Manipulation des vecteurs sous R

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Saisie, l'opérateur `c()` , accès aux valeurs…

CRÉATION ET OPÉRATIONS SUR LES VECTEURS

Vecteur

C'est l'objet de base dans R

On peut y mettre des numeric, logical, character…

Pour en connaître le type, on utilise class()

Création via une saisie console

Utilisation de la commande scan()

Tant que saisie de valeurs, la taille du vecteur s'accroit

Arrêt dès que l'utilisateur fait ENTREE 2 fois consécutivement

Création via la commande c() directement dans le code prog.

```
v <- c(1.2,2.5,3.2,1.8) #crée un vecteur de réels de taille 4
length(v) #renvoie le nombre d'éléments
```

C'est l'approche que tout le monde adopte pour la saisie de valeurs, ça ne pose pas de soucis parce R est un langage interprété.

De fait, nous allons utiliser un autre éditeur de code (soit celui de R, soit des outils tels que R-Studio, soit Eclipse avec StatET…).

```
#une suite arithmétique de raison 1
v <- 1 : 4 # v contient les valeurs (1, 2, 3, 4)
class(v) # renvoie numeric

#utilisation de la commande seq()
v <- seq(from=1,to=2,by=0.2) # v contient (1.0,1.2,1.4,1.6,1.8,2.0)

#on peut utiliser rep() également
v <- rep(1:2,times=2,each=2) # (1,1,2,2,1,1,2,2)

#vecteur de valeurs logiques
b <- c(T,T,F,T) # crée un vecteur de taille 4 contenant des logical
class(b) #renvoie logical
```

« elementwise »

Les opérations s'effectuent élément par élément

```
v1 ← c(1.2,1.3,1.0)
v2 ← c(2.1,0.8,1.3)
z ← v1 * v2 #renvoie (2.52,1.04,1.30)
```

« replication »

Si les vecteurs sont de taille différentes, R utilise le principe de la réplication c.-à-d. il ajuste la taille du vecteur court en dupliquant les valeurs

```
v1 ← c(2,4,5,3,1)
v2 ← c(1,4,8)
z ← v1 * v2 #renvoie (2,16,40,3, 4)
```

v2 a été *rallongé* en interne en c(1,4,8,1,4)

« replication » pour les scalaires

Un scalaire est un vecteur de taille 1. Utilisé dans une opération, il est dupliqué autant de fois que nécessaire

```
v2 <- c(1,4,8)
z <- v2 +1 #renvoie (2,5,9)
#parce que 1 a été répliqué en (1,1,1)
```

Opérations sur les vecteurs de booléens

Les opérateurs ET et OU s'appliquent aux vecteurs de booléens, mais ils s'écrivent différemment : **&** et **|** (une seule fois)

```
b1 ← c(T,F,T,T)
b2 ← c(F,F,T,F)
b ← b1 & b2 #renvoie (F, F, T, F)
```

En revanche…

```
b ← b1 && b2 #renvoie FALSE
```

R n'utilise que la 1ère valeur dans les 2 vecteurs

Opérateurs pouvant s'appliquer sur des vecteurs

Ils sont nombreux et très performants. Ex. sum(), length(), mean(), max(), etc.

Sans titre - Editeur R

```
#un vecteur
v <- c(1.5,0.2,0.9,2.8,3.5,9.5,1.4)
#3e quartile
q3 <- quantile(v,probs=0.75)
print(q3)
#1er et 3e quartiles
q13 <- quantile(v,probs=c(0.25,0.75))
print(q13)
print(length(q13))
```

R Console

```
> #un vecteur
> v <- c(1.5,0.2,0.9,2.8,3.5,9.5,1.4)
> #3e quartile
> q3 <- quantile(v,probs=0.75)
> print(q3)
 75%
3.15
> #1er et 3e quartiles
> q13 <- quantile(v,probs=c(0.25,0.75))
> print(q13)
 25%  75%
1.15 3.15
> print(length(q13))
[1] 2
>
```

Editeur de code R

Sortie console

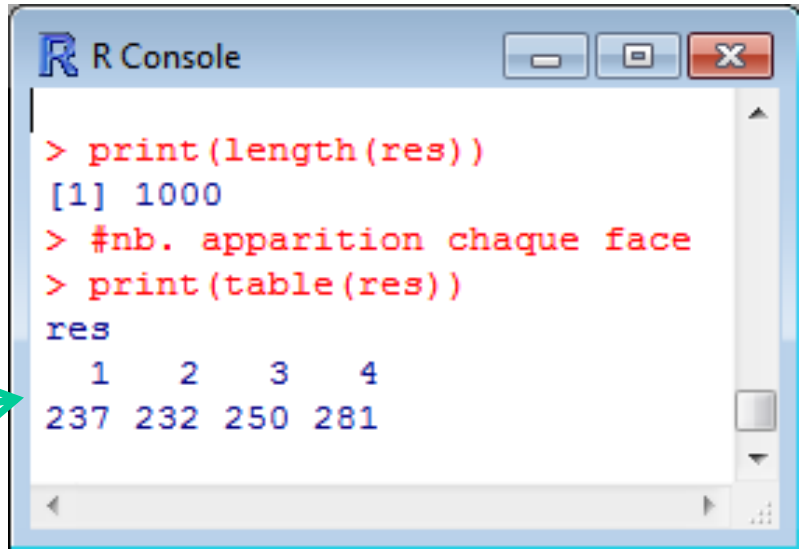
replicate permet de lancer n fois une expression ou une fonction, typiquement une simulation, et collecter les résultats dans une structure appropriée (ex. une simulation = une valeur ➔ les résultats sont collectés dans un vecteur).

Ex. Simulation de lancer de dés à « m » faces

```
#fonction paramétré par m
#un dé, par défaut c'est m = 6 faces
lance.un.de <- function(m=6){
 #runif génère 1 valeur selon U(0,1)
 #trunc prend la partie entière
 return(1+trunc(m*runif(1)))
}
#lancer 1000 fois un dé à 4 faces (!)
res <- replicate(1000,lance.un.de(m=4))
print(length(res))
#nb. apparition chaque face
print(table(res))
```

L'outil est vraiment très souple.

Pas besoin d'utiliser une boucle !



On peut accéder aux valeurs à l'aide d'un indice (entier)

Le 1er indice est 1

Le dernier correspond à length()

```
v <- c(1.5,0.2,0.9,2.8,3.5,9.5,1.4)

v[3] # affiche 0.9

a <- v[1] # affecte à a la valeur 1.5

v[4] <- 2 * v[3] #remplace 2.8 par 1.8
```

On peut avoir besoin d'un vecteur sans en connaître la taille à l'avance.

2 étapes : (1) création de l'objet vide ; (2) l'insertion à une case inexistante allonge automatiquement le vecteur.

Note : une case non affectée contient alors la valeur NA.

```
w <- double(0) #prépare le pointeur et effectue le typage
w[3] <- 1.2
print(w) #affiche (NA, NA, 1.2), la taille du vecteur est ajustée !!!
```

Autres initialisations :

```
w <- NULL # pointeur nul
w <- c() #idem, pointeur nul
```

→ Le typage se fait alors lors de la 1ère affectation

Concaténation de 2 vecteurs

L'opérateur c() permet de concaténer 2 ou plusieurs vecteurs

```
x <- c(1.2,3.3)

y <- c(1.3,4.2,1.8)

z <- c(x,y) #renvoie (1.2, 3.3, 1.3, 4.2, 1.8)
```

Extraire un sous-ensemble de valeurs d'un vecteur

EXTRACTIONS

L'accès indicé vu précédemment est un cas particulier

On peut utiliser une plage d'indices pour extraire un sous-vecteur

```
v <- c(1.2, 2.3, 4.2, 8.5, 6.3)
```

Indices contigus

```
v[2:4] # (2.3, 4.2, 8.5)
m <- 1:3 #m représente un vecteur d'indices (1,2,3)
v[m] # (1.2,2.3,4.2)
```

Indices non-contigus

```
v[c(1:3,5)] # (1.2,2.3, 4.2, 6.3)
```

Indices négatifs

```
v[-2] # (1.2,4.2,8.5, 6.3), tous sauf la 2ème valeur
v[-(2:4)] #(1.2,6.3), ne conserve que la 1ère et la 5ème
```

On peut affecter les valeurs extraites à un autre vecteur

```
x <- v[c(1,4:5)]
print(x) # (1.2, 8.5, 6.3)
```

On peut utiliser un vecteur de booléens pour l'extraction

Vecteur de booléens

```
v <- c(1.2, 2.3, 4.2, 8.5, 6.3)
b <- c(T,T,F,F,T)
v[b] #(1.2, 2.3, 6.3)
```

Tailles non conformes des vecteurs

```
b <- c(T,F)
v[b] #(1.2, 4.2, 6.3) car b est répliqué en (T,F,T,F,T)
```

```
d <- c(T,F,F,F,T,T) #d est trop long
v[d] #(1.2, 6.3,NA) : NA pour les cases manquantes
```

Une condition correspond à un vecteur de booléens

```
z <- c(0.1, 1.5, 3.6, 1.1, 3.2)
v[(z > 2)] #(4.2,6.3), parce que (z > 2) ➔ (F,F,T,F,T)
v[(z <1 | z >3.5)] #(1.2,4.2), parce que (T,F,T,F,F)
```

Il est possible d'attribuer des noms aux cases du vecteur avec l'instruction names()

```
v <- c(1.2, 2.3, 4.2, 8.5, 6.3)
names(v) <- c(« thierry », « david », « louis », « nicolas », « franck »)
x <- v[« david »]
print(x) #x vaut 2.3
#on peut également procéder à une affectation par nom
v[« thierry »] <- 10
print(v) # donne (10.0, 2.3, 4.2, 8.5, 6.3)
```

Le type FACTOR pour la représentation des variables qualitatives

Un factor est un vecteur d'entiers : (1) chaque valeur correspond à un code de modalité ; (2) on peut attribuer des noms aux modalités

```
v ← c(1, 2, 1, 1, 2)
# 1 signifie « masculin » et 2 « féminin ». Comment indiquer cela à R ?
sexe ← factor(v) #sexe est un type FACTOR, 1 et 2 deviennent des codes
print(sexe) # renvoie (1, 2, 1, 1, 2) et levels : 1, 2
print(is.factor(sexe)) # renvoie TRUE, confirme que sexe est bien un factor
levels(sexe) # renvoie (« 1 », « 2 ») un vecteur de chaînes de carac.
levels(sexe) <- c(« masc », « fem ») #renommer les niveaux de manière explicite
# R sait maintenant que 1 c'est « masculin », 2 c'est « féminin »
print(sexe) # renvoie (masc,fem,masc,masc,fem) et levels : masc, fem
#pour connaître le nombre de modalités
print(nlevels(sexe)) # renvoie 2
#accès aux modalités. Ex. 1ère modalité
print(levels(sexe)[1]) #renvoie « masc »
```

On peut revenir aux codes

```
x <- unclass(sexe)
print(x) # renvoie (1, 2, 1, 1, 2), vecteur d'entiers
```

tapply() permet de réaliser des calculs sur un vecteur, conditionnellement aux valeurs prises par un ou plusieurs facteurs

```
#un vecteur de 6 valeurs
v <- c(1.2,2.3,4.1,2.5,1.4,2.7)
#un factor = genre des individus
x <- factor(c(1,2,1,1,2,1))
levels(x) <- c("masculin","feminin")
#moyenne de v conditionnellement à x
print(tapply(v,x,mean))
```

```
> print(tapply(v,x,mean))
masculin  feminin
    2.625    1.850
```

Conclusion - R est magique

De la documentation à profusion (**n'achetez jamais des livres sur R**)

Site du cours

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Programmation R

http://www.duclert.org/

Quick-R

http://www.statmethods.net/

POLLS (Kdnuggets)

Data Mining / Analytics Tools Used

(R, 2nd ou 1er depuis 2010)

What languages you used for data mining / data analysis?

http://www.kdnuggets.com/polls/2013/languages-analytics-data-mining-data-science.html

(Août 2013, langage R en 1ère position)

Article New York Times (Janvier 2009)

"**Data Analysts Captivated by R's Power**" - http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=1