Introduction à R

Programmation & Logiciels Statistiques

TD 3 (1ère Partie)

Salim Lardjane - Université de Bretagne-Sud
2012-2013

while, repeat

- *Exemple : Script nx.R*

```
 nx <- function()
{
 # plus petit entier n tq 2^n >= 15
 x<-15
 n<-0
 while (2^n < x)
 { n <- n+1
 cat(n, "\n") # Affiche le résultat obtenu
}}

 Ctrl a/Ctrl r
> nx()
```

while, repeat

- Au lieu de while, on aurait pu utiliser un repeat de la façon suivante :
-

```
 nx2 <- function()
{
 # plus petit entier n tq 2^n >= 15 (version 2)
 x<-15
 n<-0
 repeat {
                n <- n+1
                if (2^n >= x) break
               }
 cat(n," \n") # Affiche le résultat obtenu
}
```

Ctrl a/Ctrl r

```
> nx2()
```

stop, warning

- stop renvoie un message d'erreur et interrompt l'exécution de la fonction ou du script en cours.

- *Exemple : Script sf.R*

```
 sf <- function(x)
{
 # Une fonction simple
 if (x>0) return(log(x)) else stop("x est inférieur ou égal à 0")
}
> sf(2)
> sf(-2)
```

stop, warning

- warning renvoie un message mais n'interrompt pas l'exécution de la fonction ou du script en cours.
- *Exemple : Script narm.R*

```
 narm <- function(x)
{
 # Supprime les valeurs manquantes
 if (sum(is.na(x))>0) {
 warning("Les valeurs manquantes ont été supprimées")
 return(x[!is.na(x)])
}
}
> x<-c(1:5,NA)
> x<-narm(x)
> x
```

switch

```
centre <- function(x, type) {
switch(type,
mean = mean(x),
median = median(x),
trimmed = mean(x, trim = .2),
stop("Choix non valide"))
}
> x <- rcauchy(10)
> centre(x, "mean")
> centre(x, "median")
> centre(x,"trimmed")
> centre(x,"trim")
```

dump, dput, dget

- Les scripts peuvent également être utilisés pour sauvegarder des objets R. Cela évite d'avoir à distinguer fichiers de code et fichiers de données.

```
> x<-1:10
> y<-
matrix(seq(0,1,length=6),nrow=2,byrow=TRUE)
> dump("x","x.R")
```

dump, dput, dget

```
> rm(x)
> x
> source("x.R")
> x
> dput(y,"y.R")
> rm(y)
> y
> dget("y.R")
> y
> y<-dget("y.R")
> y
```

- On pourra ouvrir les scripts x.R et y.R avec l'éditeur de scripts afin de visualiser le code obtenu.

Entrées/Sorties Console et sorties fichiers

- *Affichage à la console*

```
> x<-1:10
> x
> print(x)
> cat(x, " \n")
> x<-"Bonjour"
> x
> print(x)
> cat(x," \n")
```

- Remarque : " \n" signifie un retour à la ligne.

Lecture de vecteurs à la console

```
 exemple1 <- function()
{
 # Lecture de 10 chiffres à la console et calcul de
 # leur somme
 cat("Saisissez 10 chiffres :"," \n")
 x <- scan(n=10)
 total <- sum(x)
 cat("Le total des 10 chiffres est :",total," \n")
}
> exemple1()
```

Lecture de chaînes de caractères à la console

```
 exemple2 <- function()
{
# Simulation de 10 réalisation issues de la loi normale ou
# uniforme, au choix
cat("Nom de la loi des 10 réalisations ?"," \n")
cat("normale ou uniforme : ")
loi <- readline()
switch(loi,normale=rnorm(10),uniforme=runif(10),
          stop("Choix non valide"))
}
> exemple2()
```

Ecriture simple dans un fichier

```
> x<-1:10
> cat(x," \n",file="afile.txt")
> x<-"Bonjour"
> cat(x," \n",file="afile.txt",append=TRUE)
```

- Le fichier "afile.txt" est créé par cat s'il n'existe pas. L'option append=TRUE spécifie que l'écriture doit se faire en fin de fichier, c'est-à-dire à la suite du contenu existant. On pourra ouvrir le fichier "afile.txt" pour visualiser le résultat obtenu.

Redirections

- La fonction sink permet de rediriger les sorties R vers un fichier -- en créant celui-ci si nécessaire -- tout en conservant la possibilité de les afficher également à la console (option split=TRUE). L'utilisation de sink est particulièrement intéressante pour des sorties longues ou des sorties qui ne sont pas effectuées à l'aide de cat.

Redirections

```
> sink("myfile.txt",split=FALSE)
> cat("Un exemple simple :"," \n")
> cat(letters," \n")
> sink()
```

Redirections

```
> sink("myfile.txt",append=TRUE,split=FALSE)
> cat("En majuscules :"," \n")
> cat(LETTERS," \n")
> sink()
```

Redirections

```
> sink("myfile.txt",append=TRUE,split=FALSE)
> cat("Et pourquoi pas du numérique :"," \n")
> log(10)
> sin(10)
> A <- matrix(1:6,nrow=2)
> print(A)
> sink()
```

- L'option append=TRUE spécifie que l'on doit écrire en fin de fichier et non au début (valeur par défaut). On pourra ouvrir le fichier "myfile.txt" pour visualiser le résultat obtenu.

Connexions

- Il est possible d'ouvrir un fichier en écriture sous R à l'aide de la commande file. On dit qu'on ouvre une connexion. Après usage, une connexion doit être fermée à l'aide de l'instruction close.
- Lors d'un recours à sink("fichier.txt"), une connexion est en fait automatiquement ouverte vers "fichier.txt"; elle est refermée par la commande sink(). L'intérêt de file est que la connexion est ouverte indépendamment de sink et peut donc être utilisée par d'autres fonctions.

Connexions

```
> f<-file("newfile.txt",open="w")
> f
```

- L'option `open="w"` spécifie que le fichier est ouvert en écriture (le fichier est créé s'il n'existe pas).

Connexions

```
> sink(f,split=TRUE)
> cat("Un exemple simple"," \n")
> cat("sur deux lignes"," \n")
> cat("ou plutôt trois")
> sink()
> A<-matrix(1:6,nrow=2)
> write(A,file=f,ncolumns=3,append=TRUE)
```

Connexions

- La fonction sink permet de réorienter les sorties à suivre de la console vers la connexion f, c'est-à-dire vers le fichier "newfile.txt". L'option split=TRUE spécifie que les sorties doivent également apparaître à la console. La fonction write est spécifiquement adaptée à la sauvegarde de matrices au format texte (consulter l'aide). On pourra ouvrir le fichier "newfile.txt" avec l'éditeur de scripts pour visualiser la sortie affichée.

Connexions

```
> close(f)
> f
> unlink("newfile.txt")
> unlink("myfile.txt")
> unlink("afile.txt")
```

- La commande close(f) ferme la connexion. La commande unlink("fichier.txt") permet d'effacer le fichier "fichier.txt".

Listes

- La fonction list peut être utilisée pour combiner des objets de différents types ou modes en un seul objet de type, de mode et par défaut de classe list. La longueur d'une liste correspond au nombre d'objets qui y sont stockés.

```
> rm(list=ls())
> f <- function(x,y) {return(x+y-2)}
> L <- list(1:3,"hello",f)
> L
> typeof(L)
> mode(L)
> class(L)
> length(L)
```

Listes

- On peut assigner des labels ("noms") aux différents éléments d'une liste à l'aide de l'attribut names. Ceci peut être fait soit lors de la création de la liste, soit à l'aide de la fonction names.

```
> names(L)
> names(L)<-c("A","B","C")
> L
> L2 <- list(A2=1:3,B2="hello",C2=f)
> L2
> names(L2)
```

Extraction d'élements d'une liste

```
> L[[1]]
> L[[1]][2]
> L[[2]]
> L[[3]]
> L[[3]](1,2)
> L$A
> L$A[2]
> L$B
> L$C
> L$C(1,2)
```

Extraction d'élements d'une liste

```
> L[["A"]]
> L[["C"]]
> L[["C"]](1,2)
```

- *Distinction entre* L[1] *et* L[[1]]

```
> L[1]
> mode(L[1])
> L[[1]]
> mode(L[[1]])
```

Concaténation de listes

```
> LL <- c(L,L2)
> mode(LL)
```

Commentaires

- De même que pour les vecteurs, matrices, fonctions et expressions, on peut utiliser l'attribut comment pour associer un commentaire à une liste.

```
> comment(LL)<- c("Liste obtenue par concaténation","29.09.2011")
> comment(LL)
```

Attacher/Détacher une liste

- Afin de faciliter l'accès aux éléments d'une liste et pour en alléger la notation, il est possible d' *attacher* une liste. On peut alors accéder aux composantes de la liste directement avec leurs noms. Une fois cette utilisation terminée, il est nécessaire de *détacher* la liste. On ne peut alors accéder aux composantes de la liste qu'avec la syntaxe usuelle ($ ou [[ ]]).

Attacher/Détacher une liste

> LL

> attach(LL)

> A

> B

> detach(LL)

> A

Attribut dimnames des objets de classe matrix

```
> y <- matrix(1:6,nrow=3,ncol=2,byrow=TRUE)
> rownames(y)<-c("a","b","c")
> colnames(y)<-c("x1","x2)
> y
> dimnames(y)
> dimnames(y) <-
list(LIGNES=c("i1","i2","i3"),COLONNES=c("V1","V2"
))
> y
> dimnames(y)
```

Return

- Lorsque le résultat d'une fonction est multiple, on peut utiliser la fonction return pour le renvoyer sous la forme d'une liste.

- *Exemple*

```
> mvs<-function(x)
{
x.mean <- mean(x)
x.var <- var(x)
x.sum <- sum(x)
return(value=list(MOY=x.mean,VAR=x.var,SOM=x.sum))
}
> x<-1:10
> mvs(x)
> class(mvs(x))
```

Return

- *Quelques fonctions de matrices dont le résultat est une liste*

qr(x)	décomposition QR de la matrice x
chol(x)	décomposition de Cholesky - - -
svd(x)	décomposition en valeurs singulières - - -
eigen(x)	valeurs et vecteurs propres - - -

Facteurs

- La fonction factor peut être utilisée pour définir des objets de classe *factor*. La notion de facteur correspond en Statistique à celle de variable catégorielle. Des facteurs peuvent également être définis par conversion directe de vecteurs ou par regroupements d'éléments d'un vecteur.

```
> rm(list=ls())
> age <- factor(c(1,1,2,2,1,3,1,2),labels=c("20-35ans","35-55ans","+55ans"))
> age
```

Facteurs

> typeof(age)

> mode(age)

> length(age)

> class(age)

Facteurs

- L'attribut levels correspond aux valeurs possibles des éléments d'un facteur. Il peut être obtenu ou modifié à l'aide de la fonction levels.

```
> levels(age)
> levels(age)<-c("20-35ans","35-50ans","+50ans")
> age
```

Conversion d'un vecteur de mode numérique

```
> age <- c(1,1,2,2,1,3,1,2)
> age <- factor(age,labels=c("20-35ans","35-
      50ans","+50ans"))
> age
> age <- c(1,1,2,2,1,3,1,2)
> age <- as.factor(age)
> age
> levels(age)<-c("20-35ans","35-50ans","+50ans")
> age
```

Conversion d'un vecteur de mode caractère

```
> age <- c("20-35ans","20-35ans","35-
      50ans","35-50ans","20-35ans",
      "+50ans","20-35ans","35-50ans")
> age
> age <- as.factor(age)
> age
```

Regroupement de valeurs d'un vecteur de mode numérique

```
> age <- c(22,31,37,49,27,60,34,47)
> age <- cut(age,breaks=c(20,35,50,80),labels=c("20-
        35ans","35-50ans","+50ans"))
> age
```

- cut peut également être utilisée pour obtenir des regroupements en classes d'amplitudes égales.

```
> age <- c(22,31,37,49,27,60,34,47)
> age <- cut(age,breaks=3)
> age
```

Regroupement de valeurs d'un vecteur de mode numérique

- La fonction pretty permet d'obtenir des limites de classes plus raisonnables.

```
> age <- c(22,31,37,49,27,60,34,47)
> age <- cut(age,breaks=pretty(age))
> age
```

Facteurs ordonnés

```
> age <-
ordered(c(1,1,2,2,1,3,1,2),levels=c(1,2,3),
labels=c("20-35ans","35-50ans","+50ans"))
> age
> typeof(age)
> mode(age)
> length(age)
> class(age)
```

Facteurs ordonnés

- La classe d'un facteur ordonné est c("ordered","factor"). Cela signifie qu'outre les fonctions spécifiques à la classe ordered, les fonctions spécifiques à la classe factor et sans équivalent pour la classe ordered pourront être appliquées à un tel objet.

Définition d'un facteur ordonné par conversion

```
> age <- c(1,1,2,2,1,3,1,2)
> age <- ordered(age,labels=c("20-35ans","35-
      50ans","+50ans"))
> age
> age <- c("20-35ans","20-35ans","35-50ans","35-
      50ans","20-35ans",
      "+50ans","20-35ans","35-50ans")
> age
> age <- as.ordered(age)
> age
```

Définition d'un facteur ordonné par conversion

```
> levels(age) <- c("20-35ans","35-
      50ans","+50ans")
> age
> age2 <- c(22,31,37,49,27,60,34,47)
> age2 <- cut(age2,breaks=pretty(age2))
> age2 <- as.ordered(age2)
> age2
```

Tableaux de contingence

```
> table(age)
> sexe <-
factor(c(1,2,2,1,2,1,2,2),labels=c("F","H"))
> table(sexe,age)
```

tapply

- Afin d'appliquer une même fonctions aux cellules d'un tableau de contingence, on peut utiliser la fonction R tapply. Supposons qu'on souhaite obtenir la pression artérielle moyenne pour chacun des groupes sexe/age.

```
> press <- c(118,125,128,127,110,140,130,120)
> tapply(press,list(sexe,age),mean)
```

grep

- De façon analogue à son utilisation pour les chaînes de caractères, la fonction grep peut être utilisée pour obtenir les indices correspondant à un niveau donné d'un facteur.

```
> grep("20-35ans",age)
```

Tableaux de Données (Data Frames)

- Les *tableaux de données* R (data frames) sont des objets permettant de représenter des données de type "feuille de calcul". Chaque ligne correspond à une unité statistique (individu) et chaque colonne à une variable.
- L'avantage des tableaux de données par rapport aux matrices est que les colonnes peuvent être des vecteurs de classes différentes: numérique, logique, facteur...etc.

Définition d'un tableau de données

```
> rm(list=ls())
> td <-
      data.frame(sexe=rep(c("H","F"),c(2,3)),temps
      =round(rexp(5),2))
> td
> typeof(td)
> mode(td)
> length(td)
> class(td)
```

Extraction d'éléments d'un tableau de données

- Les tableaux de données sont de mode list et de ce fait, des éléments peuvent en être extraits de la même façon que pour les listes.

```
> td$temps
> td$sexe
> td[[1]]
> td[[2]]
```

Tableaux de données

- De même que les matrices, les tableaux de données possèdent les attributs dim, rownames, colnames, dimnames et comment.

```
> dim(td)
> rownames(td)
> colnames(td)
> dimnames(td)
> comment(td)<-c("Un exemple simple de data frame", "11.09.2011")
> comment(td)
> dimnames(td)<- list(letters[1:5],c("Sexe","Temps"))
> td
```

Ajout d'une ligne

```
> f<-
data.frame(Sexe="H",Temps=round(rexp(1),2))
> td<-rbind(td,f)
> rownames(td)[6]<-"f"
> td
```

Ajout d'une colonne

```
> Age<-c(28,21,35,22,22,26)
> td<-cbind(td,Age)
> td
```

Attacher/Détacher un tableau de données

- De même que pour les listes, il est possible d'attacher un tableau de données de façon à appeler les colonnes du tableau simplement par leur nom.

```
> attach(td)
> Sexe
> Age
> detach(td)
> Sexe
> Age
> td$Sexe
```

Attacher/Détacher un tableau de données

- *Remarque importante* : La commande attach crée en fait une copie du tableau de données. Aucun changement effectué sur une des colonnes alors que le tableau est attaché n'est sauvegardé dans le tableau initial.

Fusion de tableaux de données

```
> td2 <- data.frame(Temps2=round(rexp(6),2))
> rownames(td2) <- c("e","f","a","d","b","c")
> td2
```

- Les individus ne sont pas dans le même ordre dans td2. On ne peut donc recoller les tableaux de données directement.

Fusion de tableaux de données

```
> Noms<-rownames(td2)
> td2 <- cbind(Noms,td2)
> index <- order(td2$Noms)
> td2 <- td2[index,]
> td2
```

Fusion de tableaux de données

- Les individus sont à présent dans le même ordre dans td2 et td. On peut effectuer la fusion par simple concaténation de colonnes.

```
> td3 <- cbind(td,td2)
```

- Lorsqu'il n'y a pas d'ordre évident sur les identifiants des individus, on peut utiliser la fonction match.

```
> match(c("B","O","N","J","O","U","R"),LETTERS)
> letters[match(c("B","O","N","J","O","U","R"),LETTERS)]
```

Fusion de tableaux de données

- Supposons qu'on dispose d'un tableau de données td4 où tous les individus ne sont pas représentés et où un individu supplémentaire apparaît.

```
> td4 <-
data.frame(Age=c(21,26,28,37),Temps3=round(rexp
(4),2))
> rownames(td4) <- c("b","f","a","g")
> td4
```

Fusion de tableaux de données

- Nous pouvons effectuer une fusion à l'aide des identifiants des individus ("Noms").

```
> Noms<-rownames(td4)
> td4 <- cbind(Noms,td4)
> td5 <- merge(td3,td4,by="Noms",all=TRUE)
> td5
```

- L'option by permet de spécifier la variable selon laquelle le recollement est effectué; l'option all=TRUE spécifie que tous les individus apparaissant dans au moins l'un des deux tableaux de données doivent être inclus dans le tableau résultant.

Sauvegarde de tableaux de données au format texte

```
> write.table(td5,file="td5.txt")
```

- Par défaut, le séparateur de champs est " ". Si l'on souhaite que ce soit ",", on peut procéder de la façon suivante :

```
> write.table(td5,file="td5.csv",sep=",")
```

Lecture de tableaux de données au format texte

```
> td6 <- read.table(file="td5.txt",header=TRUE)
```

- Si le séparateur de champs est "," (fichiers .csv ou .CSV) au lieu d'être " ", on pourra procéder de la façon suivante :

```
> read.table(file="td5.csv",header=TRUE,sep=",")
```

ou

```
> read.csv(file="td5.csv",header=TRUE)
```

- Pour d'autres options et variantes de write.table et read.table, on pourra consulter l'aide en ligne. Pour l'importation de données aux formats Stata, SPSS, SAS, Fortran, Epiinfo, on consultera le menu Aide/Manuels (en PDF)/R data Import/Export.

Editeur de tableaux de données

- Le sexe de l'individu g est en fait H. Nous pouvons apporter cette correction au tableau de données à l'aide de l'éditeur R.

```
> td6 <-edit(td6)
> td6
```

Résumés d'un tableau de donnée

- Les fonctions str et summary sont bien définies pour les tableaux de données et permettent d'en donner un aperçu succinct; du point de vue structurel pour str et du point de vue statistique pour summary.

```
> td6
> str(td6)
> summary(td6)
```

Séries Temporelles

- Les objets R de classe ts ou mts peuvent être utilisés pour représenter des séries temporelles. La fonction ts permet de créer un objet de classe ts à partir d'un vecteur (série temporelle simple) ou de classe mts à partir d'une matrice (série temporelle multiple).

```
> s<-ts(data=1:20, start = 1959)
> s
> typeof(s)
> mode(s)
> length(s)
> class(s)
```

Séries Temporelles

```
> s<-ts(data=1:10, frequency=4, start =
        c(1959,2))
> s
> typeof(s)
> mode(s)
> length(s)
> class(s)
```

Séries Temporelles

```
> s<-ts(data=cumsum(rexp(25)),  frequency=12,
        start = c(1959,2))
> s
> typeof(s)
> mode(s)
> length(s)
> class(s)
```

Séries Temporelles

- L'option frequency spécifie le nombre d'observations par année, start l'instant de la première observation sous la forme start=année ou start=c(année,numéro de période).

```
> s2<-
ts(data=matrix(abs(cumsum(rnorm(26))),nrow=13),frequency
        =12,start=c(1971,3))
> s2
> typeof(s2)
> mode(s2)
> length(s2)
> class(s2)
```

Séries Temporelles

- Une série temporelle multiple est en fait de classe c("mts","ts"). Cela signifie qu'outre les fonctions spécifiques à la casse mts, les fonctions spécifiques à la classe ts et sans équivalent pour la classe mts pourront être appliquées à une série temporelle multiple sous R.

- On peut spécifier le nom des séries à l'aide de l'option names de ts (voir l'aide) ou de l'attribut colnames de la série obtenue.

Séries Temporelles

```
> colnames(s2)<-c("X1","X2")
> s2
```

Représentations graphiques

```
> plot(s)
> windows()
> plot(s2)
```

Packages

- Un des grands avantages de R est qu'il est enrichi en permanence de nouveaux *packages*, c'est-à-dire de "paquets" de programmes spécifiquement conçus pour traiter un certain type de problèmes ou dédiés à un certain type d'applications. Il existe par exemple des packages dédiés à la Statistique non paramétrique, au Traitement du signal, aux Statistiques pour l'environnement, aux représentations graphiques...etc.

Packages

- Sous Windows, l'installation d'un nouveau package se fait via le menu "Packages". La plupart des packages peuvent être téléchargés directement à partir du site cran.r-project.org. Une fois un package installé, il doit être chargé sous R, à l'aide de la commande library. La commande library() donne la liste des packages installés et pouvant être chargés.

Packages

```
> library()
> library(graphics)
> library(datasets)
```

Packages

- Certains packages sont automatiquement chargés au démarrage de R. La liste de ces packages peut être obtenue de la façon suivante :

```
> search()
```

- Pour obtenir de l'aide à propos d'un package donné (notamment un descriptif et une liste des principales fonctions définies dans le package) :

```
> library(help=graphics)
```

Packages

- Le package datasets est spécifiquement dédié à la mise à disposition de jeux de données pour les différentes applications de R. Pour obtenir la liste des jeux de données disponible dans le package dataset :

```
> data()
```

Packages

- Pour obtenir la liste des jeux de données disponibles dans un package installé :

```
> data(package="cluster")
```

- Pour obtenir la liste des jeux de données disponibles dans tous les package installés :

```
> data(package = .packages(all.available = TRUE))
```

Packages

- Pour charger un jeux de données :

```
> data(LakeHuron)
> LakeHuron
> data(flower,package="cluster")
> flower
```

Packages

- Pour obtenir un descriptif du jeux de données chargé :

```
> help(LakeHuron)
> help(flower,package="cluster")
```