SeqinR 1.1–3

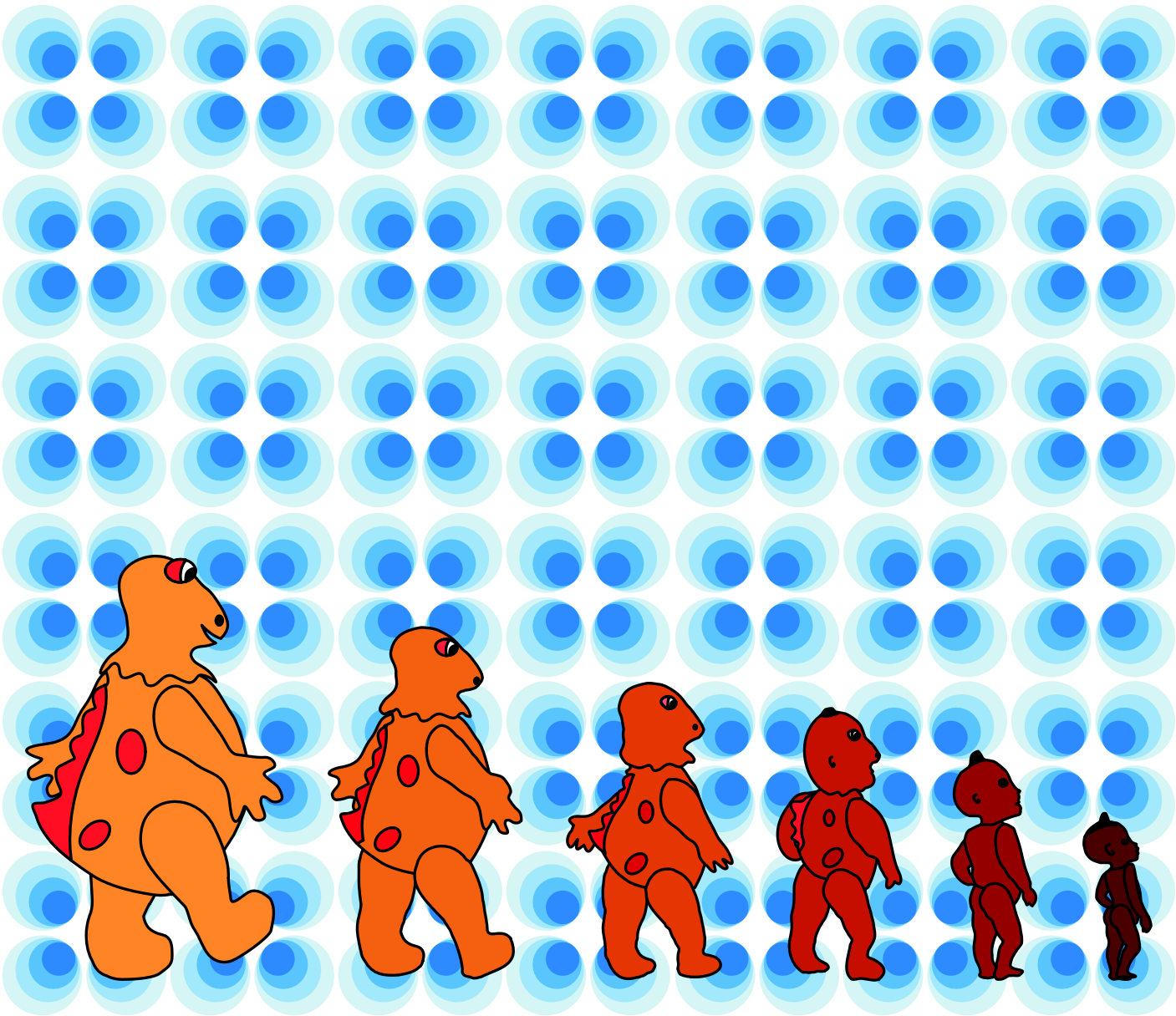


Figure 1: The march of progress icon is very common in popular press. This example is from page 46 of a 1984 summer issue of the tchek edition of *Playboy*.

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" [21]. 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* [20].

Note that the underlying conception predates Darwin [50]. 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

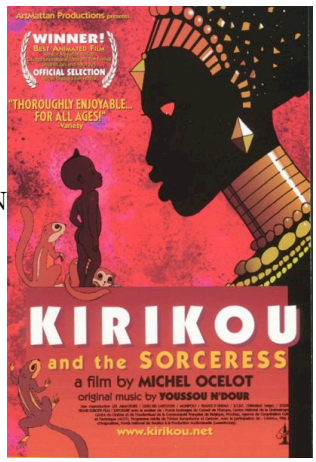
[1] with Canvas from ACD Systems.

lab about molecular evolution [26]. 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] [33] 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 [25]), there is only a reference manual in french [33] 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.



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 code based on `grid` [66], `XML` [12] and `grImport` [53]:

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

[2] thanks to `aaindex` database [37, 79, 54].
[3] not to be confused with the ANALYSEQ program by Rodger Staden [76].
[4] Comprehensive R Archive Network.
[5] Encapsulated Postscrit.
[6] RDF (Resource Description Framework) Graph Modeling Language (http://www.cs.rpi.edu/~puninj/RGML/).

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

```
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 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, grid, methods, stats, utils
- Other packages: MASS 7.2-36, XML 1.93-2, ade4 1.4-4, ape 2.0-1, gee 4.13-13, grImport 0.2-4, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): rcompgen 0.1-15, tools 2.6.0

There were two compilation steps:

- R compilation time was: Mon Oct 29 17:40:22 2007
- LaTeX compilation time was: December 2, 2007

SeqinR 1.1-3: 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.

December 2, 2007

CONTENTS

Licence of this document

Licence

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 `doc/src/template` folder which is distributed with the seqinR package.

CHAPTER 1

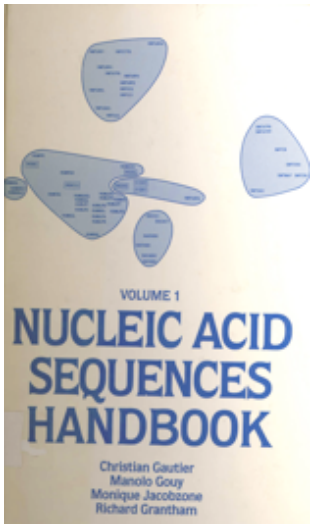
Introduction

Lobry, J.R.

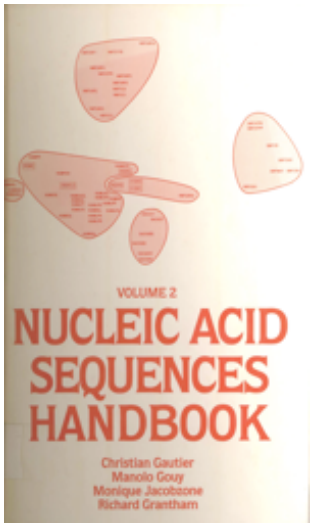
Contents

1.1 About ACNUC

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 [18, 19] 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



Cover of ACNUC book vol. 1



Cover of ACNUC book vol. 2

[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

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 [18, 19] 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 1.1.

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 1.1: The list of journals that were manually scanned for nucleic sequences that were included in the ACNUC books [18, 19]

ACNUC books are about 4.5 cm width

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
acnucbp <- 526506
mybank <- choosebank("genbank")
closebank()
mybank$details
```

```
[1] "             ****     ACNUC Data Base Content      ****                          "
[2] "         GenBank Rel. 162 (15 October 2007) Last Updated: Oct 30, 2007"
[3] "81,770,960,650 bases; 77,863,951 sequences; 4,534,640 subseqs; 481,662 refers."
[4] "Software by M. Gouy, Lab. Biometrie et Biologie Evolutive, Universite Lyon I "
```

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

```
[1] "81,770,960,650"
```

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

```
[1] 6.988891
```

It would be about 7 kilometer long in ACNUC book format to print GenBank today (December 2, 2007).

[6] National Institute of Genetics

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

1.2 About R and CRAN

R [32, 67] 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 60 mirrors available from 29 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()`.

1.3 About this document

In the terminology of the R project [32, 67], 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 2.6.0 (2007-10-03)` on Tue Oct 30 09:08:05 2007 with Sweave [42]. There is a section at the end of each chapter called **Session Informations** that gives details about packages and package versions that were involved[8]. The last compiled version of this document is distributed along with the **seqinR** package in the `/doc` folder. Once **seqinR** has been installed, the full path to the package is given by the following R code :

```
.find.package("seqinr")
```

```
[1] "/Users/lobry/seqinr.Rcheck/seqinr"
```

1.4 About sequin and seqinR

Sequin is the well known sofware used to submit sequences to GenBank, **seqinR** [7] 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.

1.5 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

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

(`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] "AAstat"      "EXP"          "GC"           "GC1"          "GC2"
[6] "GC3"         "SEQINR.UTIL" "a"            "aaa"
```

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.

1.6 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 :

```
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 [30] 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")`).

1.7 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.

1.7.1 Wheel (the)

Do not re-invent (there's a patent [38] on it anyway). At the compilation time of this document there were 1165 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.

1.7.2 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 `fortunes` package.

1.7.3 Automation

Consider the 178 pages of figures in the additional data file 1 (`http://genomebiology.com/2002/3/10/research/0058/suppl/S1`) from [49]. 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 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.

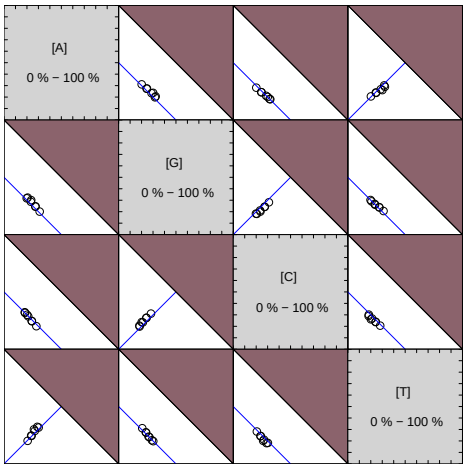
1.7.4 Reproducibility

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 [5] 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 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 [8].

1.7.5 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 [68] 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)
```

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 1.2: Genetic code number 3: yeast.mitochondrial.

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 1.2 and table 1.3 were automatically generated with the following R code:

```
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}
```

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 1.3: Genetic code number 4: protozoan.mitochondrial+mycoplasma.

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

1.7.6 Data as fast moving targets

In research area, data are not always stable. Consider figure 1 from [46] which is reproduced here in figure 1.1. Data have been updated since then, but we can re-use the same code[9] to update the figure:

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

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

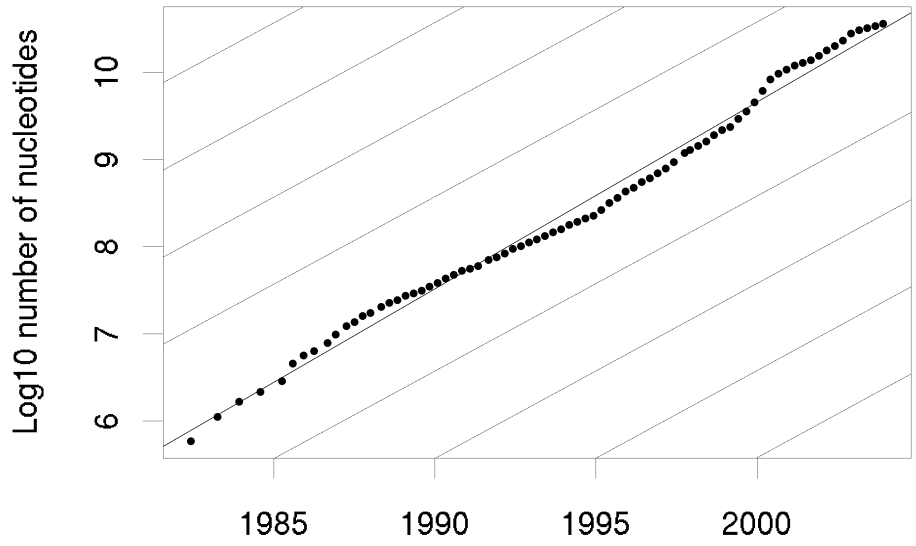
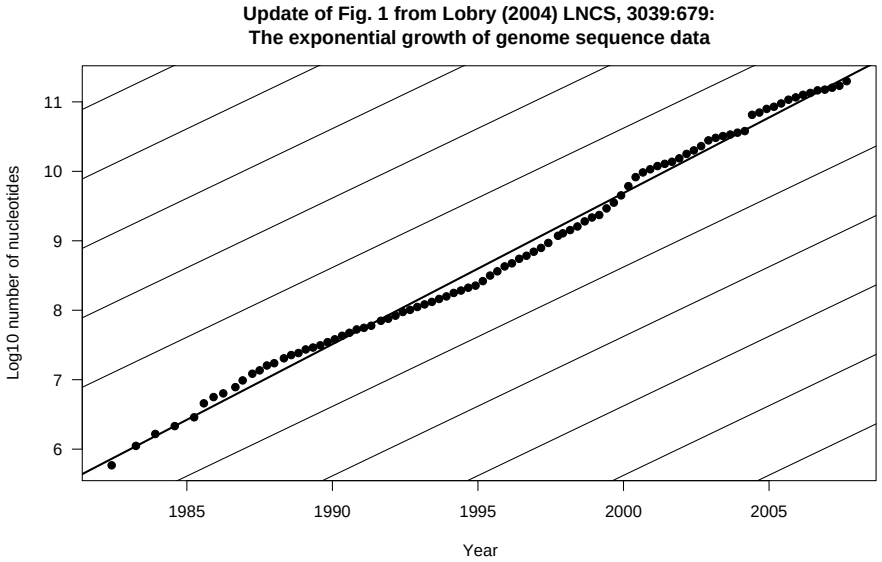


Figure 1.1: Screenshot of figure 1 from [46]. The exponential growth of genomic sequence data mimics Moore's law. The source of data is the december 2003 release note (realnote.txt) from the EMBL database available at http://www.ebi.ac.uk/. External lines correspond to what would be expected with a doubling time of 18 months. The central line through points is the best least square fit, corresponding to a doubling time of 16.9 months.



The doubling time is now 16.6 months.

1.7.7 `Sweave()` and `xtable()`

For LaTeX users, it's worth mentioning the fantastic tool contributed by Friedrich Leish [42] 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` [9] to coerce R data into LaTeX tables.

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15, tools 2.6.0

There were two compilation steps:

- R compilation time was: Tue Oct 30 09:08:08 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 2

Importing sequences from flat files

Charif, D. Lobry, J.R.

Contents

2.1 Importing raw sequence data from FASTA files

2.1.1 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 [61]. It begins with a single-line description starting with a character `>`, followed by lines of sequence data of maximum 80 character each. 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] "ATH1_pep_cm_20040228.fasta" "Anouk.fasta"
 [3] "Sakai.fasta"                "bb.fasta"
 [5] "bordetella.fasta"           "ct.fasta"
 [7] "ecolicgpe5.fasta"           "edl933.fasta"
 [9] "gopher.fasta"               "louse.fasta"
[11] "malM.fasta"                 "ortho.fasta"
[13] "seqAA.fasta"                "smallAA.fasta"
```

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*
```

2.1.2 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
```

2.1.3 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)
```

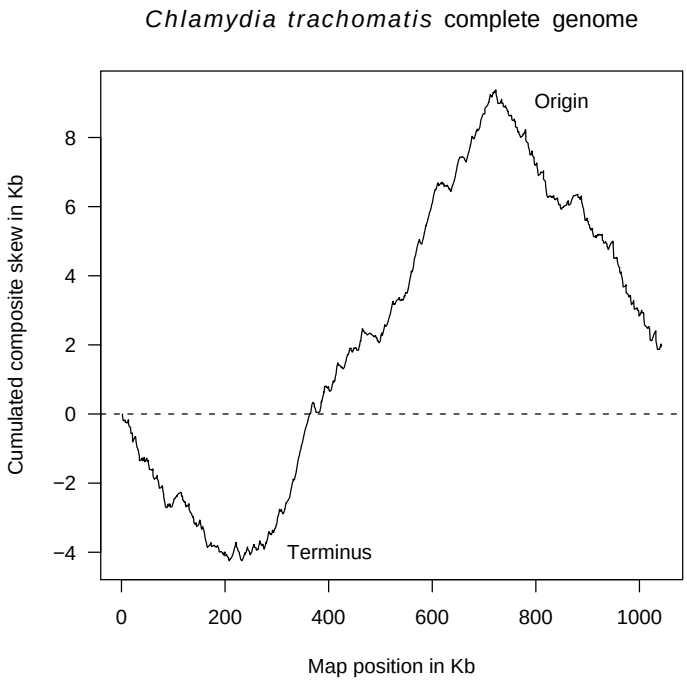
```
[1] TRUE
```

2.1.4 Big room examples

Oriloc example (*Chlamydia trachomatis* complete genome)

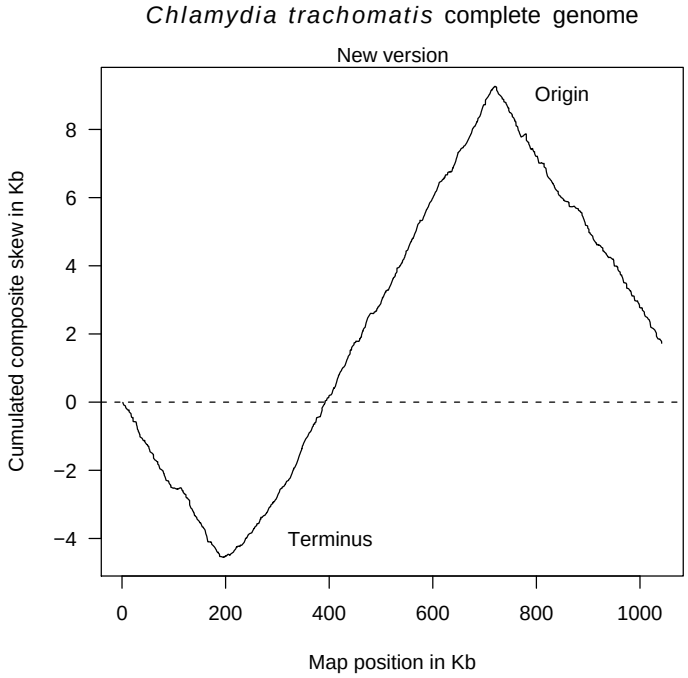
A more consequent example is given in the fasta file `ct.fasta` which contains the complete genome of *Chlamydia trachomatis* that was used in [14]. You should be able to reproduce figure 1b from this paper with the following code:

```
out <- oriloc(seq.fasta = system.file("sequences/ct.fasta",
    package = "seqinr"), g2.coord = system.file("sequences/ct.coord",
    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 [51] for a recent review on this topic.

```
out <- oriloc(seq.fasta = system.file("sequences/ct.fasta",
    package = "seqinr"), g2.coord = system.file("sequences/ct.coord",
    package = "seqinr"))
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")
```



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 [27], 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 [16], 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 <- system.file("sequences/ATH1_pep_cm_20040228.fasta",
    package = "seqinr")
system.time(ath <- read.fasta(athfile, seqtype = "AA", as.string = TRUE,
    set.attributes = FALSE))
```

```
utilisateur     syst`eme     'ecoul'e
     10.178        0.048       10.247
```

It's about one minute here to read 21,161 protein sequences. We save them in XDR binary format to read them faster later at will:

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

```
system.time(load("ath.RData"))
```

```
utilisateur     syst`eme     'ecoul'e
      0.392        0.009        0.402
```

Now it's about one 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
```

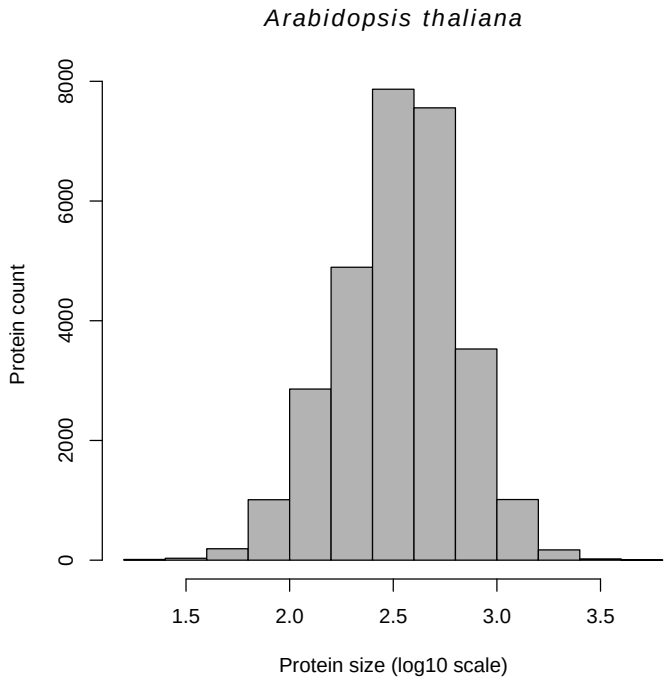
```
[1] 14.65537
```

```
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)))
```



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:

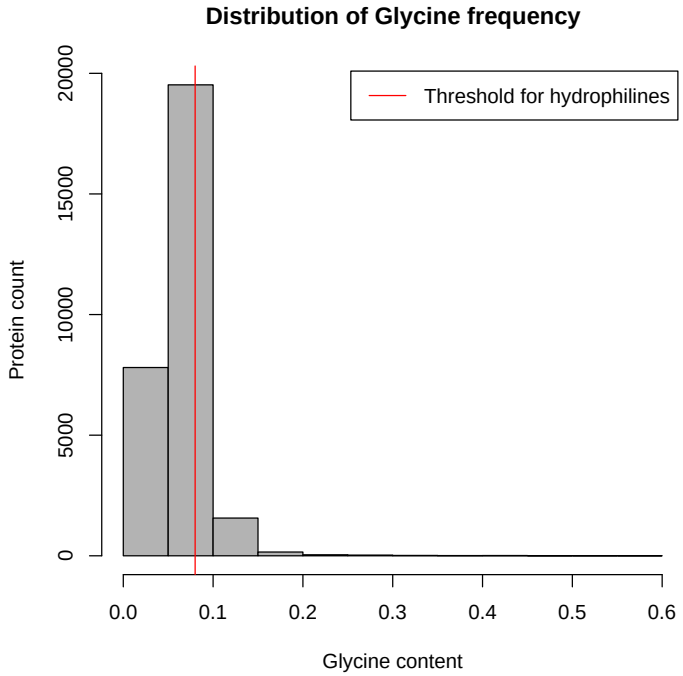
```
ngly(ath[1:10])
```

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

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

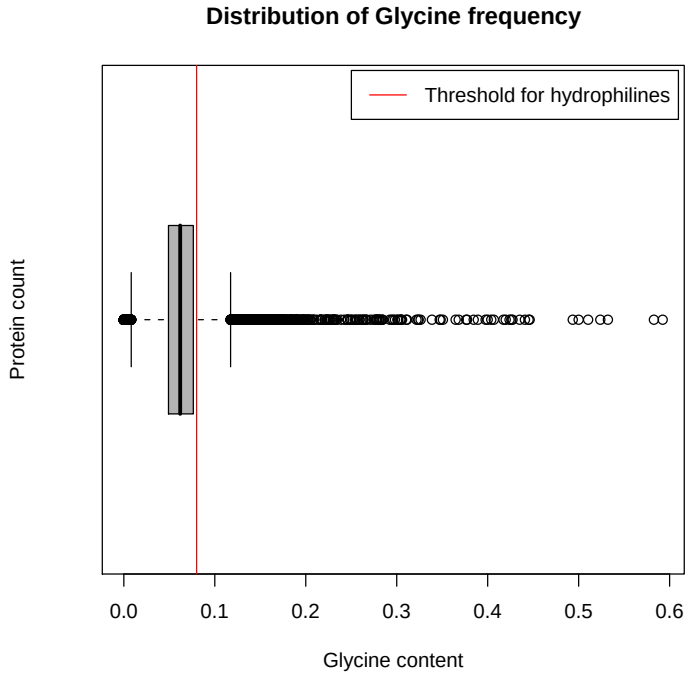
And we can have a look at the distribution:

```
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[40] 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 [27] :

```
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)))
```

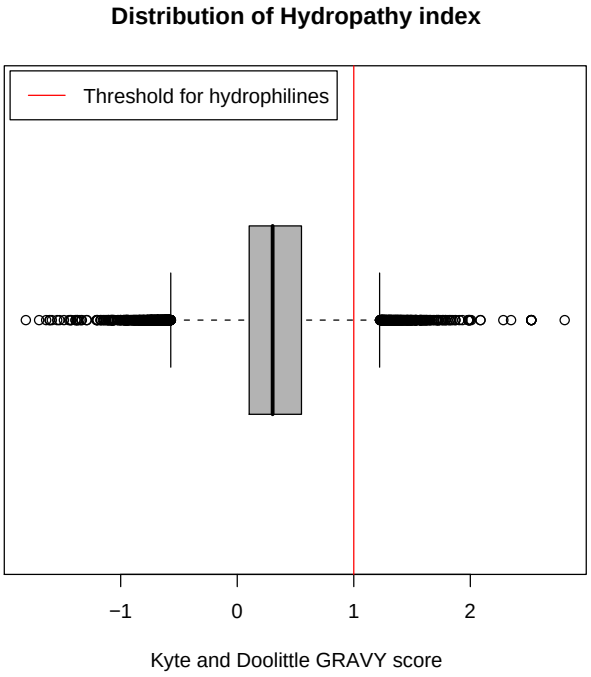
```
* A C D E F G H I K L M N P Q R S T V W Y
1 1 1 0 1 0 2 0 0 3 1 1 0 1 1 3 2 2 0 0 0
```

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:

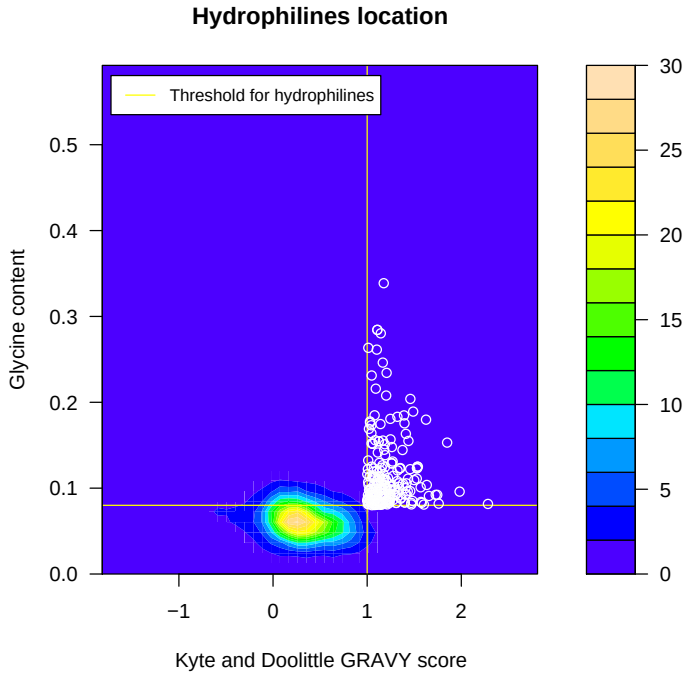
```
hydrophilines <- fgly > 0.08 & kdath > 1
names(ath)[hydrophilines]
```

```
  [1] "At1g02840.1" "At1g02840.2" "At1g02840.3" "At1g03320.1" "At1g03820.1"
  [6] "At1g04450.1" "At1g12810.1" "At1g13020.1" "At1g13290.1" "At1g13890.1"
 [11] "At1g15270.1" "At1g15280.1" "At1g15280.2" "At1g15400.1" "At1g15400.2"
 [16] "At1g20220.1" "At1g21520.1" "At1g23530.1" "At1g23860.1" "At1g23860.2"
 [21] "At1g26210.1" "At1g27650.1" "At1g31750.1" "At1g42430.1" "At1g44478.1"
 [26] "At1g47970.1" "At1g48920.1" "At1g49680.1" "At1g50300.1" "At1g51510.1"
 [31] "At1g52300.1" "At1g52690.1" "At1g52690.2" "At1g52710.1" "At1g53160.1"
 [36] "At1g53160.2" "At1g53260.1" "At1g53645.1" "At1g54410.1" "At1g54926.1"
 [41] "At1g55310.1" "At1g56085.1" "At1g60650.1" "At1g60650.2" "At1g62045.1"
 [46] "At1g63310.1" "At1g64360.1" "At1g64370.1" "At1g67325.1" "At1g67870.1"
 [51] "At1g72050.1" "At1g72080.1" "At1g73350.1" "At1g75560.1" "At1g76010.1"
 [56] "At1g77030.1" "At1g77765.1" "At1g80000.1" "At1g80000.2" "At2g02160.1"
 [61] "At2g03440.1" "At2g04380.1" "At2g04870.1" "At2g07675.1" "At2g11910.1"
 [66] "At2g11910.2" "At2g13125.1" "At2g17950.1" "At2g19750.1" "At2g20430.1"
 [71] "At2g20760.1" "At2g21490.1" "At2g22080.1" "At2g22100.1" "At2g22820.1"
 [76] "At2g23110.1" "At2g24590.1" "At2g25990.1" "At2g27100.1" "At2g32710.1"
 [81] "At2g32710.2" "At2g33460.1" "At2g33690.1" "At2g34310.1" "At2g34310.2"
 [86] "At2g34310.3" "At2g35230.2" "At2g37340.1" "At2g37340.2" "At2g37340.3"
 [91] "At2g39855.2" "At2g40040.1" "At2g40080.1" "At2g40170.1" "At2g41730.1"
 [96] "At2g42190.1" "At2g43370.1" "At2g43970.2" "At2g44710.1" "At2g44820.1"
[101] "At2g44820.2" "At2g46270.2" "At3g02480.1" "At3g05220.1" "At3g05220.2"
[106] "At3g07195.1" "At3g10020.1" "At3g13570.1" "At3g13780.1" "At3g15270.1"
[111] "At3g16040.1" "At3g16080.1" "At3g17160.1" "At3g17190.1" "At3g20910.1"
```

```
[116] "At3g21270.1" "At3g21290.1" "At3g22660.1" "At3g23140.1" "At3g23380.1"
[121] "At3g23390.1" "At3g24100.1" "At3g24740.1" "At3g25590.1" "At3g26400.1"
[126] "At3g26420.1" "At3g29075.1" "At3g29470.1" "At3g29600.1" "At3g47120.1"
[131] "At3g48180.1" "At3g49430.1" "At3g50320.1" "At3g50610.1" "At3g50670.1"
[136] "At3g50690.1" "At3g50970.1" "At3g50980.1" "At3g51810.1" "At3g51940.1"
[141] "At3g53500.1" "At3g53500.2" "At3g55460.1" "At3g56720.1" "At3g57930.1"
[146] "At3g58070.1" "At3g60230.1" "At3g62790.1" "At3g63100.1" "At3g63400.1"
[151] "At3g63400.2" "At4g02425.1" "At4g02430.2" "At4g07940.1" "At4g09840.1"
[156] "At4g09850.1" "At4g09980.1" "At4g12610.1" "At4g12670.1" "At4g13560.1"
[161] "At4g13615.1" "At4g14020.1" "At4g14320.1" "At4g16830.1" "At4g17520.1"
[166] "At4g22740.1" "At4g22740.2" "At4g24500.1" "At4g24500.2" "At4g24680.1"
[171] "At4g25340.1" "At4g25500.1" "At4g25500.2" "At4g26640.1" "At4g26640.2"
[176] "At4g28870.1" "At4g28990.1" "At4g29390.1" "At4g31580.1" "At4g36600.1"
[181] "At4g36730.1" "At4g36730.2" "At4g38710.1" "At5g02020.1" "At5g03990.1"
[186] "At5g04290.1" "At5g11260.1" "At5g11970.1" "At5g12840.1" "At5g12840.2"
[191] "At5g12840.3" "At5g12840.4" "At5g16470.1" "At5g17650.1" "At5g19960.1"
[196] "At5g22040.1" "At5g22650.1" "At5g22650.2" "At5g26890.1" "At5g27690.1"
[201] "At5g28610.1" "At5g28630.1" "At5g44200.1" "At5g47210.1" "At5g50830.1"
[206] "At5g51840.1" "At5g52200.1" "At5g52310.1" "At5g52530.1" "At5g52530.2"
[211] "At5g53820.1" "At5g55430.1" "At5g55640.1" "At5g55670.1" "At5g56670.1"
[216] "At5g58040.1" "At5g58470.1" "At5g58470.2" "At5g59080.1" "At5g59950.1"
[221] "At5g59950.3" "At5g60800.1" "At5g60880.1" "At5g61890.1" "At5g64130.1"
[226] "At5g66400.1" "At5g66780.1" "At5g67190.1" "At5g67490.1" "AtMg00880"
```

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:

```
athres <- data.frame(list(name = names(ath), KD = kdath, Gly = fgly))
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 [27] 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:

```
hannah <- read.table(system.file("sequences/hannah.txt", package = "seqinr"),
    sep = "\t", header = TRUE)
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 [27], 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
```

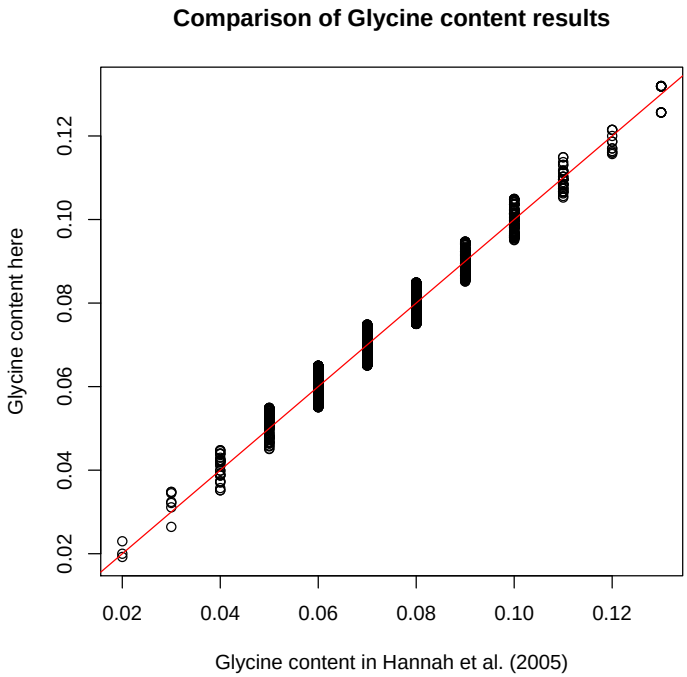
We join now the two data frames thanks to their common key:

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

```
          AGI Hydrophilicity Glycine         KD        Gly
1 At1g01120.1          -0.10    0.06 0.10699433 0.05871212
2 At1g01390.1           0.02    0.06 0.00914761 0.06458333
3 At1g01390.1           0.02    0.06 0.00914761 0.06458333
4 At1g01420.1          -0.05    0.07 0.06203320 0.07276507
5 At1g01420.1          -0.05    0.07 0.06203320 0.07276507
6 At1g01480.1          -0.20    0.07 0.20080483 0.06653226
```

Let's compare the glycine content :

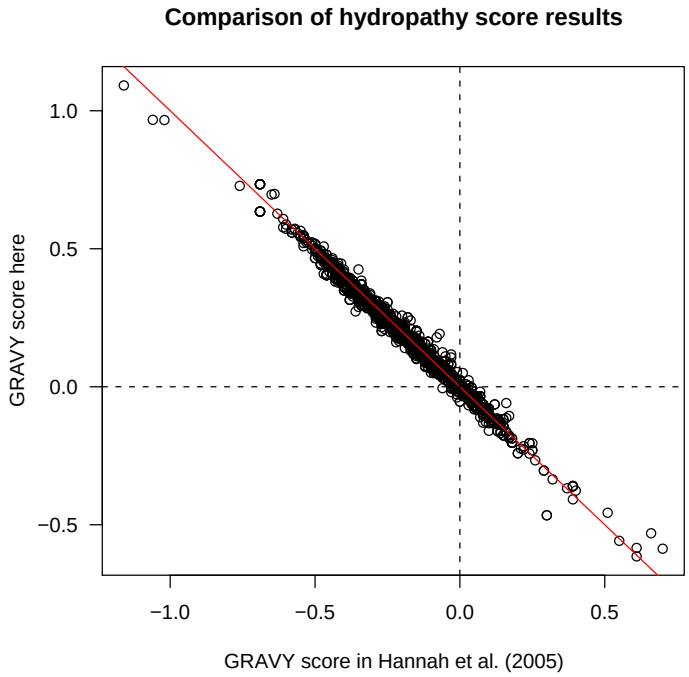
```
plot(join$Glycine, join$Gly, xlab = "Glycine content in Hannah et al. (2005)",
    ylab = "Glycine content here", main = "Comparison of Glycine content results")
abline(c(0, 1), col = "red")
```



The results are consistent, we have just lost some resolution because there are only two figures after the decimal point in the Excel[1] file. Let's have a look at the GRAVY score now:

[1] 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 [27] they used the mean over a sliding window).

2.2 Importing aligned sequence data

2.2.1 Aligned sequences files examples

mase

Mase format is a flatfile format use by the SeaView multiple alignment editor [15], 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.

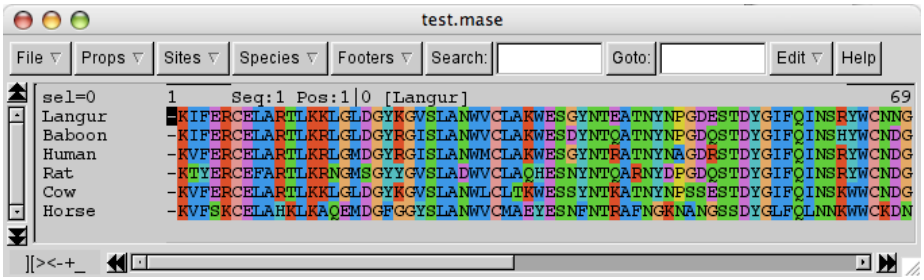


Figure 2.1: The file `test.mase` under SeaView. This is a graphical multiple sequence alignment editor developped by Manolo Gouy [15]. SeaView is able to read and write various alignment formats (NEXUS, MSF, CLUSTAL, FASTA, PHYLIP, MASE). It allows to manually edit the alignment, and also to run DOT-PLOT 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 2.1.

clustal

The CLUSTAL format (*.aln) is the format of the ClustalW multialignment tool output [29, 81]. 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      GGPSTSTTTSGPVSARPARARPRRPREETLTPEEEEKRRVRRERNKLAAAKCRNRRRELT 180
FOSB_HUMAN      GGPSTSGTTSGPGPARPARARPRRPREETLTPEEEEKRRVRRERNKLAAAKCRNRRRELT 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 [13]. 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
```

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

2.2.3 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 [41]), 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" "gi|548119|gb|L32668.1|GYDCYTOXIC"
[3] "gi|548121|gb|L32669.1|GYDCYTOXID" "gi|548125|gb|L32671.1|GYDCYTOXIF"
[5] "gi|548127|gb|L32672.1|GYDCYTOXIG" "gi|548131|gb|L32675.1|GYDCYTOXII"
[7] "gi|548133|gb|L32676.1|GYDCYTOXIJ" "gi|548137|gb|L32678.1|GYDCYTOXIL"
```

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

```
[1] "gi|548223|gb|L32683.1|PPGCYTOXIA" "gi|548197|gb|L32686.1|OGOCYTOXIA"
[3] "gi|548199|gb|L32687.1|OGOCYTOXIB" "gi|548201|gb|L32691.1|OGOCYTOXIC"
[5] "gi|548203|gb|L32692.1|OGOCYTOXID" "gi|548229|gb|L32693.1|PPGCYTOXID"
[7] "gi|548231|gb|L32694.1|PPGCYTOXIE" "gi|548205|gb|L32696.1|OGOCYTOXIE"
```

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 2.2 :

```
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
```

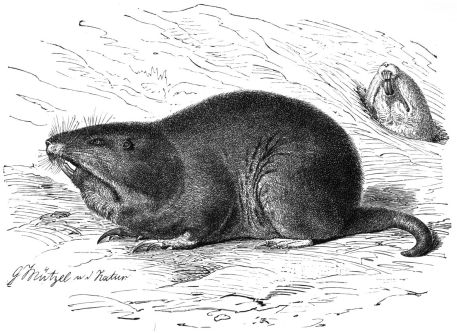


Figure 2.2: Louse (left) and gopher (right). Images are from the wikipedia (http://www.wikipedia.org/). The picture of the chewing louse *Damalinia limbata* found on Angora goats was taken by Fiorella Carnevali (ENEA, Italy). The gopher drawing is from Gustav Mützel, Brehms Tierleben, Small Edition 1927.

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

SeqinR has very few methods devoted to phylogenetic analyses but many are available in the ape package [59]. 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 [69] with Jukes and Cantor's correction [35] for multiple substitutions.

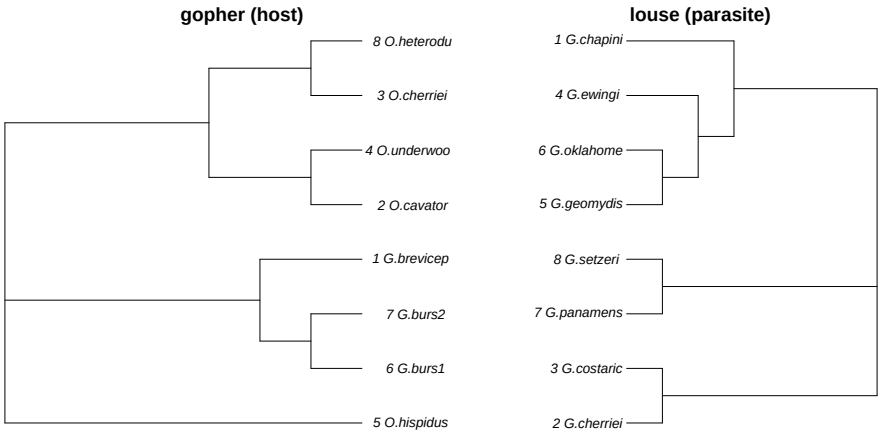
```
library(ape)
x.louse <- lapply(louse$seq, function(x) as.DNAbin(s2c(x)))
x.gopher <- lapply(gopher$seq, function(x) as.DNAbin(s2c(x)))
louse.JC <- dist.dna(x = x.louse, model = "JC69")
gopher.JC <- dist.dna(x = x.gopher, model = "JC69")
l <- nj(louse.JC)
g <- nj(gopher.JC)
```

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 :

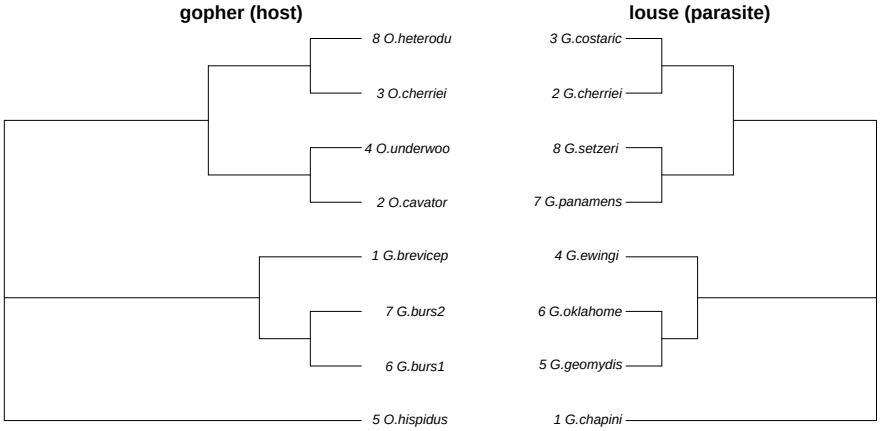
```
louse <- read.alignment(system.file("sequences/louse.fasta", package = "seqinr"), format = "fasta")
gopher <- read.alignment(system.file("sequences/gopher.fasta", package = "seqinr"), format = "fasta")
l.names <- readLines(system.file("sequences/louse.names", package = "seqinr"))
g.names <- readLines(system.file("sequences/gopher.names", package = "seqinr"))
library(ape)
x.louse <- lapply(louse$seq, function(x) as.DNAbin(s2c(x)))
x.gopher <- lapply(gopher$seq, function(x) as.DNAbin(s2c(x)))
louse.JC <- dist.dna(x = x.louse, model = "JC69")
gopher.JC <- dist.dna(x = x.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 [35] and wants you to use the Kimura's 2-parameters distance

[39] 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.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 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15, tools 2.6.0

There were two compilation steps:

- R compilation time was: Thu Nov 1 10:42:10 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 3

Importing sequences from ACNUC databases

Charif, D. Lobry, J.R.

Contents

3.1 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 [18, 19, 25, 23, 24].

3.1.1 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"      "refseq"       "hobacnucl"    "hobacprot"
 [9] "hovernucl"    "hoverprot"    "hogennucl"    "hogenprot"
[13] "hogen4nucl"   "hogen4prot"   "homolensprot" "homolensnucl"
[17] "greview"      "HAMAPnucl"    "HAMAPprot"    "hoppsigen"
[21] "nurebnucl"    "nurebprot"    "taxobacgen"
```

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 (December 2, 2007) of this document:

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

```
    bank status
1 genbank     on
2    embl     on
3 emblwgs     on
                                                                    info
1       GenBank Rel. 162 (15 October 2007) Last Updated: Oct 30, 2007
2 EMBL Library Release 92 (September 2007) Last Updated: Oct 18, 2007
3    EMBL Whole Genome Shotgun sequences Release 92  (September 2007)
```

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      emblTP     on                        frozen EMBL release
2 swissprotTP     on                    frozen SwissProt release
3 hoverprotTP     on    frozen Hovergen release - protein sequences
4 hovernuclTP     on frozen Hovergen release - nucleotide sequences
5     trypano     on                     frozen trypano database
```

Now, if you want to work with a given database, say GenBank, just call `choosebank()` with `"genbank"` as its first argument, the result is saved in the variable `banknameSocket` in the workspace:

```
choosebank("genbank")
str(banknameSocket)
```

```
List of 9
 $ socket  :Classes 'sockconn', 'connection'  atomic [1:1] 5
  .. ..- attr(*, "conn_id")=<externalptr>
 $ bankname: chr "genbank"
 $ banktype: chr "GENBANK"
 $ totseqs : num 82398592
 $ totspecs: num 496373
 $ totkeys : num 7273360
 $ release : chr "         GenBank Rel. 162 (15 October 2007) Last Updated: Oct 30, 2007"
 $ status  :Class 'AsIs'  chr "on"
 $ details : chr [1:4] "             ****     ACNUC Data Base Content      ****                          " "         G
```

```
closebank()
```

The components of `banknameSocket` means that in the database called `genbank` at the compilation time of this document there were `82,398,592` sequences from `496,373` species and a total of `7,273,360` keywords. The status of the bank was `on`, and the release information was `GenBank Rel. 162 (15 October 2007) Last Updated: Oct 30, 2007`. For specialized databases, some relevant informations are also given in the `details` component, for instance:

```
choosebank("taxobacgen")
cat(banknameSocket$details, sep = "\n")
```

```
               ****     ACNUC Data Base Content      ****
                 TaxoBacGen Rel. 7 (September 2005)
1,151,149,763 bases; 254,335 sequences; 847,767 subseqs; 63,879 refers.
        Data compiled from GenBank by Gregory Devulder
        Laboratoire de Biometrie & Biologie Evolutive, Univ Lyon I
------------------------------
This database is a taxonomic genomic database.
It results from an expertise crossing the data nomenclature database DSMZ
[http://www.dsmz.de/species/bacteria.htm Deutsche Sammlung von
Mikroorganismen und Zellkulturen GmbH, Braunschweig, Germany]
and GenBank.
- Only contains sequences described under species present in
Bacterial Nomenclature Up-to-date.
- Names of species and genus validly published according to the
Bacteriological Code (names with standing in nomenclature) is
added in field "DEFINITION".
- A keyword "type strain" is added in field "FEATURES/source/strain" in
GenBank format definition to easyly identify Type Strain.
Taxobacgen is a genomic database designed for studies based on a strict
respect of up-to-date nomenclature and taxonomy.
```

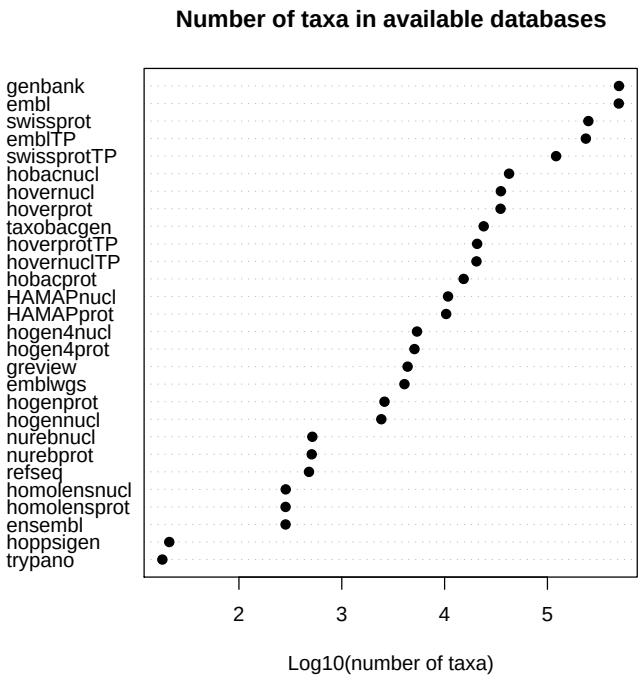
```
closebank()
```

As from **seqinR** 1.0-3, the result of the `choosebank()` function is automatically stored in a global variable named `banknameSocket`, 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].

[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.

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)
for (i in seq_len(nbanks)) {
    bkopenres <- try(choosebank(banks[i]))
    if (inherits(bkopenres, "try-error")) {
        ntaxa[i] <- NA
    }
    else {
        ntaxa[i] <- as.numeric(banknameSocket$totspecs)
        closebank()
    }
}
names(ntaxa) <- banks


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



3.1.2 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:

```
choosebank("genbank")
```

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. 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`.


Felis catus. Source: wikipedia

```
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:

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

```
[1] "AB000483.PE1"    "AB000484.PE1"    "AB000485.PE1"    "AB004237"
[5] "AB004238"        "AB009279.PE1"    "AB009280.PE1"    "AB010872.UGT1A1"
[9] "AB011965.SDF-1A" "AB011966.SDF-1B"
```

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 3.1.

	Type	Description
1	CDS	.PE protein coding region
2	LOCUS	sequenced DNA fragment
3	MISC_RNA	.RN other structural RNA coding region
4	RRNA	.RR mature ribosomal RNA
5	SCRNA	.SC small cytoplasmic RNA
6	SNRNA	.SN small nuclear RNA
7	TRNA	.TR mature transfer RNA

Table 3.1: Available subsequences in genbank

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:

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

```
[1] 58
```

Hence, there were 58 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 :

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

```
[1] 4857947
```

There are therefore `4,857,947` coding sequences in this version of GenBank[2]. 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*:

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

```
[1] 931
```

There are then `931` elements in the list `small`, so that we can safely repeat the previous query without asking for a virtual list:

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

```
 [1] "AY191424" "AY386807" "AY386808" "AY386809" "AY386810" "AY386811"
 [7] "AY386812" "AY386813" "AY386814" "AY386815"
```

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

[2] which is stored in the `release` component of the object `banknameSocket` and current value is today (December 2, 2007): `banknameSocket$release = GenBank Rel. 162 (15 October 2007) Last Updated: Oct 30, 2007`.

[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.

Man. How many sequences are available for our species?

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

```
[1] 11260600
```

There are 11,260,600 sequences from *Homo sapiens*.

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

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

```
[1] 1196
```

There are 1,196 such sequences.

tRNA. How many complete tRNA sequences are available?

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

```
[1] 254050
```

There are 254,050 complete tRNA sequences.

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

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

```
[1] 1533556
```

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

```
[1] 1284219
```

There are 1,533,556 sequences published in *Nature* and 1,284,219 sequences published in *Science*, so that the winner is *Nature*.

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

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

```
[1] 4107145
```

There are 4,107,145 such sequences.

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

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

```
[1] 66366478
```

There are 66,366,478 sequences published after year 2000.

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

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

```
[1] 151023
```

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

```
[1] 516327
```

There are `151,023` sequences from chloroplast genomes and `516,327` sequences from mitochondrion genomes, so that the winner is mitochondrion.

```
closebank()
```

3.1.3 Extract sequences of interest

Introduction

There are two functions to get the sequences. The first one, `getSequence()`, uses regular socket connections, the second one, `extractseq()`, uses zlib compressed sockets, which is faster but the function is experimental and has not been extensively tested.

Extacting sequences with `getSequence()`

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* :

```
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:

```
myseq <- 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] "atgaccaacattcgaaaatcacacccccttaccaaaattattaatcactc"
```

Note that what is done by `getSequence()` is much more complex than a substring extraction because subsequences of biological interest are not necessarily contiguous or even on the same DNA strand. Consider for instance the following coding sequence from sequence `AE003734`:

```
query("trs", "N=AE003734.PE35")
annots <- getAnnot(trs$req[[1]])
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:

```
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 transsplicing 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]
```

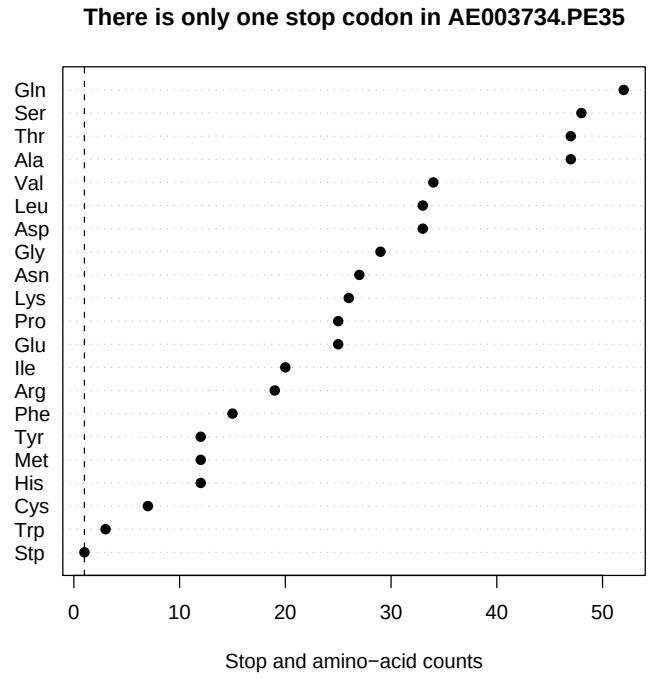
```
 [1] "M" "A" "D" "D" "E" "Q" "F" "S" "L" "C" "W" "N" "N" "F" "N" "T" "N" "L"
[19] "S" "A" "G" "F" "H" "E" "S" "L" "C" "R" "G" "D" "L" "V" "D" "V" "S" "L"
[37] "A" "A" "E" "G" "Q" "I" "V" "K" "A" "H" "R" "L" "V" "L"
```

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

```
*  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]])
```

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

```
[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.

```
closebank()
```

Extacting sequences with `extractseqs()` (experimental)

The timings here were from an home-ADSL connection, and are only indicative. 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.

```
rm(list = ls(all = TRUE))
gc()
```

```
         used (Mb) gc trigger  (Mb) max used  (Mb)
Ncells 244254  6.6    4195948 112.1  5860551 156.5
Vcells 570263  4.4   29477531 224.9 37726935 287.9
```

```
(tcb <- system.time(choosebank("emblTP")))
```

```
utilisateur    syst`eme    'ecoul'e
      0.059       0.001       5.392
```

```
(tqu <- system.time(query("hsCDS", "sp=Homo sapiens AND t=cds AND o=nuclear AND NOT k=partial",
    virtual = TRUE)))
```

```
utilisateur    syst`eme    'ecoul'e
      0.001       0.000      11.256
```

```
(nseq <- hsCDS$nelem)
```

```
[1] 78573
```

```
showConnections()
```

```
  description                class    mode text   isopen   can read
3 "getseqacnuc.tex"          "file"   "w+" "text" "opened" "yes"
4 ""                         "file"   "w+" "text" "opened" "yes"
5 "->pbil.univ-lyon1.fr:5558" "socket" "a+" "text" "opened" "yes"
  can write
3 "yes"
4 "yes"
5 "yes"
```

```
Sys.sleep(1)
(tex <- system.time(mycds <- extractseqs("hsCDS", verbose = TRUE)))
```

```
I'm checking the arguments...
... and everything is OK up to now.
Format is  fasta
Operation is  simple
The rank of the list  hsCDS is  2 .
request :  extractseqs&lrank=2&format=fasta&operation=simple&zlib=T
Running getzlibsock...
'con' is a connection...
Socket number is 5....
Trying to get answer from socket...

-->[code=0]
Socket answer is ok code=0(6)
n=1000, nn=1000,nnn=1000
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
Increasing memory...
extractseqs successfully ended ...
Number of lines     : 1799153
Number of sequences : 78573
extractseqs OK, program carry on...
Ok, everything is fine!
Closing socket close_sock_gz_r  status = 0
utilisateur    syst`eme     'ecoul'e
      9.024       1.121      60.955
```

It was then about 5 seconds to select the relevant database. We suppose that the sequences we are interested in are all the complete coding sequences from *Homo sapiens* that are encoded in the nucleus (we don't want sequences from human mitochondrion). We have used a virtual query to speed up things: it was about 11 seconds to create on the server a list of 78573 sequences. We have downloaded the sequences in zlib compressed mode: it was about 61 seconds to dowload the sequences in the object `mycds`, which looks like :

```
cat(head(mycds), sep = "\n")
```

```
>A00127.PE1             2217 residues
ATGCGGGGTCCGAGCGGGGCTCTGTGGCTGCTCCTGGCTCTGCGCACCGTGCTCGGAGGC
ATGGAGGTGCGGTGGTGCGCCACCTCGGACCCAGAGCAGCACAAGTGCGGCAACATGAGC
GAGGCCTTCCGGGAAGCGGGCATCCAGCCCTCCCTCCTCTGCGTCCGGGGCACCTCCGCC
GACCACTGCGTCCAGCTCATCGCGGCCCAGGAGGCTGACGCCATCACTCTGGATGGAGGA
GCCATCTATGAGGCGGGAAAGGAGCACGGCCTGAAGCCGGTGGTGGGCGAAGTGTACGAT
```

```
cat(tail(mycds), sep = "\n")
```

```
ATCACTGCGGCCCCAGAGAGAGAGGGCATAGGCCACGGCGGCCCCAAGCTATGCTGCACA
CTGAGCTCCCTCAGCTCCGCTGCTGAGACTGGCCGGGACCCGCTGGACAGCGAGGAGGAG
GCAACCAGCGGCGCCCAGGATGAACGTGGCCTGAAGCCGCCTTCCCGGGGCCAGTTTCCT
TCCCTCTCAGCCAGGGATGCCTCGAGCAGCCACAGGGGCAGGAACGTCCTGACTGCCATC
CTGCTGCTGCTGCGGGAGCTGGATGCAGAGGGGCTGGAGGCCGTGCAGCAGACTGTGGGC
AGCCGGCTGCAGGCCCTGCGTGGGGAAGAGGTGCAGGAGCACGCCGAGTGA
```

We save now the sequences in a local FASTA file for future use:

```
(twl <- system.time(writeLines(mycds, "mycds.fasta")))
```

```
utilisateur    syst`eme     'ecoul'e
      0.875       0.791       2.764
```

It was then about 3 seconds to dump the sequences on a local file. We read the sequences as strings without setting attributes to save time:

```
(trf <- system.time(mycdss <- read.fasta("mycds.fasta", as.string = TRUE,
    set.attributes = FALSE)))
```

```
utilisateur    syst`eme    'ecoul'e
     34.280       0.308      34.792
```

It was then about 35 seconds to read the sequences as strings. We save them in XDR format:

```
(tsrd <- system.time(save(mycdss, file = "mycdss.RData")))
```

```
utilisateur    syst`eme    'ecoul'e
     38.849       0.338      39.813
```

It was then about 40 seconds to save the sequences in XDR format. How long is it to load the sequences from XDR format?

```
(tlrd <- system.time(load("mycdss.RData")))
```

```
utilisateur    syst`eme    'ecoul'e
      1.764       0.041       1.813
```

It was then about 2 seconds to load the sequences from an XDR formated file. Now, we also want the corresponding proteins. We download the translated CDS from the server:

```
(texp <- system.time(myprot <- extractseqs("hsCDS", operation = "translate")))
```

```
utilisateur    syst`eme    'ecoul'e
      3.680       0.647      60.748
```

It was then about 61 seconds to get the protein sequences from the server. The object `myprot` looks like:

```
cat(head(myprot), sep = "\n")
```

```
>A00127.PE1              739 residues
MRGPSGALWLLLALRTVLGGMEVRWCATSDPEQHKCGNMSEAFREAGIQPSLLCVRGTSA
DHCVQLIAAQEADAITLDGGAIYEAGKEHGLKPVVGEVYDQEVGTSYYAVAVVRRSSHVT
IDTLKGVKSCHTGINRTVGWNVPVGYLVESGRLSVMGCDVLKAVSDYFGGSCVPGAGETS
YSESLCRLCRGDSSGEGVCDKSPLERYYDYSGAFRCLAEGAGDVAFVKHSTVLENTDGKT
LPSWGQALLSQDFELLCRDGSRADVTEWRQCHLARVPAHAVVVRADTDGGLIFRLLNEGQ
```

```
cat(tail(myprot), sep = "\n")
```

```
>Z93322.PE1              257 residues
MKLTRKMVLTRAKASELHSVRKLNCWGSRLTDISICQEMPSLEVITLSVNSISTLEPVSR
CQRLSELYLRRNRIPSLAELFYLKGLPRLRVLWLAENPCCGTSPHRYRMTVLRTLPRLQK
LDNQAVTEEELSRALSEGEEITAAPEREGIGHGGPKLCCTLSSLSSAAETGRDPLDSEEE
ATSGAQDERGLKPPSRGQFPSLSARDASSSHRGRNVLTAILLLLRELDAEGLEAVQQTVG
SRLQALRGEEVQEHAE*
```

We save the protein sequences in a local FASTA file for future use:

```
(twl2 <- system.time(writeLines(myprot, "myprot.fasta")))
```

```
utilisateur    syst`eme    'ecoul'e
      0.330       0.274       0.605
```

It was then about 1 seconds to dump the protein sequences on a local file. We read the sequences as strings without setting attributes to save time:

```
(trf2 <- system.time(myprots <- read.fasta("myprot.fasta",
    as.string = TRUE, set.attributes = FALSE)))
```

```
utilisateur    syst`eme    'ecoul'e
     22.588       0.148      22.859
```

It was then about 23 seconds to read the protein sequences as strings. We save them in XDR format:

```
(tsrd2 <- system.time(save(myprots, file = "myprots.RData")))
```

```
utilisateur    syst`eme    'ecoul'e
      3.766       0.154       4.282
```

It was then about 4 seconds to save the protein sequences in XDR format. How long is it to load the protein sequences from XDR format?

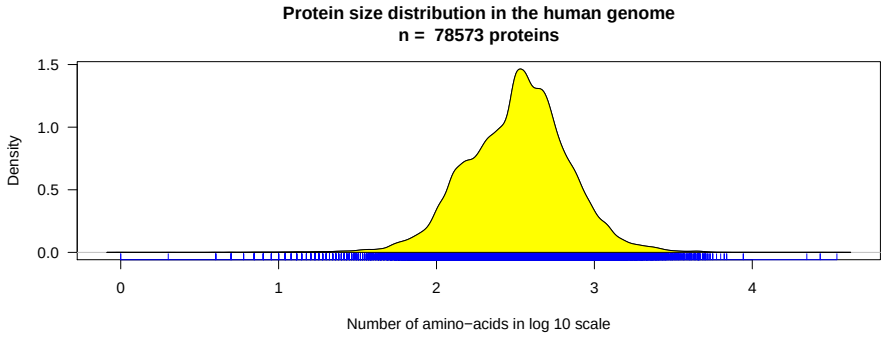
```
(tlrd2 <- system.time(load("myprots.RData")))
```

```
utilisateur    syst`eme    'ecoul'e
      1.092       0.027       1.121
```

It was then about 1 seconds to load the protein sequences from an XDR formated file.

As a quick sanity check, we plot the distribution of protein size:

```
x <- log10(nchar(myprots) - 1)
dstx <- density(x)
plot(dstx, main = paste("Protein size distribution in the human genome\nn = ",
    length(myprots), "proteins"), xlab = "Number of amino-acids in log 10 scale",
    las = 1)
rug(x, col = "blue")
polycurve <- function(x, y, base.y = min(y), ...) polygon(x = c(min(x),
    x, max(x)), y = c(base.y, y, base.y), ...)
polycurve(dstx$x, dstx$y, col = "yellow")
```




```
closebank()
```

3.2 The query language

3.2.1 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.

3.2.2 Case sensitivity and ambiguities resolution

The query language is case insensitive, for instance:

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

```
[1] 18
```

```
closebank()
```

3.2.3 Selection criteria

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.

`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")
query("bb", "sp=Borrelia burgdorferi", virtual = T)
bb$nelem
```

```
[1] 1682
```

```
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")
query("sapiens", "sp=@sapiens@", virtual = T)
sapiens$nelem
```

```
[1] 2216556
```

```
query("sapienspecies", "PS sapiens")
sapply(sapienspecies$req, getName)
```

```
 [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

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")
query("hsn", "TID=63221", virtual = T)
hsn$nelem
```

```
[1] 1337
```

```
query("hsnsp", "PS hsn")
sapply(hsnsp$req, getName)
```

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

```
closebank()
```

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

```
[1] 105
```

```
closebank()
```

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

```
[1] 150513
```

```
closebank()
```

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")
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()
```

`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")
query("rpro", "R=Nature/396/133")
sapply(rpro$req, getName)
```

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

```
closebank()
```

`AU=name`

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

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

```
[1] 48
```

```
closebank()
```

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")
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()
```

N=seq_name

This is used to select sequences of a given name[5]. 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.

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

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

`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")
query("Yexample", "Y=1999", virtual = TRUE)
Yexample$nelem
```

```
[1] 955274
```

```
closebank()
```

`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
3657   CHLOROPLAST       Chloroplast genome
3658 MITOCHONDRION     Mitochondrial genome
3659   NUCLEOMORPH       Nucleomorph genome
3660       PLASTID non-green plastid genome
```

```
closebank()
```

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

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

```
[1] 65011
```

```
closebank()
```

`M=molecule`

This is used to select sequences according to the chemical nature of the sequenced molecule[6]. 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                                      <NA>
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   SCRNA sequenced molecule is small cytoplasmic RNA
10 SNORNA   sequenced molecule is small nucleolar RNA
11  SNRNA     sequenced molecule is small nuclear RNA
12   TRNA                  sequenced molecule is tRNA
```

```
closebank()
```

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

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

```
[1] 7421752
```

```
closebank()
```

[6] as named in ID or LOCUS annotation records

`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
```

```
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")
query("STexample", "ST=REVIEWED", virtual = TRUE)
STexample$nelem
```

```
[1] 285335
```

```
closebank()
```

`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
```

```
clfcd("listSQ", file = fileSQ, type = "SQ")
sapply(listSQ$req, getName)
```

```
 [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()
```

`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
```

```
clfcd("listAC", file = fileAC, type = "AC")
sapply(listAC$req, getName)
```

```
 [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()
```

`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
```

```
clfcd("listKW", file = fileKW, type = "KW")
sapply(listKW$req, getName)
```

```
 [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()
```

`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
```

```
clfcd("listSP", file = fileSP, type = "SP")
sapply(listSP$req, getName)
```

```
 [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()
```

list_name

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

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

```
[1] 885225
```

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

```
[1] 1682
```

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

```
[1] 131
```

```
closebank()
```

3.2.4 Operators

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

```
[1] 3218
```

```
closebank()
```

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

```
[1] 28584
```

```
closebank()
```

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

```
[1] 3266
```

```
closebank()
```

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

```
[1] 0
```

```
closebank()
```

SUB

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

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

```
[1] 70
```

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

```
[1] 943
```

```
closebank()
```

PS

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

```
choosebank("emblTP")
query("PSexample", "K=hyperthermo@", virtual = T)
query("PSexample2", "PS PSexample")
sapply(PSexample2$req, getName)
```

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

```
closebank()
```

PK

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

```
choosebank("emblTP")
query("PKexample", "AC=AE000783", virtual = T)
query("PKexample2", "PK PKexample")
sapply(PKexample2$req, getName)
```

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

```
closebank()
```

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
```

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

```
[1] 2786
```

```
closebank()
```

SD

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

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

```
[1] 19
```

```
query("SDexample", "SD hsp")
sapply(SDexample$req, getName)
```

```
 [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()
```

KD

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

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

```
[1] 540
```

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

```
[1] 572
```

```
closebank()
```

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Tue Oct 30 17:41:35 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 4

How to deal with sequences

Charif, D. Lobry, J.R.

Contents

4.1 Sequence classes

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

- **seqFasta** is the class for the 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

4.2 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:

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

4.3 Internal representation of sequences

The current mode of sequence storage is done with vectors of characters instead of strings. 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.

4.3.1 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:

```
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
```

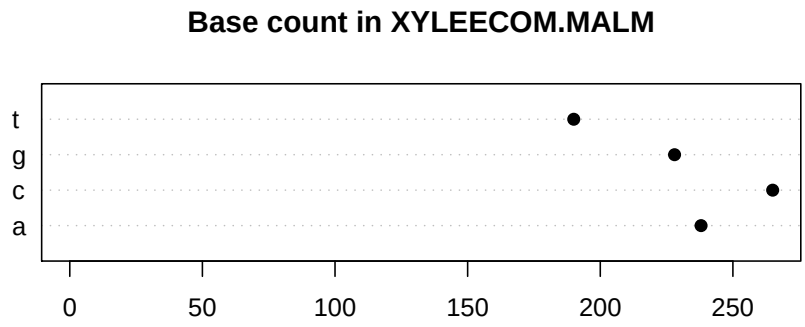
This recycling rule is extremely convenient in practice but may have surprising effects if you assume (incorrectly) that there is a stringent dimension control for R vectors as in linear algebra.

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:

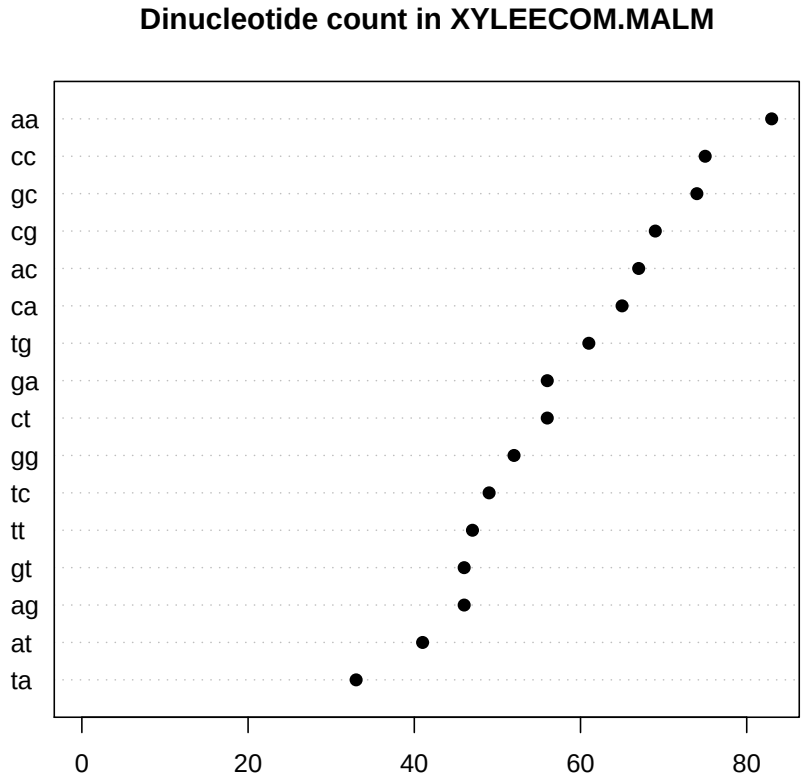
```
tota <- sum(myseq == "a")
```

The total number of `a` in our sequence is 238. Let's compare graphically the different base counts in our sequence :

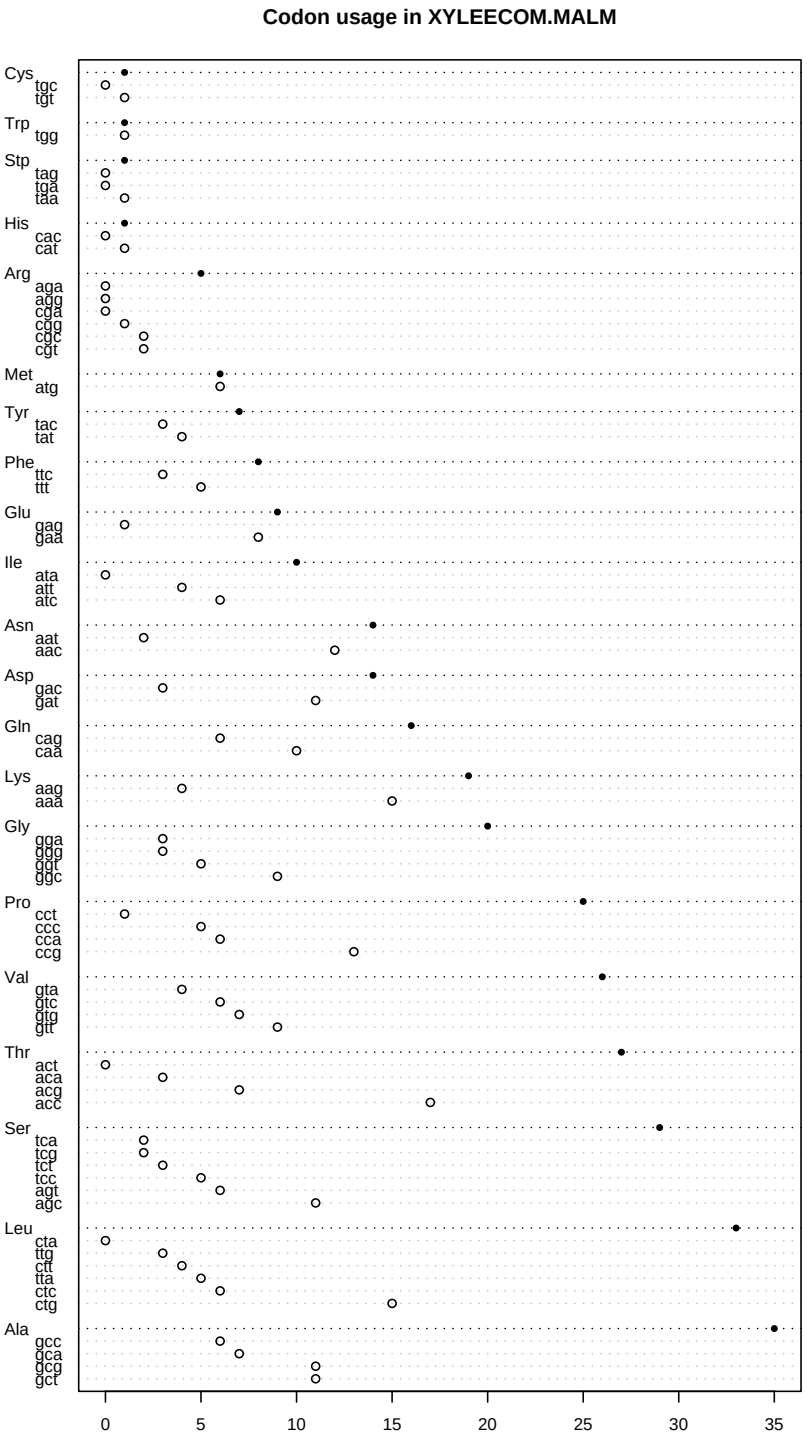
```
basecount <- table(myseq)
myseqname <- getName(myseq)
dotchart(basecount, xlim = c(0, max(basecount)), pch = 19,
    main = paste("Base count in", myseqname))
```




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




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


Codon usage in XYLEECOM.MALM
Cys
tgc
tgt
Trp
tgg
Stp
tag
tga
taa
His
cac
cat
Arg
aga
agg
cga
cgg
cgc
cgt
Met
atg
Tyr
tac
tat
Phe
ttc
ttt
Glu
gag
gaa
Ile
ata
att
atc
Asn
aat
aac
Asp
gac
gat
Gln
cag
caa
Lys
aag
aaa
Gly
gga
ggg
ggt
ggc
Pro
cct
ccc
cca
ccg
Val
gta
gtc
gtg
gtt
Thr
act
aca
acg
acc
Ser
tca
tcg
tct
tcc
agt
agc
Leu
cta
ttg
ctt
tta
ctc
ctg
Ala
gcc
gca
gcg
gct
0
5
10
15
20
25
30
35

4.3.2 Sequences as strings

If you are interested in (fuzzy) pattern matching, then it is advisable to work with sequence as strings to take advantage of *regular expression* implemented in R. The function `words.pos()` returns the positions of all occurrences of a given regular expression. Let's suppose we want to know where are the trinucleotides "cgt" in a sequence, that is the fragment CpGpT in the direct strand:

```
mystring <- c2s(myseq)
words.pos("cgt", mystring)
```

```
[1]  24  90 216 245 252 315 330 405 432 452 552 592 648 836 883
```

We can also look for the fragment CpGpTpY to illustrate fuzzy matching because Y (IUPAC code for pyrimidine) stands C or T:

```
words.pos("cgt[ct]", mystring)
```

```
[1]  24 216 252 315 432 452 552 592 836 883
```

To look for all CpC dinucleotides separated by 3 or 4 bases:

```
words.pos("cc.{3,4}cc", mystring)
```

```
numeric(0)
```

Virtually any pattern is easily encoded with a regular expression. This is especially useful at the protein level because many functions can be attributed to short linear motifs.

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Wed Nov 7 11:58:18 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 5

Installation of a local ACNUC socket server and of a local ACNUC database on your machine.

Penel, S.

Contents

5.1 Introduction

This chapter is under development.

5.2 System requirement

Basically if you are installing R from the sources, you should be able to build a ACNUC socket server. The socket server will build under a number of common

Unix and Unix-alike platforms. You will need several tools: programs are written in C thus you will need a means of compiling C (as gcc compilation tools for linux or unix, Apple Developer Tools for MacOSX). You need as well library zlib and sockets (standards on linux and unix).

5.3 Setting a local ACNUC database to be queried by the server

First of all yo need an ACNUC database, built by yourself or downloaded from the PBIL ftp server. An ACNUC database is composed of two sets of files:

1. the acnuc index files.
2. the database files (*i.e.* flat files in EMBL/GenBank or SwissProt format).

These two sets will be located in the `index` and `flat_files` directories respectively.

An example of an ACNUC database is available on the PBIL ftp server at this url: `ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/acnucdatabase.tar.Z`.

You may install the database as it follows: Let ACNUC_HOME be the base directory for ACNUC intallation.

```
dir.create("./ACNUC_HOME", showWarning = FALSE)
```

Let ACNUC_HOME/ACNUC_DB be the directory where you want to install the databases and ACNUC_HOME/ACNUC_DB/demoacnuc the directory where you want to install the demo database.

```
dir.create("./ACNUC_HOME/ACNUC_DB", showWarning = FALSE)
dir.create("./ACNUC_HOME/ACNUC_DB/demoacnuc", showWarning = FALSE)
```

- Dowload the ACNUC database in the ./ACNUC_HOME/ACNUC_DB/demoacnuc/ directory.

```
download.file("ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/acnucdatabase.tar.Z",
    destfile = "./ACNUC_HOME/ACNUC_DB/demoacnuc/acnucdatabase.tar.Z")
```

- Uncompress and untar the `acnucdatabase.tar.Z` file

```
pwd <- getwd()
setwd("./ACNUC_HOME/ACNUC_DB/demoacnuc/")
system("gunzip -f acnucdatabase.tar.Z")
system("tar -xvf acnucdatabase.tar")
system("rm -f acnucdatabase.tar")
setwd(pwd)
```

Now you sould get the following directories:

`ACNUC_HOME/ACNUC_DB/demoacnuc/index`
`ACNUC_HOME/ACNUC_DB/demoacnuc/flat_files`

The directory ACNUC<_HOME/ACNUC_DB/demoacnuc contains:

```
dir("./ACNUC_HOME/ACNUC_DB/demoacnuc")
```

```
[1] "flat_files" "index"
```

These directories contains respectively:

```
dir("./ACNUC_HOME/ACNUC_DB/demoacnuc/index")
```

```
 [1] "ACCESS"                  "AUTHOR"
 [3] "BIBLIO"                  "EXTRACT"
 [5] "HELP"                    "HELP_WIN"
 [7] "KEYWORDS"                "LOCUS"
 [9] "LONGL"                   "MERES"
[11] "SHORTL"                  "SMJYT"
[13] "SPECIES"                 "SUBSEQ"
[15] "TAXIDS"                  "TEXT"
[17] "custom_qualifier_policy"
```

```
dir("./ACNUC_HOME/ACNUC_DB/demoacnuc/flat_files")
```

```
[1] "escherichia.dat" "id.log"          "yeast.dat"
```

This database contains the complete genome of *Escherichia coli K12 W3110* and *Saccharomyces cerevesiae*.

5.4 Build the ACNUC sockets server from the sources.

Once you have a local ACNUC database available on your server you need to install the sockets server.

5.4.1 Download the sources.

The code source of the racnucd server is available on the PBIL server at this url:

`http://pbil.univ-lyon1.fr/databases/acnuc/racnucd.html`

Alternatively you can download directly the source from the ftp at:

`ftp://pbil.univ-lyon1.fr/pub/acnuc/unix/racnucd.tar`

5.4.2 Build the ACNUC sockets server.

You may install the racnucd server as it follows: let ACNUC_HOME/ACNUC_SOFT/ be the base directory for the ACNUC softs.

```
dir.create("./ACNUC_HOME/ACNUC_SOFT", showWarning = FALSE)
```

- Dowload the `racnucd.tar` file into ACNUC_HOME/ACNUC_SOFT.

```
download.file("ftp://pbil.univ-lyon1.fr/pub/acnuc/unix/racnucd.tar",
    destfile = "./ACNUC_HOME/ACNUC_SOFT/racnucd.tar")
```

- Untar the `racnucd.tar` file

```
setwd("./ACNUC_HOME/ACNUC_SOFT/")
system("tar -xvf racnucd.tar")
system("rm -f racnucd.tar")
setwd(pwd)
```

Now you sould get the following directory:

```
dir("./ACNUC_HOME/ACNUC_SOFT/")
```

```
[1] "racnucd"
```

```
dir("./ACNUC_HOME/ACNUC_SOFT/racnucd/")
```

```
 [1] "bit.c"                "dbplaces"             "dir_acnuc.h"
 [4] "dir_io.c"             "dir_io.h"             "execute.c"
 [7] "execute.h"            "extract.c"            "knowndbs"
[10] "lngbit.c"             "makefile"             "md5.c"
[13] "misc_acnuc.c"         "ordre.h"              "parser.c"
[16] "prep_acnuc_requete.c" "pretty_seq.c"         "proc_requete.c"
[19] "racnucd.ini"          "requete_acnuc.h"      "serveur.c"
[22] "serveur.h"            "simext.h"             "use_acnuc.c"
[25] "utilquery.c"          "zsockw.c"
```

Go into ACNUC_HOME/ACNUC_SOFT/racnucd/ and type;

```
make
```

This should create the **racnucd** executable.

```
setwd("./ACNUC_HOME/ACNUC_SOFT/racnucd")
system("make")
dir(pattern = "racnucd")
```

```
[1] "racnucd"     "racnucd.ini"
```

```
setwd(pwd)
```

5.4.3 Setting the ACNUC sockets server.

The server is configured by several parameters described in a configuration file **racnuc.ini**. The **racnucd.ini** file is structued as follows:

```
cat(readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/racnucd.ini"),
    sep = "\n")
```

```
port=5558
maxtime=8000
known_db_file=knowndbs
db_env_names=dbplaces
```

- **port** is the port of the socket server
- **maxtimle** is the time delay of the connection
- **knowndbs** is a file containing the list of available databases
- **dbplaces** is a file containing the path of the available databases

You may want to change the port of the socket server, according to the availabilities and restricttions on your machine. For example , lets use the port 49152 in a new racnucd.new file.

```
initline <- readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/racnucd.ini")
initline[1] = "port=49152"
writeLines(initline, "./ACNUC_HOME/ACNUC_SOFT/racnucd/racnucd.new")
cat(readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/racnucd.new"),
    sep = "\n")
```

```
port=49152
maxtime=8000
known_db_file=knowndbs
db_env_names=dbplaces
```

Configuring the knowndbs file.

The **knowndbs** contains:

```
cat(readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/knowndbs"),
    sep = "\n")
```

```
embl | on |    | EMBL sequence data library |
swissprot   | on |  | UniProt |
```

Each line defines a database, the four fields indicating respectively the name of the database, its status (*on* or *off*), a tag and a short description.

You should set the files **knowndbs** according to your installation. Let's call the database you installed previously *demoacnuc*. Modify the **knowndbs** as follows:

```
demoacnuc | on |    | Demo Database |
```

```
writeLines("demoacnuc | on |    | Demo Database | ", "./ACNUC_HOME/ACNUC_SOFT/racnucd/knowndbs")
knowndbs <- readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/knowndbs")
cat(knowndbs, sep = "\n")
```

```
demoacnuc | on |    | Demo Database |
```

Configuring the dbplaces file.

The **dbplaces** contains:

```
dbplaces <- readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/dbplaces")
cat(dbplaces, sep = "\n")
```

```
#defines location of acnuc databases index files and flat files

setenv         swissprot         '/Users/mgouy/Documents/acnuc/petite/swissprot /Users/mgouy/Documents/acnuc/petite/sw
setenv         embl         '/Users/mgouy/Documents/acnuc/petite/embl /Users/mgouy/Documents/acnuc/petite/embl'
```

Each line set the acnuc and gcgacnuc variables for each database.

You should set the files **dbplaces** according to your installation: modify the **dbplaces** as follows:

```
setenv  demoacnuc       'ACNUC_HOME/ACNUC_DB/demoacnuc/index ACNUC_HOME/ACNUC_DB/demoacnuc/flat_
```

```
indexpath <- normalizePath("./ACNUC_HOME/ACNUC_DB/demoacnuc/index")
ffpath <- normalizePath("./ACNUC_HOME/ACNUC_DB/demoacnuc/flat_files")
newdb <- paste("setenv demoacnuc '", indexpath, " ", ffpath,
    "'", sep = "", collapse = "")
writeLines(newdb, "./ACNUC_HOME/ACNUC_SOFT/racnucd/dbplaces")
dbplaces <- readLines("./ACNUC_HOME/ACNUC_SOFT/racnucd/dbplaces")
cat(dbplaces, sep = "\n")
```

```
setenv demoacnuc '/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/demoacnuc/index /Users/lobry/seqinr/
```

Launch the server.

Finaly, in the ACNUC_HOME/ACNUC_SOFT/racnucd/ directory, lauche the server as follow :

```
setwd("./ACNUC_HOME/ACNUC_SOFT/racnucd")
system("./racnucd racnucd.new > racnucd.log &")
Sys.sleep(1)
system("ps | grep racnucd", intern = TRUE)
```

```
[1] "29658  p1  S+     0:00.01 ./racnucd racnucd.new"
[2] "29659  p1  S+     0:00.01 sh -c ps | grep racnucd"
[3] "29661  p1  U+     0:00.00 grep racnucd"
```

```
cat(readLines("racnucd.log"), sep = "\n")
```

```
******************************************************
Start of remote acnuc server : Fri Nov 23 17:30:55 2007
```

```
setwd(pwd)
```

The server is now ready.

5.4.4 Using seqinR to query your local socket server.

Launch R, load the `seqinr` package and type

```
choosebank(host="my_machine",port=49152,info=T)
```

for example:

```
library(seqinr)
hostname <- system("hostname", intern = TRUE)
hostname
```

```
[1] "sueoka.local"
```

```
choosebank(host = hostname, port = 49152, info = TRUE)
```

```
       bank status
1 demoacnuc     on
                                                              info
1 ACNUC database example. (September 2007) Last Updated: Oct 15, 2007
```

You can query the database. For example:

```
choosebank(bank = "demoacnuc", host = hostname, port = 49152)
query("mylist", "k=rib@ prot@")
mylist$nelem
```

```
[1] 39
```

```
sapply(mylist$req, getName)
```

```
 [1] "AP009048.PE25"   "AP009048.PE405"  "AP009048.PE830"  "AP009048.PE3223"
 [5] "AP009048.PE3465" "AP009048.PE3466" "AP009048.PE3516" "U00091.PE38"
 [9] "U00093.PE119"    "U00093.PE123"    "U00094.PE65"     "U00094.PE87"
[13] "U00094.PE262"    "U00094.PE393"    "U00094.PE400"    "X59720.PE36"
[17] "Y13134.PE91"     "Y13134.PE272"    "Y13135.PE271"    "Y13137.PE286"
[21] "Y13138.PE70"     "Y13138.PE198"    "Y13138.PE280"    "Y13139.PE53"
[25] "Y13139.PE110"    "Y13139.PE316"    "Y13140.PE89"     "Z47047.PE177"
[29] "Z47047.PE180"    "Z71256.PE178"    "Z71256.PE289"    "Z71256.PE313"
[33] "Z71256.PE317"    "Z71256.PE534"    "Z71256.PE637"    "Z71256.PE694"
[37] "Z71257.PE43"     "Z71257.PE75"     "Z71257.PE263"
```

5.5 Building your own ACNUC database.

One of the interest of a local server is to be able use your own ACNUC database.

5.5.1 Database flatfiles formats.

ACNUC database are build from flat files in several possible format : EMBL, Genbank or SwissProt. Instructions to install ACNUC databases are given at this url :

`http://pbil.univ-lyon1.fr/databases/acnuc/localinstall.html`

5.5.2 Download the ACNUC dababase management tools.

The code source of the ACNUC tools server are available on the PBIL server at this url:

`ftp://pbil.univ-lyon1.fr/pub/acnuc/unix/acnucsoft.tar`

5.5.3 Install the ACNUC dababase management tools.

ANCUC management tools are described at this url :

`http://pbil.univ-lyon1.fr/databases/acnuc/acnuc_gestion.html`

Let ACNUC_HOME/ACNUC_SOFT/tools be the base directory for the ACNUC tools.

```
dir.create("./ACNUC_HOME/ACNUC_SOFT/tools", showWarning = FALSE)
```

- Dowload the `acnucsoft.tar` file into ACNUC_HOME/ACNUC_SOFT/tools.

  ```
  download.file("ftp://pbil.univ-lyon1.fr/pub/acnuc/unix/acnucsoft.tar",
      destfile = "./ACNUC_HOME/ACNUC_SOFT/tools/acnucsoft.tar")
  ```

- Untar the `acnucsoft.tar` file

  ```
  setwd("./ACNUC_HOME/ACNUC_SOFT/tools/")
  system("tar -xvf acnucsoft.tar")
  system("rm -f acnucsoft.tar")
  setwd(pwd)
  ```

- Go into ACNUC_SOFT/ and type;

  ```
  make
  ```

  This should create the ACNUC management tools and ACNUC querying tools.

  ```
  setwd("./ACNUC_HOME/ACNUC_SOFT/tools/")
  system("make")
  dir()
  ```

```
 [1] "acnuc2fasta"          "acnuc2fasta.c"       "acnucf2c.c"
 [4] "acnucf2c.o"           "arbrebin.c"          "arbrebin.o"
 [7] "bit.c"                "bit.o"               "compressnewdiv"
[10] "compressnewdiv.c"     "connectindex"        "connectindex.c"
[13] "conv_to_bigannots"    "conv_to_bigannots.c" "coperations.c"
[16] "dir_acnuc.h"          "dir_io.c"            "dir_io.h"
[19] "dir_io.o"             "dynlist.c"           "dynlist.o"
[22] "extract.c"            "extract.o"           "fortran_ex.f"
[25] "gbemgener"            "gbemgener.c"         "gestion_acnuc.c"
[28] "gestion_acnuc.o"      "hashacc.c"           "hashacc.o"
[31] "initf"                "initf.c"             "libcacnuc.a"
[34] "lngbit.c"             "lngbit.o"            "makefile"
[37] "mdshrt_lng.c"         "mdshrt_lng.o"        "misc_acnuc.c"
[40] "misc_acnuc.o"         "ncbitaxo.h"          "newordalphab"
[43] "newordalphab.c"       "pretty_seq.c"        "pretty_seq.o"
[46] "proc_requete.c"       "proc_requete.o"      "query"
[49] "query.c"              "query.o"             "readidreport.c"
[52] "readidreport.o"       "readncbitaxo"        "readncbitaxo.c"
[55] "renamediv"            "renamediv.c"         "simext.h"
[58] "smjytload"            "smjytload.c"         "sortsubseq"
[61] "sortsubseq.c"         "supold"              "supold.c"
[64] "swgener"              "swgener.c"           "testmatchindex"
[67] "testmatchindex.c"     "two_banks.c"         "two_banks.o"
[70] "updatehelp"           "updatehelp.c"        "use_acnuc.c"
[73] "use_acnuc.o"          "utilgener.c"         "utilgener.h"
[76] "utilgener.o"          "utilgener2.c"        "utilgener2.o"
[79] "utilquery.c"          "utilquery.o"
```

```
setwd(pwd)
```

5.5.4 Database building : index generation

You can now build your own database. All you need is a flat files in EMBL, GenBank or SwissProt format. You can download a file example at :

`ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/escherichia_uniprot.dat.Z`

Let's use this SwissProt file to build your database

- Let ACNUC_HOME/ACNUC_DB/mydb be the directory for your databases.

```
dir.create("./ACNUC_HOME/ACNUC_DB/mydb", showWarning = FALSE)
```

This directory should contain the *index* and *flat_files* directories.

```
dir.create("./ACNUC_HOME/ACNUC_DB/mydb/index", showWarning = FALSE)
dir.create("./ACNUC_HOME/ACNUC_DB/mydb/flat_files", showWarning = FALSE)
```

- Download the `escherichia_uniprot.dat.Z` file into ACNUC_HOME/ACNUC_DB/mydb/flat_fil

```
download.file("ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/escherichia_uniprot.dat.Z",
    destfile = "./ACNUC_HOME/ACNUC_DB/mydb/flat_files/escherichia_uniprot.dat.Z")
```

- Uncompress the `escherichia_uniprot.dat.Z` file

```
setwd("./ACNUC_HOME/ACNUC_DB/mydb/flat_files/")
system("gunzip -f escherichia_uniprot.dat.Z")
setwd(pwd)
```

- A simple building of the index can be done with the script `buildindex.csh` available at:

ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/buildindex.csh

You can copy this file in ACNUC_HOME/ACNUC_DB/mydb and execute it by typing::

./buildindex.csh escherichia_uniprot

```
 download.file("ftp://pbil.univ-lyon1.fr/pub/seqinr/demoacnuc/buildindex.csh",
     destfile = "./ACNUC_HOME/ACNUC_DB/mydb/buildindex.csh")
 setwd("./ACNUC_HOME/ACNUC_DB/mydb/")
 system("chmod +x ./buildindex.csh")
 system("./buildindex.csh escherichia_uniprot >  ./build.log")
 cat(readLines("build.log", 50), sep = "\n")


 Build a protein database in:
 ============================
 ->/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/mydb

ACNUC environment:
 ->/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/mydb/index
 ->/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/mydb/flat_files

ACNUC tools in:
 ->/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/mydb/../../ACNUC_SOFT/tools

flat file: escherichia_uniprot.dat
 ->/Users/lobry/seqinr/inst/doc/src/mainmatter/ACNUC_HOME/ACNUC_DB/mydb/flat_files/escherichia_uniprot.dat

Begin to build index...
Fri Nov 23 17:31:29 CET 2007
====================================
Initialise
Normal end.

====================================
Generation des index
Program started at Fri Nov 23 17:31:30 2007

New division created: escherichia_uniprot
Start loading ncbi species taxonomy
Warning: file $acnuctaxo/id.report not found
Sequence loading started at Fri Nov 23 17:31:30 2007

---->3MG1_ECOLI
---->3MG2_ECOLI
---->6PGD_ECOLI
---->6PGL_ECOLI
---->A4UR75_ECOLI
---->A4UR76_ECOLI
---->A4UR77_ECOLI
---->A4UR78_ECOLI
---->A4UR79_ECOLI
---->A4UR80_ECOLI
---->A4UR81_ECOLI
---->A4UR83_ECOLI
---->A4UR84_ECOLI
---->A4UR86_ECOLI
---->A5A605_ECOLI
---->A5A607_ECOLI
---->A5A609_ECOLI
---->A5A611_ECOLI
---->A5A612_ECOLI
---->A5A613_ECOLI
---->A5A614_ECOLI


 cat(tail(readLines("build.log"), 50), sep = "\n")


---->ZRAS_ECOLI
---->ZUPT_ECOLI
---->ZUR_ECOLI
Program finished at Fri Nov 23 17:31:34 2007

lues=4461  chargees=4461  difference=0 seqs/second=1115.25
```

```
====================================
run newordalphab

Sorting file SUBSEQ.NEW
Writing list of loci and unvalid seqs
Sorting file SMJYT.NEW
Computing sequence hashing
Writing SPECIES.NEW
Writing KEYWORDS.NEW
Writing hashing data
Short lists of keywords and info records
Sorting file ACCESS.NEW
Sorting file BIBLIO.NEW
Writing LOCUS.NEW and lists of access#s and refers
Sorting file AUTHOR.NEW
Writing lists of seqs and authors for refers
Writing lists of refers for authors
Writing tree structure of keywords
Writing tree structure of species
Replacing old index files by new ones
Normal end

====================================
run updatehelp
Fri Nov 23 17:31:36 CET 2007
Index have been sucessfully build.
====================================

Testing the index:
==================
Opening a flat database in 2 divisions
Sorry, no help available for this command: CONT
[27 free lists available]

Command? (or H for help)
Enter your selection criteria, or H(elp) (EX: sp=equus and k=globin@)
List LIST1 contains 4461 sequences

Command? (or H for help)
List name,  or H(elp) ?  [LIST1] Name of file to write list content? [default= list1.mne]
Command? (or H for help)
End of ACNUC retrieval program
    4461    4461  182901 test.mn


 setwd(pwd)
```

You can check the building in the `build.log` file.

5.6 Misc

5.6.1 Other tools for acnuc

Several powerful tools dedicated to query ACNUC databases are available. The programs **query** and **query_win** allow to query an ACNUC database according to the same criteria than described in **seqinR**. It allows as well several functionality to extract biological data. **query_win** is a graphical version of query (*cf* figure 5.1). **query** is an command-line version which allows to query and ACNUC database through scripts. Both **query** and **query_win** are available as a *client* or a *local* application. More information on these programs can be found at: http://pbil.univ-lyon1.fr/software/query_win.html

Note: The *local* version of **query** is distributed with the ACNUC management tools, thus it is already available in your ./ACNUC_HOME/ACNUC_SOFT/tools/ directory. Before using it you need to set two environment variables, *acnuc* and *gcganuc* :

```
setenv acnuc MYDATABASE/index
setenv gcgacnuc MYDATABASE/flat_files
```

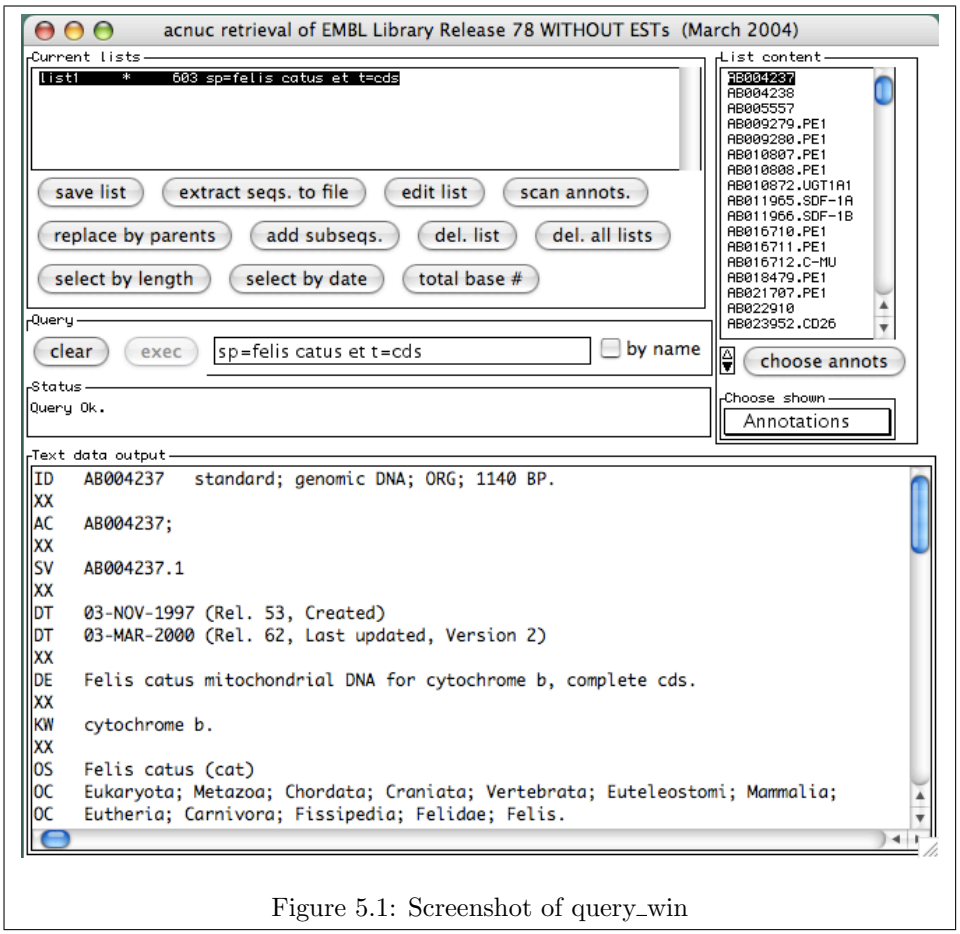


Figure 5.1: Screenshot of query_win

where MYDATABASE is the path to the database you want to query (for example: ./ACNUC_HOME/ACNUC_DB/demoacnuc/ or ./ACNUC_HOME/ACNUC_DB/mydb/)

5.7 Technical description of the racnucd daemon

Technical information about the acnuc socket server is available at this url: http://pbil.univ-lyon1.fr/databases/acnuc/racnucd.html.

5.8 ACNUC remote access protocol

Description of the socket communication protocol with acnuc is availble at this url: http://pbil.univ-lyon1.fr/databases/acnuc/remote_acnuc.html

5.9 Citation

You can use a citation along these lines:

> Sequences from [*cite your source of data*] were structured under the ACNUC model [23], hosted [at *give your URL if public*] by an ACNUC server [22] and analyzed with the **seqinR** client [7] under the R statistical environment [67].

For LaTeX users, these references are available in the `book.bib` file that ships with **seqinR** in the `seqinr/doc/src/config/` folder. To locate this file on your computer try:

```
(seqinrloc <- normalizePath(.path.package("seqinr")))
```

```
[1] "/Users/lobry/seqinr.Rcheck/seqinr"
```

```
setwd(seqinrloc)
dir()
```

```
 [1] "CITATION"    "CONTENTS"    "DESCRIPTION" "INDEX"       "Meta"
 [6] "R"           "R-ex"        "data"        "demo"        "doc"
[11] "exec"        "help"        "html"        "latex"       "libs"
[16] "man"         "sequences"
```

```
setwd("./doc/src/config")
dir()
```

```
[1] "atxy.sty"                  "authors.tex"
[3] "book.bib"                  "book.bib.~1.1.2.24.~"
[5] "commonrnw.rnw"             "commonrnw.rnw.~1.1.2.1.~"
[7] "commontex.tex"             "sessionInfo.rnw"
[9] "sessionInfo.rnw.~1.1.2.2.~"
```

```
cat(readLines("book.bib", n = 5), sep = "\n")
```

```
@incollection{seqinr,
    author = {Charif, D. and Lobry, J.R.},
    title = {Seqin{R} 1.0-2: a contributed package to the {R} project for statistical computing devoted t
    booktitle = {Structural approaches to sequence evolution: Molecules, networks, populations},
    year = {2007},
```

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Fri Nov 23 17:31:37 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 6

Multivariate analyses

Lobry, J.R.

Contents

6.1 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 [63]. 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 6.1 inspired from [17]) 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
```

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

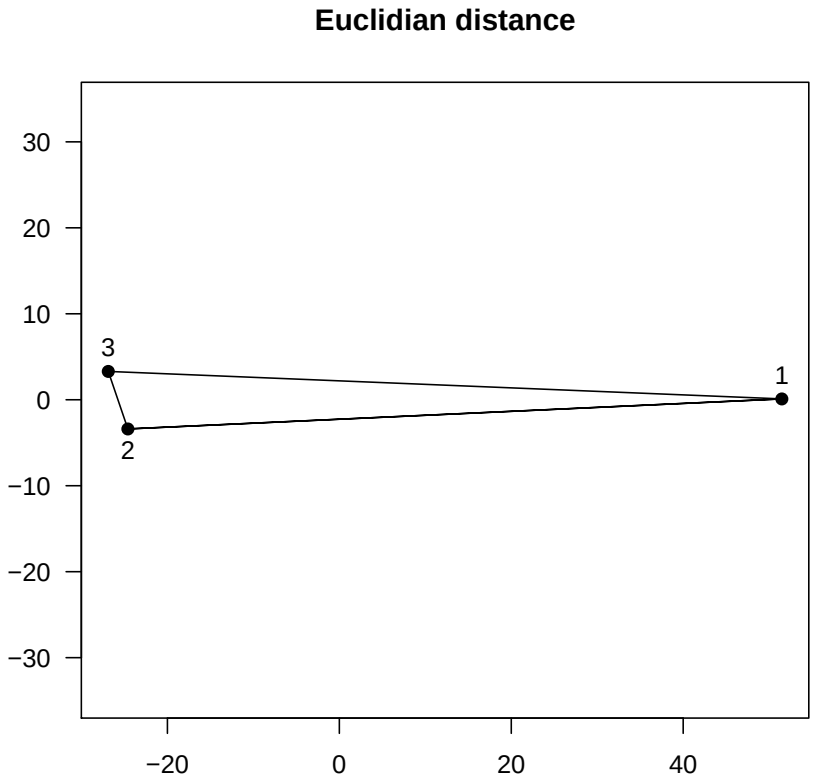
$$d^2(i,i') = \sum_{j=1}^{J}(n_{ij} - n_{i'j})^2 \tag{6.1}$$

to visualize this small data set:

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

Table 6.1: A very simple example of amino-acid counts in three proteins to be loaded with `data(toyaa)`.

```
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 (6.1) 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,

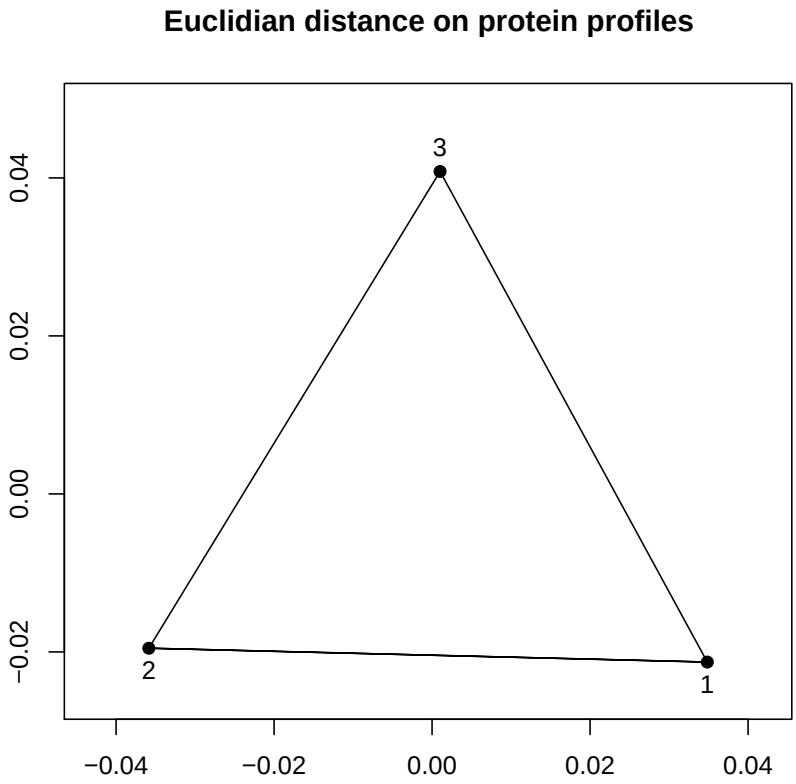
$$d^2(i,i') = \sum_{j=1}^{J} \left(\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}}\right)^2 \tag{6.2}$$

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
```

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

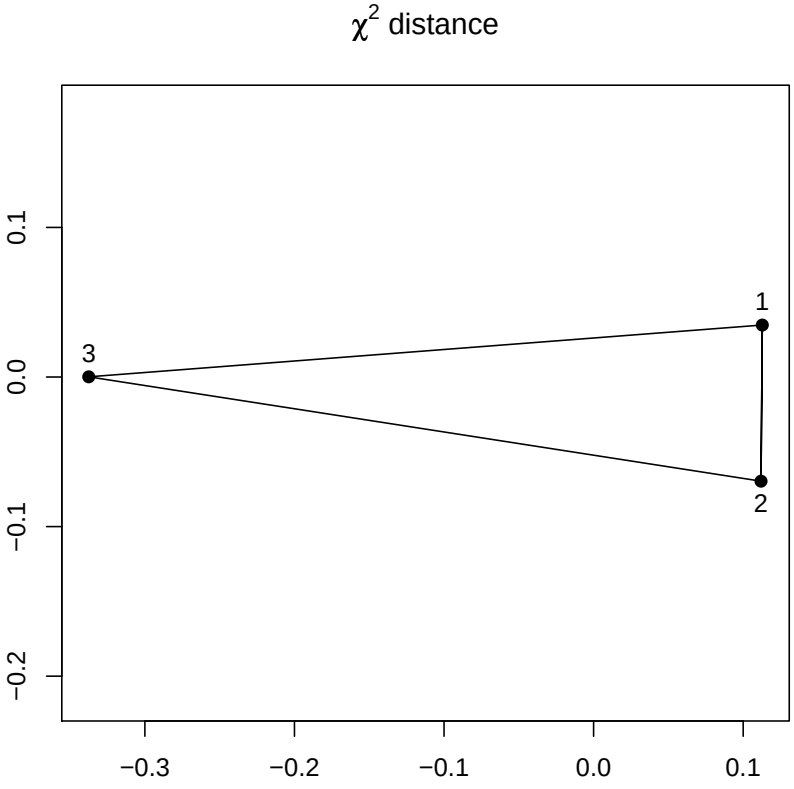


The pattern is now completely different with the three protein equally spaced. This is normal because in terms of relative amino-acid composition they are all differing two-by-two by 5% at the level of two amino-acids only. We have clearly removed the trivial protein size effect, but this is still not 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 (6.2) in that each square is weighted by the inverse of the frequency corresponding to each term,

$$d^2(i,i') = n_{\bullet\bullet} \sum_{j=1}^{J} \frac{1}{n_{\bullet j}} \left(\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}}\right)^2 \tag{6.3}$$

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 = "")
```



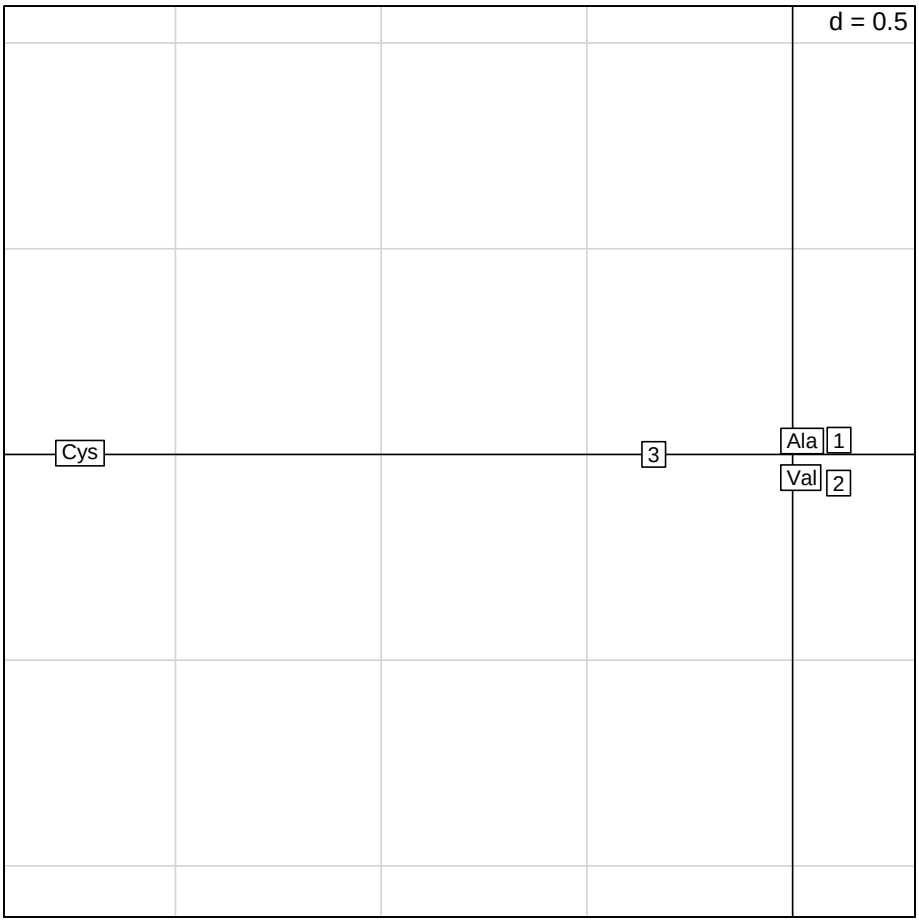
The pattern is completely different with now protein number 3 which is far away from the others because it is enriched in the rare amino-acid `Cys` as compared to others.

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")
```

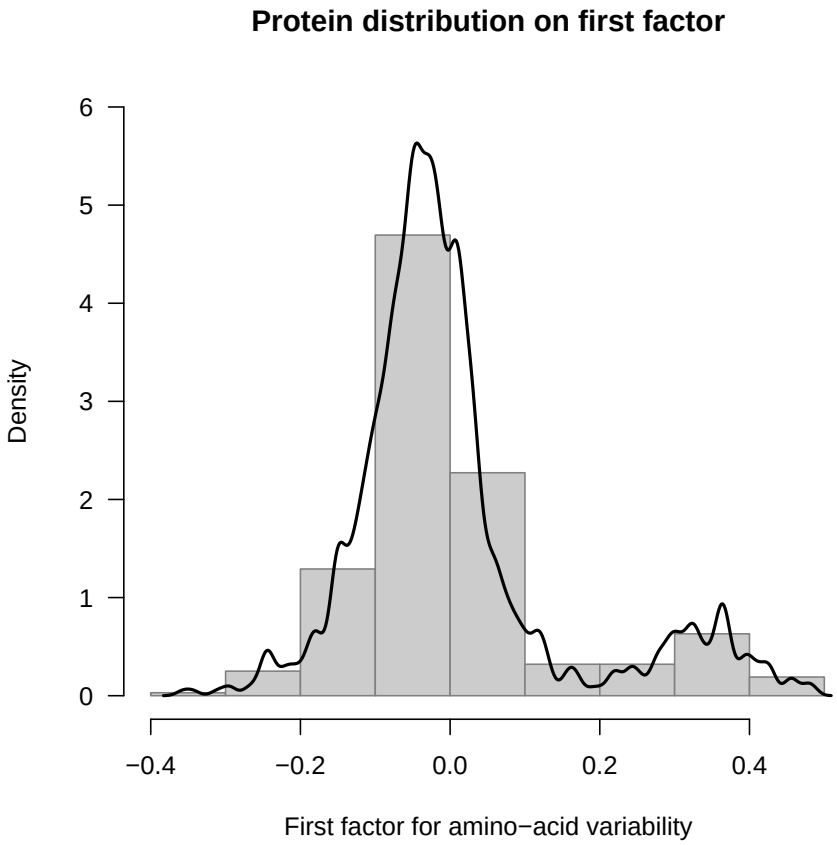
```
NULL
```



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 6.1 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).

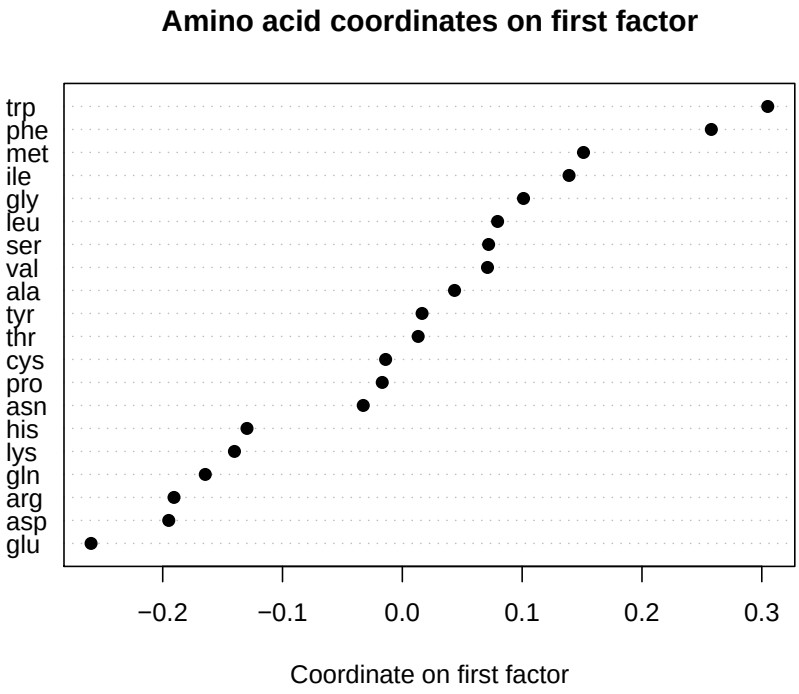
Data are from [48], a sample of the proteome of *Escherichia coli*. According to the title of this paper, the most important factor for the between-protein variability is hydrophilic - hydrophobic gradient. Let's try to reproduce this assertion :

```
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[40]) 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
```

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

```
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]
```

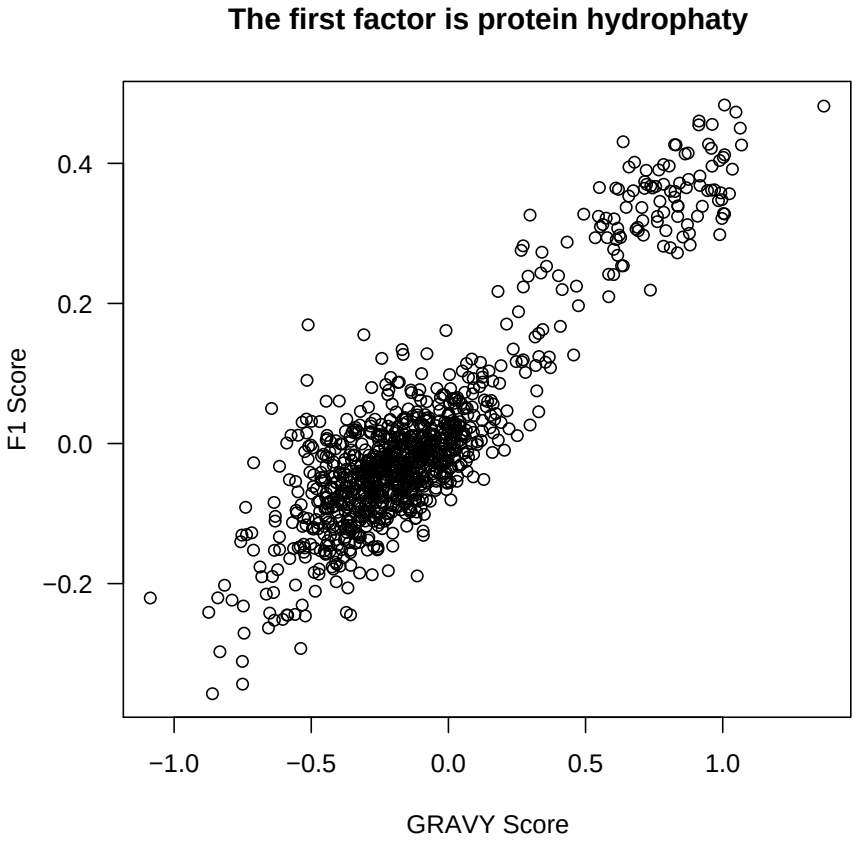
```
            ala        arg        asn        asp         cys
FOLE 0.08520179 0.05829596 0.04035874 0.05381166 0.008968610
MSBA 0.08247423 0.06529210 0.03608247 0.05154639 0.003436426
NARV 0.05309735 0.06637168 0.01769912 0.02212389 0.013274336
NARW 0.09090909 0.05627706 0.02597403 0.09090909 0.017316017
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:

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



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 6.2), 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:

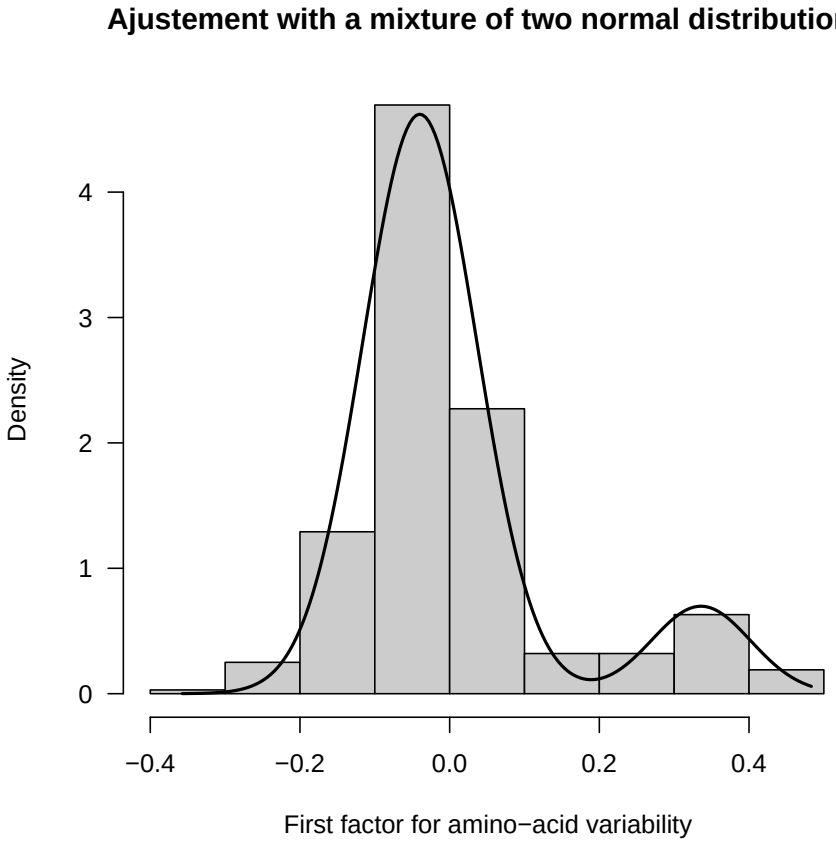
```
dmixnor <- function(x, p, m1, sd1, m2, sd2) {
    p * dnorm(x, m1, sd1) + (1 - p) * dnorm(x, m2, sd2)
}
library(MASS)
e <- fitdistr(F1, dmixnor, list(p = 0.88, m1 = -0.04, sd1 = 0.076,
    m2 = 0.34, sd2 = 0.07))$estimate
e
```

```
         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)
```

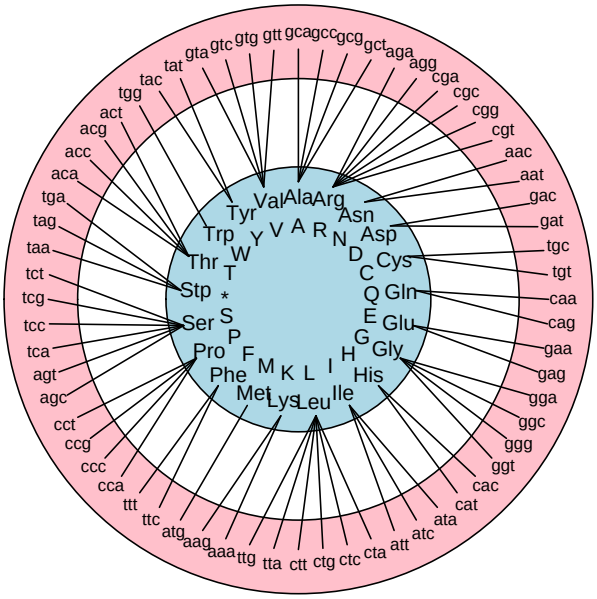
	d	p	q	r
beta	dbeta	pbeta	qbeta	rbeta
binom	dbinom	pbinom	qbinom	rbinom
cauchy	dcauchy	pcauchy	qcauchy	rcauchy
chisq	dchisq	pchisq	qchisq	rchisq
exp	dexp	pexp	qexp	rexp
f	df	pf	qf	rf
gamma	dgamma	pgamma	qgamma	rgamma
geom	dgeom	pgeom	qgeom	rgeom
hyper	dhyper	phyper	qhyper	rhyper
lnorm	dlnorm	plnorm	qlnorm	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 6.2: Density, distribution function, quantile function and random generation for the predefined distributions under R



6.2 Synonymous and non-synonymous analyses

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



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 [43]. The $\frac{K_a}{K_s}$ ratio is used as tool to evaluate selective pressure (see [31] 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 HSU78678.PE1
HSU78678.PE1    0.1243472
RNU73525.PE1    0.1405012    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 [63]. 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 [47], 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 6.3 to illustrate how to study synonymous and non-synonymous codon usage.

```
data(toycodon)
toycodon
```

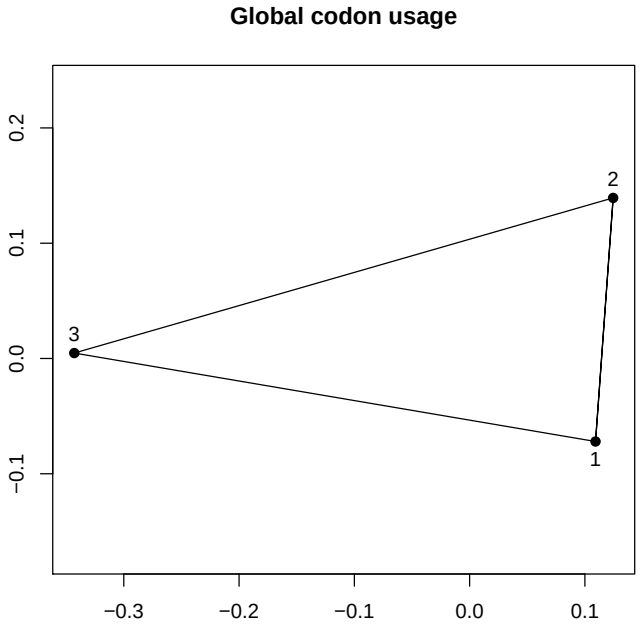
```
  gca gcc gcg gct gta gtc gtg gtt tgt tgc
1  33  32  32  33  18  17  17  18   0   0
2  13  17  17  13   8  12  12   8   0   0
3  16  14  14  16   8   9  10   8   3   2
```

	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 6.3: 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] "gcagccgcggctgtagtcgtggtttgttgc"
```

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"
```

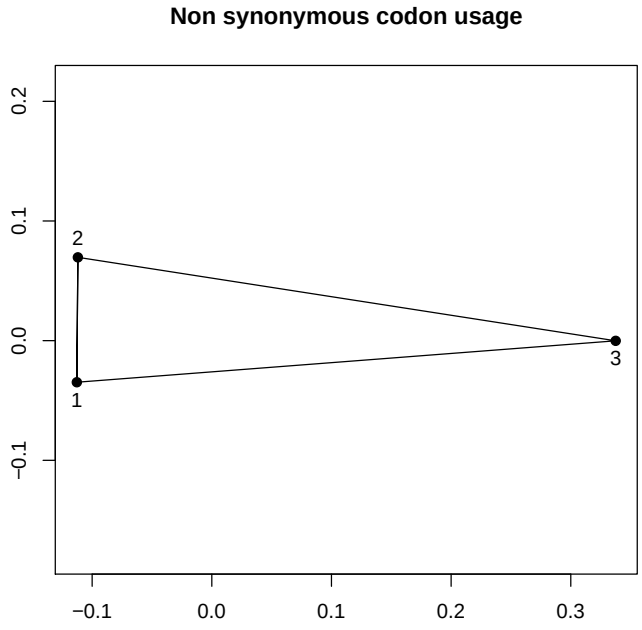
Make this a factor:

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



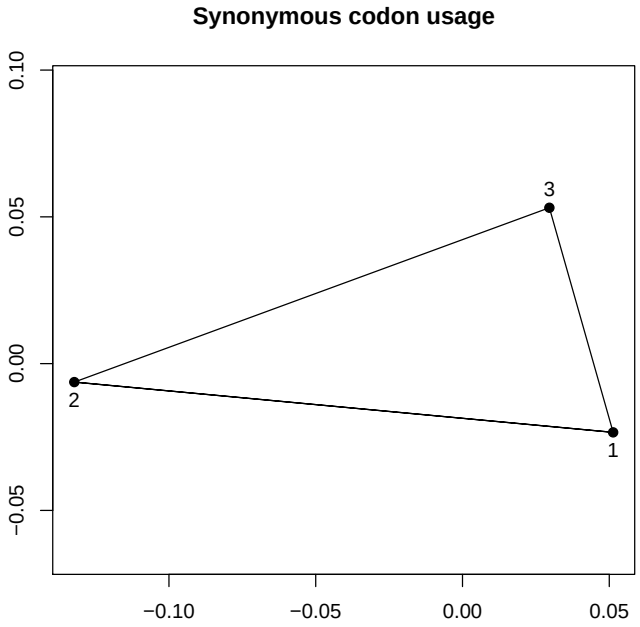
This is reminiscent of something, let's have a look at amino-acid counts:

```
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
```

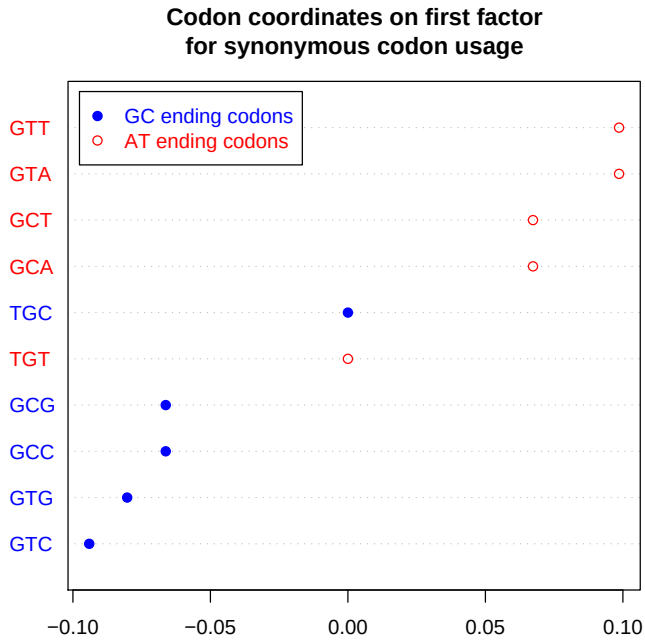
This is exactly the same data set that we used previously (table 6.1) at the amino-acid level. The non synonymous codon usage analysis is exactly the same as the amino-acid analysis. Coding sequence number 3 is far away because it codes for many Cys, a rare amino-acid. Note that at the global codon usage 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(within(dudi = 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:

```
tmp <- synonymous$co[, 1, drop = FALSE]
tmp <- tmp[order(tmp$Axis1), , drop = FALSE]
colcod <- sapply(rownames(tmp), function(x) ifelse(substr(x,
    3, 3) == "c" || substr(x, 3, 3) == "g", "blue", "red"))
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.

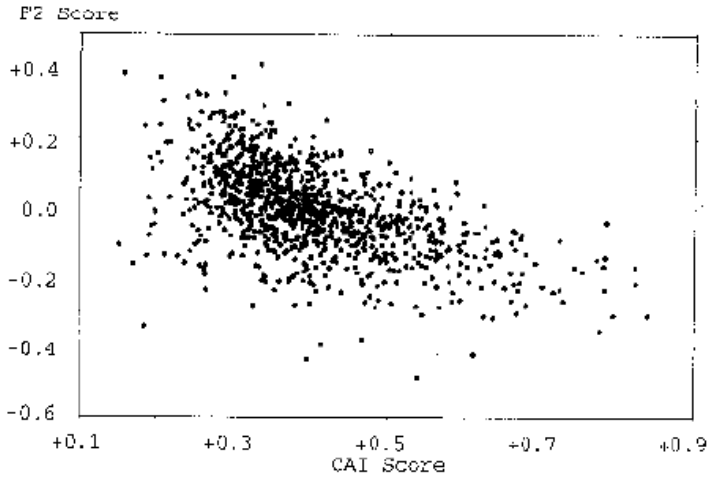
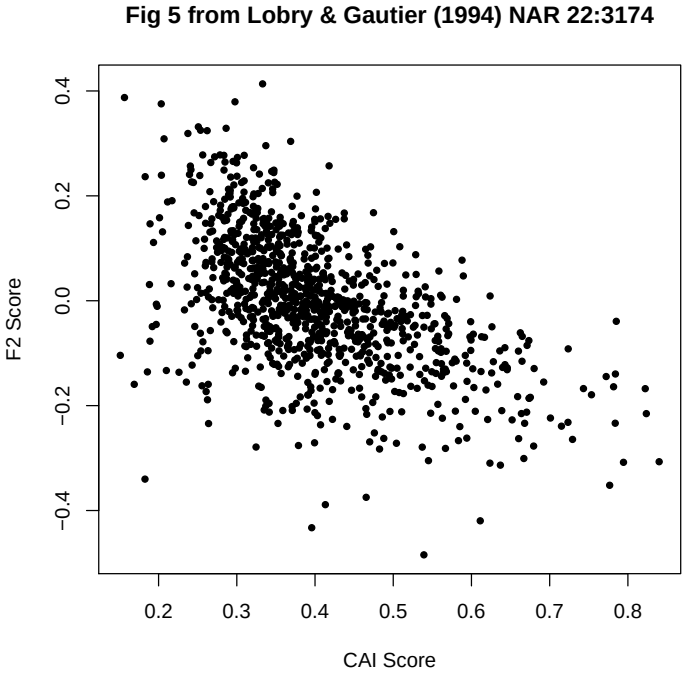


Figure 6.1: Screenshot of figure 5 from [48]. Each point represents a protein. This was to show the correlation between the codon adaptation index (CAI Score) with the second factor of correspondence analysis at the amino-acid level (F2 Score). Highly expressed genes have a high CAI value.

To illustrate the interest of synonymous codon usage analyses, let's use now a more realistic example. In [48] 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 6.1), 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 [73]) 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 [48] 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 [48] 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)
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))
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]])
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)
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))))))
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 <- within(dudi = ec999.coa, fac = facaa, scannf = FALSE)
ec999.syn
```

```
Within analysis
call: within(dudi = 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 table 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.

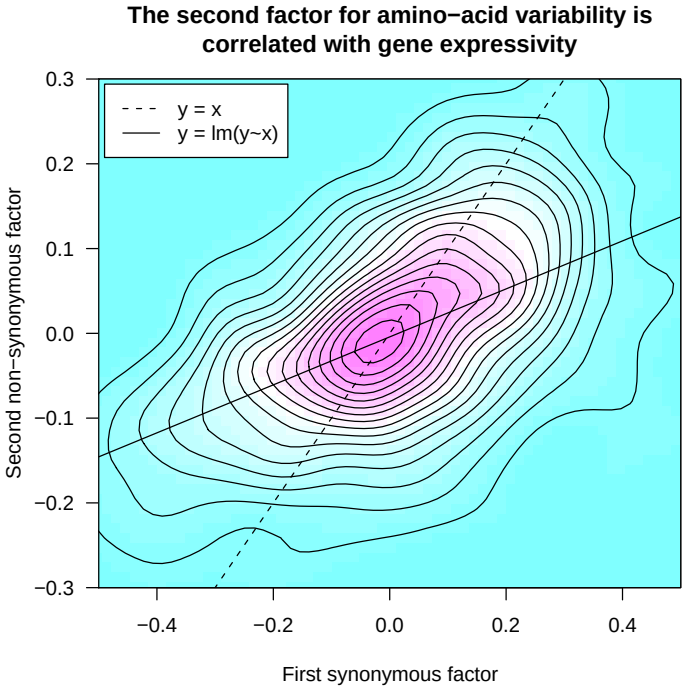
```
ec999.btw <- between(dudi = ec999.coa, fac = facaa, scannf = FALSE)
ec999.btw
```

```
Between analysis
call: between(dudi = 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
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`.

To be completed

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15, tools 2.6.0

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

There were two compilation steps:

- R compilation time was: Wed Nov 7 12:01:38 2007
- LaTeX compilation time was: December 2, 2007

CHAPTER 7

Nonparametric statistics

Palmeira, L. Lobry, J.R.

Contents

7.1 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 7.3).

7.2 Elementary nonparametric statistics

7.2.1 Introduction

Those rank statistics are those that were available under the ANALSEQ software [33, 25]. 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
```

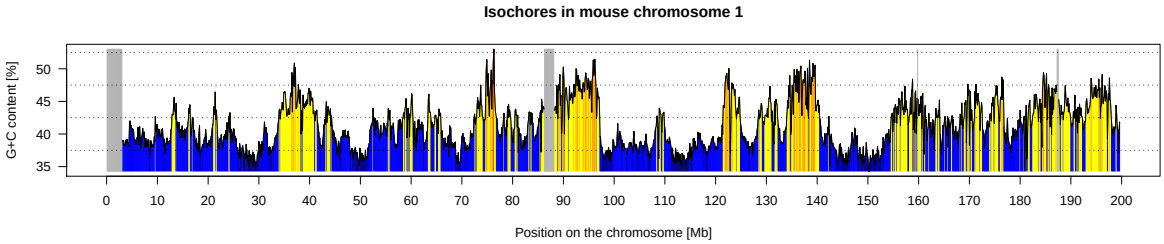
With one exception, the statistics names are the same as in the ANALSEQ software.

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 <- 201
res <- rep(-1, 10 * n)
chr1 <- paste("MOUSE1_", 1:n, sep = "")
i <- 1
for (frag in chr1) {
    myseq <- gfrag(frag, 1, 10^7)
    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")
```

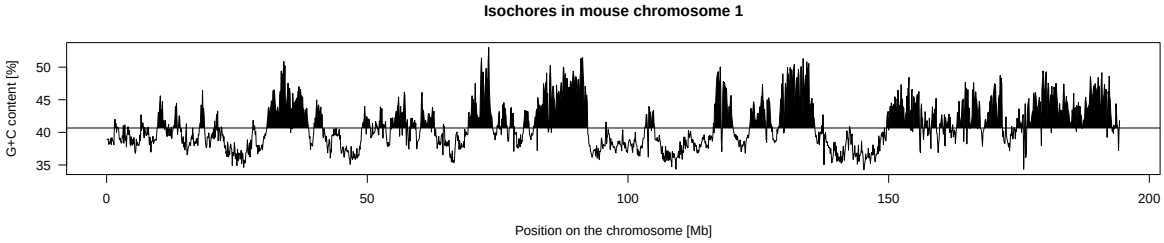
The folowing representation follows the conventions used in Fig 2 from [60].

```
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)
lev <- cut(res, breaks = breaks, labels = c("darkblue", "blue",
    "yellow", "orange", "red"), ordered = T)
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)
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
```

7.2.2 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
```

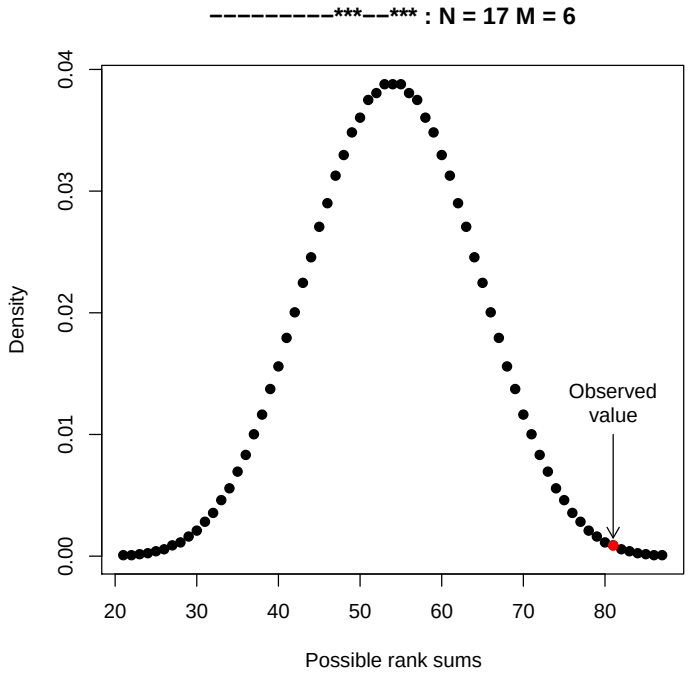
Here is a way to obtain the same result using the standard `wilcox.test()` function to make a Wilcoxon's rank sum test [82]:

```
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
pdf <- dwilcox(x = 0:(n * m), m = m, n = n)
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] 10.81602
```

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.

7.2.3 Rank variance

This statistics is the variance of ranks:

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

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

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

```
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("------****-------") == "*")
```

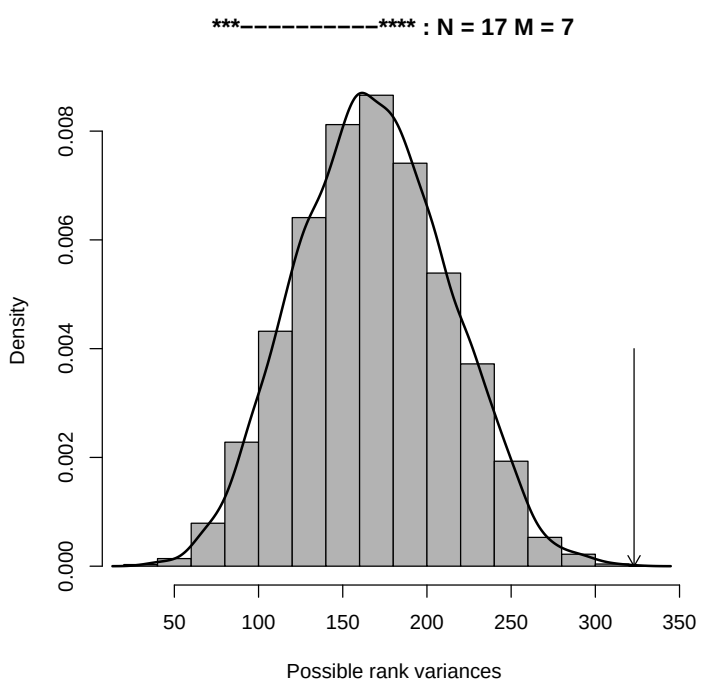
```
$VR
[1] 6

$stat
[1] -2.337860
```

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



Real case application

```
VR(appli)$stat
```

```
[1] 4.587283
```

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.

7.2.4 Clustering around the observed centre

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

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

The statistic CC[1] is the dispersion around $C(\omega)$ is defined by:

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

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

Noting $[x]$ the floor of x, we have:

$$E(CC) = \frac{(N+1)\left[\frac{M}{2}\right]\left[\frac{M+1}{2}\right]}{M+1}$$

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] 3.266275
```

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

[1] 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.

7.2.5 Number of runs

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

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

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

```
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** [80] this way:

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

```
Standard Normal
       3.786054
```

Real case application

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

```
[1] -33.78859
```

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.

7.2.6 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)
```

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

```
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] 320.8337
```

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.

7.3 Dinucleotides over- and under-representation

7.3.1 Introduction

We will briefly describe two statistics for the measure of dinucleotide over- and under-representation in sequences [36, 58], 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 [74, 2].

7.3.2 The *rho* statistic

The ρ statistic (`rho()`), presented in [36], 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.

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. 7.1)

```
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.

7.3.3 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

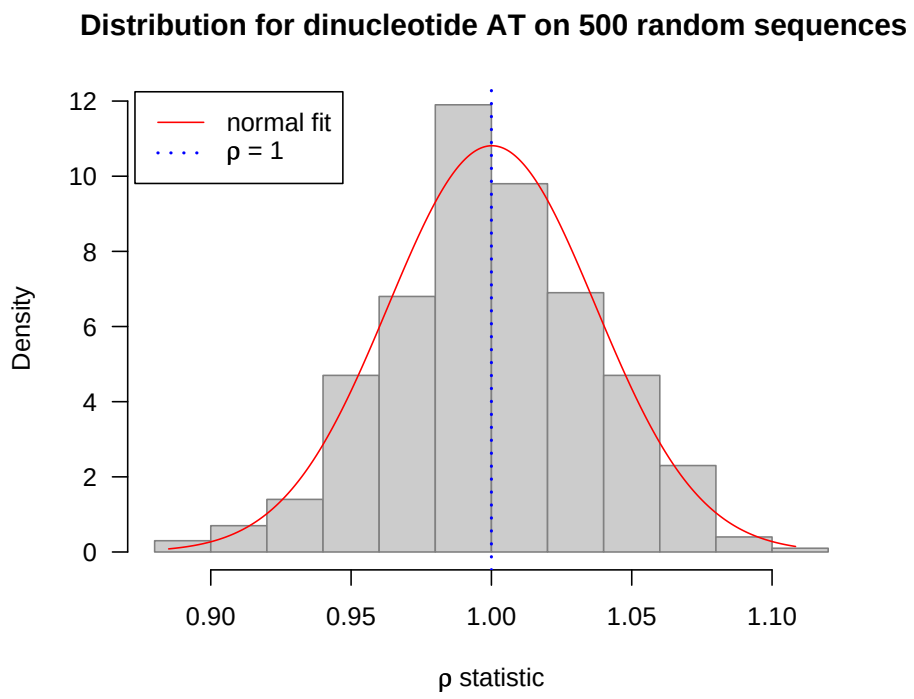


Figure 7.1: Distribution of the ρ statistic computed on 500 random sequences of length 6000. The vertical dotted line is centered on 1. The curve draws the fitted normal distribution.

under-representation [58]. 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.

7.3.4 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 CDS with accession number `U00096.PE448`:

```
choosebank("greview")
query("coli", "N=U00096.PE448")
sequence <- getSequence(coli$req[[1]])
annot <- getAnnot(coli$req[[1]])
closebank()
cat(annot, sep = "\n")
```

```
FT   CDS             complement(478591..479142)
FT                   /codon_start=1
FT                   /gene="maa"
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="InterPro:IPR001451"
FT                   /db_xref="InterPro:IPR011004"
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"
```

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 [71], and the new version of this calculation can be obtained with the argument `exact` set to `TRUE` (`FALSE` being the default). The following code was used to produce figure 7.2:

```
rhocoli <- rho(sequence)
zcolibase <- zscore(sequence, model = "base", exact = TRUE)
zcolicodon <- zscore(sequence, model = "codon")
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 7.2 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 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...

7.4 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 [58].

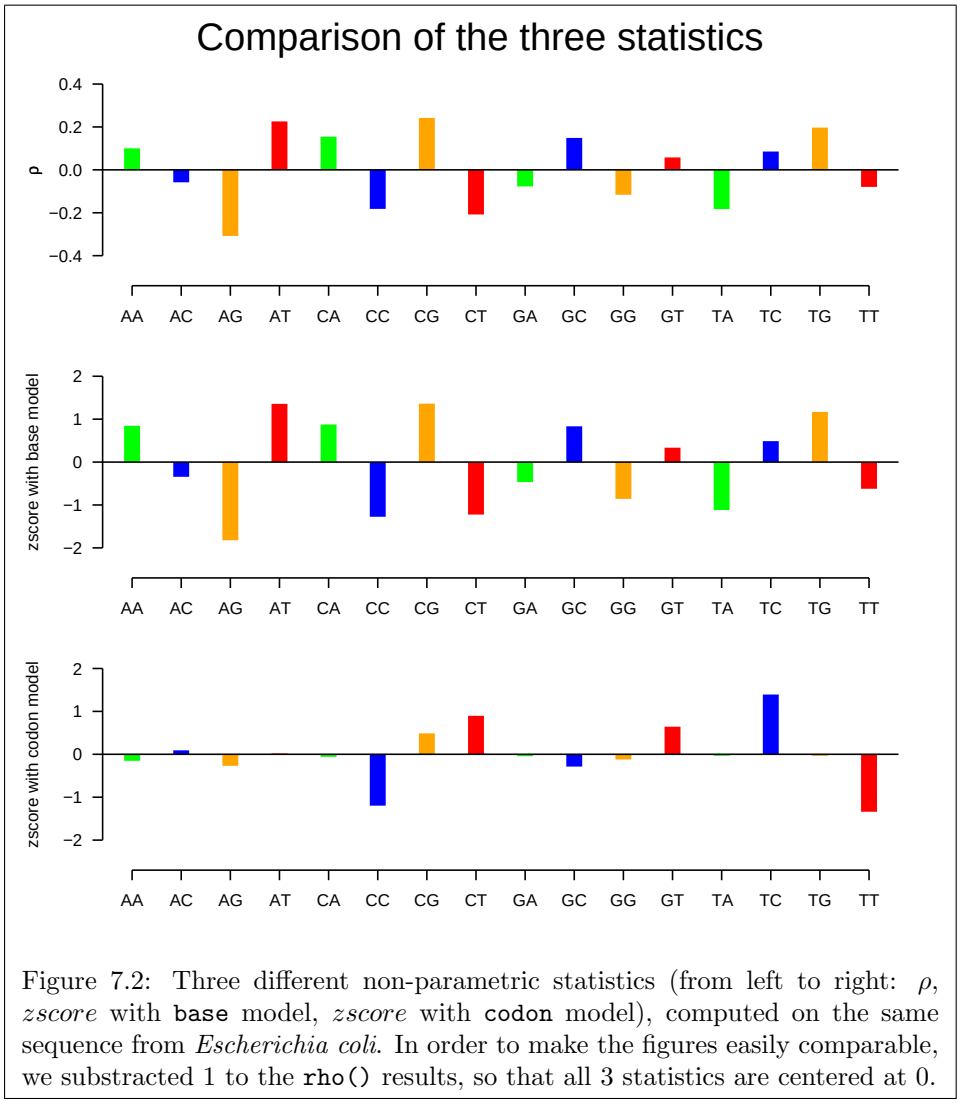


Figure 7.2: Three different non-parametric statistics (from left to right: ρ, *zscore* with `base` model, *zscore* with `codon` model), computed on the same sequence from *Escherichia coli*. In order to make the figures easily comparable, we substracted 1 to the `rho()` results, so that all 3 statistics are centered at 0.

7.4.1 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 [72]; the sensitivities of the four pyrimidine dinucleotides to UV wavelengths differ and depend on the micro-organism [72]:

	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 7.1: Proportion of dimers formed in the DNA of three bacteria after irradiation with 265 nm UV light. Table adapted from [72].

The hypothesis presented by Singer and Ames [74] 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 7.3.

```
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)
yrand <- sapply(xval/100, randomcase)
yworst <- sapply(xval/100, worstcase)
ybest <- sapply(xval/100, bestcase)
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)
```

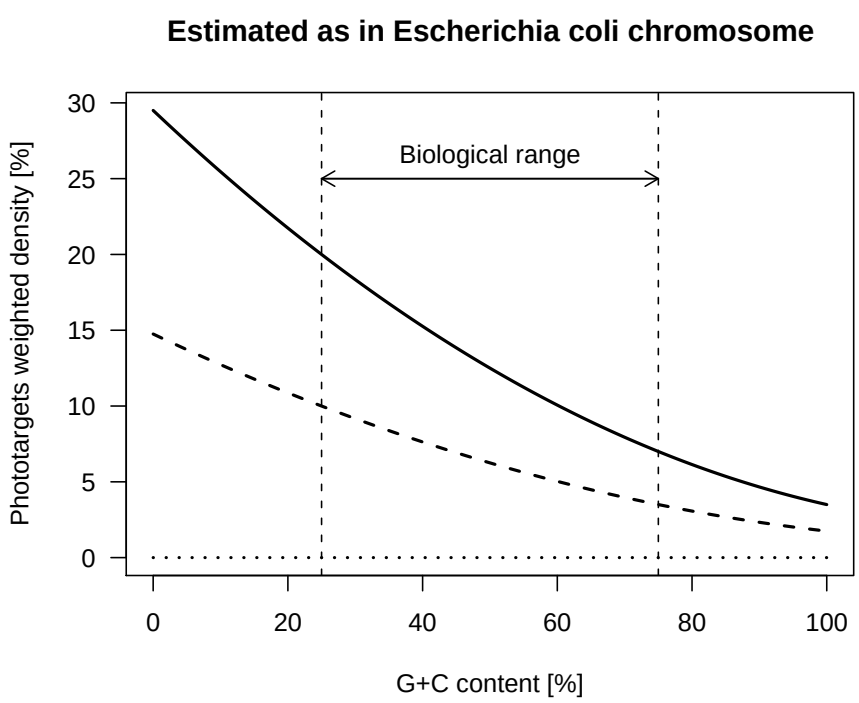


Figure 7.3: Density of phototargets, weighted by their frequency in the *Escherichia coli* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.59 * f_{tt} + 0.34 * (f_{tc} + f_{ct}) + 0.07 * f_{cc}$ (where f_{xy} is the frequency 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 ($f_{xy} = f_x \times f_y$).

In a *Micrococcus lysodeikticus* sequence (the following code was used to produce figure 7.4), 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)
yrand <- sapply(xval/100, randomcase)
yworst <- sapply(xval/100, worstcase)
ybest <- sapply(xval/100, bestcase)
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 7.3 and 7.4) show that the density of phototargets depends on:

- the degree of aggregation of pyrimidine dinucleotides in the sequence,
- 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?

7.4.2 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 7.5 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) {
```

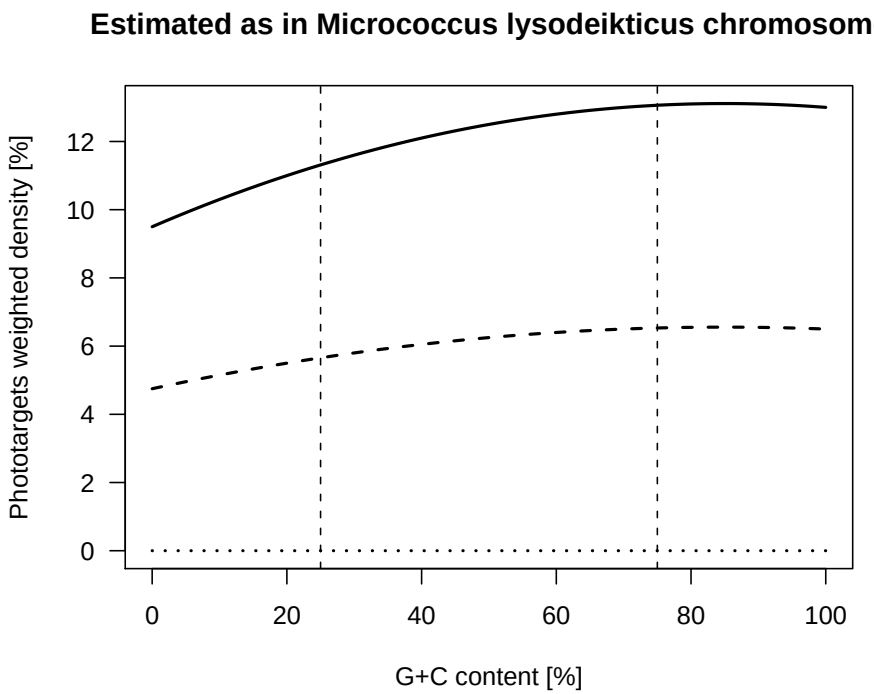


Figure 7.4: Density of phototargets, weighted by their frequency in the *Micrococcus lysodeikticus* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.19 * f_{tt} + 0.55 * (f_{tc} + f_{ct}) + 0.26 * f_{cc}$. See figure 7.3 for more details.

```
    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(0.1, 0.1, 0.1, 0.5), bg = rgb(0.5, 0.5,
            0.5, 0.5))
    legend("bottomright", inset = 0.02, legend = paste(substr(x,
        1, 1), "p", substr(x, 2, 2), " bias", sep = ""), cex = 1.25,
        bg = "white")
    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. 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 7.6 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")
```

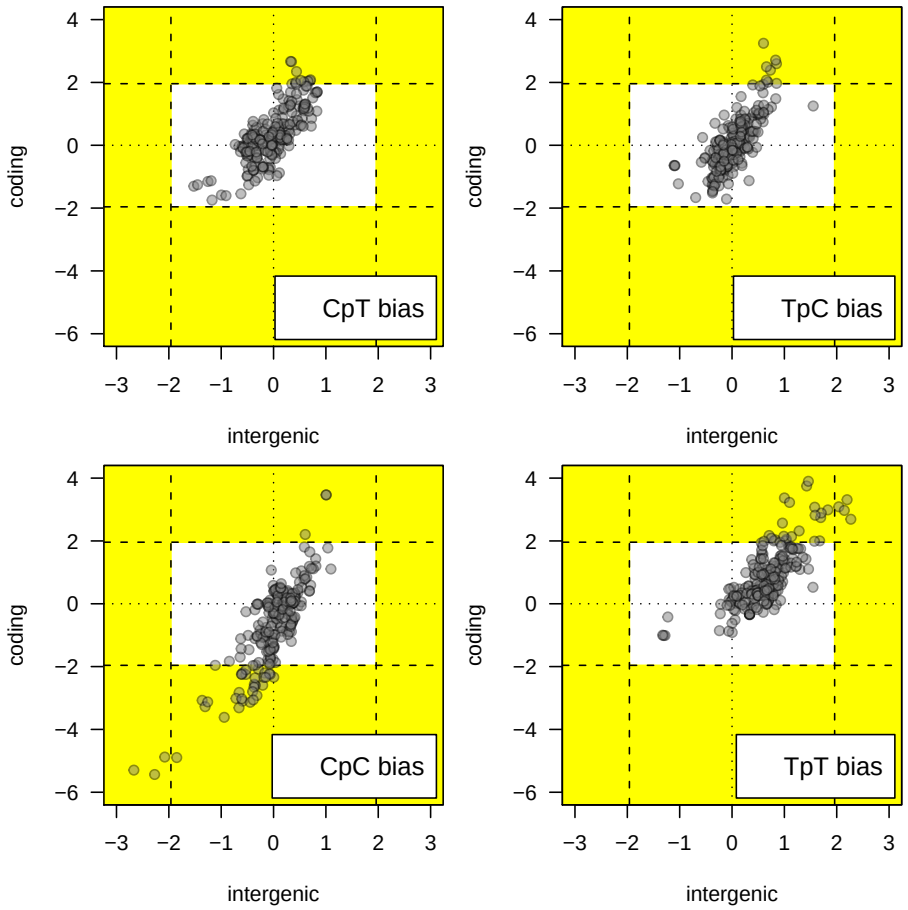


Figure 7.5: Plot of the mean *zscore* statistics for **intergenic sequences** (x-axis) and for **coding sequences** (y-axis), for each of the four pyrimidine dinucleotides. On each plot, a dot corresponds to the mean of these two statistics in a given prokaryote chromosome. The null x and y axis (dotted 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.

```
oneplot("CC")
oneplot("TT")
```

Figure 7.6 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 [74] no longer stands [58].

Session Informations

This part was compiled under the following R environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- 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, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, quadprog 1.4-11, seqinr 1.1-3, tseries 0.10-11, xtable 1.5-1, zoo 1.4-0
- Loaded via a namespace (and not attached): grid 2.6.0, tools 2.6.0

There were two compilation steps:

- R compilation time was: Sat Nov 24 19:01:55 2007
- LaTeX compilation time was: December 2, 2007

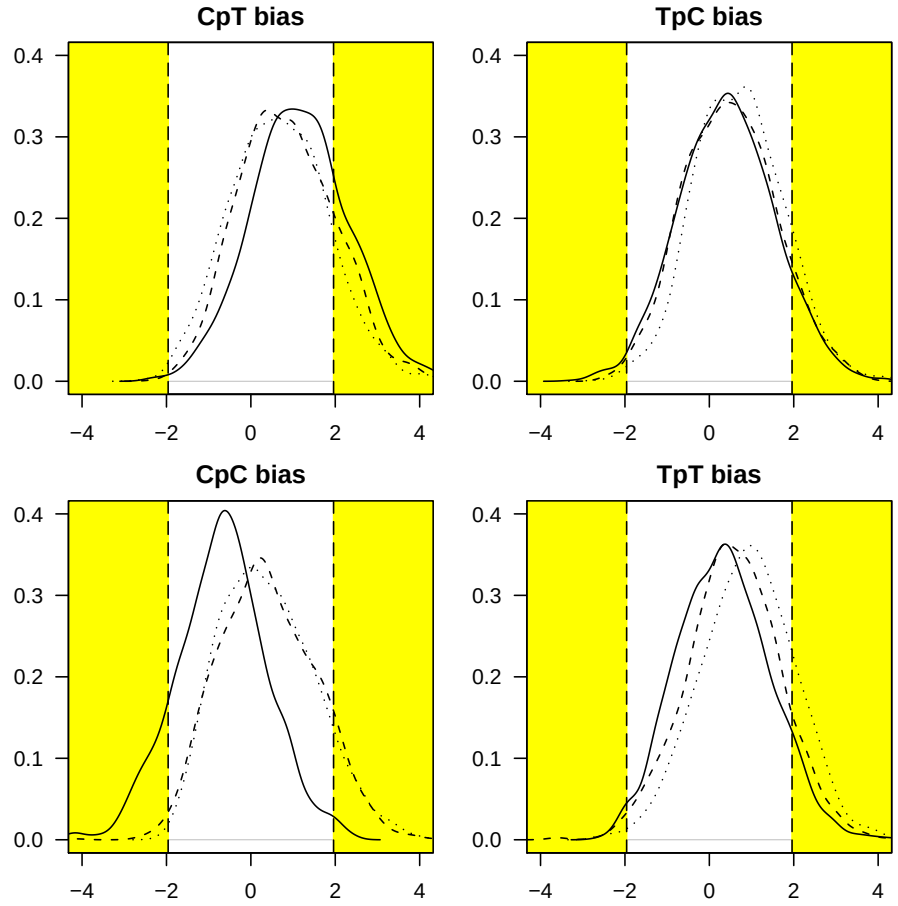


Figure 7.6: Each figure shows the distributions of the *zscore* in all **coding sequences** corresponding to each of the three strains of *Prochlorococcus marinus*. In each figure, the distribution for the MED4 (a high-light adapted strain) is shown as a solid line; the distribution for the SS120 (a 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.

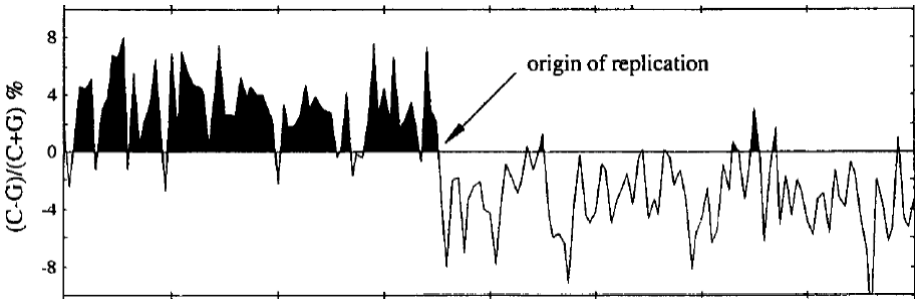
APPENDIX A

FAQ: Frequently Asked Questions

Lobry, J.R.

A.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 [45] whose screenshot is given just below:



The score represented 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. Let's put this fragment into the string `myseq`:

```
choosebank("greview")
myseq1 <- gfrag("U00096", start = 3217270, length = 10^7)
myseq2 <- gfrag("U00096", start = 1, length = 194133)
closebank()
myseq <- paste(myseq1, myseq2, sep = "")
nchar(myseq)
```

```
[1] 1616539
```

This is not exactly the same sequence that was used in [45] 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:

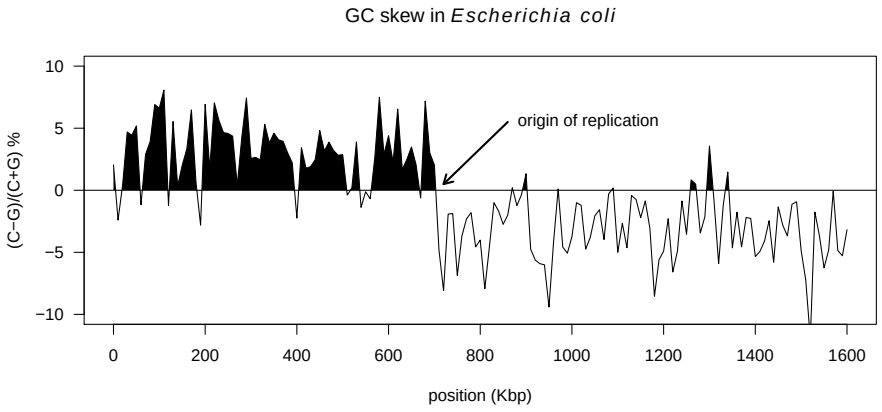
```
step <- 10000
wsize <- 10000
starts <- seq(from = 1, to = nchar(myseq), by = step)
starts <- starts[-length(starts)]
n <- length(starts)
result <- numeric(n)
for (i in seq_len(n)) {
    result[i] <- gcskew(substr(myseq, starts[i], starts[i] +
        wsize - 1))
}
```

And we plot[2] the result:

[1] The sequence used in [45] was a 1,616,174 bp fragment obtained from the concatenation of nine overlapping sequences (U18997, U00039, L10328, M87049, L19201, U00006, U14003, D10483, D26562 [75, 6, 10, 64, 3, 83]). Ambiguities have been resolved since then and its was a chimeric sequence from K-12 strains MG1655 and W3110 [28], the sequence used here is from strain MG1655 only [4].

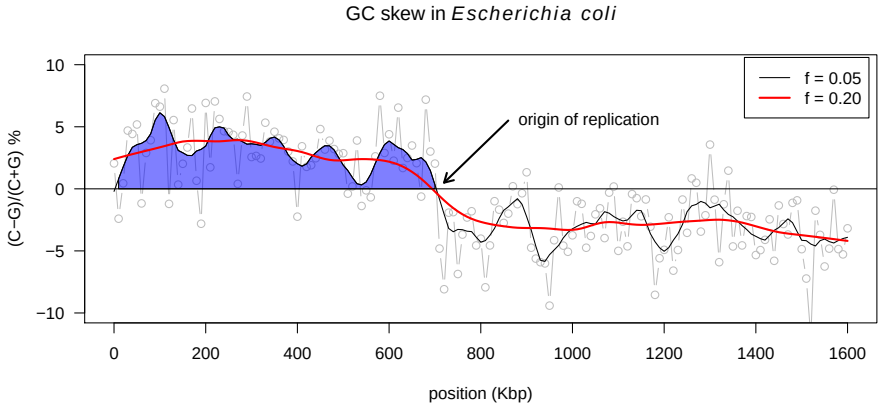
[2] This code is adapted from the code at http://www.stat.auckland.ac.nz/~paul/RGraphics/chapter3.html for figure 3.25 in Paul Murrell's book [52]. This book is a must read if you are interested by R's *force de frappe* in the graphic domain.

```
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)
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` parameter 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. This is for instance what can be obtained with the same data with the `lowess()` function with two values for the smoothing parameter `f`:

```
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)
```



A.2 How can I extract just a fragment from my sequence?

Use the generic function `getFrag()` :

```
choosebank("emblTP")
query("mylist", "AC=A00001")
getFrag(mylist$req[[1]], begin = 10, end = 20)
```

```
 [1] "g" "a" "t" "g" "g" "a" "g" "a" "a" "t" "t"
attr(,"seqMother")
[1] "A00001"
attr(,"begin")
[1] 10
attr(,"end")
[1] 20
attr(,"class")
[1] "SeqFrag"
```

```
closebank()
```

A.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")
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 11.777516 10.619175 11.365738  9.460987 10.864793
[50] 13.002380  9.845859 10.584868 11.421257 10.248320 11.031938 10.402075
[57]  4.863862  6.612505  9.681066 11.150310 11.182505 11.043602  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
```

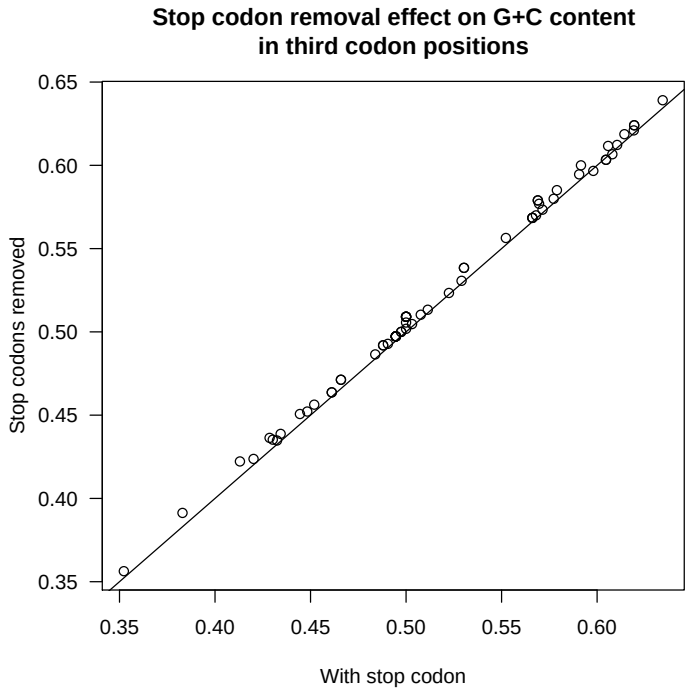
A.4 Why do I have not exactly the same G+C content as in codonW?

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 [62] 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\nin 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
AAAAGACTTATGAATGGCATAATCTTCAAGTTTGACTTCTTTGGTTTTGCCCTCTGCTCT
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)
```

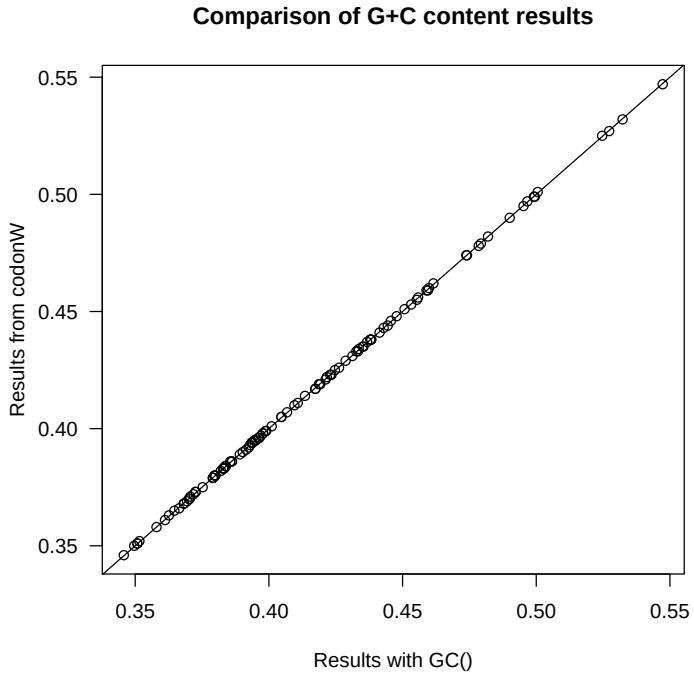
```
                       title  GC3s    GC
1 YCG9_Probable_____________13 0.335 0.394
2 YCG8_________573_residues_ 0.439 0.446
3 ALPHA2_________633_residue 0.328 0.351
4 ALPHA1_________528_residue 0.345 0.379
5 CHA1___________1083_residue 0.328 0.394
6 KRR1___________951_residue 0.364 0.384
```

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))
```

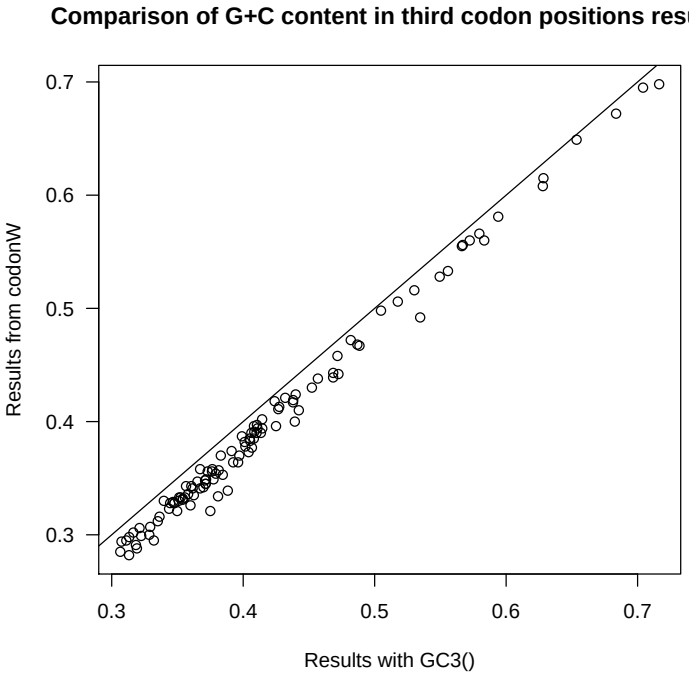


The results are consistent if we consider that we have 3 significant digits in the file `input.out`. Now, let's try to reproduce the results for G+C in third codon positions:

```
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))
```

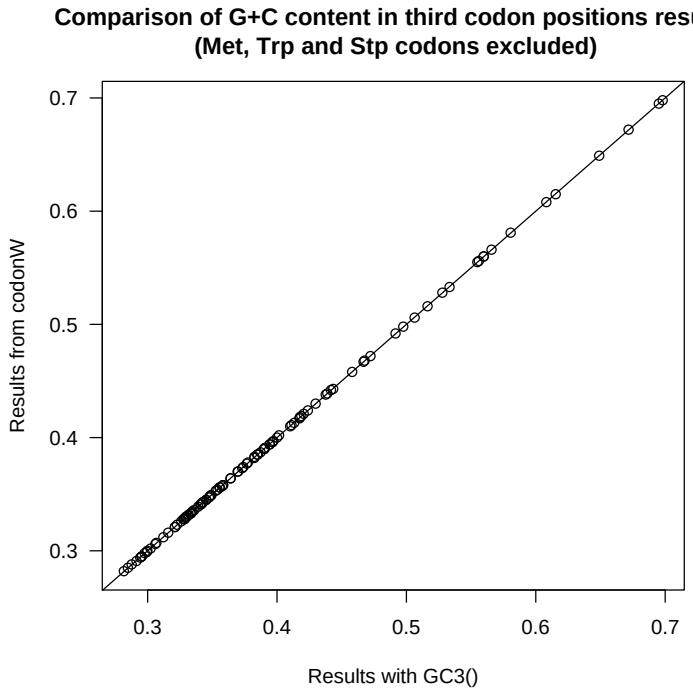


There is clearly a problem here. Looking into the documentation of `codonW`, GC3s is the G+C content in third codon position after removing non-synonymous and stop codons (those corresponding to Met, Trp, Stp). Let's remove these codons:

```
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
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 [77] by Sueoka in 1988:

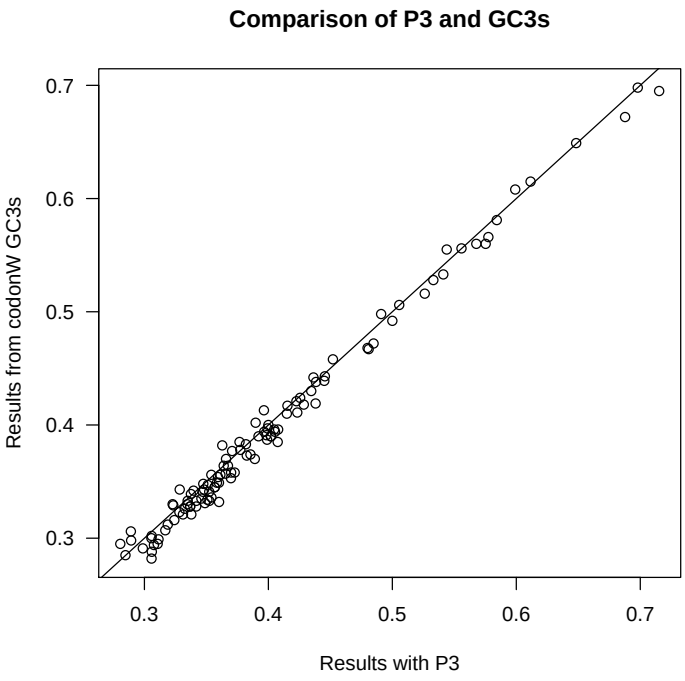
> **G + C Content of the Three Codons Positions.** In the present analysis, observed G + C contents of the first, second, and third codon positions (P_1, P_2, and P_3, respectively) are corrected average G + C contents of the three codon positions that are calculated from 56 triplets out of 64. Because 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 [78].

A.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")
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 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Wed Nov 7 12:30:15 2007
- LaTeX compilation time was: December 2, 2007

APPENDIX B

GNU Free Documentation License

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.

B.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.

B.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.

B.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.

B.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.

B.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".

B.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.

B.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.

B.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.

B.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.

B.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.

APPENDIX C

Genetic codes

Lobry, J.R.

C.1 Standard genetic code

The standard genetic code given in table C.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")
```

C.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 [55, 34].

```
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
```

[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 C.1: Genetic code number 1: standard.

```
16        chlorophycean.mitochondrial
21            trematode.mitochondrial
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), sep = "")
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 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils

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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.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 C.17: Genetic code number 23: hraustochytrium.mitochondria.

- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Wed Nov 7 12:41:49 2007
- LaTeX compilation time was: December 2, 2007

APPENDIX D

Release notes

Lobry, J.R. Necşulea, A. Palmeira, L. Penel, S.

1.1 series

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 [45].
- New dataset `waterabs` giving the absorption of light by water. This dataset was compiled by Palmeira [57] from [44, 65].
- 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 semicolon 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 [56]). 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 NNNNNNNNNNN). 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 [70, 11].

- 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 [18, 19] : 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 [37, 79, 54] 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 6.3.
- 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 1.2 and table 1.3 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 [36].

  - `zscore()` computes the zscore statistic on dinucleotides as defined in [58].

- 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)

A BibTeX entry for LaTeX users is

  @incollection{,
    author = {D. Charif and J.R. Lobry},
    title = {Seqin{R} 1.0-2: a contributed package to the {R} project for statistical computing dev
    booktitle = {Structural approaches to sequence evolution: Molecules, networks, populations},
    year = {2007},
    editor = {U. Bastolla, M. Porto, 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},
  }

Note that the orginal article updated is available in the
/Users/lobry/seqinr.Rcheck/seqinr/doc/ folder in PDF format
```

- 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 2.6.1 (2007-11-26), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-38, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.17-2, nlme 3.1-86, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.1, rcompgen 0.1-17

There were two compilation steps:

- R compilation time was: Sun Dec 2 17:47:16 2007
- LaTeX compilation time was: December 2, 2007

APPENDIX E

Test suite: run the don't run

Lobry, J.R.

E.1 Introduction

Many seqinR functions use socket connections to retrieve information from the internet. As a consequence, most of examples should be protected by a `\dontrun{}` to pass the `R CMD CHECK`. In this section we want to run automatically all these examples to check that everything is OK.

E.2 Stop list

This is the list of function that don't run for now and need to be fixed.

```
stoplist <- c("reverse.align", "extractseqs")
```

E.3 Don't run generator

This code chunk generates the `dontrun.rnw` file that is included there after. This file should be pre-existent, and two `Sweave()` passes are necessary.

```
outfile <- file(paste(pwd, "dontrun.rnw", sep = "/"), open = "w")
fex <- dir()
for (f in fex) {
    fctname <- substr(x = f, start = 1, stop = nchar(f) -
        2)
    if (fctname %in% stoplist)
        next
    lines <- readLines(f)
    dontrun <- lines[which(substring(lines, 1, 3) == "##D")]
    if (length(dontrun) == 0)
        next
    dontrun <- sapply(dontrun, function(x) substr(x, 5, nchar(x)))
    writeLines(paste("\\subsection{\\texttt{", fctname, "()}}",
        sep = ""), outfile)
    writeLines(paste("<<", fctname, ",fig=F,keep.source=T>>=",
        sep = ""), outfile)
```

```
    writeLines(dontrun, outfile)
    writeLines("@", outfile)
}
close(outfile)
```

E.3.1 GC()

```
# Too long for routine check
# This is a benchmark to compare the effect of various parameter
# setting on computation time
n <- 10
from <-10^4
to <- 10^5
size <- seq(from = from, to = to, length = n)
res <- data.frame(matrix(NA, nrow = n, ncol = 5))
colnames(res) <- c("size", "FF", "FT", "TF", "TT")
res[, "size"] <- size
for(i in seq_len(n)){
  myseq <- sample(x = s2c("acgtws"), size = size[i], replace = TRUE)
  res[i, "FF"] <- system.time(GC(myseq, forceToLower = FALSE, exact = FALSE))[3]
  res[i, "FT"] <- system.time(GC(myseq, forceToLower = FALSE, exact = TRUE))[3]
        res[i, "TF"] <- system.time(GC(myseq, forceToLower = TRUE, exact = FALSE))[3]
        res[i, "TT"] <- system.time(GC(myseq, forceToLower = TRUE, exact = TRUE))[3]
}
par(oma = c(0,0,2.5,0), mar = c(4,5,0,2) + 0.1, mfrow = c(2, 1))
plot(res$size, res$TT, las = 1,
xlab = "Sequence size [bp]",
ylim = c(0, max(res$TT)), xlim = c(0, max(res$size)), ylab = "")
title(ylab = "Observed time [s]", line = 4)
abline(lm(res$TT~res$size))
points(res$size, res$FT, col = "red")
abline(lm(res$FT~res$size), col = "red", lty = 3)
points(res$size, res$TF, pch = 2)
abline(lm(res$TF~res$size))
points(res$size, res$FF, pch = 2, col = "red")
abline(lm(res$FF~res$size), lty = 3, col = "red")
legend("topleft", inset = 0.01, legend = c("forceToLower = TRUE", "forceToLower = FALSE"), col = c("blac
legend("bottomright", inset = 0.01, legend = c("exact = TRUE", "exact = FALSE"),
pch = c(1,2))
mincpu <- lm(res$FF~res$size)$coef[2]
barplot(
c(lm(res$FF~res$size)$coef[2]/mincpu,
  lm(res$TF~res$size)$coef[2]/mincpu,
  lm(res$FT~res$size)$coef[2]/mincpu,
  lm(res$TT~res$size)$coef[2]/mincpu),
horiz = TRUE, xlab = "Increase of CPU time",
col = c("red", "black", "red", "black"),
names.arg = c("(F,F)", "(T,F)", "(F,T)", "(T,T)"), las = 1)
title(ylab = "forceToLower,exact", line = 4)
mtext("CPU time as function of options", outer = TRUE, line = 1, cex = 1.5)
```

E.3.2 SeqAcnucWeb()

```
# Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds et o=mitochondrion")
  stopifnot(is.SeqAcnucWeb(mylist$req[[1]]))
  closebank()
```

E.3.3 acnucopen()

```
  mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
    port = 5558, server = FALSE, blocking = TRUE)
  readLines(mysocket, n = 1) # OK acnuc socket started
```

```
[1] "OK acnuc socket started"
```

```
  acnucopen("emblTP", socket = mysocket) -> res
  expected <- c("EMBL", "14138095", "236401", "1186228", "8",
    "16", "40", "40", "20", "20", "40", "60", "63")
  stopifnot(all(unlist(res) == expected))
  tryalreadyopen <- try(acnucopen("emblTP", socket = mysocket))
  stopifnot(inherits(tryalreadyopen, "try-error"))
  # Need a fresh socket because acnucopen() close it if error:
  mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
    port = 5558, server = FALSE, blocking = TRUE)
  tryoff <-  try(acnucopen("off", socket = mysocket))
  stopifnot(inherits(tryoff, "try-error"))
  mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
    port = 5558, server = FALSE, blocking = TRUE)
  tryinexistent <-  try(acnucopen("tagadatagadatsointsoin", socket = mysocket))
  stopifnot(inherits(tryinexistent, "try-error"))
  mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
    port = 5558, server = FALSE, blocking = TRUE)
  trycloseunopened <- try(acnucclose(mysocket))
  stopifnot(inherits(trycloseunopened, "try-error"))
```

E.3.4 alllistranks()

```
 # Need internet connection
  choosebank("emblTP")
  query("tmp1", "sp=Borrelia burgdorferi", virtual = TRUE)
  query("tmp2", "sp=Borrelia burgdorferi", virtual = TRUE)
  query("tmp3", "sp=Borrelia burgdorferi", virtual = TRUE)
  (result <- alllistranks())
```

```
$count
[1] 3

$ranks
[1] 2 3 4
```

```
  stopifnot(result$count == 3)   # Three ACNUC lists
  stopifnot(result$ranks == 2:4) # Starting at rank 2
  #
  # Summay of current lists defined on the ACNUC server:
  #
  sapply(result$ranks, getliststate)
```

```
      [,1]   [,2]   [,3]
type  "SQ"   "SQ"   "SQ"
name  "TMP1" "TMP2" "TMP3"
count 1682   1682   1682
locus TRUE   TRUE   TRUE
```

```
  closebank()
```

E.3.5 autosocket()

```
  #Need internet connection
   choosebank("emblTP")
   autosocket()
```

```
                description                    class
"->pbil.univ-lyon1.fr:5558"                 "socket"
                       mode                     text
                        "a+"                   "text"
                     opened                  can read
                  "opened"                     "yes"
                 can write
                      "yes"
```

```
   closebank()
```

E.3.6 choosebank()

```
# Need internet connection
  # Show available databases:
  choosebank()
```

```
 [1] "genbank"      "embl"         "emblwgs"      "swissprot"
 [5] "ensembl"      "refseq"       "hobacnucl"    "hobacprot"
 [9] "hovernucl"    "hoverprot"    "hogennucl"    "hogenprot"
[13] "hogen4nucl"   "hogen4prot"   "homolensprot" "homolensnucl"
[17] "greview"      "HAMAPnucl"    "HAMAPprot"    "hoppsigen"
[21] "nurebnucl"    "nurebprot"    "taxobacgen"
```

```
  # Show frozen databases:
  choosebank(tag = "TP")
```

```
[1] "emblTP"      "swissprotTP" "hoverprotTP" "hovernuclTP" "emglib"
[6] "trypano"
```

```
  # Select a database:
  choosebank("emblTP", tag = "TP")
  # Do something with the database:
  myseq <- gfrag("LMFLCHR36", start = 1, length = 30)
  stopifnot(myseq == "cgcgtgctggcggcaatgaagcgttcgatg")
  # Close the database:
  closebank()
```

E.3.7 closebank()

```
# Need internet connection
   choosebank("emblTP")
   closebank()
```

E.3.8 countfreelists()

```
 # Need internet connection
  choosebank("emblTP")
  (rescountfreelists <- countfreelists())
```

```
$free
[1] 48

$annotlines
 [1] "ALL" "AC"  "PR"  "DT"  "KW"  "OS"  "OC"  "OG"  "RN"  "RC"  "RP"  "RX"
[13] "RG"  "RA"  "RT"  "RL"  "DR"  "CC"  "AH"  "AS"  "FH"  "FT"  "CO"  "SQ"
[25] "SEQ"
```

```
  stopifnot(all(rescountfreelists$annotlines ==
   c("ALL", "AC",  "PR",  "DT",  "KW",  "OS",  "OC",
   "OG",  "RN",  "RC",  "RP",  "RX", "RG",  "RA",  "RT",  "RL",  "DR",
   "CC",  "AH",  "AS",  "FH",  "FT",  "CO",  "SQ", "SEQ")))
  closebank()
```

E.3.9 countsubseqs()

```
  # Need internet connection
   choosebank("emblTP")
   query("mylist", "N=@", virtual = TRUE) # select all (seqs + subseqs)
   mylist$nelem   # 14138094 seqs + subseqs
```

```
[1] 14138094
```

```
   stopifnot(mylist$nelem == 14138094)
   css(glr("mylist")) # 1604500 subsequences only
```

```
[1] 1604500
```

```
   stopifnot(css(glr("mylist")) == 1604500)
   closebank()
```

E.3.10 crelistfromclientdata()

```
  # Need internet connection
  choosebank("emblTP")
  #
  # Example with a file that contains sequence names:
  #
  fileSQ <- system.file("sequences/bb.mne", package = "seqinr")
  crelistfromclientdata("listSQ", file = fileSQ, type = "SQ")
  sapply(listSQ$req, getName)
```

```
 [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"
```

```
  #
  # Example with a file that contains sequence accession numbers:
  #
  fileAC <- system.file("sequences/bb.acc", package = "seqinr")
  crelistfromclientdata("listAC", file = fileAC, type = "AC")
  sapply(listAC$req, getName)
```

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

```
  #
  # Example with a file that contains species names:
  #
  fileSP <- system.file("sequences/bb.sp", package = "seqinr")
  crelistfromclientdata("listSP", file = fileSP, type = "SP")
  sapply(listSP$req, getName)
```

```
 [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"
```

```
  #
  # Example with a file that contains keywords:
  #
  fileKW <- system.file("sequences/bb.kwd", package = "seqinr")
  crelistfromclientdata("listKW", file = fileKW, type = "KW")
  sapply(listKW$req, getName)
```

```
 [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"
```

```
  #
  # Summary of ACNUC lists:
  #
  sapply(alr()$rank, getliststate)
```

```
      [,1]     [,2]     [,3]     [,4]
type  "SQ"     "SQ"     "SP"     "KW"
name  "LISTSQ" "LISTAC" "LISTSP" "LISTKW"
count 20       20       20       20
locus FALSE    TRUE     TRUE     TRUE
```

```
  closebank()
```

E.3.11 draw.rearranged.oriloc()

```
r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
    g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
```

E.3.12 extract.breakpoints()

```
r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
    g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
```

E.3.13 getAnnot()

```
   # Need internet connection
   choosebank("emblTP")
   query("fc", "sp=felis catus et t=cds et O=mitochondrion et Y>2001 et no k=partial")
   # get the first 5 lines annotating the first sequence:
   annots <- getAnnot(fc$req[[1]], nbl = 5)
   cat(annots, sep = "\n")
```

```
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
```

```
   # or use the list method to get them all at once:
   annots <- getAnnot(fc$req, nbl = 5)
   cat(annots, sep = "\n")
```

```
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
FT   CDS             100..303
FT                   /db_xref="GOA:Q94NW9"
FT                   /db_xref="TrEMBL:Q94NW9"
FT                   /transl_table=2
FT                   /gene="ATPase8"
```

```
closebank()
```

E.3.14 getKeyword()

```
# Need internet connection
choosebank("emblTP")
query("fc", "sp=felis catus et t=cds et o=mitochondrion")
getKeyword(fc$req[[1]])
```

```
[1] "DIVISION ORG" "RELEASE 62"   "CYTOCHROME B" "SOURCE"
[5] "CDS"
```

```
# Should be:
# [1] "DIVISION ORG" "RELEASE 62"   "CYTOCHROME B" "SOURCE"       "CDS"
closebank()
```

E.3.15 getLength()

```
# Need internet connection
choosebank("emblTP")
query("fc", "sp=felis catus et t=cds et o=mitochondrion")
getLength(fc)
```

```
 [1] 1140 1140  300  402  402   27  204   95   27  204   95   27  204   95
[15]   27  204   95   27  204   95   27  204   95   27  204   95  316  402
[29]  957 1042 1545  684  204  681  784  347  297 1378 1821  528 1140 1140
[43]  345  319  237  250  345  237  249
```

```
closebank()
```

E.3.16 getLocation()

```
# Need internet connection
choosebank("emblTP")
query("fc", "sp=felis catus et t=cds et o=mitochondrion")
getLocation(fc$req[[5]])
```

```
[1]  27 428
```

```
closebank()
```

E.3.17 getName()

```
# Need internet connection
choosebank("emblTP")
query("fc", "sp=felis catus et t=cds et o=mitochondrion")
getName(fc)
```

```
 [1] "AB004237"        "AB004238"        "AF172359"        "FCA300702"
 [5] "FCA441328.CYTB"  "FSI409128.COII"  "FSI409128.PE2"   "FSI409128.PE3"
 [9] "FSI409129.COII"  "FSI409129.PE2"   "FSI409129.PE3"   "FSI409130.COII"
[13] "FSI409130.PE2"   "FSI409130.PE3"   "FSI409131.COII"  "FSI409131.PE2"
[17] "FSI409131.PE3"   "FSI409132.COII"  "FSI409132.PE2"   "FSI409132.PE3"
[21] "FSI409133.COII"  "FSI409133.PE2"   "FSI409133.PE3"   "FSI409134.COII"
[25] "FSI409134.PE2"   "FSI409134.PE3"   "MI1290634.PE1"   "MIFCCBD"
[29] "MIFCCU207.ND1"   "MIFCCU207.ND2"   "MIFCCU207.COI"   "MIFCCU207.COII"
[33] "MIFCCU207.PE5"   "MIFCCU207.PE6"   "MIFCCU207.COIII" "MIFCCU207.ND3"
[37] "MIFCCU207.ND4L"  "MIFCCU207.ND4"   "MIFCCU207.ND5"   "MIFCCU207.ND6"
[41] "MIFCCU207.CYTB"  "MIFDCYTB"        "S75096"          "S75098"
[45] "S75099.COI"      "S75101"          "S75328"          "S75331.COII"
[49] "S75332.COI"
```

```
closebank()
```

E.3.18 getSequence()

```
# Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getSequence(fc$req[[1]])
```

```
  [1] "a" "t" "g" "a" "c" "c" "a" "a" "c" "a" "t" "t" "c" "g" "a" "a" "a"
 [18] "a" "t" "c" "a" "c" "a" "c" "c" "c" "c" "t" "t" "a" "c" "c" "a"
 [35] "a" "a" "a" "t" "t" "a" "t" "t" "a" "a" "t" "c" "a" "c" "t" "c" "a"
 [52] "t" "t" "c" "a" "t" "c" "g" "a" "c" "c" "t" "a" "c" "c" "t" "g" "c"
 [69] "c" "c" "c" "a" "t" "c" "t" "a" "a" "c" "a" "t" "c" "t" "c" "a" "g"
 [86] "c" "a" "t" "g" "a" "t" "g" "a" "a" "a" "c" "t" "t" "c" "g" "g" "c"
[103] "t" "c" "c" "c" "t" "t" "c" "t" "a" "g" "g" "a" "g" "t" "c" "t" "g"
[120] "c" "c" "t" "a" "a" "t" "c" "t" "t" "a" "c" "a" "a" "a" "t" "c" "c"
[137] "t" "c" "a" "c" "c" "g" "g" "c" "c" "t" "c" "t" "t" "t" "t" "g"
[154] "g" "c" "c" "a" "t" "a" "c" "a" "c" "t" "a" "c" "a" "c" "a" "t" "c"
[171] "a" "g" "a" "c" "a" "c" "a" "a" "c" "a" "a" "c" "c" "g" "c" "c" "t"
[188] "t" "t" "t" "c" "a" "t" "c" "a" "g" "t" "t" "a" "c" "c" "c" "a" "c"
[205] "a" "t" "c" "t" "g" "t" "c" "g" "c" "g" "a" "c" "g" "t" "t" "a" "a"
[222] "t" "t" "a" "t" "g" "g" "c" "t" "g" "a" "a" "t" "c" "a" "t" "c" "c"
[239] "g" "a" "t" "a" "t" "t" "t" "a" "c" "a" "c" "g" "c" "c" "a" "a" "c"
[256] "g" "g" "a" "g" "c" "t" "t" "c" "t" "a" "t" "a" "t" "t" "c" "t" "t"
[273] "t" "a" "t" "c" "t" "g" "c" "c" "t" "g" "t" "a" "c" "a" "t" "a" "c"
[290] "a" "t" "g" "t" "a" "g" "g" "a" "c" "g" "g" "g" "g" "a" "a" "t" "a"
[307] "t" "a" "c" "t" "a" "c" "g" "g" "c" "t" "c" "c" "t" "a" "c" "a" "c"
[324] "c" "t" "t" "c" "t" "c" "a" "g" "a" "g" "a" "c" "a" "t" "g" "a" "a"
[341] "a" "c" "a" "t" "t" "g" "g" "a" "a" "t" "c" "a" "t" "a" "c" "t" "a"
[358] "t" "t" "a" "t" "t" "t" "a" "c" "a" "g" "t" "c" "a" "t" "a" "g" "c"
[375] "c" "a" "c" "a" "g" "c" "t" "t" "t" "t" "a" "t" "g" "g" "g" "a" "t"
```

```
 [392] "a" "c" "g" "t" "c" "c" "t" "a" "c" "c" "a" "t" "g" "a" "g" "g" "c"
 [409] "c" "a" "a" "a" "t" "g" "t" "c" "c" "t" "t" "c" "t" "g" "a" "g" "g"
 [426] "a" "g" "c" "a" "a" "c" "c" "g" "t" "a" "a" "t" "c" "a" "c" "t" "a"
 [443] "a" "c" "c" "t" "c" "c" "t" "g" "t" "c" "a" "g" "c" "a" "a" "t" "t"
 [460] "c" "c" "a" "t" "a" "c" "a" "t" "c" "g" "g" "g" "a" "c" "t" "g" "a"
 [477] "a" "c" "t" "a" "g" "t" "a" "g" "a" "a" "t" "g" "a" "a" "t" "c" "t"
 [494] "g" "a" "g" "g" "g" "g" "g" "c" "t" "t" "c" "t" "c" "a" "g" "t" "a"
 [511] "g" "a" "c" "a" "a" "a" "g" "c" "c" "a" "c" "c" "c" "t" "a" "a" "c"
 [528] "a" "c" "g" "a" "t" "t" "c" "t" "t" "t" "g" "c" "t" "t" "t" "c" "c"
 [545] "a" "c" "t" "t" "c" "a" "t" "t" "c" "t" "t" "c" "c" "a" "t" "t" "c"
 [562] "a" "t" "t" "a" "t" "c" "t" "c" "a" "g" "c" "c" "t" "t" "a" "g" "c"
 [579] "a" "g" "c" "a" "g" "t" "a" "c" "a" "c" "c" "t" "c" "t" "t" "a" "t"
 [596] "t" "c" "c" "t" "t" "c" "a" "t" "g" "a" "a" "a" "c" "a" "g" "g" "a"
 [613] "t" "c" "t" "a" "a" "c" "a" "a" "c" "c" "c" "c" "t" "c" "a" "g" "g"
 [630] "a" "a" "t" "t" "a" "c" "a" "t" "c" "c" "g" "a" "t" "t" "c" "a" "g"
 [647] "a" "c" "a" "a" "a" "a" "t" "c" "c" "c" "a" "t" "t" "c" "c" "a" "c"
 [664] "c" "c" "a" "t" "a" "c" "t" "a" "t" "a" "c" "a" "a" "t" "c" "a" "a"
 [681] "a" "g" "a" "c" "a" "t" "c" "c" "t" "a" "g" "g" "t" "c" "t" "t" "c"
 [698] "t" "a" "g" "t" "a" "c" "t" "a" "g" "t" "t" "t" "t" "a" "a" "c" "a"
 [715] "c" "t" "c" "a" "t" "a" "c" "t" "a" "c" "t" "c" "g" "t" "c" "c" "t"
 [732] "a" "t" "t" "t" "t" "c" "a" "c" "c" "a" "g" "a" "c" "c" "t" "g" "c"
 [749] "t" "a" "g" "g" "a" "g" "a" "c" "c" "c" "a" "g" "a" "c" "a" "a" "c"
 [766] "t" "a" "c" "a" "t" "c" "c" "c" "a" "g" "c" "c" "a" "a" "c" "c" "c"
 [783] "t" "t" "t" "a" "a" "a" "t" "a" "c" "c" "c" "c" "t" "c" "c" "c" "c"
 [800] "a" "t" "a" "t" "t" "a" "a" "a" "c" "c" "t" "g" "a" "a" "t" "g" "a"
 [817] "t" "a" "c" "t" "t" "c" "c" "t" "a" "t" "t" "c" "g" "c" "a" "t" "a"
 [834] "c" "g" "c" "a" "a" "t" "t" "c" "t" "c" "c" "g" "a" "t" "c" "c" "a"
 [851] "t" "c" "c" "c" "c" "a" "a" "c" "a" "a" "a" "c" "t" "a" "g" "g" "g"
 [868] "g" "g" "a" "g" "t" "c" "c" "t" "a" "g" "c" "c" "c" "t" "a" "g" "t"
 [885] "a" "c" "t" "c" "t" "c" "c" "a" "t" "c" "c" "t" "a" "g" "t" "a" "c"
 [902] "t" "a" "g" "c" "a" "a" "t" "c" "a" "t" "t" "c" "c" "a" "a" "t" "c"
 [919] "c" "t" "c" "c" "a" "c" "a" "c" "c" "t" "c" "c" "a" "a" "a" "c" "a"
 [936] "a" "c" "g" "a" "g" "g" "a" "a" "t" "a" "a" "t" "g" "t" "t" "t" "c"
 [953] "g" "a" "c" "c" "a" "c" "t" "a" "a" "g" "c" "c" "a" "a" "t" "g" "t"
 [970] "c" "t" "a" "t" "t" "c" "t" "g" "a" "c" "t" "c" "c" "t" "a" "g" "t"
 [987] "a" "g" "c" "g" "g" "a" "t" "c" "t" "c" "c" "t" "a" "a" "c" "c" "c"
[1004] "t" "a" "a" "c" "a" "t" "g" "a" "a" "t" "c" "g" "g" "t" "g" "g" "c"
[1021] "c" "a" "a" "c" "c" "t" "g" "t" "a" "g" "a" "a" "c" "a" "t" "c" "c"
[1038] "a" "t" "t" "c" "a" "t" "c" "a" "c" "c" "a" "t" "c" "g" "g" "c" "c"
[1055] "a" "a" "c" "t" "a" "g" "c" "c" "t" "c" "c" "a" "t" "c" "c" "t" "a"
[1072] "t" "a" "t" "t" "t" "c" "t" "c" "a" "a" "c" "c" "c" "t" "c" "c" "t"
[1089] "a" "a" "t" "c" "c" "t" "a" "a" "t" "a" "c" "c" "c" "a" "t" "c" "t"
[1106] "c" "a" "g" "g" "c" "a" "t" "t" "a" "t" "t" "g" "a" "a" "a" "a" "c"
[1123] "c" "g" "c" "c" "t" "a" "c" "t" "c" "a" "a" "a" "t" "g" "a" "a" "g"
[1140] "a"
```

```
getSequence(fc$req[[1]], as.string = TRUE)
```

```
[1] "atgaccaacattcgaaaatcacacccccttaccaaaattattaatcactcattcatcgacctacctgccccatctaacatctcagcatgatgaaacttcggctcccttctag
```

```
closebank()
```

E.3.19 getTrans()

```
# Need internet connection.
# Translation of the following EMBL entry:
#
# FT   CDS             join(complement(153944..154157),complement(153727..153866),
# FT                   complement(152185..153037),138523..138735,138795..138955)
# FT                   /codon_start=1
choosebank("emblTP")
query("trans", "N=AE003734.PE35")
getTrans(trans$req[[1]])
```

```
  [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" "S" "V" "C" "S"
 [55] "P" "F" "F" "R" "K" "M" "F" "T" "Q" "M" "P" "S" "N" "T" "H" "A" "I" "V"
 [73] "F" "L" "N" "N" "V" "S" "H" "S" "A" "L" "K" "D" "L" "I" "Q" "F" "M" "Y"
 [91] "C" "G" "E" "V" "N" "V" "K" "Q" "D" "A" "L" "P" "A" "F" "I" "S" "T" "A"
[109] "E" "S" "L" "Q" "I" "K" "G" "L" "T" "D" "N" "D" "P" "A" "P" "Q" "P" "P"
[127] "Q" "E" "S" "S" "P" "P" "P" "A" "A" "P" "H" "V" "Q" "Q" "Q" "Q" "I" "P"
[145] "A" "Q" "R" "V" "Q" "R" "Q" "Q" "P" "R" "A" "S" "A" "R" "Y" "K" "I" "E"
[163] "T" "V" "D" "D" "G" "L" "G" "D" "E" "K" "Q" "S" "T" "T" "Q" "I" "V" "I"
[181] "Q" "T" "T" "A" "A" "P" "Q" "A" "T" "I" "V" "Q" "Q" "Q" "Q" "P" "Q" "Q"
[199] "A" "A" "Q" "Q" "I" "Q" "S" "Q" "Q" "L" "Q" "T" "G" "T" "T" "T" "T" "A"
[217] "T" "L" "V" "S" "T" "N" "K" "R" "S" "A" "Q" "R" "S" "S" "L" "T" "P" "A"
[235] "S" "S" "S" "A" "G" "V" "K" "R" "S" "K" "T" "S" "T" "S" "A" "N" "V" "M"
[253] "D" "P" "L" "D" "S" "T" "T" "E" "T" "G" "A" "T" "T" "T" "A" "Q" "L" "V"
[271] "P" "Q" "Q" "I" "T" "V" "Q" "T" "S" "V" "V" "S" "A" "A" "E" "A" "K" "L"
[289] "H" "Q" "Q" "S" "P" "Q" "Q" "V" "R" "Q" "E" "E" "A" "E" "Y" "I" "D" "L"
[307] "P" "M" "E" "L" "P" "T" "K" "S" "E" "P" "D" "Y" "S" "E" "D" "H" "G" "D"
[325] "A" "A" "G" "D" "A" "E" "G" "T" "Y" "V" "E" "D" "D" "T" "Y" "G" "D" "M"
[343] "R" "Y" "D" "D" "S" "Y" "F" "T" "E" "N" "E" "D" "A" "G" "N" "Q" "T" "A"
[361] "A" "N" "T" "S" "G" "G" "G" "V" "T" "A" "T" "T" "S" "K" "A" "V" "V" "K"
[379] "Q" "Q" "S" "Q" "N" "Y" "S" "E" "S" "S" "F" "V" "D" "T" "S" "G" "D" "Q"
[397] "G" "N" "T" "E" "A" "Q" "V" "T" "Q" "H" "V" "R" "N" "C" "G" "P" "Q" "M"
[415] "F" "L" "I" "S" "R" "K" "G" "G" "T" "L" "L" "T" "I" "N" "N" "F" "V" "Y"
[433] "R" "S" "N" "L" "K" "F" "F" "G" "K" "S" "N" "N" "I" "L" "Y" "W" "E" "C"
[451] "V" "Q" "N" "R" "S" "V" "K" "C" "R" "S" "R" "L" "K" "T" "I" "G" "D" "D"
[469] "L" "Y" "V" "T" "N" "D" "V" "H" "N" "H" "M" "G" "D" "N" "K" "R" "I" "E"
[487] "A" "A" "K" "A" "A" "G" "M" "L" "I" "H" "K" "K" "L" "S" "S" "L" "T" "A"
[505] "A" "D" "K" "I" "Q" "G" "S" "W" "K" "M" "D" "T" "E" "G" "N" "P" "D" "H"
[523] "L" "P" "K" "M" "*"
```

E.3.20 getType()

```
# Need internet connection
  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
```

E.3.21 getlistrank()

```
  # Need internet connection
  choosebank("emblTP")
  query("MyListName", "sp=Borrelia burgdorferi", virtual = TRUE)
  (result <- getlistrank("MyListName"))
```

```
[1] 2
```

```
  stopifnot(result == 2)
  closebank()
```

E.3.22 getliststate()

```
  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
  getliststate(glr("mylist")) # SQ, MYLIST, 603, FALSE
```

```
$type
[1] "SQ"

$name
[1] "MYLIST"

$count
[1] 603

$locus
[1] FALSE
```

```
   gln(glr("mylist")) # MYLIST (upper case letters on server)
```

```
[1] "MYLIST"
```

```
   closebank()
```

E.3.23 gfrag()

```
 # Need internet connection
   choosebank("emblTP")
   gfrag("LMFLCHR36", start = 1, length = 3529852) -> myseq
   stopifnot(nchar(myseq) == 3529852)
   closebank()
```

E.3.24 ghelp()

```
   ### Need internet connection
   choosebank("emblTP")
   ghelp()
```

```
      ---- General Information on ACNUC nucleic acid data base ----
HELP:
   A detailed explanation of purpose and usage of each command is obtained
by typing the command name and requesting help when the dialog suggests it.
SEQUENCES AND SUBSEQUENCES:
   In addition to sequences as published in research articles, ACNUC contains
subsequences which are sequence segments with specific coding function (e.g.
protein, tRNA, rRNA genes...). Sequence type distinguishes parent from sub-
sequences: parent sequences have ID type, subsequences have a type that
indicates their function (CDS, TRNA, RRNA,...). Most subsequence names derive
from the parent sequence's name by addition of suffixes .PEn, .TRn, .RRn, .SNn
.RNn for CDS, TRNA, RRNA, snRNA or misc_RNA-typed subsequences, respectively.
When the gene name is known, it is used as a suffix in the corresponding
subsequence name.
SEQUENCE LISTS:
   This program deals with sequence lists which group sequences selected from
the data base using one or more selection criteria (see SELECT help). Many
sequence lists can be handled simultaneously by the program and previous lists
can be used to define new ones.
   Typical use of program is:
 - SPECIES command to know which species names are to be used in selection.
 - KEYWORDS command to know which keywords are to be used in selection.
 - SELECT command to select sequences from data base combining various
criteria. This command produces the list of sequences that fit the criteria.
 - SHORT command to obtain a brief description of selected sequences
or - INFO command to get more detailed information.
 - EXTRACT command to copy selected sequences to a user file.
LIST NAMES:
   Lists are created by commands SELECT or FIND. They are given automatically a
name (LIST1, LIST2,...) by the program, unless the user enters his own list
name by appending /l=my_list_name to the command name at the "Command?" prompt.
Most commands operate either on a sequence list or on an individual sequence.
Reply to question "List, sequence, or accession #? [default=...]" with
<RETURN> to access the default (list of) sequence(s) or with any list name,
sequence name, or accession number.
FILE OUTPUT:
```

```
If /lpt is appended to command name at the "Command?" prompt, the output of
commands SPECIES, KEYWORDS, INFO, SHORT, NAMES, CODES, BASES goes to a file
named `query.out'.
CODED NAMES:
   Coded names are to be used when specifying species, keywords, journals,
sequence types, organelles, molecules. Specific commands
(SPECIES, KEYWORDS, CODES) allow you to find these names easily.
REFERENCES:
   To find a sequence from a bibliographical reference use the selection
criterion "R=reference-code" of SELECT command. Build the reference code as
follows (journal names are given by CODES command):
  journal_name/volume/first_page         for journal articles
  book/year/name_of_1st_author           for books
  thesis/year/name_of_1st_author         for thesis
  patent/patent_number                   for patented sequences
  unpubl/year/name_of_1st_author         for unpublished sequences
Example: nar/8/2173 stands for Nucleic Acids Research 8:2173-2192 (1980).
```

```
ghelp("SELECT")
```

```
     In addition to functions described in the help for the simple usage of
command SELECT, other selection criteria and operations between lists exist.
Specifically, it is also possible to build lists of species and lists of
keywords for further retrieval capabilities.

Criteria      Resulting selection
FK=file name  List of keywords taken from a file (which may have been created
              by a SAVE command).
FS=file name  List of species taken from a file (which may have been created
              by a SAVE command).

Operation     Result
ME list       Replaces subsequences in list by sequences from which they
              are extracted (equivalent to option 4 of command MODIFY).
FI list       Sequences in list plus all of their subsequences (equivalent to
              option 5 of command MODIFY).
PS list       Produces the list of species names attached to sequences in list.
PK list       Produces the list of keyword names attached to sequences in list.
UN list       If applied to a species list, produces the list of sequences from
              species in the list; if applied to a keyword list, produces the
              list of sequences attached to keywords in list.
SD spec-list  Applied to a list of species, produces the list of all descendants
              from them in the species tree. The list itself can easily be
              created by command FIND.
KD keyw-list  Applied to a list of keywords, produces the list of all
              descendants from them in the keywords tree. The list itself can
              easily be created by command FIND.

Operators PS, PK, and UN allow to solve the problem "find all genes
simultaneously sequenced in a given series of species".
First, build the lists of sequences from each of these species. Next project
each of these lists to attached keywords by applying operator PK. Then compute
the list of keywords in common by combining the keyword lists with operator ET.
Then, remove from this list of common keywords, those which are
uncharacteristic (e.g. partial) by employing command MODIFY. Finally, produce
the lists of sequences attached to common keywords from each species by
applying operator UN combined with initial species-based sequence lists.
Species and keyword lists can be listed with command NAMES and saved with SAVE.
```

```
# To get info about current database:
ghelp("CONT")
```

```
            ****     ACNUC Data Base Content      ****
             EMBL Library Release 78 WITHOUT ESTs  (March 2004)
27,571,397,913 bases; 12,533,594 sequences; 1,604,500 subseqs; 339,186 refers.
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I
```

E.3.25 isenum()

```
### Need internet connection
choosebank("emblTP")
isenum("LMFLCHR36")
```

```
$number
[1] 13682678

$length
[1] 3529852

$frame
[1] 0

$gencode
[1] 0

$ncbigc
[1] 1

$otheraccessmatches
[1] FALSE
```

```
isn("LMFLCHR36")
```

```
[1] 13682678
```

```
stopifnot(isn("LMFLCHR36") == 13682678)
# Example with CDS:
isenum("AB004237")
```

```
$number
[1] 66351

$length
[1] 1140

$frame
[1] 0

$gencode
[1] 2

$ncbigc
[1] 2

$otheraccessmatches
[1] FALSE
```

E.3.26 knowndbs()

```
### Need internet connection
choosebank("emblTP")
kdb()
```

```
          bank status
1       genbank     on
2          embl     on
3       emblwgs     on
4     swissprot     on
5       ensembl     on
6        refseq     on
7     hobacnucl     on
8     hobacprot     on
9     hovernucl     on
10    hoverprot     on
11    hogennucl     on
12    hogenprot     on
```

```
13   hogen4nucl     on
14   hogen4prot     on
15 homolensprot     on
16 homolensnucl     on
17      greview     on
18    HAMAPnucl     on
19    HAMAPprot     on
20    hoppsigen     on
21    nurebnucl     on
22    nurebprot     on
23   taxobacgen     on
                                                                        info
1                  GenBank Rel. 162 (15 October 2007) Last Updated: Nov 30, 2007
2            EMBL Library Release 92 (September 2007) Last Updated: Nov 30, 2007
3               EMBL Whole Genome Shotgun sequences Release 92  (September 2007)
4        UniProt Rel. 12 (SWISS-PROT 54 + TrEMBL 37): Last Updated: Nov 19, 2007
5                                                  Ensembl databases release 41
6                        RefSeq 15.0 (1 January 2006) Last Updated: Jan 23, 2006
7                        HOBACGEN - genomic data - Release 10 (February 12 2002)
8                        HOBACGEN - protein data - Release 10 (February 12 2002)
9  HOVERGEN - genomic data - Release 48 (May 24 2007) Last Updated: May 24, 2007
10 HOVERGEN - protein data - Release 48 (May 24 2007) Last Updated: May 24, 2007
11  HOGENOM - genomic data - Release 03 (Oct 14 2005) Last Updated: Nov  7, 2005
12  HOGENOM - protein data - Release 03 (Oct 14 2005) Last Updated: Mar 10, 2006
13 HOGENOM - genomic data - Release 04 (Sept 18,2007) Last Updated: Oct  2, 2007
14 HOGENOM - protein data - Release 04 (Sept 18,2007) Last Updated: Nov  6, 2007
15         HOMOLENS 3 - Homologous genes from Ensembl Last Updated: Jan 19, 2007
16        HOMOLENS 3 Homologous genes from Ensembl 41 Last Updated: Jan 19, 2007
17              EBI Genome Reviews. Acnuc Release 7. Last Updated: Feb 26, 2007
18             HAMAP - Acnuc Release - Nucleotides - Last Updated: Jun  5, 2007
19                  HAMAP - Acnuc Release -Proteins- Last Updated: Jun  5, 2007
20                                                                   Hoppsigen
21                  Nurebase 4.0 (26 September 2003) Last Updated: NOV 27, 2003
22                  Nurebase 4.0 (26 September 2003) Last Updated: NOV 27, 2003
23                                        TaxoBacGen Rel. 7 (September 2005)
```

```
closebank()
```

E.3.27 modifylist()

```
choosebank("emblTP")
query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
mylist$nelem # 603 sequences
```

```
[1] 603
```

```
stopifnot(mylist$nelem == 603)
# select sequences with at least 1000 bp:
modifylist("mylist", operation = ">1000", virtual = TRUE)
mylist$nelem # now, only 132 sequences
```

```
[1] 132
```

```
stopifnot(mylist$nelem == 132)
# scan for "felis" in annotations:
modifylist("mylist", op = "felis", type = "scan", virtual = TRUE)
mylist$nelem # now, only 33 sequences
```

```
[1] 33
```

```
stopifnot(mylist$nelem == 33)
# modify by date:
modifylist("mylist", op = "> 1/jul/2001", type = "date", virtual = TRUE)
mylist$nelem # now, only 15 sequences
```

```
[1] 15
```

```
   stopifnot(mylist$nelem == 15)
   # Summary of current ACNUC lists, one list called MYLIST on sever:
   sapply(alr()$rank, getliststate)
```

```
      [,1]
type  "SQ"
name  "MYLIST"
count 15
locus FALSE
```

```
   closebank()
```

E.3.28 oriloc()

```
 #
 # A little bit too long for routine checks because oriloc() is already
 # called in draw.oriloc.Rd documentation file. Try example(draw.oriloc)
 # instead, or copy/paste the following code:
 #
 out <- oriloc()
 plot(out$st, out$sk, type = "l", xlab = "Map position in Kb",
     ylab = "Cumulated composite skew",
     main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome))
```

E.3.29 plot.SeqAcnucWeb()

```
   ### Need internet connection
   choosebank("hovernucl")
   query("list", "AC=AB000425")
   plot(list$req[[1]])
```

E.3.30 prettyseq()

```
   ### Need internet connection
   choosebank("emblTP")
   prettyseq(111)
```

```
Name: A00165           Length:108
Genetic code used: NUG=AUN=M when initiation codon

                 10         20         30         40         50         60
          Q  Y  C   G  N  L  S   T  C  M   L  G  T   Y  T  Q  D   F  N  K
         cagtactgcg gtaatctgag tacttgcatg ctgggcacat acacgcagga cttcaacaag
         >A00165

                 70         80         90        100        110
          F  H  T   F  P  Q  T   A  I  G   V  G  A   P  G  *
         tttcacacgt tcccccaaac tgcaattggg gttggagcac ctggttga
                                                  A00165<
```

E.3.31 print.SeqAcnucWeb()

```
   ### Need internet connection
   choosebank("emblTP")
   query("mylist", "sp=felis catus")
   mylist$req[[1]]
```

```
    name   length    frame   ncbicg
"A06937"     "34"      "0"      "1"
```

E.3.32 query()

```
  # Need internet connection
  choosebank("genbank")
  query("bb", "sp=Borrelia burgdorferi")
  # To get the names of the 4 first sequences:
  sapply(bb$req[1:4], getName)
```

```
[1] "A04009" "A22442" "A24006" "A24008"
```

```
  # To get the 4 first sequences:
  sapply(bb$req[1:4], getSequence, as.string = TRUE)
```

```
[1] "aagcttaattagaaccaaacttaattaaaaccaaacttaattgaagttattatcattttatttttttttcaattttctatttgttatttgttaatcttata
[2] "atgaaaaaatatttattgggaataggtctaatattagccttaatagcatgtaagcaaaatgttagcagccttgacgagaaaaacagcgtttcagtagatt
[3] "atgaaaaaatatttattgggaataggtctaatattagccttaatagcatgtaagcaaaatgttagcagccttgatgaaaaaaatagcgtttcagtagatt
[4] "atgaaaaaatatttattgggaataggtctaatattagccttaatagcatgtaagcaaaatgttagcagccttgacgagaaaaacagcgtttcagtagatg
```

E.3.33 readfirstrec()

```
 # Need internet connection
   choosebank("genbank")
   allowedtype <- readfirstrec()
   sapply(allowedtype, function(x) readfirstrec(type = x))
```

```
    AUT       BIB       ACC       SMJ       SUB       LOC       KEY
 165152    484468  79360574      4932  83935189  79794229   7401652
   SPEC      SHRT       LNG       EXT       TXT
 503804 714581316  22543462   6989121    405605
```

E.3.34 rearranged.oriloc()

```
 r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
     g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
```

E.3.35 residuecount()

```
   ### Need internet connection
   choosebank("emblTP")
   query("mylist", "t=CDS", virtual = TRUE)
   stopifnot(residuecount(glr("mylist")) == 1611439240)
   stopifnot(is.na(residuecount(glr("unknowlist")))) # A warning is issued
```

E.3.36 savelist()

```
   ### Need internet connection
   choosebank("emblTP")
   query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
   savelist(glr("mylist"))
```

```
603 sequence mnemonics written into file: MYLIST.mne
```

```
   # 603 sequence mnemonics written into file: MYLIST.mne
   savelist(glr("mylist"), type = "A")
```

```
603 sequence accession numbers written into file: MYLIST.acc
```

E.3.37 setlistname()

```
### Need internet connection
choosebank("emblTP")
query("mylist", "sp=felis catus et t=CDS", virtual = TRUE)
# Change list name on server:
setlistname(lrank = glr("mylist"), name = "feliscatus") # 0, OK.
```

```
[1] 0
```

```
glr("mylist") # 0, list doesn't exist no more.
```

```
[1] 0
```

```
glr("feliscatus") # 2, this list exists.
```

```
[1] 2
```

E.3.38 translate()

```
## Need internet connection.
## Translation of the following EMBL entry:
##
## 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"
choosebank("emblTP")
query("trans", "N=AE003734.PE35")
getTrans(trans$req[[1]])
```

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

```
[343] "R" "Y" "D" "D" "S" "Y" "F" "T" "E" "N" "E" "D" "A" "G" "N" "Q" "T" "A"
[361] "A" "N" "T" "S" "G" "G" "G" "V" "T" "A" "T" "T" "S" "K" "A" "V" "V" "K"
[379] "Q" "Q" "S" "Q" "N" "Y" "S" "E" "S" "S" "F" "V" "D" "T" "S" "G" "D" "Q"
[397] "G" "N" "T" "E" "A" "Q" "V" "T" "Q" "H" "V" "R" "N" "C" "G" "P" "Q" "M"
[415] "F" "L" "I" "S" "R" "K" "G" "G" "T" "L" "L" "T" "I" "N" "N" "F" "V" "Y"
[433] "R" "S" "N" "L" "K" "F" "F" "G" "K" "S" "N" "N" "I" "L" "Y" "W" "E" "C"
[451] "V" "Q" "N" "R" "S" "V" "K" "C" "R" "S" "R" "L" "K" "T" "I" "G" "D" "D"
[469] "L" "Y" "V" "T" "N" "D" "V" "H" "N" "H" "M" "G" "D" "N" "K" "R" "I" "E"
[487] "A" "A" "K" "A" "A" "G" "M" "L" "I" "H" "K" "K" "L" "S" "S" "L" "T" "A"
[505] "A" "D" "K" "I" "Q" "G" "S" "W" "K" "M" "D" "T" "E" "G" "N" "P" "D" "H"
[523] "L" "P" "K" "M" "*"
```

```
setwd(pwd)
```

Session Informations

This part was compiled under the following R environment:

- R version 2.6.1 (2007-11-26), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-38, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.17-2, nlme 3.1-86, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.1, rcompgen 0.1-17

There were two compilation steps:

- R compilation time was: Sun Dec 2 17:49:23 2007
- LaTeX compilation time was: December 2, 2007

APPENDIX F

Informations about databases available at pbil

Lobry, J.R.

F.1 Introduction

This section was compiled on December 2, 2007. The list of available database at pbil (`http://pbil.univ-lyon1.fr/`) was:

```
bankDefault <- choosebank()
bankTP <- choosebank(tagbank = "TP")
bankDEV <- choosebank(tagbank = "DEV")
(banknames <- c(bankDefault, bankTP, bankDEV))
```

```
 [1] "genbank"      "embl"         "emblwgs"      "swissprot"
 [5] "ensembl"      "refseq"       "hobacnucl"    "hobacprot"
 [9] "hovernucl"    "hoverprot"    "hogennucl"    "hogenprot"
[13] "hogen4nucl"   "hogen4prot"   "homolensprot" "homolensnucl"
[17] "greview"      "HAMAPnucl"    "HAMAPprot"    "hoppsigen"
[21] "nurebnucl"    "nurebprot"    "taxobacgen"   "emblTP"
[25] "swissprotTP"  "hoverprotTP"  "hovernuclTP"  "emglib"
[29] "trypano"      "ensembl41"    "ensembl34"    "genomicro1"
[33] "genomicro2"   "microbes"     "macaca"       "canis"
[37] "mouse38"      "homo46"
```

This LaTeX file was automatically generated by the following R code:

```
for (b in banknames) {
    cat(paste("\\section{", b, "}"), sep = "\n")
    openTry <- try(choosebank(b))
    if (inherits(openTry, "try-error")) {
        cat("There was a problem while trying to open this bank.\n")
        next
    }
    bankdetails <- sapply(banknameSocket$details, stresc,
        USE.NAMES = FALSE)
    cat("\\subsection{Bank details}", sep = "\n")
    cat(bankdetails, sep = "\\\\\n")
    cat("\n")
    cat("\\subsection{Type names}", sep = "\n")
```

```
    types <- getType()
    if (is.null(nrow(types))) {
        cat("There are no subsequence type in this database",
            sep = "\n")
    }
    else {
        cat("\\noindent\\begin{tabular}{llr}", sep = "\n")
        cat("\\hline \\hline", sep = "\n")
        cat("name & description & count \\\\", sep = "\n")
        cat("\\hline", sep = "\n")
        sumnelem <- 0
        for (i in 1:nrow(types)) {
            querytry <- try(query("mylist", paste("T=", types[i,
                "sname"]), virtual = TRUE))
            if (inherits(querytry, "try-error")) {
                nelem <- 0
            }
            else {
                nelem <- mylist$nelem
            }
            sumnelem <- sumnelem + nelem
            cat(paste(stresc(types[i, "sname"]), " & ", stresc(types[i,
                "libel"]), " & ", formatC(nelem, big.mark = ",",
                format = "d"), "\\\\"), sep = "\n")
        }
        cat("\\hline", sep = "\n")
        cat(paste(" & Total: &", formatC(sumnelem, big.mark = ",",
            format = "d"), "\\\\"), sep = "\n")
        cat("\\hline \\hline", sep = "\n")
        cat("\\end{tabular}", sep = "\n")
        cat("\n")
    }
    closebank()
}
```

F.2 genbank

F.2.1 Bank details

**** ACNUC Data Base Content ****
GenBank Rel. 162 (15 October 2007) Last Updated: Nov 7, 2007
81,938,639,153 bases; 77,983,785 sequences; 4,560,520 subseqs; 482,575 refers.
Software by M. Gouy, Lab. Biometrie et Biologie Evolutive, Universite Lyon I

F.2.2 Type names

name	description	count
CDS	.PE protein coding region	4,884,506
LOCUS	sequenced DNA fragment	75,664,837
MISC_RNA	.RN other structural RNA coding region	525,754
RRNA	.RR mature ribosomal RNA	1,160,620
SCRNA	.SC small cytoplasmic RNA	161
SNRNA	.SN small nuclear RNA	452
TRNA	.TR mature transfer RNA	307,975
	Total:	82,544,305

F.3 embl

F.3.1 Bank details

**** ACNUC Data Base Content ****
EMBL Library Release 92 (September 2007) Last Updated: Nov 7, 2007
99,074,185,873 bases; 80,771,301 sequences; 11,262,120 subseqs; 478,786 refers.
Software by M. Gouy, Laboratoire de biometrie, Universite Lyon I

F.3.2 Type names

name	description	count
CDS	.PE protein coding region	11,630,177
ID	Locus entry	78,403,512
MISC_RNA	.RN other structural RNA coding region	523,679
RRNA	.RR Ribosomal RNA coding gene	1,162,292
SCRNA	.SC small cytoplasmic RNA	5
SNRNA	.SN small nuclear RNA	79
TRNA	.TR Transfer RNA coding gene	313,677
	Total:	92,033,421

F.4 emblwgs

F.4.1 Bank details

**** ACNUC Data Base Content ****
EMBL Whole Genome Shotgun sequences Release 92 (September 2007)
102,118,953,790 bases; 25,405,475 sequences; 1,087,882 subseqs; 377 refers.
Retrieval software by M. Gouy, Biometrie et Biologie Evolutive, Univ Lyon I.

F.4.2 Type names

name	description	count
CDS	.PE protein coding region	1,071,882
ID	EMBL sequence data library entry	25,404,989
MISC_RNA	.RN other structural RNA coding region	483
RRNA	.RR ribosomal RNA coding region	1,696
SCRNA	.SC small cytoplasmic RNA coding region	4
SNRNA	.SN small nuclear RNA coding region	100
TRNA	.TR transfer RNA coding region	14,203
	Total:	26,493,357

F.5 swissprot

F.5.1 Bank details

**** ACNUC Data Base Content ****
UniProt Rel. 12 (SWISS-PROT 54 + TrEMBL 37): Last Updated: Nov 5, 2007
1,723,945,386 amino acids; 5,275,429 sequences; 298,590 references.

Non-redundant compilation of SWISS-PROT + TrEMBL
Software by M. Gouy & L. Duret, Laboratoire de biometrie, Universite Lyon I.

F.5.2 Type names

There are no subsequence type in this database

F.6 ensembl

F.6.1 Bank details

**** ACNUC Data Base Content ****
Ensembl databases release 41
Espece Release/#CDS(1)/STOP(2)/N(3)/miss(4)
Aedes aegypti 41_1a 11360/0/2/0 (0%/0%/0%)
Anopheles gambiae 41_3d 13510/0/31/0 (0%/0%/0%)
Apis mellifera 38_2d 27755/1/269/0 (0%/0%/0%)
Bos taurus 41_2 32556/7/620/12 (0%/1%/0%)
Caenorhabditis elegans 41_160 25218/1/0/0 (0%/0%/0%)
Canis familiaris 41_1j 29813/0/0/0 (0%/0%/0%)
Caenorhabditis briggsae 25 14712/0/23/1 (0%/0%/0%)
Ciona intestinalis 41_2c 20000/0/128/0 (0%/0%/0%)
Ciona savignyi 41_2b 20150/1/27/0 (0%/0%/0%)
Danio rerio 41_6b 36065/5/361/0 (0%/1%/0%)
Dasypus novemcinctus 40_1 13567/12/8857/0 (0%/65%/0%)
Drosophila melanogaster 41_43 19577/33/1/0 (0%/0%/0%)
Echinops telfairi 40_1 14309/8/9348/0 (0%/65%/0%)
Gallus gallus 41_1p 20667/13/455/0 (0%/2%/0%)
Gasterosteus aculeatus 41_1a 27181/13/138/0 (0%/0%/0%)
Homo sapiens 41_36c 47004/41/6/0 (0%/0%/0%)
Loxodonta africana 40_1 14366/10/9618/0 (0%/66%/0%)
Macaca mulatta 41_10a 36446/14/491/0 (0%/1%/0%)
Monodelphis domestica 41_3a 30358/0/80/0 (0%/0%/0%)
Mus musculus 41_36b 29026/34/2/0 (0%/0%/0%)
Oryctolagus cuniculus 41_1a 13705/4/8615/0 (0%/62%/0%)
Oryzias latipes 41_1 25880/0/546/0 (0%/2%/0%)
Pan troglodytes 41_21 32667/4/739/0 (0%/2%/0%)
Rattus norvegicus 41_34k 32996/34/686/0 (0%/2%/0%)
Saccharomyces cerevisiae 41_1d 4767/2/0/0 (0%/0%/0%)

F.6.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	188,480
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	485,916
CDS	.PE protein coding region	659,922
ID	EMBL sequence data library entry	742,978
INT_INT	.IN internal intron	4,342,137
MISC_RNA	.RN other structural RNA coding region	54,036
MRNA	.RN mRNA	659,922
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	7,133,391

F.7 refseq

F.7.1 Bank details

**** ACNUC Data Base Content ****
RefSeq 15.0 (1 January 2006) Last Updated: Jan 23, 2006
1,055,245,496 bases; 625,928 sequences; 254,162 subseqs; 205,831 refers.
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.7.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	624,776
INT_INT	.IN internal intron	0
LOCUS	sequenced DNA fragment	255,273
MISC_RNA	.RN other structural RNA coding region	8
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	2
SNRNA	.SN small nuclear RNA coding region	22
TRNA	.TR transfer RNA coding region	9
	Total:	880,090

F.8 hobacnucl

F.8.1 Bank details

**** ACNUC Data Base Content ****
HOBACGEN - genomic data - Release 10 (February 12 2002)
432,023,804 bases; 168,814 sequences; 293,669 subseqs; 52,735 references.

Bacteria + Archaea + Saccharomyces cerevisiae
Genomic data from EMBL Release 69 (December 2001)

F.8.2 Type names

name	description	count
CDS	.PE protein coding region	306,455
ID	EMBL sequence data library entry	94,694
MISC_RNA	.RN other structural RNA coding region	2,299
RRNA	.RR ribosomal RNA coding region	51,562
SCRNA	.SC small cytoplasmic RNA coding region	41
SNRNA	.SN small nuclear RNA coding region	193
TRNA	.TR transfer RNA coding region	7,239
	Total:	462,483

F.9 hobacprot

F.9.1 Bank details

**** ACNUC Data Base Content ****
HOBACGEN - protein data - Release 10 (February 12 2002)
79,755,852 amino acids; 260,025 sequences; 37,383 references.

Bacteria + Archaea + Saccharomyces cerevisiae
Protein data from SWISS-PROT 40 + TrEMBL 19 + TrEMBL_NEW: January 25, 2002

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & G. Perriere

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.9.2 Type names

There are no subsequence type in this database

F.10 hovernucl

F.10.1 Bank details

**** ACNUC Data Base Content ****
HOVERGEN - genomic data - Release 48 (May 24 2007) Last Updated: May 24, 2007
2,500,248,516 bases; 541,405 sequences; 1,005,089 subseqs; 117,556 refers.

Vertebrate (chordata)
Genomic data from EMBL Library Release 90 (March 2007)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.
Data maintenance: L. Duret & S. Penel

F.10.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	129,921
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	120,642
CDS	.PE protein coding region	613,473
ID	EMBL sequence data library entry	371,759
INT_INT	.IN internal intron	172,068
MISC_RNA	.RN other structural RNA coding region	249
RRNA	.RR ribosomal RNA coding region	9,873
SCRNA	.SC small cytoplasmic RNA coding region	24
SNRNA	.SN small nuclear RNA coding region	55
TRNA	.TR transfer RNA coding region	128,430
	Total:	1,546,494

F.11 hoverprot

F.11.1 Bank details

**** ACNUC Data Base Content ****
HOVERGEN - protein data - Release 48 (May 24 2007) Last Updated: May 24, 2007
142,891,140 amino acids; 415,383 sequences; 114,560 references.

Vertebrate (chordata)
Protein data from UniProt Rel. 10 (SWISS-PROT 52 + TrEMBL 35) May 2007

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1

43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.11.2 Type names

There are no subsequence type in this database

F.12 hogennucl

F.12.1 Bank details

**** ACNUC Data Base Content ****
HOGENOM - genomic data - Release 03 (Oct 14 2005) Last Updated: Nov 7, 2005
2,538,433,251 bases; 227,950 sequences; 4,136,134 subseqs; 82,281 refers.

Fully Sequenced Organisms
Protein data from http://www.ebi.ac.uk/proteome/ (August, 2005)
Genomic data from GenomeReview (June 2005)
and EMBL (June 2005)
(263 fully sequenced organisms)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.12.2 Type names

name	description	count
ID	EMBL sequence data library entry	204,502
CDS	.PE protein coding region	1,060,241
TRNA	.TR transfer RNA coding region	49,216
RRNA	.RR ribosomal RNA coding region	5,813
MISC_RNA	.RN other structural RNA coding region	861
SCRNA	.SC small cytoplasmic RNA coding region	29
SNRNA	.SN small nuclear RNA coding region	459
3'INT	.3I 3'intron	309
3'NCR	.3F 3'-non coding region	1,247,297
5'INT	.5I 5'intron	1,263
5'NCR	.5F 5'-non coding region	1,158,238
INT_INT	.IN internal intron	635,856
	Total:	4,364,084

F.13 hogenprot

F.13.1 Bank details

**** ACNUC Data Base Content ****
HOGENOM - protein data - Release 03 (Oct 14 2005) Last Updated: Mar 10, 2006
339,891,443 amino acids; 950,216 sequences; 92,805 references.

Fully Sequenced Organisms
Protein data from http://www.ebi.ac.uk/proteome/ (August 2005)
(263 fully sequenced organisms)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.13.2 Type names

There are no subsequence type in this database

F.14 hogen4nucl

F.14.1 Bank details

**** ACNUC Data Base Content ****
HOGENOM - genomic data - Release 04 (Sept 18,2007) Last Updated: Oct 2, 2007
14,692,834,718 bases; 134,844 sequences; 7,862,206 subseqs; 512 refers.

Fully Sequenced Organisms
Genomes
511 fully sequenced organisms (eukarya, bacteria, archaea)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.14.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	1,476,296
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	1,720,484
CDS	.PE protein coding region	2,125,031
ID	EMBL sequence data library entry	54,323
INT_INT	.IN internal intron	2,560,918
MISC_RNA	.RN other structural RNA coding region	22,520
RRNA	.RR ribosomal RNA coding region	6,378
SCRNA	.SC small cytoplasmic RNA coding region	11
SNRNA	.SN small nuclear RNA coding region	231
TRNA	.TR transfer RNA coding region	30,858
	Total:	7,997,050

F.15 hogen4prot

F.15.1 Bank details

**** ACNUC Data Base Content ****
HOGENOM - protein data - Release 04 (Sept 18,2007) Last Updated: Nov 6, 2007
755,031,736 amino acids; 2,142,639 sequences; 0 references.

Fully Sequenced Organisms
Protein data
511 fully sequenced organisms (eukarya, bacteria, archaea)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.15.2 Type names

There are no subsequence type in this database

F.16 homolensprot

F.16.1 Bank details

**** ACNUC Data Base Content ****
HOMOLENS 3 - Homologous genes from Ensembl Last Updated: Jan 19, 2007
224,528,520 amino acids; 474,339 sequences; 0 references.

Ensembl 41 Organisms Translated CDS
Aedes aegypti 41_1a 11360/0/2/0 (0%/0%/0%)
Anopheles gambiae 41_3d 13510/0/31/0 (0%/0%/0%)
Apis mellifera 38_2d 27755/1/269/0 (0%/0%/0%)
Bos taurus 41_2 32556/7/620/12 (0%/1%/0%)
Caenorhabditis elegans 41_160 25218/1/0/0 (0%/0%/0%)
Canis familiaris 41_1j 29813/0/0/0 (0%/0%/0%)
Caenorhabditis briggsae 25 14712/0/23/1 (0%/0%/0%)
Ciona intestinalis 41_2c 20000/0/128/0 (0%/0%/0%)
Ciona savignyi 41_2b 20150/1/27/0 (0%/0%/0%)
Danio rerio 41_6b 36065/5/361/0 (0%/1%/0%)
Dasypus novemcinctus 40_1 13567/12/8857/0 (0%/65%/0%)
Drosophila melanogaster 41_43 19577/33/1/0 (0%/0%/0%)
Echinops telfairi 40_1 14309/8/9348/0 (0%/65%/0%)
Gallus gallus 41_1p 20667/13/455/0 (0%/2%/0%)
Gasterosteus aculeatus 41_1a 27181/13/138/0 (0%/0%/0%)
Homo sapiens 41_36c 47004/41/6/0 (0%/0%/0%)
Loxodonta africana 40_1 14366/10/9618/0 (0%/66%/0%)
Macaca mulatta 41_10a 36446/14/491/0 (0%/1%/0%)
Monodelphis domestica 41_3a 30358/0/80/0 (0%/0%/0%)
Mus musculus 41_36b 29026/34/2/0 (0%/0%/0%)
Oryctolagus cuniculus 41_1a 13705/4/8615/0 (0%/62%/0%)
Oryzias latipes 41_1 25880/0/546/0 (0%/2%/0%)
Pan troglodytes 41_21 32667/4/739/0 (0%/2%/0%)
Rattus norvegicus 41_34k 32996/34/686/0 (0%/2%/0%)
Saccharomyces cerevisiae 41_1d 4767/2/0/0 (0%/0%/0%)
Takifugu rubripes 41_4c 22102/0/283/0 (0%/1%/0%)
Tetraodon nigroviridis 41_1g 15841/1/225/0 (0%/1%/0%)
Xenopus tropicalis 41_41b 28324/0/626/0 (0%/2%/0%)

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.16.2 Type names

There are no subsequence type in this database

F.17 homolensnucl

F.17.1 Bank details

**** ACNUC Data Base Content ****
HOMOLENS 3 Homologous genes from Ensembl 41 Last Updated: Jan 19, 2007
32,635,729,329 bases; 81,903 sequences; 5,717,782 subseqs; 0 refers.

Aedes aegypti 41_1a 11360/0/2/0 (0%/0%/0%)
Anopheles gambiae 41_3d 13510/0/31/0 (0%/0%/0%)
Apis mellifera 38_2d 27755/1/269/0 (0%/0%/0%)
Bos taurus 41_2 32556/7/620/12 (0%/1%/0%)
Caenorhabditis elegans 41_160 25218/1/0/0 (0%/0%/0%)
Canis familiaris 41_1j 29813/0/0/0 (0%/0%/0%)
Caenorhabditis briggsae 25 14712/0/23/1 (0%/0%/0%)
Ciona intestinalis 41_2c 20000/0/128/0 (0%/0%/0%)
Ciona savignyi 41_2b 20150/1/27/0 (0%/0%/0%)
Danio rerio 41_6b 36065/5/361/0 (0%/1%/0%)
Dasypus novemcinctus 40_1 13567/12/8857/0 (0%/65%/0%)
Drosophila melanogaster 41_43 19577/33/1/0 (0%/0%/0%)
Echinops telfairi 40_1 14309/8/9348/0 (0%/65%/0%)
Gallus gallus 41_1p 20667/13/455/0 (0%/2%/0%)
Gasterosteus aculeatus 41_1a 27181/13/138/0 (0%/0%/0%)
Homo sapiens 41_36c 47004/41/6/0 (0%/0%/0%)
Loxodonta africana 40_1 14366/10/9618/0 (0%/66%/0%)
Macaca mulatta 41_10a 36446/14/491/0 (0%/1%/0%)
Monodelphis domestica 41_3a 30358/0/80/0 (0%/0%/0%)
Mus musculus 41_36b 29026/34/2/0 (0%/0%/0%)
Oryctolagus cuniculus 41_1a 13705/4/8615/0 (0%/62%/0%)
Oryzias latipes 41_1 25880/0/546/0 (0%/2%/0%)
Pan troglodytes 41_21 32667/4/739/0 (0%/2%/0%)
Rattus norvegicus 41_34k 32996/34/686/0 (0%/2%/0%)
Saccharomyces cerevisiae 41_1d 4767/2/0/0 (0%/0%/0%)
Takifugu rubripes 41_4c 22102/0/283/0 (0%/1%/0%)
Tetraodon nigroviridis 41_1g 15841/1/225/0 (0%/1%/0%)
Xenopus tropicalis 41_41b 28324/0/626/0 (0%/2%/0%)

Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.17.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	188,371
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	485,692
CDS	.PE protein coding region	659,680
ID	EMBL sequence data library entry	81,903
INT_INT	.IN internal intron	4,339,670
MISC_RNA	.RN other structural RNA coding region	44,369
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	5,799,685

F.18 greview

F.18.1 Bank details

**** ACNUC Data Base Content ****
EBI Genome Reviews. Acnuc Release 7. Last Updated: Feb 26, 2007
1,454,629,225 bases; 757 sequences; 2,619,601 subseqs; 346 refers.
385 organisms
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.18.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	619,385
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	647,404
CDS	.PE protein coding region	1,233,635
ID	EMBL sequence data library entry	757
INT_INT	.IN internal intron	92,062
MISC_RNA	.RN other structural RNA coding region	0
RRNA	.RR ribosomal RNA coding region	4,288
SCRNA	.SC small cytoplasmic RNA coding region	11
SNRNA	.SN small nuclear RNA coding region	54
TRNA	.TR transfer RNA coding region	22,762
	Total:	2,620,358

F.19 HAMAPnucl

F.19.1 Bank details

**** ACNUC Data Base Content ****
HAMAP - Acnuc Release - Nucleotides - Last Updated: Jun 5, 2007
1,645,184,936 bases; 12,518 sequences; 1,529,575 subseqs; 11,742 refers.

Microbian genomes

Nucleotide data from EMBL cross-references
in protein data from HAMAP Database
(http://us.expasy.org/sprot/hamap/)
developed by the Swiss-Prot group at the Swiss Institut of Bioinformatics

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.19.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	1,494,504
ID	EMBL sequence data library entry	10,986
INT_INT	.IN internal intron	0
MISC_RNA	.RN other structural RNA coding region	1,496
RRNA	.RR ribosomal RNA coding region	5,925
SCRNA	.SC small cytoplasmic RNA coding region	12
SNRNA	.SN small nuclear RNA coding region	46
TRNA	.TR transfer RNA coding region	29,124
	Total:	1,542,093

F.20 HAMAPprot

F.20.1 Bank details

**** ACNUC Data Base Content ****
HAMAP - Acnuc Release -Proteins- Last Updated: Jun 5, 2007
59,598,968 amino acids; 189,746 sequences; 10,068 references.

Microbian genomes

Protein data from HAMAP Database
(http://us.expasy.org/sprot/hamap/)
developed by the Swiss-Prot group at the Swiss Institut of Bioinformatics

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.20.2 Type names

There are no subsequence type in this database

F.21 hoppsigen

F.21.1 Bank details

NA

F.21.2 Type names

name	description	count
ID	EMBL sequence data library entry	9,757
CDS	.PE protein coding region	3,814
TRNA	.TR transfer RNA coding region	0
RRNA	.RR ribosomal RNA coding region	0
MISC_RNA	.RN other structural RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
CDE	.PS	9,757
PPGENE	.PP	9,757
3'FL	.3F	3,656
5'FL	.5F	730
DIRECT_REPEAT	.DR	15,592
REPEAT_REGION	.RR	133,215
POLYA_REGION	.PA	1,694
FL_REPEAT	.FR	0
	Total:	187,972

F.22 nurebnucl

F.22.1 Bank details

**** ACNUC Data Base Content ****
Nurebase 4.0 (26 September 2003) Last Updated: NOV 27, 2003

2,356,663 bases; 664 sequences; 518 subseqs; 787 refers.
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.22.2 Type names

name	description	count
CDS	.PE protein coding region	767
ID	EMBL sequence data library entry	415
MISC_RNA	.RN other structural RNA coding region	0
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	1,182

F.23 nurebprot

F.23.1 Bank details

**** ACNUC Data Base Content ****
Nurebase 4.0 (26 September 2003) Last Updated: NOV 27, 2003
277,024 amino acids; 525 sequences; 634 references.
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.23.2 Type names

There are no subsequence type in this database

F.24 taxobacgen

F.24.1 Bank details

**** ACNUC Data Base Content ****
TaxoBacGen Rel. 7 (September 2005)
1,151,149,763 bases; 254,335 sequences; 847,767 subseqs; 63,879 refers.
Data compiled from GenBank by Gregory Devulder
Laboratoire de Biometrie & Biologie Evolutive, Univ Lyon I

This database is a taxonomic genomic database.
It results from an expertise crossing the data nomenclature database DSMZ

http : *//www.dsmz.de/species/bacteria.htmDeutscheSammlungvonMikroorganismenundZellkultu*

and GenBank.
- Only contains sequences described under species present in
Bacterial Nomenclature Up-to-date.
- Names of species and genus validly published according to the

Bacteriological Code (names with standing in nomenclature) is
added in field "DEFINITION".
- A keyword "type strain" is added in field "FEATURES/source/strain" in
GenBank format definition to easyly identify Type Strain.
Taxobacgen is a genomic database designed for studies based on a strict
respect of up-to-date nomenclature and taxonomy.

F.24.2 Type names

name	description	count
CDS	.PE protein coding region	879,340
LOCUS	sequenced DNA fragment	168,243
MISC_RNA	.RN other structural RNA coding region	3,720
RRNA	.RR ribosomal RNA coding region	34,965
SCRNA	.SC small cytoplasmic RNA coding region	36
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	15,798
	Total:	1,102,102

F.25 emblTP

F.25.1 Bank details

**** ACNUC Data Base Content ****
EMBL Library Release 78 WITHOUT ESTs (March 2004)
27,571,397,913 bases; 12,533,594 sequences; 1,604,500 subseqs; 339,186 refers.
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.25.2 Type names

name	description	count
CDS	.PE protein coding region	1,746,728
ID	Locus entry	11,856,048
MISC_RNA	.RN other structural RNA coding region	109,101
RRNA	.RR Ribosomal RNA coding gene	320,935
SCRNA	.SC small cytoplasmic RNA	311
SNRNA	.SN small nuclear RNA	1,687
TRNA	.TR Transfer RNA coding gene	103,284
	Total:	14,138,094

F.26 swissprotTP

F.26.1 Bank details

**** ACNUC Data Base Content ****
UniProt Rel. 1 (SWISS-PROT 43 + TrEMBL 26 + NEW): Last Updated: May 3, 2004
459,974,342 amino acids; 1,451,384 sequences; 200,578 references.

Non-redundant compilation of SWISS-PROT + TrEMBL (minus data
integrated into SWISS-PROT)
Software by M. Gouy & L. Duret, Laboratoire de biometrie, Universite Lyon I.

F.26.2 Type names

There are no subsequence type in this database

F.27 hoverprotTP

F.27.1 Bank details

**** ACNUC Data Base Content ****
HOVERGEN - Release 45 (Jan 22 2004) Last Updated: Jan 22, 2004
77,617,436 amino acids; 227,047 sequences; 85,918 references.

Vertebrate (chordata)
Protein data from SWISS-PROT Rel. 42 + TrEMBL Rel. 25 + TrEMBL_NEW:
Dec 1, 2003

Software: M. Gouy & M. Jacobzone
Data maintenance: L. Duret & S. Penel

Laboratoire de Biometrie et Biologie Evolutive
UMR CNRS 5558, Universite Claude Bernard - Lyon 1
43, bd du 11 Novembre 1918 F-69622 Villeurbanne Cedex

F.27.2 Type names

There are no subsequence type in this database

F.28 hovernuclTP

F.28.1 Bank details

**** ACNUC Data Base Content ****
HOVERGEN - Release 45 (Jan 22 2004) Last Updated: Jan 22, 2004
844,876,418 bases; 300,108 sequences; 757,209 subseqs; 97,608 refers.

Vertebrate (chordata)
Genomic data from EMBL Release 77 (December 2003)

Retrieval software by M. Gouy & M. Jacobzone, Lab. de Biometrie, UCB Lyon.

F.28.2 Type names

name	description	count
3'INT	.3I 3'intron	535
3'NCR	.3F 3'-non coding region	170,566
5'INT	.5I 5'intron	1,381
5'NCR	.5F 5'-non coding region	159,238
CDS	.PE protein coding region	274,599
ID	EMBL sequence data library entry	210,301
INT_INT	.IN internal intron	132,033
MISC_RNA	.RN other structural RNA coding region	164
RRNA	.RR ribosomal RNA coding region	2,426
SCRNA	.SC small cytoplasmic RNA coding region	9
SNRNA	.SN small nuclear RNA coding region	41
TRNA	.TR transfer RNA coding region	35,309
	Total:	986,602

F.29 emglib

F.29.1 Bank details

**** ACNUC Database Content ****
EMGLib Release 5 (December 9, 2003)
434,648,385 bases; 174 sequences; 413,521 subseqs; 169 refers.
Data compiled from various sources by Guy Perriere

F.29.2 Type names

name	description	count
CDS	.PE protein coding region	404,721
LOCUS	sequenced DNA fragment	174
MISC_RNA	.RN other structural RNA coding region	239
RRNA	.RR ribosomal RNA coding region	1,409
SCRNA	.SC small cytoplasmic RNA coding region	8
SNRNA	.SN small nuclear RNA coding region	6
TRNA	.TR transfer RNA coding region	7,138
	Total:	413,695

F.30 trypano

F.30.1 Bank details

**** ACNUC Data Base Content ****
trypano Rel. 1 (27 Janvier 2004) Last Updated: Jan 27, 2004
117,177,046 bases; 158,838 sequences; 4,744 subseqs; 2,114 refers.
Genomic data from GenBank Rel. 139 (15 December 2003)
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.30.2 Type names

name	description	count
LOCUS	sequenced DNA fragment	157,983
CDS	.PE protein coding region	5,137
TRNA	.TR transfer RNA coding region	38
RRNA	.RR ribosomal RNA coding region	206
MISC_RNA	.RN other structural RNA coding region	192
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	26
	Total:	163,582

F.31 ensembl41

F.31.1 Bank details

**** ACNUC Data Base Content ****
Ensembl databases release 41
Espece Release/#CDS(1)/STOP(2)/N(3)/miss(4)
Aedes aegypti 41_1a 11360/0/2/0 (0%/0%/0%)
Anopheles gambiae 41_3d 13510/0/31/0 (0%/0%/0%)
Apis mellifera 38_2d 27755/1/269/0 (0%/0%/0%)
Bos taurus 41_2 32556/7/620/12 (0%/1%/0%)
Caenorhabditis elegans 41_160 25218/1/0/0 (0%/0%/0%)
Canis familiaris 41_1j 29813/0/0/0 (0%/0%/0%)
Caenorhabditis briggsae 25 14712/0/23/1 (0%/0%/0%)
Ciona intestinalis 41_2c 20000/0/128/0 (0%/0%/0%)
Ciona savignyi 41_2b 20150/1/27/0 (0%/0%/0%)
Danio rerio 41_6b 36065/5/361/0 (0%/1%/0%)
Dasypus novemcinctus 40_1 13567/12/8857/0 (0%/65%/0%)
Drosophila melanogaster 41_43 19577/33/1/0 (0%/0%/0%)
Echinops telfairi 40_1 14309/8/9348/0 (0%/65%/0%)
Gallus gallus 41_1p 20667/13/455/0 (0%/2%/0%)
Gasterosteus aculeatus 41_1a 27181/13/138/0 (0%/0%/0%)
Homo sapiens 41_36c 47004/41/6/0 (0%/0%/0%)
Loxodonta africana 40_1 14366/10/9618/0 (0%/66%/0%)
Macaca mulatta 41_10a 36446/14/491/0 (0%/1%/0%)
Monodelphis domestica 41_3a 30358/0/80/0 (0%/0%/0%)
Mus musculus 41_36b 29026/34/2/0 (0%/0%/0%)
Oryctolagus cuniculus 41_1a 13705/4/8615/0 (0%/62%/0%)
Oryzias latipes 41_1 25880/0/546/0 (0%/2%/0%)
Pan troglodytes 41_21 32667/4/739/0 (0%/2%/0%)
Rattus norvegicus 41_34k 32996/34/686/0 (0%/2%/0%)
Saccharomyces cerevisiae 41_1d 4767/2/0/0 (0%/0%/0%)

F.31.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	188,480
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	485,916
CDS	.PE protein coding region	659,922
ID	EMBL sequence data library entry	742,978
INT_INT	.IN internal intron	4,342,137
MISC_RNA	.RN other structural RNA coding region	54,036
MRNA	.RN mRNA	659,922
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	7,133,391

F.32 ensembl34

F.32.1 Bank details

**** ACNUC Data Base Content ****
Ensembl databases release 34
Espece #CDS(1)/STOP(2)/N(3)/miss(4)
Apis mellifera 27736/1/269 (0%/0%/0%)
Caenorhabditis briggsae 14712/0/23 (0%/0%/0%)
Caenorhabditis elegans 25797/1/0 (0%/0%/0%)
Gallus gallus 28392/20/298 (0%/1%/0%)
Pan troglodytes 39538/6129/770 (15%/1%/0%)
Ciona intestinalis 21574/0/58 (0%/0%/0%)
Bos taurus 32647/7/617 (0%/1%/0%)
Canis familiaris 29998/0/0 (0%/0%/1%)
Drosophila melanogaser 19350/18/1 (0%/0%/0%)
Fugu rubripes 22099/0/283 (0%/1%/0%)
Homo sapiens 36919/48/24 (0%/0%/2%)
Macaca mulatta 31370/94/8360 (0%/26%/0%)
Anopheles gambiae 15799/0/19 (0%/0%/0%)
Mus musculus 35075/36/60 (0%/0%/1%)
Monodelphis domestica 13249/0/59 (0%/0%/0%)
Rattus norvegicus 32241/25/607 (0%/1%/2%)
Tetraodon nigroviridis 16275/1/233 (0%/1%/0%)
Xenopus tropicalis 52684/1/906 (0%/1%/0%)
Saccharomyces cerevisiae 6680/22/0 (0%/0%/0%)
Danio rerio 32109/0/281 (0%/0%/0%)
1:# of CDS;2:CDS with internal stop;3:CDS with undetermined codon;4:missing CDS
warning : cds located on contigs were removed
29,605,509,937 bases; 368,619 sequences; 5,302,323 subseqs; 0 refers.

Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.32.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	156,270
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	280,705
CDS	.PE protein coding region	534,246
ID	EMBL sequence data library entry	368,619
INT_INT	.IN internal intron	3,763,554
MISC_RNA	.RN other structural RNA coding region	33,511
MRNA	.RN mRNA	534,037
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	5,670,942

F.33 genomicro1

F.33.1 Bank details

**** ACNUC Data Base Content ****
Genomicro1 (15 June 2006) Last Updated: Jul 6, 2006
10,758,321,631 bases; 203,021 sequences; 1,870,190 subseqs; 0 refers.

F.33.2 Type names

name	description	count
3'NCR	.3F 3'-non coding region	34,406
5'NCR	.5F 5'-non coding region	84,333
CDS	.PE protein coding region	91,991
EXON	.EX exon	545,221
GENE	.GE gene	74,019
ID	EMBL sequence data library entry	203,020
INT_INT	.IN internal intron	531,587
MISC_FEATURE	.MF misc feature	397,178
MISC_RNA	.RN other structural RNA coding region	19,549
MRNA	.MA mrna	91,907
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	2,073,211

F.34 genomicro2

F.34.1 Bank details

**** ACNUC Data Base Content ****
Genomicro (15 June 2006) Last Updated: Jul 6, 2006
9,861,694,164 bases; 280 sequences; 2,858,384 subseqs; 0 refers.

F.34.2 Type names

name	description	count
3'NCR	.3F 3'-non coding region	45,675
5'NCR	.5F 5'-non coding region	99,248
CDS	.PE protein coding region	135,798
EXON	.EX exon	831,047
GENE	.GE gene	97,094
ID	EMBL sequence data library entry	278
INT_INT	.IN internal intron	1,037,668
MISC_FEATURE	.MF misc feature	457,916
MISC_RNA	.RN other structural RNA coding region	18,143
MRNA	.MA mrna	135,797
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	2,858,664

F.35 microbes

F.35.1 Bank details

**** ACNUC Data Base Content ****
NCBI Microbial Genomes. Acnuc Release 1. Last Updated: Mar 6, 2007
1,628,177,183 bases; 826 sequences; 1,511,540 subseqs; 853 refers.
464 organisms
Software by M. Gouy & M. Jacobzone, Laboratoire de biometrie, Universite Lyon I

F.35.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	1,478,713
INT_INT	.IN internal intron	0
LOCUS	sequenced DNA fragment	826
MISC_RNA	.RN other structural RNA coding region	1,407
RRNA	.RR ribosomal RNA coding region	5,324
SCRNA	.SC small cytoplasmic RNA coding region	6
SNRNA	.SN small nuclear RNA coding region	6
TRNA	.TR transfer RNA coding region	26,084
	Total:	1,512,366

F.36 macaca

F.36.1 Bank details

**** ACNUC Data Base Content ****
Ensembl Macaca mulatta (Rel. 45_10e) Last Updated: Jun 25, 2007
3,028,222,859 bases; 94,529 sequences; 41,046 subseqs; 0 refers.
Software by M. Gouy, Laboratoire de biometrie, Universite Lyon I

F.36.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	35,875
ID	EMBL sequence data library entry	94,529
INT_INT	.IN internal intron	0
MISC_RNA	.RN other structural RNA coding region	5,171
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	135,575

F.37 canis

F.37.1 Bank details

**** ACNUC Data Base Content ****
Ensembl Canis familiaris (Rel. 45_2c) Last Updated: Jul 4, 2007
2,531,673,731 bases; 2,585 sequences; 29,227 subseqs; 0 refers.
Software by M. Gouy, Laboratoire de biometrie, Universite Lyon I

F.37.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	25,559
ID	EMBL sequence data library entry	2,585
INT_INT	.IN internal intron	0
MISC_RNA	.RN other structural RNA coding region	3,668
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	31,812

F.38 mouse38

F.38.1 Bank details

**** ACNUC Data Base Content ****
Ensembl Mus musculus (Rel.38_35) Last Updated: Jul 6, 2007

2,676,276,909 bases; 7,505 sequences; 35,002 subseqs; 0 refers.
Software by M. Gouy, Laboratoire de biometrie, Universite Lyon I

F.38.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	0
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	0
CDS	.PE protein coding region	31,984
ID	EMBL sequence data library entry	7,505
INT_INT	.IN internal intron	0
MISC_RNA	.RN other structural RNA coding region	3,018
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	42,507

F.39 homo46

F.39.1 Bank details

**** ACNUC Data Base Content ****
Ensembl Homo sapiens rel 46_36h (September 2007) Last Updated: Sep 3, 2007
3,662,694,094 bases; 3,859 sequences; 551,227 subseqs; 0 refers.
MENU Nber of lines= 21

F.39.2 Type names

name	description	count
3'INT	.3I 3'intron	0
3'NCR	.3F 3'-non coding region	21,970
5'INT	.5I 5'intron	0
5'NCR	.5F 5'-non coding region	44,246
CDS	.PE protein coding region	50,736
ID	EMBL sequence data library entry	3,859
INT_INT	.IN internal intron	424,658
MISC_RNA	.RN other structural RNA coding region	9,617
RRNA	.RR ribosomal RNA coding region	0
SCRNA	.SC small cytoplasmic RNA coding region	0
SNRNA	.SN small nuclear RNA coding region	0
TRNA	.TR transfer RNA coding region	0
	Total:	555,086

Session Informations

This part was compiled under the following  environment:

- R version 2.6.0 (2007-10-03), `i386-apple-darwin8.10.1`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.2-36, ade4 1.4-4, ape 2.0-1, gee 4.13-13, lattice 0.16-5, nlme 3.1-85, seqinr 1.1-3, xtable 1.5-1
- Loaded via a namespace (and not attached): grid 2.6.0, rcompgen 0.1-15

There were two compilation steps:

- R compilation time was: Wed Nov 7 12:40:01 2007
- LaTeX compilation time was: December 2, 2007

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. 52

[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. 120, 125

[3] 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. 134

[4] F.R. Blattner, G. Plunkett III, C.A. Bloch, N.T. Perna, V. Burland, M. Riley, 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. 134

[5] J. Buckheit and D. L. Donoho. *Wavelets and Statistics*, chapter Wavelab and reproducible research. Springer-Verlag, Berlin, New York, 1995. A. Antoniadis editor. 5

[6] 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. 134

[7] 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. 3, 84

[8] 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. 5

[9] D.B Dahl and *et al.* *xtable: Export tables to LaTeX or HTML*, 2005. R package version 1.3-0. 10

[10] 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. 134

[11] 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. 168

[12] 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

[13] J. Felsenstein. PHYLIP-phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989. 28

[14] A.C. Frank and J.R. Lobry. Oriloc: prediction of replication boundaries in unannotated bacterial chromosomes. *Bioinformatics*, 16(6):560–561, 2000. 15

[15] 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. 26, 27

[16] 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. 16

[17] C. Gautier. *Analyses statistiques et évolution des séquences d'acides nucléiques*. PhD thesis, Université Claude Bernard - Lyon I, 1987. 87

[18] 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. 1, 2, 35, 169

[19] 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. 1, 2, 35, 169

[20] S.J. Gould. *Wonderful life*. Norton, New York, USA, 1989. 2

[21] 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

[22] M. Gouy and S. Delmotte. Remote access to ACNUC nucleotide and protein sequence databases at PBIL. *Biochimie*, in press:000–000, 2007. 84

[23] 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. 35, 84

[24] M. Gouy, C. Gautier, and F. Milleret. System analysis and nucleic acid sequence banks. *Biochimie*, 67:433–436, 1985. 35

[25] 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, 35, 111

[26] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185:862–864, 1974. 3

[27] 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. 16, 21, 24, 26

[28] 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. 134

[29] D. G. Higgins and P. M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988. 27

[30] 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/`. 4

[31] L.D. Hurst. The Ka/Ks ratio: diagnosing the form of sequence evolution. *Trends Genet.*, 18:486–487, 2002. 99

[32] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *J. Comp. Graph. Stat.*, 3:299–314, 1996. 3

[33] M. Jacobzone and C. Gautier. *ANALSEQ Manuel d'utilisation*. UMR CNRS 5558, Biométrie, Génétique et Biologie des Populations, 1989. 3, 111

[34] T. H. Jukes and S. Osawa. Evolutionary changes in the genetic code. *Comp. Biochem. Physiol. B.*, 106:489–494, 1993. 153

[35] 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. 32, 33

[36] S. Karlin and V. Brendel. Chance and statistical significance in protein and dna sequence analysis. *Science*, 257:39–49, 1992. 120, 171

[37] S. Kawashima and M. Kanehisa. AAindex: amino acid index database. *Nucleic Acids Res.*, 28:374–374, 2000. 3, 170

[38] J. Keogh. Circular transportation facilitation device, 2001. 5

[39] 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. 34

[40] 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. 20, 94

[41] P. Legendre, Y. Desdevises, and E. Bazin. A statistical test for host-parasite coevolution. *Syst. Biol.*, 51:217–234, 2002. 31

[42] F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. *Proceedings in Computational Statistics*, Compstat 2002:575–580, 2002. 3, 10

[43] W.-H. Li. Unbiased estimation of the rates of synonymous and nonsynonymous substitution. *J. Mol. Evol.*, 36:96–99, 1993. 99

[44] 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. 165

[45] J.R. Lobry. Asymmetric substitution patterns in the two DNA strands of bacteria. *Molecular Biology and Evolution*, 13:660–665, 1996. 133, 134, 165

[46] 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. 8, 9

[47] 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. 100

[48] 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. 92, 104, 105, 106

[49] J.R. Lobry and N. Sueoka. Asymmetric directional mutation pressures in bacteria. *Genome Biology*, 3(10):research0058.1–research0058.14, 2002. 5

[50] A.O. Lovejoy. *The Great Chain of Being: A Study of the History of an Idea*. Harvard University Press, Cambridge, Massachusetts, USA, 1936. 2

[51] 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. 15

[52] 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`. 134

[53] Paul Murrell and Richard Walton. *grImport: Importing Vector Graphics*, 2006. R package version 0.2. 3

[54] 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, 170

[55] S. Osawa, T. H. Jukes, K. Watanabe, and A. Muto. Recent evidence for evolution of the genetic code. *Microbiol. Rev.*, 56:229–264, 1992. 153

[56] H. Pages, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*, 2007. R package version 2.6.4. 165

[57] 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. 165

[58] 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. 120, 122, 123, 131, 172

[59] E. Paradis, J. Claude, and K. Strimmer. Ape: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004. 32

[60] J. Pačes, R. Zíka, V. Pačes, A. Pavlíček, O. Clay, and G. Bernardi. Representing GC variation along eukaryotic chromosomes. *Gene*, 333:135–141, 2004. 112

[61] 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. 11

[62] J.F. Peden. *Analysis of codon usage*. PhD thesis, University of Nottingham, 1999. 137

[63] G. Perrière and J. Thioulouse. Use and misuse of correspondence analysis in codon usage studies. *Nucleic Acids Res.*, 30:4548–4555, 2002. 87, 100

[64] 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. 134

[65] T.I. Quickenden and J.A. Irvin. The ultraviolet absorption spectrum of liquid water. *The Journal of Chemical Physics*, 72:4416–4428, 1980. 165

[66] 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

[67] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0. 3, 84

[68] 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. 5

[69] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1984. 32

[70] 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. 168

[71] 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. 123

[72] R. B. Setlow. Cyclobutane-type pyrimidine dimers in polynucleotides. *Science*, 153:379–386, 1966. 125

[73] 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. 105

[74] C.E. Singer and B.N. Ames. Sunlight ultraviolet and bacterial DNA base ratios. *Science*, 170:822–826, 1970. 120, 125, 131

[75] 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. 134

[76] R. Staden. Graphic methods to determine the function of nucleic acid sequences. *Nucleic Acids Res.*, 12:521–538, 1984. 3

[77] 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. 142

[78] 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. 143

[79] 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, 170

[80] Adrian Trapletti and Kurt Hornik. *tseries: Time Series Analysis and Computational Finance*, 2007. R package version 0.10-11. 118

[81] I.M. Wallace, G. Blackshields, and D.G. Higgins. Multiple sequence alignments. *Curr. Opin. Struct. Biol.*, 15:261–266, 2005. 27

[82] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1:80–83, 1945. 114

[83] 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. 134