81] that, on average, we have also [A]=[T] and [C]=[G] in single-stranded DNA. These data from Chargaff's lab give the base composition of the L (Ligth) strand for 7 bacterial chromosomes.

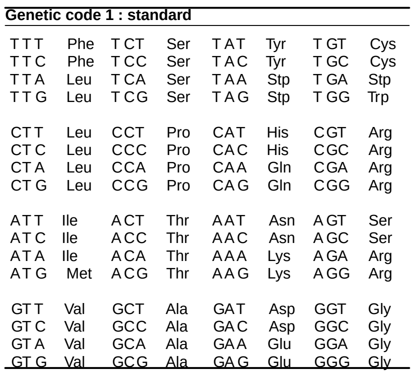
```
example(chargaff, ask = FALSE)
```



This is a very specialised graph. The filled areas correspond to non-allowed values beause the sum of the four bases frequencies cannot exceed 100%. The white areas correspond to possible values (more exactly to the projection from $\mathbb{R}^4$ to the corresponding $\mathbb{R}^2$ planes of the region of allowed values). The lines correspond to the very small subset of allowed values for which we have in addition [A]=[T] and [C]=[G]. Points represent observed values in the 7 bacterial chromosomes. The whole graph is entirely defined by the code given in the example of the `chargaff` dataset (`?chargaff` to see it).

Another example of highly specialised graph is given by the function `tablecode()` to display a genetic code as in textbooks :

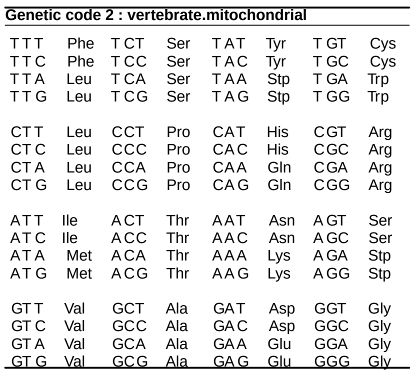
```
tablecode()
```


Genetic code 1 : standard

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

It's very convenient in practice to have a genetic code at hand, and moreover here, all genetic code variants are available :

```
tablecode(numcode = 2)
```


Genetic code 2 : vertebrate.mitochondrial

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Trp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Met	ACA	Thr	AAA	Lys	AGA	Stp
ATG	Met	ACG	Thr	AAG	Lys	AGG	Stp
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

As from **seqinR** 1.0-4, it is possible to export the table of a genetic code into a LaTeX document, for instance table 2.2 and table 2.3 were automatically generated with the following R code:

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Trp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Thr	CCT	Pro	CAT	His	CGT	Arg
CTC	Thr	CCC	Pro	CAC	His	CGC	Arg
CTA	Thr	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Thr	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Met	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 2.2: Genetic code number 3: yeast.mitochondrial.

```
tablecode(numcode = 3, latexfile = "../tables/code3.tex", size = "small")
tablecode(numcode = 4, latexfile = "../tables/code4.tex", size = "small")
```

The tables were then inserted in the LaTeX file with:

```
\input{../tables/code3.tex}
\input{../tables/code4.tex}
```

2.7.6 Data as fast moving targets

In research area, data are not always stable. Consider figure 1 from [57] which is reproduced here in figure 2.1. Data have been updated since then, but we can re-use the same R code[10] to update the figure:

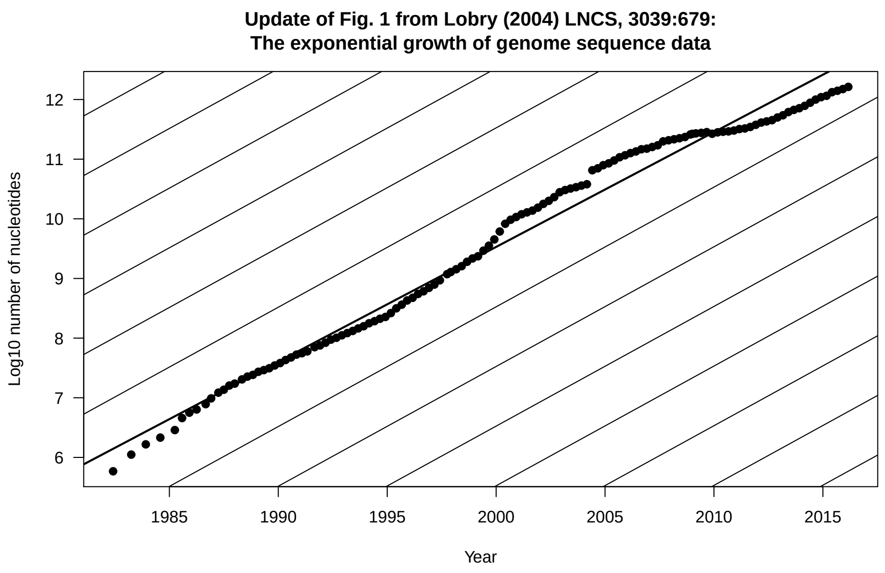
```
data <- get.db.growth()
scale <- 1
   ltymoore <- 1 # line type for Moore's law
    date <- data$date
    Nucleotides <- data$Nucleotides
    Month <- data$Month
    plot.default(date, log10(Nucleotides),
        main = "Update of Fig. 1 from Lobry (2004) LNCS, 3039:679:\nThe exponential growth of genome sequence data",
        ylab = "Log10 number of nucleotides", pch = 19, las = 1,
        cex = scale, cex.axis = scale, cex.lab = scale)
    abline(lm(log10(Nucleotides) ~ date), lwd = 2)
    lm1 <- lm(log(Nucleotides) ~ date)
    mu <- lm1$coef[2]
    dbt <- log(2)/mu
    dbt <- 12 * dbt
    x <- mean(date)
    y <- mean(log10(Nucleotides))
    a <- log10(2)/1.5
```

[10] This code was adapted from http://pbil.univ-lyon1.fr/members/lobry/repro/lncs04/.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Trp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 2.3: Genetic code number 4: protozoan.mitochondrial+mycoplasma.

```
b <- y - a * x
lm10 <- lm(log10(Nucleotides) ~ date)
for (i in seq(-10, 10, by = 1)) if (i != 0)
        abline(coef = c(b + i, a), col = "black", lty = ltymoore)
```



The doubling time is now 18.8 months.

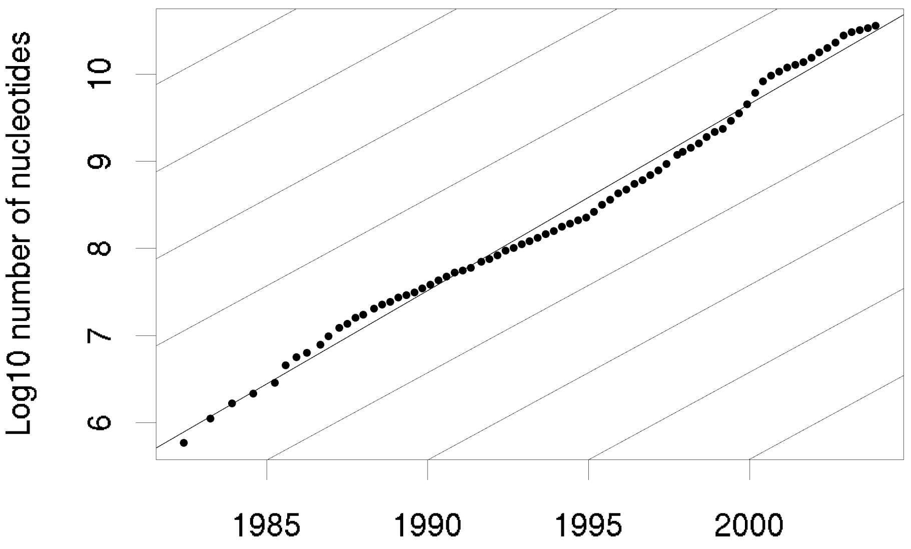


Figure 2.1: Screenshot of figure 1 from [57]. The exponential growth of genomic sequence data mimics Moore's law. The source of data is the december 2003 release note (realnote.txt) from the EMBL database available at `http://www.ebi.ac.uk/`. External lines correspond to what would be expected with a doubling time of 18 months. The central line through points is the best least square fit, corresponding to a doubling time of 16.9 months.

2.7.7 `Sweave()` and `xtable()`

For LaTeX users, it's worth mentioning the fantastic tool contributed by Friedrich Leish [51] called `Sweave()` that allows for the automatic insertion of R outputs (including graphics) in a LaTeX document. In the same spirit, there is a package called `xtable` [12] to coerce R data into LaTeX tables.

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.0-11, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Tue May 31 18:00:39 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 3

Importing sequences from flat files

Charif, D. Lobry, J.R.

3.1 Importing raw sequence data from FASTA files

3.1.1 FASTA files examples

The FASTA format is very simple and widely used for simple import of biological sequences. It was used originally by the FASTA program [72]. It begins with a single-line description starting with a character '>', followed by lines of sequence data of maximum 80 character each. Lines starting with a semi-colon character ';' are comment lines. Examples of files in FASTA format are distributed with the **seqinR** package in the `sequences` directory:

```
list.files(path = system.file("sequences", package = "seqinr"), pattern = ".fasta")
```

```
 [1] "Anouk.fasta"        "bordetella.fasta"   "ct.fasta.gz"
 [4] "DarrenObbard.fasta" "ecolicgpe5.fasta"   "gopher.fasta"
 [7] "humanMito.fasta"    "legacy.fasta"       "louse.fasta"
[10] "malM.fasta"         "ortho.fasta"        "seqAA.fasta"
[13] "smallAA.fasta"      "smallAA.fasta.gz"
```

Here is an example of a FASTA file:

```
cat(readLines(system.file("sequences/seqAA.fasta", package = "seqinr")), sep = "\n")
```

```
>A06852                 183 residues
MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEE
PQLETGPPAETMPSSITKDAEILKMMLEFVPNLPQELKATLSERQPSLRELQQSASKDSN
LNFEEFKKIILNRQNEAEDKSLLELKNLGLDKHSRKKRLFRMTLSEKCCQVGCIRKDIAR
LC*
```

Here is an example of a FASTA file with comment lines:

```
cat(readLines(system.file("sequences/legacy.fasta", package = "seqinr")), sep = "\n")
```

```
>LEGACY  921 bp
;
; Example of a FASTA file using comment lines starting with a semicolon
; as allowed in the original FASTA program:
;
;     if (line[0]!='>'&& line[0]!=';') {
;      for (i=l_offset; (n<maxs && rn < sstop)&&
;             ((ic=qascii[line[i]&AAMASK])<EL); i++)
;        if (ic<NA && ++rn > sstart) seq[n++]= ic;
;      if (ic == ES || rn > sstop) break;
;    }
;
; From file getseq.c in FASTA program version 35.2.5
;
ATGAAAATGAATAAAAGTCTCATCGTCCTCTGTTTATCAGCAGGGTTACTGGCAAGCGCG
CCTGGAATTAGCCTTGCCGATGTTAACTACGTACCGCAAAACACCAGCGACGCGCCAGCC
ATTCCATCTGCTGCGCTGCAACAACTCACCTGGACACCGGTCGATCAATCTAAAACCCAG
ACCACCCAACTGGCGACCGGCGGCCAACAACTGAACGTTCCCGGCATCAGTGGTCCGGTT
GCTGCGTACAGCGTCCCGGCAAACATTGGCGAACTGACCCTGACGCTGACCAGCGAAGTG
AACAAACAAACCAGCGTTTTTGCGCCGAACGTGCTGATTCTTGATCAGAACATGACCCCA
TCAGCCTTCTTCCCCAGCAGTTATTTCACCTACCAGGAACCAGGCGTGATGAGTGCAGAT
CGGCTGGAAGGCGTTATGCGCCTGACACCGGCGTTGGGGCAGCAAAAACTTTATGTTCTG
GTCTTTACCACGGAAAAAGATCTCCAGCAGACGACCCAACTGCTCGACCCGGCTAAAGCC
TATGCCAAGGGCGTCGGTAACTCGATCCCGGATATCCCCGATCCGGTTGCTCGTCATACC
ACCGATGGCTTACTGAAACTGAAAGTGAAAACGAACTCCAGCTCCAGCGTGTTGGTAGGA
CCCTTATTTGGTTCCTCCGCTCCAGCTCCGGTTACGGTAGGTAACACGGCGGCACCAGCT
GTGGCTGCACCCGCTCCGGCACCGGTGAAGAAAAGCGAGCCGATGCTCAACGACACGGAA
AGTTATTTTAATACCGCGATCAAAAACGCTGTCGCGAAAGGTGATGTTGATAAGGCGTTA
AAACTGCTTGATGAAGCTGAACGCTTGGGATCGACATCTGCCCGTTCCACCTTTATCAGC
AGTGTAAAAGGCAAGGGGTAA
```

3.1.2 The function `read.fasta()`

The function **`read.fasta()`** imports sequences from FASTA files into your workspace.

DNA file example

The example file looks like:

```
 dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
 cat(readLines(dnafile), sep = "\n")
>XYLEECOM.MALM  921 bp ACCESSION  E00218, X04477
ATGAAAATGAATAAAAGTCTCATCGTCCTCTGTTTATCAGCAGGGTTACTGGCAAGCGCG
CCTGGAATTAGCCTTGCCGATGTTAACTACGTACCGCAAAACACCAGCGACGCGCCAGCC
ATTCCATCTGCTGCGCTGCAACAACTCACCTGGACACCGGTCGATCAATCTAAAACCCAG
ACCACCCAACTGGCGACCGGCGGCCAACAACTGAACGTTCCCGGCATCAGTGGTCCGGTT
GCTGCGTACAGCGTCCCGGCAAACATTGGCGAACTGACCCTGACGCTGACCAGCGAAGTG
AACAAACAAACCAGCGTTTTTGCGCCGAACGTGCTGATTCTTGATCAGAACATGACCCCA
TCAGCCTTCTTCCCCAGCAGTTATTTCACCTACCAGGAACCAGGCGTGATGAGTGCAGAT
CGGCTGGAAGGCGTTATGCGCCTGACACCGGCGTTGGGGCAGCAAAAACTTTATGTTCTG
GTCTTTACCACGGAAAAAGATCTCCAGCAGACGACCCAACTGCTCGACCCGGCTAAAGCC
TATGCCAAGGGCGTCGGTAACTCGATCCCGGATATCCCCGATCCGGTTGCTCGTCATACC
ACCGATGGCTTACTGAAACTGAAAGTGAAAACGAACTCCAGCTCCAGCGTGTTGGTAGGA
CCCTTATTTGGTTCCTCCGCTCCAGCTCCGGTTACGGTAGGTAACACGGCGGCACCAGCT
GTGGCTGCACCCGCTCCGGCACCGGTGAAGAAAAGCGAGCCGATGCTCAACGACACGGAA
AGTTATTTTAATACCGCGATCAAAAACGCTGTCGCGAAAGGTGATGTTGATAAGGCGTTA
AAACTGCTTGATGAAGCTGAACGCTTGGGATCGACATCTGCCCGTTCCACCTTTATCAGC
AGTGTAAAAGGCAAGGGGTAA
```

With default arguments the output looks like:

```
 read.fasta(file = dnafile)
$XYLEECOM.MALM
  [1] "a" "t" "g" "a" "a" "a" "a" "t" "g" "a" "a" "t" "a" "a" "a" "a" "g" "t"
 [19] "c" "t" "c" "a" "t" "c" "g" "t" "c" "c" "t" "c" "t" "g" "t" "t" "t" "a"
 [37] "t" "c" "a" "g" "c" "a" "g" "g" "g" "t" "t" "a" "c" "t" "g" "g" "c" "a"
 [55] "a" "g" "c" "g" "c" "g" "c" "c" "t" "g" "g" "a" "a" "t" "t" "a" "g" "c"
 [73] "c" "t" "t" "g" "c" "c" "g" "a" "t" "g" "t" "t" "a" "a" "c" "t" "a" "c"
 [91] "g" "t" "a" "c" "c" "g" "c" "a" "a" "a" "a" "c" "a" "c" "c" "a" "g" "c"
[109] "g" "a" "c" "g" "c" "g" "c" "c" "a" "g" "c" "c" "a" "t" "t" "c" "c" "a"
```

```
[127] "t" "c" "t" "g" "c" "t" "g" "c" "g" "c" "t" "g" "c" "a" "a" "c" "a" "a"
[145] "c" "t" "c" "a" "c" "c" "t" "g" "g" "a" "c" "a" "c" "c" "g" "g" "t" "c"
[163] "g" "a" "t" "c" "a" "a" "t" "c" "t" "a" "a" "a" "a" "c" "c" "c" "a" "g"
[181] "a" "c" "c" "a" "c" "c" "c" "a" "a" "c" "t" "g" "g" "c" "g" "a" "c" "c"
[199] "g" "g" "c" "g" "g" "c" "c" "a" "a" "c" "a" "a" "c" "t" "g" "a" "a" "c"
[217] "g" "t" "t" "c" "c" "c" "g" "g" "c" "a" "t" "c" "a" "g" "t" "g" "g" "t"
[235] "c" "c" "g" "g" "t" "t" "g" "c" "t" "g" "c" "g" "t" "a" "c" "a" "g" "c"
[253] "g" "t" "c" "c" "c" "g" "g" "c" "a" "a" "a" "c" "a" "t" "t" "g" "g" "c"
[271] "g" "a" "a" "c" "t" "g" "a" "c" "c" "c" "t" "g" "a" "c" "g" "c" "t" "g"
[289] "a" "c" "c" "a" "g" "c" "g" "a" "a" "g" "t" "g" "a" "a" "c" "a" "a" "a"
[307] "c" "a" "a" "a" "c" "c" "a" "g" "c" "g" "t" "t" "t" "t" "t" "g" "c" "g"
[325] "c" "c" "g" "a" "a" "c" "g" "t" "g" "c" "t" "g" "a" "t" "t" "c" "t" "t"
[343] "g" "a" "t" "c" "a" "g" "a" "a" "c" "a" "t" "g" "a" "c" "c" "c" "c" "a"
[361] "t" "c" "a" "g" "c" "c" "t" "t" "c" "t" "t" "c" "c" "c" "c" "a" "g" "c"
[379] "a" "g" "t" "t" "a" "t" "t" "t" "c" "a" "c" "c" "t" "a" "c" "c" "a" "g"
[397] "g" "a" "a" "c" "c" "a" "g" "g" "c" "g" "t" "g" "a" "t" "g" "a" "g" "t"
[415] "g" "c" "a" "g" "a" "t" "c" "g" "g" "c" "t" "g" "g" "a" "a" "g" "g" "c"
[433] "g" "t" "t" "a" "t" "g" "c" "g" "c" "c" "t" "g" "a" "c" "a" "c" "c" "g"
[451] "g" "c" "g" "t" "t" "g" "g" "g" "g" "c" "a" "g" "c" "a" "a" "a" "a" "a"
[469] "c" "t" "t" "t" "a" "t" "g" "t" "t" "c" "t" "g" "g" "t" "c" "t" "t" "t"
[487] "a" "c" "c" "a" "c" "g" "g" "a" "a" "a" "a" "a" "g" "a" "t" "c" "t" "c"
[505] "c" "a" "g" "c" "a" "g" "a" "c" "g" "a" "c" "c" "c" "a" "a" "c" "t" "g"
[523] "c" "t" "c" "g" "a" "c" "c" "c" "g" "g" "c" "t" "a" "a" "a" "g" "c" "c"
[541] "t" "a" "t" "g" "c" "c" "a" "a" "g" "g" "g" "c" "g" "t" "c" "g" "g" "t"
[559] "a" "a" "c" "t" "c" "g" "a" "t" "c" "c" "c" "g" "g" "a" "t" "a" "t" "c"
[577] "c" "c" "c" "g" "a" "t" "c" "c" "g" "g" "t" "t" "g" "c" "t" "c" "g" "t"
[595] "c" "a" "t" "a" "c" "c" "a" "c" "c" "g" "a" "t" "g" "g" "c" "t" "t" "a"
[613] "c" "t" "g" "a" "a" "a" "c" "t" "g" "a" "a" "a" "g" "t" "g" "a" "a" "a"
[631] "a" "c" "g" "a" "a" "c" "t" "c" "c" "a" "g" "c" "t" "c" "c" "a" "g" "c"
[649] "g" "t" "g" "t" "t" "g" "g" "t" "a" "g" "g" "a" "c" "c" "c" "t" "t" "a"
[667] "t" "t" "t" "g" "g" "t" "t" "c" "c" "t" "c" "c" "g" "c" "t" "c" "c" "a"
[685] "g" "c" "t" "c" "c" "g" "g" "t" "t" "a" "c" "g" "g" "t" "a" "g" "g" "t"
[703] "a" "a" "c" "a" "c" "g" "g" "c" "g" "g" "c" "a" "c" "c" "a" "g" "c" "t"
[721] "g" "t" "g" "g" "c" "t" "g" "c" "a" "c" "c" "c" "g" "c" "t" "c" "c" "g"
[739] "g" "c" "a" "c" "c" "g" "g" "t" "g" "a" "a" "g" "a" "a" "a" "a" "g" "c"
[757] "g" "a" "g" "c" "c" "g" "a" "t" "g" "c" "t" "c" "a" "a" "c" "g" "a" "c"
[775] "a" "c" "g" "g" "a" "a" "a" "g" "t" "t" "a" "t" "t" "t" "t" "a" "a" "t"
[793] "a" "c" "c" "g" "c" "g" "a" "t" "c" "a" "a" "a" "a" "a" "c" "g" "c" "t"
[811] "g" "t" "c" "g" "c" "g" "a" "a" "a" "g" "g" "t" "g" "a" "t" "g" "t" "t"
[829] "g" "a" "t" "a" "a" "g" "g" "c" "g" "t" "t" "a" "a" "a" "a" "c" "t" "g"
[847] "c" "t" "t" "g" "a" "t" "g" "a" "a" "g" "c" "t" "g" "a" "a" "c" "g" "c"
[865] "t" "t" "g" "g" "g" "a" "t" "c" "g" "a" "c" "a" "t" "c" "t" "g" "c" "c"
[883] "c" "g" "t" "t" "c" "c" "a" "c" "c" "t" "t" "t" "a" "t" "c" "a" "g" "c"
[901] "a" "g" "t" "g" "t" "a" "a" "a" "a" "g" "g" "c" "a" "a" "g" "g" "g" "g"
[919] "t" "a" "a"
attr(,"name")
[1] "XYLEECOM.MALM"
attr(,"Annot")
[1] ">XYLEECOM.MALM  921 bp ACCESSION  E00218, X04477"
attr(,"class")
[1] "SeqFastadna"
```

As from **seqinR** 1.0-5 the automatic conversion of sequences into vector of single characters can be neutralized, for instance:

```
 read.fasta(file = dnafile, as.string = TRUE)
```

```
$XYLEECOM.MALM
[1] "atgaaaatgaataaaagtctcatcgtcctctgtttatcagcagggttactggcaagcgcgcctggaattagccttgccgatgttaactacgtaccgcaaaacaccagcgacg
attr(,"name")
[1] "XYLEECOM.MALM"
attr(,"Annot")
[1] ">XYLEECOM.MALM  921 bp ACCESSION  E00218, X04477"
attr(,"class")
[1] "SeqFastadna"
```

Forcing to lower case letters can be disabled this way:

```
 read.fasta(file = dnafile, as.string = TRUE, forceDNAtolower = FALSE)
```

```
$XYLEECOM.MALM
[1] "ATGAAAATGAATAAAAGTCTCATCGTCCTCTGTTTATCAGCAGGGTTACTGGCAAGCGCGCCTGGAATTAGCCTTGCCGATGTTAACTACGTACCGCAAAACACCAGCGACG
```

```
attr(,"name")
[1] "XYLEECOM.MALM"
attr(,"Annot")
[1] ">XYLEECOM.MALM  921 bp ACCESSION  E00218, X04477"
attr(,"class")
[1] "SeqFastadna"
```

Protein file example

The example file looks like:

```
 aafile <-  system.file("sequences/seqAA.fasta", package = "seqinr")
 cat(readLines(aafile), sep = "\n")
```

```
>A06852                 183 residues
MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEE
PQLETGPPAETMPSSITKDAEILKMMLEFVPNLPQELKATLSERQPSLRELQQSASKDSN
LNFEEFKKIILNRQNEAEDKSLLELKNLGLDKHSRKKRLFRMTLSEKCCQVGCIRKDIAR
LC*
```

Read the protein sequence file, looks like:

```
 read.fasta(aafile, seqtype = "AA")
```

```
$A06852
  [1] "M" "P" "R" "L" "F" "S" "Y" "L" "L" "G" "V" "W" "L" "L" "L" "S" "Q" "L"
 [19] "P" "R" "E" "I" "P" "G" "Q" "S" "T" "N" "D" "F" "I" "K" "A" "C" "G" "R"
 [37] "E" "L" "V" "R" "L" "W" "V" "E" "I" "C" "G" "S" "V" "S" "W" "G" "R" "T"
 [55] "A" "L" "S" "L" "E" "E" "P" "Q" "L" "E" "T" "G" "P" "P" "A" "E" "T" "M"
 [73] "P" "S" "S" "I" "T" "K" "D" "A" "E" "I" "L" "K" "M" "M" "L" "E" "F" "V"
 [91] "P" "N" "L" "P" "Q" "E" "L" "K" "A" "T" "L" "S" "E" "R" "Q" "P" "S" "L"
[109] "R" "E" "L" "Q" "Q" "S" "A" "S" "K" "D" "S" "N" "L" "N" "F" "E" "E" "F"
[127] "K" "K" "I" "I" "L" "N" "R" "Q" "N" "E" "A" "E" "D" "K" "S" "L" "L" "E"
[145] "L" "K" "N" "L" "G" "L" "D" "K" "H" "S" "R" "K" "K" "R" "L" "F" "R" "M"
[163] "T" "L" "S" "E" "K" "C" "C" "Q" "V" "G" "C" "I" "R" "K" "D" "I" "A" "R"
[181] "L" "C" "*"
attr(,"name")
[1] "A06852"
attr(,"Annot")
[1] ">A06852                 183 residues"
attr(,"class")
[1] "SeqFastaAA"
```

The same, but as string and without attributes setting, looks like:

```
 read.fasta(aafile, seqtype = "AA", as.string = TRUE, set.attributes = FALSE)
```

```
$A06852
[1] "MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEEPQLETGPPAETMPSSITKDAEILKMMLEFVPNLPQELKAT
```

Compressed file example

The original file before compression looks like:

```
 uncompressed <- system.file("sequences/smallAA.fasta", package = "seqinr")
 cat(readLines(uncompressed), sep = "\n")
```

```
>smallAA    A very small AA file in FASTA format
SEQINRSEQINRSEQINRSEQINR*
```

The compressed file example is full of mojibakes because of its binary nature, but the `readLines()` is still able to read it correctly:

```
 compressed <- system.file("sequences/smallAA.fasta.gz", package = "seqinr")
 readChar(compressed, nchar = 1000, useBytes = TRUE)
```

```
[1] "\037\x8b\b\b\xd4\024PW"
```

```
 cat(readLines(compressed), sep = "\n")
```

```
>smallAA    A very small AA file in FASTA format
SEQINRSEQINRSEQINRSEQINR*
```

We can therefore import the sequences directly from a gzipped file:

```
res1 <- read.fasta(uncompressed)
res2 <- read.fasta(compressed)
identical(res1, res2)
```

```
[1] TRUE
```

This automatic conversion works well for local files but is no more active when you read the data from an URL, for instance:

```
myurl <- "ftp://ftp.ncbi.nlm.nih.gov/refseq/release/plasmid/plasmid.1.rna.fna.gz"
try.res <- try(read.fasta(myurl))
try.res
```

```
[1] "Error in read.fasta(myurl) : no line starting with a > character found\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in read.fasta(myurl): no line starting with a > character found>
```

A simple workthrough is to encapsulate this into `gzcon()` :

```
myseq <- read.fasta(gzcon(url(myurl)))
getName(myseq)
```

```
[1] "gi|470467018|ref|NR_074151.1|" "gi|444303868|ref|NR_074290.1|"
[3] "gi|452192228|ref|NR_075742.1|" "gi|451991842|ref|NR_075394.1|"
[5] "gi|451991838|ref|NR_075390.1|" "gi|444303919|ref|NR_074342.1|"
[7] "gi|470486111|ref|NR_076736.1|" "gi|470480648|ref|NR_076426.1|"
[9] "gi|470478007|ref|NR_076423.1|"
```

3.1.3 The function `write.fasta()`

This function writes sequences to a file in FASTA format. Read 3 coding sequences sequences from a FASTA file:

```
ortho <- read.fasta(file = system.file("sequences/ortho.fasta", package = "seqinr"))
length(ortho)
```

```
[1] 3
```

```
ortho[[1]][1:12]
```

```
 [1] "a" "t" "g" "g" "c" "t" "c" "a" "g" "c" "g" "g"
```

Select only third codon positions:

```
ortho3 <- lapply(ortho, function(x) x[seq(from = 3, to = length(x), by = 3)])
ortho3[[1]][1:4]
```

```
[1] "g" "t" "g" "g"
```

Write the modified sequences to a file:

```
tmpf <- tempfile()
write.fasta(sequences = ortho3, names = names(ortho3), nbchar = 80, file.out = tmpf)
```

Read them again from the same file and check that sequences are preserved:

```
ortho3bis <- read.fasta(tmpf, set.attributes = FALSE)
identical(ortho3bis, ortho3)
```

```
[1] TRUE
```

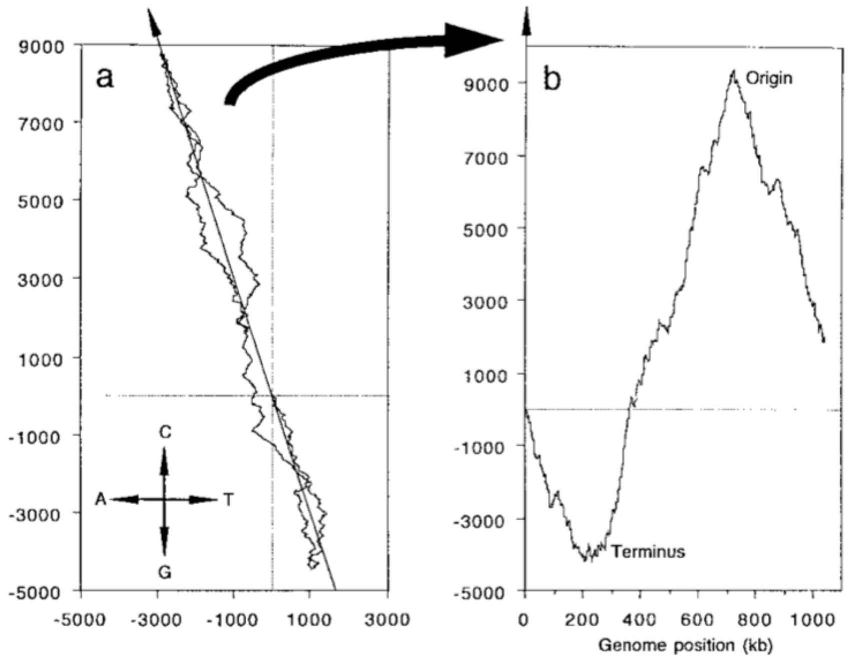


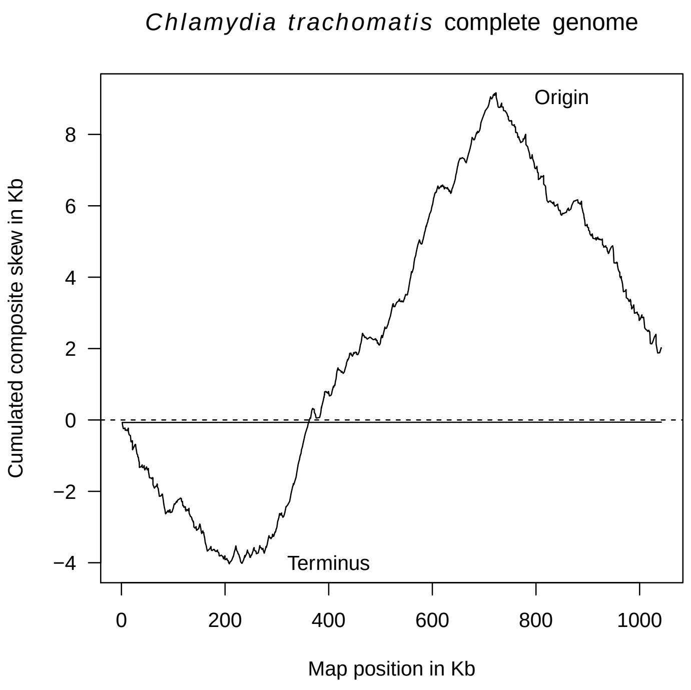
Figure 3.1: . Screenshot copy of figure 1 from [18]. The complete genome sequence of *Chlamydia trachomatis* (accession number: `AE001273`) was used to illustrate the method used by oriloc. (**a**) A DNA-walk is performed by reading the sequence in the third codon positions predicted by glimmer and walking into the plane according to the four directions defined by the four bases as indicated on the bottom left of the figure. The resulting DNA-walk is then summarized by projection onto the orthogonal regression line pointing out at about 11 o'clock in the figure. (**b**) The projected values are used as a composite skew index plotted versus map position on the chromosome. The origin is predicted at the maximum skew value while the terminus is predicted at the minimum.

3.1.4 Big room examples

Oriloc example (*Chlamydia trachomatis* complete genome)

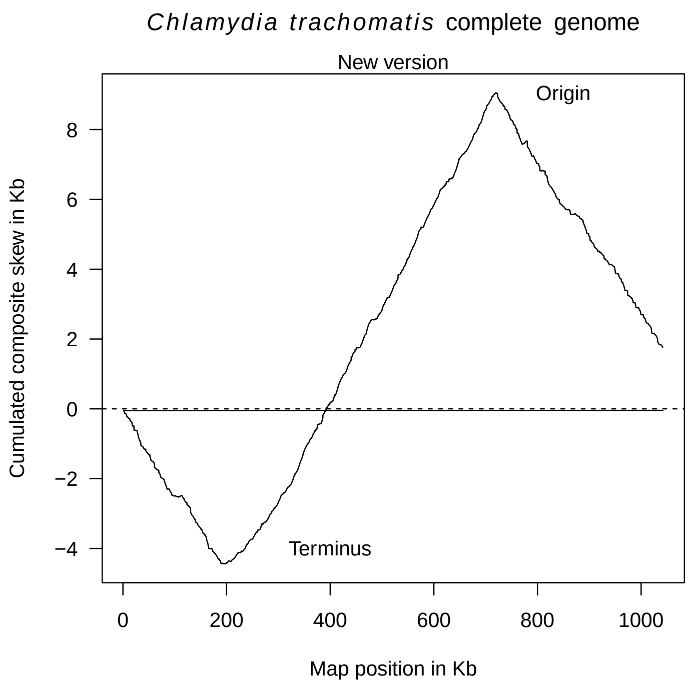
A more consequent example is given in the fasta file `ct.fasta.gz` which contains the complete genome of *Chlamydia trachomatis* that was used in [18]. You should be able to reproduce figure 1b from this paper (*cf.* screenshot in figure 3.1) with the following code:

```
out <- oriloc(seq.fasta = system.file("sequences/ct.fasta.gz", package ="seqinr"),
      g2.coord = system.file("sequences/ct.predict", package = "seqinr"),
     oldoriloc = TRUE)
plot(out$st, out$sk/1000, type="l", xlab = "Map position in Kb",
         ylab = "Cumulated composite skew in Kb",
         main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome), las = 1)
abline(h = 0, lty = 2)
text(400, -4, "Terminus")
text(850, 9, "Origin")
```



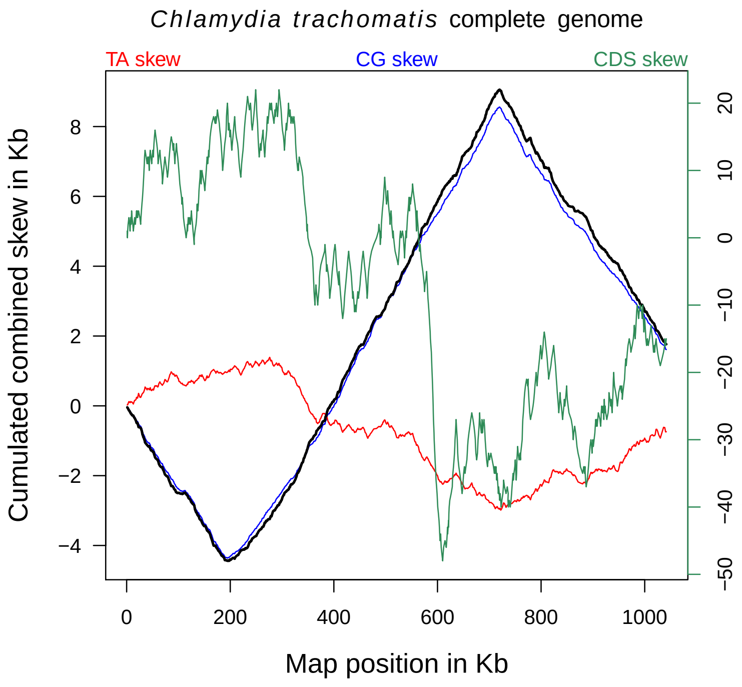
Note that the algorithm has been improved since then and that it's more advisable to use the default option `oldoriloc = FALSE` if you are interested in the prediction of origins and terminus of replication from base composition biases (more on this at `http://pbil.univ-lyon1.fr/software/oriloc.html`). See also [62] for a review on this topic. Here is the improved version:

```
out <- oriloc()
plot(out$st, out$sk/1000, type="l", xlab = "Map position in Kb",
         ylab = "Cumulated composite skew in Kb",
         main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome), las = 1)
mtext("New version")
abline(h = 0, lty = 2)
text(400, -4, "Terminus")
text(850, 9, "Origin")
```



You can also call the `draw.oriloc()` function for the simultaneous representation of the CDS, AT and GC skew along with the combined skew of the previous plots:

```
draw.oriloc(out,
  main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome),
  ta.mtext = "TA skew", ta.col = "red",
  cg.mtext = "CG skew", cg.col = "blue",
  cds.mtext = "CDS skew", cds.col = "seagreen",
  add.grid = FALSE)
```



Arabidobpsis thaliana. Source: wikipedia.

Example with 21,161 proteins from *Arabidobpsis thaliana*

As from **seqinR** 1.0-5 the automatic conversion of sequences into vector of single characters and the automatic attribute settings can be neutralized, for instance :

```
smallAA <- system.file("sequences/smallAA.fasta", package = "seqinr")
read.fasta(smallAA, seqtype = "AA", as.string = TRUE, set.attributes = FALSE)
```

```
$smallAA
[1] "SEQINRSEQINRSEQINRSEQINR*"
```

This is interesting to save time and space when reading large FASTA files. Let's give a practical example. In their paper [32], Matthew Hannah, Arnd Heyer and Dirk Hincha were working on *Arabidobpsis thaliana* genes in order to detect those involved in cold acclimation. They were interested by the detection of proteins called hydrophilins, that had a mean hydrophilicity of over 1 and glycine content of over 0.08 [20], because they are though to be important for freezing tolerance. The starting point was a FASTA file called `ATH1_pep_cm_20040228` downloaded from the Arabidopsis Information Ressource (TAIR at `http://www.arabidopsis.org/`) which contains the sequences of 21,161 proteins.

```
athfile <- "ATH1_pep_cm_20040228.fasta"
download.file(paste("http://seqinr.r-forge.r-project.org", athfile, sep = "/"),
              athfile)
system.time(ath <- read.fasta(athfile, seqtype = "AA", as.string = TRUE,
                               set.attributes = FALSE))
```

```
   user  system elapsed
  3.827   0.036   3.863
```

It's about 10 seconds here to read 21,161 protein sequences. We save them in XDR binary format[1] to read them faster later at will:

```
save(ath, file = "ath.RData")
```

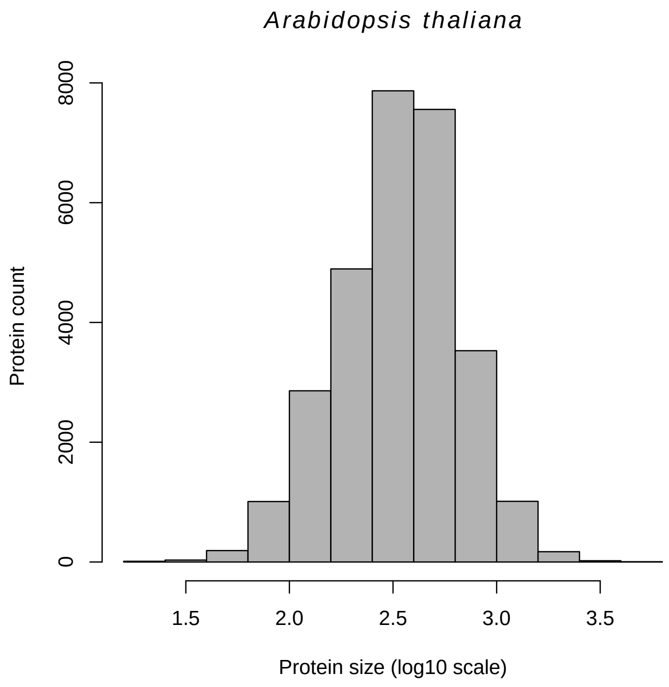
```
system.time(load("ath.RData"))
   user  system elapsed
  0.161   0.002   0.162
```

Now it's less than a second to load the whole data set thanks to the XDR format. The object size is about 15 Mo in RAM, that is something very close to the flat file size on disk:

```
object.size(ath)/2^20
16.2128143310547 bytes
file.info(athfile)$size/2^20
[1] 15.89863
```

Using strings for sequence storage is very comfortable when there is an efficient function to compute what you want. For instance, suppose that you are interested by the distribution of protein size in *Arabidopsis thaliana*. There is an efficient vectorized function called `nchar()` that will do the job, we just have to remove one unit because of the stop codon which is translated as a star (*) in this data set. This is a simple and direct task under R:

```
nres <- nchar(ath) - 1
hist(log10(nres), col = grey(0.7), xlab = "Protein size (log10 scale)",
ylab = "Protein count",
main = expression(italic(Arabidopsis~~thaliana)))
```



[1] this is a multi-platform compatible binary format: you can save data under unix and load them under Mac OS X, for instance, without problem.

However, sometimes it is more convenient to work with the single character vector representation of sequences. For instance, to count the number of glycine (G), we first play with one sequence, let's take the smallest one in the data set:

```
 which.min(nres)
```

```
At2g25990.1
       9523
```

```
 ath[[9523]]
```

```
[1] "MAGSQREKLKPRTKGSTRC*"
```

```
 s2c(ath[[9523]])
```

```
 [1] "M" "A" "G" "S" "Q" "R" "E" "K" "L" "K" "P" "R" "T" "K" "G" "S" "T" "R"
[19] "C" "*"
```

```
 s2c(ath[[9523]]) == "G"
```

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

```
 sum(s2c(ath[[9523]]) == "G")
```

```
[1] 2
```

We can now easily define a vectorised function to count the number of glycine:

```
 ngly <- function(data){
  res <- sapply(data, function(x) sum(s2c(x) == "G"))
  names(res) <- NULL
  return(res)
 }
```

Now we can use `ngly()` in the same way that `nchar()` so that computing glycine frequencies is very simple:

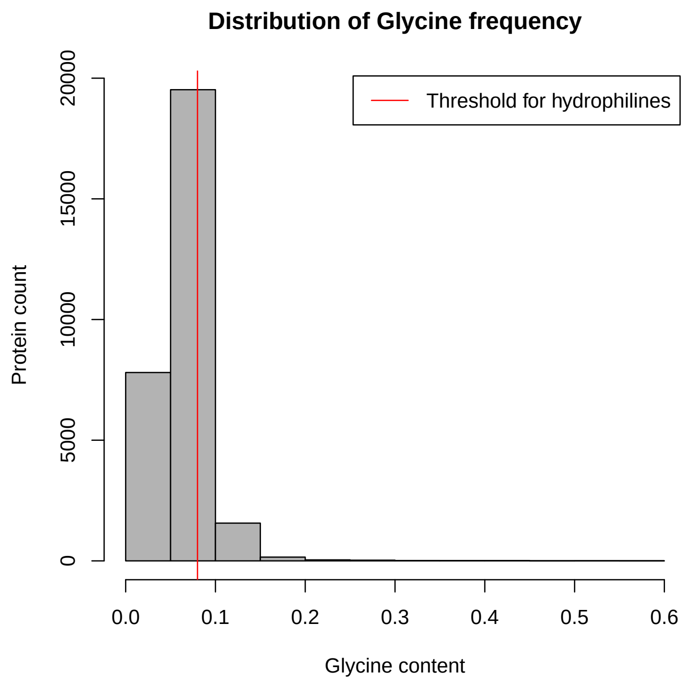
```
 ngly(ath[1:10])
```

```
 [1]  25   5  29 128   8  27  27  26  21  18
```

```
 fgly <- ngly(ath)/nres
```

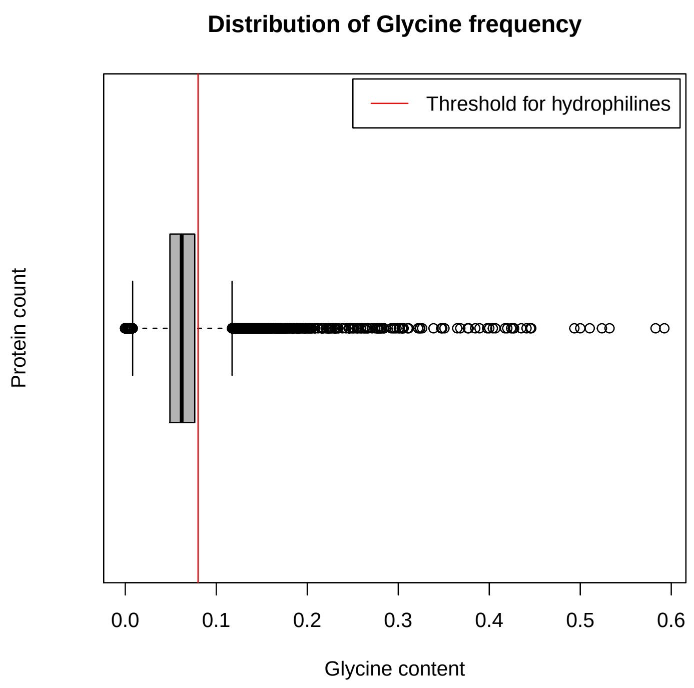
And we can have a look at the distribution:

```
 hist(fgly, col = grey(0.7), main = "Distribution of Glycine frequency",
 xlab = "Glycine content", ylab = "Protein count")
 abline(v = 0.08, col = "red")
 legend("topright",inset=0.01,lty=1,col="red",legend="Threshold for hydrophilines")
```



Let's use a boxplot instead:

```
boxplot(fgly, horizontal = TRUE, col = grey(0.7), main = "Distribution of Glycine frequency",
xlab = "Glycine content", ylab = "Protein count")
abline(v = 0.08, col = "red")
legend("topright",inset=0.01,lty=1,col="red",legend="Threshold for hydrophilines")
```



The threshold value for the glycine content in hydrophilines is therefore very close to the third quartile of the distribution:

```
summary(fgly)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.04907 0.06195 0.06475 0.07639 0.59240
```

We want now to compute something relatively more complex, we want the Kyte and Doolittle [49] hydropathy score of our proteins (aka GRAVY score). This is basically a linear form on amino acid frequencies:

$$s = \sum_{i=1}^{20} \alpha_i f_i$$

where α_i is the coefficient for amino acid number i and f_i the relative frequency of amino acid number i. The coefficients α_i are given in the `KD` component of the data set `EXP`:

```
data(EXP)
EXP$KD
 [1] -3.9 -3.5 -3.9 -3.5 -0.7 -0.7 -0.7 -0.7 -4.5 -0.8 -4.5 -0.8  4.5  4.5
[15]  1.9  4.5 -3.5 -3.2 -3.5 -3.2 -1.6 -1.6 -1.6 -1.6 -4.5 -4.5 -4.5 -4.5
[29]  3.8  3.8  3.8  3.8 -3.5 -3.5 -3.5 -3.5  1.8  1.8  1.8  1.8 -0.4 -0.4
[43] -0.4 -0.4  4.2  4.2  4.2  4.2  0.0 -1.3  0.0 -1.3 -0.8 -0.8 -0.8 -0.8
[57]  0.0  2.5 -0.9  2.5  3.8  2.8  3.8  2.8
```

This is for codons in lexical order, that is:

```
words()
 [1] "aaa" "aac" "aag" "aat" "aca" "acc" "acg" "act" "aga" "agc" "agg" "agt"
[13] "ata" "atc" "atg" "att" "caa" "cac" "cag" "cat" "cca" "ccc" "ccg" "cct"
[25] "cga" "cgc" "cgg" "cgt" "cta" "ctc" "ctg" "ctt" "gaa" "gac" "gag" "gat"
[37] "gca" "gcc" "gcg" "gct" "gga" "ggc" "ggg" "ggt" "gta" "gtc" "gtg" "gtt"
[49] "taa" "tac" "tag" "tat" "tca" "tcc" "tcg" "tct" "tga" "tgc" "tgg" "tgt"
[61] "tta" "ttc" "ttg" "ttt"
```

But since we are working with protein sequences here we name the coefficient according to their amino acid :

```
names(EXP$KD) <- sapply(words(),function(x) translate(s2c(x)))
```

We just need one value per amino acid, we sort them in the lexical order, and we reverse the scale so as to have positive values for hydrophilic proteins as in [32] :

```
kdc <- EXP$KD[unique(names(EXP$KD))]
kdc <- -kdc[order(names(kdc))]
kdc

   *    A    C    D    E    F    G    H    I    K    L    M    N    P    Q
 0.0 -1.8 -2.5  3.5  3.5 -2.8  0.4  3.2 -4.5  3.9 -3.8 -1.9  3.5  1.6  3.5
   R    S    T    V    W    Y
 4.5  0.8  0.7 -4.2  0.9  1.3
```

Now that we have the vector of coefficient α_i, we need the amino acid relative frequencies f_i, let's play with one protein first:

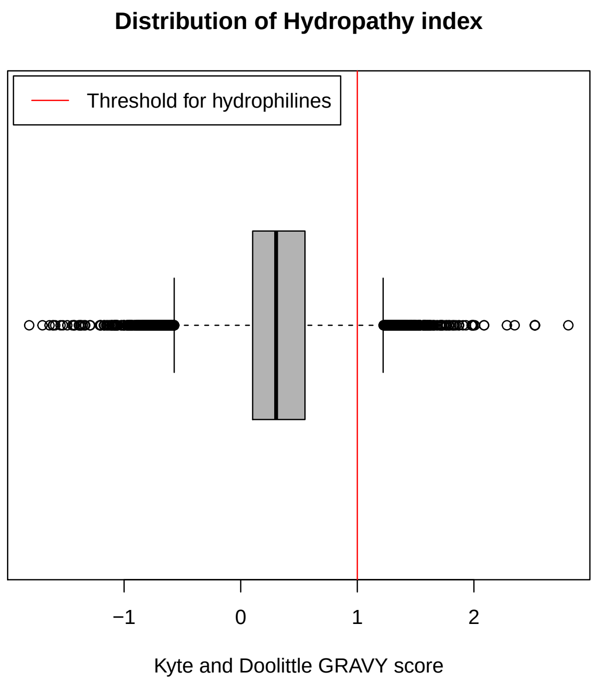
```
ath[[9523]]

[1] "MAGSQREKLKPRTKGSTRC*"

s2c(ath[[9523]])

 [1] "M" "A" "G" "S" "Q" "R" "E" "K" "L" "K" "P" "R" "T" "K" "G" "S" "T" "R"
[19] "C" "*"

table(s2c(ath[[9523]]))

* A C E G K L M P Q R S T
1 1 1 1 2 3 1 1 1 1 3 2 2

table(factor(s2c(ath[[9523]]), levels = names(kdc)))

* A C D E F G H I K L M N P Q R S T V W Y
1 1 1 0 1 0 2 0 0 3 1 1 0 1 1 3 2 2 0 0 0
```

Now that we know how to count amino acids it's relatively easy thanks to R's matrix operator `%*%` to define a vectorised function to compute a linear form on amino acid frequencies:

```
linform <- function(data, coef){
 f <- function(x){
   aaseq <- s2c(x)
   freq <- table(factor(aaseq, levels = names(coef)))/length(aaseq)
   return(coef %*% freq)
 }
 res <- sapply(data, f)
 names(res) <- NULL
 return(res)
}
kdath <- linform(ath,kdc)
```

Let's have a look at the distribution:

```
boxplot(kdath, horizontal = TRUE, col = grey(0.7),
main = "Distribution of Hydropathy index",
xlab = "Kyte and Doolittle GRAVY score")
abline(v = 1, col = "red")
legend("topleft",inset=0.01,lty=1,col="red",legend="Threshold for hydrophilines")
```



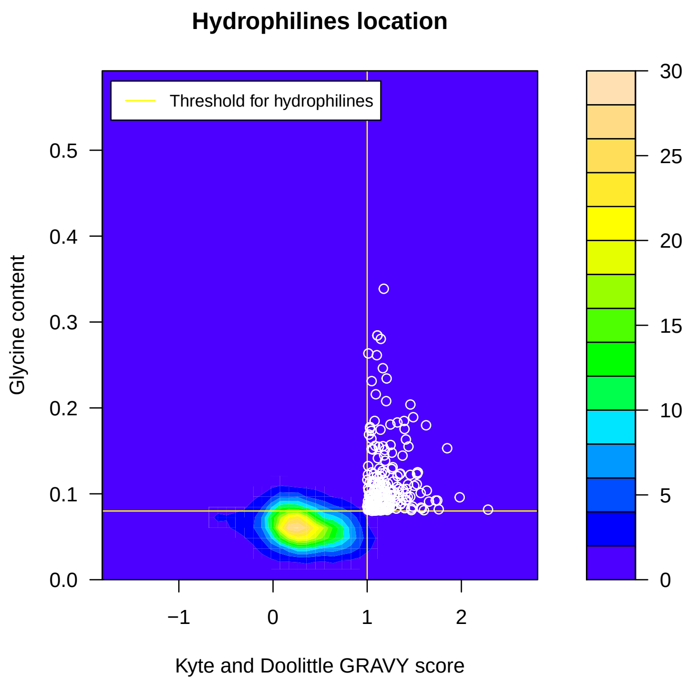
The threshold is therefore much more stringent here than the previous one on glycine content. Let's define a vector of logicals to select the hydrophilines:

```
hydrophilines <- fgly > 0.08 & kdath > 1
head(names(ath)[hydrophilines])
```

```
[1] "At1g02840.1" "At1g02840.2" "At1g02840.3" "At1g03320.1" "At1g03820.1"
[6] "At1g04450.1"
```

Check with a simple graph that there is no mistake here:

```
library(MASS)
dst <- kde2d(kdath,fgly, n = 50)
filled.contour(x = dst, color.palette = topo.colors,
plot.axes = {
  axis(1)
  axis(2)
  title(xlab="Kyte and Doolittle GRAVY score", ylab = "Glycine content",
    main = "Hydrophilines location")
  abline(v=1, col = "yellow")
  abline(h=0.08, col = "yellow")
  points(kdath[hydrophilines], fgly[hydrophilines], col = "white")
  legend("topleft",inset=0.02,lty=1,col="yellow", bg="white", legend="Threshold for hydrophilines", cex = 0.8)
  }
)
```



Everything seems to be OK, we can save the results in a data frame:

```
data.frame(list("name"=names(ath),
"KD"=kdath, "Gly"=fgly)) -> athres
head(athres)
```

```
                    name         KD        Gly
At1g01010.1 At1g01010.1  0.7297674 0.05827506
At1g01020.1 At1g01020.1 -0.1674419 0.03906250
At1g01030.1 At1g01030.1  0.8136490 0.08100559
At1g01040.1 At1g01040.1  0.4159686 0.06705081
At1g01050.1 At1g01050.1  0.4460094 0.03773585
At1g01060.1 At1g01060.1  0.7444272 0.04186047
```

We want to check now that the results are consistent with those reported previously. The following table is extracted from the file `pgen.0010026.st003.xls` provided as the supplementary material table S3 in [32] and available at `http://www.pubmedcentral.nih.gov/picrender.fcgi?artid=1189076&blobname=pgen.0010026.st003.xls`. Only the protein names, the hydrophilicity and the glycine content were extracted:

```
read.table(system.file("sequences/hannah.txt", package = "seqinr"), sep = "\t", header = TRUE)->hannah
head(hannah)
```

```
        AGI Hydrophilicity Glycine
1 At2g19570          -0.10    0.07
2 At2g45290          -0.25    0.09
3 At4g29570          -0.05    0.07
4 At4g29580          -0.10    0.06
5 At4g29600          -0.14    0.06
6 At5g28050          -0.11    0.08
```

The protein names are not exactly the same because they have no extension. As explained in [32], when multiple gene models were predicted only the first was one used. Then:

```
hannah$AGI <- paste(hannah$AGI, "1", sep = ".")
head(hannah)
```

```
          AGI Hydrophilicity Glycine
1 At2g19570.1          -0.10    0.07
2 At2g45290.1          -0.25    0.09
3 At4g29570.1          -0.05    0.07
4 At4g29580.1          -0.10    0.06
5 At4g29600.1          -0.14    0.06
6 At5g28050.1          -0.11    0.08
```

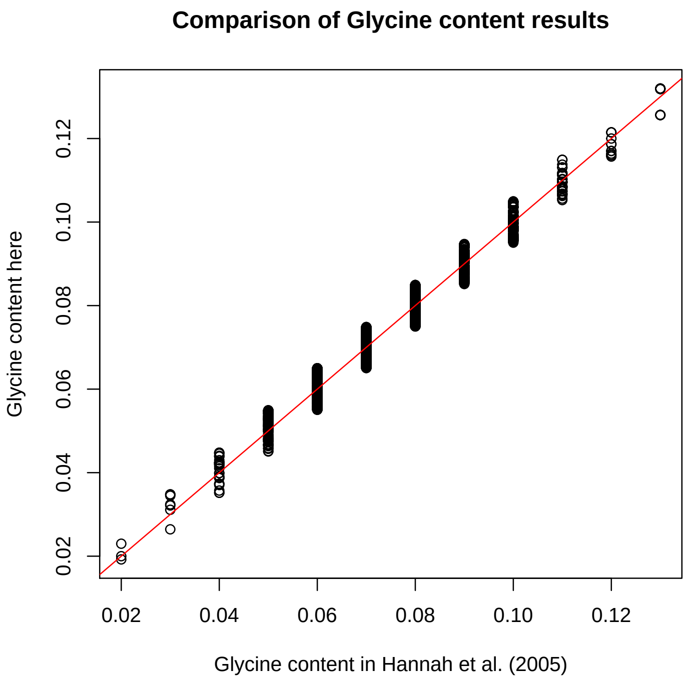
We join now the two data frames thanks to their common key:

```
join <- merge(hannah, athres, by.x = "AGI", by.y = "name")
head(join)
```

```
          AGI Hydrophilicity Glycine          KD        Gly
1 At1g01120.1          -0.10    0.06 0.106994329 0.05871212
2 At1g01390.1           0.02    0.06 0.009147609 0.06458333
3 At1g01390.1           0.02    0.06 0.009147609 0.06458333
4 At1g01420.1          -0.05    0.07 0.062033195 0.07276507
5 At1g01420.1          -0.05    0.07 0.062033195 0.07276507
6 At1g01480.1          -0.20    0.07 0.200804829 0.06653226
```

Let's compare the glycine content :

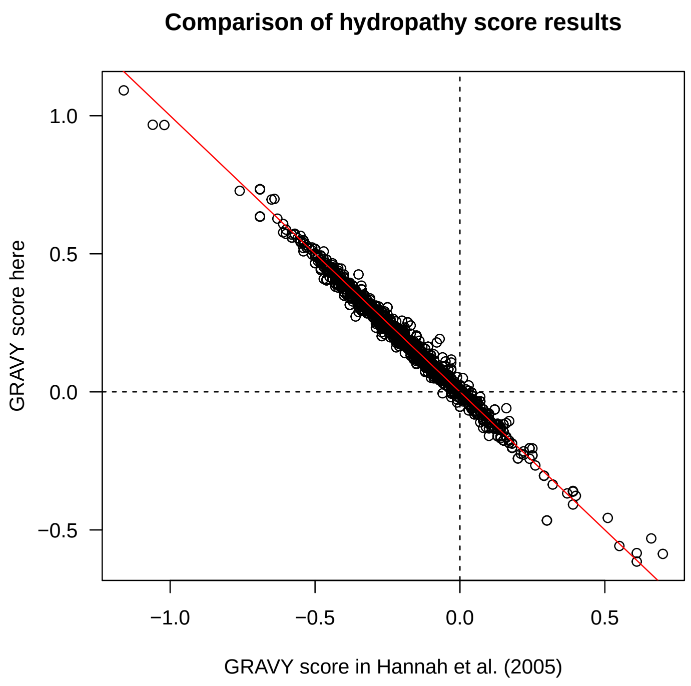
```
plot(join$Glycine, join$Gly, xlab = "Glycine content in Hannah et al. (2005)",
ylab = "Glycine content here", main = "Comparison of Glycine content results")
abline(c(0,1), col = "red")
```



The results are consistent, we have just lost some resolution because there are only two figures after the decimal point in the Excel[2] file. Let's have a look at the GRAVY score now:

[2] this software is a real **pain** for the reproducibility of results. This is well documented, see `http://www.burns-stat.com/pages/Tutor/spreadsheet_addiction.html` and references therein.

```
plot(join$Hydrophilicity, join$KD, xlab = "GRAVY score in Hannah et al. (2005)",
ylab = "GRAVY score here", main = "Comparison of hydropathy score results", las = 1)
abline(c(0,-1), col = "red")
abline(v=0, lty=2)
abline(h=0, lty=2)
```



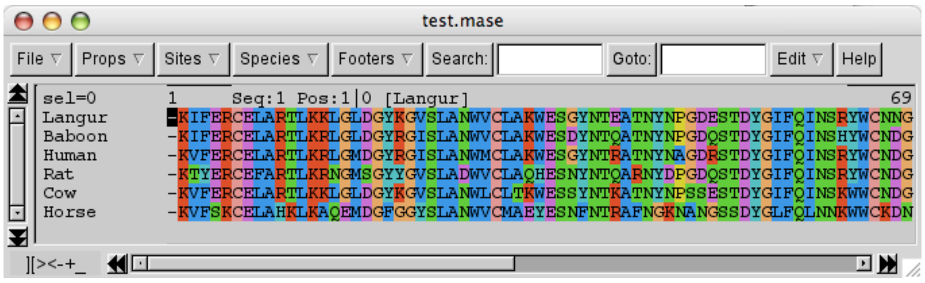
The results are consistent, it's hard to say whether the small differences are due to Excel rounding errors or because the method used to compute the GRAVY score was not exactly the same (in [32] they used the mean over a sliding window).

3.2 Importing aligned sequence data

3.2.1 Aligned sequences files examples

mase

Mase format is a flatfile format use by the SeaView multiple alignment editor [19], developed by Manolo Gouy and available at `http://pbil.univ-lyon1.fr/software/seaview.html`. The mase format is used to store nucleotide or protein multiple alignments. The beginning of the file must contain a header containing at least one line (but the content of this header may be empty). The header lines must begin by ;;. The body of the file has the following structure: First, each entry must begin by one (or more) commentary line. Commentary lines begin by the character ;. Again, this commentary line may be empty. After the commentaries, the name of the sequence is written on a separate line. At last, the sequence itself is written on the following lines.



```
sel=0        1      Seq:1 Pos:1|0 [Langur]                                   69
Langur       -KIFERCELARTLKKLGLDGYKGVSLANWVCLAKWESGYNTEATNYNPGDESTDYGIFQINSRYWCNNG
Baboon       -KIFERCELARTLKRLGLDGYRGISLANWVCLAKWESDYNTQATNYNPGDQSTDYGIFQINSHYWCNDG
Human        -KVFERCELARTLKRLGMDGYRGISLANWMCLAKWESGYNTRATNYNAGDRSTDYGIFQINSRYWCNDG
Rat          -KTYERCEFARTLKRNGMSGYYGVSLADWVCLAQHESNYNTQARNYDPGDQSTDYGIFQINSRYWCNDG
Cow          -KVFERCELARTLKKLGLDGYKGVSLANWLCLTKWESSYNTKATNYNPSSESTDYGIFQINSKWWCNDG
Horse        -KVFSKCELAHKLKAQEMDGFGGYSLANWVCMAEYESNFNTRAFNGKNANGSSDYGLFQLNNKWWCKDN
```

Figure 3.2: The file `test.mase` under SeaView. This is a graphical multiple sequence alignment editor developped by Manolo Gouy [19]. SeaView is able to read and write various alignment formats (NEXUS, MSF, CLUSTAL, FASTA, PHYLIP, MASE). It allows to manually edit the alignment, and also to run DOT-PLOT or CLUSTALW programs to locally improve the alignment.

```
 masef <- system.file("sequences/test.mase", package = "seqinr")
 cat(readLines(masef), sep = "\n")
```

```
;;Aligned by clustal on Tue Jun 30 17:36:11 1998
;empty description
Langur
-KIFERCELARTLKKLGLDGYKGVSLANWVCLAKWESGYNTEATNYNPGDESTDYGIFQINSRYWCNNGKPGAVDACHISCSALLQNNIADAVACAKRVVSDQGIRAWVAWRNHCQN
;
Baboon
-KIFERCELARTLKRLGLDGYRGISLANWVCLAKWESDYNTQATNYNPGDQSTDYGIFQINSHYWCNDGKPGAVNACHISCNALLQDNITDAVACAKRVVSDQGIRAWVAWRNHCQN
;
Human
-KVFERCELARTLKRLGMDGYRGISLANWMCLAKWESGYNTRATNYNAGDRSTDYGIFQINSRYWCNDGKPGAVNACHLSCSALLQDNIADAVACAKRVVRDQGIRAWVAWRNRCQN
;
Rat
-KTYERCEFARTLKRNGMSGYYGVSLADWVCLAQHESNYNTQARNYDPGDQSTDYGIFQINSRYWCNDGKPRAKNACGIPCSALLQDDITQAIQCAKRVVRDQGIRAWVAWQRHCKN
;
Cow
-KVFERCELARTLKKLGLDGYKGVSLANWLCLTKWESSYNTKATNYNPSSESTDYGIFQINSKWWCNDGKPNAVDGCHVSCSELMENDIAKAVACAKKIVSEQGITAWVAWKSHCRD
;
Horse
-KVFSKCELAHKLKAQEMDGFGGYSLANWVCMAEYESNFNTRAFNGKNANGSSDYGLFQLNNKWWCKDNKRSSSNACNIMCSKLLDENIDDDISCAKRVVRDKGMSAWKAWVKHCKD
```

A screenshot copy of the same file as seen under SeaView is given in figure 3.2.

clustal

The CLUSTAL format (*.aln) is the format of the ClustalW multialignment tool output [34, 97]. It can be described as follows. The word CLUSTAL is on the first line of the file. The alignment is displayed in blocks of a fixed length, each line in the block corresponding to one sequence. Each line of each block starts with the sequence name (maximum of 10 characters), followed by at least one space character. The sequence is then displayed in upper or lower cases, '-' denotes gaps. The residue number may be displayed at the end of the first line of each block.

```
 clustalf <-system.file("sequences/test.aln", package = "seqinr")
 cat(readLines(clustalf), sep = "\n")
```

```
CLUSTAL W (1.82) multiple sequence alignment


FOSB_MOUSE      MFQAFPGDYDSGSRCSSSPSAESQYLSSVDSFGSPPTAAASQECAGLGEMPGSFVPTVTA 60
FOSB_HUMAN      MFQAFPGDYDSGSRCSSSPSAESQYLSSVDSFGSPPTAAASQECAGLGEMPGSFVPTVTA 60
                ************************************************************
```

```
FOSB_MOUSE      ITTSQDLQWLVQPTLISSMAQSQGQPLASQPPAVDPYDMPGTSYSTPGLSAYSTGGASGS 120
FOSB_HUMAN      ITTSQDLQWLVQPTLISSMAQSQGQPLASQPPVVDPYDMPGTSYSTPGMSGYSSGGASGS 120
                ********************************.***************:*.**:******

FOSB_MOUSE      GGPSTSTTTSGPVSARPARARPRRPREETLTPEEEKRRVRRERNKLAAAKCRNRRRELT 180
FOSB_HUMAN      GGPSTSGTTSGPGPARPARARPRRPREETLTPEEEKRRVRRERNKLAAAKCRNRRRELT 180
                ****** ***** .**********************************************

FOSB_MOUSE      DRLQAETDQLEEEKAELESEIAELQKEKERLEFVLVAHKPGCKIPYEEGPGPGPLAEVRD 240
FOSB_HUMAN      DRLQAETDQLEEEKAELESEIAELQKEKERLEFVLVAHKPGCKIPYEEGPGPGPLAEVRD 240
                ************************************************************

FOSB_MOUSE      LPGSTSAKEDGFGWLLPPPPPPPLPFQSSRDAPPNLTASLFTHSEVQVLGDPFPVVSPSY 300
FOSB_HUMAN      LPGSAPAKEDGFSWLLPPPPPPPLPFQTSQDAPPNLTASLFTHSEVQVLGDPFPVVNPSY 300
                ****:.******.**************:*:**************************.***

FOSB_MOUSE      TSSFVLTCPEVSAFAGAQRTSGSEQPSDPLNSPSLLAL 338
FOSB_HUMAN      TSSFVLTCPEVSAFAGAQRTSGSDQPSDPLNSPSLLAL 338
                ***********************:**************
```

phylip

PHYLIP is a tree construction program [17]. The format is as follows: the number of sequences and their length (in characters) is on the first line of the file. The alignment is displayed in an interleaved or sequential format. The sequence names are limited to 10 characters and may contain blanks.

```
phylipf <- system.file("sequences/test.phylip", package = "seqinr")
cat(readLines(phylipf), sep = "\n")
```

```
  5    42
Turkey    AAGCTNGGGC ATTTCAGGGT
Salmo gairAAGCCTTGGC AGTGCAGGGT
H. SapiensACCGGTTGGC CGTTCAGGGT
Chimp     AAACCCTTGC CGTTACGCTT
Gorilla   AAACCCTTGC CGGTACGCTT

GAGCCCGGGC AATACAGGGT AT
GAGCCGTGGC CGGGCACGGT AT
ACAGGTTGGC CGTTCAGGGT AA
AAACCGAGGC CGGGACACTC AT
AAACCATTGC CGGTACGCTT AA
```

msf

MSF is the multiple sequence alignment format of the GCG sequence analysis package (http://www.accelrys.com/products/gcg/index.html). It begins with the line (all uppercase) !!NA_MULTIPLE_ALIGNMENT 1.0 for nucleic acid sequences or !!AA_MULTIPLE_ALIGNMENT 1.0 for amino acid sequences. Do not edit or delete the file type if its present (optional). A description line which contains informative text describing what is in the file. You can add this information to the top of the MSF file using a text editor (optional). A dividing line which contains the number of bases or residues in the sequence, when the file was created, and importantly, two dots (..) which act as a divider between the descriptive information and the following sequence information (required). msf files contain some other information: the Name/Weight, a Separating Line which must include two slashes (//) to divide the name/weight information from the sequence alignment (required) and the multiple sequence alignment.

```
msff <- system.file("sequences/test.msf", package = "seqinr")
cat(readLines(msff), sep = "\n")
```

```
PileUp of: @Pi3k.Fil

Symbol comparison table: GenRunData:Pileuppep.Cmp  CompCheck: 1254

                   GapWeight: 3.000
             GapLengthWeight: 0.100

 Pi3k.Msf  MSF: 377  Type: P  July 12, 1996 10:40  Check: 167 ..

 Name: Tor1_Yeast       Len:   377  Check: 7773  Weight:  1.00
 Name: Tor2_Yeast       Len:   377  Check: 8562  Weight:  1.00
 Name: Frap_Human       Len:   377  Check: 9129  Weight:  1.00
 Name: Esr1_Yeast       Len:   377  Check: 8114  Weight:  1.00
 Name: Tel1_Yeast       Len:   377  Check: 1564  Weight:  1.00
 Name: Pi4k_Human       Len:   377  Check: 8252  Weight:  1.00
 Name: Stt4_Yeast       Len:   377  Check: 9117  Weight:  1.00
 Name: Pik1_Yeast       Len:   377  Check: 3455  Weight:  1.00
 Name: P3k1_Soybn       Len:   377  Check: 4973  Weight:  1.00
 Name: P3k2_Soybn       Len:   377  Check: 4632  Weight:  1.00
 Name: Pi3k_Arath       Len:   377  Check: 3585  Weight:  1.00
 Name: Vp34_Yeast       Len:   377  Check: 5928  Weight:  1.00
 Name: P11a_Human       Len:   377  Check: 6597  Weight:  1.00
 Name: P11b_Human       Len:   377  Check: 8486  Weight:  1.00

//

            1                                                  50
Tor1_Yeast  .......GHE DIRQDSLVMQ LFGLVNTLLK NDSECFKRHL DIQQYPAIPL
Tor2_Yeast  .......GHE DIRQDSLVMQ LFGLVNTLLQ NDAECFRRHL DIQQYPAIPL
Frap_Human  .......GHE DLRQDERVMQ LFGLVNTLLA NDPTSLRKNL SIQRYAVIPL
Esr1_Yeast  .......KKE DVRQDNQYMQ FATTMDFLLS KDIASRKRSL GINIYSVLSL
Tel1_Yeast  .KALMKGSND DLRQDAIMEQ VFQQVNKVLQ NDKVLRNLDL GIRTYKVVPL
Pi4k_Human  ..AAIFKVGD DCRQDMLALQ IIDLFKNIFQ LV....GLDL FVFPYRVVAT
Stt4_Yeast  ..AAIFKVGD DCRQDVLALQ LISLFRTIWS SI....GLDV YVFPYRVTAT
Pik1_Yeast  ...VIAKTGD DLRQEAFAYQ MIQAMANIWV KE....KVDV WVKRMKILIT
P3k1_Soybn  TCKIIFKKGD DLRQDQLVVQ MVSLMDRLLK LE....NLDL HLTPYKVLAT
P3k2_Soybn  ....IFKKGD DIRQDQLVVQ MVSLMDRLLK LE....NLDL HLTPYKVLAT
Pi3k_Arath  ..KLIFKKGD DLRQDQLVVQ MVWLMDRLLK LE....NLDL CLTPYKVLAT
Vp34_Yeast  .YHLMFKVGD DLRQDQLVVQ IISLMNELLK NE....NVDL KLTPYKILAT
P11a_Human  ...IIFKNGD DLRQDMLTLQ IIRIMENIWQ NQ....GLDL RMLPYGCLSI
P11b_Human  ...VIFKNGD DLRQDMLTLQ MLRLMDLLWK EA....GLDL RMLPYGCLAT

            51                                                100
Tor1_Yeast  SPKSGLLGWV PNSDTFHVLI REHRDAKKIP LNIEHWVMLQ MAPDYENLTL
Tor2_Yeast  SPKSGLLGWV PNSDTFHVLI REHREAKKIP LNIEHWVMLQ MAPDYDNLTL
Frap_Human  STNSGLIGWV PHCDTLHALI RDYREKKKIL LNIEHRIMLR MAPDYDHLTL
Esr1_Yeast  REDCGILEMV PNVVTLRSIL STKYESLKIK Y.....SLKS LHDRWQHTAV
Tel1_Yeast  GPKAGIIEFV ANSTSLHQIL SKLHTNDKIT FDQARKGMKA VQTKSN....
Pi4k_Human  APGCGVIECI PDCTS..... .......... RDQLGRQTDF GMYDYFTRQY
Stt4_Yeast  APGCGVIDVL PNSVS..... .......... RDMLGREAVN GLYEYFTSKF
Pik1_Yeast  SANTGLVETI TNAMSVHSIK KALTKKMIED AELDDKGGIA SLNDHFLRAF
P3k1_Soybn  GQDEGMLEFI P.SRSLAQI. .......... ..LSENRSII SYLQ......
P3k2_Soybn  GQDEGMLEFI P.SRSLAQI. .......... ..LSENRSII SYLQ......
Pi3k_Arath  GHDEGMLEFI P.SRSLAQI. .......... ..LSEHRSIT SYLQ......
Vp34_Yeast  GPQEGAIEFI P.NDTLASI. .......... ..LSKYHGIL GYLK......
P11a_Human  GDCVGLIEVV RNSHTIMQI. .......... ..Q.CKGGLK GALQFNSHTL
P11b_Human  GDRSGLIEVV STSETIADI. .......... ..QLNSSNVA AAAAFNKDAL
```

FASTA

Sequence in fasta format begins with a single-line description (distinguished by a greater-than (>) symbol), followed by sequence data on the next line.

```
 fastaf <- system.file("sequences/Anouk.fasta", package = "seqinr")
 cat(readLines(fastaf), sep = "\n")
```

```
>LmjF01.0030
ATGATGTCGGCCGAGCCGCCGTCGTCGCAGCCGTACATCAGCGACGTGCTGCGGCGGTAC
CAGCTGGAGCGCTTTCAGTGTGCCTTTGCATCGAGCATGACCATCAAGGACCTCCTCGCC
CTGCAGCCAGAGGACTTCAACCGCTACGGCGTCGTAGAGGCGATGGACATTTTGCGGCTG
CGTGACGCCATCGAGTACATCAAGGCTAATCCGCTCCCCGCCTCGCGCTCTGGCAGTGAC
GTGCTCGACAACGACGGCGACGGCGACGGCGACGACAGTACGCCGGAGGGGAAGGAGGGG
TGCTCGACGGAGCGCCGGCGGCAGTACACAGCACGCGGAACCACAGTCCTTTGCCGGTCG
ACCGACACCGCCGAGGAGGTGAAGCGCAAGAGCCGCATCCTCGTCGCCATTCGCAAGCGT
CCGCTCAGCGCCGGGGAGCAGACGAACGGCTTCACGGACATCATGGACGCCGACAACAGC
GGCGAGATTGTGCTGAAGGAGCCAAAGGTGAAGGTCGACCTCCGCAAGTACACCCACGTG
CACCGCTTCTTCTTCGACGAGGTTTTCGACGAGGCCTGCGACAACGTCGACGTGTACAAC
CGCGCTGCCCGCGCGCTGATCGACACCGTCTTCGACGGCGGCTGCGCGACATGCTTCGCC
TATGGACAGACAGGGAGCGGCAAGACACACACGATGCTGGGCAAGGGCCCCGAGCCGGGC
```

```
CTCTACGCACTCGCCGCCAAAGACATGTTTGACCGCCTCACGAGCGACACGCGCATCGTC
GTTTCCTTTTACGAGATCTACAGCGGGAAGCTCTTTGACTTGCTGAACGGCCGGCGACCC
CTGCGAGCCCTCGAGGACGACAAGGGCCGGGTGAACATCCGCGGCCTCACCGAACACTGC
TCTACCAGCGTGGAGGACCTCATGACGATCATCGACCAGGGCAGCGGTGTTCGCAGCTGC
GGCTCCACCGGCGCCAATGACACAAGCTCCCGCTCCCACGCCATTCTCGAGATCAAGCTC
AAGGCGAAACGGACGTCGAAGCAGAGCGGCAAGTTCACGTTCATCGACCTCGCTGGAAGC
GAGCGCGGCGCTGACACGGTGGACTGCGCGCGACAGACACGCCTCGAAGGGGCGGAGATC
AACAAGAGCCTACTCGCGCTGAAGGAGTGCATTCGTTTTTTAGATCAGAACAGGAAGCAC
GTCCCGTTCCGCGGCTCGAAGCTGACTGAGGTGCTCCGCGACTCGTTTATCGGCAACTGC
CGCACGGTGATGATCGGCGCCGTCTCTCCGTCGAACAACAATGCCGAGCACACGCTGAAC
ACGCTGCGCTACGCCGATCGTGTCAAGGAGCTGAAGCGCAACGCCACGGAGCGGCGCACT
GTGTGCATGCCCGACGACCAGGAAGAGGCCTTCTTTGACACGACCGAGAGCAGGCCACCG
TCGCGGAGGACGACAACTCGCCTTTCTACGGCCGCCCCGCTTTTCTCCGGCTCTTCGACG
GCTGCGCCAGCACTTAGAAGCACGCTACTCAGCAGCCGCTCCGTCAACACACTCTCGCCG
TCGTCGCAGGCCAAGTCGACTCTCGTCACCCCGAAGCCGCCGTCGCGCGATCGGACTCCG
GACATGGTGTGCACTAAGCGGCCCCGCGACTCAGACAGAAGCGGCGAGGACGAAGTGGTA
GCGCGGCCGAGTGGGCGCCCAAGCTTCAAGCGCTTCGAGAGCGGCGCCGAGCTTGTCGCG
GCCCAGCGCAGTCGCGTCATTGACCAATACAACGCCTACCTCGAGACGGACATGAACTGT
ATCAAGGAGGAGTACCAGGTGAAGTACGACGCAGAGCAGATGAACGCCAACACGCGCAGC
TTTGTGGAGCGCGCACGTCTGCTGGTGAGCGAGAAACGGCGCGCGATGGAGTCCTTCCTA
ACGCAGCTGGAGGAGCTCGACAAGATCGCGCAGCAGGTCGCCGACATCACCGCCTTTCAG
CAGCACCTGCCGCCAACG
>LinJ01.0030
ATGATGTCGGCCGAGCCGCCGTCGTCGCAGCCGTACATCAGCGACGTGCTGCGGCGGTAC
CAGCTGGAGCGCTTTCAGAGTTCCTTTGCATCGAGCATGACCATCAAGGACCTCCTCGCC
CTGCAGCCGGAGGACTTCAACCGCTACGGCGTCGTAGAGGCAATGGACATTTTGCGGCTG
CGCGACGCCATCGAGTACATCAAGGCCAACCCGCTCCCCGCCTCGCGCTCCGGCAGTGAC
GTGCTCGACAACGACGGCGACGGCGACGGCGACGACAGTACGCCGGAGGGGAAGGAGGGG
TGCTCGACGGAGCGCCGACGGCAGTACACAGCACGCGGAACCACCGTCCTTTGCGGGTCG
ACCGACACCGCCGAGGAGGTGAAGCGCAAGAGCCGCATCATCGTCGCCATTCGCAAGCGT
CCGCTCAGCGCCGGGGAGCAGACGAACGGCTTCACGGACATCATGGACGCCGACAACAAC
GGCGAGATTGTGCTGAAGGAGCCAAAGGTGAAGGTCGACCTCCGCAAGTACACCCACGTG
CACCGCTTCTTCTTCGACGAGGTTTTCGACGAGGCGTGCGACAACGTCGACGTGTACAAC
CGCGCTGCCCGCGCGCTGATCGACACCGTCTTCGACGGCGGCTGCGCGACATGCTTCGCC
TATGGGCAGACAGGGAGCGGCAAGACACACACGATGCTCGGCAAGGGCCCCGAGCCGGGC
CTGTACGCACTCGCCGCCAAAGACATGTTTGACCGCCTCACGAGCGACACGCGCATCGTT
GTTTCCTTTTACGAGATCTACAGCGGGAAGCTCTTTGACTTGCTGAACGGCCGGCGACCA
CTGCGAGCCCTCGAGGACGACAAGGGGAGGGTGAACATCCGCGGCCTCACCGAACACTGC
TCTACCAGCGTGGAGGACCTCATGACGATCATCGACCAGGGCAGCGGCGTTCGCAGCTGC
GGCTCCACCGGCGCCAACGACACGAGCTCCCGCTCCCACGCCATTCTCGAGATCAAGCTC
AAGGCGAAACGGACGTCGAAGCAGAGCGGCAAGTTCACATTCATCGACCTCGCTGGAAGC
GAGCGCGGCGCCGACACGGTGGATTGCGCGCGACAGACACGCCTCGAAGGGGCGGAGATT
AACAAGAGCCTACTCGCTCTGAAGGAGTGCATTCGTTTTTTAGATCAGAACAGGAAGCAC
GTCCCGTTCCGCGGCTCGAAGCTGACTGAGGTGCTCCGCGACTCGTTTATCGGCAACTGC
CGCACGGTGATGATCGGCGCCGTCTCTCCGTCCAACAACAATGCCGAGCACACGCTGAAC
ACGTTGCGCTACGCCGATCGCGTCAAGGAGCTGAAGCGCAACGCCACGGAGCGGCGCACC
GTGTGCGTGCCCAACGACCAGGAAGAGGCCTTCTTTGACACGACCGAGAGCAGGCCACCG
TCGCGGAGGACGACAACTCGGCTTTCTGCGGCCGCCCCGCTTTTCTCCGGCACTTCGACG
GCTGCCCCAGCATGTAAAAGCACGTTGCTCAGCAGCCGCTCCGTCAACACACTCTCGCCG
TCGTCGCAGGGCAAGTCGACTCTCGTCACCCCGAAGCCACTGTCGCGCGATCGGACTCCG
GACATGGTGTGCGCTAAGCGGCCCCGCGACTCAGACCGAAGCGGCGAAGACGAAGTGGTG
GCGCGGCCGAGTGGGCGCCCAAGCTTCAAGCGCTTCGAGGGCGGCGCCGAGCTCGTGGCG
GCCCAGCGCAGTCGTGTCATTGACCAATACAACGCCTACCTCGAGACGGACATGAACTGT
ATCAAGGAGGAGTACCAGGTGAAGTACGACGCAGAGCAGATGAACGCCAACACGCGCACC
TTTGTCGAGCGCGCACGCCTGCTGGTGAGCGAGAAGCGGCGCGCGATGGAGTCCTTCCTA
ACGCAGCTGGACGAGCTCGATAAGATCGCGCAGCAGGTCGCCAGCATCACCGCCTTTCAG
CAGCACCTGCCGCCAACG
```

3.2.2 The function `read.alignment()`

Aligned sequence data are very important in evolutionary studies, in this representation all vertically aligned positions are supposed to be homologous, that is sharing a common ancestor. This is a mandatory starting point for comparative studies. There is a function in **seqinR** called `read.alignment()` to read aligned sequences data from various formats (`mase`, `clustal`, `phylip`, `fasta` or `msf`) produced by common external programs for multiple sequence alignment.

```
 example(read.alignment)
```

```
rd.lgn mase.res   <- read.alignment(file = system.file("sequences/test.mase", package = "seqinr"),
rd.lgn  format = "mase")
```

```
rd.lgn clustal.res <- read.alignment(file = system.file("sequences/test.aln", package = "seqinr"),
rd.lgn  format="clustal")
rd.lgn phylip.res  <- read.alignment(file = system.file("sequences/test.phylip", package = "seqinr"),
rd.lgn  format = "phylip")

rd.lgn msf.res      <- read.alignment(file = system.file("sequences/test.msf", package = "seqinr"),
rd.lgn  format = "msf")

rd.lgn fasta.res    <- read.alignment(file = system.file("sequences/Anouk.fasta", package = "seqinr"),
rd.lgn  format = "fasta")

rd.lgn #
rd.lgn # Quality control routine sanity checks:
rd.lgn #
rd.lgn
rd.lgn data(mase); stopifnot(identical(mase, mase.res))

rd.lgn data(clustal); stopifnot(identical(clustal, clustal.res))

rd.lgn data(phylip); stopifnot(identical(phylip, phylip.res))

rd.lgn data(msf); stopifnot(identical(msf, msf.res))

rd.lgn data(fasta); stopifnot(identical(fasta, fasta.res))
```

3.2.3 A simple example with the louse-gopher data

Let's give an example. The gene coding for the mitochondrial cytochrome oxidase I is essential and therefore often used in phylogenetic studies because of its ubiquitous nature. The following two sample tests of aligned sequences of this gene (extracted from ParaFit [50]), are distributed along with the **seqinR** package:

```
louse <- read.alignment(system.file("sequences/louse.fasta", package = "seqinr"), format = "fasta")
louse$nam
```

```
[1] "gi|548117|gb|L32667.1|GYDCYTOXIB Geomydoecus chapini mitochondrial cytochrome oxidase I gene, partial cds"
[2] "gi|548119|gb|L32668.1|GYDCYTOXIC Geomydoecus cherriei mitochondrial cytochrome oxidase I gene, partial cds"
[3] "gi|548121|gb|L32669.1|GYDCYTOXID Geomydoecus costaricensis mitochondrial cytochrome oxidase I gene, partial cds"
[4] "gi|548125|gb|L32671.1|GYDCYTOXIF Geomydoecus ewingi mitochondrial cytochrome oxidase I gene, partial cds"
[5] "gi|548127|gb|L32672.1|GYDCYTOXIG Geomydoecus geomydis mitochondrial cytochrome oxidase I gene, partial cds"
[6] "gi|548131|gb|L32675.1|GYDCYTOXII Geomydoecus oklahomensis mitochondrial cytochrome oxidase I gene, partial cds"
[7] "gi|548133|gb|L32676.1|GYDCYTOXIJ Geomydoecus panamensis mitochondrial cytochrome oxidase I gene, partial cds"
[8] "gi|548137|gb|L32678.1|GYDCYTOXIL Geomydoecus setzeri mitochondrial cytochrome oxidase I gene, partial cds"
```

```
gopher <- read.alignment(system.file("sequences/gopher.fasta", package = "seqinr"), format = "fasta")
gopher$nam
```

```
[1] "gi|548223|gb|L32683.1|PPGCYTOXIA Geomys breviceps mitochondrial cytochrome oxidase I gene, partial cds"
[2] "gi|548197|gb|L32686.1|OGOCYTOXIA Orthogeomys cavator mitochondrial cytochrome oxidase I gene, partial cds"
[3] "gi|548199|gb|L32687.1|OGOCYTOXIB Orthogeomys cherriei mitochondrial cytochrome oxidase I gene, partial cds"
[4] "gi|548201|gb|L32691.1|OGOCYTOXIC Orthogeomys underwoodi mitochondrial cytochrome oxidase I gene, partial cds"
[5] "gi|548203|gb|L32692.1|OGOCYTOXID Orthogeomys hispidus mitochondrial cytochrome oxidase I gene, partial cds"
[6] "gi|548229|gb|L32693.1|PPGCYTOXID Geomys bursarius mitochondrial cytochrome oxidase I gene, partial cds"
[7] "gi|548231|gb|L32694.1|PPGCYTOXIE Geomys bursarius mitochondrial cytochrome oxidase I gene, partial cds"
[8] "gi|548205|gb|L32696.1|OGOCYTOXIE Orthogeomys heterodus mitochondrial cytochrome oxidase I gene, partial cds"
```

The aligned sequences are now imported in your R environment. The 8 genes of the first sample are from various species of louse (insects parasitics on warm-blooded animals) and the 8 genes of the second sample are from their corresponding gopher hosts (a subset of rodents), see figure 3.3 :

```
l.names <- readLines(system.file("sequences/louse.names", package = "seqinr"))
l.names
```

```
[1] "G.chapini "  "G.cherriei " "G.costaric " "G.ewingi   "  "G.geomydis "
[6] "G.oklahome " "G.panamens " "G.setzeri  "
```

```
g.names <- readLines(system.file("sequences/gopher.names", package = "seqinr"))
g.names
```

```
[1] "G.brevicep " "O.cavator "  "O.cherriei " "O.underwoo " "O.hispidus "
[6] "G.burs1 "    "G.burs2 "    "O.heterodu"
```

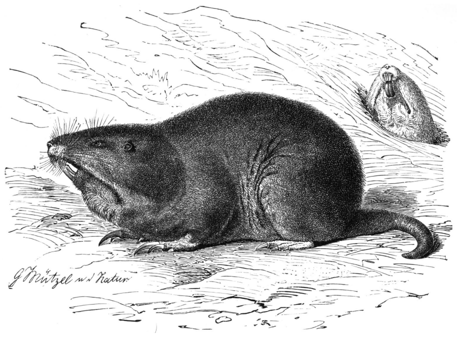


Figure 3.3: Louse (left) and gopher (right). Images are from the wikipedia (`http://www.wikipedia.org/`). The picture of the chewing louse *Damalinia limbata* found on Angora goats was taken by Fiorella Carnevali (ENEA, Italy). The gopher drawing is from Gustav Mützel, Brehms Tierleben, Small Edition 1927.

SeqinR has very few methods devoted to phylogenetic analyses but many are available in the `ape` package [70]. This allows for a very fine tuning of the graphical outputs of the analyses thanks to the power of the R facilities. For instance, a natural question here would be to compare the topology of the tree of the hosts and their parasites to see if we have congruence between host and parasite evolution. In other words, we want to display two phylogenetic trees face to face. This would be tedious with a program devoted to the display of a single phylogenetic tree at time, involving a lot of manual copy/paste operations, hard to reproduce, and then boring to maintain with data updates.

How does it looks under R? First, we need to *infer* the tree topologies from data. Let's try as an *illustration* the famous neighbor-joining tree estimation of Saitou and Nei [82] with Jukes and Cantor's correction [40] for multiple substitutions.

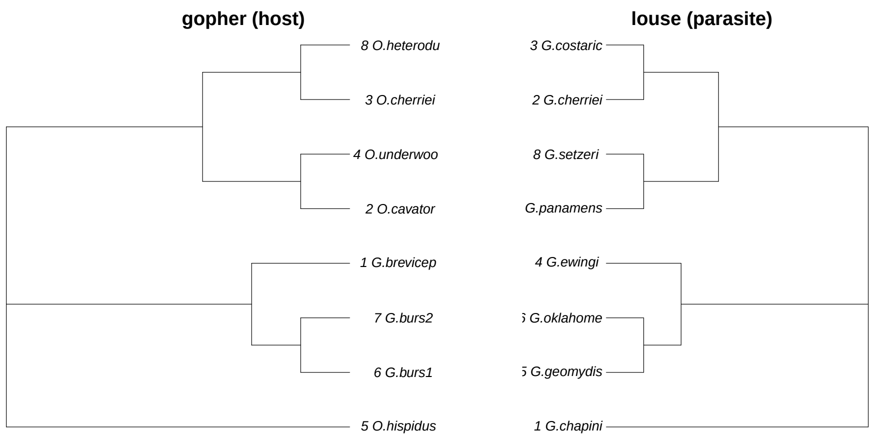
```
library(ape)
louse.JC <- dist.dna(as.DNAbin(louse), model = "JC69")
gopher.JC <- dist.dna(as.DNAbin(gopher), model = "JC69")
l <- nj(louse.JC)
g <- nj(gopher.JC)
```

Now we have an estimation for *illustrative* purposes of the tree topology for the parasite and their hosts. We want to plot the two trees face to face, and for this we must change R graphical parameters. The first thing to do is to save the current graphical parameter settings so as to be able to restore them later:

```
op <- par(no.readonly = TRUE)
```

The meaning of the `no.readonly = TRUE` option here is that graphical parameters are not all settable, we just want to save those we can change at will. Now, we can play with graphics :

```
g$tip.label <- paste(1:8, g.names)
l$tip.label <- paste(1:8, l.names)
layout(matrix(data = 1:2, nrow = 1, ncol = 2), width=c(1.4, 1))
par(mar=c(2,1,2,1))
plot(g, adj = 0.8, cex = 1.4, use.edge.length=FALSE,
  main = "gopher (host)", cex.main = 2)
plot(l,direction="l", use.edge.length=FALSE, cex = 1.4,
  main = "louse (parasite)", cex.main = 2)
```



We now restore the old graphical settings that were previously saved:

```
par(op)
```

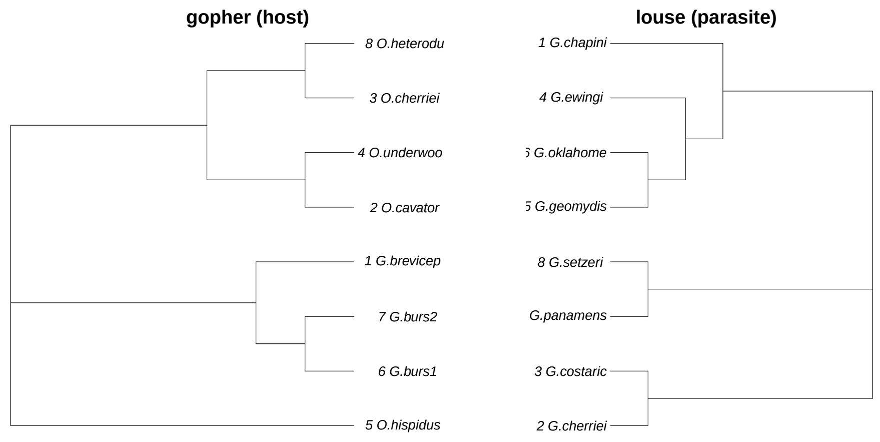
OK, this may look a little bit obscure if you are not fluent in programming, but please try the following experiment. In your current working directory, that is in the directory given by the `getwd()` command, create a text file called `essai.r` with your favourite text editor, and copy/paste the previous R commands, that is :

```
louse <- read.alignment(system.file("sequences/louse.fasta", package = "seqinr"), format = "fasta")
gopher <- read.alignment(system.file("sequences/gopher.fasta", package = "seqinr"), format = "fasta")
l.names <- readLines(system.file("sequences/louse.names", package = "seqinr"))
g.names <- readLines(system.file("sequences/gopher.names", package = "seqinr"))
library(ape)
louse.JC <- dist.dna(as.DNAbin(louse), model = "JC69")
gopher.JC <- dist.dna(as.DNAbin(gopher), model = "JC69")
l <- nj(louse.JC)
g <- nj(gopher.JC)
g$tip.label <- paste(1:8, g.names)
l$tip.label <- paste(1:8, l.names)
layout(matrix(data = 1:2, nrow = 1, ncol = 2), width=c(1.4, 1))
par(mar=c(2,1,2,1))
plot(g, adj = 0.8, cex = 1.4, use.edge.length=FALSE,
  main = "gopher (host)", cex.main = 2)
plot(l,direction="l", use.edge.length=FALSE, cex = 1.4,
  main = "louse (parasite)", cex.main = 2)
```

Make sure that your text has been saved and then go back to R console to enter the command :

```
source("essai.r")
```

This should reproduce the previous face-to-face phylogenetic trees in your R graphical device. Now, your boss is unhappy with working with the Jukes and Cantor's model [40] and wants you to use the Kimura's 2-parameters distance [45] instead. Go back to the text editor to change `model = "JC69"` by `model = "K80"`, save the file, and in the R console `source("essai.r")` again, you should obtain the following graph :



Now, something even worst, there was a error in the aligned sequence set: the first base in the first sequence in the file `louse.fasta` is not a C but a T. To locate the file on your system, enter the following command:

```
system.file("sequences/louse.fasta", package = "seqinr")
[1] "/Users/lobry/seqinr/pkg.Rcheck/seqinr/sequences/louse.fasta"
```

Open the `louse.fasta` file in your text editor, fix the error, go back to the R console to `source("essai.r")` again. That's all, your graph is now consistent with the updated dataset.

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Thu Jun 2 15:58:57 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 4

Importing sequences from ACNUC databases

Charif, D. Lobry, J.R.

Introduction

As a rule of thumb, after compression one nucleotide needs one octet of disk space storage (because you need also the annotations corresponding to the sequences), so that most likely you won't have enough space on your computer to work with a local copy of a complete DNA database. The idea is to import under R only the subset of sequences you are interested in. This is done in three steps:

1. Choose the bank you want to work with.
2. Select the sequences you are interested in.
3. Retrieve sequences from server into your workspace.

We now give a full example of those three steps under the ACNUC system [22, 23, 30, 28, 29].

4.1 Choose a bank

Select the database from which you want to extract sequences with the `choosebank()` function. This function initiates a remote access to an ACNUC database. Called without arguments, `choosebank()` returns the list of available databases:

```
choosebank()
```

```
 [1] "genbank"       "embl"          "emblwgs"       "swissprot"
 [5] "ensembl"       "hogenom"       "hogenomdna"    "hovergendna"
 [9] "hovergen"      "hogenom5"      "hogenom5dna"   "hogenom4"
[13] "hogenom4dna"   "homolens"      "homolensdna"   "hobacnucl"
```

```
[17] "hobacprot"       "phever2"         "phever2dna"      "refseq"
[21] "greviews"        "bacterial"       "archaeal"        "protozoan"
[25] "ensprotists"     "ensfungi"        "ensmetazoa"      "ensplants"
[29] "ensemblbacteria" "mito"            "polymorphix"     "emglib"
[33] "refseqViruses"   "ribodb"          "taxodb"
```

Biological sequence databases are fast moving targets, and for publication purposes it is recommended to specify on which release you were working on when you made the job. To get more informations about available databases on the server, just set the `infobank` parameter to `TRUE`. For instance, here is the result for the three first databases on the default server at the compilation time (June 2, 2016) of this document:

```
 choosebank(infobank = TRUE)[1:3, ]

     bank status
1 genbank     on
2    embl     on
3 emblwgs     on
                                                                      info
1              GenBank Release 213 (15 April 2016) Last Updated: May 22, 2016
2 EMBL Nucleotide Archive Release 127 (March 2016) Last Updated: May 21, 2016
3                EMBL Whole Genome Shotgun sequences Release 127 (March 2016)
```

Note that there is a `status` column because a database could be unavailable for a while during updates. If you try call `choosebank(bank = "bankname")` when the bank called `bankname` is off from server, you will get an explicit error message stating that this bank is temporarily unavailable, for instance:

```
 res <- try(choosebank("off"))
 cat(res)

Error in acnucopen(bank, socket) :
  Database with name -->off<-- is currently off for maintenance, please try again later.
```

Some special purpose databases are not listed by default. These are *tagged* databases that are only listed if you provide an explicit `tagbank` argument to the `choosebank()` function. Of special interest for teaching purposes is the `TP` tag, an acronym for *Travaux Pratiques* which means "practicals", and corresponds to *frozen* databases so that you can set up a practical whose results are stable from year to year. Currently available frozen databases at the default server are:

```
 choosebank(tagbank = "TP", infobank = TRUE)

     bank status                    info
1 trypano     on frozen trypano database
2  emblTP     on     frozen EMBL release
```

Now, if you want to work with a given database, say GenBank, just call `choosebank()` with `"genbank"` as its first argument:

```
 mybank <- choosebank("genbank")
 str(mybank)

List of 9
 $ socket  :Classes 'sockconn', 'connection'  atomic [1:1] 5
  .. ..- attr(*, "conn_id")=<externalptr>
 $ bankname: chr "genbank"
 $ banktype: chr "GENBANK"
 $ totseqs : num 2.26e+08
 $ totspecs: num 1585615
 $ totkeys : num 69548858
 $ release : chr "         GenBank Release 213 (15 April 2016) Last Updated: May 22, 2016"
 $ status  :Class 'AsIs'  chr "on"
 $ details : chr [1:4] "              ****     ACNUC Data Base Content      ****                         "
```

```
closebank()
```

The components of `mybank` means that in the database called `genbank` at the compilation time of this document there were `225,750,303` sequences from `1,585,615` species and a total of `69,548,858` keywords. The status of the bank was `on`, and the release information was `GenBank Release 213 (15 April 2016) Last Updated: May 22, 2016`. For specialized databases, some relevant informations are also given in the `details` component.

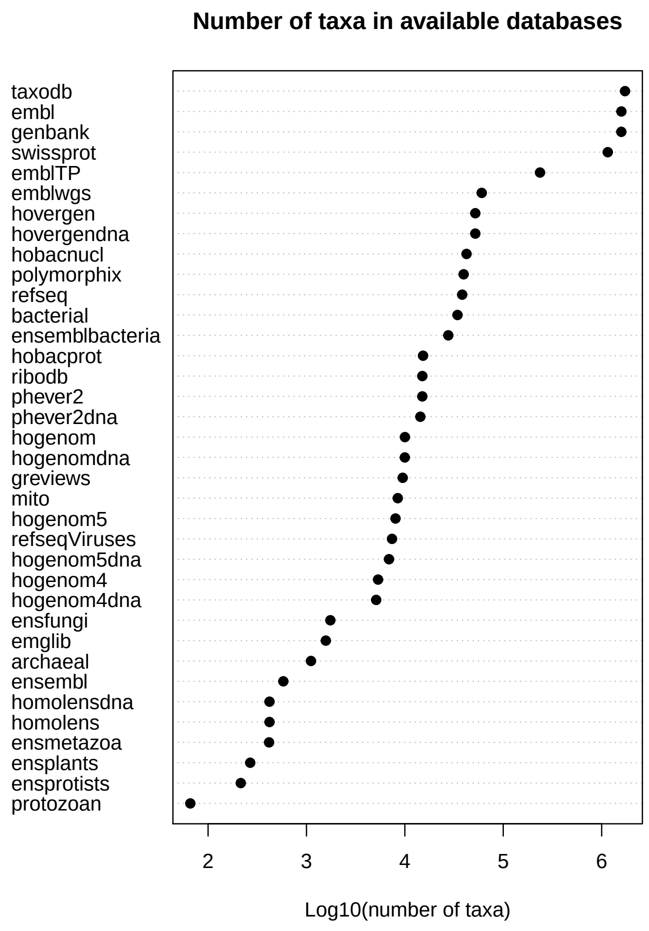
As from **seqinR** 1.0-3, the result of the `choosebank()` function is automatically stored in a variable named `banknameSocket` in the `.seqinrEnv` environment, so that if no socket argument is given to the `query()` function, the last opened database will be used by default for your requests. This is just a matter of convenience so that you don't have to explicitly specify the details of the socket connection when working with the last opened database. You have, however, full control of the process since `choosebank()` returns (invisibly) all the required details. There is no trouble to open *simultaneously* many databases. You are just limited by the number of simultaneous connections your build of R is allowed[1].

For advanced users who may wish to access to more than one database at time, a good advice is to close them with the function `closebank()` as soon as possible so that the maximum number of simultaneous connections is never reached. In the example below, we want to display the number of taxa (*i.e.* the number of nodes) in the species taxonomy associated with each available database (including frozen databases). For this, we loop over available databases and close them as soon as the information has been retrieved.

```
banks <- c(choosebank(), choosebank(tagbank = "TP"))
nbanks <- length(banks)
ntaxa <- numeric(nbanks) # pre-allocate
for(i in seq_len(nbanks)){
  bkopenres <- try(choosebank(banks[i]))
  if(inherits(bkopenres, "try-error")){
    ntaxa[i] <- NA
  } else {
    ntaxa[i] <- as.numeric(bkopenres$totspecs)
    closebank()
  }
}
names(ntaxa) <- banks
```

```
ntaxa <- ntaxa[!is.na(ntaxa)]
dotchart(log10(ntaxa[order(ntaxa)]), pch = 19,
main = "Number of taxa in available databases",
xlab = "Log10(number of taxa)")
```

[1] As from R 2.4.0 he maximum number of open connections has been increased from 50 to 128. Note also that there is a very convenient function called `closeAllConnections()` in the R base package if you want to close all open connections at once.



4.2 Make your query

For this section, set up the default bank to GenBank, so that you don't have to provide the sockets details for the `query()` function. We set the `verbose` argument to `TRUE`, just for the fun[2], this is not really usefull here:

```
 choosebank("genbank", verbose = TRUE)
```

```
Verbose mode is on, parameter values are:
  bank =  "genbank"
  host =  "pbil.univ-lyon1.fr"
  port =  5558
  timeout =  5 seconds
  infobank =  FALSE
  tagbank =  NA
I'm ckecking that sockets are available on this build of R...
... yes, sockets are available on this build of R.
I'm trying to open the socket connection...
... yes, I was able to open the socket connection.
I'm trying to read answer from server...
... answer from server is: OK acnuc socket started
clientid(): sending clientid&id=seqinr_3.0-11
... answer from server is: code=0
```

[2]This option is however usefull for trouble shooting.

```
parser.socket received: -->code=0<--
I'm trying to open the bank from server...
... and everything is OK up to now.
I'm trying to get information on the bank...
... and everything is OK up to now.
```

Felis catus. Source: wikipedia

Then, you have to say what you want, that is to compose a query to select the subset of sequences you are interested in. The way to do this is documented under `?query`, we just give here a simple example (more details are given in chapter 5 page 61). In the query below, we want to select all the coding sequences (`t=cds`) from cat (`AND sp=felis catus`) that are not (`AND NOT`) partial sequences (`k=partial`). We want the result to be stored in an object called `completeCatsCDS`.

```
completeCatsCDS <- query("completeCatsCDS", "sp=felis catus AND t=cds AND NOT k=partial")
```

Now, there is in the workspace an object called `completeCatsCDS`, which does not contain the sequences themselves but the *sequence names* (and various relevant informations such as the genetic code and the frame) that fit the query. They are stored in the `req` component of the object, let's see the name of the first ten of them:

```
getName(completeCatsCDS$req[1:10])
```

```
[1] "AB000483.PE1"    "AB000484.PE1"    "AB000485.PE1"    "AB003366"
[5] "AB003367"        "AB004237"        "AB004238"        "AB009279.PE1"
[9] "AB009280.PE1"    "AB010872.UGT1A1"
```

The first sequence that fit our request is `AB000483.PE1`, the second one is `AB000484.PE1`, and so on. Note that the sequence name may have an extension, this corresponds to *subsequences*, a specificity of the ACNUC system that allows to handle easily a subsequence with a biological meaning, typically a gene. The list of available subsequences in a given database is given by the function `getType()`, for example the list of available subsequences in GenBank is given in table 4.1.

	Type	Description
1	CDS	.PE protein coding region
2	LOCUS	sequenced DNA fragment
3	MISC_RNA	.RN other structural RNA coding region
4	NCRNA	.NC non-protein-coding gene other than rRNA and tRNA
5	RRNA	.RR mature ribosomal RNA
6	TMRNA	.TM transfer messenger RNA
7	TRNA	.TR mature transfer RNA

Table 4.1: Available subsequences in GenBank Release 213 (15 April 2016) Last Updated: May 22, 2016

The component `call` of `completeCatsCDS` keeps automatically a trace of the way you have selected the sequences:

```
completeCatsCDS$call
```

```
query(listname = "completeCatsCDS", query = "sp=felis catus AND t=cds AND NOT k=partial")
```

At this stage you can quit your R session saving the workspace image. The next time an R session is opened with the workspace image restored, there will be an object called `completeCatsCDS`, and looking into its `call` component will tell you that it contains the names of complete coding sequences from *Felis catus*.

In practice, queries for sequences are rarely done in one step and are more likely to be the result of an iterative, progressively refining, process. An important point is that a list of sequences can be re-used. For instance, we can re-use `completeCatsCDS` to get only the list of sequences that were published in 2004:

```
ccc2004 <- query("ccc2004", "completeCatsCDS AND y=2004")
length(ccc2004$req)
```

```
[1] 60
```

```
ccc2004$nelem
```

```
[1] 60
```

Hence, there were 60 complete coding sequences published in 2004 for *Felis catus* in GenBank.

As from release 1.0-3 of the **seqinR** package, there is new parameter `virtual` which allows to disable the automatic retrieval of information for all list elements. This is interesting for list with many elements, for instance :

```
allcds <- query("allcds", "t=cds", virtual = TRUE)
allcds$nelem
```

```
[1] 34260201
```

There are therefore `34,260,201` coding sequences in this version of GenBank. It would be long to get all the informations for the elements of this list, so we have set the parameter `virtual` to `TRUE` and the `req` component of the list has not been documented:

```
allcds$req
```

```
[1] NA
```

However, the list can still be re-used[3], for instance we may extract from this list all the sequences from, say, *Mycoplasma genitalium*:

```
small <- query("small", "allcds AND sp=mycoplasma genitalium", virtual = TRUE)
small$nelem
```

```
[1] 3382
```

There are then `3,382` elements in the list `small`, so that we can safely repeat the previous query without asking for a virtual list:

```
small <- query("small", "allcds AND sp=mycoplasma genitalium")
getName(small$req[1:10])
```

```
 [1] "AB919117" "AB919118" "AB919119" "AB919120" "AB919121" "AY191424"
 [7] "AY386807" "AY386808" "AY386809" "AY386810"
```

Here are some illustrations of using virtual list to answer simple questions about the current GenBank release.

Man. How many sequences are available for our species?

```
man <- query("man","sp=homo sapiens",virtual=T)
man$nelem
```

[3]of course, as long as the socket connection with the server has not been lost: virtual lists details are only known by the server.

```
[1] 23519997
```

There are 23,519,997 sequences from *Homo sapiens*.

Sex. How many sequences are annotated with a keyword starting by sex?

```
sex <- query("sex","k=sex@",virtual=T)
sex$nelem
```

```
[1] 3577
```

There are 3,577 such sequences.

tRNA. How many complete tRNA sequences are available?

```
trna <- query("trna","t=trna AND NOT k=partial",virtual=T)
trna$nelem
```

```
[1] 1810833
```

There are 1,810,833 complete tRNA sequences.

Nature vs. Science. In which journal were the more sequences published?

```
nature <- query("nature","j=nature",virtual=T)
nature$nelem
```

```
[1] 2645373
```

```
science <- query("science","j=science",virtual=T)
science$nelem
```

```
[1] 2244003
```

There are 2,645,373 sequences published in *Nature* and 2,244,003 sequences published in *Science*, so that the winner is *Nature*.

Smith. How many sequences have Smith (last name) as author?

```
smith <- query("smith","au=smith",virtual=T)
smith$nelem
```

```
[1] 6433901
```

There are 6,433,901 such sequences.

YK2. How many sequences were published after year 2000 (included)?

```
yk2 <- query("yk2","y>2000",virtual=T)
yk2$nelem
```

```
[1] 182398606
```

There are 182,398,606 sequences published after year 2000.

Organelle contest. Do we have more sequences from chloroplast genomes or from mitochondion genomes?

```
chloro <- query("chloro","o=chloroplast",virtual=T)
chloro$nelem
```

```
[1] 870722
```

```
mito <- query("mito","o=mitochondrion",virtual=T)
mito$nelem
```

```
[1] 3479548
```

There are `870,722` sequences from chloroplast genomes and `3,479,548` sequences from mitochondrion genomes, so that the winner is mitochondrion.

```
closebank()
```

4.3 Extract sequences of interest

4.3.1 Introduction

For this section we set up the bank to `emblTP` which is a frozen subset of EMBL database to allow for the reproducibility of results.

```
choosebank("emblTP")
```

We suppose that the sequences we are interested in are all the complete coding sequences from *Felis catus* :

```
completeCatsCDS <- query("completeCatsCDS", "sp=felis catus AND t=cds AND NOT k=partial")
(nseq <- completeCatsCDS$nelem)
```

```
[1] 257
```

Thus, there were 257 complete CDS from *Felis catus* in this release of EMBL.

The sequences are obtained with the function `getSequence()`. For example, the first 50 nucleotides of the first sequence of our request are:

```
mysect <- getSequence(completeCatsCDS$req[[1]])
myseq[1:50]
```

```
 [1] "a" "t" "g" "a" "c" "c" "a" "a" "c" "a" "t" "t" "c" "g" "a" "a" "a" "a"
[19] "t" "c" "a" "c" "a" "c" "c" "c" "c" "c" "t" "t" "a" "c" "c" "a" "a" "a"
[37] "a" "t" "t" "a" "t" "t" "a" "a" "t" "c" "a" "c" "t" "c"
```

They can also be coerced as string of character with the function `c2s()`:

```
c2s(myseq[1:50])
```

```
[1] "atgaccaacattcgaaaatcacaccccccttaccaaaattattaatcactc"
```

We can also use the argument `as.string` to retrive sequences directly as strings:

```
substr(getSequence(completeCatsCDS$req[[1]], as.string = TRUE), 1, 50)
```

```
[1] "atgaccaacattcgaaaatcacaccccccttaccaaaattattaatcactc"
```

Note that what is done by `getSequence()` is much more complex than a simple substring extraction because subsequences of biological interest are not necessarily contiguous, nor on the same DNA strand, nor even from the same entry.

4.3.2 Extracting sequences with trans-splicing

Consider for instance the following coding sequence from sequence AE003734:

```
 trs <- query("trs","N=AE003734.PE35")
 getAnnot(trs$req[[1]]) -> annots
 cat(annots, sep="\n")
```

```
FT   CDS             join(complement(153944..154157),complement(153727..153866),
FT                   complement(152185..153037),138523..138735,138795..138955)
FT                   /codon_start=1
FT                   /db_xref="FLYBASE:FBgn0002781"
FT                   /db_xref="GOA:Q86B86"
FT                   /db_xref="TrEMBL:Q86B86"
FT                   /note="mod(mdg4) gene product from transcript CG32491-RZ;
FT                   trans splicing"
FT                   /gene="mod(mdg4)"
FT                   /product="CG32491-PZ"
FT                   /locus_tag="CG32491"
FT                   /protein_id="AAO41581.1"
FT                   /translation="MADDEQFSLCWNNFNTNLSAGFHESLCRGDLVDVSLAAEGQIVKA
FT                   HRLVLSVCSPFFRKMFTQMPSNTHAIVFLNNVSHSALKDLIQFMYCGEVNVKQDALPAF
FT                   ISTAESLQIKGLTDNDPAPQPPQESSPPPAAPHVQQQQIPAQRVQRQQPRASARYKIET
FT                   VDDGLGDEKQSTTQIVIQTTAAPQATIVQQQQPQQAAQQIQSQQLQTGTTTTATLVSTN
FT                   KRSAQRSSLTPASSSAGVKRSKTSTSANVMDPLDSTTETGATTTAQLVPQQITVQTSVV
FT                   SAAEAKLHQQSPQQVRQEEAEYIDLPMELPTKSEPDYSEDHGDAAGDAEGTYVEDDTYG
FT                   DMRYDDSYFTENEDAGNQTAANTSGGGVTATTSKAVVKQQSQNYSESSFVDTSGDQGNT
FT                   EAQVTQHVRNCGPQMFLISRKGGTLLTINNFVYRSNLKFFGKSNNILYWECVQNRSVKC
FT                   RSRLKTIGDDLYVTNDVHNHMGDNKRIEAAKAAGMLIHKKLSSLTAADKIQGSWKMDTE
FT                   GNPDHLPKM"
```

To get the coding sequence manually you would have join 5 different pieces from AE003734 and some of them are in the complementary strand. With getSequence() you don't have to think about this. Just make a query with the sequence name:

```
 transspliced <- query("transspliced", "N=AE003734.PE35")
 length(transspliced$req)
```

```
[1] 1
```

```
 getName(transspliced$req[[1]])
```

```
[1] "AE003734.PE35"
```

Ok, now there is in your workspace an object called transspliced which req component is of length one (because you have asked for just one sequence) and the name of the single element of the req component is AE003734.PE35 (because this is the name of the sequence you wanted). Let see the first 50 base of this sequence:

```
 getSequence(transspliced$req[[1]])[1:50]
```

```
 [1] "a" "t" "g" "g" "c" "g" "g" "a" "c" "g" "a" "c" "g" "a" "g" "c" "a" "a"
[19] "t" "t" "c" "a" "g" "c" "t" "t" "g" "t" "g" "c" "t" "g" "g" "a" "a" "c"
[37] "a" "a" "c" "t" "t" "c" "a" "a" "c" "a" "c" "g" "a" "a"
```

All the complex trans-splicing operations have been done here. You can check that there is no in-frame stop codons[4] with the getTrans() function to translate this coding sequence into protein:

```
 getTrans(transspliced$req[[1]])[1:50]
```

```
 [1] "M" "A" "D" "D" "E" "Q" "F" "S" "L" "C" "W" "N" "N" "F" "N" "T" "N" "L"
[19] "S" "A" "G" "F" "H" "E" "S" "L" "C" "R" "G" "D" "L" "V" "D" "V" "S" "L"
[37] "A" "A" "E" "G" "Q" "I" "V" "K" "A" "H" "R" "L" "V" "L"
```

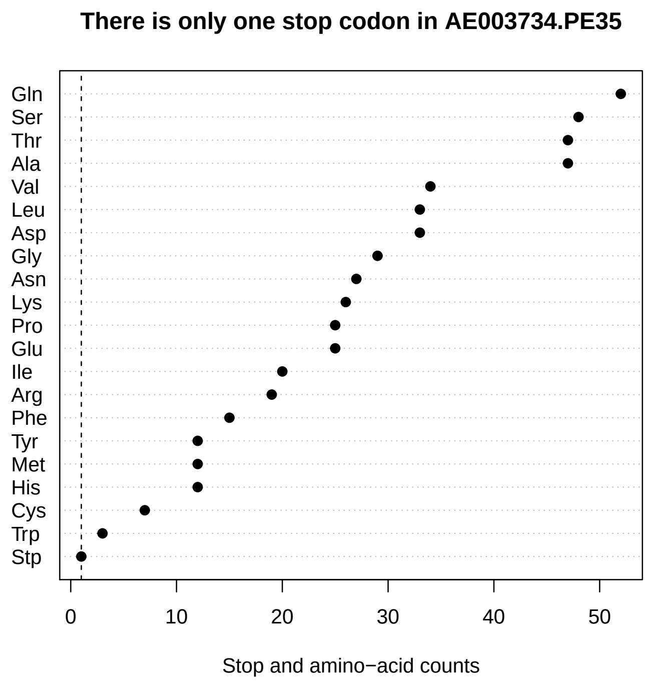
```
 table(getTrans(transspliced$req[[1]]))
```

[4]Stop codons are represented by the character * when translated into protein.

```
*  A  C  D  E  F  G  H  I  K  L  M  N  P  Q  R  S  T  V  W  Y
1 47  7 33 25 15 29 12 20 26 33 12 27 25 52 19 48 47 34  3 12
```

In a more graphical way:

```
aacount <- table(getTrans(transspliced$req[[1]]))
aacount <- aacount[order(aacount)]
names(aacount) <- aaa(names(aacount))
dotchart(aacount, pch = 19, xlab = "Stop and amino-acid counts",
main = "There is only one stop codon in AE003734.PE35")
abline(v=1, lty = 2)
```



Note that the relevant variant of the genetic code was automatically set up during the translation of the sequence into protein. This is because the `transspliced$req[[1]]` object belongs to the `SeqAcnucWeb` class:

```
class(transspliced$req[[1]])
```

```
[1] "SeqAcnucWeb"
```

Therefore, when you are using the `getTrans()` function, you are automatically redirected to the `getTrans.SeqAcnucWeb()` function which knows how to take into account the relevant frame and genetic code for your coding sequence.

4.3.3 Extracting sequences from many entries

Consider the following CDS from `M19233`:

```
multi <- query("multi", "AC=M19233 AND T=CDS")
cat(getAnnot(multi$req[[1]]), sep = "\n")
```

```
FT   CDS             join(M17883.1:988..1155,M17883.1:1504..1650,
FT                   M17883.1:2451..2648,M17883.1:3098..3328,625..758)
FT                   /codon_start=1
FT                   /db_xref="GOA:Q13763"
FT                   /db_xref="TrEMBL:Q13763"
FT                   /partial
FT                   /gene="AMY1A"
FT                   /product="alpha-amylase"
FT                   /protein_id="AAA57345.1"
FT                   /translation="MKLFWLLFTIGFCWAQYSSNTQQGRTSIVHLFEWRWVDIALECER
FT                   YLAPKGFGGVQVSPPNENVAIHNPFRPWWERYQPVSYKLCTRSGNEDEFRNMVTRCNNV
FT                   GVRIYVDAVINHMCGNAVSAGTSSTCGSYFNPGSRDFPAVPYSGWDFNDGKCKTGSGDI
FT                   ENYNDATQVRDCRLSGLLDPALGKDYVRSKIAEYMNHLIDIGVAGFRIDASKHMWPGDI
FT                   KAILDKLHNLNSNWFPEGSKPFIYQEVIDLGGEPIKSSDYFGNGRVTEFKYGAKLGTVI
FT                   RKWTGEKMSYL"
```

The CDS here is obtained by joining pieces from different entries, but this is not a problem:

```
 getTrans(multi$req[[1]])
```

```
  [1] "M" "K" "L" "F" "W" "L" "L" "F" "T" "I" "G" "F" "C" "W" "A" "Q" "Y" "S"
 [19] "S" "N" "T" "Q" "Q" "G" "R" "T" "S" "I" "V" "H" "L" "F" "E" "W" "R" "W"
 [37] "V" "D" "I" "A" "L" "E" "C" "E" "R" "Y" "L" "A" "P" "K" "G" "F" "G" "G"
 [55] "V" "Q" "V" "S" "P" "P" "N" "E" "N" "V" "A" "I" "H" "N" "P" "F" "R" "P"
 [73] "W" "W" "E" "R" "Y" "Q" "P" "V" "S" "Y" "K" "L" "C" "T" "R" "S" "G" "N"
 [91] "E" "D" "E" "F" "R" "N" "M" "V" "T" "R" "C" "N" "N" "V" "G" "V" "R" "I"
[109] "Y" "V" "D" "A" "V" "I" "N" "H" "M" "C" "G" "N" "A" "V" "S" "A" "G" "T"
[127] "S" "S" "T" "C" "G" "S" "Y" "F" "N" "P" "G" "S" "R" "D" "F" "P" "A" "V"
[145] "P" "Y" "S" "G" "W" "D" "F" "N" "D" "G" "K" "C" "K" "T" "G" "S" "G" "D"
[163] "I" "E" "N" "Y" "N" "D" "A" "T" "Q" "V" "R" "D" "C" "R" "L" "S" "G" "L"
[181] "L" "D" "P" "A" "L" "G" "K" "D" "Y" "V" "R" "S" "K" "I" "A" "E" "Y" "M"
[199] "N" "H" "L" "I" "D" "I" "G" "V" "A" "G" "F" "R" "I" "D" "A" "S" "K" "H"
[217] "M" "W" "P" "G" "D" "I" "K" "A" "I" "L" "D" "K" "L" "H" "N" "L" "N" "S"
[235] "N" "W" "F" "P" "E" "G" "S" "K" "P" "F" "I" "Y" "Q" "E" "V" "I" "D" "L"
[253] "G" "G" "E" "P" "I" "K" "S" "S" "D" "Y" "F" "G" "N" "G" "R" "V" "T" "E"
[271] "F" "K" "Y" "G" "A" "K" "L" "G" "T" "V" "I" "R" "K" "W" "T" "G" "E" "K"
[289] "M" "S" "Y" "L"
```

```
 table(aaa(getTrans(multi$req[[1]])))
```

```
Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser Thr Trp Tyr
 15  16  19  17   8   7  14  28   6  17  18  16   6  15  14  21  12  10  14
Val
 19
```

There is no stop codon here because the sequence is partial. If you are experiencing a strong closure issue here, just close the bank:

```
 closebank()
```

Feeling better now ?

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.0-11, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 14:33:37 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 5

The query language

Lobry, J.R.

5.1 Where to find information

The last version of the documentation for the query language is available online at http://pbil.univ-lyon1.fr/databases/acnuc/cfonctions.html#QUERYLANGUAGE. This documentation has been imported within the documentation of the `query()` function, but the last available update is the online version. The query language is a specificity of the ACNUC system [30, 28, 29, 27].

5.2 Case sensitivity and ambiguities resolution

The query language is case insensitive, for instance:

```
choosebank("emblTP")
lowercase <- query("lowercase", "sp=escherichia coli", virtual = TRUE)
uppercase <- query("uppercase", "SP=Escherichia coli", virtual = TRUE)
lowercase$nelem == uppercase$nelem
```

```
[1] TRUE
```

```
closebank()
```

Three operators (AND, OR, NOT) can be ambiguous because they can also occur within valid criterion values. Such ambiguities can be solved by encapsulating elementary selection criteria between escaped double quotes. For example:

```
choosebank("emblTP")
ambig <- query("ambig","\"sp=Beak and feather disease virus\" AND \"au=ritchie\"", virtual = T)
ambig$nelem
```

```
[1] 18
```

```
closebank()
```

5.3 Selection criteria

5.3.1 Introduction

Selection criteria are in the form `c=something` (without space before the = sign) or `list_name` where `list_name` is a previously constructed list.

5.3.2 `SP=taxon`

This is used to select sequences attached to a given taxon or any other below in the tree. The at sign @ substitutes as a wildcard character for any zero or more characters. Here are some examples:

```
choosebank("emblTP")
bb <- query("bb","sp=Borrelia burgdorferi",virtual=T)
bb$nelem
[1] 1682
borrelia <- query("borrelia","sp=Borrelia",virtual=T)
borrelia$nelem
[1] 3173
closebank()
```

Here is an example of use of the wildcard @ to look for sapiens species:

```
choosebank("emblTP")
sapiens <- query("sapiens","sp=@sapiens@",virtual=T)
sapiens$nelem
[1] 2216556
sapienspecies <- query("sapienspecies","PS sapiens")
getName(sapienspecies)
 [1] "HOMO SAPIENS"
 [2] "HOMO SAPIENS NEANDERTHALENSIS"
 [3] "HOMO SAPIENS X HUMAN PAPILLOMAVIRUS TYPE"
 [4] "HOMO SAPIENS X SIMIAN VIRUS 40"
 [5] "HOMO SAPIENS X HUMAN ENDOGENOUS RETROVIR"
 [6] "HOMO SAPIENS X HUMAN T-CELL LYMPHOTROPIC"
 [7] "HEPATITIS B VIRUS X HOMO SAPIENS"
 [8] "HOMO SAPIENS X HEPATITIS B VIRUS"
 [9] "HOMO SAPIENS X HUMAN IMMUNODEFICIENCY VI"
[10] "SYNTHETIC CONSTRUCT X HOMO SAPIENS"
[11] "HUMAN PAPILLOMAVIRUS X HOMO SAPIENS"
[12] "MUS SP. X HOMO SAPIENS"
[13] "HOMO SAPIENS X HUMAN PAPILLOMAVIRUS"
[14] "HOMO SAPIENS X HUMAN ADENOVIRUS TYPE 5"
[15] "HOMO SAPIENS X HERV-H/ENV62"
[16] "HOMO SAPIENS X HERV-H/ENV60"
[17] "HOMO SAPIENS X HERV-H/ENV59"
[18] "EXPRESSION VECTOR PTH-HIN X HOMO SAPIENS"
[19] "ADENO-ASSOCIATED VIRUS 2 X HOMO SAPIENS"
[20] "SIMIAN VIRUS 40 X HOMO SAPIENS"
[21] "HOMO SAPIENS X MUS MUSCULUS"
[22] "HOMO SAPIENS X INFLUENZA B VIRUS (B/LEE/"
[23] "MUS MUSCULUS X HOMO SAPIENS"
[24] "CRICETULUS GRISEUS X HOMO SAPIENS"
[25] "TRYPANOSOMA CRUZI X HOMO SAPIENS"
[26] "HOMO SAPIENS X TRYPANOSOMA CRUZI"
closebank()
```

Homo neanderthalensis. Source: wikipedia

5.3.3 `TID=id`

This is used to select sequences attached attached to a given numerical NCBI's taxonomy ID. For instance, the taxonomy ID for *Homo sapiens neanderthalensis* is 63221:

```
choosebank("genbank")
hsn <- query("hsn","TID=63221", virtual=T)
hsn$nelem
```

```
[1] 1358
```

```
hsnsp <- query("hsnsp","PS hsn")
getName(hsnsp)
```

```
[1] "HOMO SAPIENS NEANDERTHALENSIS"
```

```
closebank()
```

5.3.4 K=keyword

This is used to select sequences attached to a given keyword or any other below in the tree. The at sign @ substitutes as a wildcard character for any zero or more characters. Example:

```
choosebank("emblTP")
ecoliribprot <- query("ecoliribprot","sp=escherichia coli AND k=rib@ prot@", virtual=T)
ecoliribprot$nelem
```

```
[1] 105
```

```
closebank()
```

5.3.5 T=type

This is used to select sequences of specified type. The list of available type for the currently opened database is given by function `getType()`:

```
choosebank("emblTP")
getType()
```

```
        sname                               libel
2661      CDS              .PE protein coding region
2662       ID                            Locus entry
2663 MISC_RNA .RN other structural RNA coding region
2664     RRNA           .RR Ribosomal RNA coding gene
2665    SCRNA              .SC small cytoplasmic RNA
2666    SNRNA                  .SN small nuclear RNA
2667     TRNA            .TR Transfer RNA coding gene
```

```
closebank()
```

For instance, to select all coding sequences from *Homo sapiens* we can use:

```
choosebank("emblTP")
hscds <- query("hscds","sp=Homo sapiens AND t=cds", virtual=T)
hscds$nelem
```

```
[1] 150513
```

```
closebank()
```

5.3.6 J=journal_name

This is used to select sequences published in journal specified using defined journal code. For instance to select all sequences published in *Science*:

```
choosebank("emblTP")
allseqsfromscience <- query("allseqsfromscience","J=Science", virtual=TRUE)
allseqsfromscience$nelem
```

```
[1] 930397
```

```
closebank()
```

The list of available journal code can be obtained from the `readsmj()` function this way:

```
choosebank("emblTP")
nl <- readfirstrec(type = "SMJ")
smj <- readsmj(nl = nl, all.add = TRUE)
head(smj[!is.na(smj$nature) & smj$nature == "journal", c("sname","libel")])
```

```
                sname                                             libel
21                ABP                                  Acta Biochim. Pol.
22  ABSTR-SOCNEUROSCI                             Abstr. - Soc. Neurosci.
23 ABSTRGENMEETAMSOCM                  Abstr. Gen. Meet. Am. Soc. Microbiol.
24 ABSTRMIDWINTERRESM Abstr. Midwinter Res. Meet. Assoc. Res. Otolaryngol.
25 ACTAAGRICSCANDAANI                        Acta Agric. Scand. A Anim. Sci.
26 ACTABIOCHIMBIOPHYS                            Acta Biochim. Biophys. Sin.
```

```
closebank()
```

5.3.7 R=refcode

This is used to select sequences from a given bibliographical reference specified as `jcode/volume/page`. For instance, to select sequences associated with the first publication [1] of the complete genome of *Rickettsia prowazekii*, we can use:

```
choosebank("emblTP")
rpro <- query("rpro","R=Nature/396/133")
getName(rpro)
```

```
[1] "RPDNAOMPB" "RPXX01"    "RPXX02"    "RPXX03"    "RPXX04"
```

```
closebank()
```

5.3.8 AU=name

This is used to select sequences having a specified author (only last name, no initial).

```
choosebank("emblTP")
Graur <- query("Graur","AU=Graur")
Graur$nelem
```

```
[1] 48
```

```
closebank()
```

5.3.9 AC=accession_no

This is used to select sequences attached to specified accession number. For instance if we are looking for sequences attached to the accession number AY382159:

```
choosebank("emblTP")
ACexample <- query("ACexample","AC=AY382159")
getName(ACexample$req[[1]])
```

```
[1] "AY382159"
```

```
annotations <- getAnnot(ACexample$req[[1]])
cat(annotations, sep ="\n")
```

```
ID   AY382159   standard; genomic DNA; PRO; 783 BP.
XX
AC   AY382159;
XX
SV   AY382159.1
XX
DT   08-OCT-2003 (Rel. 77, Created)
DT   08-OCT-2003 (Rel. 77, Last updated, Version 1)
XX
DE   Borrelia burgdorferi strain FP1 OspA gene, partial cds.
```

```
XX
KW   .
XX
OS   Borrelia burgdorferi (Lyme disease spirochete)
OC   Bacteria; Spirochaetes; Spirochaetales; Spirochaetaceae; Borrelia;
OC   Borrelia burgdorferi group.
XX
RN   [1]
RP   1-783
RA   Hao Q., Wan K.;
RT   ;
RL   Submitted (03-SEP-2003) to the EMBL/GenBank/DDBJ databases.
RL   Department of Lyme Spirochetosis, CDC, Beijing 102206, China
XX
FH   Key             Location/Qualifiers
FH
FT   source          1..783
FT                   /db_xref="taxon:139"
FT                   /mol_type="genomic DNA"
FT                   /organism="Borrelia burgdorferi"
FT                   /strain="FP1"
FT   CDS             <1..>783
FT                   /codon_start=1
FT                   /transl_table=11
FT                   /product="OspA"
FT                   /protein_id="AAQ89576.1"
FT                   /translation="ALIACKQNVSSLDEKNSASVDLPGEMKVLVSKEKDKDGKYSLKAT
FT                   VDKLELKGTSDKNNGSGTLEGEKTDKSKAKLTISDDLSKTTFEVFKEDGKTLVSRKVSS
FT                   KDKTSTDEMFNEKGELSAKTMTRENGTKLEYTEMKSDGTGKTKEVLKNFTLEGRVANDK
FT                   VTLEVKEGTVTLSKEIAKSGEVTVALNDTNTTQATKKTGAWDSKTSTLTISVNSKKTTQ
FT                   LVFTKQDTITVQKYDSAGTNLEGTAVEIKTLDELKNALK"
XX
SQ   Sequence 783 BP; 342 A; 124 C; 145 G; 172 T; 0 other;
```

```
closebank()
```

5.3.10 N=seq_name

This is used to select sequences of a given name[1]. Sequences names are not necessarily stable, so that it's almost always better to work with accession numbers. Anyway, the distinction between sequence names and accession numbers is on a vanishing way because they tend more and more to be the same thing (as in the example just below). The use of the at sign @ to substitute as a wildcard character for any zero or more characters is possible here.

```
choosebank("emblTP")
Nexample <- query("Nexample","N=AY382159")
getName(Nexample$req[[1]])
```

```
[1] "AY382159"
```

```
annotations <- getAnnot(Nexample$req[[1]])
cat(annotations, sep ="\n")
```

```
ID   AY382159   standard; genomic DNA; PRO; 783 BP.
XX
AC   AY382159;
XX
SV   AY382159.1
XX
DT   08-OCT-2003 (Rel. 77, Created)
DT   08-OCT-2003 (Rel. 77, Last updated, Version 1)
XX
DE   Borrelia burgdorferi strain FP1 OspA gene, partial cds.
XX
KW   .
XX
OS   Borrelia burgdorferi (Lyme disease spirochete)
OC   Bacteria; Spirochaetes; Spirochaetales; Spirochaetaceae; Borrelia;
OC   Borrelia burgdorferi group.
XX
RN   [1]
RP   1-783
```

[1] *i.e.* what is documented in the ID or the LOCUS field

```
RA   Hao Q., Wan K.;
RT   ;
RL   Submitted (03-SEP-2003) to the EMBL/GenBank/DDBJ databases.
RL   Department of Lyme Spirochetosis, CDC, Beijing 102206, China
XX
FH   Key             Location/Qualifiers
FH
FT   source          1..783
FT                   /db_xref="taxon:139"
FT                   /mol_type="genomic DNA"
FT                   /organism="Borrelia burgdorferi"
FT                   /strain="FP1"
FT   CDS             <1..>783
FT                   /codon_start=1
FT                   /transl_table=11
FT                   /product="OspA"
FT                   /protein_id="AAQ89576.1"
FT                   /translation="ALIACKQNVSSLDEKNSASVDLPGEMKVLVSKEKDKDGKYSLKAT
FT                   VDKLELKGTSDKNNGSGTLEGEKTDKSKAKLTISDDLSKTTFEVFKEDGKTLVSRKVSS
FT                   KDKTSTDEMFNEKGELSAKTMTRENGTKLEYTEMKSDGTGKTKEVLKNFTLEGRVANDK
FT                   VTLEVKEGTVTLSKEIAKSGEVTVALNDTNTTQATKKTGAWDSKTSTLTISVNSKKTTQ
FT                   LVFTKQDTITVQKYDSAGTNLEGTAVEIKTLDELKNALK"
XX
SQ   Sequence 783 BP; 342 A; 124 C; 145 G; 172 T; 0 other;
```

```
 closebank()
```

5.3.11 NS=taxon_name

This is used to get the number of taxon of given name, with the use of the at sign @ to substitute as a wildcard character for any zero or more characters possible here. For instance, we want to know how many taxon have *sapiens* inside :

```
 choosebank("emblTP")
 NSexample <- query("NSexample","NS=@sapiens@")
 NSexample
26 SP for NS=@sapiens@
 closebank()
```

5.3.12 NK=keyword_name

This is used to get the number of keyword of given name, with the use of the at sign @ to substitute as a wildcard character for any zero or more characters possible here. For instance, we want to know how many keywords have *sex* inside :

```
 choosebank("emblTP")
 NKexample <- query("NKexample","NK=@sex@")
 NKexample
277 KW for NK=@sex@
 closebank()
```

5.3.13 Y=year or Y>year or Y<year

This is used to select sequences published in a given year (`Y=year`), or in a given year and after this year (`Y>year`), or in a given year and before this year (`Y<year`).

```
 choosebank("emblTP")
 Yexample <- query("Yexample","Y=1999", virtual=TRUE)
 Yexample$nelem
[1] 955274
 closebank()
```

5.3.14 O=organelle

This is used to select sequences from specified organelle named following defined code (*e.g.*, chloroplast). The list of available organelle codes can be obtained from the `readsmj()` function this way:

```
choosebank("genbank")
nl <- readfirstrec(type = "SMJ")
smj <- readsmj(nl = nl, all.add = TRUE)
smj[!is.na(smj$nature) & smj$nature == "organelle", c("sname","libel")]
```

```
             sname                    libel
5914   CHLOROPLAST       Chloroplast genome
5915 CHROMATOPHORE                     <NA>
5916 HYDROGENOSOME                     <NA>
5917 MITOCHONDRION     Mitochondrial genome
5918   NUCLEOMORPH       Nucleomorph genome
5919       PLASTID non-green plastid genome
```

```
closebank()
```

To select for instance all sequences from chloroplast genome we can use:

```
choosebank("emblTP")
Oexample <- query("Oexample","O=chloroplast", virtual=TRUE)
Oexample$nelem
```

```
[1] 65011
```

```
closebank()
```

5.3.15 M=molecule

This is used to select sequences according to the chemical nature of the sequenced molecule[2]. The list of available organelle code can be obtained from the `readsmj()` function this way:

```
choosebank("genbank")
nl <- readfirstrec(type = "SMJ")
smj <- readsmj(nl = nl, all.add = TRUE)
smj[!is.na(smj$nature) & smj$nature == "molecule", c("sname","libel")]
```

```
  sname                                libel
4  CRNA Sequenced molecule is complementary RNA
5   DNA               Sequenced molecule is DNA
6  MRNA              sequenced molecule is mRNA
7   RNA               Sequenced molecule is RNA
8  RRNA              sequenced molecule is rRNA
9  TRNA              sequenced molecule is tRNA
```

```
closebank()
```

To select for instance all sequences sequenced from DNA we can use:

```
choosebank("emblTP")
Mexample <- query("Mexample","M=DNA", virtual=TRUE)
Mexample$nelem
```

```
[1] 7421752
```

```
closebank()
```

[2]as named in ID or LOCUS annotation records

5.3.16 ST=status

This is used to select sequences from specified data class (EMBL) or review level (UniProt). The list of status codes can be obtained from the `readsmj()` function this way:

```
choosebank("embl")
nl <- readfirstrec(type = "SMJ")
smj <- readsmj(nl = nl, all.add = TRUE)
smj[!is.na(smj$nature) & smj$nature == "status", c("sname","libel")]
```

```
   sname                                libel
1    ANN                Annotated CON data class
2    EST          Expressed Sequence Tags data class
3    GSS            Genome Survey Sequence data class
4    HTC              High Throughput cDNA data class
5    HTG High Throughput Genome sequencing data class
6    PAT                         Patent data class
7    STD                       standard data class
8    STS              Sequence Tagged Site data class
9    TPA           Third Party Annotation data class
10   TSA    Transcriptome Shotgun Assembly data class
```

```
closebank()
choosebank("swissprot")
nl <- readfirstrec(type = "SMJ")
smj <- readsmj(nl = nl, all.add = TRUE)
smj[!is.na(smj$nature) & smj$nature == "status", c("sname","libel")]
```

```
       sname                                           libel
1   REVIEWED Entry was reviewed and annotated by UniProtKB curators
2 UNREVIEWED                              Computer-annotated entry
```

```
closebank()
```

To select for instance all fully annotated sequences from Uniprot we can use:

```
choosebank("swissprot")
STexample <- query("STexample","ST=REVIEWED", virtual=TRUE)
STexample$nelem
```

```
[1] 549008
```

```
closebank()
```

5.3.17 F=file_name

This is used to select sequences whose names are in a given file, one name per line. This is not directly implemented in seqinR, you have to use the function `crelistfromclientdata()` or its short form `clfcd()` for this purpose. Here is an example with a file of sequence names distributed with the seqinR package:

```
choosebank("emblTP")
fileSQ <- system.file("sequences/bb.mne", package = "seqinr")
cat(readLines(fileSQ),sep="\n")
```

```
A04009.OSPA
A04009.OSPB
A22442
A24006
A24008
A24010
A24012
A24014
A24016
A33362
A67759.PE1
AB011063
AB011064
AB011065
AB011066
AB011067
AB035616
AB035617
AB035618
AB041949.VLSE
```

```
listSQ <-clfcd("listSQ", file = fileSQ, type = "SQ")
getName(listSQ)
```

```
 [1] "A04009.OSPA"   "A04009.OSPB"   "A22442"        "A24006"
 [5] "A24008"        "A24010"        "A24012"        "A24014"
 [9] "A24016"        "A33362"        "A67759.PE1"    "AB011063"
[13] "AB011064"      "AB011065"      "AB011066"      "AB011067"
[17] "AB035616"      "AB035617"      "AB035618"      "AB041949.VLSE"
```

```
closebank()
```

5.3.18 FA=file_name

This is used to select sequences whose accession numbers are in a given file, one name per line. This is not directly implemented in seqinR, you have to use the function crelistfromclientdata() or its short form clfcd() for this purpose. Here is an example with a file of sequence accession numbers distributed with the seqinR package:

```
choosebank("emblTP")
fileAC <- system.file("sequences/bb.acc", package = "seqinr")
cat(readLines(fileAC),sep="\n")
```

```
AY382159
AY382160
AY491412
AY498719
AY498720
AY498721
AY498722
AY498723
AY498724
AY498725
AY498726
AY498727
AY498728
AY498729
AY499181
AY500379
AY500380
AY500381
AY500382
AY500383
```

```
listAC <- clfcd("listAC", file = fileAC, type = "AC")
getName(listAC)
```

```
 [1] "AY382159" "AY382160" "AY491412" "AY498719" "AY498720" "AY498721"
 [7] "AY498722" "AY498723" "AY498724" "AY498725" "AY498726" "AY498727"
[13] "AY498728" "AY498729" "AY499181" "AY500379" "AY500380" "AY500381"
[19] "AY500382" "AY500383"
```

```
closebank()
```

5.3.19 FK=file_name

This is used to produces the list of keywords named in given file, one keyword per line. This is not directly implemented in seqinR, you have to use the function crelistfromclientdata() or its short form clfcd() for this purpose. Here is an example with a file of keywords distributed with the seqinR package:

```
choosebank("emblTP")
fileKW <- system.file("sequences/bb.kwd", package = "seqinr")
cat(readLines(fileKW),sep="\n")
```

```
PLASMID
CIRCULAR
PARTIAL
5'-PARTIAL
3'-PARTIAL
MOTA GENE
MOTB GENE
```

```
DIVISION PRO
GYRB GENE
JOINING REGION
FTSA GENE
RPOB GENE
RPOC GENE
FLA GENE
DNAJ GENE
TUF GENE
PGK GENE
RUVA GENE
RUVB GENE
PROMOTER REGION
```

```
 listKW <- clfcd("listKW", file = fileKW, type = "KW")
 getName(listKW)
```

```
 [1] "PLASMID"         "CIRCULAR"        "PARTIAL"         "5'-PARTIAL"
 [5] "3'-PARTIAL"      "MOTA GENE"       "MOTB GENE"       "DIVISION PRO"
 [9] "GYRB GENE"       "JOINING REGION"  "FTSA GENE"       "RPOB GENE"
[13] "RPOC GENE"       "FLA GENE"        "DNAJ GENE"       "TUF GENE"
[17] "PGK GENE"        "RUVA GENE"       "RUVB GENE"       "PROMOTER REGION"
```

```
 closebank()
```

5.3.20 FS=file_name

This is used to produces the list of species named in given file, one species per line. This is not directly implemented in seqinR, you have to use the function `crelistfromclientdata()` or its short form `clfcd()` for this purpose. Here is an example with a file of species names distributed with the seqinR package:

```
 choosebank("emblTP")
 fileSP <- system.file("sequences/bb.sp", package = "seqinr")
 cat(readLines(fileSP),sep="\n")
```

```
BORRELIA ANSERINA
BORRELIA CORIACEAE
BORRELIA PARKERI
BORRELIA TURICATAE
BORRELIA HERMSII
BORRELIA CROCIDURAE
BORRELIA LONESTARI
BORRELIA HISPANICA
BORRELIA BARBOURI
BORRELIA THEILERI
BORRELIA DUTTONII
BORRELIA MIYAMOTOI
BORRELIA PERSICA
BORRELIA RECURRENTIS
BORRELIA BURGDORFERI
BORRELIA AFZELII
BORRELIA GARINII
BORRELIA ANDERSONII
BORRELIA VALAISIANA
BORRELIA JAPONICA
```

```
 listSP <- clfcd("listSP", file = fileSP, type = "SP")
 getName(listSP)
```

```
 [1] "BORRELIA ANSERINA"    "BORRELIA CORIACEAE"   "BORRELIA PARKERI"
 [4] "BORRELIA TURICATAE"   "BORRELIA HERMSII"     "BORRELIA CROCIDURAE"
 [7] "BORRELIA LONESTARI"   "BORRELIA HISPANICA"   "BORRELIA BARBOURI"
[10] "BORRELIA THEILERI"    "BORRELIA DUTTONII"    "BORRELIA MIYAMOTOI"
[13] "BORRELIA PERSICA"     "BORRELIA RECURRENTIS" "BORRELIA BURGDORFERI"
[16] "BORRELIA AFZELII"     "BORRELIA GARINII"     "BORRELIA ANDERSONII"
[19] "BORRELIA VALAISIANA"  "BORRELIA JAPONICA"
```

```
 closebank()
```

5.3.21 list_name

A list name can be re-used, for instance:

```
choosebank("emblTP")
MyFirstListName <- query("MyFirstListName", "Y=2000", virtual = TRUE)
MyFirstListName$nelem
```

```
[1] 885225
```

```
MySecondListName <- query("MySecondListName", "SP=Borrelia burgdorferi", virtual = TRUE)
MySecondListName$nelem
```

```
[1] 1682
```

```
MyThirdListName <- query("MyThirdListName", "MyFirstListName AND MySecondListName", virtual = TRUE)
MyThirdListName$nelem
```

```
[1] 131
```

```
closebank()
```

5.4 Operators

5.4.1 AND

This is the binary operator for the logical and: a sequence belongs to the resulting list if, and only if, it is present in both operands. To select for instance sequences from *Borrelia burgdorferi* that are also coding sequences we can use:

```
choosebank("emblTP")
ANDexample <- query("ANDexample","SP=Borrelia burgdorferi AND T=CDS", virtual=TRUE)
ANDexample$nelem
```

```
[1] 3218
```

```
closebank()
```

5.4.2 OR

This is the binary operator for the logical or: a sequence belongs to the resulting list if it is present in at least one of the two operands. To select for instance sequences from *Borrelia burgdorferi* or from *Escherichia coli* we can use:

```
choosebank("emblTP")
ORexample <- query("ORexample","SP=Borrelia burgdorferi OR SP=Escherichia coli", virtual=TRUE)
ORexample$nelem
```

```
[1] 28584
```

```
closebank()
```

5.4.3 NOT

This is the unary operator for the logical negation. To select for instance sequences from *Borrelia burgdorferi* that are not partial we can use:

```
choosebank("emblTP")
NOTexample <- query("NOTexample","SP=Borrelia burgdorferi AND NOT K=PARTIAL", virtual=TRUE)
NOTexample$nelem
```

```
[1] 3266
```

```
closebank()
```

5.4.4 PAR

This is a unary operator to compute the list of parent sequences of a list of sequences. The reciprocal operator is SUB. To check the reciprocity we can use for instance:

```
choosebank("emblTP")
A <- query("A","T=TRNA", virtual=TRUE)
B <- query("B","PAR A", virtual=TRUE)
C <- query("C","SUB B", virtual=TRUE)
D <- query("D","PAR C", virtual=TRUE)
emptySet <- query("emptySet", "B AND NOT D", virtual=TRUE)
emptySet$nelem
```

```
[1] 0
```

```
closebank()
```

5.4.5 SUB

This is a unary operator to add all subsequences of members of the single list operand.

```
choosebank("emblTP")
SUBexample <- query("SUBexample","AC=AE000783",virtual=T)
SUBexample$nelem
```

```
[1] 70
```

```
SUBexample2 <- query("SUBexample2","SUB SUBexample",virtual=T)
SUBexample2$nelem
```

```
[1] 943
```

```
closebank()
```

5.4.6 PS

This unary operator is used to get the list of species attached to member sequences of the operand list.

```
choosebank("emblTP")
PSexample <- query("PSexample","K=hyperthermo@",virtual=T)
PSexample2 <- query("PSexample2","PS PSexample")
getName(PSexample2)
```

```
[1] "BACILLUS LICHENIFORMIS" "DESULFUROCOCCUS"
[3] "PYROCOCCUS FURIOSUS"
```

```
closebank()
```

5.4.7 PK

This unary operator is used to get the list of keywords attached to member sequences of the operand list.

```
choosebank("emblTP")
PKexample <- query("PKexample","AC=AE000783",virtual=T)
PKexample2 <- query("PKexample2","PK PKexample")
getName(PKexample2)
```

```
[1] "DIVISION PRO" "CDS"          "RRNA"         "TRNA"
[5] "SOURCE"       "RELEASE 75"
```

```
closebank()
```

5.4.8 UN

This unary operator is used to get the list of sequences attached to a list of species or keywords.

```
 choosebank("emblTP")
 fileSP <- system.file("sequences/bb.sp", package = "seqinr")
 cat(readLines(fileSP),sep="\n")
```

```
BORRELIA ANSERINA
BORRELIA CORIACEAE
BORRELIA PARKERI
BORRELIA TURICATAE
BORRELIA HERMSII
BORRELIA CROCIDURAE
BORRELIA LONESTARI
BORRELIA HISPANICA
BORRELIA BARBOURI
BORRELIA THEILERI
BORRELIA DUTTONII
BORRELIA MIYAMOTOI
BORRELIA PERSICA
BORRELIA RECURRENTIS
BORRELIA BURGDORFERI
BORRELIA AFZELII
BORRELIA GARINII
BORRELIA ANDERSONII
BORRELIA VALAISIANA
BORRELIA JAPONICA
```

```
 listSP <- clfcd("listSP", file = fileSP, type = "SP")
 UNexample <- query("UNexample", "UN listSP", virtual = TRUE)
 UNexample$nelem
```

```
[1] 2877
```

```
 closebank()
```

5.4.9 SD

This unary operator computes the list of species placed in the tree below the members of the species list operand.

```
 choosebank("emblTP")
 hominidae <- query("hominidae","SP=Hominidae",virtual=T)
 hsp <- query("hsp","PS hominidae",virtual=T)
 hsp$nelem
```

```
[1] 19
```

```
 SDexample <- query("SDexample","SD hsp")
 getName(SDexample)
```

```
 [1] "HOMINIDAE"                      "PONGO"
 [3] "PONGO PYGMAEUS"                 "PONGO PYGMAEUS ABELII"
 [5] "PONGO PYGMAEUS PYGMAEUS"        "PONGO SP."
 [7] "HOMO/PAN/GORILLA GROUP"         "GORILLA"
 [9] "GORILLA GORILLA"                "GORILLA GORILLA BERINGEI"
[11] "GORILLA GORILLA GRAUERI"        "GORILLA GORILLA GORILLA"
[13] "GORILLA GORILLA UELLENSIS"      "PAN"
[15] "PAN TROGLODYTES"                "PAN TROGLODYTES SCHWEINFURTHII"
[17] "PAN TROGLODYTES TROGLODYTES"    "PAN TROGLODYTES VERUS"
[19] "PAN TROGLODYTES VELLEROSUS"     "PAN PANISCUS"
[21] "HOMO"                           "HOMO SAPIENS"
[23] "HOMO SAPIENS NEANDERTHALENSIS"
```

```
 closebank()
```

5.4.10 KD

This unary operator computes the list of keywords placed in the tree below the members of the keywords list operand.

```
choosebank("emblTP")
cat <- query("cat","SP=Felis catus", virtual = TRUE)
catkw <- query("catkw","PK cat", virtual = TRUE)
catkw$nelem
```

```
[1] 540
```

```
KDexample <- query("KDexample","KD catkw", virtual = TRUE)
KDexample$nelem
```

```
[1] 572
```

```
closebank()
```

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 16:07:13 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 6

How to deal with sequences

Charif, D. Lobry, J.R.

6.1 Sequence classes

There are currently 5 classes of sequences, depending on the way they were obtained:

- **SeqFastadna** is the class for nucleic acid sequences that were imported from a fasta file.
- **SeqFastaAA** is the class for amino-acid acid sequences that were imported from a fasta file.
- **seqAcnucWeb** is the class for the sequences coming from an ACNUC database server.
- **SeqFrag** is the class for the sequences that are fragments of other sequences.
- **qaw** is the class for the result of a call to the `query()` function.

6.2 Generic methods for sequences

All sequence classes are sharing a common interface, so that there are very few method names we have to remember. In addition, all classes have their specific `as.ClassName` method that return an instance of the class, and `is.ClassName` method to check whether an object belongs or not to the class. Available methods are summarized in table 6.1.

Methods	Result	Type of result
getFrag	a sequence fragment	a sequence fragment
getSequence	the sequence	vector of characters
getName	the name of a sequence	string
getLength	the length of a sequence	numeric vector
getTrans	translation into amino-acids	vector of characters
getAnnot	sequence annotations	vector of string
getLocation	position of a Sequence on its parent sequence	list of numeric vector

Table 6.1: Available methods for sequence classes.

6.2.1 From classes to methods

To obtain the list of methods available for a given class, try this at your R prompt:

```
 methods(class = "SeqFastadna")
[1] getAnnot    getFrag     getLength   getName     getSequence getTrans
[7] summary
see '?methods' for accessing help and source code
 methods(class = "SeqFastaAA")
[1] getAnnot    getFrag     getLength   getName     getSequence summary
see '?methods' for accessing help and source code
 methods(class = "SeqAcnucWeb")
 [1] getAnnot    getFrag     getKeyword  getLength   getLocation getName
 [7] getSequence getTrans    plot        print
see '?methods' for accessing help and source code
 methods(class = "SeqFrag")
[1] getFrag     getLength   getName     getSequence getTrans
see '?methods' for accessing help and source code
 methods(class = "qaw")
[1] getAnnot    getFrag     getKeyword  getLength   getLocation getName
[7] getSequence getTrans    print
see '?methods' for accessing help and source code
```

6.2.2 From methods to classes

To obtain the list of classes for which a given method exists, try this at your R prompt:

```
 methods(getFrag)
[1] getFrag.character   getFrag.default     getFrag.list
[4] getFrag.logical     getFrag.qaw         getFrag.SeqAcnucWeb
[7] getFrag.SeqFastaAA  getFrag.SeqFastadna getFrag.SeqFrag
see '?methods' for accessing help and source code
 methods(getSequence)
[1] getSequence.character   getSequence.default     getSequence.list
[4] getSequence.logical     getSequence.qaw         getSequence.SeqAcnucWeb
[7] getSequence.SeqFastaAA  getSequence.SeqFastadna getSequence.SeqFrag
see '?methods' for accessing help and source code
 methods(getName)
[1] getName.default     getName.list        getName.logical
[4] getName.qaw         getName.SeqAcnucWeb getName.SeqFastaAA
[7] getName.SeqFastadna getName.SeqFrag
see '?methods' for accessing help and source code
 methods(getLength)
```

```
[1] getLength.character   getLength.default     getLength.list
[4] getLength.logical     getLength.qaw         getLength.SeqAcnucWeb
[7] getLength.SeqFastaAA  getLength.SeqFastadna getLength.SeqFrag
see '?methods' for accessing help and source code
```

```
 methods(getTrans)
```

```
[1] getTrans.character   getTrans.default     getTrans.list
[4] getTrans.logical     getTrans.qaw         getTrans.SeqAcnucWeb
[7] getTrans.SeqFastadna getTrans.SeqFrag
see '?methods' for accessing help and source code
```

```
 methods(getAnnot)
```

```
[1] getAnnot.default     getAnnot.list        getAnnot.logical
[4] getAnnot.qaw         getAnnot.SeqAcnucWeb getAnnot.SeqFastaAA
[7] getAnnot.SeqFastadna
see '?methods' for accessing help and source code
```

```
 methods(getLocation)
```

```
[1] getLocation.default     getLocation.list        getLocation.logical
[4] getLocation.qaw         getLocation.SeqAcnucWeb
see '?methods' for accessing help and source code
```

6.3 Internal representation of sequences

The default mode of sequence storage is done with vectors of characters instead of strings[1]. This is very convenient for the user because all R tools to manipulate vectors are immediatly available. The price to pay is that this storage mode is extremly expensive in terms of memory. They are two utilities called `s2c()` and `c2s()` that allows to convert strings into vector of characters, and *vice versa*, respectively.

6.3.1 Sequences as vectors of characters

In the vectorial representation mode, all the very convenient R tools for indexing vectors are at hand.

1. Vectors can be indexed by a vector of *positive* integers saying which elements are to be selected. As we have already seen, the first 50 elements of a sequence are easily extracted thanks to the binary operator `from:to`, as in:

    ```
     dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
     myseq <- read.fasta(file = dnafile)[[1]]
     1:50
    ```

    ```
     [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
    [25] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
    [49] 49 50
    ```

    ```
     myseq[1:50]
    ```

    ```
     [1] "a" "t" "g" "a" "a" "a" "a" "t" "g" "a" "a" "t" "a" "a" "a" "a" "g" "t"
    [19] "c" "t" "c" "a" "t" "c" "g" "t" "c" "c" "t" "c" "t" "g" "t" "t" "t" "a"
    [37] "t" "c" "a" "g" "c" "a" "g" "g" "g" "t" "t" "a" "c" "t"
    ```

    The `seq()` function allows to build more complexe integer vectors. For instance in coding sequences it is very common to focus on third codon positions where selection is weak. Let's extract bases from third codon positions:

[1] This default behaviour can be neutralized by setting the `as.string` argument to TRUE.

```
tcp <- seq(from = 3, to = length(myseq), by = 3)
tcp[1:10]
```

```
[1]  3  6  9 12 15 18 21 24 27 30
```

```
myseqtcp <- myseq[tcp]
myseqtcp[1:10]
```

```
[1] "g" "a" "g" "t" "a" "t" "c" "c" "c" "c"
```

2. Vectors can also be indexed by a vector of *negative* integers saying which elements have to be removed. For instance, if we want to keep first and second codon positions, the easiest way is to remove third codon positions:

   ```
   -tcp[1:10]
   ```

   ```
   [1]  -3  -6  -9 -12 -15 -18 -21 -24 -27 -30
   ```

   ```
   myseqfscp <- myseq[-tcp]
   myseqfscp[1:10]
   ```

   ```
   [1] "a" "t" "a" "a" "a" "t" "a" "a" "a" "a"
   ```

3. Vectors are also indexable by a vector of *logicals* whose `TRUE` values say which elements to keep. Here is a different way to extract all third coding positions from our sequence. First, we define a vector of three logicals with only the last one true:

   ```
   ind <- c(F, F, T)
   ind
   ```

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

   This vector seems too short for our purpose because our sequence is much more longer with its 921 bases. But under R vectors are automatically *recycled* when they are not long enough:

   ```
   (1:30)[ind]
   ```

   ```
   [1]  3  6  9 12 15 18 21 24 27 30
   ```

   ```
   myseqtcp2 <- myseq[ind]
   ```

   The result should be the same as previously:

   ```
   identical(myseqtcp, myseqtcp2)
   ```

   ```
   [1] TRUE
   ```

   This recycling rule is extremely convenient in practice but may have surprising effects if you assume (incorrectly) that there is a stringent dimension control for R vectors as in linear algebra.

Another advantage of working with vector of characters is that most R functions are vectorized so that many things can be done without explicit looping. Let's give some very simple examples:

```
(tota <- sum(myseq == "a"))
```

```
[1] 238
```

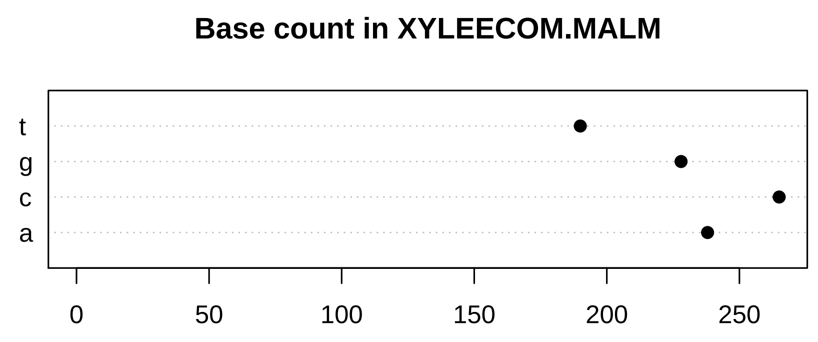


Figure 6.1: Visual representation of the base counts in a nucleic acid sequence.

The total number of `a` in our sequence is 238. Let's compare graphically the different base counts in our sequence. The following code was used to produce figure 6.1:

```
basecount <- table(myseq)
myseqname <- getName(myseq)
dotchart(basecount, xlim = c(0, max(basecount)), pch = 19,
  main = paste("Base count in",  myseqname))
```

The following code was used to display (*cf* figure 6.2) the dinucleotide counts in the sequence:

```
dinuclcount <- count(myseq, 2)
dotchart(dinuclcount[order(dinuclcount)], xlim = c(0, max(dinuclcount)), pch = 19,
  main = paste("Dinucleotide count in",  myseqname))
```

The following code was used to display (*cf* figure 6.3) the codon usage in the sequence:

```
codonusage <- uco(myseq)
dotchart.uco(codonusage, main = paste("Codon usage in",  myseqname))
```

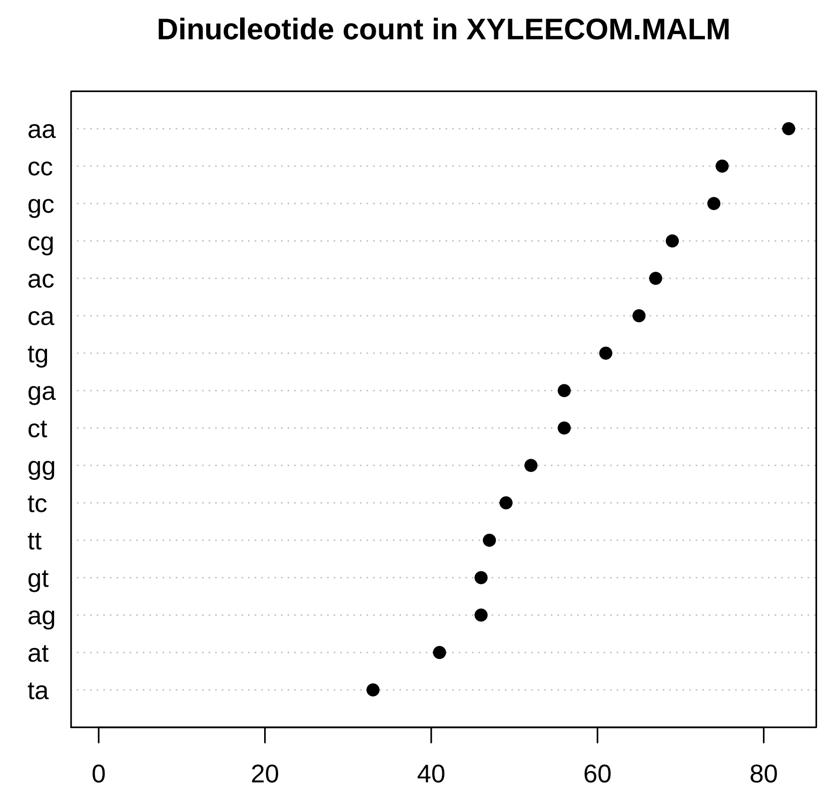


Figure 6.2: Visual representation of dinucleotide counts in a nucleic acid sequence.

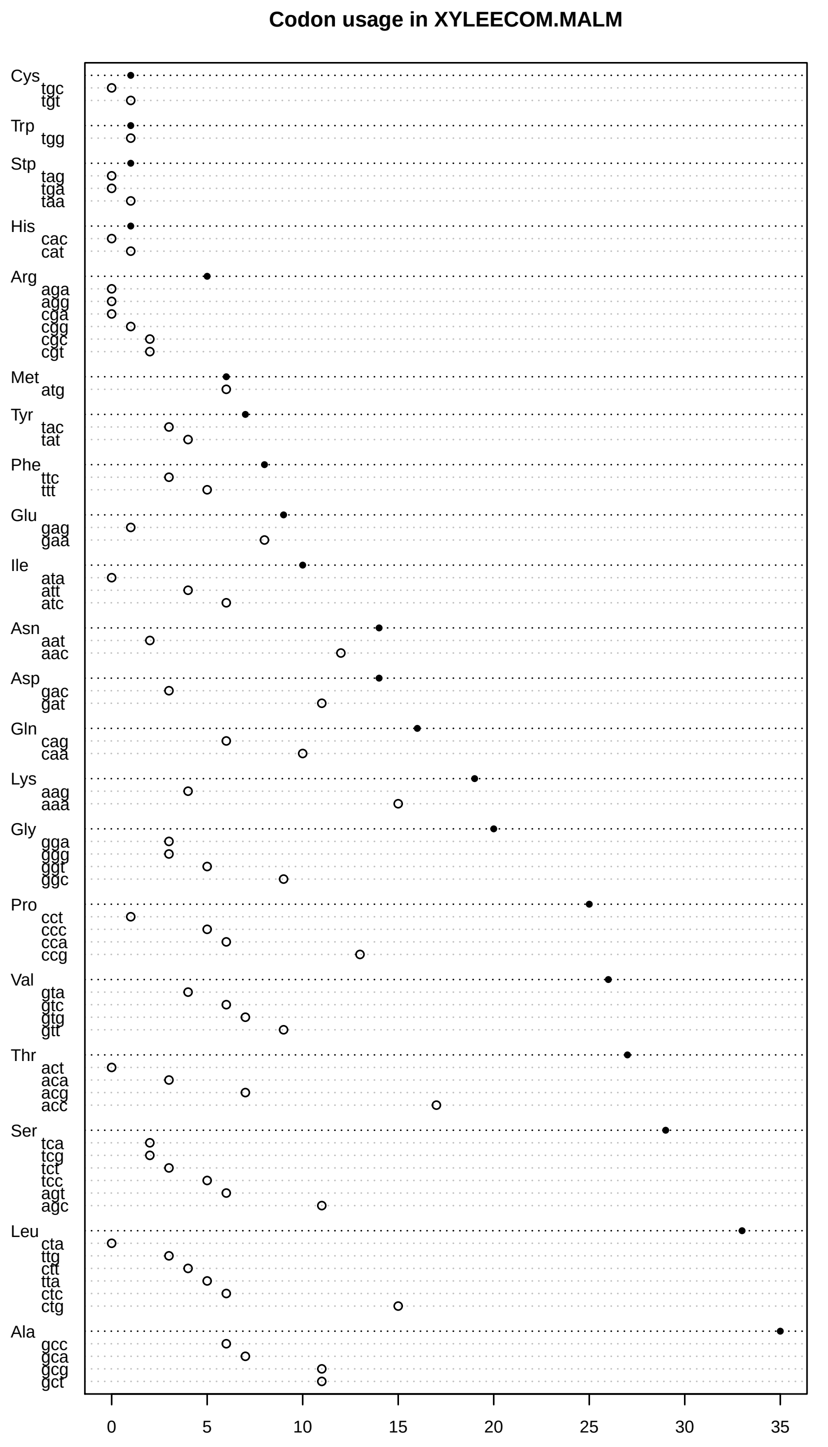


Figure 6.3: Visual representation of codon usage in a coding sequence with the function `dotchart.uco()`. Codons are grouped by amino-acid for a given genetic code. Black dots are the sums by synonymous codons, that is the amino-acid count.

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 16:24:00 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 7

Multivariate analyses

Lobry, J.R.

7.1 Correspondence analysis

This is the most popular multivariate data analysis technique for amino-acid and codon count tables, its application, however, is not without pitfalls [74]. Its primary goal is to transform a table of counts into a graphical display, in which each gene (or protein) and each codon (or amino-acid) is depicted as a point. Correspondence analysis (CA) may be defined as a special case of principal components analysis (PCA) with a different underlying metrics. The interest of the metrics in CA, that is the way we measure the distance between two individuals, is illustrated bellow with a very simple example (Table 7.1 inspired from [21]) with only three proteins having only three amino-acids, so that we can represent exactly on a map the consequences of the metric choice.

```
data(toyaa)
toyaa
```

```
  Ala Val Cys
1 130  70   0
2  60  40   0
3  60  35   5
```

	Ala	Val	Cys
1	130	70	0
2	60	40	0
3	60	35	5

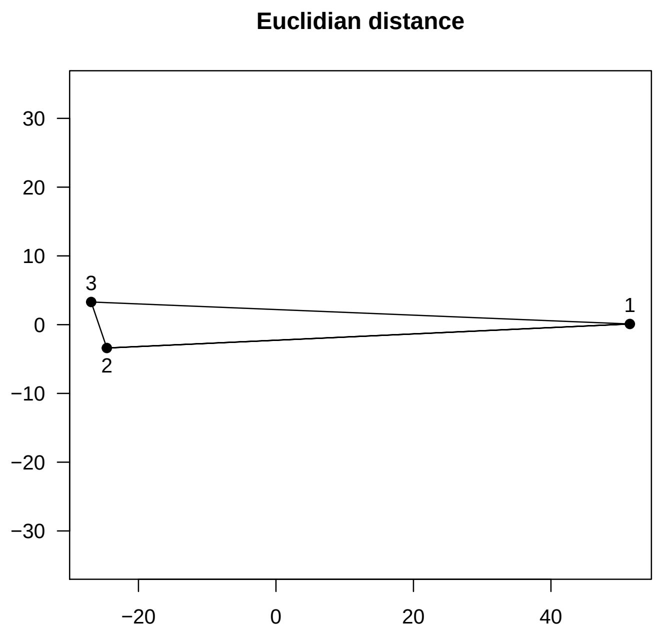
Table 7.1: A very simple example of amino-acid counts in three proteins to be loaded with `data(toyaa)`.

Let's first use the regular Euclidian metrics between two proteins i and i',

$$d^2(i, i') = \sum_{j=1}^{J} (n_{ij} - n_{i'j})^2 \qquad (7.1)$$

to visualize this small data set:

```
library(ade4)
pco <- dudi.pco(dist(toyaa), scann = F, nf = 2)
myplot <- function( res, ... )
{
  plot(res$li[ , 1], res$li[ , 2], ...)
  text(x = res$li[ , 1], y = res$li[ , 2], labels = 1:3, pos = ifelse(res$li[ , 2] < 0, 1, 3))
  perm <- c(3, 1, 2)
  lines( c(res$li[ , 1], res$li[perm, 1]), c(res$li[ , 2], res$li[perm, 2]))
}
myplot(pco, main = "Euclidian distance", asp = 1, pch = 19, xlab = "", ylab = "", las = 1)
```



From this point of view, the first individual is far away from the two others. But thinking about it, this is a rather trivial effect of protein size:

```
rowSums(toyaa)
```

```
  1   2   3
200 100 100
```

With 200 amino-acids, the first protein is two times bigger than the others so that when computing the Euclidian distance (7.1) its n_{ij} entries are on average bigger, sending it away from the others. To get rid of this trivial effect, the first obvious idea is to divide counts by protein lengths so as to work with *protein profiles*. The corresponding distance is,

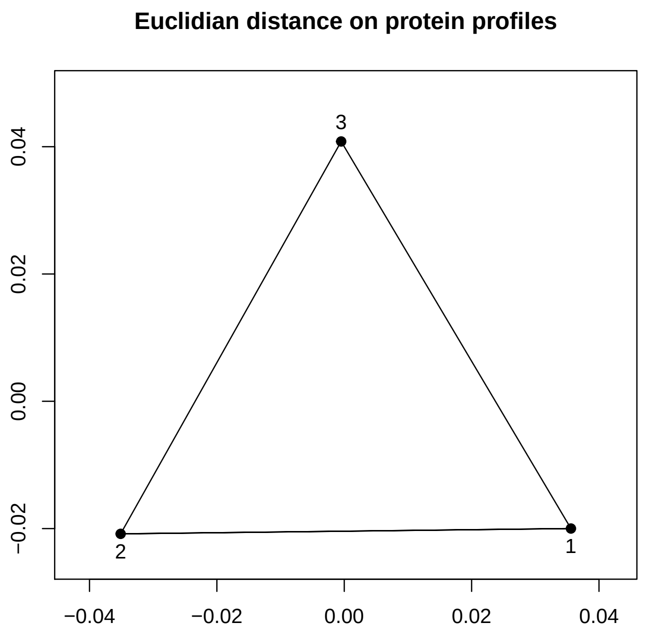
$$d^2(i,i') = \sum_{j=1}^{J} (\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}})^2 \qquad (7.2)$$

where $n_{i\bullet}$ and $n_{i'\bullet}$ are the total number of amino-acids in protein i and i', respectively.

```
profile <- toyaa/rowSums(toyaa)
profile
```

```
   Ala  Val  Cys
1 0.65 0.35 0.00
2 0.60 0.40 0.00
3 0.60 0.35 0.05
```

```
dudi.pco(dist(profile), scann = F, nf = 2) -> pco1
myplot(pco1, main = "Euclidian distance on protein profiles", asp = 1, pch = 19, xlab = "", ylab = "",
  ylim = range(pco1$li[ , 2])*1.2)
```

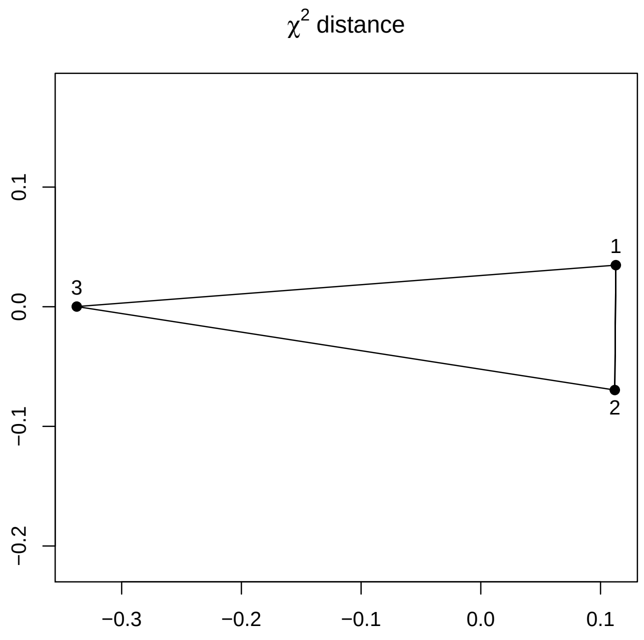


The pattern is now completely different with the three protein equally spaced. This is normal because in terms of relative amino-acid composition they are all differing two-by-two by 5% at the level of two amino-acids only. We have clearly removed the trivial protein size effect, but this is still not completely satisfactory. The proteins are differing by 5% for all amino-acids but the situation is somewhat different for `Cys` because this amino-acid is very rare. A difference of 5% for a rare amino-acid has not the same significance than a difference of 5% for a common amino-acid such as `Ala` in our example. To cope with this, CA make use of a variance-standardizing technique to compensate for the larger variance in high frequencies and the smaller variance in low frequencies. This is achieved with the use of the *chi-square distance* (χ^2) which differs from the previous Euclidean distance on profiles (7.2) in that each square is weighted by the inverse of the frequency corresponding to each term,

$$d^2(i,i') = n_{\bullet\bullet} \sum_{j=1}^{J} \frac{1}{n_{\bullet j}} (\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}})^2 \tag{7.3}$$

where $n_{\bullet j}$ is the total number of amino-acid of kind j and $n_{\bullet\bullet}$ the total number of amino-acids. With this point of view, the map is now like this:

```
coa <- dudi.coa(toyaa, scann = FALSE, nf = 2)
myplot(coa, main = expression(paste(chi^2," distance")),
  asp = 1, pch = 19, xlab = "", ylab = "")
```



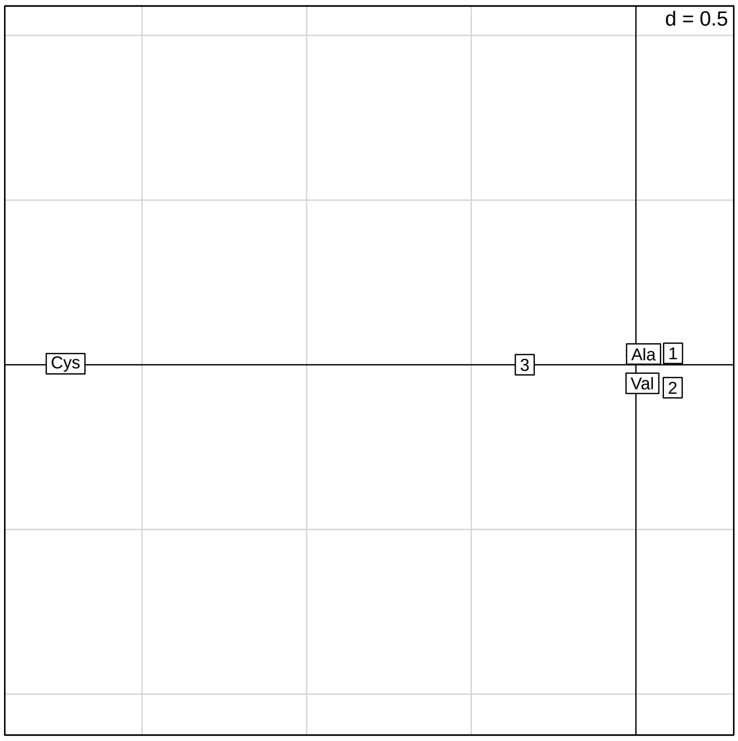
The pattern is completely different with now protein number 3 which is far away from the others because it is enriched in the rare amino-acid `Cys` as compared to others.

The purpose of this small example was to demonstrates that the metric choice is not without dramatic effects on the visualisation of data. Depending on your objectives, you may agree or disagree with the χ^2 metric choice, that's not a problem, the important point is that you should be aware that there is an underlying model there, *chacun a son goût* ou *chacun à son goût*, it's up to you.

Now, if you agree with the χ^2 metric choice, there's a nice representation that may help you for the interpretation of results. This is a kind of "biplot" representation in which the lines and columns of the dataset are simultaneously represented, in the right way, that is as a graphical *translation* of a mathematical theorem, but let's see how does it look like in practice:

```
scatter(coa, clab.col = 0.8, clab.row = 0.8, posi = "none")
```

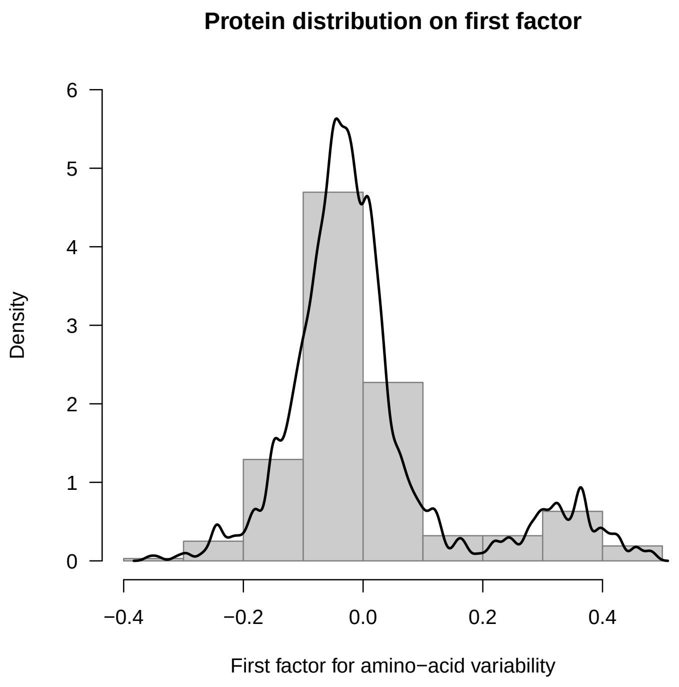
```
NULL
```



What is obvious is that the Cys content has a major effect on protein variability here, no scoop. Please note how the information is well summarised here: protein number 3 differs because it's enriched in in Cys ; protein number 1 and 2 are almost the same but there is a small trend protein number 1 to be enriched in Ala. As compared to to table 7.1 this graph is of poor information here, so let's try a more big-rooom-sized example (with 20 columns so as to illustrate the dimension reduction technique).

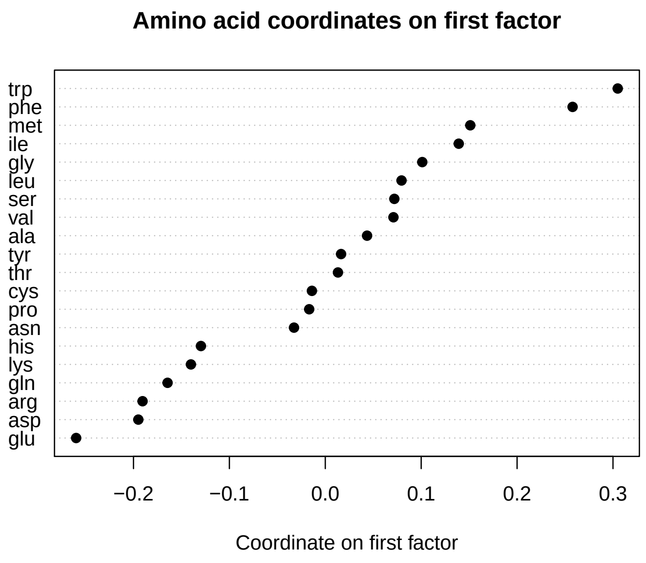
Data are from [59], a sample of the proteome of *Escherichia coli.* According to the title of this paper, the most important factor for the between-protein variability is hydrophilic - hydrophobic gradient. Let's try to reproduce this assertion :

```
download.file(url="ftp://pbil.univ-lyon1.fr/pub/datasets/NAR94/data.txt", destfile = "data.txt")
ec <- read.table(file = "data.txt", header = TRUE,
    row.names = 1)
ec.coa <- dudi.coa(ec, scann = FALSE, nf = 1)
F1 <- ec.coa$li[,1]
hist(F1, proba = TRUE, xlab = "First factor for amino-acid variability",
col = grey(0.8), border = grey(0.5), las = 1, ylim = c(0,6),
        main="Protein distribution on first factor")
lines(density(F1, adjust = 0.5), lwd = 2)
```



There is clearly a bimodal distribution of proteins on the first factor. What are the the amino-acid coordinates on this factor?

```
aacoo <- ec.coa$co[ , 1]
names(aacoo) <- rownames(ec.coa$co)
aacoo <- sort( aacoo)
dotchart(aacoo, pch = 19, xlab = "Coordinate on first factor",
main = "Amino acid coordinates on first factor")
```



Aliphatic and aromatic amino-acids have positive values while charged amino-acids have negative values[1]. Let's try to compute the GRAVY score (*i.e.* the Kyte and Doolittle hydropathic index[49]) of our proteins to compare this with their coordinates on the first factor. We need first the amino-acid *relatives* frequencies in the proteins, for this we divide the all the amino-acid counts by the total by row:

```
 ecfr <- ec/rowSums(ec)
 ecfr[1:5, 1:5]

            arg        leu        ser        thr        pro
FOLE 0.05829596 0.10313901 0.06278027 0.08520179 0.03587444
MSBA 0.06529210 0.10309278 0.08591065 0.06185567 0.02233677
NARV 0.06637168 0.12831858 0.06637168 0.05752212 0.03539823
NARW 0.05627706 0.16450216 0.05627706 0.03030303 0.04329004
NARY 0.06614786 0.06420233 0.05058366 0.03891051 0.06031128
```

We need also the coefficients corresponding to the GRAVY score:

```
 gravy <- read.table(file ="ftp://pbil.univ-lyon1.fr/pub/datasets/NAR94/gravy.txt")
 gravy[1:5, ]

   V1   V2
1 Ala  1.8
2 Arg -4.5
3 Asn -3.5
4 Asp -3.5
5 Cys  2.5

 coef <- gravy$V2
```

The coefficient are given in the alphabetical order of the three letter code for the amino acids, that is in a different order than in the object `ecfr`:

```
 names(ecfr)

 [1] "arg" "leu" "ser" "thr" "pro" "ala" "gly" "val" "lys" "asn" "gln" "his"
[13] "glu" "asp" "tyr" "cys" "phe" "ile" "met" "trp"
```

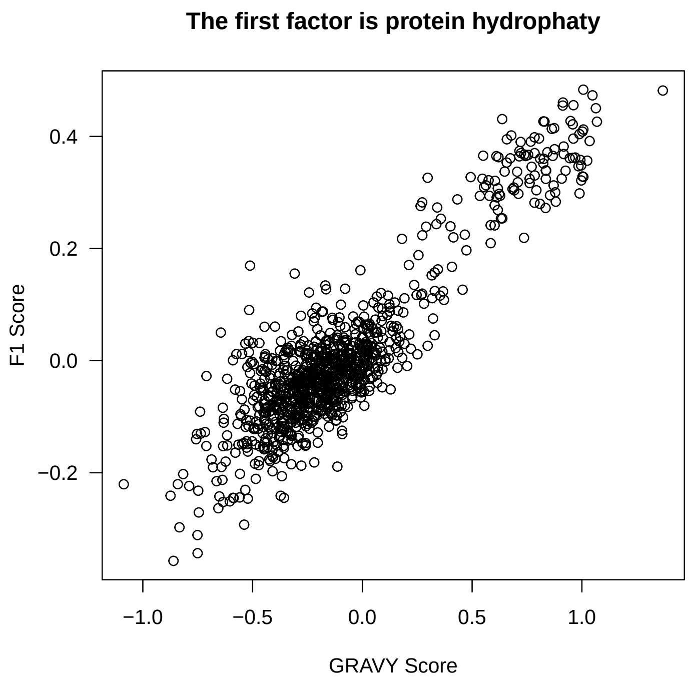
We then re-order the columns of the data set and check that everthing is OK:

```
 ecfr <- ecfr[ , order(names(ecfr))]
 ecfr[1:5,1:5]

            ala        arg        asn        asp         cys
FOLE 0.08520179 0.05829596 0.04035874 0.05381166 0.008968610
MSBA 0.08247423 0.06529210 0.03608247 0.05154639 0.003436426
NARV 0.05309735 0.06637168 0.01769912 0.02212389 0.013274336
NARW 0.09090909 0.05627706 0.02597403 0.09090909 0.017316017
NARY 0.06225681 0.06614786 0.03891051 0.05642023 0.035019455

 all(names(ecfr) == tolower(as.character(gravy$V1)))

[1] TRUE
```

Now, thanks to R build-in matrix multiplication, it's only one line to compute the GRAVY score:

```
 as.matrix(ecfr) %*% coef -> gscores
 plot(gscores,F1,xlab="GRAVY Score", ylab="F1 Score",las=1,main="The first factor is protein hydrophaty")
```

[1]The physico-chemical classes for amino acids are given in the component `AA.PROPERTY` of the `SEQINR.UTIL` object.



The proteins with high GRAVY scores are integral membrane proteins, and those with low scores are cytoplasmic proteins. Now, suppose that we want to adjust a mixture of two normal distributions to get an estimate of the proportion of cytoplasmic and integral membrane proteins. We first have a look on the predefined distributions (Table 7.2), but there is apparently not an out of the box solution. We then define our own probability density function and then use `fitdistr` from package `MASS` to get a maximum likelihood estimate of the parameters:

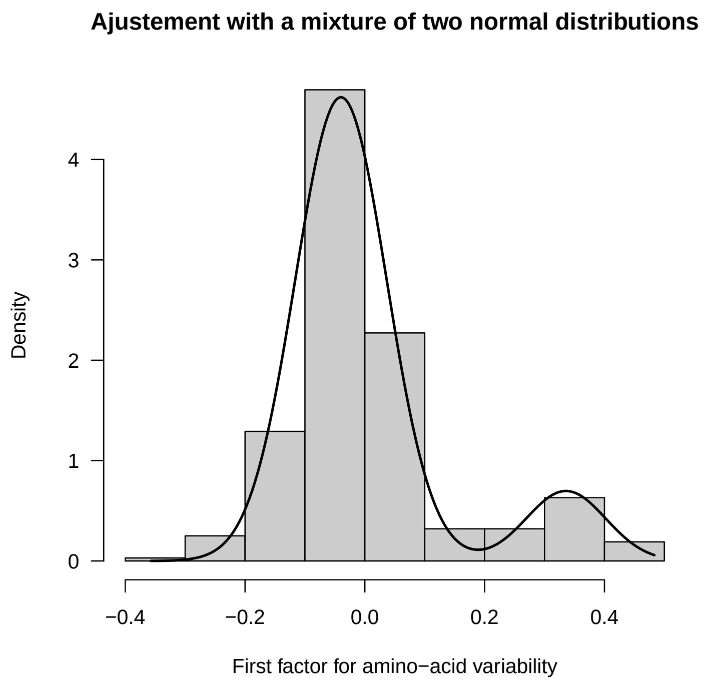
```
dmixnor <- function(x, p, m1, sd1, m2, sd2){
  p*dnorm(x, m1, sd1) + (1 - p)*dnorm(x, m2, sd2)
}
library(MASS)
fitdistr(F1, dmixnor, list(p=0.88, m1=-0.04, sd1=0.076, m2=0.34, sd2=0.07))$estimate -> e
e
```

```
         p          m1         sd1          m2         sd2
0.88405009 -0.03989489  0.07632235  0.33579162  0.06632259
```

```
hist(F1, proba = TRUE, col = grey(0.8),
main = "Ajustement with a mixture of two normal distributions",
xlab = "First factor for amino-acid variability", las = 1)
xx <- seq(from = min(F1), to = max(F1), length = 200)
lines(xx, dmixnor(xx,e[1],e[2],e[3],e[4],e[5]), lwd = 2)
```

	d	p	q	r
beta	dbeta	pbeta	qbeta	rbeta
binom	dbinom	pbinom	qbinom	rbinom
cauchy	dcauchy	pcauchy	qcauchy	rcauchy
chisq	dchisq	pchisq	qchisq	rchisq
exp	dexp	pexp	qexp	rexp
f	df	pf	qf	rf
gamma	dgamma	pgamma	qgamma	rgamma
geom	dgeom	pgeom	qgeom	rgeom
hyper	dhyper	phyper	qhyper	rhyper
lnorm	dlnorm	plnorm	qlnorm	rlnorm
logis	dlogis	plogis	qlogis	rlogis
nbinom	dnbinom	pnbinom	qnbinom	rnbinom
norm	dnorm	pnorm	qnorm	rnorm
pois	dpois	ppois	qpois	rpois
signrank	dsignrank	psignrank	qsignrank	rsignrank
t	dt	pt	qt	rt
unif	dunif	punif	qunif	runif
weibull	dweibull	pweibull	qweibull	rweibull
wilcox	dwilcox	pwilcox	qwilcox	rwilcox

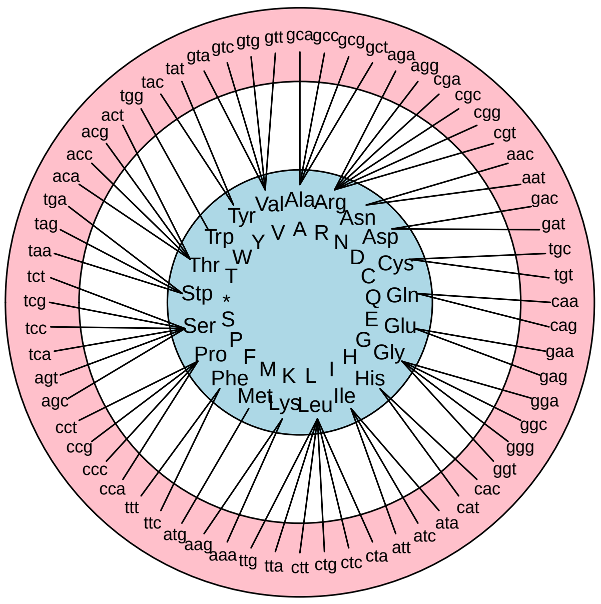
Table 7.2: Density, distribution function, quantile function and random generation for the predefined distributions under R



7.2 Synonymous and non-synonymous analyses

Genetic codes are surjective applications from the set codons ($n = 64$) into the set of amino-acids ($n = 20$) :

The surjective nature of genetic codes
Genetic code number 1


Val Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser Stp Thr Trp Tyr
V A R N D C Q E G H I L K M F P S * T W Y
gta gtc gtg gtt gca gcc gcg gct aga agg cga cgc cgg cgt aac aat gac gat tgc tgt caa cag gaa gag gga ggc ggg ggt cac cat ata atc att cta ctc ctg ctt tta ttg aaa aag atg ttc ttt cca ccc ccg cct agc agt tca tcc tcg tct taa tag tga aca acc acg act tgg tac tat

Adapted from insert 2 in Lobry & Chessel (2003) JAG 44:235

Two codons encoding the same amino-acid are said synonymous while two codons encoding a different amino-acid are said non-synonymous. The distinction between the synonymous and non-synonymous level are very important in evolutionary studies because most of the selective pressure is expected to work at the non-synonymous level, because the amino-acids are the components of the proteins, and therefore more likely to be subject to selection.

K_s and K_a are an estimation of the number of substitutions per synonymous site and per non-synonymous site, respectively, between two protein-coding genes [52]. The $\frac{K_a}{K_s}$ ratio is used as tool to evaluate selective pressure (see [36] for a nice back to basics). Let's give a simple illustration with three orthologous genes of the thioredoxin familiy from *Homo sapiens*, *Mus musculus*, and *Rattus norvegicus* species:

```
ortho <- read.alignment(system.file("sequences/ortho.fasta", package = "seqinr"), format="fasta")
kaks.ortho <- kaks(ortho)
kaks.ortho$ka/kaks.ortho$ks
                                            AK002358.PE1              501 residues
HSU78678.PE1              501 residues                                   0.1243472
```

```
RNU73525.PE1            501 residues                            0.1405012
                                     HSU78678.PE1            501 residues
HSU78678.PE1            501 residues
RNU73525.PE1            501 residues                            0.1356036
```

The $\frac{K_a}{K_s}$ ratios are less than 1, suggesting a selective pressure on those proteins during evolution.

For transversal studies (*i.e.* codon usage studies in a genome at the time it was sequenced) there is little doubt that the strong requirement to distinguish between synonymous and an non-synonymous variability was the source of many mistakes [74]. We have just shown here with a scholarship example that the metric choice is not neutral. If you consider that the χ^2 metric is not too bad, with respect to your objectives, and that you want to quantify the synonymous and an non-synonymous variability, please consider reading this paper [58], and follow this link `http://pbil.univ-lyon1.fr/members/lobry/repro/jag03/` for on-line reproducibility.

Let's now use the toy example given in table 7.3 to illustrate how to study synonymous and non-synonymous codon usage.

```
 data(toycodon)
 toycodon
```

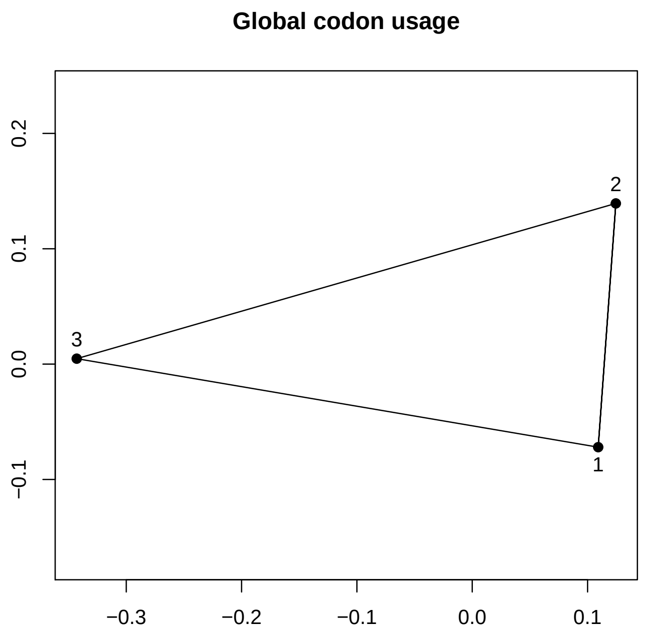
```
  gca gcc gcg gct gta gtc gtg gtt tgt tgc
1  33  32  32  33  18  17  17  18   0   0
2  13  17  17  13   8  12  12   8   0   0
3  16  14  14  16   8   9  10   8   3   2
```

	gca	gcc	gcg	gct	gta	gtc	gtg	gtt	tgt	tgc
1	33	32	32	33	18	17	17	18	0	0
2	13	17	17	13	8	12	12	8	0	0
3	16	14	14	16	8	9	10	8	3	2

Table 7.3: A very simple example of codon counts in three coding sequences to be loaded with `data(toycodon)`.

Let's first have a look to global codon usage, we do not take into account the structure of the genetic code:

```
 global <- dudi.coa(toycodon, scann = FALSE, nf= 2)
 myplot(global, asp = 1, pch = 19, xlab = "", ylab = "", main = "Global codon usage")
```



From a global codon usage point of view, coding sequence number 3 is away. To take into account the genetic code structure, we need to know for which amino-acid the codons are coding. The codons are given by the names of the columns of the object `toycodon`:

```
names(toycodon)
 [1] "gca" "gcc" "gcg" "gct" "gta" "gtc" "gtg" "gtt" "tgt" "tgc"
```

Put all codon names into a single string:

```
c2s(names(toycodon))
[1] "gcagccgcggctgtagtcgtggttttgttgc"
```

Transform this string as a vector of characters:

```
s2c(c2s(names(toycodon)))
 [1] "g" "c" "a" "g" "c" "c" "g" "c" "g" "g" "c" "t" "g" "t" "a" "g" "t" "c"
[19] "g" "t" "g" "g" "t" "t" "t" "g" "t" "t" "g" "c"
```

Translate this into amino-acids using the default genetic code:

```
translate(s2c(c2s(names(toycodon))))
 [1] "A" "A" "A" "A" "V" "V" "V" "V" "C" "C"
```

Use the three letter code for amino-acid instead:

```
aaa(translate(s2c(c2s(names(toycodon)))))
 [1] "Ala" "Ala" "Ala" "Ala" "Val" "Val" "Val" "Val" "Cys" "Cys"
```

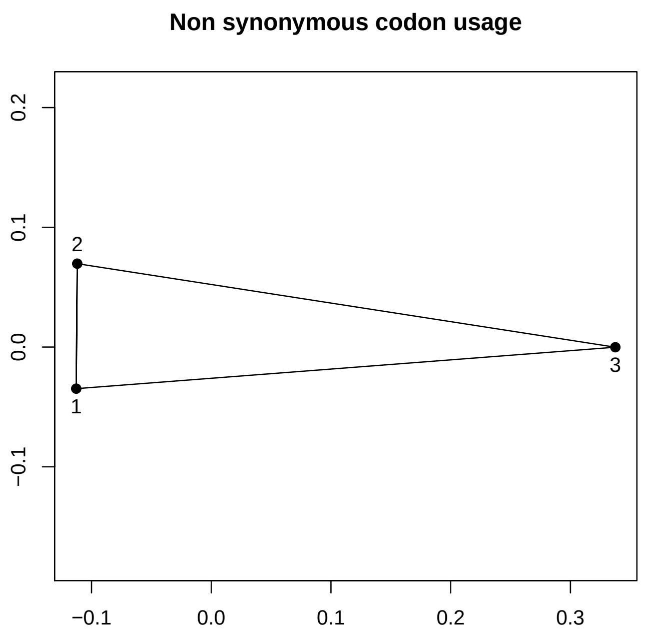
Make this a factor:

```
facaa <- factor(aaa(translate(s2c(c2s(names(toycodon))))))
facaa
```

```
[1] Ala Ala Ala Ala Val Val Val Val Cys Cys
Levels: Ala Cys Val
```

The non synonymous codon usage analysis is the between amino-acid analysis:

```
nonsynonymous <- t(bca(x = t(global), fac = facaa, scann = FALSE, nf = 2))
myplot(nonsynonymous, asp = 1, pch = 19, xlab = "", ylab = "", main = "Non synonymous codon usage")
```



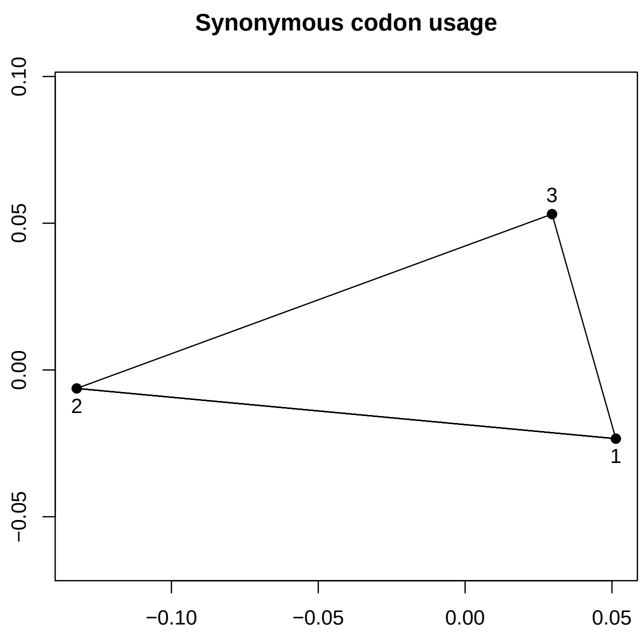
This is reminiscent of something, let's have a look at amino-acid counts:

```
by(t(toycodon), facaa, colSums)
```

```
INDICES: Ala
  1   2   3
130  60  60
---------------------------------------------------------
INDICES: Cys
1 2 3
0 0 5
---------------------------------------------------------
INDICES: Val
 1  2  3
70 40 35
```

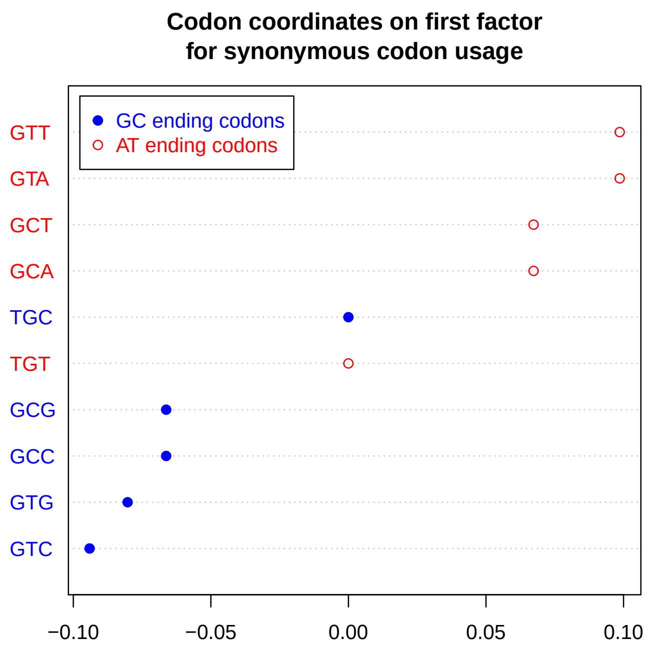
This is exactly the same data set that we used previously (table 7.1) at the amino-acid level. The non synonymous codon usage analysis is exactly the same as the amino-acid analysis. Coding sequence number 3 is far away because it codes for many Cys, a rare amino-acid. Note that at the global codon usage level, this is also the major visible structure. To get rid of this amino-acid effect, we use the synonymous codon usage analysis, that is the within amino-acid analysis:

```
synonymous <- t(wca(x = t(global), fac = facaa, scann = FALSE, nf = 2))
myplot(synonymous, asp = 1, pch = 19, xlab = "", ylab = "", main = "Synonymous codon usage")
```



Now, coding sequence number 2 is away. When the amino-acid effect is removed, the pattern is then completely different. To interpret the result we look at the codon coordinates on the first factor of synonymous codon usage:

```
synonymous$co[ , 1, drop = FALSE] -> tmp
tmp <- tmp[order(tmp$Axis1), , drop = FALSE]
colcod <- sapply(rownames(tmp), function(x) ifelse(substr(x,3,3) == "c" || substr(x,3,3) == "g", "blue",
pchcod <- ifelse(colcod=="red",1,19)
dotchart(tmp$Axis1, labels = toupper(rownames(tmp)),
color = colcod, pch = pchcod,
main = "Codon coordinates on first factor\nfor synonymous codon usage")
legend("topleft", inset = 0.02, legend = c("GC ending codons", "AT ending codons"),
text.col = c("blue", "red"), pch = c(19,1), col = c("blue","red"), bg = "white")
```



At the synonymous level, coding sequence number 2 is different because it is enriched in GC-ending codons as compared to the two others. Note that this is hard to see at the global codon usage level because of the strong amino-acid effect.

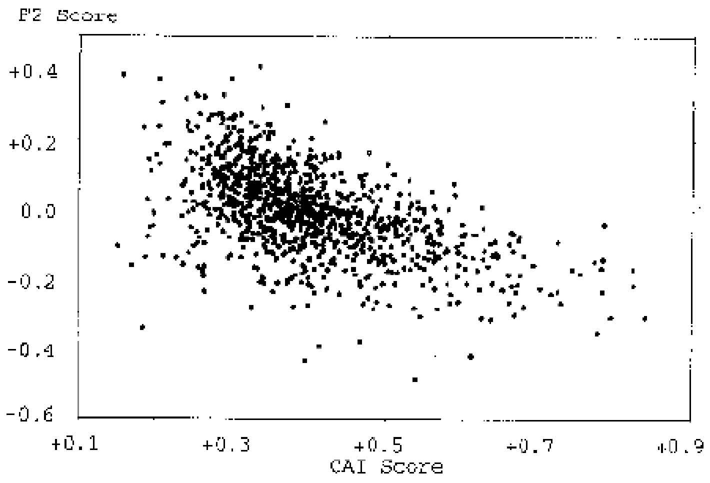
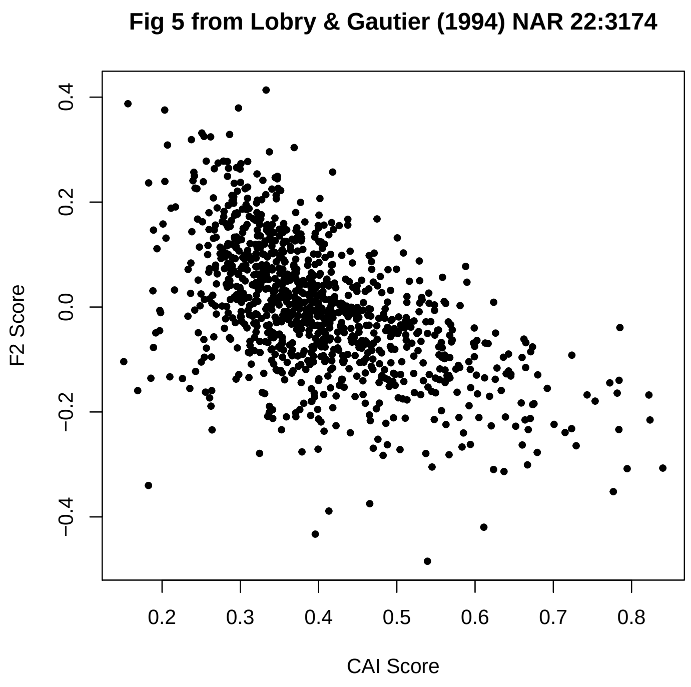


Figure 7.1: Screenshot of figure 5 from [59]. Each point represents a protein. This was to show the correlation between the codon adaptation index (CAI Score) with the second factor of correspondence analysis at the amino-acid level (F2 Score). Highly expressed genes have a high CAI value.

To illustrate the interest of synonymous codon usage analyses, let's use now a more realistic example. In [59] there was an assertion stating that selection for translation optimisation in *Escherichia coli* was also visible at the amino-acid level. The argument was in figure 5 of the paper (*cf* fig 7.1), that can be reproduced[2] with the following R code:

```
ec <- read.table(
     file = "ftp://pbil.univ-lyon1.fr/pub/datasets/NAR94/data.txt",
     header = TRUE,
     row.names = 1)
ec.coa <- dudi.coa(ec, scann = FALSE, nf=3)
F2 <- ec.coa$li[,2]
tmp <- read.table(
     file = "ftp://pbil.univ-lyon1.fr/pub/datasets/NAR94/ecoli999.cai")
cai <- exp(tmp$V2)
if(cor(cai,F2) > 0) F2 <- -F2
plot(cai, F2, pch=20, xlab="CAI Score", ylab="F2 Score",
     main="Fig 5 from Lobry & Gautier (1994) NAR 22:3174")
```



So, there was a correlation between the CAI (Codon Adaptation Index [87]) and the second factor for amino-acid composition variability. However, this is not completely convincing because the CAI is not completely independent of the amino-acid composition of the protein. Let's use within amino-acid correspondence analysis to remove the amino-acid effect. Here is a commented step-by-step analysis:

```
data(ec999)
class(ec999)
```

```
[1] "list"
```

```
names(ec999)[1:10]
```

[2] the code to reproduce all figures from [59] is available at `http://pbil.univ-lyon1.fr/members/lobry/repro/nar94/`.

```
 [1] "ECFOLE.FOLE"     "ECMSBAG.MSBA"    "ECNARZYW-C.NARV" "ECNARZYW-C.NARW"
 [5] "ECNARZYW-C.NARY" "ECNARZYW-C.NARZ" "ECNIRBC.NIRB"    "ECNIRBC.NIRD"
 [9] "ECNIRBC.NIRC"    "ECNIRBC.CYSG"
 ec999[[1]][1:50]
 [1] "a" "t" "g" "c" "c" "a" "t" "c" "a" "c" "t" "c" "a" "g" "t" "a" "a" "a"
[19] "g" "a" "a" "g" "c" "g" "g" "c" "c" "c" "t" "g" "g" "t" "t" "c" "a" "t"
[37] "g" "a" "a" "g" "c" "g" "t" "t" "a" "g" "t" "t" "g" "c"
```

This is to load the data from [59] which is available as `ec999` in the **`seqinR`** package. The letters `ec` are for the bacterium *Escherichia coli* and the number `999` means that there were 999 coding sequences available from this species at that time. The class of the object `ec999` is a list, which names are the coding sequence names, for instance the first coding sequence name is `ECFOLE.FOLE`. Each element of the list is a vector of character, we have listed just above the 50 first character of the first coding sequence of the list with `ec999[[1]][1:50]`, we can see that there is a start codon (ATG) at the beginning of the first coding sequence.

```
 ec999.uco <- lapply(ec999, uco) # compute codon usage for all CDS
 class(ec999.uco)
[1] "list"
 class(ec999.uco[[1]])
[1] "table"
 ec999.uco[[1]]
aaa aac aag aat aca acc acg act aga agc agg agt ata atc atg att caa cac cag
  9   5   2   4   2   8   8   1   0   2   0   4   0   9   8   6   2   3   7
cat cca ccc ccg cct cga cgc cgg cgt cta ctc ctg ctt gaa gac gag gat gca gcc
  7   1   1   6   0   1   7   1   4   1   3  13   3  12   3   1   9   1   6
gcg gct gga ggc ggg ggt gta gtc gtg gtt taa tac tag tat tca tcc tcg tct tga
  7   5   2   3   0   4   0   5   9   4   0   2   0   2   2   3   2   1   1
tgc tgg tgt tta ttc ttg ttt
  1   0   1   1   4   2   3
```

This is to compute the codon usage, that is how many times each codon is used in each coding sequence. Because `ec999` is a list, we use the function `lapply()` to apply the same function, `uco()`, to all the elements of the list and we store the result in the object `ec999.uco`. The object `ec999.uco` is a list too, and all its elements belong to the class table.

```
 df <- as.data.frame(lapply(ec999.uco, as.vector)) # put it in a dataframe
 dim(df)
[1]  64 999
 df[1:5,1:5]
  ECFOLE.FOLE ECMSBAG.MSBA ECNARZYW.C.NARV ECNARZYW.C.NARW ECNARZYW.C.NARY
1           9           15               2               6              23
2           5           18               2               4              16
3           2            8               1               3               4
4           4            3               2               2               4
5           2            3               1               1               0
```

This is to put the codon usage into a data.frame. Note that the codons are in row and the coding sequences are in columns. This is more convenient for the following because groups for within and between analyses are usually handled by row.

```
 row.names(df) <- names(ec999.uco[[1]]) # add codon names
 df[1:5,1:5]
    ECFOLE.FOLE ECMSBAG.MSBA ECNARZYW.C.NARV ECNARZYW.C.NARW ECNARZYW.C.NARY
aaa           9           15               2               6              23
aac           5           18               2               4              16
aag           2            8               1               3               4
aat           4            3               2               2               4
aca           2            3               1               1               0
```

This is to keep a trace of codon names, just in case we would like to re-order the dataframe `df`. This is important because we can now play with the data at will without loosing any critical information.

```
 ec999.coa <- dudi.coa(df = df, scannf = FALSE) # run global correspondence analysis
 ec999.coa
Duality diagramm
class: coa dudi
$call: dudi.coa(df = df, scannf = FALSE)

$nf: 2 axis-components saved
$rank: 63
eigen values: 0.05536 0.02712 0.02033 0.01884 0.01285 ...
  vector length mode    content
1 $cw    999    numeric column weights
2 $lw    64     numeric row weights
3 $eig   63     numeric eigen values

  data.frame nrow ncol content
1 $tab       64   999  modified array
2 $li        64   2    row coordinates
3 $l1        64   2    row normed scores
4 $co        999  2    column coordinates
5 $c1        999  2    column normed scores
other elements: N
```

This is to run global correspondence analysis of codon usage. We have set the `scannf` parameter to `FALSE` because otherwise the eigenvalue bar plot is displayed for the user to select manually the number of axes to be kept.

```
 facaa <- as.factor(aaa(translate(s2c(c2s(rownames(df)))))) # define a factor for amino-acids
 facaa
 [1] Lys Asn Lys Asn Thr Thr Thr Thr Arg Ser Arg Ser Ile Ile Met Ile Gln His
[19] Gln His Pro Pro Pro Pro Arg Arg Arg Arg Leu Leu Leu Leu Glu Asp Glu Asp
[37] Ala Ala Ala Ala Gly Gly Gly Gly Val Val Val Val Stp Tyr Stp Tyr Ser Ser
[55] Ser Ser Stp Cys Trp Cys Leu Phe Leu Phe
21 Levels: Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe ... Val
```

This is to define a factor for amino-acids. The function `translate()` use by default the standard genetic code and this is OK for *E. coli*.

```
 ec999.syn <- wca(x = ec999.coa, fac = facaa, scannf = FALSE) # run synonymous codon usage analysis
 ec999.syn
Within analysis
call: wca.dudi(x = ec999.coa, fac = facaa, scannf = FALSE)
class: within dudi

$nf (axis saved) : 2
$rank:  43
$ratio:  0.6438642

eigen values: 0.04855 0.0231 0.01425 0.007785 0.006748 ...

  vector length mode    content
1 $eig   43     numeric eigen values
2 $lw    64     numeric row weigths
3 $cw    999    numeric col weigths
4 $tabw  21     numeric class weigths
5 $fac   64     numeric factor for grouping

  data.frame nrow ncol content
1 $tab       64   999  array class-variables
2 $li        64   2    row coordinates
3 $l1        64   2    row normed scores
4 $co        999  2    column coordinates
5 $c1        999  2    column normed scores
6 $ls        64   2    supplementary row coordinates
7 $as        2    2    inertia axis onto within axis
```

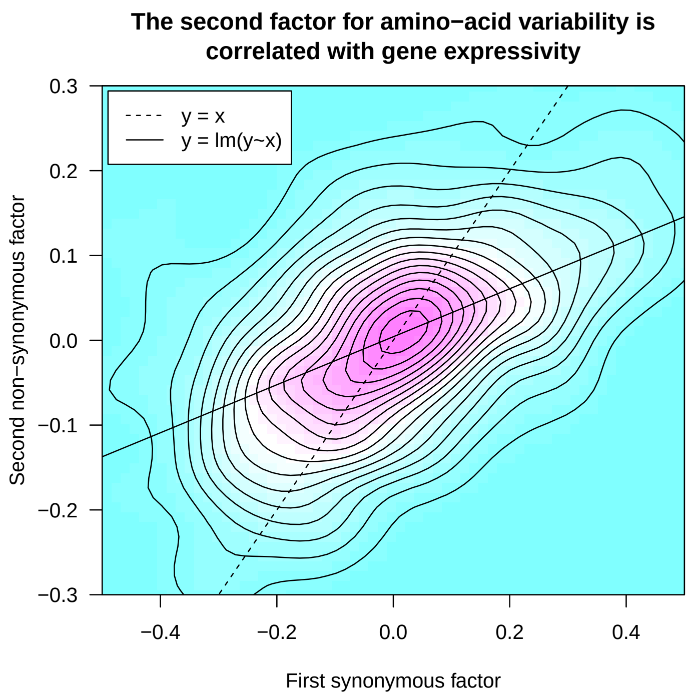
This is to run the synonymous codon usage analysis. The value of the `ratio` component of the object `ec999.syn` shows that most of the variability is at the synonymous level, a common situation in codon usage studies.

```
ec999.btw <- bca(x = ec999.coa, fac  = facaa, scannf = FALSE) # run non-sysnonymous codon usage analysis <=> amino-a
ec999.btw
```

```
Between analysis
call: bca.dudi(x = ec999.coa, fac = facaa, scannf = FALSE)
class: between dudi

$nf (axis saved) : 2
$rank:  20
$ratio:  0.3561358

eigen values: 0.01859 0.0152 0.01173 0.01051 0.008227 ...

  vector length mode    content
1 $eig   20     numeric eigen values
2 $lw    21     numeric group weigths
3 $cw    999    numeric col weigths

  data.frame nrow ncol content
1 $tab       21   999  array class-variables
2 $li        21   2    class coordinates
3 $l1        21   2    class normed scores
4 $co        999  2    column coordinates
5 $c1        999  2    column normed scores
6 $ls        64   2    row coordinates
7 $as        2    2    inertia axis onto between axis
```

This is to run the non-sysnonymous codon usage analysis, or amino-acid usage analysis.

```
x <- ec999.syn$co[,1]
y <- ec999.btw$co[,2]
if(cor(x,y) < 0) y <- -y
kxy <- kde2d(x,y, n = 100)
nlevels <- 25
breaks <- seq(from = min(kxy$z), to = max(kxy$z), length = nlevels + 1)
col <- cm.colors(nlevels)
image(kxy, breaks = breaks, col = col, xlab = "First synonymous factor",
ylab = "Second non-synonymous factor", xlim = c(-0.5, 0.5),
ylim  = c(-0.3, 0.3), las = 1,
main = "The second factor for amino-acid variability is\ncorrelated with gene expressivity")
contour(kxy, add = TRUE, nlevels = nlevels, drawlabels = FALSE)
box()
abline(c(0,1), lty=2)
abline(lm(y~x))
legend("topleft", lty = c(2,1), legend = c("y = x", "y = lm(y~x)"), inset = 0.01, bg = "white")
```



This is to plot the whole thing. We have extracted the coding sequences coordinates on the first synonymous factor and the second non-synonymous factor within `x` and `y`, respectively. Because we have many points, we use the two-dimensional kernel density estimation provided by the function `kde2d()` from package `MASS`.

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 16:40:24 2016
- LaTeX compilation time was: June 2, 2016

	aaa	a	prec	p	h	tot	gc
1	Ala	A	pyr	1	5	12	h
2	Cys	C	3pg	7	9	25	m
3	Asp	D	oaa	1	6	13	m
4	Glu	E	akg	3	6	15	m
5	Phe	F	2 pep, eryP	13	19	52	l
6	Gly	G	3pg	2	5	12	h
7	His	H	penP	20	9	38	m
8	Ile	I	pyr, oaa	4	14	32	l
9	Lys	K	oaa, pyr	4	13	30	l
10	Leu	L	2 pyr, acCoA	3	12	27	l
11	Met	M	oaa, Cys, -pyr	10	12	34	m
12	Asn	N	oaa	3	6	15	l
13	Pro	P	akg	4	8	20	h
14	Gln	Q	akg	4	6	16	m
15	Arg	R	akg	11	8	27	h
16	Ser	S	3pg	2	5	12	m
17	Thr	T	oaa	3	8	19	m
18	Val	V	2 pyr	2	11	23	m
19	Trp	W	2 pep, eryP, PRPP, -pyr	28	23	74	m
20	Tyr	Y	eryP, 2 pep	13	18	50	l

Table 7.4: Aerobic cost of amino-acids in *Escherichia coli* and G+C classes to be loaded with `data(aacost)`.

CHAPTER 8

Nonparametric statistics

Palmeira, L. Lobry, J.R.

8.1 Introduction

Nonparametric statistical methods were initially developped to study variables for which little or nothing is known concerning their distribution. This makes them particularly suitable for statistical analysis of biological sequences, in particular for the study of over- and under-representation of k-letter words (*cf* section number 8.3).

8.2 Elementary nonparametric statistics

8.2.1 Introduction

Those rank statistics are those that were available under the ANALSEQ software [38, 30]. Formulae were taken from [11]. We consider here a sequence of booleans, for instance:

```
 (x <- rep(c(T,F),10))
```

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

We note N the total number of elements in the vector:

```
 (N <- length(x))
```

```
[1] 20
```

We note M the total number of TRUE elements in the vector:

```
 (M <- sum(x))
```

```
[1] 10
```

We note ω the ranks of TRUE elements:

```
(omega <- which(x))

[1]  1  3  5  7  9 11 13 15 17 19
```

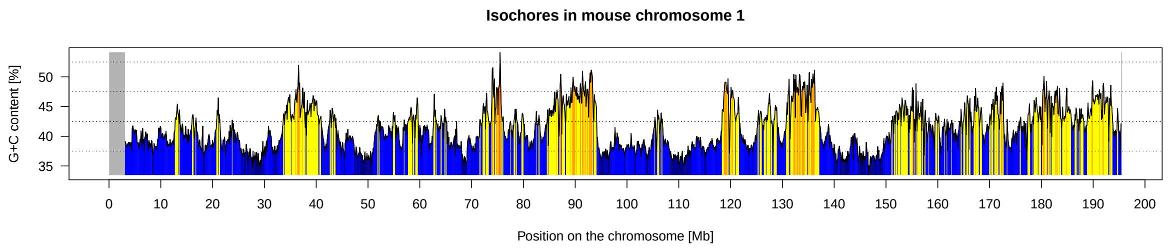
With one exception[1], the statistics names are the same as in the ANALSEQ software.

As a practical application, we want to study the isochore structure in *Mus musculus* chromosome 1 using non-overlapping windows of 100 kb. Data were computed this way:

```
choosebank("ensembl")
n <- 196 # Mus musculus chromosome size in Mb (ceil of)
myseq <- gfrag("MOUSE_1", 1, n*10^6)
res <- rep(-1, 10*n)
for(w in seq(1, nchar(myseq), by = 10^5)){
  res[i] <- GC(s2c(substr(myseq, start = w, stop = w + 10^5 - 1)))
  i <- i + 1
}
res <- res[res >= 0]
res[res == 0] <- NA
res <- 100*res
closebank()
save(res, file = "chr1.RData")
```

The folowing representation follows the conventions used in Fig 2 from [71].

```
load("chr1.RData")
n <- length(res)
xx <- seq_len(n)/10
plot(xx, res, type = "l", las = 1,
ylab = "G+C content [%]",
main = "Isochores in mouse chromosome 1", xaxt = "n",
xlab = "Position on the chromosome [Mb]")
axis(1, at = seq(0,200,by=10))
breaks <- c(0, 37.5, 42.5, 47.5, 52.5, 100)
cut(res, breaks = breaks,
labels = c("darkblue", "blue", "yellow", "orange", "red"),
ordered=T) -> lev
segments(x0 = xx, y0 = min(res, na.rm = TRUE), x1=xx, y1=res,col=as.character(lev),lend="butt")
segments(x0 = xx[is.na(res)], y0 = min(res,na.rm=T), x1 = xx[is.na(res)], y1=max(res,na.rm=T),
col = grey(0.7))
lines(xx, res)
abline(h=breaks, lty = 3)
```

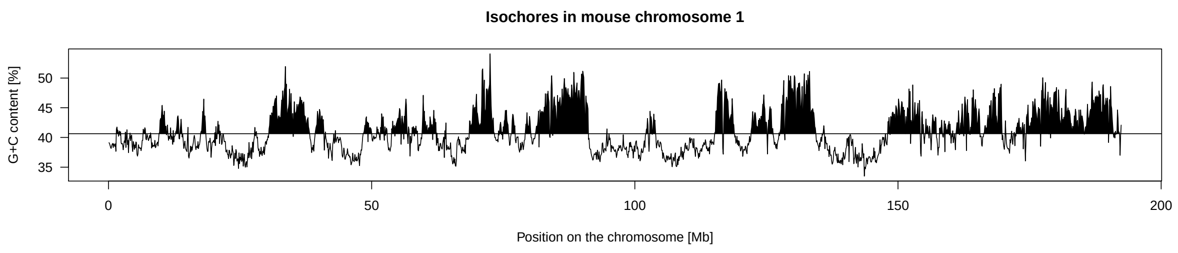


The gray area represent undocumented parts of the chromosome, we won't consider them in the following and recode the sequence in TRUE and FALSE if the values are above or below the median, respectively:

```
yy <- res[!is.na(res)]
n <- length(yy)
xx <- seq_len(n)/10
hline <- median(yy)
plot (yy ~ xx, type="n", axes=FALSE, ann=FALSE)
polygon(c(xx[1], xx, xx[n]), c(min(yy), yy, min(yy)), col = "black", border=NA)
```

[1]for GC in the ANALSEQ software renamed as CC here

```
usr <- par("usr")
rect(usr[1], usr[3], usr[2], hline, col="white", border=NA)
lines(xx, yy)
abline (h=hline)
box()
axis(1)
axis(2, las = 1)
title(xlab = "Position on the chromosome [Mb]", ylab = "G+C content [%]",
main = "Isochores in mouse chromosome 1")
```



Our logical vector is therefore defined as follows:

```
appli <- yy > median(yy)
head(appli)
```

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

```
tail(appli)
```

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

8.2.2 Rank sum

The statistic SR is the sum of the ranks of TRUE elements.

$$\mathrm{SR} = \sum_{j \in \omega} j$$

```
**-***-----------      ==> SR low  (18)
---------***--***      ==> SR high (81)
```

$$\begin{aligned}
\mathrm{E(SR)} &= \frac{\mathrm{M(N+1)}}{2} \\
\mathrm{V(SR)} &= \frac{\mathrm{M(N+1)(N-M)}}{12}
\end{aligned}$$

```
SR <- function(bool, N = length(bool), M = sum(bool)){
        stopifnot(is.logical(bool))
        SR <- sum(seq_len(N)[bool])
        E <- M*(N + 1)/2
        V <- M*(N + 1)*(N - M)/12
        return(list(SR = SR, stat = (SR - E)/sqrt(V)))
}
SR(s2c("**-***-----------") == "*")
```

```
$SR
[1] 18

$stat
[1] -2.84605
```

```
SR(s2c("---------***--***") == "*")
```

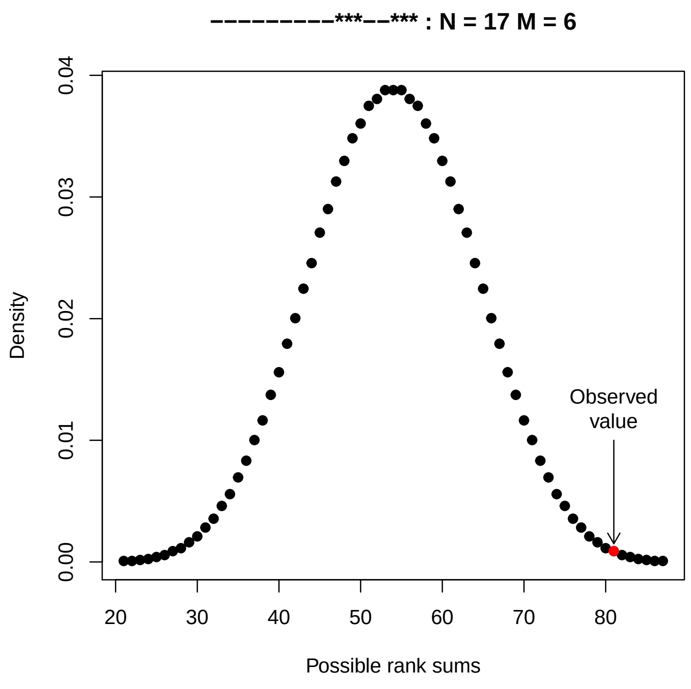
```
$SR
[1] 81

$stat
[1] 2.713602
```

Here is a way to obtain the same result using the standard R `wilcox.test()` function to make a Wilcoxon's rank sum test [98]:

```
SRh <- s2c("---------***--***") == "*"
x <- seq_len(length(SRh))
x[!SRh] <- -1*x[!SRh]
wilcox.test(x)$statistic
 V
81
```

The probabilities for all possibe outcomes for the rank sums are given by `dwilcox()` but note the $\frac{M(M+1)}{2}$ shift:

```
m <- sum(SRh)
n <- length(SRh) - m
dwilcox(x = 0:(n*m), m = m, n = n) -> pdf
plot(x = 0:(m*n) + m*(m+1)/2, y = pdf, xlab = "Possible rank sums",
ylab = "Density", main = paste("---------***--*** : N =", length(SRh), "M =", sum(SRh)),
pch = 19)
points(SR(SRh)$SR, dwilcox(x = SR(SRh)$SR - m*(m+1)/2, m = m, n = n), col = "red",
pch = 19)
arrows(x0 = SR(SRh)$SR, y0 = 0.01, x1 = SR(SRh)$SR, y1 = 0.0015, length = 0.1)
text(SR(SRh)$SR, 0.01, "Observed\nvalue", pos = 3)
```



Real case application

```
SR(appli)$stat
[1] 10.73121
```

The rank sum is higher than expected at random, there is an excess of GC rich regions at the rigth end (3'end) of the chromosome.

8.2.3 Rank variance

This statistics is the variance of ranks:

$$\text{VR} = \sum_{j \in \omega} (j - \frac{\text{N}+1}{2})^2$$

```
------***-------     ==> VR low  (6)
***----------****    ==> VR high (323)
```

$$\begin{aligned} \text{E(VR)} &= \frac{\text{M(N}+1)(\text{N}-1)}{12} \\ \text{V(VR)} &= \frac{\text{M(N}-\text{M)(N}+1)(\text{N}+2)(\text{N}-2)}{180} \end{aligned}$$

```
VR <- function(bool, N = length(bool), M = sum(bool)){
        stopifnot(is.logical(bool))
        VR <- sum ((seq_len(N)[bool] - (N + 1)/2)^2)
        E <- (M*(N + 1)*(N - 1))/12
        V <- (M*(N - M)*(N + 1)*(N + 2)*(N - 2))/180
        return(list(VR = VR, stat = (VR - E)/sqrt(V)))
}
VR(s2c("------***-------") == "*")
```

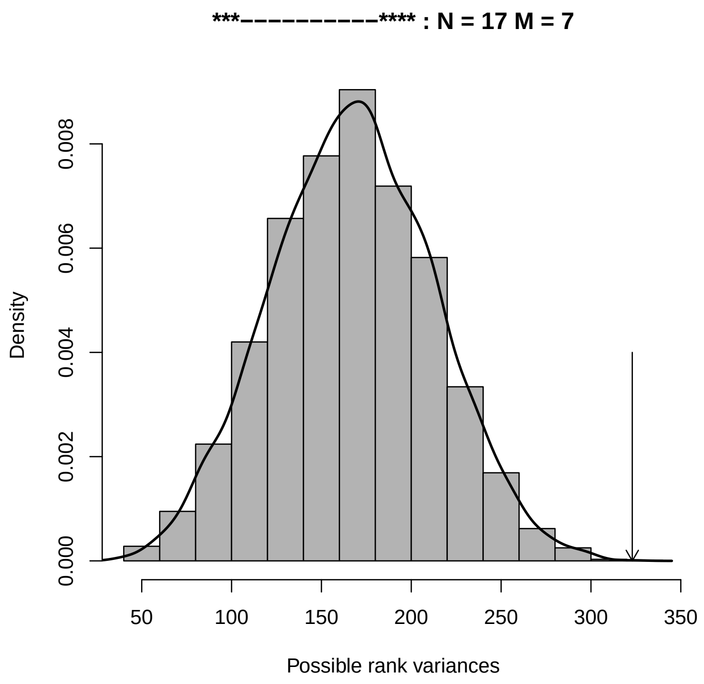
```
$VR
[1] 6

$stat
[1] -2.33786
```

```
VR(s2c("***----------****") == "*")
```

```
$VR
[1] 323

$stat
[1] 3.470246
```

We can use simulations to have an idea of the probability density function of the rank variance, for instance:

```
VRh <- s2c("***----------****") == "*"
replicate(5000, VR(sample(VRh))$VR) -> simVR
hist(simVR, col = grey(0.7), main = paste("***----------**** : N =",
length(VRh), "M =", sum(VRh)), xlab = "Possible rank variances", proba = TRUE)
lines(density(simVR), lwd = 2)
arrows(VR(VRh)$VR, 0.004, VR(VRh)$VR, 0, le = 0.1)
```



Real case application

```
VR(appli)$stat
[1] 4.43128
```

The variance of ranks is higher than expected at random, there is an excess of GC rich regions at the telomeric ends of the chromosome.

8.2.4 Clustering around the observed centre

Let note $\mathrm{C}(\omega)$ the observed centre:

$$\mathrm{C}(\omega) = \begin{cases} \omega\left(\frac{\mathrm{M}+1}{2}\right) & \text{if M is odd} \\ \omega\left(\frac{\mathrm{M}}{2}+1\right) & \text{if M is even} \end{cases}$$

The statistic CC[2] is the dispersion around $\mathrm{C}(\omega)$ is defined by:

$$\mathrm{CC} = \sum_{j \in \omega} |j - \mathrm{C}(\omega)|$$

```
---*****---------    ==> CC low  (6)
***-------***----    ==> CC high (30)
```

Noting $\lfloor x \rfloor$ the floor of x, we have:

$$\mathrm{E(CC)} = \frac{(\mathrm{N}+1)\lfloor\frac{\mathrm{M}}{2}\rfloor\lfloor\frac{\mathrm{M}+1}{2}\rfloor}{\mathrm{M}+1}$$

[2] the original notation was GC in the ANALSEQ software, we use CC instead to avoid a collision with the `GC()` function to compute the G+C content.

and

$$V(CC) = \begin{cases} \frac{(M-1)(M+3)(N+1)(N-M)}{48(M+2)} & \text{if M is odd} \\ \frac{M(N+1)(N-M)(M^2+2*M+4)}{48(M+1)^2} & \text{if M is even} \end{cases}$$

```
CC <- function(bool, N = length(bool), M = sum(bool)){
        stopifnot(is.logical(bool))
        C <- median(seq_len(N)[bool])
        GC <- sum(abs(seq_len(N)[bool] - C))
        E <- ((N + 1)*floor(M/2)*floor((M + 1)/2))/(M + 1)
        if(M %% 2 == 1)
          V <- ((M - 1)*(M + 3)*(N + 1)*(N - M))/(48*(M + 2))
        else
          V <- (M*(N + 1)*(N - M)*(M^2 + 2*M + 4))/(48*(M + 1)^2)
        return(list(GC = GC, stat = (GC - E)/sqrt(V)))
}
CC(s2c("---*****---------") == "*")
```

```
$GC
[1] 6

$stat
[1] -2.645751
```

```
CC(s2c("***-------***----") == "*")
```

```
$GC
[1] 30

$stat
[1] 1.337987
```

Real case application

```
CC(appli)$stat
```

```
[1] 3.25226
```

The dispersion around the observed centre is higher than expected at random, there is a trend for GC rich sequences to avoid this centre.

8.2.5 Number of runs

The statistics NS is the number of runs in the sequence:

```
--***---***--***-    ==> NS low  (7)
-*-*-*-*-*-*-*-*-    ==> NS high (17)
```

$$\begin{aligned} E(NS) &= \frac{2M(N-M)}{N} + 1 \\ V(NS) &= \frac{2M(N-M)(2M(N-M)-N)}{N^2(N-1)} \end{aligned}$$

```
NS <- function(bool, N = length(bool), M = sum(bool)){
        stopifnot(is.logical(bool))
        NS <- length(rle(bool)$lengths)
        DMNmM <- 2*M*(N - M)
        E <- DMNmM/N + 1
        V <- (DMNmM*(DMNmM - N))/(N*N*(N - 1))
        return(list(NS = NS, stat = (NS - E)/sqrt(V)))
}
NS(s2c("--***---***--***-") == "*")
```

```
$NS
[1] 7

$stat
[1] -1.242299
```

```
NS(s2c("-*-*-*-*-*-*-*-*-*-") == "*")
```

```
$NS
[1] 17

$stat
[1] 3.786054
```

The same result can be obtained with the function `runs.test()` from package **tseries** [96] this way:

```
library(tseries)
s2c("-*-*-*-*-*-*-*-*-*-") == "*" -> NSh
tseries::runs.test(as.factor(NSh))$statistic
```

```
Standard Normal
       3.786054
```

Real case application

```
NS(appli)$stat
```

```
[1] -34.25156
```

The number of runs is much less than expected at random, there is a trend for GC rich sequences to aggregate in consecutive runs: this is the isochore structure.

8.2.6 Multiple clusters

The statistics GM is the variance of the length n_k of FALSE runs (including runs of length zero) between two TRUE. Let note:

- $n_k(\omega)$ the number of FALSE between $\omega(k-1)$ and $\omega(k)$ for $2 \leq k \leq \mathrm{M}$.
- $n_1(\omega)$ the number of FALSE before $\omega(1)$.
- $n_{\mathrm{M}+1}(\omega)$ the number of FALSE after $\omega(\mathrm{M})$.

$$\mathrm{GM} = \frac{1}{\mathrm{M}} \sum_{i=1}^{\mathrm{M}+1} \left(n_i(\omega) - \frac{\mathrm{N}-\mathrm{M}}{\mathrm{M}+1} \right)^2$$

```
-*-*-*-*-*-*-*-*-*-    ==> GM low  (0)
***------***-***-      ==> GM high (3.5)
```

$$\begin{aligned}
\mathrm{E(GM)} &= \frac{(\mathrm{N}+1)(\mathrm{N}-\mathrm{M})}{(\mathrm{M}+1)(\mathrm{M}+2)} \\
\mathrm{V(GM)} &= \frac{4(\mathrm{N}-\mathrm{M}-1)(\mathrm{N}+1)(\mathrm{N}+2)(\mathrm{N}-\mathrm{M})}{\mathrm{M}(\mathrm{M}+2)^2(\mathrm{M}+3)(\mathrm{M}+4)}
\end{aligned}$$

```
 GM <-function(bool, N = length(bool), M = sum(bool)){
         stopifnot(is.logical(bool))
         XGM <- (N - M)/(M + 1)
         LS0 <- GM <- 0
         for(i in seq_len(N)){
                 if(bool[i]){
                         GM <- GM + (LS0 - XGM)^2
                         LS0 <- 0
                 } else {
                         LS0 <- LS0 + 1
                 }
         }
         GM <- (GM + (LS0 - XGM)^2)/M
         E <- ((N + 1)*(N - M))/((M +1)*(M + 2))
   V <- (4*(N - M - 1)*(N + 1)*(N + 2)*(N - M))/(M*(M + 2)^2*(M + 3)*(M + 4))
         return(list(GM = GM, stat = (GM - E)/sqrt(V)))
 }
 GM(s2c("-*-*-*-*-*-*-*-*-") == "*")
$GM
[1] 0

$stat
[1] -1.863782
 GM(s2c("***------***-***-") == "*")
$GM
[1] 3.511111

$stat
[1] 3.279144
```

Real case application

```
 GM(appli)$stat
[1] 353.0584
```

The number of cluster is much higher than expected at random, there is a trend for GC rich sequences to aggregate in clusters: this is again the reflect of the isochore structure in this chromosome.

8.3 Dinucleotides over- and under-representation

8.3.1 Introduction

We will briefly describe two statistics for the measure of dinucleotide over- and under-representation in sequences [41, 69], which can both be computed with **seqinR**. We will subsequently use them to answer the long-time controversial question concerning the relationship between UV exposure and genomic content in bacteria [89, 2].

8.3.2 The *rho* statistic

The ρ statistic (`rho()`), presented in [41], measures the over- and under-representation of two-letter words:

$$\rho(xy) = \frac{f_{xy}}{f_x \times f_y}$$

where f_{xy} and f_x are respectively the frequencies of dinucleotide xy and nucleotide x in the studied sequence. The underlying model of random generation considers dinucleotides to be formed according to the specific frequencies of the two nucleotides that compose it ($\rho_{xy} = 1$). Departure from this value characterizes either over- or under-representation of dinucleotide xy.

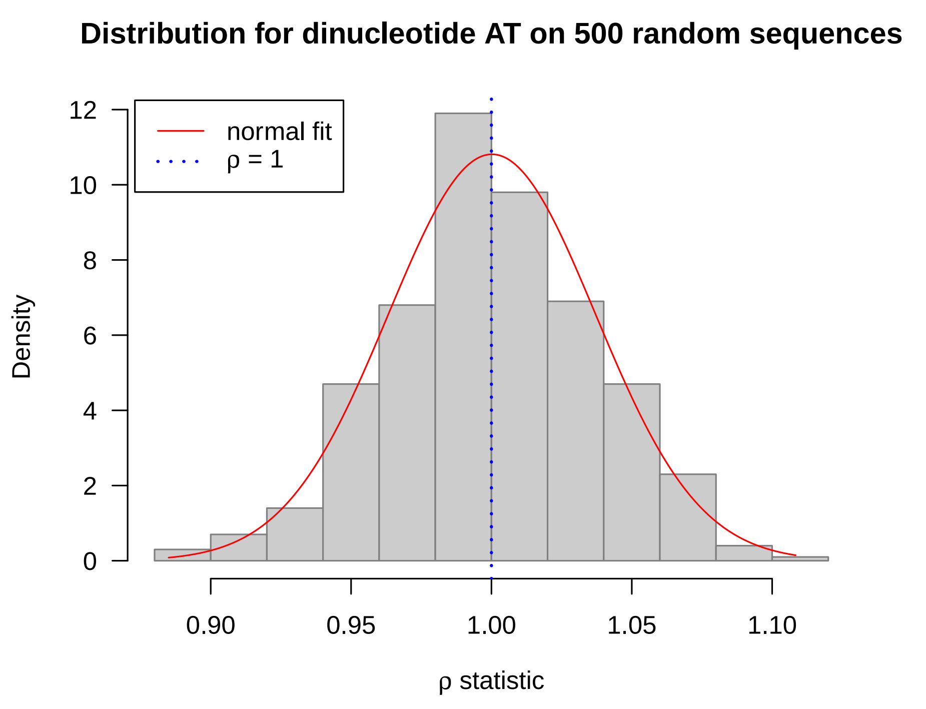


Figure 8.1: Distribution of the ρ statistic computed on 500 random sequences of length 6000. The vertical dotted line is centered on 1. The curve draws the fitted normal distribution.

We expect the ρ statistic of a randomly generated sequence to be neither over- nor under-represented. Indeed, when we compute the ρ statistic on 500 random sequences, we can fit a normal distribution which is centered on 1 (see Fig. 8.1)

```
set.seed(1)
n <- 500
di <- 4
lseq <- 6000
rhoseq <- replicate(n, rho(sample(s2c('acgt'), size = lseq, replace = TRUE)))
x <- seq(min(rhoseq[di,]), max(rhoseq[di,]), length.out = 1000)
y <- dnorm(x, mean = mean(rhoseq[di,]), sd = sd(rhoseq[di,]))
histo <- hist(rhoseq[di,], plot = FALSE)
plot(histo, freq = FALSE, xlab = expression(paste(rho, " statistic")),
  main = paste("Distribution for dinucleotide",
  toupper(labels(rhoseq)[[1]][di]), "on", n, "random sequences"),
  las = 1, col = grey(0.8), border = grey(0.5),
  ylim = c(0, max(c(y, histo$density))))
lines(x, y, lty = 1, col = "red")
abline(v = 1, lty = 3, col = "blue", lwd = 2)
legend("topleft", inset = 0.01, legend = c("normal fit",
  expression(paste(rho, " = 1"))), lty = c(1, 3),
  col = c("red", "blue"), lwd = c(1, 2))
```

The downside of this statistic, is that the model against which we compare the sequence under study is fixed. For several types of sequences, dinucleotides are far from being formed by mere chance (CDS, ...). In this case, the model used in the ρ statistic becomes trivial, and the over- or under-representations measured are mainly due to the strong constraints acting on those sequences.

8.3.3 The z-score statistic

The z-score statistic (`zscore()`) is inspired by the ρ statistic, and is defined so that several different models can be used for the determination of over- and under-representation [69]. It allows for a finer measure of over- and under-representation in sequences, according to the chosen model.

The z-score is defined as follows:

$$z_{score} = \frac{\rho_{xy} - E(\rho_{xy})}{\sqrt{Var(\rho_{xy})}}$$

where $E(\rho_{xy})$ and $Var(\rho_{xy})$ are the expected mean and variance of ρ_{xy} according to a given model that describes the sequence.

This statistic follows the standard normal distribution, and can be computed with several different models of random sequence generation based on permutations from the original sequence (`modele` argument). More details on those models can be obtained in the documentation for the `zscore()` function, by simply typing `?zscore`.

For instance, if we want to measure the over- and under-representation of dinucleotides in CDS sequences, we can use the `codon` model, which measures the over- and under-representations existing in the studied sequence once codon usage bias has been erased. For intergenic sequences, or sequences for which no good permutation model can be established, we can use the `base` model.

8.3.4 Comparing statistics on a sequence

Let's have a look at what these different statistics can show. First, we will extract a CDS sequence of *Escherichia coli*'s chromosome from the Genome Reviews database. Let's use, for instance, the following CDS:

```
choosebank("greviews")
coli <- query("coli","N=U00096_GR ET T=CDS ET K=2.3.1.79@")
sequence <- getSequence(coli$req[[1]])
annot <- getAnnot(coli$req[[1]])
closebank()
save(sequence, file = "sequence.RData")
save(annot, file = "annot.RData")
```

```
load("annot.RData")
cat(annot, sep="\n")
```

```
FT   CDS             complement(478591..479142)
FT                   /codon_start=1
FT                   /evidence="1: Evidence at protein level "
FT                   /gene_id="IGI28831209"
FT                   /gene_name="maa"
FT                   /gene_synonym="ECK0453"
FT                   /gene_synonym="ylaD"
FT                   /locus_tag="b0459"
FT                   /product="Maltose O-acetyltransferase "
FT                   /EC_number="2.3.1.79"
FT                   /function="maltose O-acetyltransferase activity "
FT                   /protein_id="AAC73561.1"
FT                   /db_xref="EMBL:AAB40214.1"
FT                   /db_xref="EMBL:CAA11147.1"
FT                   /db_xref="EcoGene:EG14239"
FT                   /db_xref="GO:0008925"
FT                   /db_xref="HOGENOM:HBG282173"
FT                   /db_xref="HOGENOM:HBG699746"
FT                   /db_xref="InterPro:IPR001451"
FT                   /db_xref="InterPro:IPR011004"
FT                   /db_xref="InterPro:IPR018357"
FT                   /db_xref="PDB:1OCX"
FT                   /db_xref="UniParc:UPI000002EA96"
FT                   /db_xref="UniProtKB/Swiss-Prot:P77791"
FT                   /transl_table=11
FT                   /translation="MSTEKEKMIAGELYRSADETLSRDRLRARQLIHRYNHSLAEEHTL
FT                   RQQILADLFGQVTEAYIEPTFRCDYGYNIFLGNNFFANFDCVMLDVCPIRIGDNCMLAP
FT                   GVHIYTATHPIDPVARNSGAELGKPVTIGNNVWIGGRAVINPGVTIGDNVVVASGAVVT
FT                   KDVPDNVVVGGNPARIIKKL"
FT                   /%(C+G)="CG<50%"
FT                   /note="C+G content in third codon positions = 47.8 % "
```

We can see that this CDS encodes a maltose O-acetyltransferase protein. We will now compare the three following nonparametric statistics:

- the ρ statistic,
- the z-score statistic with `base` model,
- and the z-score statistic with `codon` model.

The z-score statistic has been modified to incorporate an exact analytical calculation of the `base` model where the old version (seqinR 1.1-1 and previous versions) incorporated an approximation for large sequences. This has been possible with the help of Sophie Schbath [84], and the new version of this calculation can be obtained with the argument `exact` set to `TRUE` (`FALSE` being the default). The analytical solution for the `codon` model is from [24]. The following code was used to produce figure 8.2:

```
load("sequence.RData")
rhocoli <- rho(sequence)
zcolibase <- zscore(sequence, model = 'base', exact = TRUE)
zcolicodon <- zscore(sequence,model = 'codon', exact = TRUE)
par(mfrow = c(3, 1), lend = "butt", oma = c(0,0,2,0), mar = c(3,4,0,2))
col <- c("green", "blue", "orange", "red")
plot(rhocoli - 1, ylim = c(-0.5,0.5), las = 1,
  ylab = expression(rho), lwd = 10, xaxt = "n",
  col = col)
axis(1, at = 1:16, labels = toupper(words(2)))
abline(h = 0)
plot(zcolibase, ylim = c(-2.5,2.5), las = 1,
  ylab = "zscore with base model", lwd = 10, xaxt = "n",
  col = col)
axis(1, at = 1:16, labels = toupper(words(2)))
abline(h = 0)
plot(zcolicodon, ylim = c(-2.5,2.5), las = 1,
  ylab = "zscore with codon model", lwd = 10, xaxt = "n",
  col = col)
axis(1, at = 1:16, labels = toupper(words(2)))
abline(h = 0)
mtext("Comparison of the three statistics", outer = TRUE, cex = 1.5)
```

The first two panels in figure 8.2 are almost identical: this is due to the way the z-score statistic has been built. The statistic computed with the `base` model is a reflection of the ρ statistic. The difference being that the z-score follows a standard normal distribution, which makes easier the comparisons between the results from the `base` model and the ones from the `codon` model. The last pannel (z-score with `codon` model), is completely different: almost all over- and

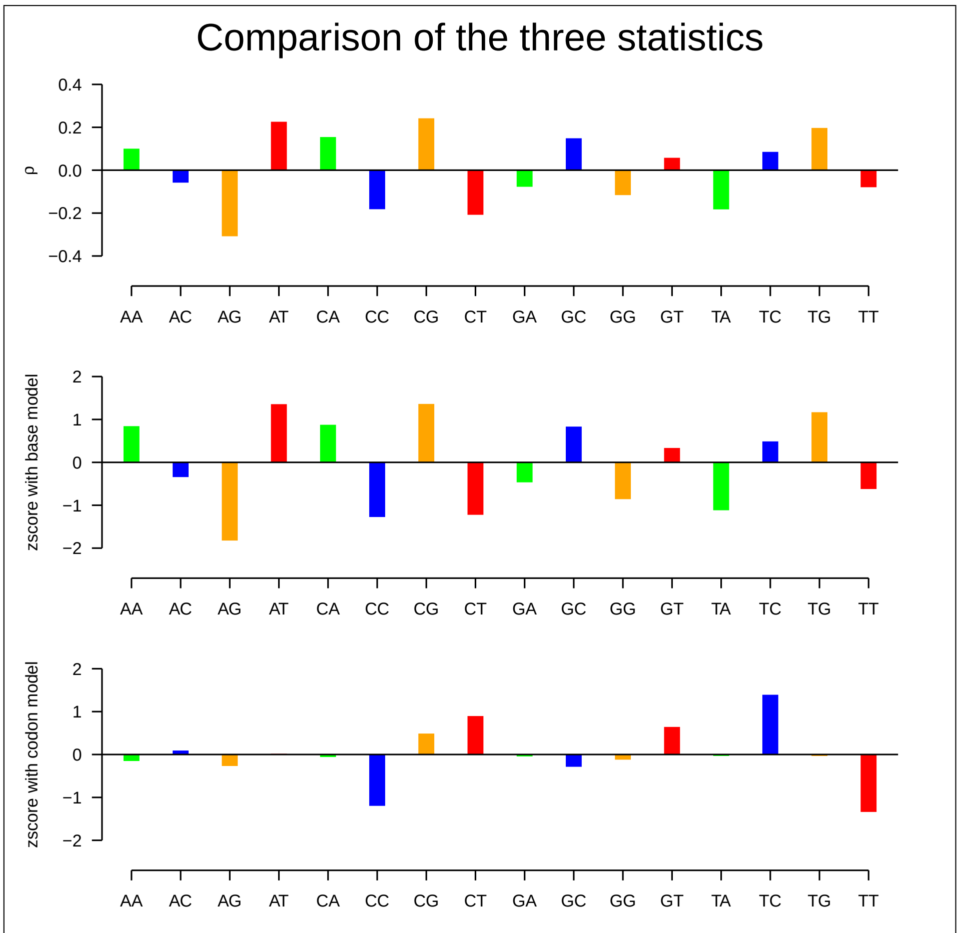


Figure 8.2: Three different non-parametric statistics (from left to right: ρ, *zscore* with `base` model, *zscore* with `codon` model), computed on the same sequence from *Escherichia coli*. In order to make the figures easily comparable, we substracted 1 to the `rho()` results, so that all 3 statistics are centered at 0.

under-representations have been erased. We can safely say that these over- and under-representations were due to codon usage bias.

On the last panel, four dinucleotides stand out: CC and TT seem rather under-represented, CT and TC rather over-represented. This means that, in this sequence, codons ending with a given pyrimidine tend to be more frequently followed by a codon starting with the other pyrimidine than expected by chance. This is not a universal feature of *Escherichia coli*, and is probably due to the amino-acid composition of this particular sequence. It seemed a funny example, as the following part will also relate to pyrimidine dinucleotides. However, what we see on this CDS from *Escherichia coli* has nothing to do with what follows...

8.4 UV exposure and dinucleotide content

In the beginning of the 1970's, two contradictory papers considered the question of the impact of UV exposure on genomic content. Both papers had strong arguments for either side, and the question remained open until recently [69].

8.4.1 The expected impact of UV light on genomic content

On this controversy, the known facts are: pyrimidine dinucleotides (CC, TT, CT and TC) are the major DNA target for UV-light [85]; the sensitivities of the four pyrimidine dinucleotides to UV wavelengths differ and depend on the micro-organism [85]:

	G+C content	CC (%)	CT + TC (%)	TT (%)
Haemophilus influenzae	62	5	24	71
Escherichia coli	50	7	34	59
Micrococcus lysodeikticus	30	26	55	19

Table 8.1: Proportion of dimers formed in the DNA of three bacteria after irradiation with 265 nm UV light. Table adapted from [85].

The hypothesis presented by Singer and Ames [89] is that pyrimidine dinucleotides are avoided in light-exposed micro-organisms. At the time, only G+C content is available, and – based exclusively on the sensitivity of the four pyrimidine dinucleotides in an *Escherichia coli* chromosome – they hypothesize that a high G+C will result in less pyrimidine target. Indeed, they find that bacteria exposed to high levels of UV have higher G+C content than the others. Bak *et al.* [2] strongly criticize their methodology, but no clear cut answer is achieved.

In an *Escherichia coli* chromosome, it is true that a sequence with a high G+C content will contain few phototargets: the following code was used to produce figure 8.3.

```
worstcase <- function(gc){
  c <- gc
  t <- (1-gc)
  (0.59*t*t + 0.34*t*c + 0.07*c*c)/2
}
randomcase <- function(gc){
  c <- gc/2
  t <- (1-gc)/2
  0.59*t*t + 0.34*t*c + 0.07*c*c
}
bestcase <- function(gc){
  c <- (gc)/2
  t <- (1-gc)/2
  if ((c + t) <= 0.5){
    0
  } else {
  c <- (c + t - 0.5)/2
  t <- (c + t - 0.5)/2
  0.59*t*t + 0.34*t*c + 0.07*c*c
  }
}
xval <- seq(from = 0, to = 100, length = 100)
sapply(xval/100, randomcase) -> yrand
sapply(xval/100, worstcase) -> yworst
sapply(xval/100, bestcase) -> ybest
plot(xval, 100*yworst, las = 1, type = "l", lwd = 2, lty = 1 ,
xlab = "G+C content [%]",
ylab = "Phototargets weighted density [%]",
main = "Estimated as in Escherichia coli chromosome",
ylim = c(0, max(100*yworst)))
points(xval,100*yrand,type='l',lwd=2,lty=2)
points(xval,100*ybest,type='l',lwd=2,lty=3)
abline(v=c(25,75),lty=2)
arrows(25, 25, 75, 25, code = 1,le = 0.1)
arrows(25, 25, 75, 25,code = 2,le = 0.1)
text(50,25,"Biological range",pos=3)
```

In a *Micrococcus lysodeikticus* sequence (the following code was used to produce figure 8.4), we can see that this is no longer true...

```
worstcase <- function(gc){
  c <- gc
  t <- (1-gc)
  (0.19*t*t + 0.55*t*c + 0.26*c*c)/2
}
randomcase <- function(gc){
  c <- gc/2
  t <- (1-gc)/2
  0.19*t*t + 0.55*t*c + 0.26*c*c
}
bestcase <- function(gc){
  c <- (gc)/2
  t <- (1-gc)/2
  if ((c + t) <= 0.5){
    0
  } else {
  c <- (c + t - 0.5)/2
  t <- (c + t - 0.5)/2
  0.19*t*t + 0.55*t*c + 0.26*c*c
  }
}
xval <- seq(from = 0, to = 100, length = 100)
sapply(xval/100, randomcase) -> yrand
sapply(xval/100, worstcase) -> yworst
sapply(xval/100, bestcase) -> ybest
plot(xval, 100*yworst, las = 1, type = "l", lwd = 2, lty = 1 ,
xlab = "G+C content [%]",
ylab = "Phototargets weighted density [%]",
main = "Estimated as in Micrococcus lysodeikticus chromosome",
ylim = c(0, max(100*yworst)))
points(xval,100*yrand,type='l',lwd=2,lty=2)
points(xval,100*ybest,type='l',lwd=2,lty=3)
abline(v=c(25,75),lty=2)
arrows(25, 25, 75, 25, code = 1,le = 0.1)
arrows(25, 25, 75, 25,code = 2,le = 0.1)
text(50,25,"Biological range",pos=3)
```

These two figures (figure 8.3 and 8.4) show that the density of phototargets depends on:

- the degree of aggregation of pyrimidine dinucleotides in the sequence,

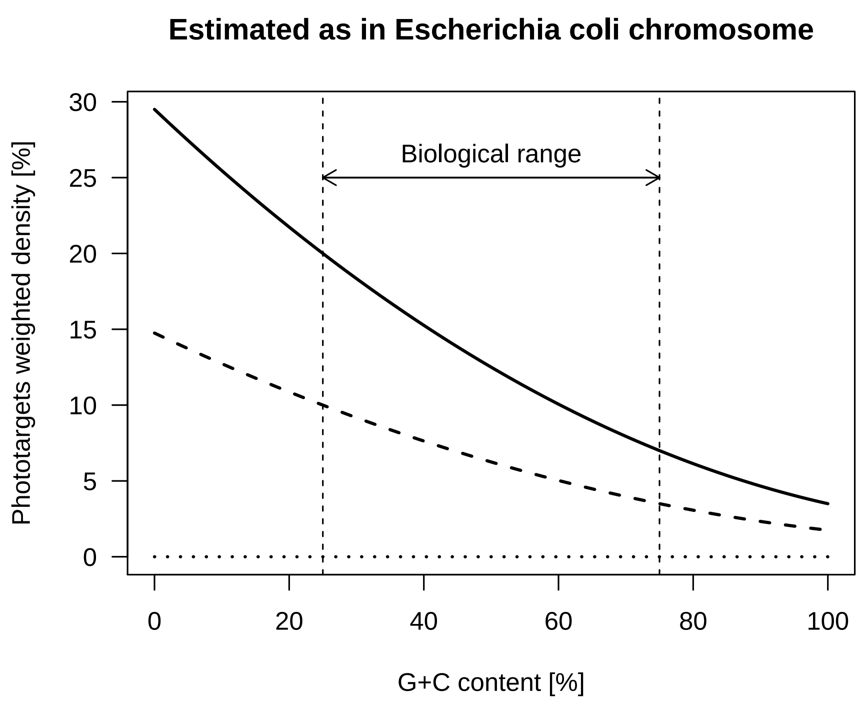


Figure 8.3: Density of phototargets, weighted by their frequency in the *Escherichia coli* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.59 * f_{tt} + 0.34 * (f_{tc} + f_{ct}) + 0.07 * f_{cc}$ (where f_{xy} is the frequency of dinucleotide xy in the specified genome). Three models of random genomes are analyzed. In the worst case (solid curve), the genome is the concatenation of a sequence of pyrimidines and a sequence of purines: all pyrimidines are involved in a pyrimidine dinucleotide. In the best case (dotted curve), the genome is an unbroken succession of pyrimidine-purine dinucleotides: no pyrimidine is involved in a pyrimidine dinucleotide. In the "random case" (dashed curve), the frequency of a pyrimidine dinucleotide is the result of chance ($f_{xy} = f_x \times f_y$).

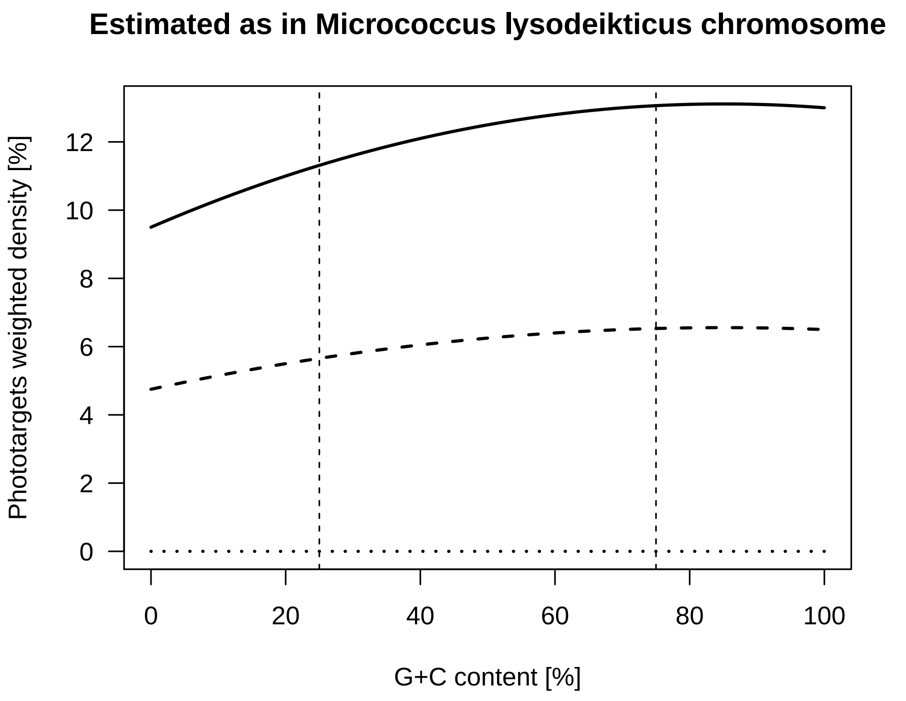


Figure 8.4: Density of phototargets, weighted by their frequency in the *Micrococcus lysodeikticus* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.19 * f_{tt} + 0.55 * (f_{tc} + f_{ct}) + 0.26 * f_{cc}$. See figure 8.3 for more details.

- the sensitivities of the four pyrimidine dinucleotides.

Instead of looking at G+C content, which is an indirect measure of the impact of UV exposure on genomic content, let us look at pyrimidine dinucleotide content.

Are CC, TT, CT and TC dinucleotides avoided in light-exposed bacteria?

8.4.2 The measured impact of UV light on genomic content

On all available genomes (as retrieved from Genome Reviews database on June 16, 2005), we have computed the mean of the z-score with the `base` model on all intergenic sequences, and the mean of the z-score with the `codon` model on all CDS. The results show that there is no systematic under-representation of none of the four pyrimidine dinucleotides (see figure 8.5 produced by the following code).

```
data(dinucl)
par(mfrow = c(2, 2), mar = c(4,4,0.5,0.5)+0.1)
myplot <- function(x){
  plot(dinucl$intergenic[, x], dinucl$coding[, x],
  xlab = "intergenic", ylab = "coding",
  las = 1, ylim = c(-6, 4),
  xlim = c(-3, 3), cex = 0)
  rect(-10,-10,-1.96,10,col="yellow", border = "yellow")
  rect(1.96,-10,10,10,col="yellow", border = "yellow")
  rect(-10,-10,10,-1.96,col="yellow", border = "yellow")
  rect(-10,1.96,10,10,col="yellow", border = "yellow")
  abline(v=0,lty=3)
  abline(h=0,lty=3)
  abline(h=-1.96,lty=2)
  abline(h=+1.96,lty=2)
  abline(v=-1.96,lty=2)
  abline(v=+1.96,lty=2)
  points(dinucl$intergenic[, x], dinucl$coding[, x], pch = 21,
  col = rgb(.1,.1,.1,.5), bg = rgb(.5,.5,.5,.5))
  legend("bottomright", inset = 0.02, legend = paste(substr(x,1,1), "p", substr(x,2,2), " bias", sep = "
  box()
}
myplot("CT")
myplot("TC")
myplot("CC")
myplot("TT")
```

However, we have little or no information on the exposure of this bacteria to UV light. In order to fully answer this question, let's do another analysis and look at *Prochlorococcus marinus* genome.

Prochlorococcus marinus seems to make an ideal model for investigating this hypothesis. Three completely sequenced strains are available in the Genome reviews database: two of these strains are adpated to living at a depth of more than 120 meters (accession numbers AE017126 and BX548175), and the other one at a depth of 5 meters (accession number BX548174).

Living at a depth of 5 meters, or at a depth of more than a 120 meters is totally different in terms of UV exposure: the residual intensity of 290 nm irradiation (UVb) in pure water can be estimated to 56% of its original intensity at 5 m depth and to less than 0.0001% at more than 120 m depth. For this reason, two of the *Prochlorococcus marinus* strains can be considered to be adapted to low levels of UV exposure, and the other one to much higher levels.

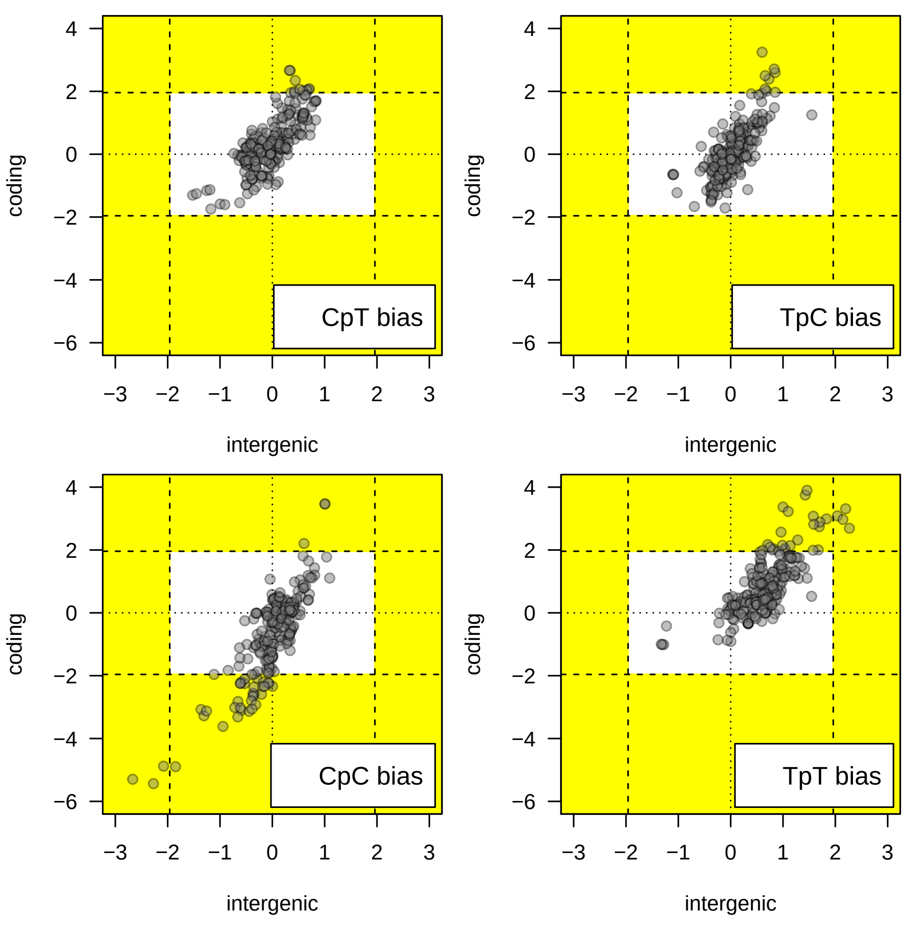


Figure 8.5: Plot of the mean *zscore* statistics for **intergenic sequences** (x-axis) and for **coding sequences** (y-axis), for each of the four pyrimidine dinucleotides. On each plot, a dot corresponds to the mean of these two statistics in a given prokaryote chromosome. The null x and y axis (dotted lines), and the 5% limits of significance for the standard normal distribution (dashed lines) are plotted as benchmarks. It should be noted that the variability within one chromosome is sometimes as great as that between different chromosomes.

Is pyrimidine dinucleotide content different in these three strains? And is it linked to their UV exposure?

We have computed the z-score with the `codon` model on all CDS from each of these three strains (as retrieved from Genome Reviews database on June 16, 2005). Figure 8.6 was produced with the following code:

```
data(prochlo)
oneplot <- function(x){
  plot(density(prochlo$BX548174[, x]),
    ylim = c(0,0.4), xlim = c(-4,4), lty=3,
    main = paste(substr(x,1,1), "p", substr(x,2,2), " bias", sep = ""),
    xlab="",ylab="",las=1, type = "n")
  rect(-10,-1,-1.96,10, col = "yellow", border = "yellow")
  rect(1.96,-1,10,10, col = "yellow", border = "yellow")
  lines(density(prochlo$BX548174[, x]),lty=3)
  lines(density(prochlo$AE017126[, x]),lty=2)
  lines(density(prochlo$BX548175[, x]),lty=1)
  abline(v=c(-1.96,1.96),lty=5)
  box()
}
par(mfrow=c(2,2),mar=c(2,3,2,0.5) + 0.1)
oneplot("CT")
oneplot("TC")
oneplot("CC")
oneplot("TT")
```

Figure 8.6 shows that there is no difference between the relative abundances of pyrimidine dinucleotides in these three strains. We can say that pyrimidine dinucleotides are not avoided, and that the hypothesis by Singer and Ames [89] no longer stands [69]. The following code was used to produce figure 8.7 that summarizes the relationship between pyrimidine dinucleotides and UV-exposure.

```
data(prochlo)
  par(oma = c(0, 0, 3, 0), mfrow = c(1, 2), mar = c(5, 4, 0, 0), cex = 1.5)
  example(waterabs, ask = FALSE) #left figure
  abline(v=260, lwd = 2, col = "red")
  par(mar = c(5, 0, 0, 2))
  plot(seq(-5, 3, by = 1), seq(0, 150, length = 9), col = "white",
    ann = FALSE, axes = FALSE, xaxs = "i", yaxs = "i")
  axis(1, at = c(-1.96, 0, 1.96), labels = c(-1.96, 0, 1.96))
  lines(rep(-1.96, 2),c(0, 150),lty=2)
  lines(rep(1.96, 2), c(0, 150),lty=2)
  title(xlab = "zscore distribution", cex = 1.5, adj = 0.65)
  selcol <- c(6, 8, 14, 16)
  z5 <- prochlo$BX548174[, selcol]
  z120 <- prochlo$AE017126[, selcol]
  z135 <- prochlo$BX548175[, selcol]
  todo <- function(who, xx, col = "black", bottom, loupe){
          dst <- density(who[, xx])
          sel <- which(dst$x >= -3)
            lines(dst$x[sel], dst$y[sel]*loupe + (bottom), col = col)
  }
  todo2 <- function(who, bottom, loupe){
    todo(who, "CC", "blue", bottom, loupe)
    todo(who, "CT", "red", bottom, loupe)
    todo(who, "TC", "green", bottom, loupe)
    todo(who, "TT", "black", bottom, loupe)
  }
  todo3 <- function(bottom, who, leg, loupe = 90){
    lines(c(-5,-3), c(150 - leg, bottom + 20))
    rect(-3,bottom,3,bottom+40)
    text(-2.6,bottom+38, paste(leg, "m"))
    todo2(who, bottom, loupe)
  }
      todo3(bottom = 110, who = z5, leg = 5)
      todo3(bottom = 50, who = z120, leg = 120)
      todo3(bottom = 5, who = z135, leg = 135)
```

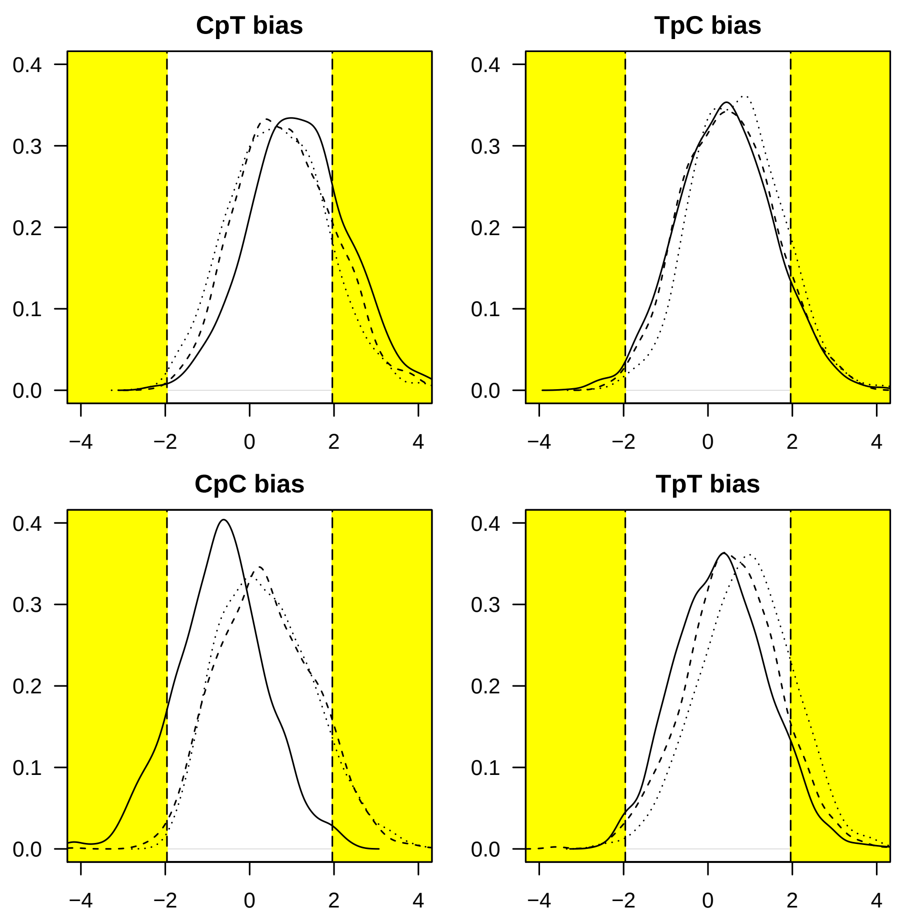


Figure 8.6: Each figure shows the distributions of the *zscore* in all **coding sequences** corresponding to each of the three strains of *Prochlorococcus marinus*. In each figure, the distribution for the MED4 (a high-light adapted strain) is shown as a solid line; the distribution for the SS120 (a low-light adapted strain) is shown as a dashed line, and the distribution for the MIT 9313 (a low-light adapted strain) is shown as a dotted line. The 5% limits of significance for the standard normal distribution (dashed vertical lines) are plotted as benchmarks.

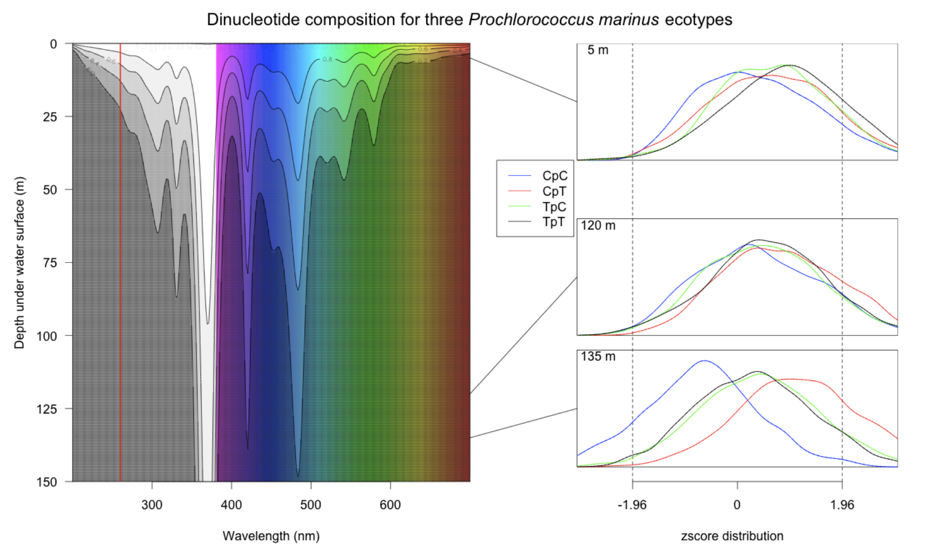


Figure 8.7: This figure is from figure 2.7 in [68], see also the example section in `data(prochlo)`. The left panel represents the absorbtion of light by pure water in the visible spectrum (gradient in color) and in the near UV (gradient in gray scale). Corresponding data were compiled from [76] and [55]. For DNA, the biological relevant wavelength is at 260 nm (red vertical line) corresponding to its maximum for light absorbtion. The right panel shows the distribution of the z-codon statistic for the four pyrimidine dinucleotides (*viz* CpC CpT TpC TpT) for the coding sequences of three different ecotypes (5 m, 120 m, 135 m) of *Prochlorococcus marinus*. The complete genome sequences accession numbers are BX548175 (*P. marinus* MIT9313 [80] 5 m, high UV exposure), AE017126 (*P. marinus* SS120 strain CCMP1375 [15] 120 m, low UV exposure) and BX548174 (*P. marinus* MED4 [80] 135 m, low UV exposure).

```
      legend(-4.5,110,c('CpC','CpT','TpC','TpT'),lty=1,pt.cex=cex,
        col=c('blue','red','green','black'))
      mtext(expression(paste("Dinucleotide composition for three ",
        italic("Prochlorococcus marinus")," ecotypes")), outer = TRUE, cex = 2,
line = 1)
```

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 16:46:33 2016
- LaTeX compilation time was: June 2, 2016

Part III

Appendix

CHAPTER 9

FAQ: Frequently Asked Questions

Lobry, J.R.

9.1 How can I compute a score over a moving window?

As an illustration, suppose that we want to reproduce a part of figure 1 from [56] whose screenshot is given is given in figure 9.1.

The score here is the GC-skew computed in non-overlapping windows of 10 Kb for a 1.6 Mb sequence. We need a fragment of *Escherchia coli* K12 chromosome from 67.4 min to 4.1 min on the genetic map. The sequence is directly available with `data(m16j)`. Let's put this fragment into the string `myseq`:

```
data(m16j)
myseq <- m16j
```

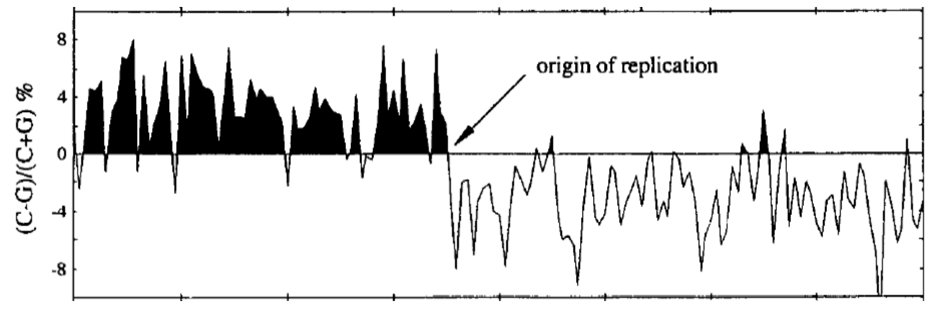


Figure 9.1: Screenshot of a part of figure 1 from [56]. The GC-skew is computed in non-overlapping windows of 10 Kb along a 1.6 Mb fragment of the *Escherichia coli* chromosome. The sequence is available with `data(m16j)`.

This is not exactly the same sequence that was used in [56] but very close to[1]. We define a function called `gcskew()` that computes our score for a given string `x`:

```
gcskew <- function(x){
  if( !is.character(x) || length(x) > 1 ) stop("single string expected")
  tmp <- tolower(s2c(x))
  nC <- sum(tmp == "c")
  nG <- sum(tmp == "g")
  if( nC + nG == 0 ) return(NA)
  return(100*(nC - nG)/(nC + nG))
}
gcskew("GCCC")
```

```
[1] 50
```

```
gcskew("GCCCNNNNNN")
```

```
[1] 50
```

Note some defensive programming tricks used here:

- We check that the argument `x` is a single string.
- We expand it as vector of single chars with `s2c()` only within the function to avoid big objects in the workspace.
- We force to lower case letters with `tolower()` so that we can use upper case letters too.
- We avoid division by zero and return `NA` in this case.
- We do not divide by the length of `x` but by the actual number of C and G so that ambiguous bases such as N do not introduce biases.

We move now along the sequence to compute the GC-skew:

```
step <- 10000
wsize <- 10000
starts <- seq(from = 1, to = nchar(myseq), by = step)
starts <- starts[-length(starts)] # remove last one
n <- length(starts)
result <- numeric(n)
for(i in seq_len(n)){
        result[i] <- gcskew(substr(myseq, starts[i], starts[i] + wsize - 1))
}
```

The following code[2] was used to produce figure 9.2.

```
xx <- starts/1000
yy <- result
n <- length(result)
hline <- 0
plot (yy ~ xx, type="n", axes=FALSE, ann=FALSE, ylim = c(-10, 10))
polygon(c(xx[1], xx, xx[n]), c(min(yy), yy, min(yy)), col = "black", border=NA)
usr <- par("usr")
rect(usr[1], usr[3], usr[2], hline, col="white", border=NA)
lines(xx, yy)
```

[1] The sequence used in [56] was a 1,616,174 bp fragment obtained from the concatenation of nine overlapping sequences (U18997, U00039, L10328, M87049, L19201, U00006, U14003, D10483, D26562 [90, 8, 13, 75, 5, 99]). Ambiguities have been resolved since then and its was a chimeric sequence from K-12 strains MG1655 and W3110 [33], the sequence used here is from strain MG1655 only [6].

[2]This code is adapted from the code at `http://www.stat.auckland.ac.nz/~paul/RGraphics/chapter3.html` for figure 3.25 in Paul Murrell's book [63]. This book is a must read if you are interested by R's *force de frappe* in the graphic domain.

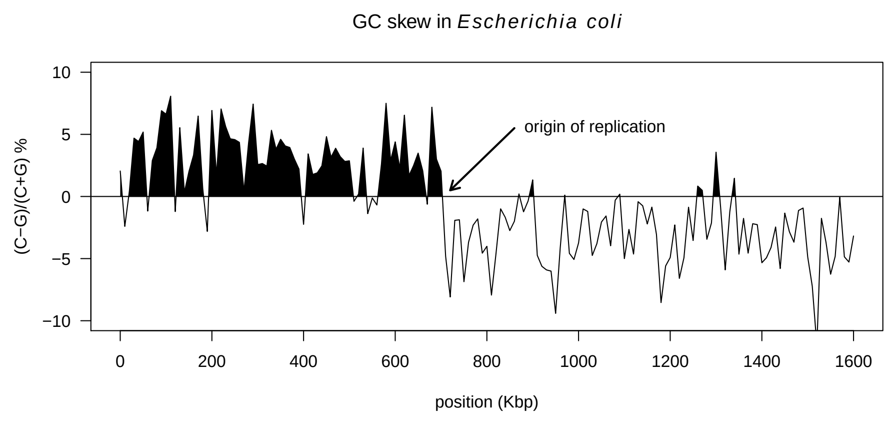


Figure 9.2: Re-creation of figure 9.1 from scratch.

```
abline (h=hline)
box()
axis(1, at = seq(0,1600, by = 200))
axis(2, las = 1)
title(xlab = "position (Kbp)", ylab = "(C-G)/(C+G) %",
main = expression(paste("GC skew in ", italic(Escherichia~~coli))))
arrows(860, 5.5, 720, 0.5, length = 0.1, lwd = 2)
text(860, 5.5, "origin of replication", pos = 4)
```

You can now play with the `wsize` and `step` parameters to explore the signal (but note that with overlapping windows your points are no more independent) or use all the smoothing tools available under R. Figure 9.3 shows for instance what can be obtained with the `lowess()` function with two values for the smoothing parameter `f`. The corresponding code is as follows:

```
plot(xx,yy, col = "grey", type = "b", ylim = c(-10,10), las = 1, xaxt = "n",
main = expression(paste("GC skew in ", italic(Escherichia~~coli))),
xlab = "position (Kbp)", ylab = "(C-G)/(C+G) %")
axis(1, at = seq(0,1600, by = 200))
lines(smooth <- lowess(xx,yy, f = 0.05), lwd = 1)
polycurve <- function(x, y, base.y = min(y), ...) polygon(x = c(min(x), x, max(x)), y = c(base.y, y, base.y), ...)
up <- smooth$y > 0
polycurve(smooth$x[up], smooth$y[up], base.y = 0, col = rgb(0,0,1,0.5))
lines(lowess(xx,yy, f = 0.2), lwd = 2, col = "red")
legend("topright", inset = 0.01, legend = c("f = 0.05", "f = 0.20"), lwd = c(1,2), col = c("black", "red"))
abline(h=0)
arrows(860, 5.5, 720, 0.5, length = 0.1, lwd = 2)
text(860, 5.5, "origin of replication", pos = 4)
```

9.2 How can I extract just a fragment from my sequence?

Use the generic function `getFrag()` :

```
choosebank("emblTP")
mylist <- query("mylist", "AC=A00001")
getFrag(mylist$req[[1]], begin = 10, end = 20)
```

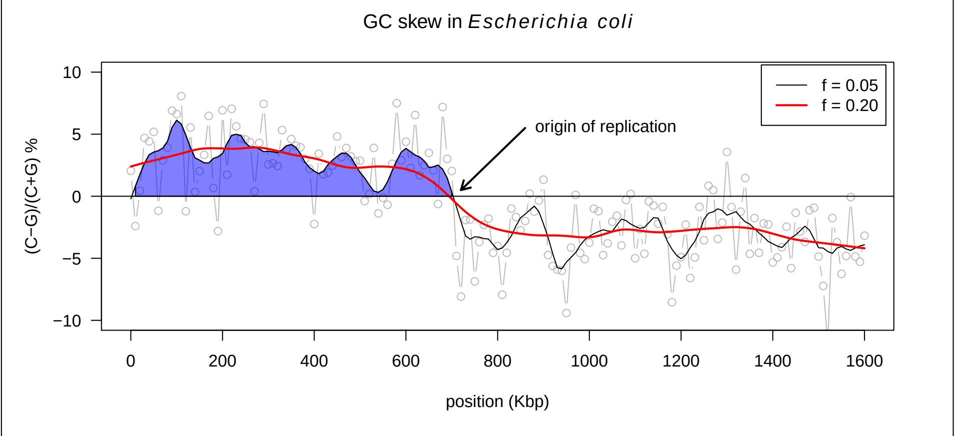


Figure 9.3: Playing with the smoothing parameter `f` of the `lowess()` function.

```
[1] "gatggagaatt"
attr(,"seqMother")
[1] "A00001"
attr(,"begin")
[1] 10
attr(,"end")
[1] 20
attr(,"class")
[1] "SeqFrag"
```

```
 closebank()
```

9.3 How do I compute a score on my sequences?

In the example below we want to compute the G+C content in third codon positions for complete ribosomal CDS from *Escherichia coli*:

```
 choosebank("emblTP")
 ecribo <- query("ecribo","sp=escherichia coli ET t=cds ET k=ribosom@ ET NO k=partial")
 myseqs <- sapply(ecribo$req, getSequence)
 (gc3 <- sapply(myseqs, GC3))
```

```
 [1] 0.4946237 0.6046512 0.5000000 0.6194030 0.5772727 0.4838710 0.5980066
 [8] 0.4974359 0.5031250 0.4324324 0.5000000 0.5113636 0.5290520 0.6142857
[15] 0.4904762 0.5714286 0.6191860 0.5906040 0.4880000 0.4880000 0.4946237
[22] 0.6046512 0.5000000 0.3522727 0.5076923 0.4343434 0.6194030 0.5522388
[29] 0.6104651 0.5661157 0.4946237 0.4946237 0.6079734 0.5000000 0.6343284
[36] 0.4659091 0.5789474 0.4946237 0.5000000 0.4974359 0.5689655 0.4611111
[43] 0.4611111 0.5303030 0.5303030 0.4482759 0.4201681 0.5915493 0.5000000
[50] 0.3829787 0.4519231 0.4302326 0.5696203 0.4285714 0.5689655 0.5000000
[57] 0.5224417 0.5661157 0.6057692 0.4444444 0.4659091 0.4130435 0.4946237
[64] 0.5661157 0.4946237 0.5680272
```

At the amino-acid level, we may get an estimate of the isoelectric point of the proteins this way:

```
 sapply( sapply(myseqs, getTrans), computePI)
```

```
 [1]  6.624309  7.801329 10.864793  5.931989  7.830476  6.624309  7.801329
 [8]  9.203410  9.826485  5.674672  7.154423  6.060457  6.313741  5.571446
[15]  9.435422  4.310747  6.145496  4.876054 11.006430 10.876041  6.624309
[22]  7.801329 10.864793  9.346289  9.203410  5.877050  5.931989  9.934988
[29]  5.920490  6.612505  6.624309  6.624309  7.801329 10.864793  5.931989
[36] 11.182505  9.598944  6.624309 10.864793  9.203410 11.031938  5.858421
```

```
[43]  5.858421 11.777516 11.777511 10.619175 11.365738  9.460987 10.864793
[50] 13.002373  9.845859 10.584868 11.421252 10.248320 11.031943 10.402075
[57]  4.863862  6.612505  9.681066 11.150310 11.182505 11.043607  6.624309
[64]  6.612505  6.624309  4.310747
```

Note that some pre-defined vectors to compute linear forms on sequences are available in the EXP data.

As a matter of convenience, you may encapsulate the computation of your favorite score within a function this way:

```
GC3m <- function(list, ind = 1:list$nelem) sapply(sapply(list$req[ind], getSequence), GC3)
GC3m(ecribo)
```

```
 [1] 0.4946237 0.6046512 0.5000000 0.6194030 0.5772727 0.4838710 0.5980066
 [8] 0.4974359 0.5031250 0.4324324 0.5000000 0.5113636 0.5290520 0.6142857
[15] 0.4904762 0.5714286 0.6191860 0.5906040 0.4880000 0.4880000 0.4946237
[22] 0.6046512 0.5000000 0.3522727 0.5076923 0.4343434 0.6194030 0.5522388
[29] 0.6104651 0.5661157 0.4946237 0.4946237 0.6079734 0.5000000 0.6343284
[36] 0.4659091 0.5789474 0.4946237 0.5000000 0.4974359 0.5689655 0.4611111
[43] 0.4611111 0.5303030 0.5303030 0.4482759 0.4201681 0.5915493 0.5000000
[50] 0.3829787 0.4519231 0.4302326 0.5696203 0.4285714 0.5689655 0.5000000
[57] 0.5224417 0.5661157 0.6057692 0.4444444 0.4659091 0.4130435 0.4946237
[64] 0.5661157 0.4946237 0.5680272
```

```
GC3m(ecribo, 1:10)
```

```
 [1] 0.4946237 0.6046512 0.5000000 0.6194030 0.5772727 0.4838710 0.5980066
 [8] 0.4974359 0.5031250 0.4324324
```

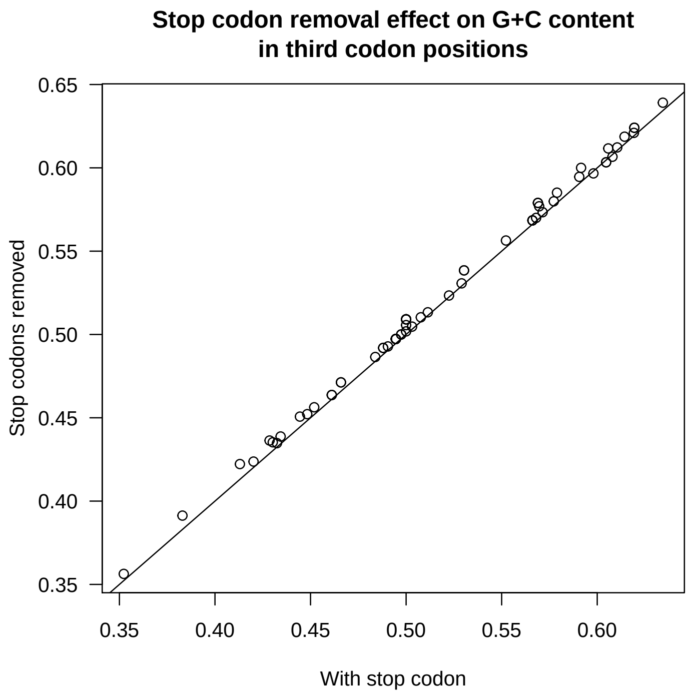
9.4 Why do I have not exactly the same G+C content as in codonW?

This question was raised (and solved) by Oliver Clay in an e-mail (23-AUG-2006). The program codonW was written in C as part of John Peden's PhD thesis on Codon Usage [73] and is available at http://codonw.sourceforge.net/. The reason for the small differences in G+C content between the two programs is that the default behavior in codonW is to remove the stop codon before computations. Here is one way of removing the stop codon under R:

```
gc3nos <- sapply(myseqs, function(s) GC3(s[1:(length(s) - 3)]))
```

As compared with the previous result, the difference is small but visible:

```
plot(x = gc3, y = gc3nos, las =1, main="Stop codon removal effect on G+C content
in third codon positions", xlab = "With stop codon", ylab ="Stop codons removed")
abline(c(0,1))
```



CodonW was released with a test file called input.dat, here are the first 10 lines of the file copied from CodonWSourceCode_1_4_4:

```
inputdatfile <- system.file("sequences/input.dat", package = "seqinr")
cat(readLines(inputdatfile,n=10), sep = "\n")
```

```
>YCG9 Probable          1377 residues Pha 0 Code 0
ATGAATATGCTCATTGTCGGTAGAGTTGTTGCTAGTGTTGGGGGAAGCGGACTTCAAACG
CTTTGCTTTGTTATTGGTTGTACGATGGTTGGTGAAAGGTCACGTCCATTGGTGATTTCC
ATCCTAAGTTGTGCATTTGCTGTAGCTGCTATCGTTGGTCCTATAATCGGAGGTGCCTTT
ACAACCCATGTTACCTGGAGGTGGTGCTTCTATATCAATCTTCCTATCGGTGGTCTTGCC
ATTATTATGTTTTTACTCACATATAAGGCCGAGAATAAGGGTATACTTCAACAAATTAAA
GATGCTATAGGAACAATCTCGAGCTTTACTTTTAGTAAGTTCAGACACCAAGTTAATTTT
AAAAGACTTATGAATGGCATAATCTTCAAGTTTGACTTCTTGGTTTTGCCCTCTGCTCT
GCAGGGCTGGTCCTTTTCCTACTGGGGCTAACCTTTGGTGGTAATAAATATAGTTGGAAC
TCTGGCCAAGTCATCGCATATTTGGTTTTGGGTGTCTTACTTTTTATTTTTTCATTGGTG
```

This is a FASTA file that we import under R with:

```
input <- read.fasta(file = inputdatfile)
names(input)
```

```
  [1] "YCG9"     "YCG8"     "ALPHA2"   "ALPHA1"   "CHA1"     "KRR1"
  [7] "PRD1"     "KAR4"     "PBN1"     "LRE1"     "APA1"     "YCE9"
 [13] "YCE8"     "YCE7"     "YCE5"     "YCE6"     "YCE4"     "PDI1"
 [19] "GLK1"     "YCD8"     "SRO9"     "YCD6"     "YCD5"     "YCD3"
 [25] "STE50"    "HIS4"     "BIK1"     "FUS1"     "YCO8"     "AGP1"
 [31] "LEU2"     "NFS1"     "BUD3"     "GBP2"     "ILV6"     "CWH36"
 [37] "PEL1"     "RER1"     "CDC10"    "MRPL32"   "YCP4"     "CIT2"
 [43] "YCP7"     "SAT4"     "RVS161"   "YCQ0"     "ADP1"     "PGK1"
 [49] "POL4"     "YCQ7"     "SRD1"     "MAK32"    "PET18"    "MAK31"
 [55] "HSP30"    "YCR3"     "SYN"      "YCR6"     "GNS1"     "FEN2"
 [61] "RIM1"     "CRY1"     "YCS2"     "YCS3"     "GNS1"     "RBK1"
 [67] "PHO87"    "BUD5"     "MATALPHA2" "MATALPHA1" "TSM1"     "YCT5"
 [73] "PETCR46"  "YCT7"     "YCT9"     "ARE1"     "RSC6"     "THR4"
 [79] "CTR86"    "PWP2"     "YCU9"     "YCV1"     "G10"      "HCM1"
 [85] "RAD18"    "CYPR"     "YCW1"     "YCW2"     "SSK22"    "SOL2"
 [91] "ERS1"     "PAT1"     "SRB8"     "YCX3"     "TUP1"     "YC16"
 [97] "ABP1"     "KIN82"    "MSH3"     "CDC39"    "YCY4"     "A2"
[103] "GIT1"     "YCZ0"     "YCZ1"     "YCZ2"     "YCZ3"     "PAU3"
[109] "YCZ5"     "YCZ6"     "YCZ7"
```

The file `input.out` contains the values obtained with `codonW` for the GC content and GC3s content:

```
inputoutfile <- system.file("sequences/input.out", package = "seqinr")
cat(readLines(inputoutfile, n=10), sep = "\n")
```

```
title                                GC3s        GC
YCG9_Probable__________13            0.335       0.394
YCG8________573_residues_            0.439       0.446
ALPHA2________633_residue            0.328       0.351
ALPHA1________528_residue            0.345       0.379
CHA1_________1083_residue            0.328       0.394
KRR1__________951_residue            0.364       0.384
PRD1_________2139_residue            0.430       0.397
KAR4_________1008_residue            0.354       0.383
PBN1_________1251_residue            0.330       0.386
```

```
input.res <- read.table(inputoutfile, header = TRUE)
head(input.res)
```

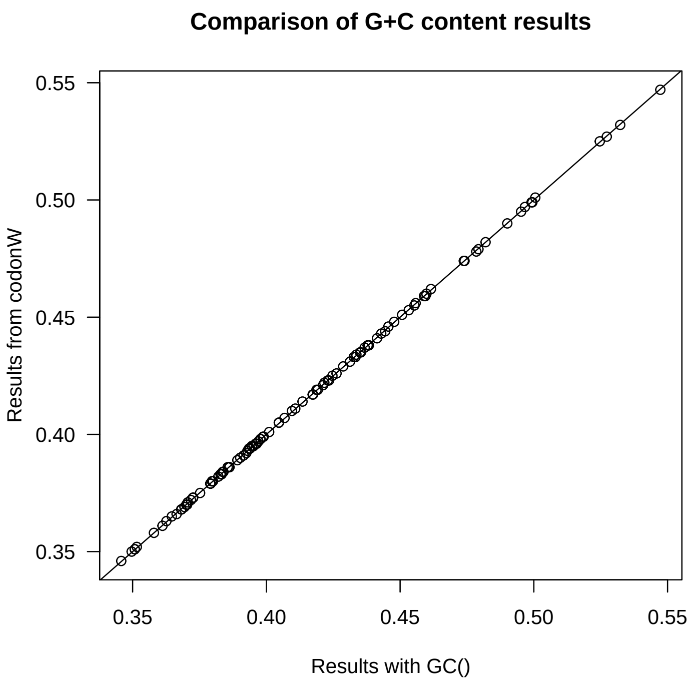
```
                      title  GC3s    GC
1 YCG9_Probable__________13 0.335 0.394
2 YCG8________573_residues_ 0.439 0.446
3 ALPHA2________633_residue 0.328 0.351
4 ALPHA1________528_residue 0.345 0.379
5 CHA1_________1083_residue 0.328 0.394
6 KRR1__________951_residue 0.364 0.384
```

Let's try to reproduce the results for the G+C content, we know that we have to remove the last stop codon:

```
input.gc <- sapply(input, function(s) GC(s[1:(length(s)-3)]))
max(abs(input.gc - input.res$GC))
```

```
[1] 0.0004946237
```

```
plot(x = input.gc, y = input.res$GC, las = 1,
xlab = "Results with GC()", ylab = "Results from codonW",
main = "Comparison of G+C content results")
abline(c(0,1))
```

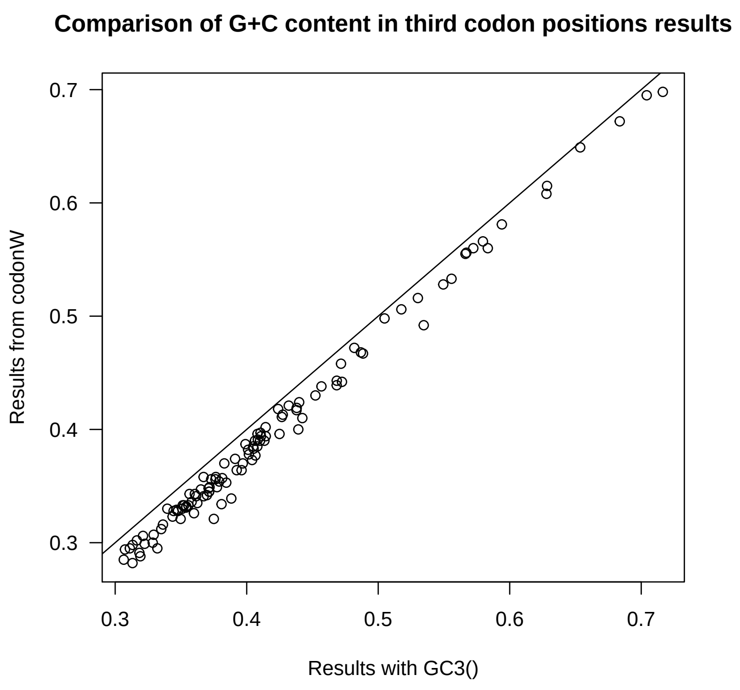


The results are consistent if we consider that we have 3 significant digits in the file `input.out`. Now, let's try to reproduce the results for G+C in third codon positions:

```
input.gc3 <- sapply(input, function(s) GC3(s[1:(length(s)-3)]))
max(abs(input.gc3 - input.res$GC3s))
```

```
[1] 0.054
```

```
plot(x = input.gc3, y = input.res$GC3s, las = 1,
xlab = "Results with GC3()", ylab = "Results from codonW",
main = "Comparison of G+C content in third codon positions results")
abline(c(0,1))
```

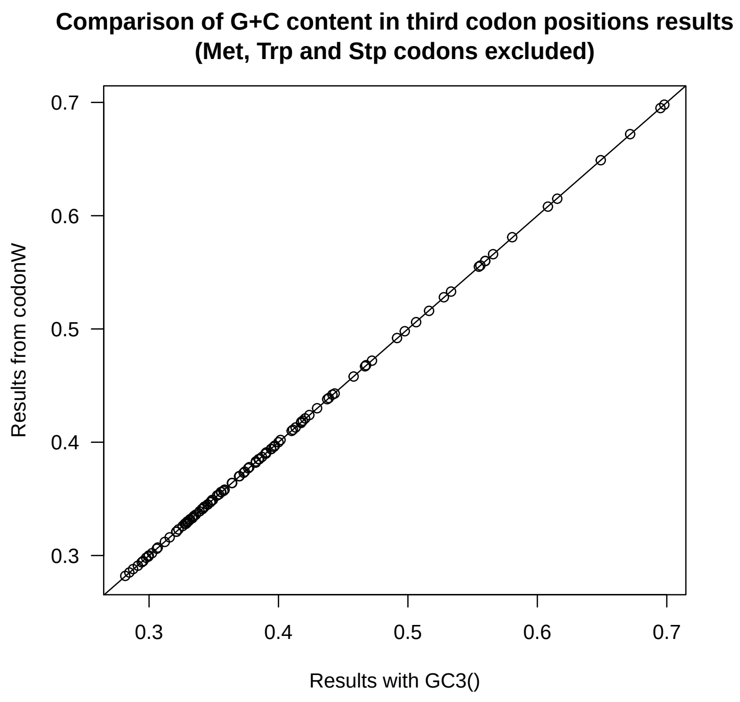


There is clearly a problem here. Looking into the documentation of `codonW`, GC3s is the G+C content in third codon position after removing non-synonymous and stop codons (those corresponding to Met, Trp, Stp). Let's remove these codons:

```
codons <- words()
names(codons) <- sapply(codons, function(c) aaa(translate(s2c(c), numcode = 1)))
okcodons <- codons[! names(codons) %in% c("Met", "Trp", "Stp")]
gc3s <- function(s){
  tmp <- splitseq(s)
  tmp <- tmp[tmp %in% okcodons]
  tmp <- s2c(paste(tmp, collapse = ""))
  GC3(tmp)
}
input.gc3s <- sapply(input, gc3s)
max(abs(input.gc3s - input.res$GC3s))
```

```
[1] 0.0004980843
```

```
plot(x = input.gc3s, y = input.res$GC3s, las = 1,
xlab = "Results with GC3()", ylab = "Results from codonW",
main = "Comparison of G+C content in third codon positions results\n(Met, Trp and Stp codons excluded)")
abline(c(0,1))
```



The results are now consistent. But thinking more about it there is still a problem with the codons for Ile:

```
codons[names(codons) == "Ile"]
```

```
  Ile   Ile   Ile 
"ata" "atc" "att" 
```

There are three codons for Ile. If the distribution of the four bases was uniform and selectively neutral in third codon position of synonymous codons, then we would expect to get a G+C of 50% in quartet and duet codons at third codons positions because they all have the same number of W (A or T)and S (C or G) bases in third position. But for Ile we have two codons ending in W versus only one in S so that we would get a G+C of $\frac{1}{3}$ instead of $\frac{1}{2}$. This point was clearly stated [92] by Sueoka in 1988:

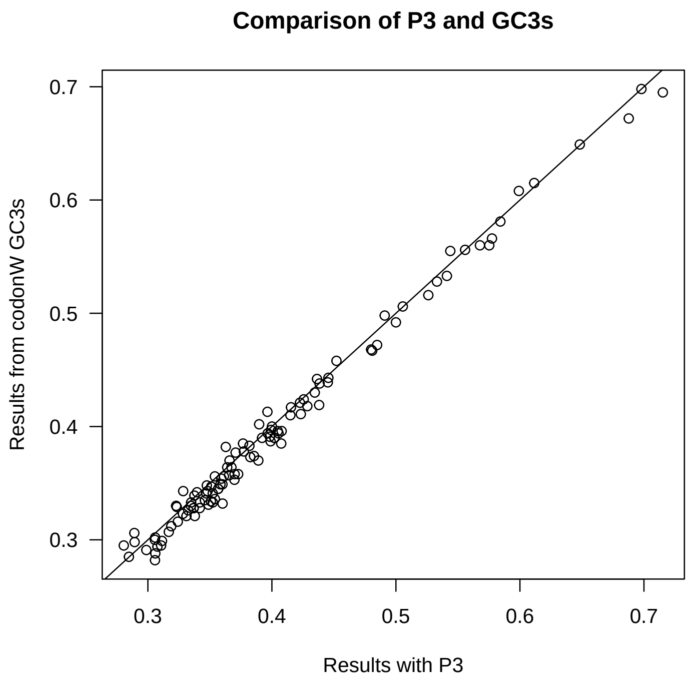
> **G + C Content of the Three Codons Positions.** In the present analysis, observed G + C contents of the first, second, and third codon positions (P_1, P_2, and P_3, respectively) are corrected average G + C contents of the three codon positions that are calculated from 56 triplets out of 64. Because of the inequality of α and γ at the third codon position, the three stop codons (TAA, TAG, and TGA) and the three codons for isoleucine (ATT, ATC, and ATA) were excluded in calculation of P_3, and two single codons for methionine (ATG) and tryptophan (TGG) were excluded in all three (P_1, P_2, and P_3)

Let's compute P_3 and compare it with GC3s:

```
P3codons <- codons[! names(codons) %in% c("Met", "Trp", "Ile", "Stp")]
P3 <- function(s){
  tmp <- splitseq(s)
  tmp <- tmp[tmp %in% P3codons]
  tmp <- s2c(paste(tmp, collapse = ""))
  GC3(tmp)
}
input.P3 <- sapply(input, P3)
max(abs(input.P3 - input.res$GC3s))
```

```
[1] 0.02821505
```

```
plot(x = input.P3, y = input.res$GC3s, las = 1,
xlab = "Results with P3", ylab = "Results from codonW GC3s",
main = "Comparison of P3 and GC3s")
abline(c(0,1))
```



This is not exactly the same, the maximum observed difference here is about 3%. In practice, P_3, GC3, and GC3s are only slightly different [93].

9.5 How do I get a sequence from its name?

This question is adapted from an e-mail (22 Jun 2006) by Gang Xu. I know that the UniProt (SwissProt) entry of my protein is `P08758`, if I know its name[3], how can I get the sequence?

```
choosebank("swissprot")
myprot <- query("myprot","AC=P08758")
getSequence(myprot$req[[1]])
```

[3]More exactly, this is the accession number. Sequence names are not stable over time, it's always better to use the accession numbers.

```
  [1] "M" "A" "Q" "V" "L" "R" "G" "T" "V" "T" "D" "F" "P" "G" "F" "D" "E" "R"
 [19] "A" "D" "A" "E" "T" "L" "R" "K" "A" "M" "K" "G" "L" "G" "T" "D" "E" "E"
 [37] "S" "I" "L" "T" "L" "L" "T" "S" "R" "S" "N" "A" "Q" "R" "Q" "E" "I" "S"
 [55] "A" "A" "F" "K" "T" "L" "F" "G" "R" "D" "L" "L" "D" "D" "L" "K" "S" "E"
 [73] "L" "T" "G" "K" "F" "E" "K" "L" "I" "V" "A" "L" "M" "K" "P" "S" "R" "L"
 [91] "Y" "D" "A" "Y" "E" "L" "K" "H" "A" "L" "K" "G" "A" "G" "T" "N" "E" "K"
[109] "V" "L" "T" "E" "I" "I" "A" "S" "R" "T" "P" "E" "E" "L" "R" "A" "I" "K"
[127] "Q" "V" "Y" "E" "E" "E" "Y" "G" "S" "S" "L" "E" "D" "D" "V" "V" "G" "D"
[145] "T" "S" "G" "Y" "Y" "Q" "R" "M" "L" "V" "V" "L" "L" "Q" "A" "N" "R" "D"
[163] "P" "D" "A" "G" "I" "D" "E" "A" "Q" "V" "E" "Q" "D" "A" "Q" "A" "L" "F"
[181] "Q" "A" "G" "E" "L" "K" "W" "G" "T" "D" "E" "E" "K" "F" "I" "T" "I" "F"
[199] "G" "T" "R" "S" "V" "S" "H" "L" "R" "K" "V" "F" "D" "K" "Y" "M" "T" "I"
[217] "S" "G" "F" "Q" "I" "E" "E" "T" "I" "D" "R" "E" "T" "S" "G" "N" "L" "E"
[235] "Q" "L" "L" "L" "A" "V" "V" "K" "S" "I" "R" "S" "I" "P" "A" "Y" "L" "A"
[253] "E" "T" "L" "Y" "Y" "A" "M" "K" "G" "A" "G" "T" "D" "D" "H" "T" "L" "I"
[271] "R" "V" "M" "V" "S" "R" "S" "E" "I" "D" "L" "F" "N" "I" "R" "K" "E" "F"
[289] "R" "K" "N" "F" "A" "T" "S" "L" "Y" "S" "M" "I" "K" "G" "D" "T" "S" "G"
[307] "D" "Y" "K" "K" "A" "L" "L" "L" "L" "C" "G" "E" "D" "D"
```

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Wed Jun 1 17:13:01 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 10

GNU Free Documentation License

Version 1.2, November 2002
Copyright ©2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

10.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **”Document”**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **”you”**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **”Modified Version”** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **”Secondary Section”** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **”Invariant Sections”** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **”Cover Texts”** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **”Transparent”** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called **”Opaque”**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **”Title Page”** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

10.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

10.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

10.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

10.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”.

10.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

10.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

10.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

CHAPTER 11

Release notes

Lobry, J.R. Necşulea, A. Palmeira, L. Penel, S.

Introduction

The release notes are listed in reverse chronological order: most recent on top.

3.1 series

release 3.1-5

- As pointed out by e-mail on 30-MAY-2016 by Haruo Suzuki a call to `getLength(ec999)` yielded spurious output and many warnings. This is now fixed.

- As pointed out by e-mail on 30-MAY-2016 by Haruo Suzuki there was a bug in the documentation of the functions `recstat()`, `draw.recstat()`, `test.co.recstat()` and `test.li.recstat()`. They were all looking for data in a package `seqinr2` that doesn't exist. This is now fixed and the `dontrun` directive has been removed to detect automatically any further problem.

- As pointed out by e-mail on 25-MAY-2016 by Haruo Suzuki the `read.fasta()` function can import sequences directly from *local* gzipped files. A new `smallAA.fasta.gz` file has been added to document this in the examples of the `read.fasta()` function. This is however no more true if you try to read directly the sequences from a compressed file accessed via its URL. A workaround now given in the manual is to use a construct like `read.fasta(gzcon(url(myurl)))`.

- As pointed out by e-mail on 12-MAY-2016 by Haruo Suzuki the documentation for the `rho()` function was misleading because in the refered article

[42] the statistic was computed from the sequence concatenated with its inverted complement. This is now fixed.

3.0 series

release 3.0-11

- In `query()`, `NS=taxon_name` and `NK=keyword_name` are now documented. The manual was also updated.
- The broken default link in `get.db.growth()` has been fixed so that now `dia.db.growth()` works as well.
- Function `write.fasta()` has gained an `as.string` argument so that it can handle sequences provided as strings instead of vectors of single character.

release 3.0-10

release 3.0-0

- As pointed out by Leonor Palmeira on the `rpourlesnuls` diffusion list on 20-MAY-2010 there was no constructor for objects of class alignment. There is now a `as.alignment()` function.

2.0 series

release 2.0-9

- As pointed out by Avril Coghlan on the seqinR diffusion list on 17-MAR-2010 there was a bug in the `getAnnot()` function. This is now fixed.
- As suggested by Avril Coghlan on the seqinR diffusion list on 02-MAR-2010 the function `rho()` has gained a `wordsize` argument.
- The argument `word` in function `count()` is now more explicitely called `wordsize`.
- The example section in file `read.alignment.Rd` has gained a new quality control sanity check.
- The `File` argument that was deprecated since seqinR release 1.1-3 in function `read.alignment()` is no more valid. Just use `file` instead.
- As pointed by Darren Obbard on the seqinr diffusion list on 05-MAR-2010 there was a memory leak problem when calling the `read.alignment()` function with the fasta format. This is now fixed for the fasta format, but the remaining formats have not been checked for this problem.

release 2.0-8

- As pointed by Oliver Clay and Lionel Guy on the seqinr diffusion list on 19-FEB-2010 there was a bug in `getSequence.list()` function that confused `write.fasta()` when all sequences were of the same length (a similar bug was reported by Yann Lesecque on 30-MAR-2009 for the `getTrans()` function). This is now fixed.

- The message printed when function `where.is.this.acc()` fails to find a database with a given accession number for a sequence is now completed to warn the user that (s)he may have supplied a sequence name instead of a genuine accession number.

- The title in the documentation for the function `write.fasta()` was changed to make clear that more than one sequence can be written at once. The function now does not return anything instead of `NULL` previously. The argument `file.out` was moved to the left so that it is easier now to use it by position during function call.

release 2.0-7

- A new utility function `where.is.this.acc()` was introduced to loop over all availabale ACNUC databases to look for a given sequence accession number. This is useful when you have a sequence accession number and you don't know in which database it is present. The documentation of the function `choosebank()` was also changed to make a link to this function. As suggested by Avril Coghlan, the function has an argument `stopAtFirst` defaulting to `TRUE` that stops the search at the first database found with the given accession number.

- As pointed out 05 Nov 2009 by Darren Obbard on the seqinr diffusion list the argument `forceToLower = FALSE` in function `comp()` was not honored. This is now fixed and a new sanity check was added in the example section of the documentation of the function.

- Documentation for the function `uco()` for codon usage table computation was updated with new bibliographical references [58, 94].

- As basic regular expressions are defunct since R 2.11, the `extended` argument in functions `words.pos()` and `trimSpace()` was no more necessary. It is now deleted.

release 2.0-6

- The old argument `File` in function `read.fasta()` that was deprecated since release 1.1-3 is no more valid. Just use `file` instead.

- New function `stutterabif()` to estimate stutter ratio.

- Function `plotabif()` has a new default value for its `ylim` argument: `c(min(y), max(y))` now instead of `c(0, max(y))` previously to help plotting data with a highly negative baseline.

- Function `peakabif()` now returns in addition an estimate of the baseline value.
- New utility `baselineabif()` to estimate the baseline value.
- There was time shift of one datapoint unit for the peak locations returned by the `peakabif()` function, this is now fixed and the documentation is more explicit for the units used.
- New utility function `fastacc()` to compute the number of alleles in common between a genetic profile and a database of genetic profiles.

release 2.0-5

- New utility function `circle()` to draw a circle.
- Two more examples of files to be imported with the `readBins()` and `readPanels()` functions are now available in the `abif` folder: `NGM_Bins.txt` and `NGM_Pa.txt`, respectively.
- New function `plotPanels()` to plot amplicon size ranges of STR kits data.
- New utility function `col2alpha()` to add a transparency chanel to a standard R color.
- New ABIF example file `samplefsa2ps.fsa` used in the `read.abif()` function to reproduce figure 1A from [47].
- New function `move()` aliased as `mv()` to rename an object without deep copy.
- New function `swap()` to exchange two objects.

release 2.0-4

- Configuration files to be imported by the `readBins()` function may have trailling tabulations, as for instance in the test file `Prototype_PowerPlex_EP01_Bins.txt` for allele 9 at locus `D3S1358` and for allele 14 at locus `D12S391`. This was a source of trouble during importation. This is now fixed and the above mentioned file is used as a quality control. A warning is now issued if the number of columns in the data.frame corresponding to a locus is not 4 as expected.
- Configuration files to be imported by the `readPanels()` function may have more than one tabulation separator between two data items in a way that could be different from one line to another one. There is an example of such a case in the test file `Prototype_PowerPlex_EP01_Pa.txt` where locus `D10S1248` and `D22S1045` are followed by a single tabulation when all remaining loci are followed by two tabulations. This was a source of trouble during importation. This is now fixed by preprocessing the input so that all consecutive tabulations are replaced by a single one. The above mentioned test file is now used as a quality control.

release 2.0-3

- As pointed out on the seqinr diffusion list on 23-APR-2009 by Darren Obbard there was an obscure error message when function `kaks()` was called with an alignment such that the number of nucleotides was not a multiple of 3 after gap removal. This check was partial as an alignment with out-of-frame gaps but with a total number of gaps multiple of 3 was not detected. The new behaviour is that if at least one non ACGT base is found in a codon, then the whole codon is forced to a gap codon (`--`). The documentation of the function has been clarified accordingly, and a new alignment file `DarrenObbard.fasta` added in the `sequences` folder to check this new behaviour.

- Function `readBins()` is now more tolerant when there is an extra column with possibly empty fields in data by forcing the `fill` argument of `read.table()` function to `TRUE`.

- As pointed out by e-mail on 30-MAR-2009 by Yann Lesecque there was a bug in the `getTrans()` function: when applied to a list of sequences with all the same length the returned result was a matrix instead of a list. This is now fixed.

- New utility functions `readPanels()` and `readBins()` to import data from GeneMapper configuration files. Four example files are now in the `abif` folder.

- Function `peakabif()` now returns the heights and surfaces of peaks in addition to their location.

- New utility function `al2bp()` to convert a forensic microsatellite allele name into its length in base pairs. Conventions used to name forensic microsatellite alleles (STR) are described in Bar *et al.* (1994) [3]. The name `9.3` means for instance that there are 9 repetitions of the complete base oligomer and an incomplete repeat with 3 bp.

release 2.0-2

- New ABIF format related functions: `plotabif()` to plot electrophoregrams with optonial internal size standard and optional allelic ladder, `peakabif()` to locate peaks in electrophoregrams, `plotladder()` to display an observed allelic ladder.

- New datasets `gs500liz` for size standards, `identifiler` for allelic ladder names, `ECH` for allelic ladder raw data and `JLO` for forensic genetic profile raw data. The last one is now used as a quality check for the `read.abif()` function.

- A new folder called `abif` has been created under the `inst` folder. The purpose of this folder is to contain examples of files in ABIF format so that the results of the `read.abif()` function can be checked against expected results for quality check. It contains for now two duplicated genetic profiles and two allelic ladders from the same batch experiment.

release 2.0-1

- The useless `itemize` in the argument section of documentation file `stresc.Rd` is now deleted.
- In function `words.pos()` the default value for parameter `extended` was changed from FALSE to TRUE to avoid warnings.
- New experimental function `read.abif()` to import files in ABIF format (`*.fsa`, (`*.ab1`).

release 2.0-0

- New draft chapter about making RISA *in silico* added.
- Objects from class `qaw` created after a call to the `query()` function have gained a new generic `print` method to focus on the most important information: number of sequences in the list, list type and the corresponding request.
- Function `query()` now allows a missing `listname` argument. In this case, `list1` is used to store the result.
- Function `autosocket()` has been changed to behave more friendly with outdated R versions. This is essentially a backward compatibility issue that will not be maintained in the future. The function `autosocket()` works hard to check that everything is OK with the last opened database, especially with the socket infos available in `banknameSocket$socket` thru its `summary()` generic. In old R versions (*e.g.* 2.6.2) this was returning `socket` instead of `sockconn` for the class, yielding an error in seqinR 1.1-7. The old result is now allowed but a warning is issued.

The 2.0 series started in summer 2008 along with the moving of the seqinr sources on R-forge.

1.1 series

release 1.1-7

- As suggested by Kurt Hornik two extra `cr` in the documentation file for `ec999` were deleted.
- Function `read.fasta()` has gained four new arguments (*viz.* `bfa`, `sizeof.longlong`, `endian`, `apply.mask`) to read DNA binary fasta files in MAQ format. There is a new `ct.bfa` file in the `sequences` folder to check for the MAQ format reading.
- New dataset `pK` for the values for the side chain of charged amino acids from various sources compiled by Joanna Kiraga [46].
- Function `words.pos()` has gained new arguments that are passed to `regexpr()` including the dot-dot-dot argument in case of need in the future. The documentation has been modified to better explain the difference with the standard `gregexpr()` function.

- As pointed by e-mail on 28 May 2008 by Kim Milferstedt a function to compute the consensus for a set of aligned sequences would be helpful. There is now a function `consensus()` aliased to `con()` for this. The input is either an object from class `alignment` or a matrix of characters. The output is either a consensus sequence (using the majority rule, the majority rule with a threshold, or IUPAC symbols for RNA and DNA sequences) or a profile, that is a matrix with the count of each possible character at each position in the alignment.

- In the documentation of the `read.alignment()` function a link was added to the `read.nexus()` function from the `ComPairWise` package [79].

- New function `bma()` to find the IUPAC symbol corresponding to a nucleic sequence.

- New function `as.matrix.alignment()` to convert an alignment into a object of class `matrix`.

- The encoding of line ends in the example file `test.mase` is now an unix-like one.

- As pointed by e-mail on 31 May 2008 by Marie Sémon there was no convenient function to compute the Codon Adaptation Index [87]. A new function `cai()` was introduced with the aim of reproducing exactly the results from the program `codonW` that was written by John Peden during his PhD thesis [73] under the supervision of P.M. Sharp (the most authorative source for CAI computation). A new dataset `caitab` that was hard-encoded in `codonW` for the `w` values for some species (*viz Escherichia coli*, *Bacillus subtilis*, *Saccharomyces cerevisiae*) was added. Care was taken to credit original sources. The *E. coli* data that was uncredited is from [87]. The *B. subtilis* data that was uncredited is from [88] (see the note of caution in `?caitab` before using this one directly to compute CAI in *B. subtilis*). The *S. cerevisiae* data that was credited to [86] dates back from [87]. A new text file `scuco.txt` produced by `codonW` was added in the `sequences` folder to check that the CAI results from `cai()` are consistents with thoses from `codonW` version 1.4.4 (03-MAR-2005). This legacy file is used in the example section of the `cai()` function.

release 1.1-6

- The construct `get(getOption("device"))(width = 18, height = 11)` that was used in the example section for `data(prochlo)` is no more valid since R 2.8.0 (fall 2008). The example has been restricted to work only with `X11`, `windows` and `quartz` devices.

- As pointed by e-mail on 12 May 2008 by Indranuj Mukherjee there was a bug in the function `oriloc()`: when called with a `gbk = NULL` argument the function was trying to remove non-existent files, yielding an error. The bug has been fixed and the documentation of the function `oriloc()` has been extended to better explain how to use the arguments `seq.fasta` and `gbk`.

- A reference to [24] was missing in the documentation of function `zscore()` for the codon model.
- As suggested by e-mail on 11 Mar 2008 by Christian Gautier, the function `count()` has gained a new argument `by` to control the window step, allowing for instant to count dinucleotides in codon position III-I in a coding sequence. The example section of the function documentation has been extended to give an example of counting dinucleotides in position III-I.

```
alldinuclIIIpI <- s2c("NNaaNatNttNtgNgtNtcNctNtaNagNggNgcNcgNgaNacNccNcaNN")
(resIIIpI <- count(alldinuclIIIpI, word = 2, start = 2, by = 3))
```

```
aa ac ag at ca cc cg ct ga gc gg gt ta tc tg tt
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

```
stopifnot(all( resIIIpI == 1))
```

- Function `reverse.align()` has gained two arguments `forceDNAtolower = TRUE` and `forceAAtolower = FALSE` that are passed to the functions used to read the sequences. There is now a new dataset `revaligntest` used to check the result in the example section of `reverse.align()`.
- As pointed by e-mail on 21 Feb 2008 by Oliver Keatinge Clay function `modifylist()` failed to scan in GenBank FEATURES annotation lines. There is now a new function called `prepgetannots()`, aliased to `pga()`, that allows to set up the annotation lines to be scanned. Called with default arguments, this function turns on all annotation lines for scan. This function can also be used to set up partly the annotation lines to be returned by `getAnnot()`.
- Function `choosebank()` has gained four arguments (`server`, `blocking`, `open`, `encoding`) that are passed to `socketConnection()`. The value of the argument `verbose` is now passed to `clientid()` which knows now how to handle it. The `encoding` argument was introduced to fix a localization bug on Mac OS X which symptom was a cryptic error message in `if (res[1] != "0") {` after a call to `choosebank()`. The culprit was an `option(encoding = "latin1")` that was set up before the call to `choosebank()` who called `socketConnection()` with its default `encoding = getOption("encoding")`, preventing `readLines()` to read from the socket. The bug was fixed by opening the socket with the native encoding, which is the current default.
- As pointed by e-mail on 15 Jan 2008 by Stefanie Hartmann, the argument `frame` in function `count()` was misleading for someone with a molecular biology background. The argument has been replaced by `start`. The old argument name is maintained as an alias for backward compatibility. The example section has been extended to give an example with the complete human mitochondrion sequence, the corresponding fasta file (`humanMito.fasta`) has been added in the `sequences` directory.

release 1.1-5

Minor release to fix mainly problems in the documentation.

- The argument section was empty in `autosocket.Rd`.
- The details section was empty in `countfreelists.Rd` and `draw.oriloc.Rd`.
- The value section was empty in `gbk2g2.Rd`. The corresponding function was changed to use a local file for the demo.
- The description section was missing in `getFrag.Rd`, `getLength.Rd`, `getName.Rd`, `getSequence.Rd`.
- Documentation of the function `dia.bactgensize()` to plot the distribution of bacterial genome size from GOLD data has been ammended to credit sources [48, 4, 54, 53]. It has gained a new argument `maxgensize` defaulting to 20000 to remove outliers. It has also gained a new argument `source` for the file to look for raw data, defaulting to an (outdated) local copy so that the function can be called even when there is no internet connection.

release 1.1-4 (10-Dec-2007)

Minor release to fix problems found by Kurt Hornik.

- In the `DESCRIPTION` file `License:  GPL (>= 2)` instead of `License:  GPL version 2 or newer`.
- The files `inst/doc/src/mainmatter/acnuc_sockets.rnw .tex` with non-portable file names were changed to `acnucsocket.rnw` and `acnucsocket.tex`.

release 1.1-3

- There is a new chapter to explain how to set up a local ACNUC server on Unix-like platforms.
- New dataset `m16j` to make a GC skew plot as in [56].
- New dataset `waterabs` giving the absorption of light by water. This dataset was compiled by Palmeira [68] from [55, 76].
- Generic functions `getAnnot()`, `getFrag()`, `getKeyword()`, `getLength()`, `getLocation()`, `getName()`, `getSequence()` and `getTrans()` have gained methods to handle objects from class `list` and `qaw`.
- Functions `getAttributsocket()` and `getNumber.socket()` are now deprecated, a warning is issued.
- There is a new appendix in which all the examples protected by a `dontrun` statment are forced to be executed.
- Function `read.fasta()` now supports comment lines starting by a semi-colon character in FASTA files. An example of such a file is provided in `sequences/legacy.fasta`. The argument `File` is now deprecated. There is a new argument `seqonly` to import just the sequences without names, annotations and coercion attempts. There is a new argument `strip.desc` to remove the leading '>' character in annotations (as in function `readFASTA` from the Biostrings package [67]). The FASTA file example `someORF.fsa` from Biostrings is also added for comparisons.

- Function `GC()` has gained a new argument `NA.GC` defaulting to `NA` to say what should be returned when the GC content cannot be computed from data (for instance with a sequence like NNNNNNNNNNNN). The argument `oldGC` is now deprecated and a warning is issued. Functions `GC1()`, `GC2()`, `GC3()` are now simple wrappers for the more general `GCpos()` function. The new argument `frame` allows to take the frame into account for CDS.
- Function `read.alignment()` has gained a new argument `forceToLower` defaulting to TRUE to force lower case in the character of the sequence (this is for a smoother interaction with the package `ape`). The argument `File` is now deprecated and a warning is issued when used instead of `file`. The example in the function `kaks()` has been corrected to avoid this warning when reading the example files.
- New low level utility function `acnucclose()` and `quitacnuc()` to close an ACNUC server. These functions are called by `closebank()` so that a simple call to it should be enough.
- New low level utility function `clientid()` to send the client ID to an ACNUC server.
- New low level utility function `countfreelists()` to get the number of free lists available in an ACNUC server.
- New low level utility function `knowndbs()` and its shortcut `kdb()` to get a description of databases known by an ACNUC server.
- New low level utility function `autosocket()` to get the socket connection to the last opened ACNUC database.
- New function `countsubseqs()` to get the number of subsequences in an ACNUC list.
- New function `savelist()` to save sequence names or accession numbers from an ACNUC list into a local file.
- New function `ghelp()` to get help from an ACNUC server.
- New function `modifylist()` to modify a previously existing ACNUC list by selecting sequences either by length, either by date, either for the presence of a given string in annotations.
- New low level function `getlistate()` to ask for information about an ACNUC list.
- New low level function `setlistname()` to set the name of a list from an ACNUC server.
- New function `residuecount()` to count the total number of residues (nucleotides or aminoacids) in all sequences of an ACNUC list of specified rank.
- New function `isenum()` and its shortcut `isn()` to get the ACNUC number of a sequence from its name or accession number.

- New function `prettyseq()` to get a text representation of a sequence from an ACNUC server.
- New function `gfrag()` to extract sequence identified by name or by number from an ACNUC server.
- The details of the socket connection are no more stored in the slot `socket` for objects of class `seqAcnucWeb`: this slot is now deleted. As a consequence, the argument `socket` in function `as.SeqAcnucWeb()` has been removed and there is now a new argument `socket = "auto"` in functions `getAnnot()`, `getFrag()`, `geyKeyword()`, `getLocation()`, and `getSequence()`. The default value `"auto"` means that the details of the socket connection are taken automatically when necessary from the last opened bank. The size of local lists of sequences is reduced by about a third now as compared to the previous version.
- New function `print.seqAcnucWeb()` to print objects from class `seqAcnucWeb`.
- Internal function `parser.socket()` has been optimized and is about four times faster now. This decreases the time needed by the `query()` function.

release 1.1-2

- New function `trimSpace()` to remove leading and trailing spaces in string vectors.
- Function `splitseq()` is no more based on `substring()`, it is now more efficient for long sequences.
- A sanity check test was added in the documentation file for the function `syncodons()`.
- The way this manual is produced is now documented in the `doc/src/template/` folder.
- A bug in function `oriloc()` was reported on 23 Jul 2007 by Michael Kube: using directly genBank files was no more possible. The culprit was `gbk2g2()` that turns genBank files into glimmer files version 2 when `oriloc()` default is to use version 3 files. The `glimmer.version` argument is now forced to 2 when working with genBank files to fix this problem.
- Function `zscore()` has now a new argument `exact` (which is only effective for the option `model = base`). This argument, when set to `TRUE` allows for the exact analytical computation of the zscore under this model, instead of the approximation for large sequences. It is set to `FALSE` by default for backward compatibility.

release 1.1-1

- A bug was reported by Sylvain Mousset on 14 Jul 2007 in function `dist.alignment()`: when called with sequences in lower case letters, some sequences were modified. This should no more be the case:

```
 ali <- list(nb=4, nam=c("speciesA", "speciesB", "speciesC", "speciesD"),
 seq=c("ACGT","acgt","ACGT","ACGT"))
 class(ali) <- "alignment"
 print(ali$seq)

[1] "ACGT" "acgt" "ACGT" "ACGT"

 print(dist.alignment(ali))

         speciesA speciesB speciesC
speciesB        0
speciesC        0        0
speciesD        0        0        0

 print(ali$seq)

[1] "ACGT" "acgt" "ACGT" "ACGT"
```

- The CITATION file has been updated so that now `citation("seqinr")` returns the full complete reference for the package seqinR.
- Non ASCII characters in documentation (*.Rd) files have been removed. Declaration of the encoding as latin1 when necessary is now present. The updated documentation files are: `dinucl.Rd, gb2fasta.Rd, get.ncbi.Rd, lseqinr.Rd, n2s.Rd, prochlo.Rd, s2c.Rd, SeqAcnucWeb.Rd, SeqFrag.Rd, toyaa.Rd, words.pos.Rd, words.Rd, zscore.Rd`.
- Function `GC()` and by propagation functions `GC1()`, `GC2()` and `GC3()` have gained a new argument `oldGC` allowing to compute the G+C content as in releases up to 1.0-6 included. The code has been also modified to avoid divisions by zero with very small sequences.
- New function `rot13()` that returns the ROT-13 encoding of a string of characters.

1.0 series

release 1.0-7

- A new *experimental* function `extractseqs()` to download sequences thru zlib compressed sockets from an ACNUC server is released. Preliminary tests suggest that working with about 100,000 CDS is possible with a home ADSL connection. See the manual for some `system.time()` examples.
- As pointed by e-mail on 16 Nov 2006 by Emmanuel Prestat the URL used in `dia.bactgensize()` was no more available, this has been fixed in the current version.
- As pointed by e-mail on 16 Nov 2006 by Guy Perrière, the function `oriloc()` was no more compatible with glimmer[1] 3.0 outputs. The function has gained a new argument `glimmer.version` defaulting to 3, but the value 2 is still functional for backward compatibility with old glimmer outputs.

[1] Glimmer is a program to predict coding sequences in microbial genomes [83, 14].

- As pointed by e-mail on 24 Oct 2006 by Lionel Guy (`http://pbil.univ-lyon1.fr/seqinr/seqinrhtmlannuel/03/0089.html`) there was no default value for the `as.string` argument in the `getSequence.SeqFastadna()`. A default `FALSE` value is now present for backward compatibility with older code.

- New utility vectorized function `stresc()` to escape LaTeX special characters present in a string.

- New low level function `readsmj()` available.

- A new function `readfirstrec()` to get the record count of the specified ACNUC index file is now available.

- Function `getType()` called without arguments will now use the default ACNUC database to return available subsequence types.

- Function `read.alignment()` now also accepts `file` in addition to `File` as argument.

- A new function `rearranged.oriloc()` is available. This method, based on `oriloc()`, can be used to detect the effect of the replication mechanism on DNA base composition asymmetry, in prokaryotic chromosomes.

- New function `extract.breakpoints()`, used to extract breakpoints in rearranged nucleotide skews. This function uses the `segmented` package to define the position of the breakpoints.

- New function `draw.rearranged.oriloc()` available, to plot nucleotide skews on artificially rearranged prokaryotic chromosomes.

- New function `gbk2g2.euk()` available. Similarly to `gbk2g2()`, this function extracts the coding sequence annotations from a GenBank format file. This function is specifically designed for eukaryotic sequences, *i.e.* with introns. The output file will contain the coordinates of the exons, along with the name of the CDS to which they belong.

- After an e-mail by Marcelo Bertalan on 26 Mar 2007, a bug in `oriloc()` when the `gbk` argument was `NULL` was found and fixed by Anamaria Necşulea.

- Functions `translate()` and `getTrans()` have gained a new argument `NAstring` to represent untranslatable amino- acids, defaulting to character "X".

- There was a typo for the total number of printed bases in the ACNUC books [22, 23] : 474,439 should be 526,506.

- Function `invers()` has been deleted.

- Functions `translate()`, `getTrans()` and `comp()` have gained a new argument `ambiguous` defaulting to FALSE allowing to handle ambiguous bases. If TRUE, ambiguous bases are taken into account so that for instance GGN is translated to Gly in the standard genetic code.

- New function `amb()` to return the list of nucleotide matching a given IUPAC nucleotide symbol.
- Function `count()` has gained a new argument `alphabet` so that oligopeptides counts are now possible. Thanks to Gabriel Valiente for this suggestion. The functions `zscore()`, `rho()` and `summary.SeqFastadna()` have also an argument `alphabet` which is forwarded to `count()`.

release 1.0-6

Release 1.0-6 is a minor release to fix a problem found and solved by Kurt Hornik (namely a change from `SET_ELEMENT` to `SET_STRING_ELT` in C code for `s2c()` in file `util.c`). The few changes are as follows.

- More typographical option for the output LaTeX table of `tablecode()` are now available to outline deviations from the standard genetic code (see example in the appendix "genetic codes" of the manual).
- A new dataset `aaindex` extracted from the aaindex database [43, 95, 65] is now available. It contains a list of 544 physicochemical and biological properties for the 20 amino-acids
- The default value for argument `dia` is now `FALSE` in function `tablecode()`.
- The example code for `data(chargaff)` has been changed.

release 1.0-5

- A new function `dotPlot()` is now available.
- A new function `crelistfromclientdata()` is now available to create a list on the server from a local file of sequence names, sequence accession numbers, species names, or keywords names.
- A new function `pmw()` to compute the molecular weight of a protein is now available.
- A new function `reverse.align()` contributed by Anamaria Necşulea is now available to align CDS at the protein level and then reverse translate this at the nucleic acid level from a `clustalw` output. This can be done on the fly if `clustalw` is available on your platform.
- An undocumented behavior was reported by Guy Perrière for `uco()` when computing RSCU on sequences where an amino-acid is missing. There is now a new argument `NA.rscu` that allows the user to force the missing values to his favorite magic value.
- There was a bug in `read.fasta()`: some sequence names were truncated, this is now fixed (thanks to Marcus G. Daniels for pointing this). In order to be more consistent with standard functions such as `read.table()` or `scan()`, the file argument starts now with a lower case letter (`file`) in function `read.fasta()`, but the old-style `File` is still functional for forward-compatibility. There is a new logical argument in `read.fasta()`

named `as.string` to allow sequences to be returned as strings instead of vector of single characters. The automatic conversion of DNA sequences into lower case letters can now be disabled with the new logical argument `forceDNAtolower`. It is also possible to disable the automatic attributes settings with the new logical argument `set.attributes`.

- A new function `write.fasta()` is now available.
- The function `kaks()` now forces character in sequences to upper case. This default behavior can be neutralized in order to save time by setting the argument `forceUpperCase` to `FALSE`.

release 1.0-4

- The scaling factor $n_{\bullet\bullet}$ was missing in equation 7.3.
- The files `louse.fasta`, `louse.names`, `gopher.fasta`, `gopher.names` and `ortho.fasta` that were used for examples in the previous version of this document are no more downloaded from the internet since they are now distributed in the `sequences/` folder of the package.
- An example of synonymous and non synonymous codon usage analysis was added to the vignette along with two toy data sets (`toyaa` and `toycodon`).
- A FAQ section was added to the vignette.
- A bug in `getAnnot()` when the number of lines was zero is now fixed.
- There is now a new argument, `latexfile`, in `tablecode()` to export genetic codes tables in a LaTeX document, for instance table 2.2 and table 2.3 here.
- There is now a new argument, `freq`, in `count()` to compute word frequencies instead of counts.
- Function `splitseq()` has been entirely rewritten to improve speed.
- Functions computing the G+C content: `GC(), GC1(), GC2(), GC3()` were rewritten to improve speed, and their document files were merged to facilitate usage.
- The following new functions have been added:
  - `syncodons()` returns all synonymous codons for a given codon. Argument `numcode` specifies the desired genetic code.
  - `ucoweight()` returns codon usage bias on a sequence as the number of synonymous codons present in the sequence for each amino acid.
  - `synsequence()` generates a random coding sequence which is synonymous to a given sequence and has a chosen codon usage bias.
  - `permutation()` generates a new sequence from a given sequence, while maintaining some constraints from the given sequence such as nucleotide frequency, codon usage bias, ...
  - `rho()` computes the rho statistic on dinucleotides as defined in [41].

  - `zscore()` computes the zscore statistic on dinucleotides as defined in [69].

- Two datasets (`dinucl` and `prochlo`) were added to illustrate these new functions.

release 1.0-3

- The new package maintainer is Dr. Simon Penel, PhD, who has now a fixed position in the laboratory that issued **seqinR** (`penel@biomserv.univ-lyon1.fr`). Delphine Charif was successful too to get a fixed position in the same lab, with now a different research task (but who knows?). Thanks to the close vicinity of our pioneering maintainers the transition was sweet. The DESCRIPTION file of the **seqinR** package has been updated to take this into account.

- The reference paper for the package is now *in press*. We do not have the full reference for now, you may use `citation("seqinr")` to check if it is complete now:

```
 citation("seqinr")
To cite seqinR in publications use:

  Charif, D. and Lobry, J.R. (2007)

Une entrée BibTeX pour les utilisateurs LaTeX est

  @InCollection{,
    author = {D. Charif and J.R. Lobry},
    title = {Seqin{R} 1.0-2: a contributed package to the {R} project for statistical computing deve
    booktitle = {Structural approaches to sequence evolution: Molecules, networks, populations},
    year = {2007},
    editor = {U. Bastolla and M. Porto and H.E. Roman and M. Vendruscolo},
    series = {Biological and Medical Physics, Biomedical Engineering},
    pages = {207-232},
    address = {New York},
    publisher = {Springer Verlag},
    note = {{ISBN :} 978-3-540-35305-8},
  }
```

- There was a bug when sending a `gfrag` request to the server for long (Mb range) sequences. The length argument was converted to scientific notations that are not understand by the server. This is now corrected and should work up the the Gb scale.

- The `query()` function has been improved by de-looping list element info request, there are now download at once which is much more efficient. For example, a query from a researcher-home ADSL connection with a list with about 1000 elements was 60 seconds and is now only 4 seconds (*i.e.* 15 times faster now).

- A new parameter `virtual` has been added to `query()` so that long lists can stay on the server without trying to download them automatically. A query like `query(s$socket,"allcds","t=cds", virtual = TRUE)` is now possible.

- Relevant genetic codes and frames are now automatically propagated.

- **`SeqinR`** sends now its name and version number to the server.
- Strict control on ambiguous DNA base alphabet has been relaxed.
- Default value for parameter `invisible` of function `query()` is now `TRUE`.

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

There were two compilation steps:

- R compilation time was: Thu Jun 2 16:39:37 2016
- LaTeX compilation time was: June 2, 2016

CHAPTER 12

Genetic codes

Lobry, J.R.

12.1 Standard genetic code

The standard genetic code given in table 12.1 was produced with the following R code and inserted with `\input{../tables/stdcode.tex}` within this LaTeX document and referenced as `\ref{stdcode}` in the text.

```
tablecode( latexfile = "../tables/stdcode.tex",
label = "stdcode", size = "small")
```

12.2 Available genetic code numbers

The genetic code numbers are those from the NCBI[1] (`http://130.14.29.110/Taxonomy/Utils/wprintgc.cgi?mode=c`). This compilation from Andrzej (Anjay) Elzanowski, Jim Ostell, Detlef Leipe, and Vladimir Soussov is based primarily on two previous reviews [66, 39].

```
codes <- SEQINR.UTIL$CODES.NCBI
availablecodes <- which(codes$CODES != "deleted")
codes[availablecodes,"ORGANISMES", drop = FALSE]
```

```
                             ORGANISMES
1                              standard
2             vertebrate.mitochondrial
3                  yeast.mitochondrial
4  protozoan.mitochondrial+mycoplasma
5           invertebrate.mitochondrial
6                 ciliate+dasycladacean
9    echinoderm+flatworm.mitochondrial
10                             euplotid
11              bacterial+plantplastid
12                     alternativeyeast
13              ascidian.mitochondrial
14   alternativeflatworm.mitochondrial
15                           blepharism
16          chlorophycean.mitochondrial
21              trematode.mitochondrial
```

[1] National Center for Biotechnology Information, Bethesda, Maryland, U.S.A.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.1: Genetic code number 1: standard.

```
22          scenedesmus.mitochondrial
23       hraustochytrium.mitochondria
```

The tables of variant genetic codes outlining the differences were produced with the following R code:

```
cdorder <- paste(paste(rep(s2c("tcag"), each =16), s2c("tcag"), sep = ""), rep(s2c("tcag"), each = 4), s
stdcode <- sapply(lapply(cdorder,s2c),translate, numcode = 1)
for(cd in availablecodes[-1]){
  Tfile <- paste("../tables/codnum", cd, ".tex", sep = "")
  preemph <- "\\textcolor{red}{\\textbf{"
  postemph <- "}}"
  stcodon <- (stdcode ==  sapply(lapply(cdorder,s2c),translate, numcode = cd))
  pre <- ifelse(stcodon, "", preemph)
  post <- ifelse(stcodon, "", postemph)
  tablecode(numcode = cd, latexfile = Tfile, size = "small", preaa = pre, postaa = post)
  cat(paste("\\input{", Tfile, "}", sep = ""), sep = "\n")
}
```

Session Informations

This part was compiled under the following R environment:

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ade4 1.7-4, ape 3.5, grImport 0.9-0, MASS 7.3-45, seqinr 3.1-5, tseries 0.10-35, XML 3.98-1.4, xtable 1.8-2
- Loaded via a namespace (and not attached): lattice 0.20-33, nlme 3.1-125, quadprog 1.5-5, tools 3.2.4, zoo 1.7-12

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	**Met**	ACA	Thr	AAA	Lys	**AGA**	**Stp**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Stp**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.2: Genetic code number 2: vertebrate.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	**Thr**	CCT	Pro	CAT	His	CGT	Arg
CTC	**Thr**	CCC	Pro	CAC	His	CGC	Arg
CTA	**Thr**	CCA	Pro	CAA	Gln	CGA	Arg
CTG	**Thr**	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	**Met**	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.3: Genetic code number 3: yeast.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.4: Genetic code number 4: protozoan.mitochondrial+mycoplasma.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	**Met**	ACA	Thr	AAA	Lys	**AGA**	**Ser**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Ser**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.5: Genetic code number 5: invertebrate.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	**TAA**	**Gln**	TGA	Stp
TTG	Leu	TCG	Ser	**TAG**	**Gln**	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.6: Genetic code number 6: ciliate+dasycladacean.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	**AAA**	**Asn**	**AGA**	**Ser**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Ser**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.7: Genetic code number 9: echinoderm+flatworm.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Cys**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.8: Genetic code number 10: euplotid.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.9: Genetic code number 11: bacterial+plantplastid.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	**Ser**	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.10: Genetic code number 12: alternativeyeast.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	**Met**	ACA	Thr	AAA	Lys	**AGA**	**Gly**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Gly**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.11: Genetic code number 13: ascidian.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	**TAA**	**Tyr**	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	**AAA**	**Asn**	**AGA**	**Ser**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Ser**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.12: Genetic code number 14: alternativeflatworm.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	**TAG**	**Gln**	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.13: Genetic code number 15: blepharism.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	**TAG**	**Leu**	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.14: Genetic code number 16: chlorophycean.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	TCA	Ser	TAA	Stp	**TGA**	**Trp**
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	**Met**	ACA	Thr	**AAA**	**Asn**	**AGA**	**Ser**
ATG	Met	ACG	Thr	AAG	Lys	**AGG**	**Ser**
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.15: Genetic code number 21: trematode.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	Leu	**TCA**	**Stp**	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	**TAG**	**Leu**	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.16: Genetic code number 22: scenedesmus.mitochondrial.

TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys
TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys
TTA	**Stp**	TCA	Ser	TAA	Stp	TGA	Stp
TTG	Leu	TCG	Ser	TAG	Stp	TGG	Trp
CTT	Leu	CCT	Pro	CAT	His	CGT	Arg
CTC	Leu	CCC	Pro	CAC	His	CGC	Arg
CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser
ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly
GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Table 12.17: Genetic code number 23: hraustochytrium.mitochondria.

There were two compilation steps:

- R compilation time was: Wed Jun 1 17:53:09 2016
- LaTeX compilation time was: June 2, 2016

BIBLIOGRAPHY

[1] S.G. Andersson, A. Zomorodipour, J.O. Andersson, T. Sicheritz-Ponten, U.C. Alsmark, R.M. Podowski, A.K. Naslund, A.S. Eriksson, H.H. Winkler, and C.G. Kurland. The genome sequence of *Rickettsia prowazekii* and the origin of mitochondria. *Nature*, 396:133–140, 1998. 64

[2] A.L. Bak, J.F. Atkins, C.E. Singer, and B.N. Ames. Evolution of dna base compositions in microorganisms. *Science*, 175:1391–1393, 1972. 113, 118

[3] W. Bar, B. Brinkmann, P. Lincoln, W.R. Mayr, and U. Rossi. DNA recommendations. 1994 report concerning further recommendations of the DNA commission of the ISFH regarding PCR-based polymorphisms in STR (short tandem repeat) systems. *Int. J. Leg. Med.*, 107:159–160, 1994. 155

[4] A. Bernal, U. Ear, and N. Kyrpides. Genomes online database (GOLD): a monitor of genome projects world-wide. *Nucleic Acids Research*, 29:126–127, 2001. 159

[5] F.R. Blattner, V. Burland, G. Plunkett, H.J. Sofia, and D.L. Daniels. Analysis of the *Escherichia coli* genome. IV. DNA sequence of the region from 89.2 to 92.8 minutes. *Nucleic Acids Research*, 21:5408–5417, 1993. 132

[6] F.R. Blattner, G. Plunkett III, C.A. Bloch, N.T. Perna, V. Burland, M. Rilley, J. Collado-Vides, J.D. Glasner, C.K. Rode, G.F. Mayhew, J. Gregor, N.W. Davis, H.A. Kirkpatrick, M.A. Goeden, D.J. Rose, B. Mau, and Y. Shao. The complete genome sequence of *Escherichia coli* K-12. *Science*, 277:1453–1462, 1997. 132

[7] J. Buckheit and D. L. Donoho. *Wavelets and Statistics*, chapter Wavelab and reproducible research. Springer-Verlag, Berlin, New York, 1995. A. Antoniadis editor. 17

[8] V. Burland, G. Plunkett, D.L. Daniels, and F.R. Blattner. DNA sequence and analysis of 136 kilobases of the *Escherichia coli* genome: organizational symmetry around the origin of replication. *Genomics*, 16:551–561, 1993. 132

[9] D. Charif and J.R. Lobry. SeqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis. In H.E. Roman U. Bastolla, M. Porto and M. Vendruscolo, editors, *Structural approaches to sequence evolution: Molecules, networks, populations*, Biological and Medical Physics, Biomedical Engineering, pages 207–232. Springer Verlag, New York, USA, 2007. ISBN 978-3-540-35305-8. 15

[10] D. Charif, J. Thioulouse, J.R. Lobry, and G. Perrière. Online synonymous codon usage analyses with the ade4 and seqinR packages. *Bioinformatics*, 21(4):545–7, 2005. 17

[11] J.-L. Chassé. *Modélisation statistique : statistique non paramétrique (fiches de cours)*. Laboratoire de Biométrie et Biologie Évolutive, Lyon, France, 1988. 1988 for the publication year is an upper limit: could be earlier. 105

[12] D.B Dahl and *et al. xtable: Export tables to LaTeX or HTML*, 2005. R package version 1.3-0. 21

[13] D.L. Daniels, G. Plunkett, V. Burland, and F.R. Blattner. Analysis of the *Escherichia coli* genome: DNA sequence of the region from 84.5 to 86.5 minutes. *Science*, 257:771–778, 1992. 132

[14] A.L. Delcher, D. Harmon, S. Kasif, O. White, and S.L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27:4636–4641, 1999. 162

[15] A. Dufresne, M. Salanoubat, F. Partensky, F. Artiguenave, I.M. Axmann, V. Barbe, S. Duprat, M.Y. Galperin, E.V. Koonin, F. Le Gall, K.S. Makarova, M. Ostrowski, S. Oztas, C. Robert, I.B. Rogozin, D.J. Scanlan, N. Tandeau de Marsac, J. Weissenbach, P. Wincker, Y.I. Wolf, and W.R. Hess. Genome sequence of the cyanobacterium *Prochlorococcus marinus* ss120, a nearly minimal oxyphototrophic genome. *Proceedings of the National Academy of Sciences of the United States of America*, 100:10020–10025, 2003. 126

[16] Duncan Temple Lang (duncan@wald.ucdavis.edu). *XML: Tools for parsing and generating XML within R and S-Plus*, 2006. R package version 0.99-8. 3

[17] J. Felsenstein. PHYLIP-phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989. 42

[18] A.C. Frank and J.R. Lobry. Oriloc: prediction of replication boundaries in unannotated bacterial chromosomes. *Bioinformatics*, 16(6):560–561, 2000. 28

[19] N. Galtier, M. Gouy, and C. Gautier. SeaView and Phylo_win, two graphic tools for sequence alignment and molecular phylogeny. *Comput. Applic. Biosci.*, 12:543–548, 1996. 40, 41

[20] A. Garay-Arroyo, J.M. Colmenero-Flores, A. Garciarrubio, and A.A. Covarrubias. Highly hydrophilic proteins in prokaryotes and eukaryotes are common during conditions of water deficit. *J. Biol. Chem.*, 275:5668–5674, 2000. 31

[21] C. Gautier. *Analyses statistiques et évolution des séquences d'acides nucléiques.* PhD thesis, Université Claude Bernard - Lyon I, 1987. 83

[22] C. Gautier, M. Gouy, M. Jacobzone, and R. Grantham. *Nucleic acid sequences handbook. Vol. 1.* Praeger Publishers, London, UK, 1982. ISBN 0-275-90798-8. 13, 14, 49, 163

[23] C. Gautier, M. Gouy, M. Jacobzone, and R. Grantham. *Nucleic acid sequences handbook. Vol. 2.* Praeger Publishers, London, UK, 1982. ISBN 0-275-90799-6. 13, 14, 49, 163

[24] C. Gautier, M. Gouy, and S. Louail. Non-parametric statistics for nucleic acid sequence study. *Biochimie*, 67:449–453, 1985. 116, 158

[25] S.J. Gould. *Wonderful life.* Norton, New York, USA, 1989. 2

[26] S.J. Gould. Ladders and cones: Constraining evolution by canonical icons. In R.B. Silvers, editor, *Hidden Histories of Science*, pages 37–67, New York, USA, 1995. New York Review of Books. 2

[27] M. Gouy and S. Delmotte. Remote access to ACNUC nucleotide and protein sequence databases at PBIL. *Biochimie*, 90:555–562, 2008. 61

[28] M. Gouy, C. Gautier, M. Attimonelli, C. Lanave, and G. di Paola. ACNUC-a portable retrieval system for nucleic acid sequence databases: logical and physical designs and usage. *Computer Applications in the Biosciences*, 1:167–172, 1985. 49, 61

[29] M. Gouy, C. Gautier, and F. Milleret. System analysis and nucleic acid sequence banks. *Biochimie*, 67:433–436, 1985. 49, 61

[30] M. Gouy, F. Milleret, C. Mugnier, M. Jacobzone, and C. Gautier. ACNUC: a nucleic acid sequence data base and analysis system. *Nucleic Acids Res.*, 12:121–127, 1984. 3, 49, 61, 105

[31] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185:862–864, 1974. 3

[32] M.A. Hannah, A.G. Heyer, and D.K. Hincha. A global survey of gene regulation during cold acclimation in *Arabidopsis thaliana*. *PLoS Genet.*, 1:e26, 2005. 31, 36, 38, 40

[33] K. Hayashi, N. Morooka, Y. Yamamoto, K. Fujita, K. Isono, S. Choi, E. Ohtsubo, T. Baba, B.L. Wanner, H. Mori, and T. Horiuchi. Highly accurate genome sequences of *Escherichia coli* K-12 strains MG1655 and W3110. *Molecular Systems Biology*, 2:2006.0007, 2006. 132

[34] D. G. Higgins and P. M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988. 41

[35] K. Hornik. *The R FAQ: Frequently Asked Questions on R (version 2.3.2006-07-13)*, 2006. ISBN 3-900051-08-9 `http://CRAN.R-project.org/doc/FAQ/`. 16

[36] L.D. Hurst. The Ka/Ks ratio: diagnosing the form of sequence evolution. *Trends Genet.*, 18:486–487, 2002. 92

[37] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *J. Comp. Graph. Stat.*, 3:299–314, 1996. 14, 15

[38] M. Jacobzone and C. Gautier. *ANALSEQ Manuel d'utilisation.* UMR CNRS 5558, Biométrie, Génétique et Biologie des Populations, 1989. 3, 105

[39] T. H. Jukes and S. Osawa. Evolutionary changes in the genetic code. *Comp. Biochem. Physiol. B.*, 106:489–494, 1993. 169

[40] T.H. Jukes and C.R. Cantor. Evolution of protein molecules. In H.N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132, New York, 1969. Academic Press. 46, 47

[41] S. Karlin and V. Brendel. Chance and statistical significance in protein and DNA sequence analysis. *Science*, 257:39–49, 1992. 113, 165

[42] S. Karlin and L.R. Cardon. Computational DNA sequence analysis. *Annual Review of Microbiology*, 48:619–654, 1994. 152

[43] S. Kawashima and M. Kanehisa. AAindex: amino acid index database. *Nucleic Acids Res.*, 28:374–374, 2000. 3, 164

[44] J. Keogh. Circular transportation facilitation device, 2001. 16

[45] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120, 1980. 47

[46] J. Kiraga. *Analysis and computer simulations of variability of isoelectric point of proteins in the proteomes.* PhD thesis, University of Wrocław, 2008. 156

[47] J. Krawczyk, A. Goesmann, R. Nolte, M. Werber, and B. Weisshaar. Trace2PS and FSA2PS: two software toolkits for converting trace and fsa files to PostScript format. *Source Code for Biology and Medicine*, 4:4, 2009. 154

[48] N.C. Kyrpides. Genomes online database (GOLD 1.0): a monitor of complete and ongoing genome projects world-wide. *Bioinformatics*, 15:773–774, 1999. 159

[49] J. Kyte and R.F. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157:105–132, 1982. 35, 89

[50] P. Legendre, Y. Desdevises, and E. Bazin. A statistical test for host-parasite coevolution. *Syst. Biol.*, 51:217–234, 2002. 45

[51] F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. *Proceedings in Computational Statistics*, Compstat 2002:575–580, 2002. 15, 21

[52] W.-H. Li. Unbiased estimation of the rates of synonymous and nonsynonymous substitution. *J. Mol. Evol.*, 36:96–99, 1993. 92

[53] K. Liolios, K. Mavrommatis, N. Tavernarakis, and N.C. Kyrpides. The genomes on line database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata. *Nucleic Acids Research*, in press:D000–D000, 2008. 159

[54] K. Liolios, N. Tavernarakis, P. Hugenholtz, and N.C. Kyrpides. The genomes on line database (GOLD) v.2: a monitor of genome projects worldwide. *Nucleic Acids Research*, 34:D332–D334, 2006. 159

[55] R.A. Litjens, T.I. Quickenden, and C.G. Freeman. Visible and near-ultraviolet absorption spectrum of liquid water. *Applied Optics*, 38:1216–1223, 1999. 126, 159

[56] J.R. Lobry. Asymmetric substitution patterns in the two DNA strands of bacteria. *Molecular Biology and Evolution*, 13:660–665, 1996. 131, 132, 159

[57] J.R. Lobry. Life history traits and genome structure: aerobiosis and G+C content in bacteria. *Lecture Notes in Computer Sciences*, 3039:679–686, 2004. 19, 21

[58] J.R. Lobry and D. Chessel. Internal correspondence analysis of codon and amino-acid usage in thermophilic bacteria. *Journal of Applied Genetics*, 44:235–261, 2003. 93, 153

[59] J.R. Lobry and C. Gautier. Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 *Escherichia coli* chromosome-encoded genes. *Nucleic Acids Res*, 22:3174–80, 1994. 87, 97, 98, 99

[60] J.R. Lobry and N. Sueoka. Asymmetric directional mutation pressures in bacteria. *Genome Biology*, 3(10):research0058.1–research0058.14, 2002. 16

[61] A.O. Lovejoy. *The Great Chain of Being: A Study of the History of an Idea.* Harvard University Press, Cambridge, Massachusetts, USA, 1936. 2

[62] P. Mackiewicz, J. Zakrzewska-Czerwińska, A. Zawilak, M.R. Dudek, and S. Cebrat. Where does bacterial replication start? rules for predicting the *oriC* region. *Nucleic Acids Research*, 32:3781–3791, 2004. 29

[63] P. Murrell. *R Graphics.* Computer Science & Data Analysis. Chapman & Hall/CRC, New York, 2005. ISBN: 9781584884866 `http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`. 132

[64] Paul Murrell and Richard Walton. *grImport: Importing Vector Graphics*, 2006. R package version 0.2. 3

[65] K. Nakai, A. Kidera, and M. Kanehisa. Cluster analysis of amino acid indices for prediction of protein structure and function. *Protein Eng.*, 2:93–100, 1988. 3, 164

[66] S. Osawa, T. H. Jukes, K. Watanabe, and A. Muto. Recent evidence for evolution of the genetic code. *Microbiol. Rev.*, 56:229–264, 1992. 169

[67] H. Pages, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*, 2007. R package version 2.6.4. 159

[68] L. Palmeira. *Analyse et modélisation des dépendances entre sites voisins dans l'évolution des séquences d'ADN*. PhD thesis, Université Claude Bernard - Lyon I, 2007. 126, 159

[69] L. Palmeira, L. Guéguen, and J.R. Lobry. UV-targeted dinucleotides are not depleted in light-exposed prokaryotic genomes. *Molecular Biology and Evolution*, 23:2214–2219, 2006. 113, 115, 118, 124, 166

[70] E. Paradis, J. Claude, and K. Strimmer. Ape: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004. 46

[71] J. Pačes, R. Zíka, V. Pačes, A. Pavlíček, O. Clay, and G. Bernardi. Representing GC variation along eukaryotic chromosomes. *Gene*, 333:135–141, 2004. 106

[72] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2444–2448, 1988. 23

[73] J.F. Peden. *Analysis of codon usage*. PhD thesis, University of Nottingham, 1999. 135, 157

[74] G. Perrière and J. Thioulouse. Use and misuse of correspondence analysis in codon usage studies. *Nucleic Acids Res.*, 30:4548–4555, 2002. 83, 93

[75] G. Plunkett, V. Burland, D.L. Daniels, and F.R. Blattner. Analysis of the *Escherichia coli* genome. III. DNA sequence of the region from 87.2 to 89.2 minutes. *Nucleic Acids Research*, 21:3391–3398, 1993. 132

[76] T.I. Quickenden and J.A. Irvin. The ultraviolet absorption spectrum of liquid water. *The Journal of Chemical Physics*, 72:4416–4428, 1980. 126, 159

[77] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. 14, 15

[78] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0. 3

[79] Trina E. Roberts. *ComPairWise: Compare phylogenetic or population genetic data alignments*, 2007. R package version 1.01. 157

[80] G. Rocap, F.W. Larimer, J. Lamerdin, S. Malfatti, P. Chain, N.A. Ahlgren, A. Arellano, M. Coleman, L. Hauser, W.R. Hess, Z.I. Johnson, M. Land, D. Lindell, A.F. Post, W. Regala, M. Shah, S.L. Shaw, C. Steglich, M.B. Sullivan, C.S. Ting, A. Tolonen, E.A. Webb, E.R. Zinser, and S.W. Chisholm. Genome divergence in two *Prochlorococcus* ecotypes reflects oceanic niche differentiation. *Nature*, 424:1042–1047, 2003. 126

[81] R. Rudner, J.D. Karkas, and E. Chargaff. Separation of microbial deoxyribonucleic acids into complementary strands. *Proceedings of the National Academy of Sciences of the United States of America*, 63:152–159, 1969. 17

[82] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1984. 46

[83] S.L. Salzberg, A.L. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26:544–548, 1998. 162

[84] Sophie Schbath. *Étude asymptotique du nombre d'occurrences d'un mot dans une chaîne de Markov et application à la recherche de mots de fréquence exceptionnelle dans les séquences d'ADN.* PhD thesis, Université René Descartes, Paris V, 1995. 116

[85] R. B. Setlow. Cyclobutane-type pyrimidine dimers in polynucleotides. *Science*, 153:379–386, 1966. 118

[86] P.M. Sharp and E. Cowe. Synonymous codon usage in *Saccharomyces cerevisiae*. *Yeast*, 7:657–678, 1991. 157

[87] P.M. Sharp and W.-H. Li. The codon adaptation index - a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, 15:1281–1295, 1987. 98, 157

[88] D.C. Shields and P.M. Sharp. Synonymous codon usage in *Bacillus subtilis* reflects both translational selection and mutational biases. *Nucleic Acids Research*, 15:8023–8040, 1987. 157

[89] C.E. Singer and B.N. Ames. Sunlight ultraviolet and bacterial DNA base ratios. *Science*, 170:822–826, 1970. 113, 118, 124

[90] H.J. Sofia, V. Burland, D.L. Daniels, G. Plunkett, and F.R. Blattner. Analysis of the *Escherichia coli* genome. V. DNA sequence of the region from 76.0 to 81.5 minutes. *Nucleic Acids Research*, 22:2576–2586, 1994. 132

[91] R. Staden. Graphic methods to determine the function of nucleic acid sequences. *Nucleic Acids Res.*, 12:521–538, 1984. 3

[92] N. Sueoka. Directional mutation pressure and neutral molecular evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2653 –2657, 1988. 139

[93] N. Sueoka. Two aspects of DNA base composition: G+C content and translation-coupled deviation from intra-strand rule of $A = T$ and $G = C$. *J. Mol. Evol.*, 49:49–62, 1999. 140

[94] H. Suzuki, C.J. Brown, L.J. Forney, and E. Top. Comparison of correspondence analysis methods for synonymous codon usage in bacteria. *DNA Research*, 15:357–365, 2008. 153

[95] K. Tomii and M. Kanehisa. Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins. *Protein Eng.*, 9:27–36, 1996. 3, 164

[96] Adrian Trapletti and Kurt Hornik. *tseries: Time Series Analysis and Computational Finance*, 2007. R package version 0.10-11. 112

[97] I.M. Wallace, G. Blackshields, and D.G. Higgins. Multiple sequence alignments. *Curr. Opin. Struct. Biol.*, 15:261–266, 2005. 41

[98] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1:80–83, 1945. 108

[99] T. Yura, H. Mori, H. Nagai, T. Nagata, A. Ishihama, N. Fujita, K. Isono, K. Mizobuchi, and A. Nakata. Systematic sequencing of the *Escherichia coli* genome: analysis of the 0-2.4 min region. *Nucleic Acids Research*, 20:3305–3308, 1992. 132