Institut de recherche
pour le développement

ape : analysis of phylogenetics and evolution

Emmanuel Paradis

St Denis, 6, 7 avril 2009
St Pierre, 9 avril 2009

Programme

I	Introduction sur R
II	Structure et manipulation des données
III	Structures de données dans ape
IV	Entrées/Sorties avec ape
V	Manipulation de données avec ape
VI	Graphiques
VII	Distances ADN
VIII	Estimation phylogénétique
IX	Bipartitions et comparaisons d'arbres
X	Datation moléculaire
XI	Vue d'ensemble sur les méthodes comparatives

I Introduction sur R

Des données aléatoires pour commencer (et se rappeler quelques concepts statistiques) :

```
> rnorm(10)
 [1]  0.7945464  1.2347852 -0.6727558 -0.1248683 -1.0731775
 [6]  0.8186927  1.2732294 -0.3175405 -1.0760822  0.5352055
> a <- rnorm(10)
> a
 [1] -1.2238241 -0.1101836 -0.7159118  0.2910358  0.7198640
 [6] -0.3708338  1.7233652 -0.8137307 -0.4111521 -0.5303543%
```

I Introduction sur R

Des données aléatoires pour commencer (et se rappeler quelques concepts statistiques) :

```
> rnorm(10)
 [1]  0.7945464  1.2347852 -0.6727558 -0.1248683 -1.0731775
 [6]  0.8186927  1.2732294 -0.3175405 -1.0760822  0.5352055
> a <- rnorm(10)
> a
 [1] -1.2238241 -0.1101836 -0.7159118  0.2910358  0.7198640
 [6] -0.3708338  1.7233652 -0.8137307 -0.4111521 -0.5303543%
> b <- rnorm(10)
> mean(a); mean(b)
[1] -0.1441725
[1] 0.3051921
```

Cette différence est-elle statistiquement significative ?

```
> t.test(a, b)

        Welch Two Sample t-test

data:  a and b
t = -1.0257, df = 17.084, p-value = 0.3193
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -1.3733489  0.4746197
sample estimates:
 mean of x  mean of y
-0.1441725  0.3051921
```

Et si le test était unilatéral (*one-tailed*) ?

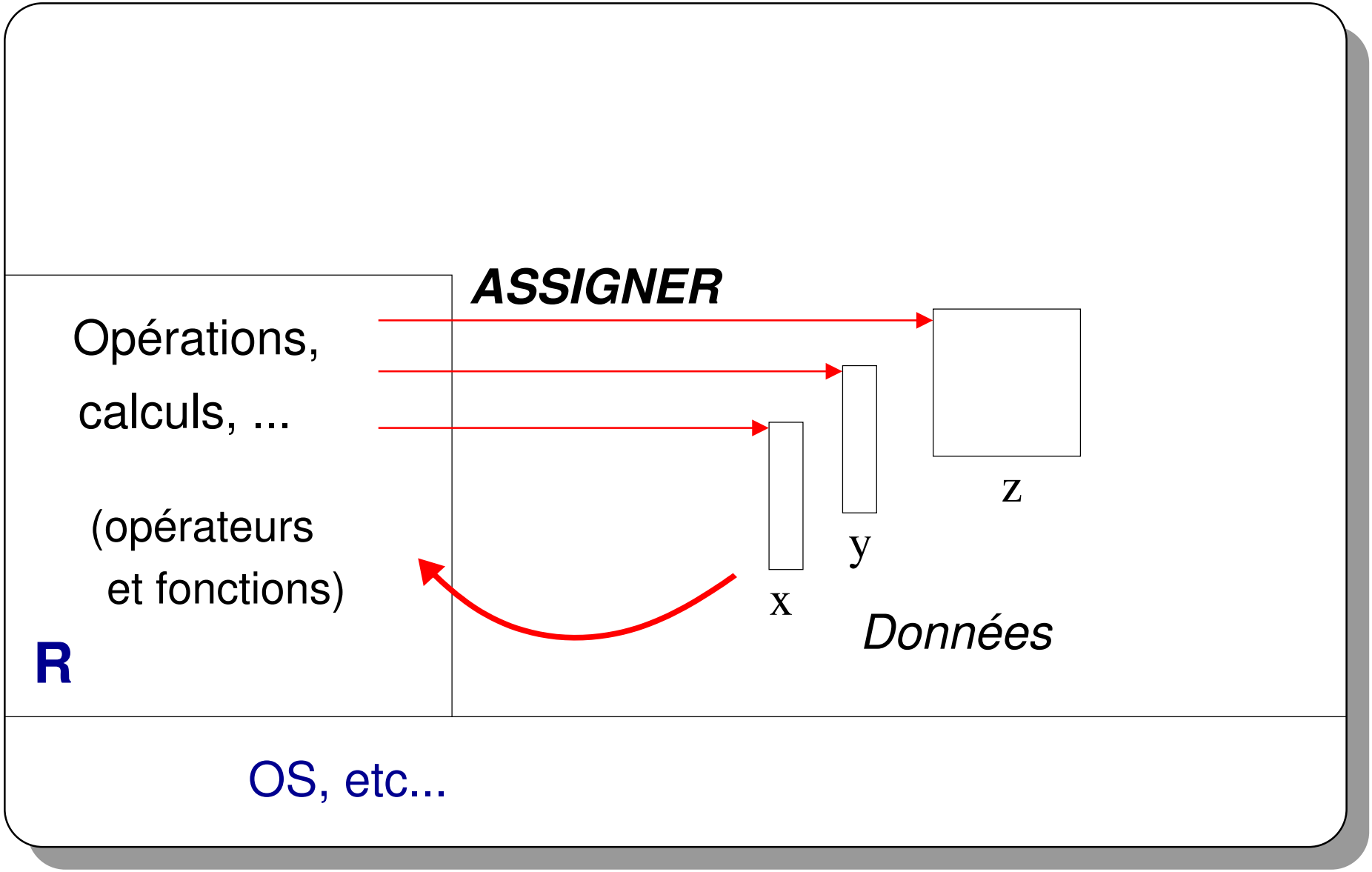
```
> t.test(a, b, alternative = "less")

        Welch Two Sample t-test
```

```
data:  a and b
t = -1.0257, df = 17.084, p-value = 0.1597
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf 0.3125583
sample estimates:
 mean of x  mean of y
-0.1441725  0.3051921
```

Ce second test était-il nécessaire ?

```
> curve(dt(x, df = 17.084), from = -3, to = 3)
> abline(v = -1.0257, col = "blue")
```


ASSIGNER
Opérations,
calculs, ...
(opérateurs
et fonctions)
R
x
y
z
Données
OS, etc...
Mémoire vive (RAM)

Notre premier programme en R : évaluer le risque de première espèce (*type I error rate*) du test de t.

```
p <- numeric(1000)
for (i in 1:1000) {
    x <- rnorm(20)
    y <- rnorm(20)
    p[i] <- t.test(x, y)$p.value
}
hist(p)
sum(p < 0.05)
```

Exercices I

1. Comment évaluer le risque de seconde espèce (*type II error rate*) ?

Quels noms pour les objets ?

Un nom d'objet doit commencer par une lettre.

Les caractères autorisés sont : `A–Z` `a–z` `0–9` `.` `_`

Tous les noms sont permis mais "quotés".

```
> "+1" <- 8
> +1
[1] 1
> get("+1")
[1] 8
```

De même pour les 15 mots réservés du language : `for`, `in`, `if`, `else`, `while`, `next`, `break`, `repeat`, `function`, `NULL`, `NA`, `NaN`, `Inf`, `TRUE` et `FALSE`.

`T` et `F` peuvent être utilisés.

L'usage des lettres accentuées dépend du "locale", donc à éviter.

Le nom d'un objet est limité à 256 caractères, alors n'hésitez pas à utiliser des noms explicites (utilisez la *tab-completion* !).

Les noms des fonctions peuvent être utilisés et cela pose généralement peu de problème (ex : `df`) :

```
> log <- 10.2
> log[1]
[1] 10.2
> log(1)
[1] 0%
```

L'usage des lettres accentuées dépend du "locale", donc à éviter.

Le nom d'un objet est limité à 256 caractères, alors n'hésitez pas à utiliser des noms explicites (utilisez la *tab-completion* !).

Les noms des fonctions peuvent être utilisés et cela pose généralement peu de problème (ex : `df`) :

```
> log <- 10.2
> log[1]
[1] 10.2
> log(1)
[1] 0%
> log <- function(x) paste("x =", x)
> log(1)
[1] "x = 1"
> base::log(1)
[1] 0
```

Comparer deux fonctions de deux packages différents :

```
library(mgcv)
library(gam)
mgcv::gam(....
gam::gam(....
```

Gérer ses scripts de commandes

Un bon éditeur est utile dès qu'on débute avec R. Il permet notamment de :

1. colorer la syntaxe,
2. "allumer" les parenthèses, crochets et accolades,
3. ajouter et éditer des commentaires,
4. éventuellement envoyer des lignes ou des blocs de commandes directement vers R.

Le principal intérêt d'un script de commandes est de pouvoir répéter les analyses (intérêt pratique mais aussi fondamental). Un autre intérêt, mais non moins négligeable, est d'ajouter des commentaires.

Pour répéter des analyses à partir d'un script :

1. copier/coller vers la console ;
2. envoyer les commandes vers R (si possible) ;
3. `source("script_Aedes_morpho_Dakar.R")`
4. `R CMD BATCH script_Aedes_morpho_Dakar.R`

L'avantage de `R CMD BATCH` est que les commandes et les résultats sont dans le même fichier ('script_Aedes_morpho_Dakar.Rout').

Avec `source` ou `R CMD BATCH`, chaque ligne de commentaire doit être précédée par `#`. Une alternative, si le bloc est syntactiquement correct :

```
if (FALSE) {
..........
... bloc exclu de l'exécution
........
}
```

Ajouter des commandes dans un script

1. copier/coller depuis la console ;
2. `savehistory("R_script_today.R")` sauvegarde toutes les commandes de la session en cours sur le disque. Si aucun nom de fichier n'est précisé, il sera nommé '.Rhistory' et sera peut-être caché (et pas forcément associé avec votre éditeur de scripts R).

 Il est vivement conseillé “d’aérer” les opérateurs :

```
x > -1              x>-1
x < -1              x<-1   # :(
```

Ajouter des messages dans un script

Le plus simple : `message("Debut des calculs...")`
Plus intéressant : `cat("n =", n, "\n")`

Le système d'aide

La plupart des fonctions de R ont des options (parfois beaucoup) : consulter l'aide est donc souvent utile.

```
> ?scan # ou help(scan)
```

Les rubriques qui sont utiles (avec l'expérience) : **See Also**, **Examples**, **Value**.

Pour chercher sur son installation personnelle :

```
> help.start()
```

. . . dans un package particulier :

```
> library(help = ape)
```

. . . sur CRAN : "Search", "Task Views"

II Structure et manipulation des données

Vecteur

1
2
3
4
5

"Homo"
"Pan"
"Gorilla"

length (= 5)

mode (= "numeric")

length (= 3)

mode (= "character")

} **Attributs**

length : nombre d'éléments ***mode*** : numeric, character, logical (complex, raw)

Comment construire un vecteur ?

1. Séries régulières : ‘`:`’ `seq(from, to, by)` `rep`

   ```
   rep(1:2, 10)
   rep(1:2, each = 10)
   rep(1:2, each = 2, length.out = 20)
   ```

2. Séries aléatoires : `rloi(n, ...)`
3. Vecteurs “par défaut” : `numeric(2)` `logical(10)` `character(5)`
4. Concaténer des vecteurs : `c(x, y)`, `c(x, y, z)`,...
5. Entrer direct au clavier avec `scan()` (numérique) ou `scan(what = "")`
6. Lecture de fichiers

Quelque soit le mode, une valeur manquante est indiquée `NA` (*not available*) mais est stockée de façon appropriée.

```
> x <- c("NA", NA)
> x
[1] "NA" NA
```

```
> is.na(x)
[1] FALSE  TRUE
```

Les valeurs numériques infinies sont indiquées `Inf` ; `NaN` signifie “not a number”.

```
> -5/0
[1] -Inf
> exp(-5/0)
[1] 0
> Inf + Inf
[1] Inf
> Inf - Inf
[1] NaN
```

Une matrice (***matrix***) est un vecteur arrangé de façon rectangulaire.

Comment construire une matrice ?

1. Avec la fonction `matrix(NA, nrow, ncol, byrow = FALSE)`
2. À partir d’un vecteur : `dim(x) <- c(nr, nc)`, si `length(x) == nr*nc`
3. En joignant des vecteurs avec `rbind` ou `cbind`.

Facteur

Un facteur (***factor***) est un vecteur d'entiers codant une variable qualitative (couleur, . . .). L'attribut `"levels"` spécifie les noms des niveaux. Certains niveaux peuvent ne pas être présents.

Un facteur ordonné (***ordered***) a une hiérarchie dans ses niveaux (ex : TB, B, m, M, TM).

Comment construire un facteur ?

1. Séries régulières : `gl(n, k, n*k)` (*generate levels*)
2. Avec la fonction `factor(x, levels = )`
3. À partir d'un vecteur numérique `x` : `cut(x, breaks)` (*cf.* `?cut pour les détails)`)
4. Lecture de fichiers

 Les facteurs ne peuvent pas être concaténés avec `c(x, y)`

Tableau de données et liste

Un tableau de données (***data frame***) est un ensemble de vecteurs et/ou de facteurs tous de la même longueur.

Comment construire un tableau de données ?

1. Avec la fonction `data.frame`
2. Lecture de fichiers

Une liste (***list***) est un ensemble d'objets quelconques.

Comment construire une liste ?

1. Avec la fonction `list`
2. Lecture de fichiers (avec `scan`)

Le système d'indexation des vecteurs : `[ ]`

Numérique : positif (extraire, modifier et/ou 'allonger') OU négatif (extraire uniquement).

Logique : le vecteur d'indices logiques est éventuellement ***recyclé*** (sans avertissement) pour extraire, modifier et/ou allonger.

Avec les noms (***names*** = vecteur de mode character) ; pour extraire ou modifier.

Extraction et “subsetting” des matrices, tableaux et listes

1. `[, ]` (les 3 systèmes) pour matrices et tableaux mais :
   - allongement impossible,
   - `drop = TRUE` par défaut.

   Il n’y a pas de names mais `colnames` et/ou `rownames` (obligatoires pour les tableaux).
2. Extraction à partir d’un tableau ou d’une liste : `$` (avec noms) `[[` (numérique ou avec noms).
3. Subsetting à partir d’un tableau ou d’une liste : `[` (les 3 systèmes).

   `subset` est une fonction qui permet de faire le même genre d’opération de sélection de lignes et/ou colonnes d’une matrice ou d’un tableau.

Les conversions

R a 118 fonctions `as.`*XXX* pour convertir les objets. Cela peut concerner le mode (`as.numeric`, `as.logical`, ...), le type de données (`as.data.frame`, `as.vector`, ...), ou même d'autres objets (`as.formula`, `as.expression`, ...).

 R effectue parfois des conversions implicites (***coercions***) :

```
> "0" == 0
[1] TRUE
> "0" == FALSE
[1] FALSE
> 0 == FALSE
[1] TRUE
> if (2) print("OK")
[1] "OK"
```

Manipulation des chaînes de caractères

```
> paste("site", 1:3)
[1] "site 1" "site 2" "site 3"
> paste("site", 1:3, sep = "")
[1] "site1" "site2" "site3"
> paste(letters[24:26], 1:3, sep = " < ")
[1] "x < 1" "y < 2" "z < 3"
> paste(1:5, "x", letters[1:2], sep = "+")
[1] "1+x+a" "2+x+b" "3+x+a" "4+x+b" "5+x+a"
> paste(LETTERS, collapse = "")
[1] "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
> paste(1:5, collapse = "-")
[1] "1-2-3-4-5"
> paste("x", 1:5, sep = "*", collapse = "; ")
[1] "x*1; x*2; x*3; x*4; x*5"
```

Substitution de caractères :

```
> sub("a", "?", c("aaa", "aba"))
[1] "?aa" "?ba"
> gsub("a", "?", c("aaa", "aba"))
[1] "???" "?b?"
```

Recherche de “pattern” :

```
> grep("A", LETTERS)
[1] 1
> grep("A", LETTERS, value = TRUE)
[1] "A"
> grep("a", LETTERS, value = TRUE)
character(0)
> grep("a", LETTERS, value = TRUE, ignore.case = TRUE)
[1] "A"
```

```
> x <- 1:3
> names(x) <- c("Pan paniscus", "Pan troglodytes",
+               "Homo sapiens")
> x["^Pan "]
<NA>
  NA
> sel <- grep("^Pan ", names(x))
> x[sel]
   Pan paniscus Pan troglodytes
              1                2
```

III Structures de données dans ape

Trois choses à savoir sur ape. . .

- proche de la logique de R $\Rightarrow$ intégration avec les méthodes statistiques classiques
- s'adresse à un continuum d'utilisateurs du niveau "bas" au niveau "haut"
- "évolutif" : nouvelles méthodes, corrections de bugs, site web

. . . et bien d'autres choses `http://ape.mpl.ird.fr/`

Les cousins de ape (sur CRAN) :

- ade4, apTreeshape, seqinr, phangorn
  connections étroites (conversions de données, peu de redondances dans les méthodes)
- geiger, laser, picante, ...
  utilisent les structures de données de ape

Christoph Heibl développe des packages qui complètent ape :

`http://www.christophheibl.de/Rpackages.html`

Arbres phylogénétiques

```
> library(ape)
> tr <- rtree(20)
> tr

Phylogenetic tree with 20 tips and 19 internal nodes.

Tip labels:
        t12, t18, t2, t13, t9, t8, ...

Rooted; includes branch lengths.
> plot(tr)



> is.rooted(tr)
[1] TRUE
> is.binary.tree(tr)
[1] TRUE
```

```
> is.ultrametric(tr)
[1] FALSE
> tu <- rcoal(20)
> is.ultrametric(tu)
[1] TRUE
```

Listes d'arbres :

```
> TR <- rmtree(10, 20)
> TR
10 phylogenetic trees

> plot(TR)
```

Séquences ADN

Les séquences d'ADN sont stockées dans un format spécial (dans la RAM ; pas sur le disque).

```
> x <- c("a", "c", "g", "t")
> x <- as.DNAbin(x)
> x
1 DNA sequence in binary format.
> base.freq(x)
   a    c    g    t
0.25 0.25 0.25 0.25
```

Ce qui n'est pas dans ape :

- Séquences d'acides aminés : voir seqinr et phangorn
- Données alléliques : adegenet et pegas (en développement*)
- SNPs, microarray, etc : voir Bioconductor†
- Réseaux : phangorn et pegas (`mst` dans ape)

*`http://ape.mpl.ird.fr/pegas/pegas.html`

†`http://www.bioconductor.org/`

IV Entrées/Sorties avec ape

`read.tree` lit des arbres au format Newick.

`read.nexus` idem mais au format Nexus.

`read.dna` lit des séquences d'ADN aux formats Phylip, FASTA, ou Clustal (voir l'option `format`).

`read.GenBank` lit des séquences d'ADN depuis GenBank à partir des numéros d'accession.

`write.tree` écrit des arbres au format Newick.

`write.nexus` idem au format Nexus.

`write.dna` écrit des séquences d'ADN aux formats Phylip ou FASTA.

V Manipulation de données avec ape

```
> TR[1:5]
5 phylogenetic trees

> TR[[1]]

Phylogenetic tree with 20 tips and 19 internal nodes.

Tip labels:
        t2, t16, t17, t1, t19, t10, ...

Rooted; includes branch lengths.
```

Fonctions spéciales pour les arbres : `root`, `unroot`, `drop.tip`, `extract.clade`, `rotate`, `ladderize`, `multi2di`, `di2multi`.

Deux fonctions (très flexibles) pour manipuler les étiquettes (*labels*) : `makeLabel`, `makeNodeLabel`.

```
> data(woodmouse)
> summary(woodmouse)
15 DNA sequences in binary format stored in a matrix.

All sequences of same length: 965

Labels: No305 No304 No306 No0906S No0908S No0909S ...

Base composition:
    a     c     g     t
0.307 0.261 0.126 0.306
> dim(woodmouse)
[1]  15 965
> dim(woodmouse[, 1:300])
[1]  15 300
> dim(woodmouse[, c(TRUE, TRUE, FALSE)])
[1]  15 644
> dim(woodmouse[, !c(TRUE, TRUE, FALSE)])
[1]  15 321
```

Les séquences non-alignées (depuis un fichier FASTA ou GenBank) sont stockées dans une liste.

Les séquences alignées (depuis un fichier Phylip ou Clustal) sont stockées dans une matrice.

`as.matrix` convertit une liste de séquences en matrice si toutes les séquences sont de même longueur.

Exercices V

1. Récupérer depuis GenBank les séquences ayant pour numéros d'accession DQ082330-DQ082374. Examiner ces données et voir si vous pouvez calculer les distances directement.
2. Aligner ces séquences aved Clustal (X or W), puis lire les séquences alignées dans R.
3. Écrire éventuellement ces commandes dans un script, et les répéter avec AF006387–AF006459 avec seulement les numéros impairs.
4. Mettre les deux jeux de données dans une matrice unique.

VI Graphiques

`plot` est une fonction ***générique***.

Beaucoup d'options : voir `?plot.phylo`

```
> example(plot)
```

Étiquettage et annotation : `add.scale.bar`, `axisPhylo`, `nodelabels`, `tiplabels`, `edgelabels` avec beaucoup d'options.

```
> example(nodelabels)
```

Exploration de grands arbres :

```
> zoom(tree.carni, 1:10)
> g1 <- grep("Felis", tree.carni$tip.label)
> g2 <- grep("Panthera", tree.carni$tip.label)
> zoom(tree.carni, list(g1, g2))
```

VII Distances ADN

```
> args(dist.dna)
function (x, model = "K80", variance = FALSE, gamma = FALSE,
    pairwise.deletion = FALSE, base.freq = NULL,
    as.matrix = FALSE)
```

Voir ausi la fonction `dist` pour le calcul de distances classiques.

Le triangle inférieur de la matrice distances est retourné (sauf si la distance est asymétrique), et est stocké dans un vecteur.

Quelle est l'influence des données manquantes ?

```
> d <- dist.dna(woodmouse)
> dp <- dist.dna(woodmouse, pairwise.deletion = TRUE)
> summary(d)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.002201 0.009979 0.013350 0.013120 0.016740 0.022410
> summary(dp)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.002086 0.010520 0.013690 0.013350 0.016920 0.022280
> cor(d, dp)
[1] 0.9879206
> plot(d, dp)
> abline(a = 0, b = 1, lty = 3)
```

Quelle est l'influence des substitutions multiples ?

```
> djc <- dist.dna(woodmouse, "JC69")
> dr <- dist.dna(woodmouse, "raw")
> plot(djc, dr)
> abline(a = 0, b = 1, lty = 3)
```

Exercices VII

1. Analyser les trois jeux de données préparés précédemment avec des distances évolutives.

VIII Estimation phylogénétique

Méthodes basées sur les distances

Trois méthodes sont disponibles dans ape : `nj`, `bionj`, `fastme.bal` (+ `fastme.ols`)

```
> tr.k80 <- nj(d)
> plot(tr.k80)
```

Un modèle de substitution différent donne-il une topologie différente ?

```
> tr.jc69 <- nj(djc)
> dist.topo(tr.k80, tr.jc69)
[1] 2
> plot(tc <- consensus(tr.k80, tr.jc69))
```

Bootstrap :

```
> args(boot.phylo)
function (phy, x, FUN, B = 100, block = 1, trees = FALSE)
> f <- function(x) nj(dist.dna(x))
> bp <- boot.phylo(tr.k80, woodmouse, f)
> bp
 [1] 100  25  48  49  54  47  74  65  86  89  88 100  48
```

```
> par(mfcol = c(1, 2))
> plot(tr.k80)
> nodelabels(bp)
> plot(tc)
```

Que donne la méthode FastME ?

```
> tr.me.k80 <- fastme.bal(d)
> dist.topo(tr.k80, tr.me.k80)
[1] 6
> dist.topo(root(tr.k80, "No305"), tr.me.k80)
[1] 0
```

Les topologies sont identiques.

FastME est plus sophistiquée que NJ :

```
> args(fastme.bal)
function (X, nni = TRUE, spr = TRUE, tbr = TRUE)
```

Maximum de vraisemblance

ape a une interface avec PhyML :

```
> args(phymltest)
function (seqfile, format = "interleaved", itree = NULL,
    exclude = NULL, execname, path2exec = NULL)
```

Les arbres estimés par PhyML peuvent ensuite être lus dans R avec `read.tree`.

Le bootstrap doit être fait depuis PhyML ; l'arbre est ensuite lu avec `read.tree` et ses valeurs de bootstrap dans l'élément `node.label`.

phangorn a une fonction `pml` assez proche de ce que fait PhyML.

Autres Méthodes

`phangorn` a la fonction `parsimony` pour les séquences d'ADN (méthodes de Sankoff et de Fitch).

`pml` fonctionne aussi avec les séquences protéiques (quatre modèles disponibles : WAG, JTT, Dayhoff et LG).

Les méthodes bayésiennes peuvent être implémentées avec les outils existants dans R (calcul de vraisemblance, simulations d'arbre aléatoires, ...).

Exercices VIII

1. Continuer l'analyse des trois jeux de données.

IX Bipartitions et comparaisons d'arbres

Comparer deux arbres : `all.equal(tr1, tr2)` ; pour ne comparer que les topologies étiquettées `all.equal(tr1, tr2, FALSE)`, ou que les topologies `all.equal(tr1, tr2, FALSE, FALSE)`

`dist.topo` calcule la distance topologique entre deux arbres.

`prop.part(tr)` retourne une liste avec tous les clades observés dans `tr` ; si c'est une liste d'arbres alors tous les clades observés sont retournés avec leurs fréquences.

`boot.phylo` utilise `prop.part`.

`consensus` calcule l'arbre de consensus (strict par défaut) à partir d'une liste d'arbres ; l'option `p = 0.5` permet de calculer le "majority-rule consensus tree".

`cophenetic` calcule la matrice de distances patristiques.

`branching.times` calcule les temps de branchement.

X Datation moléculaire

Deux méthodes sont disponibles dans ape :

1. Vraisemblance pénalisée (*penalized likelihood, PL*) :

```
> args(chronopl)
function (phy, lambda, age.min = 1, age.max = NULL,
      node = "root", S = 1, tol = 1e-08, CV = FALSE,
      eval.max = 500, iter.max = 500, ...)
```

2. Longueur moyenne des chemins *mean path lengths* :

```
> args(chronoMPL)
function (phy, se = TRUE, test = TRUE)
```

Exercices X

1. Continuer l'analyse des trois jeux de données.

XI Vue d'ensemble sur les méthodes comparatives

```
> source("tree_macro.R")
```

Il peut être nécessaire de redimensionner la fenêtre graphique.