Introduction à R

RStudio

Pascal Bessonneau

06/2015

A not so short history of R

La pratique

Un peu de R...

Qu'est ce que le langage R ?

L'environnement de travail

Opérateurs et variables

A not so short history of R

version	Date	Description
0.49	1997	La création du Comprehensive R Archive project. Les plus vieilles sources disponibles.
0.60	1997	R devient officiellement une partie du projet GNU.
0.65.1	1999	Création des fonctions de chargement et d'installation des paquets
1.0	2000	Création des fonctions de chargement et d'installation des paquets
3.0	2013	Support pour les entiers 2^31 et plus sur les systèmes 64 bits.

GNU et adopté par la communauté

Le langage R, initialement, suit en fait les spécifications d'un langage d'un logiciel propriétaire $S+$. Le langage S décrit par Chambers & al. est le langage de ce logicel.
En référence, vous pourrez entendre parler de *White book*, *Yellow book*, ... Ce sont des ouvrages de référence : ils donnent les spécificités de S et plus tard de R.

GNU et adopté par la communauté

Devenu projet GNU, son portage sur Mac OS X en 2001 et la flexibilité du langage S permet à R de devenir une référence pour les statisticiens assez rapidement.
Mais deux élements sont déterminants dans son avenir :

- c'est un langage lent
- les paquets

Les tâlons d'Achille

R a longtemps, jusqu'en 2013, été limité par le volume des données qu'on pouvait stocker en mémoire.
En 2013, ce frein a été levé avec un portage en 64 bits. Toutefois R reste limité par le fait qu'il travaille d'abord en mémoire vive et n'a pas été conçu comme d'autres logiciels pour ne stocker que le minimum en mémoire.
Ainsi on peut être limité par la RAM quand on travaille sur R.
C'est dans ces limites que s'insèrent les versions propriétaires de R qui offre des fonctionnalités spécifiques notamment pour les gros volumes de donnés et la parallélisation.

Les tâlons d'Achille

Toutefois contrairement à d'autres logiciels, les versions propriétaires restent minoritaires car leurs spécificités ne sont pas si intéressantes.
Mais c'est difficile à quantifier.
La version open source et libre R fait déjà tellement...

Les interfaces utilisateurs avec R

Les façons de faire du R :

- en ligne de commande
- éditeur + copier/coller
- dans Emacs
- RStudio
- RStudio Server (dans un navigateur)

Les paquets

Aujourd'hui nous sommes passés de 12 paquets à l'origine à près de 8300...
Ces paquets sont souvent écrits par les auteurs des nouvelles méthodes statistiques ce qui est garant de leur qualité. Ils sont souvent modifiés par la communauté pour plus de fonctionnalités et performances.
Beaucoup d'entre eux sont hébergés sur Github et des outils permettent d'installer les paquets directement depuis Github pour les versions instables ou stables.
C'est devenu une forêt dans laquelle il est facile de se perdre. Tout est déjà écrit en R, le plus dur c'est de le trouver ;)

Les vues

Les paquets les plus importants et les plus intéressants sont répertoriés dans les vues, des ensembles de paquets maintenus par des spécialistes du domaine. Il y a des vues pour :

- le *Clustering*
- le *Machine Learning*
- l'expérimentation
- ...

Un paquet *ctv* permet d'installer et de mettre à jour les paquets d'une vue en particulier. Pratique.

BioconductoR

En parallèle de du R, on trouve BioconductoR. C'est R toujours mais avec des dépôts de paquets séparés.
Dedans on trouve les outils de tous les jours pour les biologistes moléculaires, l'analyse phylogénétique, ...
Une partie importante des recrutement pour des gens compétents en R est dans la biologie et notamment la biologie moderne : séquençage, analyse des séquences, protéomique ... Cette année 5 postes ont été proposés à l'institut Pasteur.

RStudio

RStudio est un logiciel libre permettant de travailler plus facilement avec R.
Au cours de ces dernières années, il s'est imposé comme une GUI quasi incontournable à R.

Principe

C'est un langage de programmation interprété.
Le stockage se fait en RAM.

```
> a <- 3 + 4
> a

## [1] 7
```

Principe

Vous remarquerez qu'un numéro apparait à gauche... C'est le numéro de l' élement dans le vecteur.
C'est parce que dans R pratiquement tout est vecteur :

```
> a <- rnorm(10)
> a

##  [1] -1.3725905  0.8814864  0.3821462
##  [4]  3.7392059 -0.8397317 -0.9899751
##  [7] -0.8071744  0.2860360  0.4523745
## [10]  0.7768804

> class(a)

## [1] "numeric"
```

Principe

Il existe plusieurs types de base pour les variables :

- les *numeric* pour stocker les nombres à virgules (ou flottants)
- les entiers, *integer*
- les booléens, *logical*
- les variables caractères, *character*
- ...

Principe

Il existe des structures de données plus complexes. Mais ce sont ces types de base qui n'existent que sous la forme de vecteurs. Toute variable créeest au minimum un vecteur de longueur 1.

Absence de typage fixe

Une variable est versatile, elle n'est pas définie comme dans d'autres langages.

```
> a <- 5
> a

## [1] 5

> a <- "Hello World !"
> a

## [1] "Hello World !"
```

R de ce point de vue est à ranger du coté de javascript, python (hors objet), ...

Principe

Pire... R assure des interconversions automatiques...

```
> a <- 5 + TRUE
> a

## [1] 6
```

Principe

La subtilité, pourquoi la commande *a* ?

```
> a

## [1] 6

> print(a)

## [1] 6
```

Principe

Un vecteur se définit comme suit :

```
> c(1,2,3)

## [1] 1 2 3

> c("A","B","C","D")

## [1] "A" "B" "C" "D"

> c(T,F,T,T,F,T)

## [1]  TRUE FALSE  TRUE  TRUE FALSE  TRUE

> c(T,F,T,T,F,T) + 0

## [1] 1 0 1 1 0 1
```

Principe

```
> c(1,2,3)

## [1] 1 2 3

> c("A","B","C","D")

## [1] "A" "B" "C" "D"

> c(T,F,T,T,F,T)

## [1]  TRUE FALSE  TRUE  TRUE FALSE  TRUE
```

Les opérateurs

```
> 2 + 3 * 2 / 4

## [1] 3.5

> T | T

## [1] TRUE
```

Les opérateurs

```
> c(1,2,3) * 2

## [1] 2 4 6

> T & c(T,F,T)

## [1]  TRUE FALSE  TRUE
```

Les fonctions déjà développées

```
> sqrt(4)

## [1] 2

> min(c(2,3,4,20))

## [1] 2
```

Les fonctions

```
> rep(2,5)

## [1] 2 2 2 2 2

> paste("A","B")

## [1] "A B"

> 1:10

##  [1]  1  2  3  4  5  6  7  8  9 10
```

Le type de langage

R est un langage interprété. C'est-à-dire qu'il va lire le code instructions par instructions et exécuté chacune d'elles à chaque fin de ligne.
La fin de ligne peut être au clavier la touche *entrée* ou la fin de ligne dans un fichier texte.
Il existe la possibilité de mettre plusieurs commandes sur la ligne avec le caractère " ;" . Mais cela est déconseillé.

La casse

Le langage est casse-dépendant ce qui signifie que :

```
> A=5
> a=4
> A

## [1] 5

> a

## [1] 4
```

La casse

Cela est vrai pour les commandes, le nom des variables, ... mais également dans les comparaisons de chaîne de caractère.
Si on teste l'égalité de deux chaînes on obtient :

```
> "A"=="a"

## [1] FALSE

> "A"=="A"

## [1] TRUE
```

La gestion de la mémoire (1)

Directement en tapant dans la console ou un programme généralement, les variables créées atterissent dans un "environnement" de base (*GlobalEnv*).
Sauf exception, les variables vont automatiquement être ajoutées à cet environnement.
Cet environnement comme tous les autres sont en mémoire vive contrairement à d'autres logiciels de statistiques.
De plus il n'y a pas de ramasse-miettes. Les variables ne sont supprimées que sur la demande de l'utilisateur/programmeur.

La gestion de la mémoire (1)

La place en mémoire vive est à la fois une force de R et une faiblesse.
Force car la vitesse d'exéuction en mémoire vive permet généralement des performances meilleures que dans d'autres logiciels.
Faiblesse car cela est problématique pour les gros volumes de données.

La gestion de la mémoire (2)

Il existe d'autres environnements que l'environnement de base. Par exemple quand on charge un paquet, les variables de ce paquet n'entre pas en conflit avec ses propres variables car les variables (et certaines fonctions) du paquet n'atterisse pas dans l'environnement global.
La création d'environnement est soit automatisé soit volontaire.
Automatisé par exemple lorsqu'on crée une fonction, les variables de la fonction ne sont pas accessibles de l'extérieur.
Volontaire par exemple en appelant la fonction *new.env*.

Les opérateurs

Les opérateurs vont permettre l'assignation mais également les comparaisons et les opérations mathématiques.

```
> a <- 3 # Assignation
> a = 3 # Assignation (déconseillé)
> a == 3 # Comparaison

## [1] TRUE

> a != 3 # Différent de

## [1] FALSE

> a + 4  # somme

## [1] 7

> a | TRUE # Ou

## [1] TRUE
```

Les opérateurs

Comme tout langage les règles de précédence entre les arguments sont parfois complexes. Les parenthèses permettent de forcer la précédence.

```
> a <- 3+1
> (a <- 3) + 1

## [1] 4
```

Les variables

Les variables permettent de stocker les informations. Leur nom ne doit pas commencer par un nombre et ne pas contenir de caractères spéciaux tels que les blancs.
En vrai, c'est un peu plus compliqué. Il est possible d'avoir un nom de variable non standard en utilisant guillemets et/ou des guillemets inversés.

Les variables

Vous pouvez utiliser des mots réservés par R. Cela ne soulève pas d'erreurs.
Par contre le mot réservé est "masqué". Il est possible de le faire mais cela est déconseillé. Notamment car la procédure de masquage/démasquage peut amener à des bugs subtils.