1

SeqinR 1.0-3: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis

Delphine Charif[1], Jean R. Lobry[1]

Université Claude Bernard - Lyon I
Laboratoire de Biométrie, Biologie Évolutive
CNRS UMR 5558 - INRIA Helix project
43 Bd 11/11/1918
F-69622 VILLEURBANNE CEDEX, FRANCE
http://pbil.univ-lyon1.fr/members/lobry/

Summary. The **seqinR** package for the R environment is a library of utilities to retrieve and analyse biological sequences. It provides an interface between i) the R language and environment for statistical computing and graphics and ii) the ACNUC sequence retrieval system for nucleotide and protein sequence databases such as GenBank, EMBL, SWISS-PROT. ACNUC is very efficient in providing direct access to subsequences of biological interest (*e.g.* protein coding regions, tRNA or rRNA coding regions) present in GenBank and in EMBL. Thanks to a simple query language, it is then easy under R to select sequences of interest and then use all the power of the R environment to analyze them. The ACNUC databases can be locally installed but they are more conveniently accessed through a web server to take advantage of centralized daily updates. The aim of this paper is to provide a handout on basic sequence analyses under **seqinR** with a special focus on multivariate methods.

1.1 Introduction

1.1.1 About R and CRAN

R [8, 20] 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 50

mirrors available from 22 countries. Please use the CRAN mirror nearest to you to minimize network load, they are listed at `http://cran.r-project.org/mirrors.html`.

1.1.2 About this document

In the terminology of the R project [8, 20], this document is a package *vignette*. The examples given thereafter were run under `R version 2.1.0, 2005-04-18` on Thu Jul 21 15:30:14 2005 with Sweave [5]. 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] "../../../library/seqinr"
```

1.1.3 About sequin and seqinR

Sequin is the well known sofware used to submit sequences to GenBank, **seqinR** 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. This is a very good summary of a major functionality of the **seqinR** package: to provide an efficient access to sequence databases under R.

1.1.4 About getting started

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

```
library(seqinr, lib = "../../../library/")
```

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

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

```
[1] "AAstat"          "EXP"             "GC"              "GC2"
[5] "GC3"             "SEQINR.UTIL"     "a"               "aaa"
[9] "as.SeqAcnucWeb"
```

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.

1.1.5 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 [4] 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")`).

1.1.6 About the learning curve

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.

Wheel (the): do not re-invent (there's a patent [10] on it anyway). At the compilation time of this document there were 565 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.

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

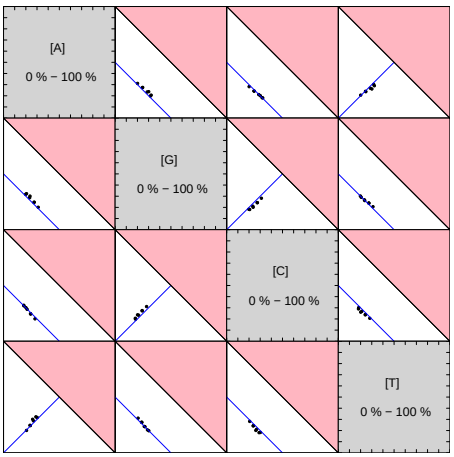
Automation: consider the 178 pages of figures in the additional data file 1 (`http://genomebiology.com/2002/3/10/research/0058/suppl/`

`S1`) from [17]. They were produced in part automatically (with a proprietary software that is no more maintained) and manually, involving a lot of tedious and repetitive manipulations (such as italicising species names by hand in subtitles). In few words, a waste of time. The advantage of the R environment is that once you are happy with the outputs (including graphical outputs) of an analysis for species x, it's very easy to run the same analysis on n species.

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 [2] is a must read. Molecular data are available in public databases, this is a necessary but not sufficient condition to allow for the reproducibility of results. Publishing the R source code that was used in your analyses is a simple way to greatly facilitate the reproduction of your results at the expense of no extra cost. At the expense of a little extra cost, you may consider to set up a RWeb server so that even the laziest reviewer may reproduce your results just by clicking on the "do it again" button in his web browser (*i.e.* without installing any software on his computer). For an example involving the **seqinR** pacakage, follow this link `http://pbil.univ-lyon1.fr/members/lobry/repro/bioinfo04/` to reproduce on-line the results from [1].

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 [21] 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)
```



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(dia = F)
```

Genetic code 1 : standard

u u u	Phe	u c u	Ser	u a u	Tyr	u g u	Cys
u u c	Phe	u c c	Ser	u a c	Tyr	u g c	Cys
u u a	Leu	u c a	Ser	u a a	Stp	u g a	Stp
u u g	Leu	u c g	Ser	u a g	Stp	u g g	Trp
c u u	Leu	c c u	Pro	c a u	His	c g u	Arg
c u c	Leu	c c c	Pro	c a c	His	c g c	Arg
c u a	Leu	c c a	Pro	c a a	Gln	c g a	Arg
c u g	Leu	c c g	Pro	c a g	Gln	c g g	Arg
a u u	Ile	a c u	Thr	a a u	Asn	a g u	Ser
a u c	Ile	a c c	Thr	a a c	Asn	a g c	Ser
a u a	Ile	a c a	Thr	a a a	Lys	a g a	Arg
a u g	Met	a c g	Thr	a a g	Lys	a g g	Arg
g u u	Val	g c u	Ala	g a u	Asp	g g u	Gly
g u c	Val	g c c	Ala	g a c	Asp	g g c	Gly
g u a	Val	g c a	Ala	g a a	Glu	g g a	Gly
g u g	Val	g c g	Ala	g a g	Glu	g g g	Gly

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

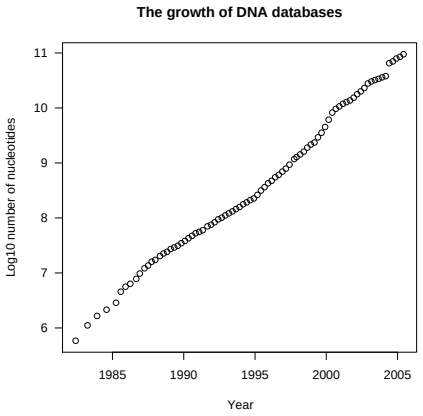
```
tablecode(numcode = 2, dia = F)
```

Genetic code 2 : vertebrate.mitochondrial

u u u	Phe	u c u	Ser	u a u	Tyr	u g u	Cys
u u c	Phe	u c c	Ser	u a c	Tyr	u g c	Cys
u u a	Leu	u c a	Ser	u a a	Stp	u g a	Trp
u u g	Leu	u c g	Ser	u a g	Stp	u g g	Trp
c u u	Leu	c c u	Pro	c a u	His	c g u	Arg
c u c	Leu	c c c	Pro	c a c	His	c g c	Arg
c u a	Leu	c c a	Pro	c a a	Gln	c g a	Arg
c u g	Leu	c c g	Pro	c a g	Gln	c g g	Arg
a u u	Ile	a c u	Thr	a a u	Asn	a g u	Ser
a u c	Ile	a c c	Thr	a a c	Asn	a g c	Ser
a u a	Met	a c a	Thr	a a a	Lys	a g a	Stp
a u g	Met	a c g	Thr	a a g	Lys	a g g	Stp
g u u	Val	g c u	Ala	g a u	Asp	g g u	Gly
g u c	Val	g c c	Ala	g a c	Asp	g g c	Gly
g u a	Val	g c a	Ala	g a a	Glu	g g a	Gly
g u g	Val	g c g	Ala	g a g	Glu	g g g	Gly

Data as fast moving targets: in research area, data are not always stable. compare the following graph :

```
dbg <- get.db.growth()
plot(x = dbg$date, y = log10(dbg$Nucl), las = 1, main = "The growth of DNA databases",
    xlab = "Year", ylab = "Log10 number of nucleotides")
```



with figure 1 in [14], data have been updated since then but the same R code was used to produce the figure, ensuring an automatic update. For LaTeX users, it's worth mentioning the fantastic tool contributed by Friedrich Leish [5] 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` to coerce R data into LaTeX tables, for instance table 1.2 here was produced this way, enforcing a complete coherence between the R code example and the table.

1.2 How to get sequence data

1.2.1 Importing raw sequence data from fasta files

The fasta format is very simple and widely used for simple import of biological sequences. 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] "bb.fasta"    "ct.fasta"    "malM.fasta"  "seqAA.fasta"
```

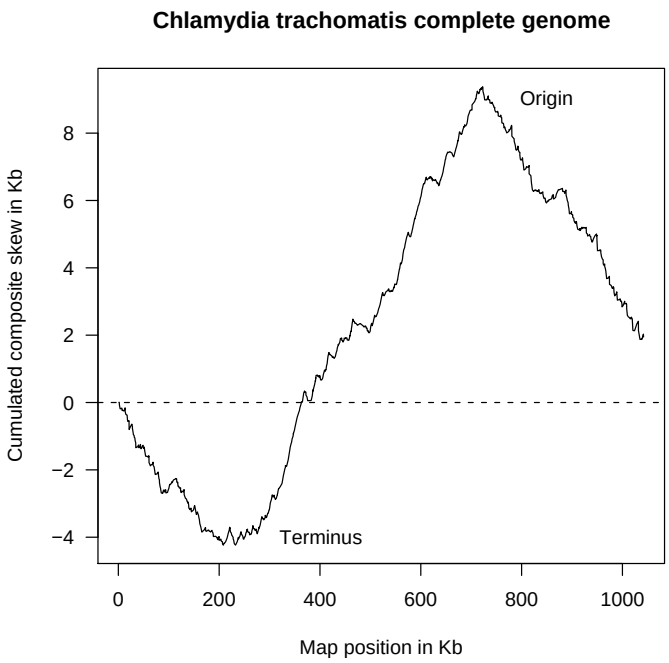
The function `read.fasta()` imports sequences from fasta files into your workspace, for example:

```
seqaa <- read.fasta(File = system.file("sequences/seqAA.fasta",
    package = "seqinr"), seqtype = "AA")
seqaa
```

```
$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"
```

A more consequent example is given in the fasta file `ct.fasta` which contains the complete genome of *Chlamydia trachomatis* that was used in [6]. 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 = "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 [18] for a recent review on this topic.

1.2.2 Importing aligned sequence data

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.

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. Download on your local computer the following two sample tests of aligned sequences of this gene (extracted from ParaFit [12]), this can be done directly at R prompt in the R console with :

```
download.file(url = "http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/louse.fasta",
    destfile = "louse.fasta")
```

```
download.file(url = "http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/gopher.fasta",
    destfile = "gopher.fasta")
```

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) :

```
l.names <- readLines("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/louse.names")
l.names
```

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

```
g.names <- readLines("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/gopher.names")
g.names
```

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

```
louse <- read.alignment("louse.fasta", format = "fasta")
gopher <- read.alignment("gopher.fasta", format = "fasta")
```

The aligned sequences are now imported in your R environment. SeqinR has very few methods devoted to phylogenetic analyses but many are available in the ape package. 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 [22] with Jukes and Cantor's correction [9] for multiple substitutions.

```
library(ape)
```

```
Loading required package: gee
Loading required package: nlme

Attaching package: 'nlme'


        The following object(s) are masked from package:stats :

         contr.SAS

Loading required package: lattice
```

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

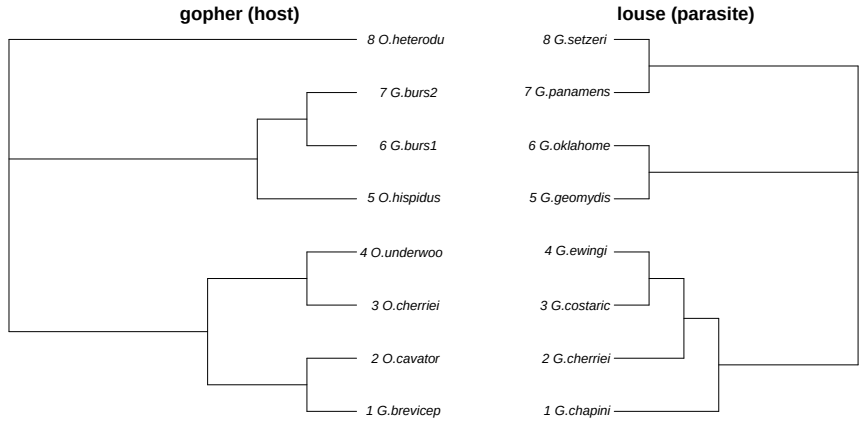
Now we have an estimation for *illustrative* purposes of the tree topology for the parasite and their hosts. We want to plot the two trees face to face,

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 :

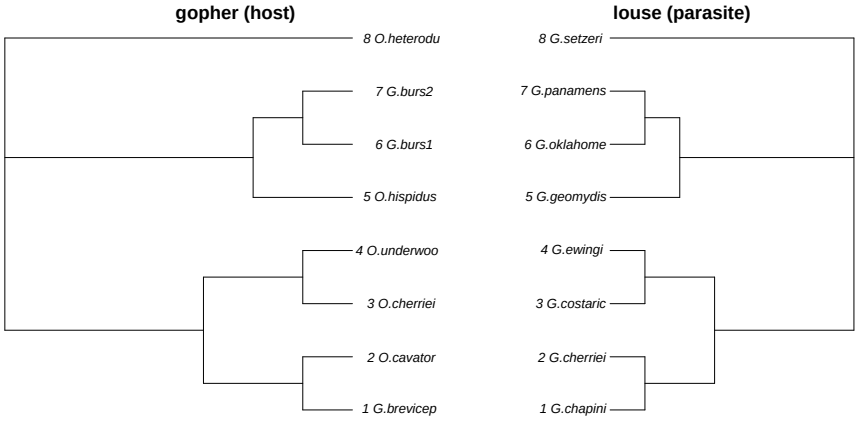
```
l.names <- readLines("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/louse.names")
g.names <- readLines("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/gopher.names")
louse <- read.alignment("louse.fasta",format="fasta")
gopher <- read.alignment("gopher.fasta",format="fasta")
louse.JC <- dist.dna(x =  lapply(louse$seq, s2c), model = "JC69" )
gopher.JC <- dist.dna(x =  lapply(gopher$seq, s2c), 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 (we have assumed here that the files `louse.fasta` and `gopher.fasta` are present in your local working directory). Now, your boss is unhappy with working with the Jukes and Cantor's model [9] and wants you to use the Kimura's 2-parameters distance [11] 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 :



Nice congruence, isn't it? 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. Open the `louse.fasta` 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.

1.2.3 Complex queries in ACNUC databases

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:

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"  "ensembl"
 [6] "emglib"     "nrsub"      "nbrf"       "hobacnucl"  "hobacprot"
[11] "hovernucl"  "hoverprot"  "hogennucl"  "hogenprot"  "hoverclnu"
[16] "hoverclpr"  "HAMAPnucl"  "HAMAPprot"  "hoppsigen"  "nurebnucl"
[21] "nurebprot"  "taxobacgen" "greview"
```

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 four first databases on the default server at the compilation time (July 21, 2005) of this document:

```
choosebank(infobank = TRUE)[1:4, ]
```

```
       bank status
1   genbank     on
2      embl     on
3   emblwgs     on
4 swissprot     on
                                                                    info
1             GenBank Rel. 148 (15 June 2005) Last Updated: Jul 21, 2005
2                                    EMBL Library Release 83 (June 2005)
3            EMBL Whole Genome Shotgun sequences Release 83  (June 2005)
4 UniProt Rel. 5 (SWISS-PROT 47 + TrEMBL 30): Last Updated: Jul 11, 2005
```

Note that there is a `status` column because a database could be unavailable for a while during updates[1].

Now, if you want to work with a given database, say GenBank, just call `choosebank()` with `"genbank"` as its first argument and store the result in a variable in the workspace, called for instance `mybank` in the example thereafter:

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

```
List of 8
 $ socket  :Classes 'sockconn', 'connection'  int 11
 $ bankname: chr "genbank"
 $ totseqs : chr "48429668"
 $ totspecs: chr "305457"
 $ totkeys : chr "1325145"
 $ release : chr "GenBank Rel. 148 (15 June 2005) Last Updated: Jul 21, 2005"
 $ status  :Class 'AsIs'  chr "on"
 $ details : chr [1:3] "GenBank Rel. 148 (15 June 2005) Last Updated: Jul 21, 2005" "50,908,937,801 bases; 46,084,115 sequences;
```

[1] If you try call `choosebank(bank = "bankname")` when the bank called `bankname` is of from server, you will get an explicit error message stating that this bank is temporarily unvailable.

The list returned by `choosebank()` here means that in the database called `genbank` at the compilation time of this document there were `48,429,668` sequences from `305,457` species and a total of `1,325,145` keywords. The status of the bank was `on`, and the release information was `GenBank Rel. 148 (15 June 2005) Last Updated: Jul 21, 2005`. For specialized databases, some relevant informations are also given in the `details` component, for instance:

```
choosebank("taxobacgen")$details
```

```
 [1] "TaxoBacGen Rel. 6 (April 2005)"
 [2] "1,039,981,089 bases; 241,483 sequences; 755,110 subseqs; 57,056 refers."
 [3] "Data compiled from GenBank by Gregory Devulder"
 [4] "Laboratoire de Biometrie & Biologie Evolutive, Univ Lyon I"
 [5] "------------------------------"
 [6] "This database is a taxonomic genomic database."
 [7] "It results from an expertise crossing the data nomenclature database DSMZ"
 [8] "[http://www.dsmz.de/species/bacteria.htm Deutsche Sammlung von"
 [9] "Mikroorganismen und Zellkulturen GmbH, Braunschweig, Germany]"
[10] "and GenBank."
[11] "- Only contains sequences described under species present in"
[12] "Bacterial Nomenclature Up-to-date."
[13] "- Names of species and genus validly published according to the"
[14] "Bacteriological Code (names with standing in nomenclature) is"
[15] "added in field \"DEFINITION\"."
[16] "- A keyword \"type strain\" is added in field \"FEATURES/source/strain\" in"
[17] "GenBank format definition to easyly identify Type Strain."
[18] "Taxobacgen is a genomic database designed for studies based on a strict"
[19] "respect of up-to-date nomenclature and taxonomy."
```

For the following, the most important item is the first one of the list, `mybank$socket`, which contains all the required details of the socket connection[2].

Make your query

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 (`sp=felis catus`) that are not (`et no`) partial sequences (`k=partial`). We want the result to be stored in an object called `list1`.

```
query(socket = mybank$socket, listname = "list1", query = "sp=felis catus et t=cds et no k=partial",
    invisible = TRUE)
```

Now, there is in the workspace an object called `list1`, which does not contain the sequences themselves but the *sequence names* that fit the query. They are stored in the `req` component of the object, let's see the first ten of them:

```
sapply(list1$req[1:10], getName)
```

[2] 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 is given to the `query()` function, the last opened database will be used by default.

```
 [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 1.1.

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 1.1. Available subsequences in genbank

Note that the component `call` of `list1` keeps a trace of the way you have selected the sequences:

```
 list1$call
```

```
query(socket = mybank$socket, listname = "list1", query = "sp=felis catus et t=cds et no k=partial",
    invisible = TRUE)
```

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 `list1`, 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 `list1` get only the list of sequences that were published in 2004:

```
 query(socket = mybank$socket, listname = "list2", query = "list1 et y=2004",
     invisible = TRUE)
 length(list2$req)
```

```
[1] 50
```

Hence, there were 50 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 not to download all information from list elements. This is interesting for list whit many elements, for instance :

```
query(socket = mybank$socket, listname = "allcds", query = "t=cds",
    invisible = TRUE, virtual = TRUE)
allcds$nelem
```

```
[1] 2528417
```

There are therefore 2,528,417 coding sequences in this version of GenBank. It would be long to get all the informations for the elements of this list, so we have set the parameter `virtual` to `TRUE` and the `req` component of the list has not been documented:

```
allcds$req
```

```
[[1]]
[1] NA
```

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

```
query(socket = mybank$socket, listname = "small", query = "allcds et sp=mycoplasma genitalium",
    invisible = TRUE, virtual = TRUE)
small$nelem
```

```
[1] 892
```

There are then 892 elements in the list `small`, so that we can safely repeat the previous query without asking for a virtual list:

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

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

Extract sequences of interest

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

```
myseq <- getSequence(list1$req[[1]])
myseq[1:50]
```

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

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

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

```
[1] "atgaatcaaggagccgtttttaggcacctgctcctggtgctgcagctggt"
```

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:

```
    AE003734.PE35        Location/Qualifiers    (length=1833 bp)
         CDS             join(complement(162997..163210),
                         complement(162780..162919),complement(161238..162090),
                         146568..146732,146806..147266)
                         /gene="mod(mdg4)"
                         /locus_tag="CG32491"
                         /note="CG32491 gene product from transcript CG32491-RT;
                         trans-splicing"
```

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(socket = mybank$socket, listname = "transspliced", query = "N=AE003734.PE35",
    invisible = TRUE)
length(transspliced$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[3] 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 55  9 38 50 22 28 11 20 40 36 10 21 35 57 22 54 50 38  1 13
```

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

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

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

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

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

1.3 How to deal with sequences

1.3.1 Sequence classes

There are at present 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

1.3.2 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

1.3.3 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.

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:

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

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" "c" "a" "g" "g" "g" "c"
```

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 1425 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
```

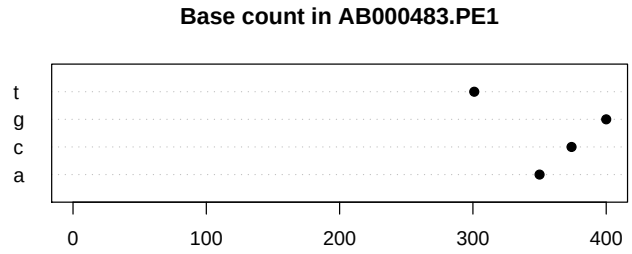
This recycling rule is extremely convenient in practice but may have surprising effects if you assume (incorrectly) that there is a stringent dimension control for R vectors as in linear algebra.

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:

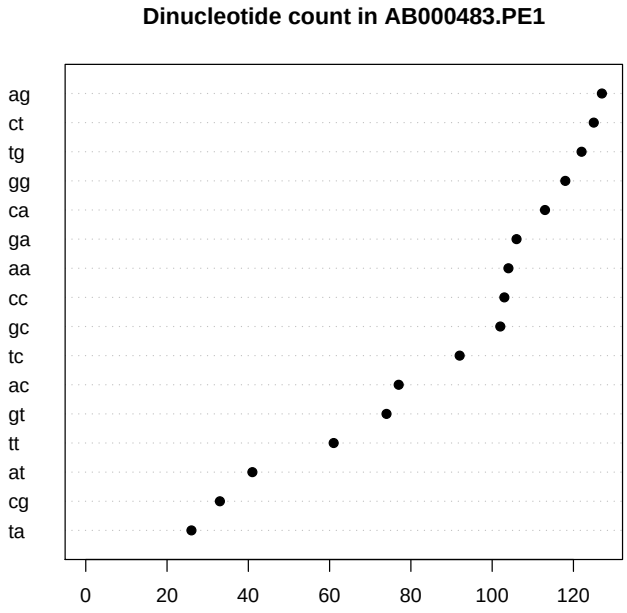
```
tota <- sum(myseq == "a")
```

The total number of `a` in our sequence is 350. Let's compare graphically the different base counts in our sequence :

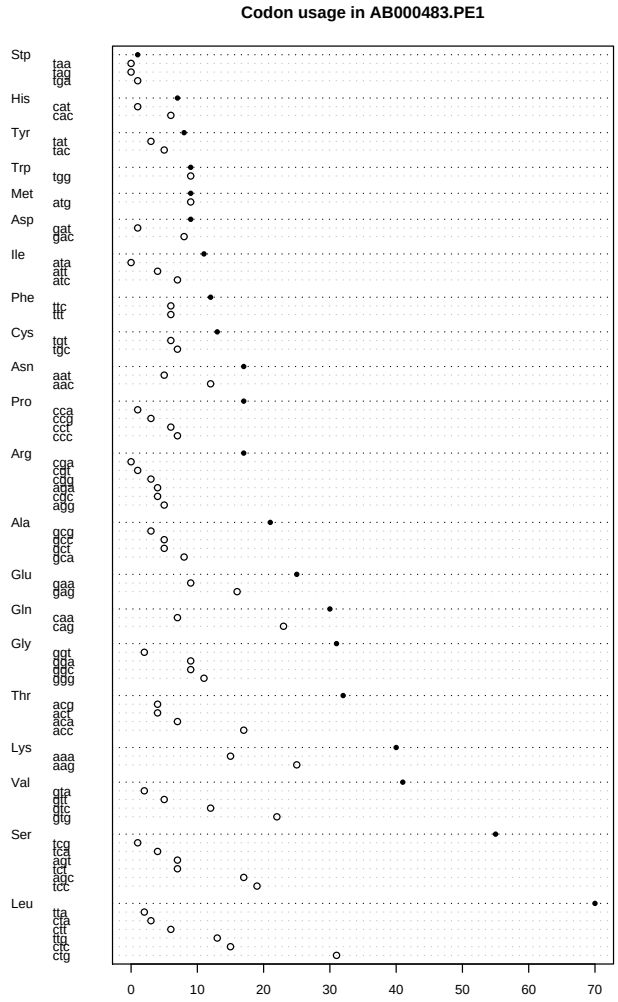
```
basecount <- table(myseq)
myseqname <- getName(list1$req[[1]])
dotchart(basecount, xlim = c(0, max(basecount)), pch = 19,
    main = paste("Base count in", myseqname))
```




```
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))
```



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]   15  854  909  919  987 1248
```

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]  15 909 919
```

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

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

```
 [1]   27  121  152  278  431  437  471  476  477  492  555  618  722  788
[15]  809  885  886  939 1043 1046 1190 1220 1263
```

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.

1.4 Multivariate analyses

1.4.1 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 [19]. 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 1.2 inspired from [3]) with only three proteins having only three amino-acids, so that we can represent exactly on a map the consequences of the metric choice.

```
df <- data.frame(matrix(c(130, 60, 60, 70, 40, 35, 0, 0, 5),
    nrow = 3))
names(df) <- c("Ala", "Val", "Cys")
df
```

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

```
Attaching package: 'xtable'


        The following object(s) are masked _by_ .GlobalEnv :

         string
```

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

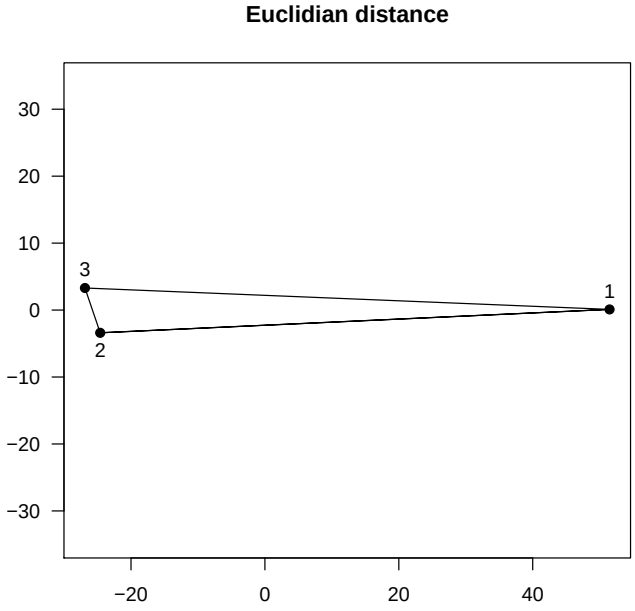
Table 1.2. A very simple example of amino-acid counts in three proteins.

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

$$d^2(i,i') = \sum_{j=1}^{J} (n_{ij} - n_{i'j})^2 \tag{1.1}$$

to visualize this small data set:

```
library(ade4)
pco <- dudi.pco(dist(df), 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(df)
```

```
  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 (1.1) its n_{ij} entries are on average bigger, sending it away from the others. To get rid of this trivial effect, the first obvious idea is to divide counts by protein lengths so as to work with *protein profiles*. The corresponding distance is,

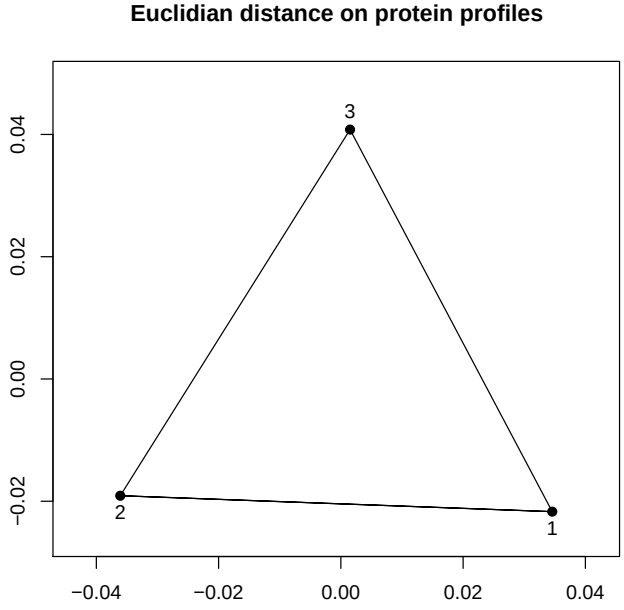
$$d^2(i,i') = \sum_{j=1}^{J} (\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}})^2 \tag{1.2}$$

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

```
df1 <- df/rowSums(df)
df1
```

```
   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(df1), 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)
```

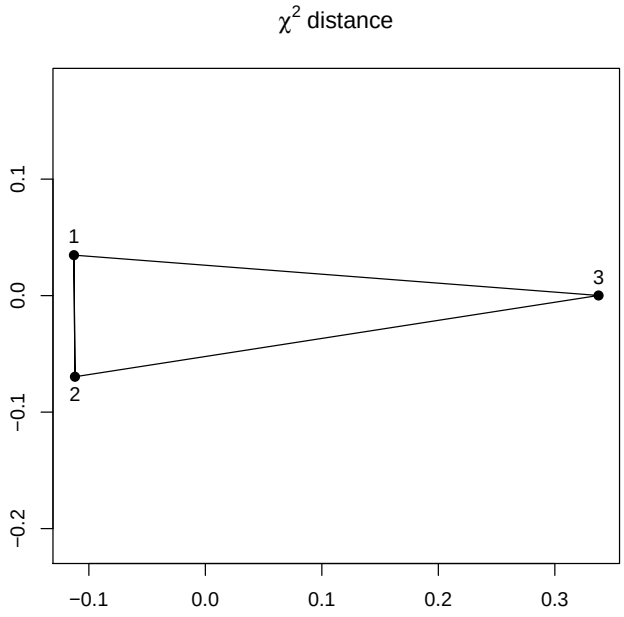


The pattern is now completely different with the three protein equally spaced. This is normal because in terms of relative amino-acid composition they are all differing two-by-two by 5% at the level of two amino-acids only. We have clearly removed the trivial protein size effect, but this is still not 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 (1.2) in that each square is weighted by the inverse of the frequency corresponding to each term,

$$d^2(i,i') = \sum_{j=1}^{J} \frac{1}{n_{\bullet j}} (\frac{n_{ij}}{n_{i\bullet}} - \frac{n_{i'j}}{n_{i'\bullet}})^2 \tag{1.3}$$

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

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

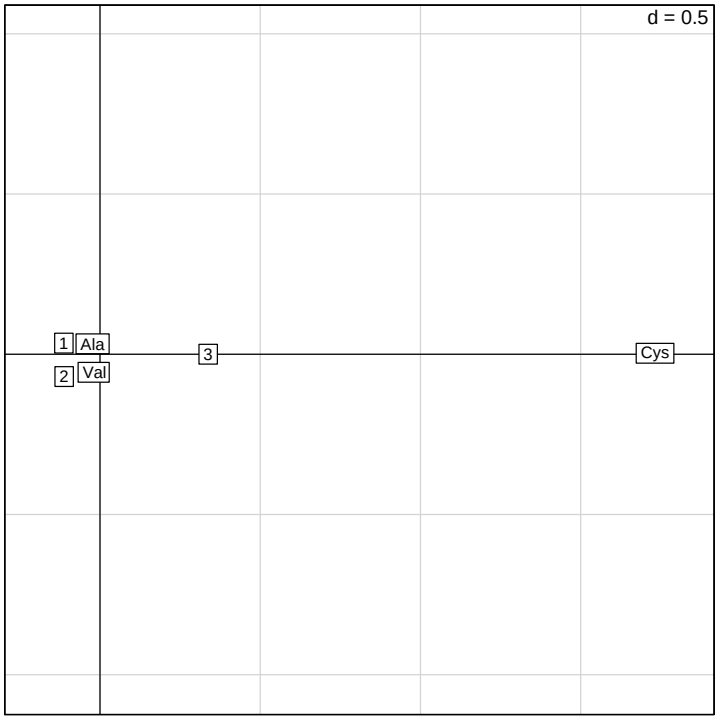


The pattern is completely different with now protein number 3 which is far away from the others because it is enriched in the rare amino-acid `Cys` as compared to others.

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:

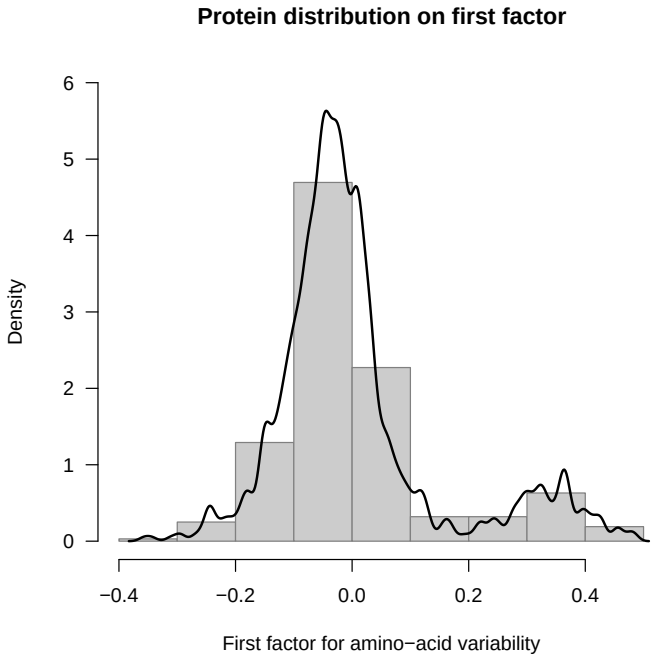
```
scatter(coa, clab.col = 0.8, clab.row = 0.8, posi = "none")
```



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 1.2 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).

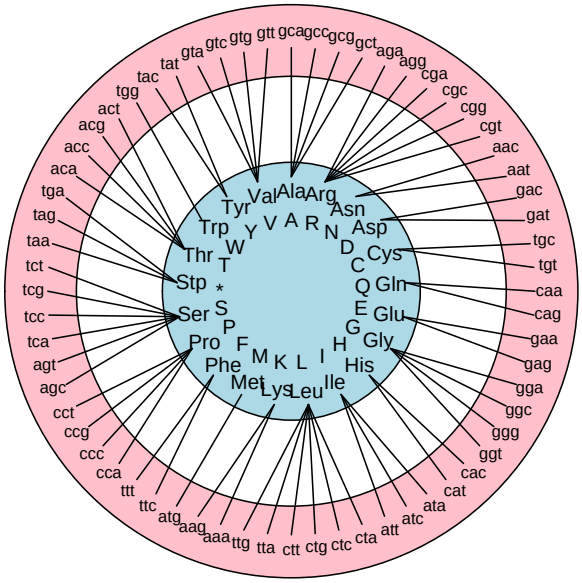
Data are from [16], a sample of the proteome of *Escherichia coli*. According to the title of this paper, the most important factor foe the between-protein variability is hydrophilic - hydrophobic gradient. Let's try to reproduce this assertion :

```
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)
```



1.4.2 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 [13]. The $\frac{K_a}{K_s}$ ratio is used as tool to evaluate selective pressure (see [7] 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:

```
download.file(url = "http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/ortho.fasta",
    destfile = "ortho.fasta")
ortho <- read.alignment("ortho.fasta", format = "fasta")
kaks.ortho <- kaks(ortho)
kaks.ortho$ka/kaks.ortho$ks
```

```
             AK002358.PE1 HSU78678.PE1
HSU78678.PE1          NaN
RNU73525.PE1          NaN          NaN
```

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 [19]. 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 [15], and follow this link `http://pbil.univ-lyon1.fr/members/lobry/repro/jag03/` for on-line reproducibility.

1.5 Releases notes

1.5.1 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:

  in the body of the text (J.R. Lobry, personal communication), or
  wait for the exact complete reference.

A BibTeX entry for LaTeX users is

  @inproceedings{,
    author = {Charif and D. and Lobry and J.R.},
    title = {seqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences r
    booktitle = {NA},
    year = {2005},
    editor = {NA},
    volume = {NA},
    series = {Biological and Medical Physics, Biomedical Engineering},
    pages = {NA},
    address = {NA},
    month = {NA},
    organization = {NA},
    publisher = {Springer Verlag},
    note = {in press},
```

```
}

Orginal article and updates are available in the
../../../library/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.
- A bug as been reported for intensive `kaks()` calls.
- Strict control on ambiguous DNA base alphabet has been relaxed.

1.6 Acknowledgments

Please enter `contibutors()` in your R console.

References

1. Charif, D., Thioulouse, J., Lobry, J.R., Perrière, G.: Online synonymous sodon usage analyses with the ade4 and seqinR packages. Bioinformatics **21** (2005) 545–547. `http://pbil.univ-lyon1.fr/members/lobry/repro/bioinfo04/`.
2. Buckheit, J., Donoho, D.L.: Wavelab and reproducible research. (1995) *In* A. Antoniadis (ed.), *Wavelets and Statistics*, Springer-Verlag, Berlin, New York.
3. Gautier, C: Analyses statistiques et évolution des séquences d'acides nucléiques. PhD thesis (1987), Université Claude Bernard - Lyon I.
4. Hornik, K.: The R FAQ. ISBN 3-900051-08-9 (2005) `http://CRAN.R-project.org/doc/FAQ/`.
5. Leisch, F.: Sweave: Dynamic generation of statistical reports using literate data analysis. Compstat 2002 — Proceedings in Computational Statistics (2002) 575–580 ISBN 3-7908-1517-9.
6. Frank, A.C., Lobry, J.R.: Oriloc: prediction of replication boundaries in unannotated bacterial chromosomes. Bioinformatics **16** (2000) 560–561
7. Hurst, L.D.: The Ka/Ks ratio: diagnosing the form of sequence evolution. Trends Genet. **18** (2002) 486–487.

8. Ihaka, R., Gentleman, R.: R: A Language for Data Analysis and Graphics. J. Comp. Graph. Stat. **3** (1996) 299–314
9. Jukes, T.H., Cantor, C.R.: Evolution of protein molecules. (1969) pp. 21–132. *In* H.N. Munro (ed.), *Mammalian Protein Metabolism*, Academic Press, New York.
10. Keogh, J.: Circular transportation facilitation device. (2001) Australian Patent Office application number *AU 2001100012 A4*. `www.ipmenu.com/archive/AUI_2001100012.pdf`.
11. Kimura, M.: A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J. Mol. Evol. **16** (1980) 111–120.
12. Legendre, P., Desdevises, Y., Bazin, E.: A statistical test for host-parasite coevolution. Syst. Biol. **51** (2002) 217–234.
13. Li, W.-H.: Unbiased estimation of the rates of synonymous and nonsynonymous substitution. J. Mol. Evol. **36** (1993) 96–9.
14. Lobry, J.R.: Life history traits and genome structure: aerobiosis and G+C content in bacteria. Lecture Notes in Computer Sciences **3039** (2004) 679–686. `http://pbil.univ-lyon1.fr/members/lobry/repro/lncs04/`.
15. Lobry, J.R., Chessel, D.: Internal correspondence analysis of codon and amino-acid usage in thermophilic bacteria. J. Appl. Genet. **44** (2003) 235–261. `http://jay.au.poznan.pl/html1/JAG/pdfy/lobry.pdf`
16. Lobry, J.R., Gautier, C.: Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 *Escherichia coli* chromosome-encoded genes. Nucleic Acids Res **22** (1994) 3174–3180. `http://pbil.univ-lyon1.fr/members/lobry/repro/nar94/`
17. Lobry, J.R., Sueoka, N.: Asymmetric directional mutation pressures in bacteria. Genome Biology **3** (2002) research0058.1–research0058.14. `http://genomebiology.com/2002/3/10/research/0058`.
18. Mackiewicz, P., Zakrzewska-Czerwińska, J., Zawilak, A., Dudek, M.R., Cebrat, S.: Where does bacterial replication start? Rules for predicting the *oriC* region. Nucleic Acids Res. **32** (2004) 3781–3791.
19. Perrière, G., Thioulouse, J: Use and misuse of correspondence analysis in codon usage studies. Nucleic Acids Res. **30** (2002) 4548–4555.
20. R Development Core Team: R: A language and environment for statistical computing (2004) ISBN 3-900051-00-3, http://www.R-project.org
21. Rudner, R., Karkas, J.D., Chargaff, E.: Separation of microbial deoxyribonucleic acids into complementary strands. Proc. Natl. Acad. Sci. USA, **63** (1969) 152–159.
22. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. **4** (1984) 406–425.