L3 Pro "Biotechnologies végétales et création variétale"
TP 0 : introduction à R

D. Chessel, A.B. Dufour, S. Penel, J. Lobry, N. Rochette & M. Bailly-Bechet

Automne 2014

1 Avant de commencer

Vous êtes dans une salle de TP de l'université : normalement R est déjà installé. Les informations suivantes pourront vous servir pour faciliter la sauvegarde des données et l'interface entre R et les fichiers stockés sur votre ordinateur.

Si vous désirez installer R sur votre machine personnelle, vous pouvez le télécharger gratuitement sur différents miroirs :

— `http://lib.stat.cmu.edu/R/CRAN/`

— `http://stat.ethz.ch/CRAN/`

On peut aussi utiliser le miroir de l'université : `http://cran.univ-lyon1.fr/`

1.1 Lancer le programme

Pour lancer R, allez dans le lanceur de programmes et trouvez-le logo R – si plusieurs versions sont disponibles utilisez la dernière. Dans la fenêtre principale du logiciel, vous remarquez en haut les barres de menus habituelles, et au centre, une fenêtre "R console". Nous allons principalement travailler dans cette fenêtre, aussi agrandissez-la. Le message affiché dans cette fenêtre contient des informations sur la version du logiciel, etc... Le plus important est le symbole > que vous voyez à la dernière ligne. Ce "prompt" est l'endroit où vous allez taper les commandes lors de vos analyses. Si à un moment ou un autre ce symbole est remplacé par autre chose (comme un "+"), quelque chose d'inattendu est en train de se produire !

Pour vérifier que tout fonctionne, nous allons poser une question simple à R. Tapez dans l'invite de commande :

```
log(3)
```

Vous devriez obtenir comme réponse :

```
[1] 1.098612
```

Dans la suite les commandes à taper seront écrites en bleu, et les résultats en rouge. Familiarisez-vous avec le logiciel en commencant par recopier "bêtement" les commandes du TP, et rassurez-vous : on vous demandera rapidement plus d'imagination !

2 Créer des objets

Les données et les fonctions sont stockées dans des *objets*. Il existe de nombreux types d'objets, ou *classes* ; on ne se servira en pratique que de quelques-uns d'entre eux.

Lorsque vous commencez votre session R, certains objets existent déjà : c'est le cas de la fonction `log()` que vous avez utilisé, ainsi que de toutes les fonctions que nous utiliserons par la suite. On reconnaît une *fonction* au fait qu'elle est toujours suivie de parenthèses, dans lesquelles on peut passer des *arguments*. D'autres objets existent aussi, comme par exemple certains constantes : tapez `pi` dans l'invite de commande.

```
pi
```

```
[1] 3.141593
```

La première chose à faire pour créer un objet est de lui trouver un nom, qui ne doit contenir que des caractères alphanumériques, le souligné et le point. Une fois que vous avez décidé du nom de l'objet, créez-le en lui *affectant* la valeur désirée à l'aide de l'opérateur d'affectation `<-`, comme dans l'exemple :

```
x <- 7
```

Si rien ne se passe, c'est que tout s'est bien passé. Si R affiche un message, lisez-le, c'est probablement que quelque chose de louche a eu lieu – le plus souvent une simple faute de frappe qui rend votre commande incompréhensible pour le logiciel. Pour vérifier ce que contient un objet, il suffit d'entrer son nom :

```
x
```

```
[1] 7
```

Le sens du `[1]` sera donné par la suite.

Cela fonctionne de la même façon pour tous les objets. Ci-dessous, on crée trois objets différents, un booléen (valeur logique), une liste et une chaîne de caractères :

```
b<-TRUE
truc_bizarre<-list()
mot<-"coucou"
```

Vérifiez que ces objets sont bien définis en demandant à R ce qu'ils contiennent. Si vous ne vous souvenez plus des noms que vous avez employé, la commande `ls()` vous renvoie tous les noms des objets que vous avez créé.

3 Consulter la documentation

Consulter la fiche de documentation est une opération fondamentale !

Pour obtenir de l'aide sur une fonction inconnue, vous pouvez utiliser de manière équivalente deux commandes, `?` et `help`

```
?log
help(log)
```

Pour chercher une fonction dont vous ne connaissez pas le nom, vous pouvez utiliser `help.search()`. Par exemple, pour chercher la fonction cosinus, si on ne devine pas son nom :

```
help.search("cos")
```

La liste que vous obtenez est celle de toutes les fonctions qui contiennent le mot "cos", et il y en a énormément. Pour s'y retrouver, au début, il faut chercher les fonctions appartenant à la bibliothèque `base`

```
help.search("cos",package="base")
```

Ici, on repère `base::Trig Trigonometric functions`. Si on veut plus de détails :

```
?Trig
```

Attention, vous avez peut-être remarqué que R est un logiciel international, et donc que le vocabulaire employé est majoritairement en anglais. Au vu de ce que nous allons faire avec, ca ne devrait pas poser de gros problèmes, les statisticiens ne sont pas des linguistes !

Chaque fiche d'aide est structurée de la même façon. En plus du nom de la fonction et du module dont elle dépend (en haut à gauche), on a les sections suivantes :

Description : une courte description du contenu de la fiche.

Usage : la liste des arguments de la ou des fonctions présentées, et leurs valeurs par défaut.

Arguments : la description détaillée des arguments.

Details

See Also : une liste de fiches à consulter sur des sujets voisins.

Examples

4 Utiliser l'historique

R se souvient des commandes que vous avez tapées. Pour retrouver une commande que vous avez tapée, utilisez les flèches du haut et du bas ; cela peut être très utile quand on fait une petite erreur dans une grosse commande, pour éviter de tout retaper.

5 Les vecteurs

5.1 Calcul sur les vecteurs

Les vecteurs sont des séries d'éléments *du même type*. Dans R même une simple valeur numérique est un vecteur :

```
pi
```
```
[1] 3.141593
```
```
is.vector(pi)
```
```
[1] TRUE
```

Prenons la définition de la variance observée d'un échantillon de taille n d'une variable, notée s^2 :

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Dans un langage de programmation classique, ou sur une calculette, il faudrait calculer la moyenne, puis toutes les différences, les mettre au carré et les additionner. Dans R, on peut aller *un peu* plus vite... voire beaucoup.

On va chercher à calculer la variance du vecteur des nombres de 1 à 10. Une façon de créer ce vecteur (vous aurez plus de détails là-dessus paragraphe suivant) est :

```
x<-1:10
x
```
```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Combien y a-t-il d'éléments dans ce vecteur ?

```
length(x)
```
```
[1] 10
```

Quelle est la somme des éléments de ce vecteur ?

```
sum(x)
```
```
[1] 55
```

Donc la moyenne de cet échantillon (ce vecteur), que l'on pourait mathématiquement écrire $\frac{1}{n}\sum_{i=1}^{n} x_i$, est :

```
x_barre<-sum(x)/length(x)
x_barre
```

```
[1] 5.5
```

On arrive à la partie compliquée : il faut enlever à chaque élément de `x` la moyenne observée. Mais R sait faire les 10 opérations en une seule fois :

```
x-x_barre
```
```
[1] -4.5 -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5  3.5  4.5
```

Puis mettre chaque élément au carré :

```
(x-x_barre)^2
```
```
[1] 20.25 12.25  6.25  2.25  0.25  0.25  2.25  6.25 12.25 20.25
```

Puis faire la somme de tout cela, et diviser par le nombre d'éléments :

```
sum((x-x_barre)^2)/length(x)
```
```
[1] 8.25
```

Bon, c'est quand même long pour calculer une variance. En pratique, une fonction sait faire ca directement :

```
var(x)
```
```
[1] 9.166667
```

Le résultat est différent. Et pourtant le calcul précédent était juste. En fait, cette fonction `var()` calcule *l'estimateur de la variance de la population*, $\hat{\sigma}^2$, et pas la variance observée s^2. On rappelle que $s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ et $\hat{\sigma}^2 = \frac{n}{n-1}s^2$. On devrait donc pouvoir retrouver la valeur de s^2 à partir du résultat de `var()`. Vérifions :

```
10/9*8.25
```
```
[1] 9.166667
```

Exercice

Dans tous les exercices, on vous indique la réponse que vous devriez obtenir si votre manipulation a bien fonctionné. Procédez par essai-erreur, n'hésitez jamais à aller consulter les pages d'aide, et ne bloquez pas sur chaque question trop longtemps !

1. Retrouvez la variance observée du vecteur composé des 1000 premiers entiers :

   ```
   [1] 83333.25
   ```

2. Donner le carré des 10 premiers entiers moins un :

   ```
   [1]  0  3  8 15 24 35 48 63 80 99
   ```

3. Donner la racine carré (`sqrt()`) des 6 premiers entiers :

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
```

4. Donner le sinus (`sin()`) des entiers de 5 à 10 :

```
[1] -0.9589243 -0.2794155  0.6569866  0.9893582  0.4121185 -0.5440211
```

5. Donner 2 à la puissance des 10 premiers entiers :

```
 [1]    2    4    8   16   32   64  128  256  512 1024
```

5.2 Séries de valeurs numériques

5.2.1 Séries d'entiers : ":"

L'opérateur deux points `:` permet de générer des séries d'entiers successifs :

```
x <-1:12
x
[1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Exercice. Générer la série suivante :

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

5.2.2 Séries de valeurs numériques : `seq()`

La fonction `seq()` permet de générer une série de nombres équidistants :

```
seq(from = 1, to = 5)
[1] 1 2 3 4 5
seq(from = 1, to = 2, by = 0.1)
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
seq(from = 10, to = 50, length = 10)
[1] 10.00000 14.44444 18.88889 23.33333 27.77778 32.22222 36.66667 41.11111
[9] 45.55556 50.00000
```

Dans le dernier exemple, on voit un `[9]` au début de la deuxième ligne. Il indique simplement que le premier élément de cette ligne est le $9^{\text{ème}}$ du vecteur créé... et donc tous les `[1]` indiquent juste que la première ligne d'un vercteur commence par le premier élément.
Exercice. Générer les séries suivantes :

```
[1]  0 -1 -2 -3 -4 -5
 [1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10
[1] 1.000000 1.666667 2.333333 3.000000
```

5.3 Construction et combinaison de vecteurs : `c()`

La fonction `c()` permet de créer un vecteur quelconque, ou de concaténer des vecteurs entre eux :

```
x <- c(1.5,7.2,pi)
x
```

```
[1] 1.500000 7.200000 3.141593
```

```
y <- c(1:3,10:12)
y
```

```
[1]  1  2  3 10 11 12
```

```
z <- c("coucou","machin","dinosaure")
z
```

```
[1] "coucou"    "machin"    "dinosaure"
```

Exercice. Construire les vecteurs suivants :

```
[1] 11.1  2.7  3.3
```

```
[1] -5.00 -4.00 -3.00 -2.00 -1.00  0.00  1.00  1.10 88.77
```

```
[1] "Alpha"   "Tango"   "Charlie"
```

5.3.1 Statistiques élémentaires

Considérons les notes de 14 étudiants d'un groupe de TP et calculons les paramètres statistiques élémentaires. Essayez de retrouver ce que calcule chaque fonction à partir de son nom, son résultats, ou la documentation associée (avec `help` ou `?`).

```
notes <- c(15,8,14,12,14,10,18,15,9,5,12,13,12,16)
sort(notes)
length(notes)
min(notes)
max(notes)
range(notes)
median(notes)
quantile(notes)
mean(notes)
var(notes)
sd(notes)
unique(notes)
```

5.4 Comparaisons et calculs logiques

Ce sont les calculs qui retournent une valeur logique vrai-faux, notée par TRUE ou FALSE. Les opérateurs de comparaison sont ==, >, >=, <, <=. Faisons quelques essais :

```
u <- 1:5
u
```

```
[1] 1 2 3 4 5
```

```
v <- 5:1
v
```

```
[1] 5 4 3 2 1
```

```
u == v
```

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

```
u <= v
```

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

```
u < v
```

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

Que se passe-t-il si on oublie un "=" dans "==" ?

```
u = v
```

Que vaut u ? v ? Comprenez-vous pourquoi ?

L'opérateur "==" peut aussi servir à comparer des chaines de caractères :

```
u <- c("Alpha", "Bravo","Charlie","toto")
u
```

```
[1] "Alpha"   "Bravo"   "Charlie" "toto"
```

```
v <- c("Alpha", "Beta", "Gamma", "toto")
v
```

```
[1] "Alpha" "Beta"  "Gamma" "toto"
```

```
u == v
```

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

5.5 Éléments des vecteurs

5.5.1 Indexation par des entiers positifs

Soit `x` le vecteur suivant :

```
x <- c(145,-1,28.88,0.02,34.5)
x
```

```
[1] 145.00  -1.00  28.88   0.02  34.50
```

Pour récuperer le quatrième élément :

```
x[4]
```

```
[1] 0.02
```

Pour récupérer éléments du deuxième au troisième :

```
x[2:3]
```

```
[1] -1.00 28.88
```

On peut reprendre plusieurs fois le même :

```
x[c(2,2,3)]
```

```
[1] -1.00 -1.00 28.88
```

Les éléments hors bornes ne sont pas disponibles (`NA`, not available, donnée manquante) :

```
x[100]
```

```
[1] NA
```

5.5.2 Indexation par des entiers négatifs

Pour tout récupérer sauf le quatrième élément :

```
x[-4]
```

```
[1] 145.00  -1.00  28.88  34.50
```

Exercice 1. Donner tous les éléments sauf le premier.

```
[1] -1.00 28.88  0.02 34.50
```

Exercice 2. A l'aide de la fonction `length()`, donner tous les éléments sauf le dernier :

```
[1] 145.00  -1.00  28.88   0.02
```

On ne peut pas mélanger les indexations par des entiers positifs et négatifs simultanément. Il faut procéder en deux temps.

5.5.3 Indexation par un vecteur logique

Ici `T` et `F` sont des abréviations de `TRUE` et `FALSE`.

```
 x[c(T,F,F,T,T)]
```

```
[1] 145.00   0.02  34.50
```

Si le vecteur logique n'est pas assez long il sera recyclé autant de fois que nécessaire.

Le vecteur logique d'indexation peut être issu d'un calcul logique, c'est l'utilisation la plus courante :

```
 x > 10
```

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

```
 x[x > 10]
```

```
[1] 145.00  28.88  34.50
```

Si on veut respecter plusieurs conditions simultanément, on peut utiliser le "et" logique, `&`, ou le "ou" |.

```
 x > 0 & x < 1
```

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

```
 x[x > 0 & x < 1]
```

```
[1] 0.02
```

Exercice. Donner les éléments compris entre 10 et 50 :

```
[1] 28.88 34.50
```

5.5.4 Indexation par des noms

Ceci ne fonctionne que si les éléments du vecteur ont un nom. Donnons comme nom les 5 premières lettres de l'alphabet aux éléments du vecteur `x` :

```
 names(x) <- letters[1:5]
 x
```

```
     a      b      c      d      e
145.00  -1.00  28.88   0.02  34.50
```

Le vecteur `letters`, comme `pi`, est pré-défini dans R. Quelles sont les valeurs des éléments `a` et `e` (attention au guillemets) ?

```
 x[c("a","e")]
```

```
    a     e
145.0  34.5
```

6 Les listes

6.1 Création d'une liste : `list()`

Les listes sont une structure de données très flexible et très utilisée dans R. Un élément d'une liste est un objet R *quelconque*, y compris une autre liste (contrairement aux vecteurs qui n'acceptent que des objets de même type). La fonction `list()` permet de créer des listes :

```
maliste <- list(a = pi, b = "une chaine", c = c(T,F,NA))
maliste
```

```
$a
[1] 3.141593

$b
[1] "une chaine"

$c
[1]  TRUE FALSE    NA
```

```
names(maliste)
```

```
[1] "a" "b" "c"
```

6.2 Extraction d'un élément d'une liste : `$` et `[[`

Un élément d'une liste est en général extrait par son nom (avec l'opérateur `$` ou `[[`) ; on peut aussi utiliser sa position dans la liste, mais uniquement avec l'opérateur `[[` :

```
maliste$c
```

```
[1]  TRUE FALSE    NA
```

```
maliste[[3]]
```

```
[1]  TRUE FALSE    NA
```

```
maliste[["c"]]
```

```
[1]  TRUE FALSE    NA
```

Attention à ne pas confondre `[` et `[[`, qui ne donnent pas le même résultat, même si les deux peuvent se ressembler à première vue !

Exercice.

1. Donner l'élément `b` de la liste :

   ```
   [1] "une chaine"
   ```

2. Donner le premier élément de la liste :

   ```
   [1] 3.141593
   ```

6.3 Ajout d'un élément dans une liste : `$`

```
maliste$d <-1:10
maliste
```

```
$a
[1] 3.141593

$b
[1] "une chaine"

$c
[1]  TRUE FALSE    NA

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

7 Les tableaux : `data frames`

Les data frames sont des listes un peu particulières, dont tous les éléments ont la même longueur : on peut les voir comme des tableaux, où chaque ligne est un individu et chaque colonne une variable.

On peut lire un fichier de données au format texte, par exemple sur un serveur Internet, avec R grace à la commande `read.table`. Observons d'abord ce qui se passe quand on lit les 5 premières lignes du tableau (l'usage de l'opérateur `[]` pour les tableaux est décrit un peu plus loin).

```
t3var<-read.table("http://pbil.univ-lyon1.fr/R/donnees/t3var.txt")
t3var[1:5,]
```

```
   V1  V2  V3
1 sexe poi tai
2    h  60 170
3    f  57 169
4    f  51 172
5    f  55 174
```

Regardez la première ligne : il y a eu confusion entre les valeurs du premier individu et les en-têtes de colonnes ! Regardez dans la documentation de `read.table` à quoi sert l'argument `header`. Ajustez votre commande pour que les en-têtes ne soient plus confondus avec le premier individu.
Dans le cas précédent, le fichier à charger dans  était sauvegardé sur un site internet. Il est également possible de charger des données sauvegardées sur un répertoire local, il suffit pour cela de donner à la fonction `read.table()` comme argument le chemin complet permettant d'accéder au fichier. Cela sera vu ultérieurement.

7.1 Indexation des data frames : `$` et `tab[i,j]`

Les data frames étant des listes, les variables (en colonne) sont directement accessibles par leur nom :

```
t3var$tai
```

```
 [1] 170 169 172 174 168 161 162 189 160 175 165 164 175 184 178 158 164 179 182
[20] 174 158 163 172 185 170 178 180 189 172 174 200 178 178 168 170 160 163 168
[39] 172 175 180 162 177 169 173 182 183 184 181 180 178 178 168 161 171 180 174
[58] 175 182 181 188 182 189 178 150 186
```

Exercice.

1. Donner la moyenne des tailles :

   ```
   [1] 174.0606
   ```

2. Donner la médiane des poids :

   ```
   [1] 65.5
   ```

3. Donner la variance des poids :

   ```
   [1] 123.5767
   ```

4. Donner l'écart-type des poids :

```
[1] 11.11651
```

On peut également utiliser la notation de type `tab[i,j]` où `i` représente l'index des lignes et `j` celui des colonnes. Par exemple, pour avoir les 5 premiers individus :

```
t3var[1:5,]
```

```
  sexe poi tai
1    h  60 170
2    f  57 169
3    f  51 172
4    f  55 174
5    f  50 168
```

Pour avoir les première et troisième colonnes des 5 premiers individus :

```
t3var[1:5, c(1,3)]
```

```
  sexe tai
1    h 170
2    f 169
3    f 172
4    f 174
5    f 168
```

Tous les types d'indexation vu pour les vecteurs (entiers positifs, négatifs, logiques, noms) sont utilisables.

Exercice.

1. Donner toutes les variables des individus 1, 5 et 55 :

```
   sexe poi tai
1     h  60 170
5     f  50 168
55    h  73 171
```

2. Pour les 10 premiers individus, donner le poids et la taille :

```
   poi tai
1   60 170
2   57 169
3   51 172
4   55 174
5   50 168
6   50 161
7   48 162
```

```
8   72 189
9   52 160
10  64 175
```

3. Donner tous les individus de sexe féminin :

```
   sexe poi tai
2     f  57 169
3     f  51 172
4     f  55 174
5     f  50 168
6     f  50 161
7     f  48 162
9     f  52 160
11    f  53 165
16    f  51 158
17    f  53 164
21    f  49 158
22    f  50 163
25    f  53 170
29    f  70 172
30    f  62 174
34    f  51 168
35    f  52 170
36    f  57 160
37    f  53 163
38    f  55 168
39    f  66 172
42    f  50 162
43    f  53 177
54    f  47 161
65    f  47 150
```

4. Donner tous les individus qui font plus de 175 cm :

```
   sexe poi tai
8     h  72 189
14    h  78 184
15    h  68 178
18    h  79 179
19    h  74 182
24    h  80 185
26    h  73 178
```

```
27    h  70 180
28    h  72 189
31    h  77 200
32    h  70 178
33    h  76 178
41    h  75 180
43    f  53 177
46    h  72 182
47    h  75 183
48    h  73 184
49    h  71 181
50    h  66 180
51    h  71 178
52    h  79 178
56    h  72 180
59    h  85 182
60    h  73 181
61    h  82 188
62    h  86 182
63    h  85 189
64    h  65 178
66    h  74 186
```

5. Donner tous les individus de sexe féminin qui font plus de 175 cm :

```
   sexe poi tai
43    f  53 177
```

6. Donner le poids et la taille tous les individus de sexe féminin qui font plus de 175 cm :

```
   poi tai
43  53 177
```