Introduction à R

Les bases du langage R

Ricco Rakotomalala

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

Mode de fonctionnement sous R

TRAVAILLER SOUS R

(1) R est un interpréteur, il permet de programmer avec le langage S (on peut parler aussi de langage R). L'objet de base est un vecteur de données.

C'est un « vrai » langage c.-à-d. types de données, branchements conditionnels, boucles, organisation du code en procédures et fonctions, découpage en modules.

Mode de d'exécution : transmettre à R le fichier script « .r »

→ C'est le mode que nous exploiterons dans ce didacticiel

(2) R est un logiciel de statistique et de data mining, pilotée en ligne de commande. Il est extensible (quasiment) à l'infini via le système des packages.

Les instructions servent à manipuler les objets R c.-à-d. les ensembles de données, les vecteurs, les modèles, etc. Nombreuses fonctions graphiques disponibles.

Mode de d'exécution : introduire commandes dans le terminal, manipulation interactive

Langage interprété : + portabilité application ; - lenteur (R, VBA, …)

Langage compilé : + rapidité ; - pas portable

(solution possible : write once, compile anywhere ; ex. Lazarus)

Langage pseudo-compilé : + portabilité plate-forme ; - lenteur (?)

(principe : write once, run anywhere ; ex. Java)

R est interprété, certes…

…mais les fonctions de bases (eigen, svd, etc.) et certaines fonctions stat sont compilées

Bref, il n'y a que les boucles qui posent réellement problème

Données typées. Dans R, le type de base est le vecteur. Il est possible de définir des tableaux et des matrices. Mais en interne R les *linéarise* sous forme de vecteurs, même si pour nous, c'est (souvent) transparent.

Structures avancées de données. Cela peut être des résultats de calculs i.e. les modèles ; des ensembles de vecteur (les data.frame) ; etc. → pour gérer ce type de données, R s'appuie sur le mécanisme des listes.

Séquences d'instructions, c'est la base même de la programmation, pouvoir écrire et exécuter une série de commandes sans avoir à intervenir entre les instructions.

Structures algorithmiques : les branchements conditionnels et les boucles.

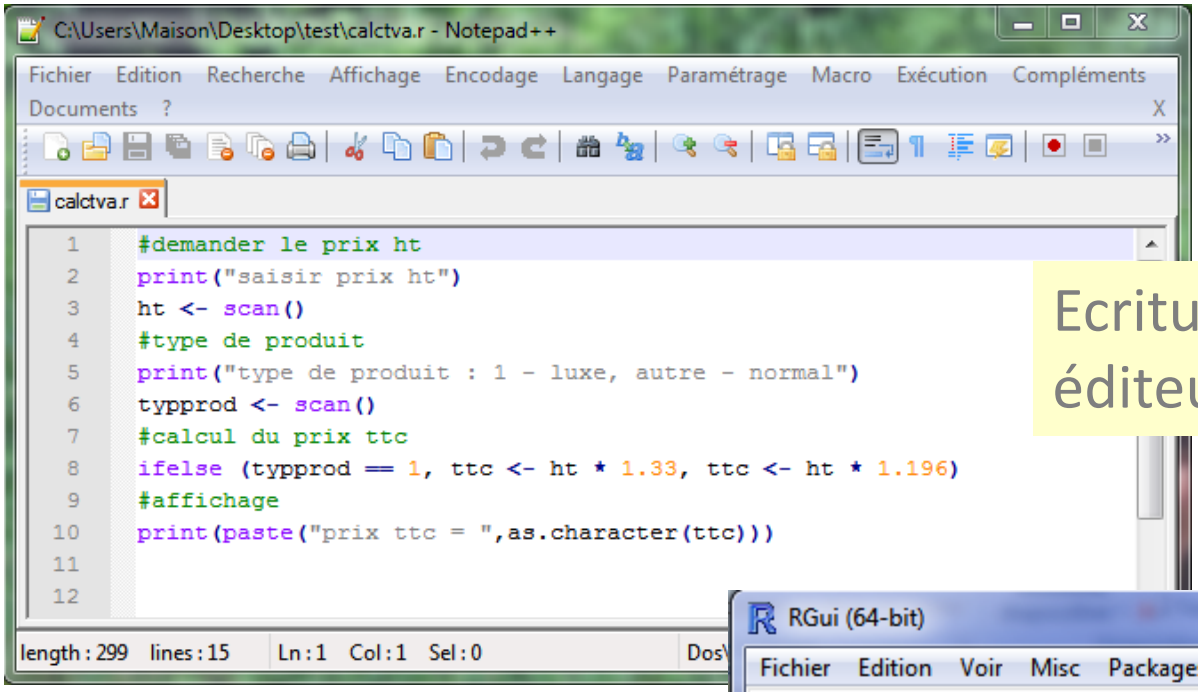
Les outils de la programmation structurée : pouvoir regrouper du code dans des procédures et des fonctions. Cela permet de ***mieux organiser*** les applications.

Organisation du code en modules. Fichiers « .r » que l'on peut appeler dans d'autres programmes.

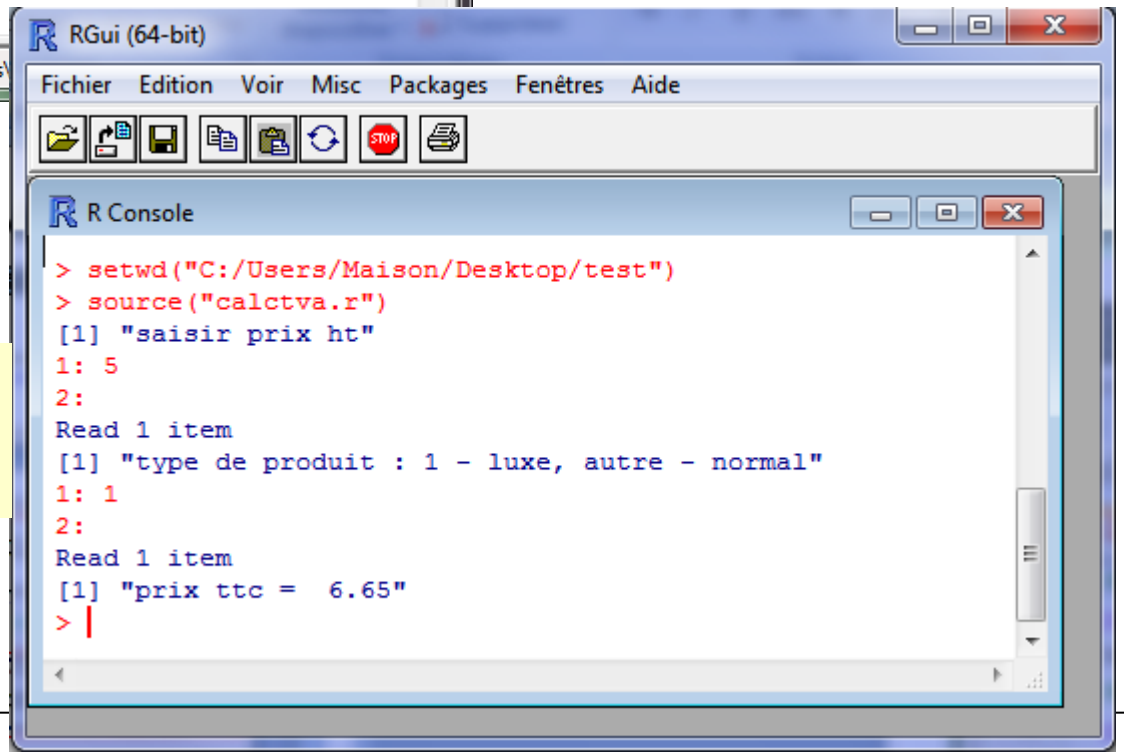
Possibilité de distribution des modules : soit directement les fichiers « .r », soit sous forme de « package ».

R est « case sensitive », il différencie les termes écrits en minuscule et majuscule. On a intérêt à écrire systématiquement le code en minuscule pour éviter les confusions.

Notre utilisation de R (dans un premier temps)



Ecriture du code dans un éditeur de code (notepad++)



Exécution du programme sous R

Premières opérations sous R

LES BASES DU LANGAGE R

Affectation

- `a <- 1.2`

a est une variable, en interne c'est un vecteur de taille 1, et il peut contenir des données numériques → le typage est donc automatique. Dans la terminologie R, **a** est un « objet ».

Lister le contenu de la mémoire

- `ls()`

Affiche tous les objets situés dans la mémoire de l'environnement R.

Supprimer un objet

- `rm(a)`

où **a** représente le nom de l'objet à supprimer.

Supprimer tous les objets en mémoire

- rm(list=ls())

On liste le nom de tous les objets en mémoire et on les efface. Attention, on supprime autant les données (vecteurs, matrices) que les modèles (ex. résultats de régression, etc.)

Accès à l'aide d'une fonction

- help(lm)

Par ex. envoie l'aide pour la fonction « lm » qui sert à produire une régression linéaire multiple.

Recherche par mot dans le fichier d'aide

- help.search(« lm »)

Recherche la chaîne de caractères « lm » dans le fichier d'aide.

Types élémentaires de R

numeric qui peut être **integer** (entier) ou **double** (double). Les opérateurs applicables sont : + , - , * , / , ^ , %% (modulo) , %/% (division entière)

ex. 5 / 2 → 2.5 (division entre valeurs réelles) ; en revanche 5 %/% 2 → 2 (division entière)

• **logical** correspond au type booléen, il prend deux valeurs possibles **TRUE** et **FALSE** (en majuscule) que l'on peut abréger avec **T** et **F**. Les opérateurs sont ! (négation), && (ET logique), || (OU logique), xor() (OU exclusif)

ex. !T → F ; xor(T,F) → T ; T && F → F ; etc.

• **character** désigner les chaînes de caractères. Une constante chaîne de caractère doit être délimitée par des guillemets

ex. a ← « toto » affecte la valeur « toto » à l'objet **a** qui devient donc une variable de type chaîne de caractères (fonctions de manipulation : paste(..), grep(..), etc.)

• Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction **class(nom_objet)**
ex. class(1.2) → renvoie la valeur **numeric**

Explicitement

La première instruction est très rarement utilisée

```
#réserver la variable
a <- logical(1)
#affectation
a <- TRUE
#affichage du type
print(class(a))
```

Implicitement

Affectation directe, R « devine » le type le plus approprié

```
#affectation
a <- TRUE
#affichage du type
print(class(a))
```

La plus couramment utilisée

```
a ← 1
b ← 5
d ← a + b
```

Autres possibilités

```
a ← 1;b ← 5 ;d ← a + b;
```

Personne n'utilise ces écritures

```
a ← 1;
b ← 5;
d ← a + b;
```

Saisie

```
a ← scan()
b ← scan()
d ← a + b
```

(1) R est interprété, on peut modifier le code source et relancer.

(2) Utiliser la fonction **scan()** qui permet d'effectuer une saisie console lors de l'exécution du programme. C'est l'approche que nous privilégierons dans un premier temps.

Affichage

```
d ← a + b
#Affichage implicite
d
#Affichage explicite
print(d)
```

N.B. L'affichage implicite ne marche pas dans les procédures et fonctions

L'affichage explicite marche toujours !

Principe

Utilisation du mot-clé « as »

> as.nom_nouveau_type(objet)

Conversion en numérique

a ← « 12 » # a est de type caractère
b ← as.numeric(a) #b est de type numeric

N.B. Si la conversion n'est pas possible ex. a ← as.numeric(« toto »), R renvoie une erreur (*plus précisément une donnée manquante, à voir plus loin…*)

Conversion en logique

a ← as.logical(« TRUE ») # a est de type logical est contient la valeur TRUE

Conversion en chaîne de caractères

a ← as.character(15) # a est de type chaîne et contient « 15 »

A voir plus tard : as.matrix(…) ; as.data.frame(…) ; etc.

Les opérateurs de comparaison servent à comparer des valeurs de même type et renvoient une valeur de type booléen.

Sous R, ces opérateurs sont <, <=, >, >=, != , ==

ex. a ← (12 == 13) # a est de type logical, il a la valeur FALSE

N.B. On utilisera principalement ces opérateurs dans les branchements conditionnels.

La valeur infinie – Inf (ou –Inf pour – l'infini)

a ← 1/0 # a prend la valeur **Inf**

t ← is.infinite(a) # t est un logical avec la valeur TRUE

b ← 1/Inf #affecte la valeur **0** à b !

La valeur indéfinie – NaN (qui veut dire « not a number »)

a ← 0/0 # envoie la valeur NaN dans a

b ← b + a # b prend aussi la valeur NaN

is.nan(b) # renvoie TRUE

Donnée manquante – NA (not available), très utile en statistique

a ← 10 + NA # affecte à **a** la valeur NA

b ← a + 5 #b prend aussi la valeur NA

is.na(b) #renvoie TRUE

Pointeur NULL (très utile pour la manipulation des listes).

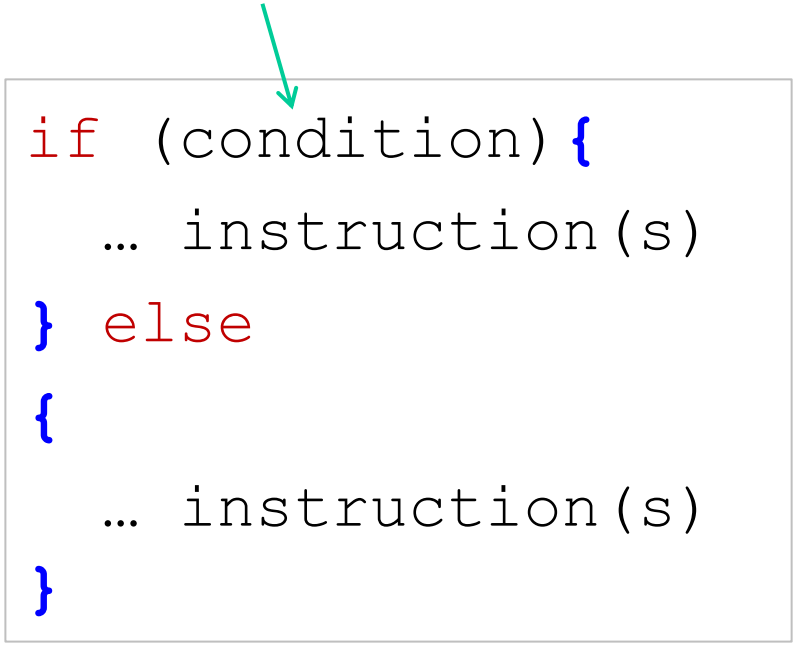
a ← NULL #a pointe sur un espace non alloué

is.null(a) #renvoie TRUE

R est un vrai langage de programmation

STRUCTURES ALGORITHMIQUES

Condition est très souvent une opération de comparaison



```
if (condition){
   … instruction(s)
} else
{
   … instruction(s)
}
```

(1) Attention aux accolades : else doit être sur la même ligne que } (je n'ai jamais su pourquoi)

(2) La partie else est facultative

Branchement conditionnel « if » (exemple)

```
#demander le prix ht
print("saisir prix ht")
ht <- scan()
#type de produit
print("type de produit : 1 - luxe, autre - normal")
typprod <- scan()
#calcul du prix ttc
if (typprod == 1)
  {
  tva <- 0.33
  } else
  {
  tva <- 0.196
  }
#calcul
ttc <- ht * (1 + tva)
#affichage
print(paste("prix ttc = ",as.character(ttc)))
```

On peut simplifier la structure de branchement, surtout quand il s'agit d'effectuer une simple affectation

Il fonctionne de 2 manières différentes :

(1) il renvoie un valeur selon que la condition est vraie ou pas

```
tva ← ifelse(typprod == 1, 0.33, 0.196)
```

(2) on procède à une opération simple suivie d'une affectation dans les deuxièmes et troisièmes paramètres de la fonction

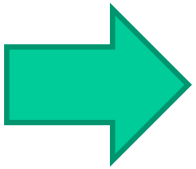
```
ifelse(typprod == 1, ttc ← ht * 1.33, ttc ← ht * 1.196)
```

On peut même mettre un bloc d'instructions délimité par { et }

```
switch(n,action si n = 1, action si n = 2,etc.)
```

Mais :

- n doit prendre les valeurs 1, 2, 3, ….
- pas de else
- action est limitée à une seule instruction, sauf à ruser en mettant un bloc entre { et }



On le voit très rarement dans les programmes !

Suite arithmétique simple (séquence de valeurs entières)

```
#génération d'une suite 1 2… 10
1:10 # on peut stocker dans une variable
#suite paramétrable
a ← 3
b ← 6
a:b #va générer 3 4 5 6
```

Séquences de valeurs réelles, avec un pas réel

```
#génération d'une suite
#1 1.5 2 2.5 … 5 (20 valeurs)
seq(from = 1, to = 5, by = 0.5)
```

Remarque : **to** et **by** peuvent ne pas correspondre, dans ce cas la séquence s'arrête à la dernière valeur précédent **to**

ex. seq(from=1, to=2, step=0.3) → 1 1.3 1.6 1.9

Répétition d'une séquence de valeurs

```
rep(séquence,times = nombre de fois,each = répétition de chaque élément)
```

Exemples

```
(1)rep(1 :4,times=2) → 1 2 3 4 1 2 3 4
(2)rep(1 :4,each=2) → 1 1 2 2 3 3 4 4
(3)rep(1 :4,times=2,each=2) → 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Séquence peut être stockée dans un vecteur
Le pas peut ne pas être régulier
La séquence n'est même pas forcément monotone !
Bref, l'outil est très (trop) souple → d'où sa lenteur

```
for (indice in séquence){
  … instruction(s)
}
```

Remarque : On peur « casser » la boucle avec break

```
#A faire : somme S = SUM_i i^2
#saisie de n
n <- scan()
#initialisation de la somme
s <- 0.0
#boucle for
for (i in 1:n){
  s <- s + i^2
  }
#affichage des résultats
print("la somme = ")
print(s)
```

Opération de comparaison
Attention à la boucle infinie !

```
while (condition){
   … instruction(s)
}
```

```
#A faire : somme S = SUM_i i^2
#saisie de n
n <- scan()
#initialisation de la somme et de l'indice
s <- 0.0
i <- 1
#boucle « tant que »
while (i < n){
  s <- s + i^2
  i <- i + 1
  }
#affichage des résultats
print("la somme = ")
print(s)
```

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