

Formation Permanente

André Bouchier

CENTRE INRA de MONTPELLIER

FORMATION AU LOGICIEL



Programmation et interfaces graphiques

(durée : environ 6 heures)

version du 13 février 2006

Copyright © André Bouchier.

© 2006, André Bouchier (20 Janvier 2006)

Permission est accordée de copier et distribuer ce document, en partie ou en totalité, dans n'importe quelle langue, sur n'importe quel support, à condition que la notice © ci-dessus soit incluse dans toutes les copies. Permission est accordée de traduire ce document, en partie ou en totalité, dans n'importe quelle langue, à condition que la notice © ci-dessus soit incluse.

1-Programmer avec R

- R est à la fois un logiciel de statistique et un langage de programmation
- Avec R, on peut, par exemple, programmer des boucles afin d'analyser successivement différents jeux de données.
- On peut aussi combiner ou empiler dans le même programme plusieurs fonctions statistiques pour réaliser des analyses complexes.
- R est un langage interprété et non compilé

2-Quelques éléments à connaître sur les listes (1)

- Une liste est formée d'éléments pouvant être de types différents
- Par exemple, une liste contenant 3 éléments : le tableau de données iris, le nom des variables de ce tableau et les fréquences de chaque espèce

```
data(iris)
```

```
maliste<-list(iris, names(iris), table(iris[,5]))
```

- Pour accéder aux élément de la liste

```
maliste[1]
```

3-Quelques éléments à connaître sur les listes (2)

- Il est conseillé de nommer chaque élément de la liste

```
names(maliste) <- c("donnees", "noms", "frequence")
```

```
maliste$noms
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"  
"Petal.Width"   "Species"
```

4-Quelques éléments à connaître sur les listes (3)

- Une autre méthode pour nommer chaque élément de la liste

```
data(iris)
```

```
maliste<-list(donnees=iris,noms= names(iris),  
frequence=table(iris[,5]))
```

- On peut alors interroger plus précisément la liste

```
typeof(maliste$noms)
```

```
[1] "character"
```

```
maliste$noms[1]
```

```
[1] "Sepal.Length"
```

5-Quelques éléments à connaître sur les listes (4)

● Exercice 1 :

On utilisera le data.frame "Orange"

Créer une liste contenant

- une description des données (en une phrase),
- le tableau de données lui-même,
- les valeurs maxi des circonférences pour chaque arbre

Remarque : vous aurez besoin de la fonction aggregate()

6-Tâches complexes et répétitives, fonction by()

- Appliquer une fonction pour chaque niveau d'un facteur

```
data(iris)
attach(iris)
by(Sepal.Length, Species, mean)
```

- Appliquer la fonction aux éléments d'une liste

```
liste<-list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
names(liste)<-c("Sepal.Length", "Sepal.Width", "Petal.Length",
"Petal.Width")
by(liste, Species, mean)
detach(iris)
```

7-Tâches complexes et répétitives, fonction `by()`

● Exercice 2 :

Calculez les coefficients de corrélation de `pearson` entre l'age et la circonférence des orangers pour chaque arbre du `data.frame Orange`

8-Tâches complexes et répétitives : **lapply()**

- **lapply()** applique une fonction à une liste de vecteurs

```
attach(iris)
```

```
liste<-list(Sepal.Length, Sepal.Width, Petal.Length,  
Petal.Width)
```

```
names(liste)<-c("Sepal.Length", "Sepal.Width",  
"Petal.Length", "Petal.Width")
```

```
lapply(liste,mean)
```

```
lapply(liste, quantile, probs = c(0, .33, 0.66, 1))
```

```
detach(iris)
```

● **Exercice 3 :**

en utilisant **lapply()**, centrer et réduire les variables quantitatives du tableau de données **Iris**. Vous devez obtenir un data.frame.

9-Tâches complexes et répétitives : replicate()

- Pour répéter l'évaluation d'une expression (met en général en oeuvre la génération de tirages aléatoires)

- **exemple :**

la moyenne d'un tirage aléatoire

```
mean(sample(iris$Sepal.Length , size=20, replace = FALSE))
```

l'histogramme de la moyenne sur 1000 tirages aléatoires

```
hist(replicate(1000,mean(sample(iris$Sepal.Length,size=20))))
```

- **Exercice 4 :**

Faites le même exercice en calculant l'écart-type sur des tailles d'échantillon de 30 individus. Pour améliorer la lisibilité du graphique, vous utiliserez la variable Sepal.Length centrée-réduite

10-Echantillonner sur plusieurs variables

- Attention, danger 

```
attach(iris)
cor(sample(Sepal.Length, size=20), sample(Petal.Length, size=20))
detach(iris)
```

11-Echantillonner sur plusieurs variables

- Pour échantillonner des paires de valeurs

```
data (swiss)
don<-as.data.frame (swiss)
echant<-sample (1:dim (don) [1], size=10)
don [echant, ]
```

12-Tâches complexes et répétitives : replicate()

● Exercice 5 :

Faites 2000 corrélations linéaires entre Sepal.Length et Petal.Length sur des échantillons de 10 plantes. Dessinez l'histogramme des coefficients **r**.

13-Les conditions : ifelse()

- `ifelse(condition, "si vrai", "si faux")`

```
attach(iris)
ifelse( mean(Sepal.Length)>3, paste("moyenne: ",
mean(Sepal.Length)), "Moyenne <= 3")
detach(iris)
```

● Exercice 6 :

Faire un tirage aléatoire de 20 plantes dans le fichier iris. Selon la valeur de la moyenne de Sepal.Length, affichez le message approprié

- "moyenne > 5.8"
- "moyenne <= 5.8"

14-Les conditions : if else

- une fonction de base des langages de programmation

if (cond) expr1 else expr2

- **Par exemple :**

On a un scalaire x

si x est positif en calculer la racine

si x est négatif, calculer la racine de la valeur absolue

Afficher un commentaire à chaque fois

```
x<- -3
{if (is.numeric(x) & x>=0) cat("Racine de",x,"=",sqrt(x),"\n")
else cat("x négatif. Racine de |",x,"| : ", sqrt(-x), "\n")}
```

- **Que se passe-t-il si x=0 ?**

15-Les conditions : if else

● Exercice 7 :

Générez un vecteur de 30 valeurs aléatoires suivant une loi uniforme (fonction `runif()`). Les valeurs seront comprises entre 1 et 100.

Si les valeurs générées suivent une loi normale (`test de shapiro`) calculez et affichez la moyenne. Sinon, affichez la médiane.

16-Les boucles : for()

- Comment réaliser des boucles. Une autre fonction de base des langages de programmation

```
attach(iris)
for (i in 1:dim(iris)[1])
{
  if (Petal.Length[i] > 5) cat(as.character(Species[i]), "\n")
}
detach(iris)
```

- **Exercice 8** : Le vecteur X est dans le mauvais sens. Je souhaite remettre la semaine à l'endroit.

```
X<-c("Dimanche", "Samedi", "Vendredi", "Jeudi", "Mercredi",
     "Mardi", "Lundi")
```

17-Quitter une boucle :

- On utilisera l'instruction `break()` pour quitter une boucle

```
for (i in 1:10000)
{
  cat(i^2, " ")
  if (i^2>100) break()
}
```

- Pour générer un message d'erreur en quittant la boucle, on utilisera la fonction `stop()`

```
options(show.error.messages=F)
for (i in 1:10000)
{
  cat(i^2, " ")
  if (i^2>100) stop("i est trop grand")
}
geterrmessage() #affiche le dernier message d'erreur
options(show.error.messages=T)
```

18-Les boucles : while()

- Comment tourner en rond jusqu'à nouvel ordre

while(cond) expr

- **exemple :**

obtenir un jeu de 20 données dont la corrélation entre **Sepal.Length** et **Petal.Length** est comprise entre 0.70 et 0.72

```
while(TRUE) # TRUE est toujours vrai
{
x<-sample(iris$Sepal.Length , size=20)
y<-sample(iris$Petal.Length , size=20)
if (cor(x,y) >= 0.70 & cor(x,y) <= 0.72)
    {
        don<-data.frame(x,y)
        break()
    }
}
cor(don)
```

19-Les boucles : while()

● Exercice 9 :

Créer un jeu simple. L'ordinateur génère un entier X de 1 à 100. A vous de le trouver.

Vous proposez un nombre et R vous indique si celui-ci est plus grand ou plus petit que X .

Continuez jusqu'à ce que votre nombre = X

remarque : vous aurez besoin de la fonction `readline()`

20-Les boucles : repeat()

- Dans les boucles `while(condition)`, rien n'est exécuté si la condition est fausse.
- Dans les boucles `repeat()`, une première exécution est effectuée avant de tester la condition.

```
j<-1
repeat
{
  j<-j+1
  if(j> 10) stop("too many iterations j")
}
```

21-Les fonctions :

● créer une fonction :

exemple 1 : calculer la norme d'un vecteur

```
norm <- fonction(x) sqrt(x%*%x)
```

```
norm(1:4)
```

exemple 2 : calcul de l'écart-type de l'échantillon

```
sd.ech<-fonction(x)
{
  sd<-sum((mean(x)-x)^2)
  sd<-sd/(length(x)-1)
  sd<-sd^0.5
  return(sd)
}
```

22-Les fonctions :

● créer une fonction :

exemple 3 : calcul de l'écart-type de la population

```
sd.pop<-function(x)
{
  sd<-sum( (mean(x) -x) ^2)
  sd<-sd/ (length(x) )
  sd<-sd^0.5
  return(sd)
}
```

● Enregistrer des fonctions perso.

enregistrer les 3 fonctions à la suite dans un fichier texte. On peut ajouter des commentaires. Sauvegarder ce fichier dans un répertoire accessible. Appelez-le : **mesfonctions.R**

23-Les fonctions :

- Charger vos fonctions perso. 3 méthodes pour charger le fichier **mesfonctions.R**

1) avec le menu « File / Source R code »

2) insérer cette instruction dans votre script

```
source("c:/temp/mesfonctions.R")
```

3) avec un peu d'interactivité dans votre programme

```
source(file.choose())
```

24-Les fonctions : exercices

● Exercice 10 :

- construisez une fonction avec le jeu de devinette de nombre. Appelez cette fonction `jeux()`
- testez cette fonction
- Si tout marche bien, sauvegardez cette fonction dans votre bibliothèque personnelle, à la suite des autres.

25-Les fonctions : le débogage

- En cas de fonction récalcitrante, on peut contrôler les valeurs des variables à un point précis de l'exécution :

```
calcul<-function(a=3)
{
  d<-a^(1/4)
  return(d)
}
calcul(-2)
[1] NaN
```

- Utilisation de la fonction `browser()`

```
calcul<-function(a=3)
{
  d<-a^(1/4)
  browser()
  return(d)
}
calcul(-2)
Browse[1]> get("a")
[1] -2
```

26-Les fonctions : exercices

● Exercice 11 :

- 1) Ajoutez un contrôle sur un nombre d'essais maximal. Celui-ci doit être passé en paramètre avec une valeur par défaut de 6. Quand le nombre maxi d'essais est atteint, la partie est perdue. Affichez un message.
- 2) Testez votre fonction pour différentes valeurs du nombre d'essais maxi.
- 3) Si tout va bien, enregistrez votre fonction dans `mesfonctions.R` , à la suite des autres.
- 4) Quittez puis relancez R
- 5) Charger la fonction de manière interactive avec `file.choose()`

27-Les fonctions : porté des variables

- La fonction **norm()** calcule la norme d'un vecteur. Elle utilise une variable "interne" **x**. Une variable x existe déjà...

```
x<-10
norm <- function(x) sqrt(x%*%x)
norm(1:5)
x
[1] 10
```

- Les variables "déclarées" à l'intérieur d'une fonction ont une porté locale

28-Les fonctions : valeurs renvoyées par la fonction

- La fonction `ecrit()` doit renvoyer 3 valeurs

```
ecrit<-function(x) { x ; x^2 ; x^3 }
```

- Mais `ecrit(2)` ne renvoie qu'une seule valeur (la dernière)

```
ecrit(2)  
[1] 8
```

- Pour renvoyer plusieurs valeurs, on utilisera les listes

```
ecrit<-function(x) list(simple=x, carre=x^2, cube=x^3)  
ecrit(2)$carre  
[1] 4
```

29-un peu d'interaction avec le système de fichier :

- choisir un fichier texte de manière interactive :

```
nom<-file.choose()
```

- des informations sur ce fichier

```
file.info (nom)
```

- afficher le contenu

```
file.show (nom)
```

- détruire un fichier

```
file.remove()
```

- essayez ces fonctions

```
basename(nom)
```

```
dirname(nom)
```

```
file.path(nom)
```

- contenu d'un répertoire

```
dir(dirname(nom), pattern = ".txt", recursive = TRUE)
```

Attention `pattern=".txt"` est différent de `pattern=".TXT"` !

30-Un peu d'interaction avec le système de fichier :

● Exercice 12 :

Allez visiter le répertoire R qui contient les bibliothèques de fonctions. Il se trouve vers "C:\Program Files\R\rw2001\library".

Récupérez la liste des noms de répertoires (ça correspond à la liste des bibliothèque)

31-Une instruction utile : substitute()

- Une fonction est un conteneur. Par exemple, le vecteur "normale" contient 100 valeurs suivant une loi normale

```
normale<-rnorm(100,10,1)
```

- On peut calculer la norme de ce vecteur avec une fonction

```
norm <- function(x) sqrt(x%*%x)
```

```
norm(normale)
```

Les calculs sont effectués sur le contenu du vecteur "normale"

- Si j'améliore ma fonction pour afficher un commentaire

```
norm<-function(x) { y<-sqrt(x%*%x); cat("La norme de",x,"est",y) }
```

Le résultat n'est pas celui attendu, puisque je souhaite afficher le nom du vecteur et pas son contenu.

32-Une instruction utile : substitute()

```
norm<-function(x) { y<-sqrt(x%*%x); cat("La norme de", x,  
"est", y) }
```

- Dans cette fonction, la variable x fait référence au vecteur "normale" qui contient 100 nombres de moyenne 10 et écart-type 1
- Si on veut faire référence au nom du vecteur et pas à son contenu, on utilisera la fonction substitute()

- La fonction deviendra :

```
norm<-function(x) {  
  y<-sqrt(x%*%x)  
  cat("La norme du vecteur '", substitute(x), "'  
  est", y, "\n")  
}
```

```
norm(normale)
```

```
La norme du vecteur ' normale ' est 101.5202
```

33-Une instruction utile : substitute()

● Exercice 13 :

Créer une fonction dessinant un graphe xy et sa droite de régression.
Donnez un titre et des labels d'axe explicites.

● Exercice 14 :

Faites le même exercice en donnant un choix interactif de couleur pour la droite de régression.

34-Des boucles et encore des boucles

- On peut utiliser les boucles itératives pour des calculs complexes. Par exemple, sur le data.frame `Orange`, on peut calculer le gain moyen journalier de la circonférence des troncs pour chaque arbre.

```
data(Orange)
arbres<-as.integer(names(summary(Orange$Tree)))
arbres<-arbres[sort.list(arbres)]
d<-length(arbres)
result<-data.frame(arbre=c(1:d), gmj=c(1:d))
for (i in arbres)
{
  mini<-min(Orange[Orange$Tree==i, 3])
  maxi<-max(Orange[Orange$Tree==i, 3])
  diff<-(maxi-mini)/7
  result[i, 1]<-i
  result[i, 2]<-diff
}
```

35-Sauvegarder plusieurs graphiques

- La fonction `savePlot()` permet d'enregistrer sur disque le graphique "en cours"
- **Exercice 15** : Utilisez le data.frame `iris`. En utilisant la fonction `for()`, dessinez les boîtes à pattes des variables quantitatives (un seul graphe à la fois) et enregistrez ces graphiques dans votre répertoire de travail au format `jpeg`.

36-Vous avez un message !

- Afficher les résultats dans une boîte à message

```
library(tcltk)
norm <- function(x)
{
  n<-round(sqrt(x%*%x),2)
  texte<-tkmessageBox(type='ok', icon="info",
    message=paste("norme",n))
}
norm(1:10)
```

- Types de boîtes : ok, yesno
- Types d'icônes : info, warning
- **Exercice 16 :**

Créez une fonction qui efface un fichier. Cette fonction doit vous demander confirmation.

37-Ecrire les résultats dans un fichier texte

- Par défaut, les résultats des analyses sont écrits dans la fenêtre "R console". On peut les réorienter vers un fichier texte. On utilisera la fonction `sink()`

```
setwd("c:/temp")
sink("toto.txt", append=TRUE)
data(PlantGrowth)
cat("Données PlantGrowth\n")
cat("Analyse de variance effectuée le",
    format(Sys.time(), "%a %d %b %X %Y %Z"), "\n")
x<-aov(weight~group , data=PlantGrowth)
summary(x)
sink()
```

- **Exercice 17** : Dans l'exercice 2, vous avez calculé les coefficients de corrélation de `pearson` entre l'age et la circonférence des orangers pour chaque arbre du data.frame `Orange`. Refaites ces calculs de manière à ce que les résultats soient copiés dans le fichier `correlation.txt`. Commentez ces résultats.

38-Le batch : la non-interactivité totale !

- Il est possible d'automatiser des tâches répétitives. Dans ce cas, le passage par l'interface graphique de R devient inutile. On travaillera en mode *traitement par lot* (batch)

```
setwd("c:/temp")
sink("anova.txt", append=TRUE)
data(PlantGrowth)
cat("Données PlantGrowth\n")
cat("Analyse de variance effectuée le",
    format(Sys.time(), "%a %d %b %X %Y %Z"), "\n")
x<-aov(weight~group , data=PlantGrowth)
summary(x)
sink()
```

- On copie les instructions ci-dessus dans un fichier texte `rprg.txt`

- On lance l'exécution de ce programme grâce à la commande :

```
"C:\Program Files\R\rw2001\bin\R.exe" CMD BATCH "c:/temp/rprg.txt"
```

- Cette ligne de commande est exécutée dans une console dos

39-Le batch : la non-interactivité totale !

- Pour ne pas avoir à retaper une ligne d'instruction compliquée (avec des risques d'erreur), on peut copier cette ligne de commande dans un fichier "batch".

- La ligne de commande :

```
"C:\Program Files\R\rw2001\bin\R.exe" CMD BATCH "c:/temp/rprg.r"
```

peut-être copiée dans le fichier `rprg.bat`

- Il suffit alors de cliquer (ou double-cliquer) sur ce fichier pour lancer l'exécution du programme R

- Les résultats seront lisibles dans le fichier `c:\temp\anova.txt`

- **Exercice 18** : Automatisez intégralement l'exercice 17

40-Le système d'exploitation

● On peut lancer des instructions vers le système d'exploitation :

- par exemple, pour exécuter un programme batch :

```
shell.exec("c:/temp/rprg.bat")
```

- ou ouvrir une page html locale

```
system(paste('"c:/Program Files/Internet Explorer/IEXPLORE.EXE"',  
'C:/Program Files/R/rw2001/doc/html/rwin.html'), wait = FALSE)
```

- ou une page html distante

```
system(paste('"c:/Program Files/Internet Explorer/IEXPLORE.EXE"',  
'-url cran.r-project.org'), wait = FALSE)
```

- ou lancer un éditeur de texte

```
system("notepad c:/temp/prog.R")
```

41-Le système d'exploitation

● On peut utiliser les associations de fichiers du système d'exploitation

- Ouvrir un fichier html avec le butineur par défaut

```
shell.exec('C:/Program Files/R/rw2001/doc/html/rwin.html')
```

- Ouvrir un fichier pdf avec l'acrobat reader

```
shell.exec('C:/temp/lea_book.pdf')
```

- Ouvrir un fichier MS-Excel (avec Open Office si c'est l'application par défaut)

```
shell.exec('C:/temp/bledur.xls')
```

42-Développer une interface graphique

- Tcl (Tool Command Language) est un langage interprété, disponible *gratuitement* et qui fonctionne sous de très nombreux systèmes d'exploitation

- Un exemple de fenêtre de saisie :

```
library(tcltk)
tt<-tktoplevel()
tktitle(tt)<-"Tcl/Tk"
lab<-tklabel(tt, text="Entrez votre nom")
unnom<-tclVar("Anonyme")
nom<-tkentry(tt, width="20", textvariable=unnom)
b1<-tkbutton(tt, text="Quitter", command= function()
{
  tkdestroy(tt)
  tkmessageBox(icon="info", type="ok", message=
    paste(tclvalue(unnom), ": joli nom !"))
})
tkpack(lab, nom, b1)
```

43-Quelques adresses utiles :

- Des exemples de l'utilisation de Tcl/Tk :

<http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/>

- Le site officiel de R

<http://lib.stat.cmu.edu/R/CRAN/>

- Introduction à Tcl/Tk

<http://www-lipn.univ-paris13.fr/~hamon/SJ-TclTk/index.html>

- Une présentation de Tcl/Tk

<http://www.uppp.free.fr/Tcltk.htm>

44-Correction des exercices :

● Exercice 1 :

```
data(Orange)
descript<-"The Orange data frame has 35 rows and 3 columns of
records of the growth of orange trees."

valeurs.maxi<-aggregate(Orange$circumference ,
list(valmaxi=Orange$Tree), max)

Orange.liste<-list(Orange, descript, valeurs.maxi)
names(Orange.liste)<-c("Orange", "descript", "valeurs.maxi")
```

● Exercise 2 :

```
attach(Orange)
by(Orange[,c(2,3)], Tree, cor, method = "pearson")
detach(Orange)
```

ou

```
attach(Orange)
liste<-list(age, circumference)
names(liste)<-c("age", "circumference")
by(liste, Tree, cor, method = "pearson")
detach(Orange)
```

● Exercise 3 :

```
attach(iris)
liste<-list(Sepal.Length, Sepal.Width, Petal.Length,
Petal.Width)
names(liste)<-c("Sepal.Length", "Sepal.Width", "Petal.Length",
"Petal.Width")
cr<-as.data.frame(lapply(liste, scale, center=T, scale=T))
mean(cr) ; sd(cr)
detach(iris)
```

● Exercice 4 :

```
hist(replicate(1000, sd(sample(scale(iris$Sepal.Length),  
size=40))))
```

● Exercice 5 :

```
attach(iris)
hist(replicate(2000,
  {
    echant<-sample(1:dim(iris)[1], size=10)
    tirage<-iris[echant,]
    cor(tirage$Sepal.Length, tirage$Petal.Length )
  }
), main="2000 corrélations", xlab="")
detach(iris)
```

● Exercice 6 :

```
attach(iris)
text1<-"moyenne > 5.8"
text2<-"moyenne <= 5.8"
ifelse(mean(sample(Sepal.Length , size=20))>5.8,text1, text2)
detach(iris)
```

● Exercice 7 :

```
x<-runif(30,0,100)
shapiro.test(x)$p
if (shapiro.test(x)$p >= 0.05)
cat("moyenne x=",mean(x),"\n") else
cat("mediane x=", median(x),"\n")
```

● Exercice 8 :

```
X<-c("Dimanche", "Samedi", "Vendredi", "Jeudi", "Mercredi",  
"Mardi", "Lundi")
```

```
Y<-X
```

```
for (i in 1:length(X)) {Y[length(X)-(i-1)]<-X[i]}
```

```
Y
```

Remarque : on peut aussi, plus simplement utiliser la commande suivante :

```
Y<-X[7:1]
```

● Exercice 9 :

```
X<-round( runif(1,1,100))
X<-as.integer(X)
while(TRUE)
{
nb<-readline(prompt = "Proposez un nombre de 1 à 100 : ")
nb<-as.integer(nb)
if (nb==X)
{
cat("Vous avez gagné, la solution est : ",X, "\n")
break()
}
if (nb>X) cat("Essayez encore : nombre est trop grand \n")
if (nb<X) cat("Essayez encore : nombre est trop petit \n")
}
```

● Exercice 10 :

```
jeu<-function()  
{  
  X<-round( runif(1,1,100) )  
  X<-as.integer(X)  
  while(TRUE)  
  {  
    nb<-readline(prompt ="Proposez un nombre de 1 à 100 : ")  
    nb<-as.integer(nb)  
    if (nb==X)  
    {  
      cat("Vous avez gagné, la solution est : ",X, "\n")  
      break()  
    }  
    if (nb>X) cat("Essayez encore : nombre trop grand \n")  
    if (nb<X) cat("Essayez encore : nombre trop petit \n")  
  }  
}
```

● Exercice 11 :

```
jeu2<-function(maxi=6)
{
  X<-round( runif(1,1,100) )
  X<-as.integer(X)
  nbessai<-0
  while(TRUE)
  {
    nbessai<-nbessai+1
    nb<-readline(prompt = "Proposez un nombre de 1 à 100 : ")
    nb<-as.integer(nb)
    if (nb==X) {
      cat("Vous avez gagné, la solution est : ",X, "\n")
      break()
    }
    if (nbessai<maxi) {
      if (nb>X) cat("Essayez encore : nombre trop grand \n")
      if (nb<X) cat("Essayez encore : nombre trop petit \n")
    }
    if (nbessai>=maxi) {
      cat("Dommage, vous n'avez droit qu'à",maxi,"essais\n")
      break()
    }
  }
}
```

● Exercice 12 :

```
nom<-file.choose()  
liste<-dir(dirname(nom), recursive = F, full.names = F)
```

● Exercice 13 :

```
graphxy<-function(x,y,donnees)
{
  attach(donnees)
  plot(x,y,main=paste("relation entre", substitute(x), "et",
  substitute(y)), xlab=substitute(x),ylab=substitute(y))
  abline(lm(y~x))
  detach(donnees)
}
```

● Exercice 14 :

```
graphxy<-function(x,y,donnees)
{
  attach(donnees)
  plot(x,y,main=paste("relation entre", substitute(x), "et",
substitute(y)), xlab=substitute(x),ylab=substitute(y))
  message<-"Couleur du trait 1 bleue, 2 vert, 3 noir :"  
  couleur<-readline(prompt=message)  
  couleur<-as.integer(couleur)  
  if (couleur==1) coul<-"blue"  
  if (couleur==2) coul<-"green"  
  if (couleur==3) coul<-"black"  
  abline(lm(y~x),col=coul)  
  detach(donnees)
}
```

● Exercise 15 :

```
data(iris)
for (i in 1:4)
{
  boxplot(iris[,i], xlab="", main=names(iris)[i])
  savePlot(filename=paste("c:/temp/boxplot",i,sep=""),
           type="jpeg")
}
```

● Exercice 16 :

```
detruire<-function()  
{  
  fichier<-file.choose("Choisissez un fichier à effacer")  
  texte<-tkmessageBox(type='yesno', icon="warning",  
message=paste("détruire", fichier, "?"))  
  if (as.character(texte)=="yes")  
    {  
      file.remove(fichier)  
      cat("fichier", fichier, "détruit.\n")  
    }  
}  
detruire()
```

● Exercice 17 :

```
attach(Orange)
setwd("c:/temp")
sink("correlation.txt", append=TRUE)
  cat("Données Orange, Formation R \n")
  cat(format(Sys.time(), "%a %d %b %X %Y %Z"), "\n")
  by(Orange[,c(2,3)], Tree, cor, method = "pearson")
sink()
detach(Orange)
```

● Exercice 18 :

- Copier les lignes suivantes dans le fichier `correlation.txt`

```
attach(Orange)
setwd("c:/temp")
sink("correlation.txt", append=TRUE)
  cat("Données Orange, Formation R \n")
  cat(format(Sys.time(), "%a %d %b %X %Y %Z"), "\n")
  by(Orange[,c(2,3)], Tree, cor, method = "pearson")
sink()
detach(Orange)
```

- Copier la ligne suivante dans le fichier `correlation.bat`

```
"C:\Program Files\R\rw2001\bin\R.exe" CMD BATCH
"c:/temp/correlation.txt"
```

- Dans l'explorateur de fichier, cliquez sur le fichier `correlation.bat`. Le programme s'exécute sans ouvrir de fenêtre R

Table des matières

1-Programmer avec R.....	3
2-Quelques éléments à connaître sur les listes (1).....	4
3-Quelques éléments à connaître sur les listes (2).....	5
4-Quelques éléments à connaître sur les listes (3).....	6
5-Quelques éléments à connaître sur les listes (4).....	7
6-Tâches complexes et répétitives, fonction by().....	8
7-Tâches complexes et répétitives, fonction by()	9
8-Tâches complexes et répétitives : lapply().....	10
9-Tâches complexes et répétitives : replicate().....	11
10-Echantillonner sur plusieurs variables.....	12
11-Echantillonner sur plusieurs variables.....	13
12-Tâches complexes et répétitives : replicate().....	14
13-Les conditions : ifelse().....	15
14-Les conditions : if else.....	16
15-Les conditions : if else.....	17
16-Les boucles : for()	18
17-Quitter une boucle :.....	19
18-Les boucles : while().....	20
19-Les boucles : while().....	21
20-Les boucles : repeat().....	22
21-Les fonctions :.....	23
22-Les fonctions :.....	24
23-Les fonctions :.....	25

24-Les fonctions : exercices.....	26
25-Les fonctions : le débogage.....	27
26-Les fonctions : exercices.....	28
27-Les fonctions : porté des variables.....	29
28-Les fonctions : valeurs renvoyées par la fonction.....	30
29-un peu d'interaction avec le système de fichier :.....	31
30-Un peu d'interaction avec le système de fichier :.....	32
31-Une instruction utile : substitute().....	33
32-Une instruction utile : substitute().....	34
33-Une instruction utile : substitute().....	35
34-Des boucles et encore des boucles.....	36
35-Sauvegarder plusieurs graphiques.....	37
36-Vous avez un message !.....	38
37-Ecrire les résultats dans un fichier texte.....	39
38-Le batch : la non-interactivité totale !.....	40
39-Le batch : la non-interactivité totale !.....	41
40-Le système d'exploitation.....	42
41-Le système d'exploitation.....	43
42-Développer une interface graphique.....	44
43-Quelques adresses utiles :.....	45
44-Correction des exercices :.....	46