


# Manipuler des données volumineuses : les génomes bactériens

J.R. Lobry & D. Chessel

---

Pour tester l'aptitude de  à manipuler de grands ensembles de données, la fiche propose d'implanter un tableau séquences - codons de taille respectable.

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Les données</b>	<b>3</b>
<b>3</b>	<b>Importer un grand tableau</b>	<b>4</b>
<b>4</b>	<b>Sauver et restaurer un grand tableau</b>	<b>5</b>
<b>5</b>	<b>Taille mémoire</b>	<b>6</b>
<b>6</b>	<b>Tableaux dérivés</b>	<b>6</b>
6.1	Créer le tableau espèces-codons . . . . .	7
6.2	Créer le tableau séquences-acides aminés . . . . .	7
6.3	Créer le tableau espèces-acides aminés . . . . .	8
6.4	Créer le tableau séquences-bases . . . . .	8
6.5	Créer le tableau espèces-bases . . . . .	9
<b>7</b>	<b>Exercices</b>	<b>9</b>
	<b>Références</b>	<b>10</b>

# 1 Introduction

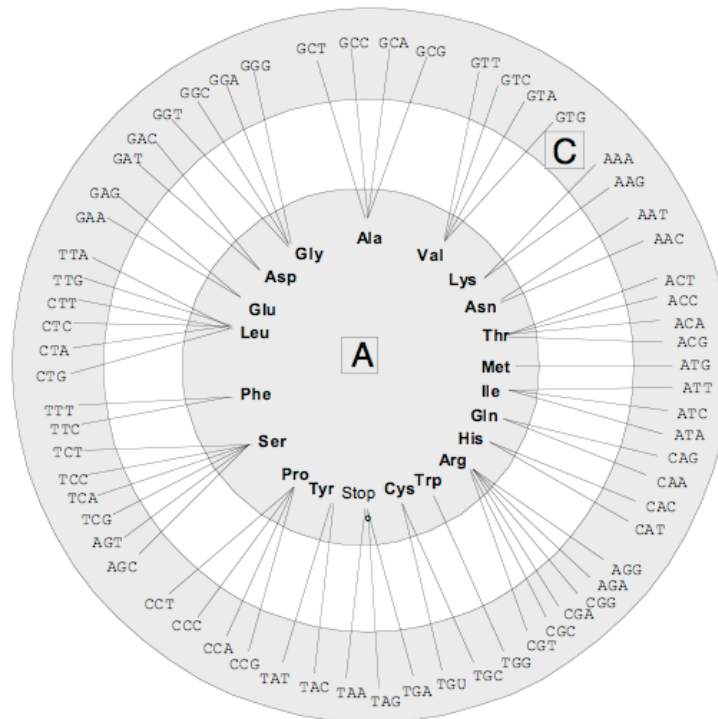
Soit  $C$  l'ensemble des 64 codons possibles dans les séquences codantes,

$$C = \{AAA, AAC, AAG, AAT, \dots, TTT\},$$

et soit  $A$  l'ensemble formé de l'union de l'ensemble vide et des 20 acides aminés des protéine,

$$A = \{\emptyset, Ala, Arg, \dots, Val\},$$

où l'ensemble vide représente les codons non-sens utilisés comme fin de signal de la traduction et les codons non-assignés. Un code génétique est une application surjective de  $C$  dans  $A$ , comme dans l'exemple ci-dessous du code génétique standard.



Les codes génétiques connus sont disponibles dans ce document : <http://seqinr.r-forge.r-project.org/src/appendix/genccodes.pdf>. Pour visualiser rapidement un code génétique sous on peut utiliser la fonction `tablecode()` du paquet `seqinr` [1], par exemple pour le code génétique standard :

```
library(seqinr)
tablecode()
```

---

**Genetic code 1 : standard**


---

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

---

## 2 Les données

Elles sont dans quatre fichiers à récupérer dans <https://pbil.univ-lyon1.fr/R/donnees>. Récupérer ces quatre fichiers dans un le dossier de travail courant :

```
path <- "https://pbil.univ-lyon1.fr/R/donnees"
telecharger <- function(quoi) {
  download.file(url = paste(path, quoi, sep = "/"), destfile = paste(getwd(),
    quoi, sep = ".Platform$file.sep"))
}
telecharger("fracod.txt")
telecharger("fracod_aa.txt")
telecharger("fracod_cod.txt")
telecharger("fracod_esp.txt")
```

Vous devez avoir les quatre fichiers dans votre espace de travail (donné par `getwd()`) pour pouvoir continuer cette fiche de TD :

```
dir(pattern = ".txt")
[1] "fracod_aa.txt" "fracod_cod.txt" "fracod_esp.txt" "fracod.txt"
```

Lire directement `fracod_cod.txt` comme vecteur de chaînes de caractères :

```
codcol <- readLines("fracod_cod.txt")
codcol
[1] "TTT" "TTC" "TTA" "TTG" "TCT" "TCC" "TCA" "TCG" "TAT" "TAC" "TAA" "TAG" "TGT"
[14] "TGC" "TGA" "TGG" "CTT" "CTC" "CTA" "CTG" "CCT" "CCC" "CCA" "CCG" "CAT" "CAC"
[27] "CAA" "CAG" "CGT" "CGC" "CGA" "CGG" "ATT" "ATC" "ATA" "ATG" "ACT" "ACC" "ACA"
[40] "ACG" "AAT" "AAC" "AAA" "AAG" "AGT" "AGC" "AGA" "AGG" "GTT" "GTC" "GTA" "GTG"
[53] "GCT" "GCC" "GCA" "GCG" "GAT" "GAC" "GAA" "GAG" "GGT" "GGC" "GGA" "GGG"
```

On a les 64 codons. Faire avec `fracod_esp.txt` un facteur appelé `codlig` :

```
codlig <- as.factor(readLines("fracod_esp.txt"))
summary(codlig)
AERPECG AQUAECG ARCFUCG BACHDCG BACSUCG BORBUGG BUCAICG CAMJECG CHLMUCG
 2619   1489   2088   3558   3627   772   523   1497   743
CHLPNACG CHLPNCCG CHLPNJCG CHLTRCG DEIRAC1 DEIRAC2 DEIRACP1 DEIRAMP1 ECOLICG
 853     968     982     832   2423   346     35     123   3914
HAENCG   HALSPCG   HALSPP1   HALSPP2   HELPJCG   HELPYCG   METJACG   METTHCG   MYCGECG
 1505   1761   163     291   1372   1392   1516   1646   451
MYCPNCG MYCTUCG   NEIMACG   NMENCG   PSEACG   PYRABCG   PYRHOCG   RICPRCG   SYN3CG
 657     3679   1753   1706   5255   1692   1973   773     2908
THEACCG  THEMACG  TREPACG  UREURCG  VIBCHC1  VIBCHC2  XYLFCAG
 1388    1686     917     560    2365     870    2041
```

Les codes des espèces sont les suivants :

```
AERPECG Aeropyrum pernix K1 complete genome.
AQUAECG Aquifex aeolicus complete genome.
ARCFUCG Archaeoglobus fulgidus complete genome.
BACHDCG Bacillus halodurans C-126, complete genome.
BACSUCG Bacillus subtilis complete genome.
BORBUGG Borrelia burgdorferi complete genome.
BUCAICG Buchnera sp. APS complete genome.
CAMJECG Campylobacter jejuni complete genome.
CHLMUCG Chlamydia muridarum complete genome.
CHLPNACG Chlamydia pneumoniae AR39 complete genome.
CHLPNCCG Chlamydia pneumoniae CWL029 complete genome.
CHLPNJCG Chlamydia pneumoniae J136 complete genome.
CHLTRCG Chlamydia trachomatis complete genome.
DEIRAC1 Deinococcus radiodurans R1 complete chromosome 1.
DEIRAC2 Deinococcus radiodurans R1 complete chromosome 2.
DEIRACP1 Deinococcus radiodurans Plasmid CP1.
DEIRAMP1 Deinococcus radiodurans Plasmid MP1.
ECOLICG Escherichia coli K-12 MG1655.
HAENCG Haemophilus influenzae Rd complete genome.
HALSPCG Halobacterium sp. NRC-1 complete genome.
HALSPP1 Halobacterium sp. NRC-1 plasmid pNRC100, complete sequence.
HALSPP2 Halobacterium sp. NRC-1 plasmid pNRC200 complete genome.
HELPICG Helicobacter pylori, strain J99 complete genome.
HELPICG Helicobacter pylori 26695.
METJACG Methanococcus jannaschii complete genome.
METTHCG Genome of Methanobacterium thermoautotrophicum delta H.
MYCGECG Mycoplasma genitalium complete genome.
MYCPNCG Mycoplasma pneumoniae complete genome.
MYCTUCG Mycobacterium tuberculosis H37Rv complete genome.
NEIMACG Neisseria meningitidis serogroup A strain Z2491 complete genome.
NMENCG Neisseria meningitidis serogroup B strain NC58 complete genome.
PSEACG Pseudomonas aeruginosa PA01, complete genome.
PYRABCG Pyrococcus abyssi complete genome.
PYRHOCG Pyrococcus horikoshii UT3 complete genome.
RICPRCG Rickettsia prowazekii strain Madrid E, complete genome.
SYN3CG Synecocystis PCC6803 complete genome.
THEACCG Thermoplasma acidophilum, complete genome.
THEMACG Thermotoga maritima complete genome.
TREPACG Treponema pallidum complete genome.
UREURCG Ureaplasma urealyticum complete genome.
VIBCHC1 Vibrio cholerae chromosome I, complete chromosome.
VIBCHC2 Vibrio cholerae chromosome II, complete chromosome.
XYLFCAG Xylella fastidiosa, complete genome.
```

### 3 Importer un grand tableau

Il faut savoir que la fonction `read.table()`, qui est très pratique pour les petits jeux de données, n'est pas adaptée aux jeux de données de taille plus conséquente. En effet, `read.table()` fait appel à la fonction `scan()` pour lire les données puis fait une analyse de ce qui a été lu pour essayer de déterminer au mieux les types des colonnes du `data.frame` qui va être construit. Quand on sait ce que l'on veut importer, il est préférable d'utiliser directement la fonction `scan()`. Cette dernière renvoie un vecteur que l'on transforme en matrice puis en `data.frame`.

```
tempsimport <- system.time(fracod <-
  as.data.frame(
    matrix(data = scan(file = "fracod.txt", what = integer(0)), nrow = 67712,
      ncol = 64, byrow = TRUE)))
tempsimport
  user system elapsed
1.400 0.047 1.446
```

Ainsi, pour importer un tableau de 67,712 lignes et 64 colonnes, soit 4,333,568 éléments en tout, nous avons consommé ici 1.4 secondes de CPU utilisateur (ce qui correspond dans la vie réelle à une attente de 1.45 secondes lors de la dernière compilation de ce document). À titre comparatif voici les performances avec la fonction `read.table()` :

```
system.time(fracod <- read.table("fracod.txt"))
  user system elapsed
 1.843   0.060   1.901
```

## 4 Sauver et restaurer un grand tableau

L'idée est qu'après avoir importé des données sous **R**, on aime bien en général les « polir ». Après cette phase de « polissage » il suffit de les sauvegarder comme des objets **R** (en binaire compatible multi-ordinateur) pour pouvoir ultérieurement les recharger très rapidement. Par exemple, dans notre cas l'objet `fracod` est un `data.frame` dont le nom des colonnes n'est pas très explicite :

```
names(fracod)
 [1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8" "V9" "V10" "V11" "V12" "V13"
[14] "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V24" "V25" "V26"
[27] "V27" "V28" "V29" "V30" "V31" "V32" "V33" "V34" "V35" "V36" "V37" "V38" "V39"
[40] "V40" "V41" "V42" "V43" "V44" "V45" "V46" "V47" "V48" "V49" "V50" "V51" "V52"
[53] "V53" "V54" "V55" "V56" "V57" "V58" "V59" "V60" "V61" "V62" "V63" "V64"
```

On décide donc de mieux documenter l'objet :

```
names(fracod) <- readLines("fracod_cod.txt")
names(fracod)
 [1] "TTT" "TTC" "TTA" "TTG" "TCT" "TCC" "TCA" "TCG" "TAT" "TAC" "TAA" "TAG" "TGT"
[14] "TGC" "TGA" "TGG" "CTT" "CTC" "CTA" "CTG" "CCT" "CCC" "CCA" "CCG" "CAT" "CAC"
[27] "CAA" "CAG" "CGT" "CGC" "CGA" "CGG" "ATT" "ATC" "ATA" "ATG" "ACT" "ACC" "ACA"
[40] "ACG" "AAT" "AAC" "AAA" "AAG" "AGT" "AGC" "AGA" "AGG" "GTT" "GTC" "GTA" "GTG"
[53] "GCT" "GCC" "GCA" "GCG" "GAT" "GAC" "GAA" "GAG" "GGT" "GGC" "GGA" "GGG"
```

Maintenant que nous sommes satisfaits, nous sauvegardons l'objet `fracod` en binaire au format XDR (c'est un format utilisé par toutes les variétés de **R**, on peut sauvegarder sous Linux et relire sous Mac ou PC, c'est rapide et portable) :

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

Vous pouvez maintenant quitter votre session **R**, puis relancer **R** pour recharger le jeu de données (n'oubliez pas de sélectionner le bon répertoire de travail avant d'exécuter le code suivant, sinon le fichier ne sera pas trouvé) :

```
tempsXRD <- system.time(load("fracod.Rdata"))
tempsXRD
  user system elapsed
 0.061   0.006   0.067
```

Ainsi, pour restaurer un tableau de 67,712 lignes et 64 colonnes, soit 4,333,568 éléments en tout, correspondant à 22,619,749 observations, nous avons consommé ici 0.06 secondes de CPU utilisateur (ce qui correspond dans la vie réelle à une attente de 0.07 secondes lors de la dernière compilation de ce document). **R** est donc bien capable de manipuler de grands ensembles de données, cependant un couplage avec un SGBD sera préférable pour extraire les informations pertinentes à la volée pour des applications plus conséquentes (il y a des paquets **R** pour cela).

## 5 Taille mémoire

Tous les objets `R` sont placés dans la mémoire vive de l'ordinateur. Une estimation de la place occupée par un objet est donnée par la fonction `object.size()` :

```
taillefracod <- object.size(fracod)
taillefracod
17341496 bytes
```

Ainsi, notre objet `fracod` occupe environ 17,341,496 octets en mémoire (soit 16.54 Mio). La taille mémoire maximum disponible dépend de votre machine et de son système d'exploitation. Si vous dépassez cette capacité vous aurez un message du type :

```
Error: cannot allocate vector of size 33856 Kb
In addition: Warning message:
Reached total allocation of 476Mb: see help(memory.size)
```

## 6 Tableaux dérivés

Faire avec `fracod_aa.txt` un facteur appelé `codaa` :

```
codaa <- factor(readLines("fracod_aa.txt"))
codaa
[1] Phe Phe Leu Leu Ser Ser Ser Ser Tyr Tyr Stp Stp Cys Cys Stp Trp Leu Leu Leu Leu
[21] Pro Pro Pro Pro His His Gln Gln Arg Arg Arg Arg Ile Ile Ile Met Thr Thr Thr Thr
[41] Asn Asn Lys Lys Ser Ser Arg Arg Val Val Val Val Ala Ala Ala Ala Asp Asp Glu Glu
[61] Gly Gly Gly Gly
21 Levels: Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser ... Val
```

Réorganiser le tableau pour que les codons apparaissent par blocs d'acides aminés et vérifier.

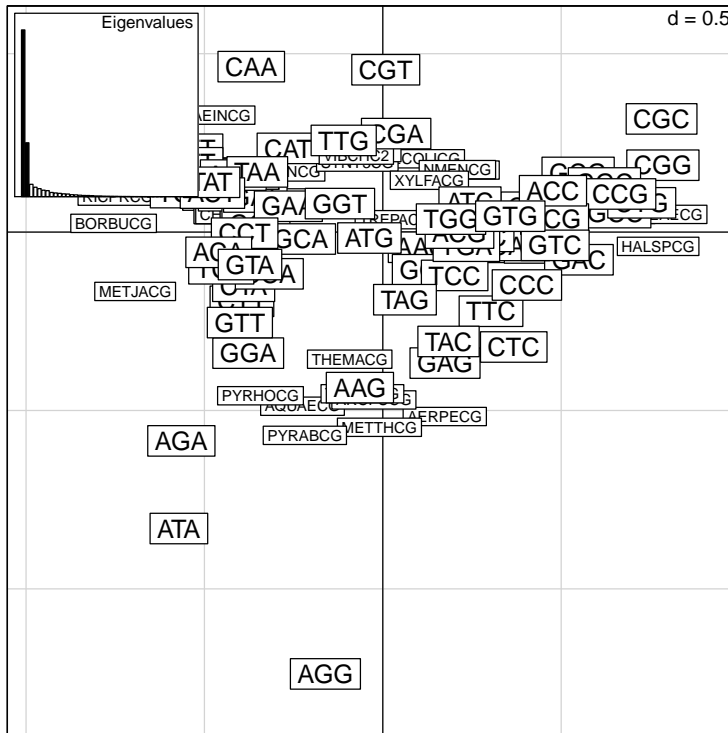
```
frasort <- fracod[, order(codaa)]
codsort <- codaa[order(codaa)]
names(frasort)
[1] "GCT" "GCC" "GCA" "GCG" "CGT" "CGC" "CGA" "CGG" "AGA" "AGG" "AAT" "AAC" "GAT"
[14] "GAC" "TGT" "TGC" "CAA" "CAG" "GAA" "GAG" "GGT" "GGC" "GGA" "GGG" "CAT" "CAC"
[27] "ATT" "ATC" "ATA" "TTA" "TTG" "CTT" "CTC" "CTA" "CTG" "AAA" "AAG" "ATG" "TTT"
[40] "TTC" "CCT" "CCC" "CCA" "CCG" "TCT" "TCC" "TCA" "TCG" "AGT" "AGC" "TAA" "TAG"
[53] "TGA" "ACT" "ACC" "ACA" "ACG" "TGG" "TAT" "TAC" "GTT" "GTC" "GTA" "GTG"
dim(frasort)
[1] 67712 64
sum(frasort)
[1] 22619749
codsort
[1] Ala Ala Ala Ala Arg Arg Arg Arg Arg Arg Asn Asn Asp Asp Cys Cys Gln Gln Glu Glu
[21] Gly Gly Gly Gly His His Ile Ile Ile Leu Leu Leu Leu Leu Leu Lys Lys Met Phe Phe
[41] Pro Pro Pro Pro Ser Ser Ser Ser Ser Ser Stp Stp Stp Thr Thr Thr Thr Trp Tyr Tyr
[61] Val Val Val Val
21 Levels: Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser ... Val
```

## 6.1 Créer le tableau espèces-codons

```

espcod <- aggregate(x = frasort, by = list(espece = codlig), FUN = sum)
rownames(espcod) <- espcod$espece
espcod <- espcod[, -1]
library(ade4)
scatter(dudi.coa(espcod, scann = FALSE, nf = 2))

```



## 6.2 Créer le tableau séquences-acides aminés

```

fbact1 <- function() {
  w <- data.frame(matrix(0, nrow(frasort), length(levels(codsort))))
  names(w) <- levels(codsort)
  row.names(w) <- row.names(frasort)
  for (i in 1:21) {
    a0 <- levels(codsort)[i]
    c0 <- which(codsort == a0)
    w[, i] <- apply(data.frame(frasort[, c0]), 1, sum)
  }
  return(w)
}
fraa <- fbact1()
frasort[1, ]

```

	GCT	GCC	GCA	GCG	CGT	CGC	CGA	CGG	AGA	AGG	AAT	AAC	GAT	GAC	TGT	TGC	CAA	CAG	GAA
AERPECG	2	2	8	3	0	1	0	1	0	3	1	1	1	0	0	0	2	1	7
AERPECG	5	9	1	7	1	2	0	12	3	12	13	7	5	4	11	3	6	3	7
AERPECG	5	5	2	0	5	1	12	1	4	3	3	2	0	0	1	4	5	2	4
AERPECG	3	11	1	8	2	11	3												

```

fraa[1, ]

```

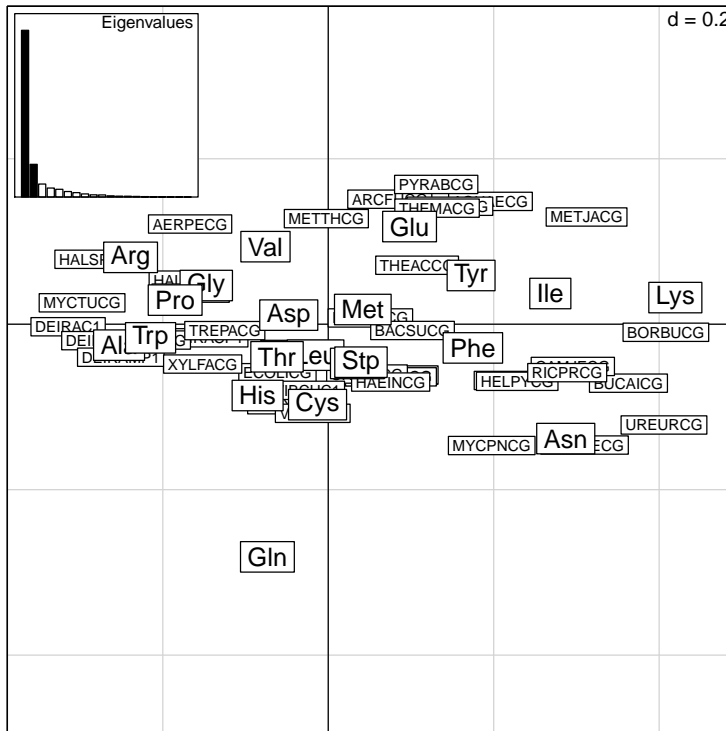
	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Stp	Thr	Trp
AERPECG	15	5	2	1	0	3	12	18	2	27	43	9	7	10	8	25	1	15	3
AERPECG																			
AERPECG	12	24																	

### 6.3 Créer le tableau espèces-acides aminés

```

espaa <- aggregate(x = fraa, by = list(espece = codlig), FUN = sum)
rownames(espaa) <- espaa$espece
espaa <- espaa[, -1]
scatter(dudi.coa(espaa, scann = FALSE, nf = 2))

```



### 6.4 Créer le tableau séquences-bases

```

fbact2 <- function() {
  codons <- names(frasort)
  pos1 <- as.factor(substr(codons, 1, 1))
  pos2 <- as.factor(substr(codons, 2, 2))
  pos3 <- as.factor(substr(codons, 3, 3))
  loc1 <- function(x) {
    x <- as.numeric(x)
    w <- xtabs(x ~ pos1)
    w <- w + xtabs(x ~ pos2)
    w <- w + xtabs(x ~ pos3)
    return(w)
  }
  w <- t(apply(frasort, 1, loc1))
  return(w)
}
frabase <- fbact2()
frabase[1, ]
  A C G T
198 124 150 254
sum(frabase)/3
[1] 22619749

```

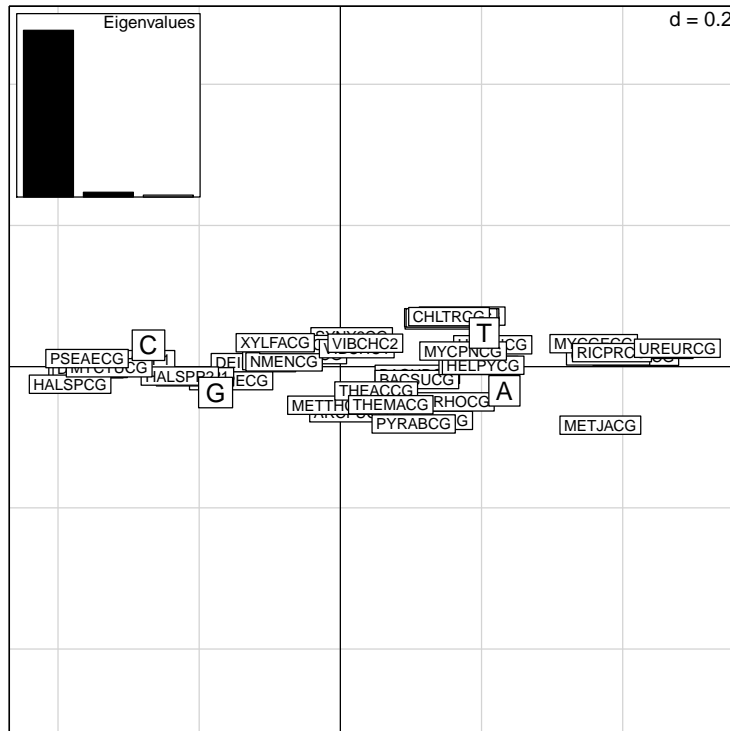


## 6.5 Créer le tableau espèces-bases

```

espbase <- aggregate(x = frabase, by = list(espece = codlig), FUN = sum)
rownames(espbase) <- espbase$espece
espbase <- espbase[, -1]
scatter(dudi.coa(espbase, scann = FALSE, nf = 2))

```



## 7 Exercices

Vérifiez sur ces données le résultat classique [2] selon lequel les organismes qui ont un fort taux de G+C utilisent préférentiellement des acides aminés codés par des codons riches en G+C. Pour cela, vous agrérez les données par espèces avant de représenter l'évolution des fréquences des acides-aminés en fonction du taux de G+C. Question subsidiaire : peut-on dire que le taux de G+C est le facteur majeur de variabilité inter-spécifique au niveau des codons, au niveau des acides-aminés ?

## Références

- [1] D. Charif and J.R. Lobry. SeqinR 1.0-2 : a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis. In H.E. Roman U. Bastolla, M. Porto and M. Vendruscolo, editors, *Structural approaches to sequence evolution : Molecules, networks, populations*, Biological and Medical Physics, Biomedical Engineering, pages 207–232. Springer Verlag, New York, USA, 2007. ISBN 978-3-540-35305-8.
- [2] N. Sueoka. Correlation between base composition of deoxyribonucleic acid and amino acid composition of protein. *Proc. Natl. Acad. Sci. USA*, 47 :1141–1149, 1961.