Année : 2008-2009 1er semestre
Niveau : MASTER IS 2ème année
Cours : Remise à niveau en R
Enseignant : A. Illig

TP2 : Remise à niveau en R

Table des matières

1 Les objets de R

1.1 Objets, modes et longueurs

Le logiciel R manipule plusieurs *objets* : `vector, factor, array, matrix, data.frame, time-series, list` décrits dans le tableau ci-dessous. Chaque objet dispose de deux attributs intrinsèques : *mode*, *longueur*. Le *mode* correspond au type des éléments d'un objet. Il existe quatre *modes* principaux : `numeric, complex, logical, character`.

Objet	Description	Exemples

`vector`	Vecteur au sens classique (éléments de même *mode*)	`> x=c(0,1,-9,7)`
		`> y=c(4 :7)`
		`> z=seq(-3,2,0.2)`
		`> t=rep(x,times=2)`
		`> text=c(‘‘grand’’,’’petit’’)`

`factor`	Variable catégorique (éléments de *modes* identiques : `numeric` ou `character`)	`> f=factor(1 :3,levels=1 :5, exclude=4)`
`array`	Tableau k-dimensionnel (éléments de même *mode*)	`> A=array(1 :20,dim=c(5,4))`
`matrix`	Cas particulier de `array` avec $k = 2$	`> B=matrix(c(1,-1,8,3), nrow=1,ncol=4)` `> C=matrix(0,nrow=8,ncol=2)`
`data.frame`	Tableau de données composé d'un ou plusieurs vecteur(s) et/ou facteur(s) ayant même longueur mais des *modes* pouvant être différents	`> Donnees=data.frame(D1=x, D2=y)` `> Donnees$D1`
`ts`	Données de type série temporelle	`> Serie1=ts(z,start=1942)` `> Serie2=ts(z,freq=12, start=c(1942,3))`
`list`	Liste d'éléments de tout *mode*	`> L=list(x,z,text)`

Les commandes `mode(x)` et `length(x)` affichent le *mode* et la *longueur* de l'objet `x` :

```
> x=c(0,1,-9,7)
> x
> mode(x)
> length(x)
```

Les commandes `ls.()` et `objects()` ont le même rôle et affichent une liste des objets enregistés dans la mémoire courante. La commande `ls.str()` permet d'afficher les différents objets en mémoire ainsi que leurs caractéristiques. Par exemple, après avoir enregistré les objets du tableau précédent, la commande `ls.str()` affiche :

```
> ls.str()
A :  int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
bool :  logi [1:2]  TRUE FALSE
B :  num [1, 1:4] 1 -1 8 3
C :  num [1:8, 1:2] 0 0 0 0 0 0 0 0 0 0 ...
Donnees : 'data.frame': 4 obs. of  2 variables:
 $ D1: num  0 1 -9 7
 $ D2: int  4 5 6 7
f :  Factor w/ 4 levels "1","2","3","5": 1 2 3
L : List of 3
 $ : num [1:4] 0 1 -9 7
 $ : num [1:26] -3.0 -2.8 -2.6 -2.4 -2.2 ...
 $ : chr [1:2] "grand" "petit"
Serie1 :  Time-Series [1:26] from 1942 to 1967: -3.0 -2.8 -2.6 -2.4 -2.2 ...
Serie2 :  Time-Series [1:26] from 1942 to 1944: -3.0 -2.8 -2.6 -2.4 -2.2 ...
t :  num [1:8] 0 1 -9 7 0 1 -9 7
text :  chr [1:2] "grand" "petit"
x :  num [1:4] 0 1 -9 7
y :  int [1:4] 4 5 6 7
z :  num [1:26] -3.0 -2.8 -2.6 -2.4 -2.2 ...
```

Enfin, la commande `rm(x,y)` efface les objets `x` et `y`. Pour effacer tous les objets en mémoire, on utilise la commande `rm(ls())`.

1.2 Vecteurs, matrices et tableaux

La fonction `c()` est utilisée pour créer un vecteur. Vous pouvez taper

```
> x=c(0,4,7)
```

et afficher le résultat

```
> x
[1] 1 4 7
```

D'autres méthodes peuvent être utilisées pour créer des vecteurs :

```
> # Suites croissantes ou décroissantes d'entiers
> nombresde20a13=20:13
> # Concaténation de vecteurs
> c(x,nombresde20a13)
> pleindenombres=c(nombresde20a13,4:10)
```

Maintenant, comment extraire certains éléments d'un vecteur ? Voici quelques exemples :

```
> # Le 10 ème élément de pleindenombres
> pleindenombres[10]
> # Certains éléments bien choisis
```

```
> pleindenombres[c(1,6,4)]
> # Tous les éléments sauf le 3 ème
> pleindenombres[-3]
> # Tous les éléments sauf les 3 derniers
> pleindenombres[-(length(pleindenombres)-3:length(pleindenombres))]
```

Pour remplir une matrice, on utilise la fonction `matrix()` :

```
> m=matrix(1:6,nrow=2,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Comme pour les vecteurs, on extrait facilement certains éléments d'une matrice :

```
> m[1,3]
[1] 5
> m[,2]
[1] 3 4
```

Les tableaux jouent le même rôle que les matrices à la différence près qu'ils peuvent avoir plus de deux indices :

```
> a=array(1:12,c(3,2,2))
> a
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

Afin de savoir si `x`, `m` ou `a` est un vecteur, une matrice, un tableau, ... on utilise les fonctions `is.vector(x)`, `is.matrix(x)`, `is.array(x)` ... :

```
> bool=c(is.vector(x),is.matrix(x),is.array(m))
> bool
> mode(bool)
> length(bool)
```

2 Programmation en R

2.1 Boucles et vectorisation en R

R est un logiciel qui permet de programmer simplement l'éxécution successive de plusieurs commandes. Comme les autres langages, il dispose de boucles et de structures de contrôle ($if \ldots else$, for, $while$, $repeat$...) analogues à celles du langage C. Les quelques exemples suivants illustrent l'usage de ces différentes structures.

Exemple 1. Formons à partir d'un vecteur `x` un autre vecteur `y` qui prend la valeur 1 pour les valeurs de `x` égales à 3 et la valeur 0 sinon :

```
> x=c(3,2,-5,3,4,3)
> y=numeric(length(x))
> for (i in 1:length(x)){
+ if (x[i]==3){
+ y[i]=1}
+ else {y[i]=0}}
```

Exemple 2. Ajoutons les vecteurs `x` et `y` pour former un nouveau vecteur `z` :

```
> z=numeric(length(x))
> for (i in 1:length(x)){
+ z[i]=x[i]+y[i]}
```

Exemple 3. Augmentons de manière aléatoire la valeur d'un réel `a` jusqu'à dépasser la valeur arbitraire 10 :

```
> a=0
> bool=T
> while (bool==T){
+ a=a+runif(1)
+ if (a>10){bool=F}}
```

Dans ces exemples, on constate que le prompt `>` de R est remplacé par `+` après l'ouverture d'une boucle par l'accolade `{` et ceci jusqu' à la fermeture de la boucle par l'accolade `}`.

Dans la plupart des cas, les boucles peuvent être évitées au moyen d'une écriture vectorielle comme dans le cas de l'*Exemple 2.* :

```
> z=x+y
```

ou par une indexation logique comme dans le cas de l'*Exemple 1.* :

```
> y[x==3]=1
> y[x!=3]=0
```

Il est également possible de faire appel à la commande `apply(X,margin,fun)` où `X` est une matrice, `margin` indique si l'opération doit être appliquée sur les lignes (1), sur les colonnes (2) ou sur les deux ($c(1,2)$) et `fun` est une fonction (c.f. paragraphe suivant) ou une opération saisie entre guillemets doubles :

```
> X=cbind(x,y)
> A=apply(X,2,median)
> B=apply(X,1,sum)
> C=apply(X,2,sum)
> bool2=logical(1)
> bool2[length(B)==length(C)]=T
```

2.2 Ecrire une fonction, un programme en R

Les manipulations en R se font principalement au moyen de fonctions de R dont les arguments sont indiqués entre parenthèses. Tout utilisateur de R peut aussi créer ses propres fonctions qui viendront se rajouter, le temps de la session, à celles déjà enregistrées dans le logiciel. La fonction suivante permet de calculer le montant des mensualités que devra payer une personne pendant n mois pour emprunter au taux i une somme P aujourd'hui :

```
> mensualite=function(P,i,n){
+ # Fonction calculant la mensualite
+ # de l'emprunt d'une somme P au taux
+ # d'intérêt i sur n mois
+ (P*i)/(1-(1+i)^{-n})}
```

On appelle ensuite la fonction `mensualite` :

```
> mensualite(1500,0.01,12)
```

En pratique, il est préférable d'enregistrer les nouvelles fonctions dans un fichier texte *Nouvelles_fonctions.R* et de l'éxécuter ensuite dans R par la commande

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

Si les commandes du TP2 enregistrées dans le fichier *tpR2.R* font appel aux fonctions du fichier *Nouvelles_fonctions.R*, il est nécessaire d'éxécuter le fichier *Nouvelles_fonctions.R* avant le fichier *tpR2.R*.

3 Exercices d'application

3.1 Suite de Fibonacci

La suite de Fibonacci est la suite $(x_n)_{n\in\mathbb{N}^*}$ vérifiant :

$$\begin{cases} x_1 &= 1 \\ x_2 &= 1 \\ x_n &= x_{n-1} + x_{n-2} \end{cases}$$

1. En utilisant une boucle `for`, calculer les 12 premiers termes de la suite de Fibonacci.
2. Changez les deux premiers termes de la suite en 2 et 2 et calculer les 12 premiers termes de la nouvelle suite.
3. En utilisant une boucle `while`, listez tous les termes de la suite de Fibonacci inférieurs à 300.
4. Ecrire une fonction calculant les n pemiers termes de la suite de Fibonacci.
5. Calculer les 30 premiers termes de la suite de terme général $y_n = x_n/x_{n-1}$, $n = 2, \ldots \infty$.
6. La suite $(y_n)_{n\geq 2}$ est-elle convergente ? Calculer $\frac{1+\sqrt{5}}{2}$. Concluez.

3.2 Calcul d'intérêts

1. Ecrire une fonction calculant le montant des intérêts simples produit par un capital P placé pendant n années au taux d'intérêt annuel i.
2. Quel est le montant des intérêts simples produit par 100 euros placés pendant 3 ans à un taux d'intérêt de 4% ?
3. Ecrire une fonction calculant le montant des intérêts composés produit par un capital P placé pendant n années au taux d'intérêt annuel i.
4. Quel est le montant des intérêts composés produit par 100 euros placés pendant 3 ans à un taux d'intérêt de 4% ?

Rappel :
- Un capital produit des intérêts simples si les intérêts sont uniquement calculés sur ce capital.
- Un capital produit des intérêts composés si à la fin de chaque période, les intérêts générés sont ajoutés au capital pour produire de nouveaux intérêts. On dit aussi que les intérêts sont capitalisés.