1

SeqinR 1.0-4: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis

Delphine Charif[1], Jean R. Lobry[1], Leonor Palmeira[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 [9, 21] 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 ???

mirrors available from ??? 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().

1.1.2 About this document

In the terminology of the R project [9, 21], this document is a package *vignette*. The examples given thereafter were run under R version 2.2.1, 2005-12-20 on Wed Mar 22 18:05:06 2006 with Sweave [6]. 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/Library/R/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)
```

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

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

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 [11] on it anyway). At the compilation time of this document there were ??? 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 [18]. 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 [22] 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)
```

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)
```

As from **seqinR** 1.0-4, it is possible to export the table of a genetic code into a LaTeX document, for instance table 1.1 and table 1.2 were automatically generated with the following R code:

```
tablecode(numcode = 3, urn.rna = s2c("TCAG"), latexfile = "code3.tex")
tablecode(numcode = 4, urn.rna = s2c("TCAG"), latexfile = "code4.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	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 1.1. Genetic code number 3: yeast.mitochondrial.

The tables were then inserted in the LaTeX file with:

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

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 [15], 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 [6] 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 **??** here was produced this way, enforcing a complete coherence between the R code example and the table.

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

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")
```

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
```

A more consequent example is given in the fasta file ct.fasta which contains the complete genome of *Chlamydia trachomatis* that was used in [7]. 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 [19] 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")
```

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. The following two sample tests of aligned sequences of this gene (extracted from ParaFit [13]), are distributed along with the **seqinR** package:

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

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 1.1 :

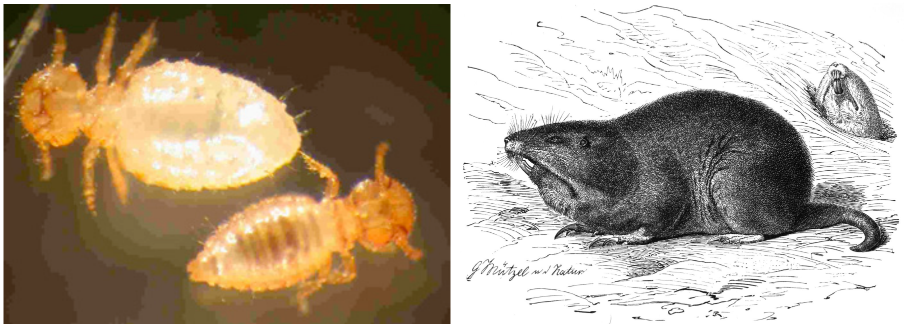

Fig. 1.1. Louse (left) and gopher (right). Images are from the wikipedia (`http://www.wikipedia.org/`). The picture of the chewing louse *Damalinia limbata* found on Angora goats was taken by Fiorella Carnevali (ENEA, Italy). The gopher drawing is from Gustav Mützel, Brehms Tierleben, Small Edition 1927.

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

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 [23] with Jukes and Cantor's correction [10] for multiple substitutions.

```
library(ape)
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 :

```
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("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/louse.names")
g.names <- readLines("http://pbil.univ-lyon1.fr/software/SeqinR/Datasets/gopher.names")
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. Now, your boss is unhappy with working with the Jukes and Cantor's model [10] and wants you to use the Kimura's 2-parameters distance [12] 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. To locate the file on your system, enter the following command:

```
system.file("sequences/louse.fasta", package = "seqinr")
```

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.

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()
```

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 (March 22, 2006) of this document:

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

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:

```
choosebank("off")
```

```
Error in choosebank("off") : 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)
```

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)
```

The list returned by `choosebank()` here means that in the database called `???` at the compilation time of this document there were `???` sequences from `???` species and a total of `???` keywords. The status of the bank was `???`, and the release information was `???`. For specialized databases, some relevant informations are also given in the `details` component, for instance:

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

The previous command has a side-effect that is worth mentioning. 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].

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"))
ntaxa <- numeric(0)
for (i in banks) {
    ntaxa[i] <- as.numeric(choosebank(i)$totspecs)
    closebank()
}
dotchart(log10(ntaxa[order(ntaxa)]), pch = 19, main = "Number of taxa in available databases",
    xlab = "Log10(number of taxa)")
```

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:

[1] There is a very convenient function called `closeAllConnections()` in the R base package if you want to close all open connections at once.

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

```
query("completeCatsCDS", "sp=felis catus et t=cds et no 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)
```

The first sequence that fit our request is `???`, the second one is `???`, 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 1.3.

	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 1.3. Available subsequences in genbank

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

```
completeCatsCDS$call
```

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 et y=2004")
length(ccc2004$req)
```

Hence, there were ??? 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
```

There are therefore `???` 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
```

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 et sp=mycoplasma genitalium", virtual = TRUE)
small$nelem
```

There are then `???` 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]
```

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

Man. How many sequences are available for our species?

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

There are `???` sequences from *Homo sapiens*.

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

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

There are `???` such sequences.

tRNA. How many complete tRNA sequences are available?

[2] which is stored in the `release` component of the object `banknameSocket` and current value is today (March 22, 2006): `banknameSocket$release = ???`.

[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.

```
query("trna", "t=trna et no k=partial", virtual = T)
trna$nelem
```

There are ??? complete tRNA sequences.

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

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

There are ??? sequences published in *Nature* and ??? sequences published in *Science*, so that the winner is *???*.

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

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

There are ??? such sequences.

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

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

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

There are ??? sequences from chloroplast genomes and ??? sequences from mitochondrion genomes, so that the winner is ???.

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(completeCatsCDS$req[[1]])
myseq[1:50]
```

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

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

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("transspliced", "N=AE003734.PE35")
length(transspliced$req)
getName(transspliced$req[[1]])
```

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 ??? (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]
```

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

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]])
```

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.

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

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

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
   myseq[1:50]
   ```

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

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

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
   ```

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

   ```
   (1:30)[ind]
   myseqtcp2 <- myseq[ind]
   ```

   The result should be the same as previously:

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

   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 ???. Let's compare graphically the different base counts in our sequence :

```
basecount <- table(myseq)
myseqname <- getName(completeCatsCDS$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)
```

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)
```

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

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

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 [20]. 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 1.4 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.

```
data(toyaa)
toyaa
```

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

Table 1.4. A very simple example of amino-acid counts in three proteins to be loaded with `data(toyaa)`.

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

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

to visualize this small data set:

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

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

```
rowSums(toyaa)
```

With ??? 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.

```
profile <- toyaa/rowSums(toyaa)
profile
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 (1.2) in that each square is weighted by the inverse of the frequency corresponding to each term,

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

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")
```

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 1.4 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 [17], 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[5]. Let's try to compute the GRAVY score (*i.e.* the Kyte and Doolittle hydropathic index[5]) 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]
```

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, ]
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)
```

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

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 1.5), 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:

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

	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 1.5. Density, distribution function, quantile function and random generation for the predefined distributions under R

```
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
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)
```

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 ($n = 20$) :

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 [14]. The $\frac{K_a}{K_s}$ ratio is used as tool to evaluate selective pressure (see [8] 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
```

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 [20]. 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 [16], 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 1.6 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

Table 1.6. 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)
```

Put all codon names into a single string:

```
c2s(names(toycodon))
```

Transform this string as a vector of characters:

```
s2c(c2s(names(toycodon)))
```

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

```
translate(s2c(c2s(names(toycodon))))
```

Use the three letter code for amino-acid instead:

```
aaa(translate(s2c(c2s(names(toycodon)))))
```

Make this a factor:

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

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)
```

This is exactly the same data set that we used previously (table 1.4) 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")
```

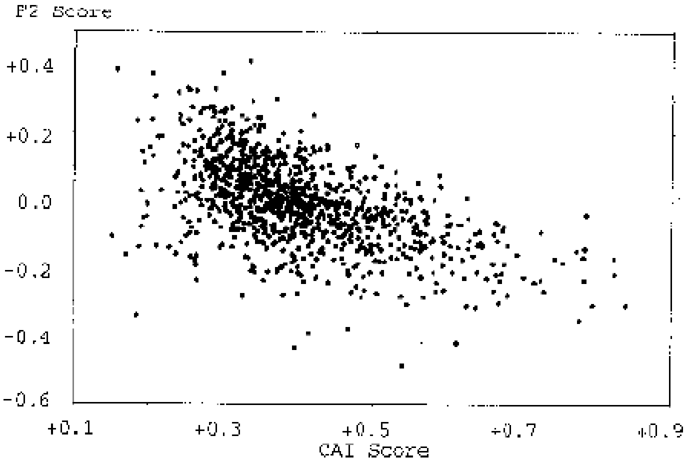

Fig. 1.2. Screenshot of figure 5 from [17]. Each point represents a protein. This was to show the correlation between the codon adaptation index (CAI Score) with the second factor of correspondence analysis at the amino-acid level (F2 Score). Highly expressed genes have a high CAI value.

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.

To illustrate the interest of synonymous codon usage analyses, let's use now a more realistic example. In [17] 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 1.2), that can be reproduced[6] 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)
tmp <- read.table(file = "ftp://pbil.univ-lyon1.fr/pub/datasets/NAR94/ecoli999.cai")
cai <- exp(tmp$V2)
plot(cai, ec.coa$li[, 2], 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 [24]) 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:

[6] the code to reproduce all figures from [17] is available at `http://pbil.univ-lyon1.fr/members/lobry/repro/nar94/`.

```
data(ec999)
class(ec999)
names(ec999)[1:10]
ec999[[1]][1:50]
```

This is to load the data from [17] 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 ???. 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)
class(ec999.uco[[1]])
ec999.uco[[1]]
```

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)
df[1:5, 1:5]
```

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]
```

This is to keep a trace of codon names, just in case we would like to reorder 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
```

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
```

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
```

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
```

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]
kxy <- kde2d(x, y, n = 100)
nlevels <- 25
breaks <- seq(from = min(kxy$z), to = max(kxy$z), length = nlevels +
    1)
col <- cm.colors(nlevels)
image(kxy, breaks = breaks, col = col, xlab = "First synonymous factor",
    ylab = "Second non-synonymous factor", xlim = c(-0.5,
        0.5), ylim = c(-0.3, 0.3), las = 1, main = "The second factor for amino-acid variability is\ncorrelated with gene expressi
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`.

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

1.5 Nonparametric statistics

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.

We will briefly describe two statistics for the measure of dinucleotide over- and under-representation in sequences [25, 26], 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 [27, 28].

1.5.1 Determining dinucleotides over- and under-representation

The *rho* statistic

The ρ statistic (`rho()`), presented in [25], 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. 1.3)

```
rhoseq <- sapply(seq(n), function(x) {
    rho(sample(s2c("acgt"), 6000, rep = TRUE))
})
hist(rhoseq[di, ], freq = FALSE, xlab = "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))
abline(v = 1, lwd = 2, lty = 3)
lines(density(rnorm(1000, mean = mean(rhoseq[di, ]), sd = sqrt(var(rhoseq[di,
    ])))), lwd = 2, lty = 2)
```

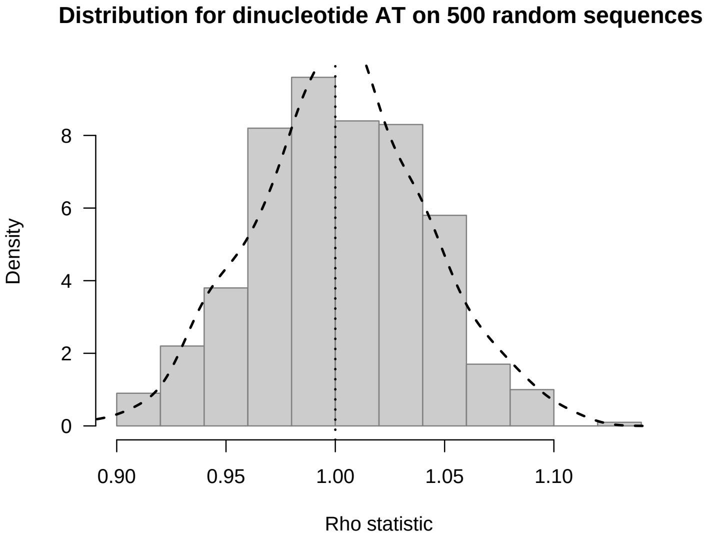

Fig. 1.3. Distribution of the ρ statistic computed on 500 random sequences. The vertical dotted line is centered on 1. The dashed curve draws the fitted normal distribution.

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.

The *zscore* statistic

The *zscore* statistic (`zscore()`) is inspired by the ρ statistic, and is defined so that several different models can be used for the determination of over- and under-representation [26]. It allows for a finer measure of over- and under-representation in sequences, according to the chosen model.

The *zscore* 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.

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. We will first make a request to retrieve all available CDS from this bacteria:

```
choosebank("greview")
query("coli", "sp=escherichia coli et t=cds et no k=partial")
sequence <- getSequence(coli$req[[448]])
```

From the 3684 sequences annotated as CDS in the Genome Reviews database, let's choose one coding sequence: say, for instance, number `448`. We can see that this CDS encodes a maltose O-acetyltransferase protein (`getAnnot(coli$req[[448]],30)`). We will now compare the three following nonparametric statistics:

- the ρ statistic,
- the *zscore* statistic with `base` model,
- and the *zscore* statistic with `codon` model.

```
rhocoli = rho(sequence)
zcolibase = zscore(sequence, mod = "base")
zcolicodon = zscore(sequence, mod = "codon")
par(mfrow = c(1, 3))
plot(rhocoli - 1, ylim = c(-0.5, 0.5), las = 1, ylab = "rho")
plot(zcolibase, ylim = c(-2.5, 2.5), las = 1, ylab = "zscore with base model")
plot(zcolicodon, ylim = c(-2.5, 2.5), las = 1, ylab = "zscore with codon model")
```

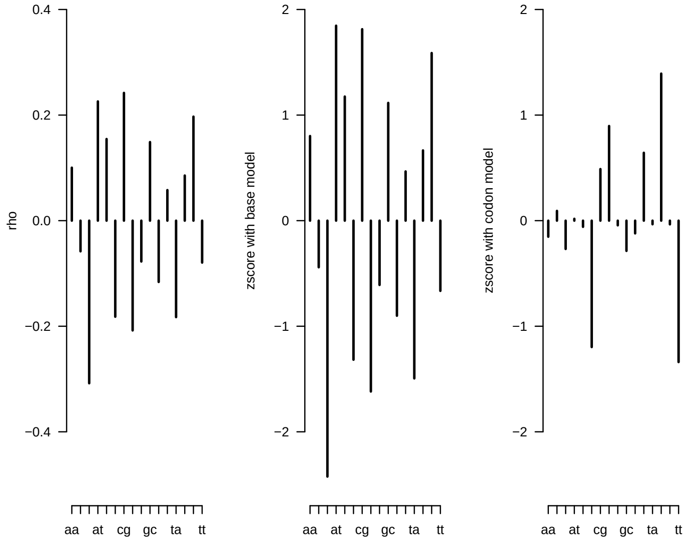

Fig. 1.4. Three different non-parametric statistics (from left to right: ρ, *zscore* with `base` model, *zscore* with `codon` model), computed on the same sequence from *Escherichia coli*. In order to make the figures easily comparable, we substracted 1 to the `rho()` results, so that all 3 statistics are centered on 0.

The first two figures are almost identical: this is due to the way the *zscore* statistic has been built. The statistic computed with the `base` model is a reflection of the ρ statistic. The difference being that the *zscore* 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 figure (*zscore* 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 this last figure, 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...

1.5.2 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 [26].

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 [29]; the sensitivities of the four pyrimidine dinucleotides to UV wavelengths differ and depend on the micro-organism [29]:

	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 1.8. Proportion of dimers formed in the DNA of three bacteria after irradiation with 265 nm UV light. Table adapted from [29].

The hypothesis presented by Singer and Ames [27] 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.* [28] 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 (see Fig. 1.5).

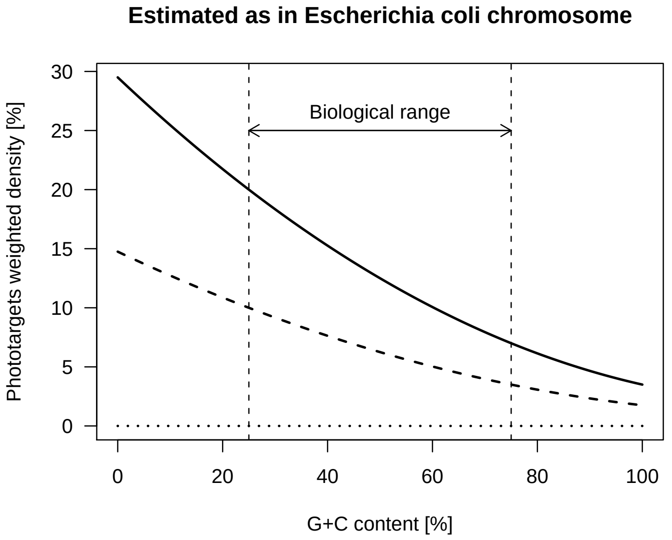

Fig. 1.5. Density of phototargets, weighted by their frequency in the *Escherichia coli* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.59 * f_{tt} + 0.34 * (f_{tc} + f_{ct}) + 0.07 * f_{cc}$ (where f_{xy} is the frequency of 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 (see Fig. 1.6), we can see that this is no longer true...

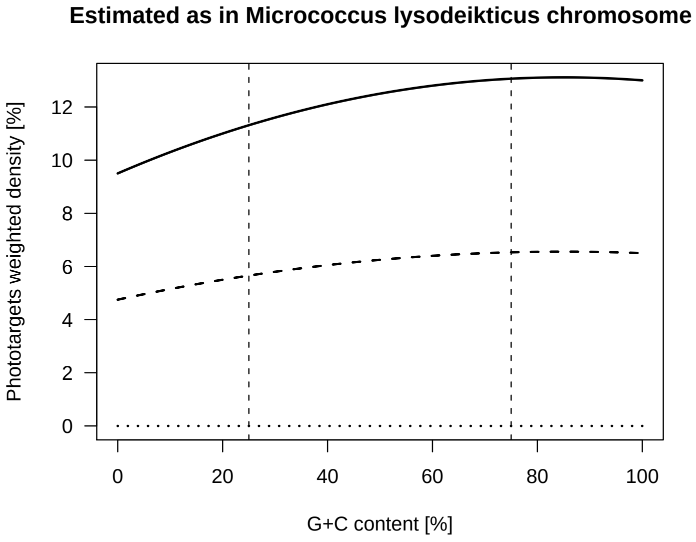

Fig. 1.6. Density of phototargets, weighted by their frequency in the *Micrococcus lysodeikticus* chromosome, and calculated for different G+C contents and for three kinds of random genomes. The weights are as follows: $0.19 * f_{tt} + 0.55 * (f_{tc} + f_{ct}) + 0.26 * f_{cc}$. See previous figure for more details.

These two figures 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?

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 *zscore* with the `base` model on all intergenic sequences, and the mean of the *zscore* with the `codon` model on all CDS:

```
data(dinucl)
```

The results show that there is no systematic under-representation of none of the four pyrimidine dinucleotides (see Fig. 1.7).

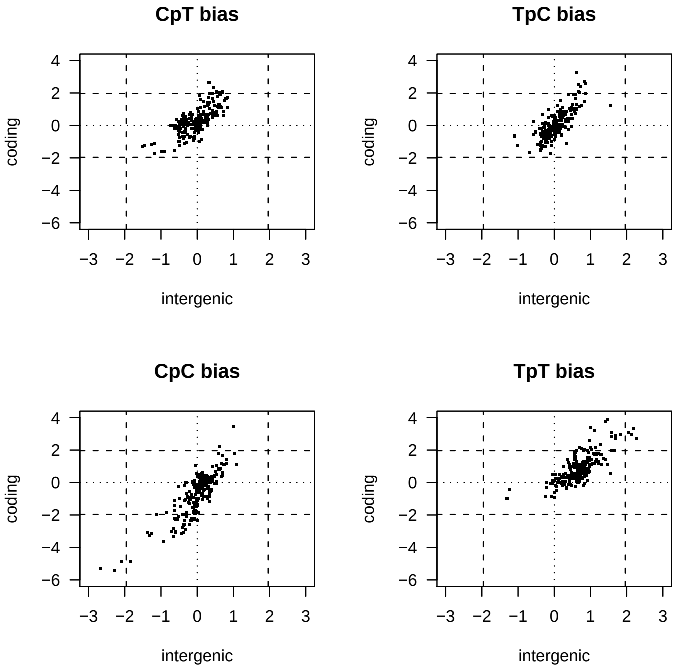

Fig. 1.7. Plot of the mean *zscore* statistics for **intergenic sequences** (x-axis) and for **coding sequences** (y-axis), for each of the four pyrimidine dinucleotides. On each plot, a dot corresponds to the mean of these two statistics in a given prokaryote chromosome. The null x and y axis (dotted 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.

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 *zscore* with the `codon` model on all CDS from each of these three strains (as retrieved from Genome Reviews database on June 16, 2005):

```
data(prochlo)
```

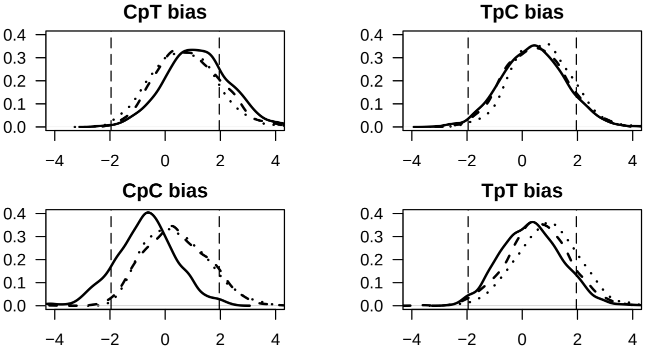

Fig. 1.8. Each figure shows the distributions of the *zscore* in all **coding sequences** corresponding to each of the three strains of *Prochlorococcus marinus*. In each figure, the distribution for the MED4 (a high-light adapted strain) is shown as a solid line; the distribution for the SS120 (a low-light adapted strain) is shown as a dashed line, and the distribution for the MIT 9313 (a low-light adapted strain) is shown as a dotted line. The 5% limits of significance for the standard normal distribution (dashed vertical lines) are plotted as benchmarks.

Figure 1.8 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 [27] no longer stands [26].

1.6 FAQ: Frequently Asked Question

1.6.1 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")
sapply(sapply(ecribo$req, getSequence), GC3)
```

At the amino-acid level, we may get an estimate of the isoelectric point of the proteins this way:

```
sapply(sapply(sapply(ecribo$req, getSequence), getTrans),
    computePI)
```

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)
GC3m(ecribo, 1:10)
```

1.7 Releases notes

1.7.1 release 1.0-4

- The scaling factor $n_{\bullet\bullet}$ was missing in equation 1.3.
- 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 1.1 and table 1.2 here.
- Function `splitseq()` has been entirely rewritten to improve speed.

1.7.2 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")
```
- 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`.

1.8 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. Kyte, J., Doolittle, R.F.: A simple method for displaying the hydropathic character of a protein. J. Mol. Biol. **157** (1982) 105–132.
6. 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.
7. Frank, A.C., Lobry, J.R.: Oriloc: prediction of replication boundaries in unannotated bacterial chromosomes. Bioinformatics **16** (2000) 560–561.
8. Hurst, L.D.: The Ka/Ks ratio: diagnosing the form of sequence evolution. Trends Genet. **18** (2002) 486–487.
9. Ihaka, R., Gentleman, R.: R: A Language for Data Analysis and Graphics. J. Comp. Graph. Stat. **3** (1996) 299–314
10. 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.
11. Keogh, J.: Circular transportation facilitation device. (2001) Australian Patent Office application number *AU 2001100012 A4*. `www.ipmenu.com/archive/AUI_2001100012.pdf`.
12. 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.
13. Legendre, P., Desdevises, Y., Bazin, E.: A statistical test for host-parasite coevolution. Syst. Biol. **51** (2002) 217–234.
14. Li, W.-H.: Unbiased estimation of the rates of synonymous and nonsynonymous substitution. J. Mol. Evol. **36** (1993) 96–9.
15. 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/`.
16. 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`
17. 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/`
18. 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`.
19. 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.
20. Perrière, G., Thioulouse, J: Use and misuse of correspondence analysis in codon usage studies. Nucleic Acids Res. **30** (2002) 4548–4555.
21. R Development Core Team: R: A language and environment for statistical computing (2004) ISBN 3-900051-00-3, http://www.R-project.org
22. Rudner, R., Karkas, J.D., Chargaff, E.: Separation of microbial deoxyribonucleic acids into complementary strands. Proc. Natl. Acad. Sci. USA, **63** (1969) 152–159.

23. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. **4** (1984) 406–425.
24. Sharp, P.M., Li, W.-H.: The codon adaptation index - a measure of directional synonymous codon usage bias, and its potential applications. Nucleic Acids Research **15** (1987) 1281—1295.
25. Karlin, S., Brendel, V.: Chance and Statistical Significance in Protein and DNA Sequence Analysis. Science **257** (1992) 39–49.
26. Palmeira, L., Guéguen L., Lobry, J.R.: UV-targeted dinucleotides are not depleted in light-exposed Prokaryotic genomes. *in prep.*
27. Singer, C.E., Ames, B.N.: Sunlight Ultraviolet and Bacterial DNA Base Ratios. Science **170** (1970) 822–826.
28. Bak, A.L., Atkins, J.F., Singer, C.E., Ames, B.N.: Evolution of DNA Base Compositions in Microorganisms. Science **175** (1972) 1391–1393.
29. Setlow, R. B.: Cyclobutane-Type Pyrimidine Dimers in Polynucleotides. Science **153** (1966) 379–386.