Introduction à R

C. Dillmann, H. Devillers

23/03/2009

Table des matières

1 Introduction

Le langage R est un langage interprété qui se présente sous la forme d'un logiciel libre (http://cran.r-project.org) et qui est devenu aujourd'hui un standard dans tous les domaines scientifiques. Les principaux sites web où vous pouvez trouver des aides sur R sont les suivants :

Statistics with R : http://zoonek2.free.fr/UNIX/48R/all.html
CRAN : http://cran.r-project.org/
Biostatistique Lyon 1 : http://pbil.univ-lyon1.fr/R/enseignement.html
Jussieu EBGM : http://www.leslieregad.com/fichiercours/IntroductionR.pdf

Ce logiciel permet de gérer des tableaux de données et de faire des analyses statistiques, des représentations graphiques, de l'analyse d'images et du calcul numérique. Ce tutoriel a pour buts d'introduire les bases du langage R, quelques rudiments de programmation, et quelques fonctions utiles pour l'analyse des données. Il vous suffit de lire le poly et de faire les exercices sous R.

Dans ce manuel, le `texte tapé par l'utilisateur` ainsi que les `réponses` du logiciel sont en police courrier.

1.1 Chargement du logiciel

Dans le menu *Demarrer* de *Windows*, choisir le programme R. Au chargement de R, une fenêtre de contrôle s'ouvre (*R console*). Dans cette fenêtre de contrôle, on peut rentrer des instructions après le prompt `>`. Le fonctionnement de R est simple. On tape des commandes dans la fenêtre de contrôle, et le logiciel exécute ces commandes et affiche le résultat. D'autres fenêtres peuvent être ouvertes à partir de la fenêtre de contrôle.

1.2 Editeur

L'éditeur de R s'ouvre en choisissant l'onglet *nouveau script* dans le menu Fichier. Lorsque l'on doit manipuler un grand nombre d'instructions, il est plus facile de les taper d'abord dans l'éditeur, puis de les copier et de les coller dans la fenêtre de contrôle. Cela permet également de garder une trace de son travail. Une ligne tapée dans l'éditeur peut être recopiée dans la fenêtre de contrôle en cliquant sur le bouton " *Run line* " ou à l'aide du raccourci " *Ctrl-R* ". Pour exécuter plusieurs lignes à la fois, il suffit de les sélectionner. Dans l'éditeur, tapez `"bonjour"` (sans oublier les guillemets) et exécutez la commande. Regardez ce qui se passe dans la fenêtre de contrôle :

```
> "bonjour"
[1] "bonjour"
```

Vous avez créé un objet qui contient un élément qui a la valeur `"bonjour"`.
Sauvegardez votre script !

1.3 Aide

Il existe plusieurs façons d'obtenir de l'aide sous R. Toutes les fonctions de base sous R possèdent une documentation spécifique qui peut être appelée à l'aide de la fonction `help`. Ainsi pour obtenir la documentation de la fonction `plot` il suffit de faire :

```
> help(plot)
```

La commande `?plot` (le `?` devant le nom de la fonction) donne le même résultat. Enfin, la fonction `help.start()` permet d'ouvrir la version online (HTML) de l'aide R.
Dans certains cas, il arrive que l'on ne connaisse pas exactement le nom d'une fonction que l'on souhaite utiliser. Il est alors possible de retrouver cette fonction grâce à `help.search("mot_clé")` qui vous affichera une liste des fonctions en rapport avec `mot_clé`.
Il arrive également que l'on ne se souvienne que d'une partie du nom d'une fonction, on peut alors utiliser la fonction `apropos("pattern")` pour lister les noms des fonctions qui contiennent `pattern`. Ainsi, par exemple, si on souhaite connaître toutes les fonctions contenant `help` :

```
> apropos("help")
[1] "help"           "help.request"   "help.search"    "help.start"
[5] "link.html.help"
```

Enfin, si aucune des fonctions précédentes n'a permis de résoudre votre problème, il reste la fonction `RSiteSearch("mot_clé1 mot_clé2 ...")` qui permet de faire directement une recherche dans la "R-help mailing list" (sorte de forum de discussion dédié à l'utilisation de R) ainsi que dans toutes les documentations de R et qui affiche les résultats dans une page Web.

1.4 Fenêtre graphique

Une fenêtre graphique s'ouvre automatiquement à l'appel de la fonction `plot`. Vous pouvez aussi en ouvrir une en appellant la fonction `X11()`.

1.5 Répertoire et environnement de travail

Une des premières étapes lorsque l'on démarre R est de définir le répertoire de travail, c'est à dire le répertoire de votre disque dur avec lequel R va communiquer (lecture/écriture des données, des scripts, des résultats, des figures, etc.). Pour cela, il suffit d'aller dans le menu *Fichier->Changer le répertoire courant*. Pour vérifier que l'on est bien dans le bon répertoire, on peut taper la commade `getwd()` dans la console.
Enfin, lorsque l'on quitte R, soit par l'instruction `q()` dans la console soit en passant par le menu *Fichier->Sortir*, R nous demande si on souhaite enregistrer l'environnement de travail (ou la session). En cliquant sur *OUI*, R va alors sauvegarder toutes les données de votre environnement dans le répertoire de travail (défini plus haut) dans un fichier nommé *.RData* ainsi que toutes les lignes de commande que vous avez exécutées dans la console dans le fichier *.Rhistory*. Ce deux fichiers peuvent ainsi être rechargés au prochain lancement de R soit en démarrant R directement dans le répertoire de travail correspondant, soit en allant dans le menu *Fichier->Charger l'environnement de travail*.

2 Scalaires, Vecteurs et Matrices

2.1 R, une machine à calculer

R peut être utilisé comme une simple calculatrice, pour cela il suffit d'écrire directement l'opération dans la console et d'appuyer sur *Entrée* :

```
> (2 + 2) * 3 + 1
[1] 13
```

On peut ainsi faire toutes les opérations que l'on souhaite grâce aux opérateurs `+`, `-`, `*`, `/`, `^` (puissance), etc.

R possède aussi un très grand nombre de fonctions mathématiques prédéfinies. En voici une liste non exhaustive :

Principales fonctions mathématiques :

`log()`/`log10()`	Logarithme népérien/décimal.
`exp()`	Exponentielle.
`cos()`/`sin()`/`tan()`	Cosinus/Sinus/Tangente.
`abs()`	Valeur absolue.
`sqrt()`	Racine carrée (*square root*).

2.2 Opérations sur les scalaires

En dehors de la fonction de machine à calculer, on peut manipuler des scalaires, vecteurs ou matrices de façon symbolique, en leur affectant un nom. Pour créer un scalaire `a` et lui donner la valeur `6`, il faut taper `a=6`. Pour voir la valeur de `a`, on tape `a`. On peut ainsi créer des variables, puis effectuer des opérations sur ces variables :

```
> a = 6
> b = 2.5
> b * a
[1] 15
> a * b + 4
[1] 19
> b^a
[1] 244.1406
> log(a)
[1] 1.791759
> exp(-b)
[1] 0.082085
```

Remarque : Il est fortement déconseillé d'utiliser des noms de fonctions pour nommer ses objets comme par exemple `c=1` car `c()` est une fonction prédéfinie de R.

2.3 Créer un vecteur ou une matrice

Essayez la suite d'instructions suivantes, en les tapant dans l'éditeur, puis en les recopiant dans la fenêtre de contrôle :

```
> a = 6
> b = 5
> x = c(a, b, a + b, a * b)
```

Tapez maintenant `x` dans la fenêtre de contrôle :

```
> x
[1]  6  5 11 30
```

Vous avez créé un vecteur de quatre éléments qui contient 5, 6, 11 et 30. Vous pouvez effectuer des opérations sur le vecteur `x`. Par exemple, pour multiplier tous les éléments de `x` par 2 et retrancher la valeur 3, il suffit de taper :

```
> 2 * x + 3
[1] 15 13 25 63
```

Il existe différentes façon de créer un vecteur :

Créer des vecteurs :
Voici une liste des principales fonctions pour créer des vecteurs.

`c(2,3,4,5)`	Vecteur constitué des éléments mis entre parenthèse.
`rep(1,10)`	Vecteur où l'élément 1 est répété 10 fois.
`seq(1,10,by=2)`	Vecteur dont les élements vont de 1 à 10 avec un pas de 2.
`seq(1,10,length=5)`	Vecteur de 5 éléments entre 1 et 10.
`1:10`	Vecteur d'incides de 1 à 10.
`c(rep(1,10),2:8)`	Vecteur à partir de deux vecteurs.
`rep(c(1,2),c(4,10))`	Vecteur où 1 est répété 4 fois et 2 est répété 10 fois.

Vous remarquerez que les noms des fonctions utilisées pour créer des vecteurs sont des abréviations (`c` pour concatenate, `seq` pour sequence et `rep` pour replicate).

Exercice
Créez les vecteurs suivants, que vous appellerez `y1`, `y2`, `y3` et `y4`, avec `d = 4` et `e = 12` :
`y1` : une suite d'indices de 1 à 30.
`y2` : trois fois l'élément `d`, puis trois fois `d` au carré, puis trois fois la racine de `d`.
`y3` : la séquence de 1 à 20 avec un pas de deux.
`y4` : 10 chiffres compris entre 1 et 30 avec un intervalle constant.

Créer des matrices :
Voici une liste des principales fonctions pour créer des matrices.

`matrix(x, ncol=2)`	Transforme le vecteur `x` en une matrice de 2 colonnes.
`matrix(x, ncol=2, byrow=T)`	Idem, mais en remplissant la matrice ligne à ligne.
`rbind(x,y)`	Concatène `x` et `y` en lignes.
`cbind(x,y)`	Concatène `x` et `y` en colonnes.

Exercice
Pour comprendre comment créer une matrice, comparez les résultats des instructions suivantes :

```
> matrix(y3, nrow = 2)
> matrix(y3, nrow = 2, byrow = T)
```

La façon la plus simple pour fabriquer une matrice sous R est de créer d'abord un vecteur contenant tous les éléments de la matrice, puis de transformer ce vecteur en matrice à l'aide de la fonction `matrix`. Fabriquez la matrice M de 3 lignes et 5 colonnes ne contenant que des zéros.

Remarque : Il existe un certain nombre d'opérations spécifiques aux vecteurs et aux matrices. Ainsi, par exemple, comparez les deux opérations suivantes :

```
> y = 1:4
> m = matrix(1:16, ncol = 4)
> y * m
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    4   12   20   28
[3,]    9   21   33   45
[4,]   16   32   48   64
> y %*% m
     [,1] [,2] [,3] [,4]
[1,]   30   70  110  150
```

Quand on utilise `*` on multiplit simplement terme à terme les élements du vecteur dans la matrice alors qu'avec `%*%` on fait un vrai produit matriciel.

Principales fonctions et opérations sur les vecteurs et les matrices :

`%*%`	Produit matriciel.
`t(x)`	Transpose `x`.
`det(x)`	Calcule le déterminant de la matrice `x`.
`diag(x)`	Retourne la diagonale de la matrice `x` dans un vecteur.

2.4 Objets booléens et instructions logiques

Pour extraire les éléments d'un vecteur, on peut utiliser des instructions logiques. Un vecteur booléen est un vecteur dont chaque élément prend la valeur `TRUE` ou `T` (*vrai*) si l'instruction logique est vraie, et la valeur `FALSE` ou `F` (*faux*) si l'instruction logique est fausse.
Par exemple, comparez le contenu du vecteur x, et le résultat de l'instruction :

```
> x > 5
[1]  TRUE FALSE  TRUE  TRUE
```

L'instruction logique `x>5` renvoie un vecteur contenant `TRUE` pour les éléments de `x` supérieurs strictement à 5 et `FALSE` sinon. Les instructions peuvent aussi être combinées à l'aide des opérateurs & (*et*) et | (*ou*) (touche *AltGr+6*) :

```
> x > 5 & x < 30
[1]  TRUE FALSE  TRUE FALSE
```

Les opérations logiques :

`a==b`	Vrai si `a` est égale à `b`.
`a!=b`	Vrai si `a` est différent de `b`.
`a<b`	Vrai si `a` est strictement inférieur à `b`.
`a>=b`	Vrai si `a` est supérieur ou égale à `b`.
`a|b`	Vrai si `a` ou `b` ou les deux sont vrais.
`xor(a,b)`	Vrai si `a` ou `b` sont vrais.
`a&b`	Vrai si `a` et `b` sont vrais.

Exercice
En utilisant les vecteurs $y1$, $y2$ et $y3$ crées dans l'exercice précédent, créez les vecteurs booléens $b1$, $b2$, $b3$ et $b4$ qui correspondent aux instructions logiques suivantes :
$b1$: vrai pour les éléments du vecteur $y3$ inférieurs à 10, faux sinon.
$b2$: vrai pour les éléments du vecteur $y2$ supérieurs à 2 et inférieurs à 16.
$b3$: vrai pour les éléments du vecteur $y2$ supérieurs à 2 ou inférieurs à 16.
$b4$: vrai si les éléments de $y1$ valent 10.

2.5 Extraire les éléments d'un vecteur

Pour extraire les éléments du vecteur, il suffit de taper la commande `x[z]`, où `z` est soit un vecteur d'indices, soit un vecteur de booléens. Dans ce dernier cas, `z` doit obligatoirement avoir la même longueur que `x`. Par exemple :

```
> x = c(1, 3, 5, 7, 9, 12, 14, 18, 22)
> x[1]
[1] 1
```

```
> x[1:3]
[1] 1 3 5
> x[x > 5]
[1]  7  9 12 14 18 22
```

Exercice
Créez le vecteur `valeur` contenant 20 nombres aléatoires entre 0 et 1 en utilisant la commande :

```
> valeur = runif(20)
```

Créez ensuite un vecteur d'indices prenant la valeur 1 pour les 10 premiers nombres, et la valeur 2 pour les 10 suivants :

```
> indix = rep(c(1, 2), c(10, 10))
```

En utilisant une instruction logique sur le vecteur `indix`, créez un nouveau vecteur `dix` contenant les 10 premières valeurs du vecteur `valeur`. Pour vous aider, vous pouvez visualiser les deux vecteurs `indix` et `valeur` côte à côte à l'aide de la fonction `cbind`.

2.6 Extraire les élements d'une matrice

Pour extraire des éléments d'une matrice `Z`, il faut choisir les lignes et les colonnes de la matrice que l'on souhaite garder, en utilisant des indices et/ou des instructions logiques. `Z[x,y]` est la sous-matrice de `Z` contenant les lignes de `Z` définies par le vecteur `x` et les colonnes de `Z` définies par le vecteur `y`. `x` et `y` peuvent être des vecteurs d'indices ou des vecteurs booléens. L'instruction `Z[x,]` renvoie toutes les colonnes de `Z` pour les lignes correspondant à `x`. L'instruction `Z[,y]` renvoie toutes les lignes de `Z` pour les colonnes correspondant à `y`.
Exercice
Construisez la matrice `Z` suivante :

$$Z = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Puis affichez :
- L'élément de `Z` contenu dans la première ligne et la troisième colonne.
- La première ligne de `Z`.
- La troisième colonne de `Z`.
- La sous-matrice après avoir enlevé la première ligne et la première colonne de `Z`.

2.7 Les chaînes de caractères

Les chaînes de caractères sont un type d'objet qui permet de manipuler des séquences de lettres. Pour créer une chaîne de caractères il suffit de l'écrire entre guillemets :

```
> a = "Bonjour"
```

Il est aussi possible de faire des vecteurs de chaînes de caractères : `c("Bon", "jour")`.

Manipulation de chaînes de caractètres :

`strsplit(x, s)`	Coupe la chaîne `x` en fonction de la chaîne `s`. *Ex.* : `strsplit("bonjour", "j")` => `"bon" "our"`
`paste(x1,x2)`	Concatène les deux chaînes `x1` et `x2` .
`as.character(x)`	Convertit l'objet `x` en une chaîne.
`as.numeric("1.5")`	Convertit la chaîne `"1.5 "`en valeur numérique `1.5`.

2.8 Les listes

Les listes sont des objets particuliers qui permettent d'associer d'autres objets de natures différentes. Une liste se crée de la façon suivante :

```
> ma_list = list(val = 1, chaine = "Bonjour", vect = 1:10)
> ma_list$chaine
[1] "Bonjour"
```

Pour appeler un élément d'une liste il suffit d'écrire le nom de l'objet liste suivi de `$` + le nom de l'objet souhaité.

3 Tableaux de données : data.frames

Sous R, les tableaux de données sont des objets particuliers appelés `data.frame`.

3.1 Lecture d'un fichier de données

Pour lire un fichier de données, il faut commencer par créer une variable qui contient le nom du fichier grâce à la fonction `file.choose()`. Nous allons ouvrir le fichier "cinetik.txt". Tapez la commande ci-dessous dans l'éditeur et évaluez là dans la fenêtre de contrôle :

```
> nomfichier = file.choose()
```

R ouvre alors une fenêtre avec l'arborescence de fichiers de *Windows*. Recherchez le fichier "cinetik.txt" et cliquez sur son nom. Ensuite on utilise la fonction `read.table()` qui permet de lire un tableau de données :

```
> data = read.table(monfichier, header = TRUE)
```

L'instruction `header=TRUE` permet de préciser que la première ligne du fichier à lire est une ligne d'en-tête contenant le nom des colonnes du tableau. Vous pouvez afficher le tableau `data` en tapant :

```
> data
```

Notons qu'il est également possible de lire un fichier de données *via* une url :

```
> data = read.table("http://moulon.inra.fr/~mag/cinetik.txt", h = T)
```

Un tableau de données est un ensemble de vecteurs rangés colonne par colonne. Chaque colonne du tableau `data` correspond à une variable, chaque ligne à un individu. Ici, les données correspondent à des mesures de densité optique dans des cultures de bactéries soumises ou non à un traitement. Il y a cinq souches de bactéries, et toutes les souches ont été observées au cours de quatre expériences différentes. La structure du fichier est la suivante :

Desciption du fichier *cinetik.txt* :

`manip`	Numéro de l'expérience.
`souche`	Lettre pour le code de la souche de bactérie.
`temps`	Temps en heure.
`temoin`	Densité optique dans la culture temoin.
`eff`	Densité optique dans la même culture soumise à un traitement.

3.2 Extraire les colonnes d'un tableau

La fonction `colnames(data)` renvoie les noms des colonnes du fichier de données `data`. Pour voir uniquement une colonne du tableau (par exemple la colonne `temps`), il suffit de taper `data$temps`.

Exercice

Affichez les noms des colonnes du tableau `data`, puis affichez la colonne correspondant à la densité optique des bactéries témoins.

3.3 Extraire des éléments d'un tableau

Pour extraire les éléments d'un tableau, on peut aussi exécuter la commande `data[x,y]`, où `x` est un vecteur d'indices ou un vecteur de booléens (dans ce cas, il doit avoir le même nombre d'éléments que le nombre de lignes du tableau `data`), et `y` est un vecteur d'indices ou un vecteur de booléens (dans ce cas, il doit avoir le même nombre d'éléments que le nombre de colonnes du tableau `data`). L'instruction `data[x,]` renvoie toutes les colonnes du tableau pour les lignes correspondant à `x`. L'instruction `data[,y]` renvoie toutes les colonnes du tableau pour les lignes correspondant à `y`.

Exercice

Que vaut `data[1,5]` ?
Que vaut `data[,1:3]` ?
Que vaut `data$manip==1` ?
Que vaut `data[data$manip==1,]` ?

Que vaut `data$souche=="A"` ?
Extraire les trois premières lignes du tableau `data`.
Extraire les valeurs de `eff` pour la souche B et la manip 2.

3.4 Fonctions de desciption d'un tableau de données

Voici quelques fonctions utiles pour décrire et analyser un tableau de données.

Fonctions de base pour étudier et analyser des tableaux de données :

`dim(data)`	Renvoie les dimensions du tableau.
`length(data$souche)`	Renvoie la longueur du vecteur `data$souche`.
`colnames(data)`	Renvoie un vecteur contenant les noms des colonnes du tableau.
`max(data$eff, na.rm=T)`	Valeur maximale du vecteur `data$eff`.
`min(data$eff, na.rm=T)`	Valeur minimale du vecteur `data$eff`.
`table(data$manip, data$souche)`	Tableau de contingence pour les deux variales.
`summary(data)`	Renvoie un résumé des distributions de chaque variable de `data`.
`head(data)`/`tail(data)`	Renvoie les 6 premières/dernières lignes de `data`.

Exercice
Combien y-a-t'il de souches et de manips dans cette expérience ?

3.5 Réalisation d'un graphique

Différentes fonctions permettent de représenter graphiquement des données. Une fenêtre graphique s'ouvre automatiquement à l'appel de la fonction `plot`. La fonction `points` permet de rajouter des points sur un graphique. Par exemple, observez le résultat des instructions suivantes :

```
> plot(data$temps, data$temoin)
> points(data$temps, data$eff, col = 2)
```

Pour réaliser des graphiques plus jolis, vous pouvez jouer avec les options de la fonction `plot`.

Principaux arguments de la fonction plot :

`xlab=`	Label de l'axe des abscisses, "entre quotes".
`ylab=`	Label de l'axe des ordonnées, "entre quotes".
`main=`	Titre du graphique, "entre quotes".
`type=`	Type de points : "p" pour des *points* (valeur par défaut), "l" pour des *lignes*, "b" pour des *points* et des *lignes* (both), "h" pour des *barres verticales*, "n" pour ne rien afficher (none).
`pch=`	Forme des points : 1=rond, 2=triangle, ... Pour voir les possibilités, tapez : `plot(1:20, 1:20, pch=1:20, cex=2)`
`cex=`	Taille des points (défaut =1).
`lty=`	Type de lignes : 1=traits plains, 2=pointillés, ...
`lwd=`	Epaisseur des lignes (défaut =1).
`col=`	Couleur des points ou des lignes. 1=noir, 2=rouge, ...

Pour voir, exécutez cet un exemple :

```
> plot(data$temps, data$temoin, xlab = "Temps (heures)", ylab = "DO",
+     pch = 19, cex = 1.5)
> points(data$temps, data$temoin, pch = 19, cex = 1.5, col = 2)
```

Les principales fonctions graphiques :

Fonctions ouvrant une fenêtre graphique.

`plot(x,y)`	Trace des points et/ou des lines.
`hist(x, breaks)`	Trace un histogramme de la distribution de `x` .
`boxplot(list(x1,x2))`	Trace des "*boîtes à moutaches*" pour `x1` , `x2` ...

Fonctions lancées après l'ouverture d'une fenêtre graphique.

`points(x,y)`	Ajouter des points.
`lines(x,y)`	Ajouter des courbes.
`legend(x,y, leg)`	Ajouter une légende.
`arrows(x0,y0,x1,y1)`	Ajouter une flèche.

4 Fonctions

Les fonctions sous R sont des objets particulers qui permettent de découper un programme en un ensemble d'actions.

4.1 Fonctions prédéfinies

Il existe beaucoup de fonctions déjà programmées sous R. Une fonction à trois attributs :

- Son nom.
- Un ensemble d'arguments qui sont les objets nécessaires à la réalisation des instructions de la fonction.
- L'objet qu'elle renvoie.

Typiquement on appelle une fonction de la façon suivante :

```
x=nom_fonction(arg1, arg2, arg3)
```

Pour connaître la liste complète des fonctions prédéfinies de R, il suffit de taper :

```
> ls("package:base")
```

L'aide en ligne de R est particulièrement bien faite. Elle comprend une description de la fonction, son utilisation, la liste des arguments avec une explication de leur usage, des exemples et les valeurs par défauts.

4.2 Fonctions écrites par l'utilisateur

Il est possible d'écrire soit même des fonctions en utilisant l'éditeur. Voici un exemple de fonction, avec la syntaxe à utiliser :

`randomvec=function(a,n) {`	En-tête de la fonction.
`  alea=runif(n)*a`	`alea` contient `n` nombres aléatoires entre 0 et `a`.
`  alea`	Renvoie le vecteur `alea`.
`}`	Fin de la fonction.

Notez que le mot `function` est un mot réservé. Les accolades `{` et `}` définissent le début et la fin de la fonction. La dernière instruction doit contenir le nom de l'objet renvoyé par la fonction (ici, le vecteur `alea`). Avant de pouvoir appeler cette fonction dans la fenêtre de contrôle, il faut l'évaluer. Il suffit pour cela de la copier dans la fenêtre de contrôle. Ensuite, elle peut être appelée :

```
> Y = randomvec(10, 100)
> length(Y)
[1] 100
> Y[1:5]
[1] 9.300530 5.970160 9.975559 8.900480 7.662304
```

Exercice
Dans certains cas, il peut être utile de simuler des données. Nous vous proposons ici de simuler deux populations de même variance mais de moyennes différentes, puis de réaliser des permutations pour tester l'hypothèse H_0 que les deux moyennes sont égales. Il s'agit bien entendu d'un cas d'école, puisque l'on connaît ici la loi de probabilité de la statistique du test d'égalité des deux moyennes.

1. Utilisez la fonction `rnorm` pour simuler deux populations *pop1* et *pop2*, de même taille, de même variance mais de moyennes différentes. Les paramètres dont vous aurez besoin ici sont `m1` et `m2`, les deux moyennes, la taille de chaque échantillon `n` et l'écart-type `sigma`. Utilisez l'aide de la fonction `rnorm` pour connaître sa syntaxe.

2. Concaténez les valeurs obtenues dans un vecteur `val`, et créez un vecteur d'indices `pop` pour repérer les populations dont sont issues les valeurs simulées. Concaténez ces deux vecteurs dans un tableau de données que vous appellerez `don` en utilisant l'instruction `don=data.frame(pop=pop, val=val)`.
Vous pouvez faire un histogramme du vecteur `data$val` pour regarder la distribution des valeurs simulées.

3. Il peut être intéressant de répéter cette simulation avec des paramètres différents. Pour cela, créez une fonction que vous appellerez `randompop` qui prend comme arguments `m1,m2,n,sigma`, qui rassemble les instructions des étapes 1 et 2, et qui renvoie le tableau `data`. Copiez la fonction dans la fenêtre de contrôle. Testez ensuite votre fonction avec par exemple les instructions suivantes :

```
> data = randompop(1, 2, 100, 1)
> hist(data$val, 20)
```

Refaites plusieurs fois la simulation en changeant la valeur de `m2`.

4. Réalisez un test de student de comparaison des deux moyennes en affichant dans la fenêtre de contrôle :

```
>t.test(data$val[data$pop==1],data$val[data$pop==2])
>true.t=t.test(data$val[data$pop==1],data$val[data$pop==2])$statistic
>pval.t=t.test(data$val[data$pop==1],data$val[data$pop==2])$p.value
```

5 Programmation

5.1 Boucles

Comme dans la plupart des langages de programmation, il y a plusieurs façons de répéter une instruction en boucle, les boucles *for* et les boucles *while*. Les instructions suivantes sont équivalentes :

```
n=4
x=2
for(i in 1:n){
   x=x+2
}
```

```
x=2
n=1
while(n<5){
   x=x+2
   n=n+1
}
```

Que vaut le `x` à la fin de la boucle ?

5.2 Instructions conditionnelles

la syntaxe `if(condition){instruction}` permet de calculer les instructions uniquement si la condition est vraie. Par exemple, `if(x<0){x=-x}` transforme `x` en nombre positif. On peut aussi définir un jeu d'instructions pour le cas où la condition est fausse :

```
if(x=0){
   x=x+1
} else{
   x=0
}
```

Exercice
A partir du tableau data que vous avez simulé, vous pouvez maintenant réaliser un test de permutation pour tester l'hypothèse d'égalité des moyennes des deux populations. Si l'hypothèse H_0 est vraie, alors vous pouvez réaffecter au hasard des indices d'appartenance à une population ou à une autre à chaque entrée du tableau data créé précédemment.

1. Utilisez la fonction `sample` pour générer une permutation au hasard des indices contenus dans le vecteur `data$pop`, que vous appellerez `p.pop`.

2. Vous pouvez réaliser un test de Student pour comparer les moyennes des deux populations fictives ainsi créées, la population des entrées du tableau data pour lesquelles `p.pop==1`, et la population des entrées pour lesquelles `p.pop==2`. Vous pouvez stocker la statistique du test dans un vecteur `H0.t`.

3. Initialisez `H0.t` en utilisant l'instruction :
`>H0.t=NULL`
Puis écrivez une boucle pour réaliser 1000 permutations. A chaque nouvelle permutation, vous incrémenterez `H0.t` de la nouvelle valeur de la statistique de test en utilisant l'instruction :
`>H0.t = c(H0.t, t.test(data$val[p.pop==1],data$val[p.pop==2])$statistic)`
Vous avez créé un vecteur contenant 1000 valeurs de la statistique du test sous l'hypothèse H_0.

4. Vous pouvez maintenant calculer une probabilité critique empirique pour votre test. Pour cela, créez un vecteur booléen qui contient `TRUE` si la valeur absolue de `H0.t` est supérieure à la valeur observée `true.t` . La valeur absolue d'un réel s'obtient en utilisant la fonction `abs`. Pour finir, vous utiliserez la propriété suivante : si `x` est un vecteur booléen, alors `sum(x)` renvoie la somme des termes vrais du vecteurs.