Fiche de commande R

Adrien Taudiere

9 octobre 2015

Version avancée

Table des matières

1 Formalisme

a et **b** sont des matrices (*matrix() pour 2 dimensions ou bien des *array() pour plus de 2 dimensions)
X et **Y** sont des vecteurs (*vector())
x et **y** sont des nombres
liste est une liste c'est à dire une groupe d'objet pouvant être différents (*list())
fact est un facteur (*factor())
df est un data.frame (liste d'éléments de même longueur ; *data.frame())
| correspond à 'ou' dans R et dans cette fiche
m est un modèle (linéaire par exemple)
obj est un objet quelconques
T = TRUE ; **F** = FALSE
élt élément
ddl degré de liberté
µ moyenne

2 Terminal Linux

2.1 Naviguer dans ses dossiers

cd ouvrir un dossier
cd .. retourner au dossier parent
ls lister les fichiers du dossier. L'option -A permet de lister également les fichiers cachés, l'option -L donne plus d'information sur les fichiers.
PWD connaître le dossier courant
mkdir créer un nouveau dossier

2.2 Connaître et agir sur les fichiers

rm supprimer un fichier. Pour supprimer un dossier il faut utiliser l'option -R (pour récursive).
mv déplacer un fichier
cp copier un fichier. Pour copier un dossier il faut utiliser l'option -R
ln créer un lien vers un fichier

2.3 Connaître et agir sur les programmes

kill pid tue le processus en donnant son pid. **xkill** tue le processus sur lequel vous cliquez avec la souris après cette commande.
htop Liste l'ensemble des processus en cours
./ Lancer un programme

2.4 Connaître et agir sur sa machine

lscpu et **hwloc -ls** Informe sur l'architecture de la machine

2.5 Connaître et agir sur le contenu des fichiers

head et **tail** donne les premières (dernières) ligne du fichier
wc compte le nombre de mots/lignes (option -L)
cat concaténer deux fichiers
grep recherche de motif (*cf* aussi **sed** et **awk**)

3 Quelques fonctions utiles de 'dialogue' avec R

R contient plusieurs types d'objets qui appartiennent à des classes3.4 et des modes3.4 différents. Pour plus d'informations : le site R advanced d'Hadley Wickham.

3.1 Répertoire de travail

Le répertoire de travail est différent de les environnements7.6 de R.
*getwd() demande le répertoire de travail actuel
*setwd("/home/nomrépértoire") changement de répertoire de travail

3.2 Aide et infos sur un élément

*?élément / *apropos(élément) / *??élément / *help.start(browser="firefox") AIDE respectivement sur un élément connu *e.g.* une fonction (?mean) ou d'autres mots clé *e.g.* *?Syntax donne l'aide de la syntaxe sous R, sur un élément ou une partie de l'élément (apropos cherche dans la liste de recherche de votre session actuelle,?? cherche sur internet y compris dans des packages non activés sur votre session) et sur l'aide R web en générale
*??? Nécessite le package *sos. Ouvre la liste de résultat d'une recherche très complète dans le navigateur. Voire aussi le site documentation R, site très utile pour naviguer dans l'ensemble des fonctions disponibles sous R.

3.3 Aide et infos sur une fonction

*getAnywhere(fonction) Recherche la fonction plus en profondeur dans les code que les points d'interrogation. Permet par exemple de retrouver des fonctions internes non documentées.
*args(fonction) Liste les arguments d'une fonction

3.4 Objet

*class(obj) donne la classe de l'objet obj. 5 classes principales : *vector, *factor, *matrix (un seul type de colonne), *dataframe (plusieurs types de colonne) et *list.
*mode(obj) donne le mode de l'objet obj. 5 modes principaux : *numeric, *character, *logical, *complex et *raw.
*typeof(obj) donne le mode de stockage interne de l'objet obj. Pour connaître le mode de stockage de chaque colonne d'un dataframe : **sapply(obj, typeof)**
*str(obj) donne la structure interne de l'objet obj de façon concise, très utile dans le cas d'objet complexe
*attributes(obj) donne les différentes structures de l'objet obj (*cf* aussi str())
*summary(obj) résumé de l'objet selon sa classe
*head(a, x) affiche les x premières lignes de la matrice
*tail(a, x) affiche les x dernières lignes de la matrice
*rm("obj") efface l'objet OBJ
*rm(list=ls()) nettoie toutes les variables
*methods(class=class(obj)) Liste des méthodes qui concerne la classe de l'objet obj, cela permet de connaitre facilement toutes les méthodes qui questionnent ou changent la classe d'un objet (respectivement via **methods(is)** et **methods(as)**)
*apropos("color") liste des objets et des fonctions comprenant le mot "color". **apropos**("lm") liste les objets et fonctions commençant par "lm".
*attach(a) l'objet ou l'ensemble d'objet ainsi attacher (ici la matrice a) est ajouté à la recherche de R lorsqu'il recherche une variable. Ainsi les objets peuvent être appeler en donnant leur noms (*e.g.* en donnant le nom d'une colonne de a on obtient directement la colonne plutôt qu'avoir à faire a$nomcolonne). Peut causer de nombreux problème, préférer la fonction with.
*mget(

4 Gestion de données

4.1 input

*read.table("nomdefichier", header=T or F, dec=".", sep="/t") ouvre le fichier, si le fichier contient des nom de colonne header=true, ici la décimale est donnée par une virgule et la séparation par une **tabulation** (dont le sigle est /t). Pour un **espace**, le sigle est //. On peut remplacer "nomfichier" par "clipboard" pour ouvrir le tableau copié dans le clipboard. Voire aussi les fonctions *read.csv et *read.delim.

*source("nomdefichier.text") importe et exécute les commandes contenues dans le fichier

4.2 information et gestion de données

*NA Donnée manquantes (Not Available)

*is.na(a) Demande si la matrice (ou un autre objet) contient des NA

*na.fail(a)

*na.omit(X) / *na.exclude(X) / **a[!is.na(a)]** supprime les informations manquantes (trois méthodes quasiment équivalentes, attention à vérifier le résultat de ces fonctions parfois capricieuses)

*identical(obj1, obj2) Test si deux objets sont identiques

4.2.1 sur le fichier

*nrow(nomdefichier) nombre de lignes du fichier

*rownames(a)<-rownames(b) on renomme les lignes de a comme les lignes de b

*ncol(nomdefichier) nombre de colonnes du fichier

*colnames(a)<-colnames(b) on renomme les colonnes de a comme les colonnes de b

*dim(a) donne les dimensions du vecteur | d'une matrice

*edit(objet) ouvre le tableau du fichier, attention refermer avant de continuer (*cf* aussi *View() et *fix())

4.2.2 sur un vecteur

X<-*seq(from=1, to=100, by=1) formation de vecteur allant de 1 à 100 avec un pas de 1

X<-*seq(from=1, to=100, lenght=100) formation de vecteur allant de 1 à 100 avec 100 valeurs/nombres équidistant(e)s

X<-*c(a, b, c, d) affecte les valeurs au vecteur X , *c= concatener

Y<-*c(1* :10) affecte valeurs entière de 1 à 10

*Y[10] affiche la dixième valeur du vecteur Y

*Y<2 affiche true pour les valeur de Y <2 et false pour les autres

*Y[Y<2] affiche les valeurs de Y qui sont <2

*is.vector(Y) R répond si Y est un vecteur | non

*a[1 :X,-y] renvoie la matrice a avec les lignes de 1 à X et toutes les colonnes excepté la numéro y.

*X[X==0] indexation. Renvoie les valeurs de X pour X égale à 0. Pour X différent de 0, remplacer *== par *!=

X<-1^(0* :5) donne au vecteur X les valeurs 1^0, 1^1 ... 1^5

*sort(X, decreasing=T) ordonne les éléments de X (un vecteur ou un facteur), decreasing permet de choisir le sens du tri

*order(a) donne les coordonnées du plus petit élément de a puis du deuxième, ect. order() peut aussi être utiliser pour un ou plusieurs vecteurs

*rev(X) renverse les éléments du vecteur. Pour une matrice : *t(a)

*rank(X) donne les rangs des éléments de X

*diff(X) différences entre les élts consécutifs de X

*is.numeric() et *as.numeric() demande ou change le vecteur en numérique. Attention à vérifier votre vecteur après l'utilisation de cette fonction qui modifie parfois les valeurs de vos données

*cut(X, breaks=c(x,y)) divise le vecteur numérique en un facteur selon des points de coupure définient par l'argument breaks (ici pour les valeurs x et y).

4.2.3 sur une matrice

a<-*as.matrix(X) *cf* methode(as)

*is.matrix(a) Est ce que a est une matrice? Cf > methode(is)

a<-*matrix(y, nrow=5, ncol=2) transforme vecteur y en matrice a avec 5 lignes et 2 colonnes (le nombre de colonne est facultatif)

*a[2,2] affiche l'élément de a de ligne 2 et de colonne 2

a$nomcol1 = a[, nomcol1]=X affiche les éléments de a de la colonne nommé nomcol1

a[,-1] affiche toutes les colonnes de a sauf la colonne 1

a[c(-10 ;-22),] affiche toutes les lignes sauf la ligne 10 et la ligne 23

b<- matrix(0, nrow=3, ncol=2) création d'une matice b à 3 lignes et 2 colonnes *puis* b[,1]<-*scan() permet de saisir les valeurs de la colonne 1 de la matrice b, scan() peut également être utilisé seul

a[a[,3]==1,] affiche les valeurs de a pour lesquelles la colonne 3 est égale à 1

*t(a) transpose la matrice a (lignes deviennent colonnes et réciproquement)

*subset(a, X!=0, select=) donne une sous division de la matrice a pour laquelle la 1ère colonne est différente de 0, select permet de sélectionner les colonnes concernées. Fonctionne aussi sur les dataframes.

*diag(10) matrice identité de dimension 10*10

*diag(x, nrow, ncol) donne une matrice de dimension nrow*ncol diagonalisée par la valeur de x

a%%b multiplication matricielle

*eigen(a) valeurs propres et vecteurs propres de la matrice a

*sweep**(a, margin= 1, STATS = rowSums(a), FUN="/")** divise (on peut modifier la fonction FUN) chaque valeur de la matrice a par la moyenne par ligne (on peut modifier rowSums par une autre fonction), margin donne la dimension (*e.g.* 1=ligne et 2= colonne) sur laquelle s'applique la fonction

4.2.4 sur un dataframe

*as.data.frame(a) change la matrice a en un data.frame *cf* > methode(as)

*is.data.frame(a) est ce que a est un data frame? *cf* > methode(is)

*summary(as.dataframe(a)) donne le résumé statistique (min, max, µ...) par colonne

*split(df, fact) sépare le dataframe df selon les modalités de fact

*stack(df) transforme un dataframe à plusieurs colonnes en un dataframe avec seulement 2 colonnes. Les valeurs du dataframe obtenu correspond aux noms des variables dans le premier dataframe. Essayez, c'est très simple. *unstack est la fonction inverse.

*with(df, sum(X)) évalue une fonction à l'intérieur d'un data.frame. Permet de ne pas nommer en entier les noms des variables. Dans cette exemple on calcul la somme de la colonne nommée X dans le data.frame.

4.2.5 sur un facteur

Rmq : Les facteurs sont codés en numériques.

*gl(x, y) créer un vecteur à x niveaux (modalités) en répétant chaque niveau y fois.
*as.factor change un vecteur (de valeur quantitative) en un facteur (avec des catégories qualitatives)
*nlevels(fact) donne le nombre de modalités du facteur fact
*levels(fact) donne les modalités du facteur fact
*table(fact) retourne une table des valeurs
*ordered(fact) transforme un facteur en facteur ordonné
*as.integer(fact) transforme le facteur en entier. Pour obliger des nombres à être entiers il faut faire **c(1*L, 2L)** au lieu de **c(1,2)**.
*as.numeric(fact) transforme le facteur en numérique
*split(X,fact) sépare le vecteur X selon les modalités de fact
*by(X,fact,median) applique la fonction médiane par modalité fact sur le vecteur X (*cf* la fonction tapply)
*table(fact1, fact2) donne la matrice d'association entre les deux facteurs

4.2.6 sur une liste

*list(obj1,obj2) produit une liste avec les objets
*as.list(a) transforme la matrice a en une liste
*list[1] extraction de la 1ère valeur de la 1ère composante de la liste
*list[[x]] extraction de la xéme composante de la liste.
*list$**nomcomp** extraction de la composante nomcomp de la liste
*unlist(list) transformation de la liste en vecteur

4.2.7 caractères

*as.character(x) change les éléments de x en caractères
*nchar(X) nombre de caractères des éléments de X
*paste(X,Y) concaténation des vecteurs de caractères
*substr(X,2,3) extraction des caractères 2 et 3 de X (élt par élt)

4.3 output

*save(nomdefichier)
*write.table(a, "C :\\ mat") enregistre la matrice a sous le nom mat (peut ensuite être ouvert avec excel)
*postscript("C :\\ Rplot.eps", onefile=F) sauvegarde les images que l'on appelle en format eps jusqu'à écrire *dev.off() pour finir les sauvegardes
*bmp(filename = "C :\\ Rplot.bmp", width = 480, height = 480, units = "px", pointsize = 12, bg = "white", restoreConsole = TRUE) Sauvegarde une image en format bmp. On peut remplacer bmp par *jpeg(), *tiff() ou encore *png()

5 Statistiques

5.1 Stats descriptives

*sum(X) somme des valeurs de la colonne 1 de la matrice a, à noter que ici X correspond à un vecteur Rmq : True=1, False=0 permet par exemple sum(X.obs>X.permute) où le résultat sera égale au nombre de fois ou X.obs est supérieur au valeur du vecteur X.permute (*cf* permatswap). On peut également utiliser *rowSums(a) et *colSums(a)
*cumsum(X), *cumprod(X), *cummin(X), *cummax(X) somme, produit, minimum ou maximum cumulé du vecteur X
*mean(X) moyenne de X
*ave(X, fact) moyenne de X par modalité de fact. *cf* tapply pour plus d'option de calcul par modalité
*sd(X) écart type de X
*var(X) variance de X
*max(X) valeur max de X pmax est la version vectorisée de max
*min(X) valeur min de X pmin est la version vectorisée de max
*summary(X) quartiles, médiane, moyenne et valeurs extrêmes du vecteur
*length(X) taille d'un vecteur
*median(X) médiane du vecteur X
*quantile(X, probs=c(0.25, 0.5, 0.75, 1)) quantile déterminé par les probabilités de l'argument "probs". Ici, il s'agit de quartile
*IQR(X) écart interquartile de X
*which.max(X) & *which.min(X) détermine la localisation *i.e.* l'indice, de la valeur maximale (ou minimale) du vecteur
*diff(X) donne les différences entre les valeurs d'un vecteurs qui se suivent

5.2 Tests statistiques

Remarque : Si je rejette H0, je la rejette avec un risque de p-value de me tromper (p-value< 5% –> je rejette H0 et p-value>5% je ne rejette pas H0)

*chisq.test(a) chisq.test(x, y) test du chi2
*shapiro.test(x) test de shapiro teste la normalité d'un vecteur
*ks.test(x) test de Kolmogorov-Smirnov, H0 : X et Y sont des variables issues d'une même distribution continue.
*var.test(x, y) test bilatéral entre deux vecteurs : comparaison de 2 variances (test de Fisher et Snedecor)
*bartlett.test() test d'homogénéité ou d'égalité des variances quand il y a + de 2 variances
*t.test(X, Y, var.equal=T) Comparaison de 2 µ éch indépendants (test de Student) = test paramétrique [condit° : X1 et X2 indépendantes, suivent loi Normale (shapiro.test) et que les variances soit égales(var.test)] ajout de var.equal= T or F
*t.test(X, Y, var.equal=F, alternative = "less", "two.sided", "greater", paired=T) Comparaison de 2 µ éch appariés (test de Student) = test paramétrique [condit° : différences : X1 - X2 suit loi Normale]. Ici var.equal=F donc variance inégale donc correction de Welch automatique, alternative permet test bilatéral ou unilatéral, paired dit si les variables sont appariées et quand paired=T pas besoins de mettre de var.equal
*wilcox.test(var.equal=F, alternative="less", "two.sided", "greater", paired=F) Comparaison de 2 µ éch indépendant (test de Wilcoxon) = test non-paramétrique [condit° : X1 et X2 indépendantes], autres arguments *cf* t.test
*wilcox.test(paired=T) Comparaison de 2 µ éch appariés (test de Wilcoxon) = test non-paramétrique avec X1 et X2 appariés
*cor.test(x, y, method="spearman" | "pearson" | "kendall", alternative= "less", "two.sided", "greater") test la significativité du coefficient de corrélation [condition : linéarité + binomialité + obs. indépendantes], alternative *cf* t.test
*cor(X, Y) donne le coefficient de corrélation entre X et Y. **cor**(df) donne la matrice de corrélation entre les va-

riables du dataframe df

*kruskal.test() test anova non paramétrique ; faire un modèle linéaire avant

*anova(m) test anova paramétrique , m : résultat de la fonction *lm ! [condit° : normalité des résidus, homogénéité des variances (barlett | student)] ; faire lm avant, *puis* *summary(annova(m)) après le test anova pour afficher le tableau de sortie]

5.3 Modèles linéaires

m<-*lm(x~y) donne un modèle linéaire **Y X1+X2** la valeur yi expliquée par x1i et x2i **Y X1*X2 = Y X1+X2+X1 :X2** la valeur yi expliquée par x1i, x2i et l'interaction x1i*x2i **Y X1%in%X2** la valeur yi expliquée par x1i nidé, imbriqué, dans x2i **Y | X2** la valeur yi expliquée par x1i conditionnellement à x2i **Y -1+X** la valeur yi expliquée par x1i sans moyenne générale (*i.e.* sans intercept) **Y I(X1+X2)** la valeur yi expliquée par x1i+x2i, I() permet d'écrire la formule arithmétique sans que R comprenne la même chose que Y X1+X2

*bptest(m) Breutsch Pagan test H0 : homoscédasticité des variances des résidus (| des observations) ; library(lmtest)

*hmctest(m) Harrison-McCabe test H0 : les résidus suivent des lois normales de même variance ; library(lmtest)

*dwtest(m) Durbin Watson test H0 : indépendance des résidus (| des obs.) ; library(lmtest)

*shapiro.test(m$residuals) test de shapiro H0 : les résidus suivent des lois normales

y <-data.frame(temps=c(15, 19)) *puis* *predict(m, newdata=y) on veut calculer les valeurs au temps t = 15 et au temps t = 19 à partir des données, donne les valeurs prédites de y suivant le modèle m

*summary(m) donne estimation de a et b, écart-type, pvalue, du modèle linéaire et parfois RÂ², [condit° : residus indép et suivent N]

*m$residuals donne les résidus du modèle linéaire, on peut tester la normalité de ces résidus : *shapiro.test(modèle linéaire$residuals)

*AIC(m) donne l'AIC d'un modèle calculé selon AIC=déviance+2*np avec np le nombre de paramètres du modèle

*glm(Y X1+X2, family="gaussian", "poisson", "binomial" ...) modèle linéaire généralisé

5.4 Tirage d'une distribution

Remarque : Toutes les fonctions commençant par r peuvent être utilisées en remplaçant la lettre r (Tirage aléatoire de x valeurs) par d (fonction de densité), p (fonction de probabilités cumulés) et q (valeur des quantiles). Il existe d'autres distributions dont : Poisson (*rpois), Gamma (*rgamma), Beta (*rbeta), Student et Fischer-Snedecor si 2 ddl (*rt), Uniform (*runif), Wilcox (*rwilcox), Logistique (*rlogis), Lognormal (*rlnorm)...

*rnorm(x, 0, 1) donne x valeurs tirées au hasard d'une loi normale centrée réduite suivant µ=0 et var=1

*rbinom(x, 90, 0.5) donne x valeurs tirées au hasard d'une loi binomiale à 90 tirages (n) et une proba (p) de 0.5

*rchisq(x, 5) donne x valeurs tirées au hasard d'une loi du chi2 de ddl=5

*rexp(x, rate=1) donne x valeurs tirées au hasard d'une loi exponentielle de taux 1

*sample(X) permutation aléatoire des élément du vecteur x. L'argument REPLACE permet de choisir un tirage avec ous sans remise.

*jitter(X) rajoute du bruit à une variable

6 Graphique

6.1 Généralité

*layout(a) divise la page graphique selon la matrice a en donnant des poids de taille aux col (widths) et au lignes (heights) *puis* *layout.show(layout(a))

*save.image() sauvegarde une image

*jpeg(filename = "mongraphique.jpg", width = 480, height = 480, units = "px", res = NA, quality = 75) *puis* *plot() *puis* *points() *puis* *dev.off() enregistre le graphique sous le nom choisi ; quality est en pourcentage de qualité (75% par défaut)

*X11() ouvre une nouvelle fenêtre graphique

*locator() donne les coordonnées des points sur lesquels vous cliquez sur un graphique

6.2 diagramme de dispersion

*plot(X ~Y, option cf plus bas) affiche le diagramme de dispersion de X en fonction de la Y

*abline(h=x) ajoute sur la fenêtre graphique une ligne horizontal (v=x pour une ligne vertical) d'ordonné x

*abline(a=3, b=2) ajoute sur la fenêtre graphique une ligne d'équation y=ax+b ici y=3x+2

*abline(m) ajoute la droite de régression du modèle linéaire

*lines(X ~Y) rajoute une ligne sur le graphique courant

*legend(x, y, legend="ma légende") rajoute une légende sur le graphique courant

*text(coord x, coord y,"point moyen") nommer un point | mettre un mot sur le graphique

*symbol(X, Y) rajoute des symboles sur le graphique courant

*points(abscisse, ordonnée) pour rajouter des points sur un même graphique

*matplot(a,b) plot des colonnes de a en fonction des colonnes de b

*coplot(Y X1 | X2) plot de y en fonction de X1 conditionnellement à X2

*polygones(X, Y) rajoute des polygones sur le graphique courant

*pairs(a) représente chaque combinaison de variable issue de la matrice a.

6.3 autres graphiques

*boxplot(x) boites à moustaches (médiane, quartiles, limites de la distribution sans les outliers (dépend de la variance), valeurs dans les choux)

*barplot(a, besides=T, add=T) si besides =T : barre cumulé ; add=T : ajout au précédent plot

*dotchart(X) Cleveland dotchart, alternative au barplot

*hist(X, col=#'6caractères et 2 chiffres d'opacité de 0 à 99', xlim=c(2, 5), ylim=(15, 17), add=T, br=10 , freq=F) donne l'histogramme d'un vecteur de couleur choisie sur l'intervalle choisie, add=T signifie que l'on ajoute le diagramme sur celui déjà existant, br (break) donne le nombre de séparations à représenter sur l'histogramme (ici 10, par défault utilise la formule de Sturges pour calculer le nombre de classes), freq=F pour obtenir les fréquences relatives plutôt que les fréquences absolues

*plot.3D() représentation tri-variées, peut également être représenté par la fonction plot normale en définissant la taille des points proportionnelle à la troisième variable (cex=var3 ou cex=log(var3))
*mosaicplot(a) graphique en mosaïque, très utile pour représenter des tables de contingence

6.4 gestion des couleurs

*rainbow(x) donne x valeurs de couleurs dans l'arc en ciel
*heat.colors(x) donne x valeurs de couleurs chaudes
*cm.colors(x) donne x valeurs de couleurs froides
*rgb(0,0,1,0.5) donne une couleur avec le premier chiffre (entre 0 et 1) donne la quantité de rouge, le deuxième la quantité de vert, le troisème de bleu et le dernier donne l'opacité. Ici il s'agit d'un bleu moyennement transparent
*colors() liste des couleurs : '6 caractères et 2 chiffres d'opacité de 0 à 99'
*grep("blue",colors(), value=T) liste de toutes les couleurs comportant le terme blue
*tclvalue(tcl('tk_choseColor')) permet de sélectionner le code d'une couleur via le gestionnaire Windows
*vec2col(fact, x, nompalette) pour un facteur fact, attribue le nombre x de couleur de la palette (http ://colorbrewer2.org) ; package *MSG

6.5 Personnalisation des graphiques

Pour connaitre la plupart des paramètres graphiques : taper *?par
*lty= **1, 2 ...** type de ligne (continue, pointillée ...)
*pch= **1, 2, "*", ...** type de point
*type= **l, p, b, n** trace des lignes (l), des points (p), les deux (b pour both) ou rien (n pour none)
*par (mfrow = c(2, 3)) divise en 2 lignes et 3 colonnes la fenêtre de l'histogramme
*cex = 0.8 multiplie par 0.8 la taille (des polices, des points ...) de la figure
*col= **"blue"** / *bg= **"blue"** / *col.main= **"blue"** / *col.lab= **"blue"** couleur (ici bleu) du symbole (col), du remplissage (bg), du titre (col.main) et des titres des axes (col.lab)
*xlim= c(x,y) / *ylim= c(x,y) contraint les valeurs limites du graphique, utile dans le ces d'ajout successif de graphiques sur une même figure
*main= titre *xlab= labx *ylab= laby titre respectivement du graphique, de l'axe des x et de l'axe des y
*las= **0 ; 1 ; 2 ou 3** donne le style de l'étiquette par rapport à l'axe (0 :parallèle ; 1 :horizontal ; 2 :perpendiculaire ; 3 :vertical)
*lwd= **x** largeur du trait
*mar= **c(bottom, left, top, right)** définit la taille des marges
*grid(x,y) rajoute une grille avec x rectangles sur axes des x et y sur l'axe des y.

6.6 Galeries graphique sur internet

Galerie de graphiques par packages (pour 90 packages)
R graphical Manual : galerie très (trop ?) complète

7 'Programmation'

7.1 boucle

*for**(i in 1 : x) { commandes}** formation d'une boucle de pas 1 qui va de 1 au nombre x et qui applique les commandes à chaque boucle
res <- vector(NA, x) for (i in 1 : x) {res[i] <- f(x) } stockage résultats à chaque tour de boucle dans le vecteur *res*. Notez l'initialisation du vecteur res à la bonne taille qui permet de gagner en performance.
*breaks stoppe une boucle (for ou while)
*next passe à la prochaine itération d'une boucle (for ou while)
*foreach(i in 1 : x) %do% f(x[i]) boucle plus rapide que la boucle FOR. Utiliser %dopar% pour la version parralélisée de la boucle

7.2 Création de fonction

Les arguments de fonction sont cherchés d'abord par un match exacte du nom, puis par un match partiel et enfin par la position.

nomf° <- *function**(ech.1, ech.2) { x<-mean(ech.1) - mean(ech.2) return(x) }** Fabriquer une fonction
*if(condition) {action1} *else {action2} si la condition est respecté->action1 sinon ->action 2
*ifelse(test,oui,non) Version vectorisée de if fait un test renvoie la première valeur si test vrai (ici "oui") et deuxième valeur si test faux (ici "non")
*while(condition) {commandes} boucle la commande tant que la condition est vrai
*repeat **{commandes}** boucle infinie
*body(f) donne le corps de la fonction f
*formals(f) donne les arguments de la fonction f
*environment(f) donne l'environnement de la fonction f
*do.call(f, listargs) Applique la fonction f avec les arguments listés dans listargs

7.3 Modèle nul

*permatswap(a, times =100) création de 100 matrices aléatoires. Pour contrôler les sommes des lignes et/ou des colonnes : fixedmar=("none", "rows", "columns", "both"). Attention il existe plusieurs méthodes de randomisation (swap, quasiswap, tswap) *cf* ?permat ; package *Vegan
*sample(X, size = y, prob=Y) tirage de y éléments dans le vecteur X selon le vecteur de probabilité Y

7.4 Autres fonctions pour la programmation

*parse("text") transforme un texte en une expression qui n'est pas évaluée
*deparse("text") transforme une expression en texte
*eval(*expr*, envir=data) évalue l'expression dans un environnement (ici l'environnement data, souvent un sous forme de dataframe) *e.g.* **eval(parse**("text")) évalue une expression ayant été transformé en texte

7.5 La famille apply

cf package *dplyr pour des fonctions plus performantes.

apply(a, 1, function(x) tapply(x, fact, var)) applique la fonction x (ici variance pour chaque groupe définis par le facteur fact) aux lignes de la matrice a (remplacer 1 par 2 pour les colonnes)
*tapply(X, fact, mean) on coupe le vecteur X en autant de morceaux que de valeurs que prend le vecteur fact et on demande une stat (ici la μ) par groupe

*apply(a, 1, mean) moyenne des lignes (remplacer 1 par 2 pour les colonnes) de la matrice a

*lapply(liste, mean) application de la fonction (ici la moyenne) à chaque composante de la liste df[] <- lapply(liste, mean) stocke les résultats dans le dataframe df

*sapply identique à lapply mais retourne un vecteur ou une matrice plutôt qu'une liste. *cf* aussi les fonctions **mapply** et **vapply**

7.6 Environnement, session et debuggage

Environnement particulier : *globalenv(), *baseenv(), *emptyenv()

*search() Liste les environnements dans l'ordre de parenté.

*ls(env) liste objet de l'environnement ENV. *ls.str(env) liste les structures de l'environment ENV

*ls(pat="e") donne la liste des objets qui comportent la lettre e ("^e" -> donne les objets qui commencent par e) ls()donne la liste des objets stockés

*gc() collecte des déchets ("garbage collector")

*sessionInfo() liste les informations sur la session en cours

*traceback() donne la séquence de commandes qui a amené à la dernière erreur

*new.env(parent=*baseenv()) définit un nouvelle environnement dont le parent est l'environnement de base

*parent.env(env) donne le parent de l'environnement ENV

7.7 Benchmarking et profiling

*Sys.getpid() pour obtenir l'identité du processus R en cours

ptm <- *proc.time() ; f(x) ; *proc.time() - ptm calcul le temps mis par la fonction f(x) pour s'exécuter. Une alternative : system.time(f(x))

*microbenchmark(f1 = functionOne(x,y), f2 = functionTwo(x,y)) Package *microbenchmark. Mesure le temps de calcul de plusieurs fonctions (ici functionOne et functionTwo même sur des fonctions très rapides (plusieurs permutations).

*mem_used() donne la mémoire utilisé. *mem_change(f(X)) donne le changement de mémoire dû à la fonction f(x).

*object_size(obj) taille de l'objet

*tracemem(obj) marque un objet pour qu'un message aparaisse à chaque fois que l'objet est copié en interne.

*Rprof(nomfichier) *puis* f(x) *puis* summaryRprof("nomfichier") *cf* aussi le package lineprof avec un outils de visualisation.

8 Calcul

*sqrt(x) racine de x

x ∧ y x à la puissance y

*log(x, base=y) logarithme de x à la base y

*scale(X) centrage-réduction des éléments de X. Pour seulement centrer ou réduire, utiliser scale=F ou center=F

*solve(a, X) résoud les équations a*x=X avec X un vecteur ou une matrice et a une matrice carré d'un système linéaire

*round(x, 3) arrondi le nbre x de 3 chiffres après la virgule

9 Condition logique et selection de données

*rep(c(TRUE, FALSE),50) répète 50 fois TRUE et FALSE, on peut aussi utiliser rep() avec des nombres bien entendu

*table[a[,3]==1] donne le nombre de ligne où a[,3] est bien égale à 1 (TRUE) et le nombre de FALSE

*& ET logique

*| (alt-gr 6) OU logique

*xor OU exclusif

*which(x==0) donne un vecteur TRUE-FALSE de même longueur que x avec TRUE quand l'élément de x est égale à 0

*which.max(X) donne le numéro de ligne pour laquelle la valeur de la col 1 de la matrice a est maximale

a[which.max(X)] donne la ligne entière pour laquelle la valeur de la col 1 de la matrice a est maximale

*which(Y==1 & b[,1] > median(b[,1])) donne les lignes pour lesquelles la 2ème col = 1 et la valeur de la 1ère col est sup à sa médiane

10 Divers

*rbind(X, Y) et *cbind((X, Y)) fusionne les deux vecteurs en une matrice à 2 lignes (ou 2 colonnes) ; peut s'appliquer avec une matrice

*levels(a$attr) donne les attributs de la variables atrr de la matrice a

*q() sortir de r

*NULL vierge

*par() renvoie la liste de tous les paramètres. **oldpar <- par()** : sauvegarde le paramétrage courant dans la variable oldpar, **par(cex = oldpar$cex)** rétablit le paramètre cex à la valeur précédemment sauvegardée dans oldpar.

*# après un dièse ont peut ajouter un commentaire dans le script. Toutes les lettres après ce signe seront ignorer par R

*! non

*!= différent

*unique(a) supprime les éléments dupliquer

colnames(a) *%in% names()X donne l'intersection entre les deux vecteur de noms. Utile par exemple pour sélectionner la partie de la matrice a qui correspond aux nom du vecteur x : a[colnames(a) %in% names()X]

*drop(a) réduit les dimensions inutiles d'une matrice ou d'un array (matrice à n dimensions)

*replicate(x, f) retourne une liste (simplifié si SIMPLIFY=TRUE) des résultats de la fonction F lancé x fois.

11 Packages

De nombreuses TASK VIEW permettent de connaitre les packages disponible par thème. Par exemple en phylogénie & écologie, *cf* la task view PHYLOGENETICS du CRAN.

11.1 Gestion des packages

*install.packages("package") installe le package à partir du cran ou d'un autre dépot (argument *repos*)

*library("package") charge le package "package" préalablement installé

*require("package") test si le package "package" peut être chargé, à utiliser en interne dans les fonctions

*update.packages("package") met à jour les packages installés

*remove.packages("package") supprime les packages installés

*installed.packages("package") liste les packages installés

*updateR(F, T, T, F, T, F, T) nécessite le package `installr`, permet une mise à jour de R en gardant les packages de l'ancienne version de R. Plus d'infos.

*revdep_check() possible de résumer avec *revdep_check_summary.

nompackage* : :nomfunction Appelle la fonction nomfunction du package nompackage. En particulier utile pour les fonctions cachées (*e.g.* PLOT.DATA.FRAME du package STATS). nompackage* : : :nomfunction va chercher aussi les fonctions cachés du package (celle qui ne sont pas exportée dans le NAMESPACE)

11.2 ade4

*table.value(a) représentation graphique des valeurs de la matrice (très utile pour visualiser une matrice de distance)

*s.value()

*dist.binary(a) indice de similarité (*e.g.* Jaccard). Attention, pour jaccard l'indice n'est pas le même que celui calculé avec vegdist (calcul strict dans vegan, on retrouve la même chose qu'à la main) à cause d'une transformation. Cependant les deux similarités sont strictement proportionnelles (relation non linéaire).

11.2.1 Analyses sur un tableau

*dudi.pca(a) ACP

*dudi.coa(a) AFC

*dudi.acm(a) AFCM

*dudi.hillsmith(a) / *dudi.mix(a) analyse d'un tableau à la fois quantitatif et qualitatif (dudi.mix est une généralisation de dudi.hillsmith)

*dudi.nsc(a) analyse non symétrique de correspondance

*discrimin(obj.dudi, fac)

11.2.2 Analyses sur deux tableaux

11.2.3 Analyses sur k tableaux

11.3 ape

11.4 FactoMineR

*PCA pour voir les résultats : **PCA (a) <-res** puis appeler **res**

*names(res)

res$eig valeurs propres

res$var cos2, coord et contribution des axes

res$ind idem pour les individus

11.5 picante

*raoD indice de Rao

*pd()

*ses.pd()

11.6 spaa

*plotnetwork

11.7 Vegan

*diversity() indice de diversité traditionel

*taxondive() indice de diversité taxonomique

*vegdist() indice de similarité (Jaccard)

*permatswap modèle nul

11.8 diversité fonctionnelle

Package *FD / Package *cati

*trophdiv() div. fonctionnelle pour un trait

*ERED() div. fonctionnelle pour plusieurs traits

11.9 Analyses spatiales

Package *sp / Package *OpenStreetMap / Package *ade4 / Package *GeoXp

11.10 dplyr

cf la fiche synthétique de dplyr pour plus d'information.

*data_frame(cbind(X,Y))

*as.tbl_df(df)

*arrange(tbl_df, nomcol1)

*filter(tbl_df, nomcol1 == y & nomcol5 >= nomcol3)

*group_by(tbl_df, nomcol2)

*select

*summarise

*glimpse

*%>% Package magrittr. Équivalent d'un PIPE sous R. Injecte le résultat de droite en tant que premier argument de la fonction suivante pour n'importe quelle fonction dans R.

11.11 htmltools

*tag créer une définition pour html 5, e.g. tagList

*tagList() liste traduite en html pour incorporation dans document markdown par exemple

12 Document de travail grâce à Knitr

Knitr est un package permettant de réaliser des documents hybride mêlant du texte en format latex (ou markdown) et des zones de code R (et les résultats associés y compris les figures).

13 Liens internet

13.1 En français

13.1.1 Débutant

guide d'Emmanuel Paradis (2005)

Cours d'Emmanuel Paradis, plusieurs documents de cours de stats et de R par Emmanuel Paradis (en français et en anglais)

Aide mémoires, sites très complet par Aymeric Duclert

Tutoriels sur le logiciel R par François Huchon le développeur du package FactoMineR

Site collaboratif de partage de scripts, de codes et d'astuces

13.1.2 Avancées

Utilisation avancée de R avec Rstudio, créer de la documentation et des packages avec Rstudio, Eric Marcon (2014)
Cours de programmation sous R, nombreux cours et liens utiles par Ricco Rakotomalala
Petit Manuel de S4, guide sur la classe S4 par Christophe Genolini

13.2 En anglais

13.2.1 Débutant

documentation R, site très utile pour naviguer dans l'ensemble des fonctions disponibles sous R.
Style guide, guide de style pour le langage R, Hadley Wickham
carte de reférence assez complète par Matt Baggott (2012)
le cran, la base! A voir plus particulièrement : Import/export de données ; liste de documents sur cran
Un site assez complet sur le language R par Robert I. Kabacoff
Initiations de stats sous R par Olivier Flores
Les couleurs sous R
Un blog entièrement consacré à R
Site pour chercher des infos sur R par Jonathan Baron

13.2.2 Avancées

guide à la création d'un package, Hadley Wickham
Programming in R par Thomas Girke

Index