Initiation au logiciel R

Jean-Michel Marin

Ce document a pour objectif de familiariser son lecteur avec le langage et l'environnement de programmation R.

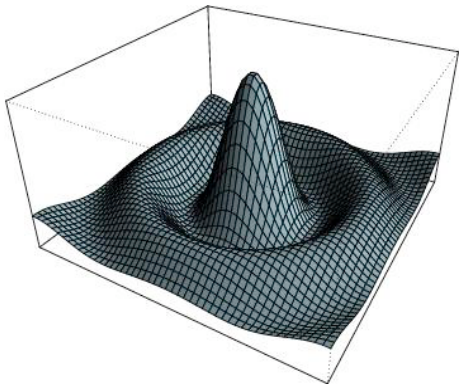

Table des matières

1 Introduction

R est un langage de programmation interactif interprété et orienté objet contenant une très large collection de méthodes statistiques et des facilités graphiques importantes.

C'est un clône gratuit du logiciel S-Plus commercialisé par MathSoft et développé par Statistical Sciences autour du langage S (conçu par les laboratoires Bell).

Initié dans les années 90 par Robert Gentleman et Ross Ihaka (Département de Statistique, Université d'Auckland, Nouvelle-Zélande), auxquels sont venus depuis s'ajouter de nombreux chercheurs, le logiciel R constitue aujourd'hui un langage de programmation intégré d'analyse statistique. Le site Internet de la "R core-development Team", `http://www.r-project.org`, est la meilleure source d'informations sur le logiciel R. Vous pourrez y trouver les différentes distributions du logiciel, de nombreuses bibliothèques de fonctions et des documents d'aide.

2 Interface d'utilisation sous Windows

L'application `Rgui.exe` forme une interface utilisateur simple pour l'environnement R. Elle est structurée autour d'une barre de menu et de diverses fenêtres.

Les menus sont très peu développés. Leur objectif est de fournir un raccourci vers certaines commandes parmi les plus utilisées, mais leur usage n'est pas exclusif, ces mêmes commandes pouvant être pour la plupart exécutées depuis la console.

Le menu `File` contient les outils nécessaires à la gestion de l'espace de travail, tels que la sélection du répertoire par défaut, le chargement de fichiers sources externes, la sauvegarde et le chargement d'historiques de commandes, etc...

Il est recommandé d'utiliser un répertoire différent pour chaque projet d'analyse afin d'éviter la confusion dans les données et les programmes au fur et à mesure de l'accumulation des projets successifs.

Le menu `Edit` contient les habituelles commandes de copier-coller, ainsi que la boîte de dialogue autorisant la personnalisation de l'apparence de l'interface, tandis que le menu `Misc` traite de la gestion des objets en mémoire et permet d'arrêter une procédure en cours de traitement.

Le menu `Packages` automatise la gestion et le suivi des librairies de fonctions, permettant leur installation et leur mise à jour de manière transparente au départ du site CRAN (Comprehensive R Archive Network, `http://cran.r-project.org/`) ou de toute autre source locale.

Enfin, les menus `Windows` et `Help` assument des fonctions similaires à celles qu'ils occupent dans les autres applications Windows, à savoir la définition spatiale des fenêtres et l'accès à l'aide en ligne et aux manuels de références de R.

Parmi les fenêtres, on distingue la console, fenêtre principale où sont réalisées par défaut les entrées de commandes et sorties de résultats en mode texte. À celles-ci peuvent s'ajouter un certain nombre de fenêtres facultatives, telles que les fenêtres graphiques et les fenêtres d'informations (historique des commandes, aide, visualisation de fichier, etc...), toutes appelées par des commandes spécifiques via la console.

3 Les objets

Les éléments de base du langage R sont des objets qui peuvent être des données (vecteurs, matrices, séries chronologiques...), des fonctions, des graphiques...

Les objets R se différencient par leur mode, qui décrit leur contenu, et leur classe, qui décrit leur structure. Les objets atomiques sont de mode homogène et les objets récursifs sont de mode hétérogène.

Les différents modes sont :

```
null (objet vide), logical, numeric, complex, character
```

Les principales classes d'objets sont :

```
vector, matrix, array, factor, time-series, data.frame, list
```

On peut avoir des vecteurs, matrices, tableaux, variables catégorielles, séries chronologiques de mode `null (objet vide), logical, numeric, complex, character` mais une liste ou un tableau de données peuvent être hétérogènes.

3.1 Les vecteurs

Il s'agit de l'objet de base dans R. Un vecteur est une entité unique formée d'une collection ordonnée d'éléments de même nature. Dans R, les vecteurs peuvent être constitués d'éléments numériques, logiques ou alphanumériques.

La constitution manuelle d'un vecteur est réalisée grâce à la fonction `c(el1,el2, ...)`. Les nombres décimaux doivent être encodés avec un point décimal, les chaînes de caractères entourées par des guillemets doubles `" "`, et les valeurs logiques sont codées par les chaînes de caractères `TRUE` et `FALSE` ou leurs abréviations respectives `T` et `F`. Enfin, les données manquantes sont codées par la chaîne de caractères `NA`.

À partir de la console, vous allez maintenant taper les séquences suivantes au clavier (excepté le premier caractère `>`, invite de R signalant que la console est en attente d'une instruction, ou encore le caractère + signalant que la console attend la suite d'une instruction).

`> a<-c(5,5.6,1,4,-5)`	Création de l'objet `a` recevant un vecteur numérique de dimension 5 et de cordonnées 5,5.6,1,4,-5
`> a`	Affichage du vecteur `a`
`> a[1]`	Affichage de la première coordonnée du vecteur `a`
`> b=a[2:4]`	Création du vecteur numérique `b` de dimension 3 et de coordonnées 5.6,1,4
`> b`	
`> a[2,4]`	Injures
`> d=a[c(1,3,5)]`	Création du vecteur numérique `d` de dimension 3 et de coordonnées 5,1,-5
`> d`	
`> 2*a`	Multiplication par deux de chacune des coordonnées du vecteur `a` et affichage du résultat
`> e=3/d`	Création du vecteur numérique `e` de dimension 3 et de coordonnées 3/5,3,-3/5
`> e`	

`> f=a-4`	Création du vecteur `f` de dimension 5 dont les coordonnées sont égales à celles de `a` moins 4
`> d=d+e`	Remplacement du vecteur `d` par le vecteur résultant de la somme des vecteurs `d` et `e`
`> d=d-e`	Remplacement du vecteur `d` par le vecteur résultant de la différence entre les vecteurs `d` et `e`
`> d*e`	Multiplication terme à terme des vecteurs `d` et `e`
`> e/d`	Division terme à terme entre les vecteurs `e` et `d`
`> sum(d)`	Calcul de la somme de `d`
`> length(d)`	Affichage de la dimension du vecteur `d`
`> t(d)`	Transposition du vecteur `d`, le résultat est un vecteur ligne
`> t(d)%*%e`	Produit scalaire entre le vecteur ligne `t(b)` et le vecteur colonne `e`
`> g=c(sqrt(2),log(10))`	Création du vecteur numérique `g` de dimension 2 et de coordonnées $\sqrt{2}, \log(10)$
`> (1+exp(2))/cos(8)`	R est aussi une calculatrice
`> bool=d==5`	Création du vecteur booléen `bool` de dimension 3 recevant `TRUE` si `d[i]=5` et `FALSE` sinon
`> text=c("grand","petit")`	Création du vecteur caractère `text` de dimension 2 de coordonnées `grand`,`petit`
`> is.vector(d)`	Utilisation de la fonction `is.vector()` renvoyant l'expression logique `TRUE` si son argument est un vecteur et `FALSE` sinon

Quelques fonctions utiles :

```
> rep(1:4,2)
> rep(c(1.4,3,5),each=2)
> ?rep
> seq(0,1, length=11)
> seq(1.575, 5.125, by=0.05)
> help(seq)
> sort(b)
> sample(c(0,1),10,rep=T)
> sample(c(1,2,3),3)
> letters[2]
> LETTERS[2]
> sample(letters[1:9],5)
> order(d)
> help(order)
> help.start()
> abs(d)
> log(abs(d))
```

3.2 Les matrices

Les matrices comme les vecteurs, sont de mode quelconque, mais elles ne peuvent pas contenir des éléments de nature différente. La syntaxe de création d'une matrice est la suivante : `matrix(vec,nrow=n,ncol=p,byrow=T)` où `vec` est le vecteur contenant les éléments de la matrice, qui seront rangés en colonne (sauf si l'option `byrow=T` est choisie).

`> a=1:20`	
`> b=sample(1:10,10)`	
`> x1=matrix(a,nrow=5)`	Création de la matrice numérique `x1` de dimension 5×4 ayant pour première ligne 1,6,11,16
`> x2=matrix(a,nrow=5,byrow=T)`	Création de la matrice numérique `x2` de dimension 5×4 ayant pour première ligne 1,2,3,4
`> x3=t(x2)`	Transposition de la matrice `x2`
`> x4=matrix(b,ncol=2)`	
`> x1;x2;x3;x4`	
`> b=x3%*%x2`	Produit matriciel entre `x2` et `x3` le langage contrôle l'adéquation des dimensions
`> f=x2%*%x4`	Injures
`> b`	
`> f`	
`> dim(x1)`	Affichage de dimension de la matrice `x1`
`> dim(x4)`	
`> b[3,2]`	Sélection de l'élément `[3,2]` de la matrice `b`
`> b[,2]`	Sélection de la deuxième colonne de `b`
`> b[c(3,4),]`	Sélection des troisième et quatrième lignes de `b`
`> b[-2,]`	Suppression de la deuxième ligne de `b`
`> b[,-c(2,4)]`	Suppression des deuxième et quatrième colonnes de `b`
`> b[,2]`	Sélection de la deuxième colonne de `b`
`> a=sample(1:10,30,rep=T)`	
`> a=matrix(a,nrow=6)`	
`> a`	
`> a>5`	Affichage d'une matrice de booléen dont l'élément `[i,j]` est égal à `T` si `a[i,j]>5`
`> a[a<5]=0`	On annule les éléments de `a` inférieurs à 5
`> rbind(x1,x2)`	Concaténation verticale des matrices `x1` et `x2`
`> cbind(x1,x4)`	Concaténation horizontale des matrices `x1` et `x4`
`> apply(x1,2,sum)`	Calcul de la somme de `x1` par colonne
`> apply(x1,1,sum)`	Calcul de la somme de `x1` par ligne

3.3 Les matrices à plus de deux dimensions

Les matrices à plus de deux dimensions sont de mode atomique. Elles sont créées à l'aide de la commande suivante : `array(vec,c(n,p,q...))` où `vec` est le vecteur contenant les éléments de la matrice qui seront rangés en colonne et l'argument `c(n,p,q...)` désigne les dimensions : `n` est le nombre de lignes, `p` le nombre de colonnes, `q` le nombre de matrices...

```
> x=array(1:50,c(2,5,5))
> x
> x[1,2,2]
> dim(x)
> aperm(x)   Transposition généralisée de x, x[i,j,k] devient x[k,j,i]
```

3.4 Les listes

Une liste est une collection ordonnée d'objets, non nécessairement de même mode. Les éléments des listes peuvent donc être n'importe quel objet défini dans R. Cette propriété est notamment utilisée par certaines fonctions pour renvoyer des résultats complexes sous forme d'un seul objet.

La constitution d'une liste passe par la fonction `list(nom1=el1,nom2=el2,...)`, l'utilisation des noms étant facultative. On peut accéder à chaque élément de la liste à l'aide de son index entre double crochets `[[...]]`, ou par son nom précédé du signe `$`.

```
> li=list(num=1:5,y="couleur",a=T)
> li
> li$num
> li$a
> li[[1]]
> li[[3]]
> a=matrix(c(6,2,0,2,6,0,0,0,36),nrow=3)
> eigen(a,symmetric=T)                                  Diagonalisation de a
> res=eigen(a,symmetric=T)
> res$values
> res$vectors
> a
> diag(res$values)
> res$vectors%*%diag(res$values)%*%t(res$vectors)
```

3.5 Les structures de données

La structure de données R de nom `data.frame` est analogue à une matrice dont les colonnes peuvent être hétérogènes. Les tableaux de données (`data.frame`) constituent une classe particulière de listes consacrée spécifiquement au stockage des données destinées à l'analyse. Chaque composante de la liste forme alors l'équivalent d'une colonne ou variable, tandis que les différents éléments de ces composantes correspondent aux lignes ou individus. À cette classe d'objets est associée une série de méthodes spécifiques.

Pour créer un tableau de données, on peut regrouper des variables de même longueur à l'aide de la commande `data.frame(nom1=var1,nom2=var2,...)`. Aussi, il est possible de transformer une matrice en tableau de données en utilisant la commande `as.data.frame(mat)`. Enfin, on peut utiliser un fichier externe (voir section 9).

`> v1=sample(1:12,30,rep=T)`	Échantillonnage avec remise dans les entiers de 1 à 12
`> v2=sample(LETTERS[1:10],30,rep=T)`	
`> v3=runif(30)`	30 réalisations indépendantes d'une loi uniforme sur [0,1] (voir section 4)
`> v4=rnorm(30)`	30 réalisations indépendantes d'une loi normale de moyenne 0 et variance 1
`> v1;v2;v3;v4`	
`> xx=data.frame(v1,v2,v3,v4)`	
`> xx`	
`> xx$v2`	
`> summary(xx)`	
`> xx=data.frame(v1,factor(v2),v3,v4)`	Constitution du tableau de données `xx` avec la variable `x2` déclarée comme facteur (variable qualitative)
`> summary(xx)`	
`> ma=matrix(1:15,nrow=3);ma`	
`> plot(ma)`	
`> ma=as.data.frame(ma)`	
`> is.data.frame(ma)`	
`> plot(ma)`	
`> data()`	Ouvre une fenêtre texte listant l'ensemble des tableaux de données disponibles dans R
`> data(women)`	Charge le tableau de données `women`
`> names(women)`	Affiche le nom des variables de `women`
`> attach(women)`	
`> height`	

4 Les distributions usuelles

Loi	Nom	Paramètres	Valeurs par défaut
Beta	`beta`	`shape1, shape2`	
Binomiale	`binom`	`size, prob`	
Cauchy	`cauchy`	`location, scale`	0, 1
Khi-Deux	`chisq`	`df`	
Exponentielle	`exp`	`1/mean`	1
Fisher	`f`	`df1, df2`	
Gamma	`gamma`	`shape,1/scale`	-, 1
Géométrique	`geom`	`prob`	
Hypergéométrique	`hyper`	`m, n, k`	
Log-Normale	`lnorm`	`mean, sd`	0, 1
Logistique	`logis`	`location, scale`	0, 1
Normale	`norm`	`mean, sd`	0, 1
Poisson	`pois`	`lambda`	
Student	`t`	`df`	
Uniforme	`unif`	`min, max`	0, 1
Weibull	`weibull`	`shape`	

Pour chacune de ces distributions, on dispose de quatre commandes préfixées par une des lettres `d, p, q, r` et suivi du nom de la distribution :

`dnomdist` : il s'agit de la fonction de densité pour une distribution de probabilité continue et de la fonction de probabilité ($\mathbb{P}(X = k)$) dans le cas d'une distribution discrète ;

`pnomdist` : il s'agit de la fonction de répartition ($\mathbb{P}(X < x)$) ;

`qnomdist` : il s'agit de la fonction des quantiles, c'est-à-dire la valeur pour laquelle la fonction de répartition atteint une certaine probabilité ; dans le cas discret, cette fonction renvoie le plus petit entier u tel que $F(u) \geq p$ où F est la fonction de répartition de la distribution considérée ;

`rnomdist` : génère des réalisations aléatoires indépendantes de la distribution `nomdist`.

```
> qnorm(0.975);dnorm(0);pnorm(1.96)
> rnorm(20);rnorm(10,mean=5,sd=0.5)
> x=seq(-3,3,0.1);pdf=dnorm(x)
> plot(x,pdf,type="l")
> runif(3)
> rt(5,10)
```

5 Quelques fonctions de statistique exploratoire

`> data(women)`	
`> names(women)`	
`> attach(women)`	
`> mean(height)`	Calcul de la moyenne empirique de la variable quantitative `height`
`> var(height)`	Calcul de la variance empirique de `height` estimateur non biaisé (diviseur $n-1$)
`> sd(height)`	Calcul de l'écart-type de `height`
`> median(height)`	Calcul de la médiane empirique de `height`
`> quantile(height)`	Calcul des quantiles empiriques de `height`
`> summary(weight)`	Résumé de `weight`
`> summary(women)`	Résumé de `women`
`> hist(weight,nclass=15)`	Histogramme de `weight` constitué de 15 classes
`> boxplot(weight)`	Diagramme en boîte de `weight`
`> cor(height,weight)`	Calcul du coefficient de corrélation linéaire empirique entre `weight` et `height`
`> v1=rnorm(100)`	
`> hist(v1)`	
`> v2=factor(sample(letters[1:4],100,rep=T))`	
`> table(v2)`	Résumé de la variable qualitative `v2`
`> barplot(table(v2))`	Diagramme en barre de `v2`
`> pie(table(v2))`	Diagramme en secteur de `v2`
`> boxplot(v1~v2)`	Diagramme en boîte de `v1` pour chaque modalité de `v2`

6 Les principales fonctions génériques

Il s'agit de fonctions qui s'appliquent à tous les types d'objets, mais qui exécutent une commande spécifique en fonction de la classe de l'objet concerné. Les trois principales fonctions génériques sont :

`print` qui optimise l'affichage écran de différents objets,

`plot` qui réalise des représentation graphiques,

`summary` qui renvoie un résumé sur le contenu d'un objet.

En pratique, les fonctions réellement exécutées sont différentes pour différentes classes d'objets. Ainsi, en tapant `print(x)`, on fait appel à la fonction `print.default` si `x` est un vecteur, à la fonction `print.ts` si `x` est une série chronologique, à la fonction `print.glm` si `x` est le résultat de la mise en oeuvre d'un modèle linéaire généralisé, à la fonction `print.aov` si `x` est la résultat de la mise en oeuvre d'une analyse de la variance...

```
> x=rnorm(20)
> print(x)
> summary(x)
> y=3*x+5+rnorm(20,sd=0.3)
> plot(x,y)
> reslm=lm(y~x)
```

Régression linéaire simple entre la variable à expliquer `y` et la variable explicative `x`

```
> print(reslm)
> summary(reslm)
> plot(reslm,ask=T)
```

7 Construction d'une nouvelle fonction

Il est possible de définir des fonctions personnalisées soit directement au départ de la console, soit via un éditeur de texte externe (par défaut le bloc-notes) grâce à la commande `fix(nom_fonction)`. La seconde possibilité permet la correction du code en cours d'édition, tandis que la première s'effectue ligne par ligne, sans retour en arrière possible.

De manière générale, la définition d'une nouvelle fonction passe par l'expression suivante :

```
nom_fonction=function(arg1[=expr1],arg2[=expr2],...) {
bloc d'instructions
}
```

Les accolades signalent à l'interpréteur de commande le début et la fin du code source de la fonction ainsi définie, tandis que les crochets ne font pas partie de l'expression mais indiquent le caractère facultatif des valeurs par défaut des arguments.

Il est également possible de créer une fonction personnalisée à partir d'une fonction existante, tout en conservant l'original intact grâce à :

```
nom_fonction2=edit(nom_fonction1);fix(nom_fonction2)
```

Lors de l'exécution, R renvoie par défaut le résultat de la dernière expression évaluée dans la fonction. Par ailleurs, les arguments sont passés à la fonction par valeur et leur portée ainsi que celle de toute assignation classique à l'intérieur d'une fonction est locale.

Lors de l'exécution, une copie des arguments est transmise à la fonction, laissant les originaux intacts.

```
> x=2
> carre=function(x) { x=x*x ; x }
> carre(2)
> x
> fix(carre)
```

Enfin, ajoutons qu'il est possible voire recommandé d'ajouter des commentaires au code des fonctions, en les faisant précéder du symbole dièse `#`. La suite de la ligne est alors ignorée lors de l'interprétation de la fonction et peut donc être complétée par un commentaire libre.

8 Quelques éléments de programmation

Nous traitons succinctement dans cette section des instructions de sélection et de répétitions. Il s'agit des commandes `if, while, for`

```
> bool=T
> i=0
> while(bool==T) {i=i+1; if (i>10) {bool=F}}
> i
> s=0
> x=rnorm(10000)
> for (i in 1:10000) {s=s+x[i]}
> s
> un=rep(1,10000)
> t(un)%*%x
> s=0
> system.time(for (i in 1:10000) {s=s+x[i]})[3]
> system.time(t(un)%*%x)[3]
```

R peut avoir un problème de mémoire si vous faites appel à un nombre très élevé de boucles, même si elles contiennent des instructions très simples. En effet, comme les deux dernières commandes le montrent, l'utilisation de boucles est très coûteuse en temps de calcul. Il est ainsi indispensable de limiter l'utilisation des boucles en les remplaçant par les outils du calcul matriciel (les opérateurs matriciel de R utilisent des boucles C beaucoup plus rapides).

9 Les entrées et sorties

Dans de nombreux cas, les données que l'on souhaite analyser proviennent de sources externes sous forme de fichiers. Aussi, les objets créés doivent pouvoir être sauvegardés dans des fichiers afin qu'ils soient complètement transportables.

C'est ainsi que divers outils d'accès aux fichiers ont été développés sous R. On distingue les accès aux fichiers propriétaires de R, aux fichiers ASCII, aux fichiers provenant d'autres logiciels d'analyse statistique, aux bases de données relationnelles et aux fichiers binaires.

Formats propriétaire

La fonction générique `save()` autorise la sauvegarde de n'importe quelle liste d'objets en mémoire, sous un chemin quelconque, aussi bien en format binaire qu'ASCII. Ces objets peuvent ensuite être rechargés en mémoire grâce à la fonction `load()` qui est le pendant de la première.

Par ailleurs, pour des raisons de compatibilité avec le langage S, il existe la fonction `dump()` qui permet d'exporter des objets vers un fichier texte pouvant alors être relu et interprété séquentiellement par la fonction `source()`. Cette dernière fonction est très utilisée en programmation car elle autorise la lecture d'une série d'instructions sauvegardées dans un fichier texte à l'aide d'un éditeur externe.

Fichiers textes ASCII

Le format d'échange le plus courant en ce qui concerne les données brutes reste le fichier ASCII. La lecture de tels fichiers est prise en charge par la commande élémentaire `scan()`. Les arguments de cette fonction permettent de décrire précisément la structure du fichier texte.

Afin de faciliter la lecture des fichiers de données structurées en colonnes plusieurs commandes spécifiques ont été développées à partir de la fonction `scan()`. Ces fonctions (`read.table()` et ses dérivés) automatisent la lecture des fichiers de données ASCII standards (csv, texte délimité, largeur fixe ...) et stockent leurs résultats dans des `data.frame`. Bien qu'elles soient plus spécifiques que `scan()`, ces fonctions conservent une grande adaptativité grâce à l'utilisation de nombreux arguments permettant de préciser le format interne du fichier (présence de titre de colonnes, type de séparateur ...).

L'exportation de tableaux de données sous forme de fichiers ASCII standards peut être réalisée par la fonction `write.table()`.

Logiciels statistiques

Lorsque les données ont été sauvegardées sous le format propriétaire d'un logiciel statistique tiers, il est nécessaire de disposer d'outils permettant leur transfert vers le système R. La librairie `foreign` offre ces outils pour une sélection des logiciels statistiques les plus courants, à savoir Minitab, S-Plus, SAS, SPSS et Stata.

Par exemple, la fonction `read.spss` prend en charge les données enregistrées au moyen des commandes `save` et `export` de SPSS.

Bases de données relationnelles

La version de base de l'environnement R n'est pas adaptée à la gestion de très grosses quantités de données. En effet, tous les objets sont chargés intégralement en mémoire centrale et plusieurs copies de ces objets peuvent être créées lors de l'exécution des procédures de traitement, ce qui peut entraîner une saturation du système dès que la taille totale des jeux de données dépasse une certaine fraction de l'espace mémoire disponible. De plus, il ne permet pas à plusieurs utilisateurs d'accéder aux mêmes données de manière concurrente, c'est-à-dire en intégrant en temps réel les modifications des uns et des autres.

Ce travail de gestion est d'ordinaire le domaine de prédilection des systèmes de gestion de bases de données (SGBD), et plus particulièrement des SGBD relationnels.

Une série de modules faisant office d'interface entre ces SGBD et l'environnement R ont été développés. On peut ainsi trouver diverses librairies de fonctions dédiées au pilotage des SGBD.

10 Gestion des objets crées

Lors de l'exécution de R, les fichiers `.RData` et `.Rhistory` sont automatiquement créés dans le répertoire de travail. Aussi, lorsque l'on quitte R à l'aide de la commande `q()`, les nouveaux objets créés peuvent être sauvegardés dans le fichier `.RData` (le choix est donné à l'utilisateur) en mode binaire et la suite des commandes que l'on a tapée est automatiquement sauvegardée dans le fichier `.Rhistory` en mode texte. Ainsi, lorsque l'on relance R dans ce même répertoire, le fichier `.RData` est automatiquement chargé et l'on retrouve l'intégralité des objets que l'on y a créés. Par ailleurs, on peut visualiser la suite de commandes que l'on a tapée à l'aide de l'instruction `history()` dans R ou en éditant simplement le fichier `.Rhistory`.

Pour connaître le répertoire de travail courant au cours d'une session, il suffit de taper `getwd()` et pour en changer, il faut utiliser la commande `setwd()`.

Pour exécuter R dans le répertoire de travail de son choix, on utilise les commandes Windows classiques sur l'icône de raccourci.

Enfin, la commande `ls()` permet de visualiser la liste des objets créés et la commande `rm()` permet de détruire des objets.