Mémo R

-

Master MIASHS M1

2016-2017

Table des matières

R est un logiciel de statistique et un langage de programmation tourné vers la statistique. Il est téléchargeable gratuitement. De l'aide et de nombreuses informations sur R peuvent être trouvées sur le site web :

`http://www.r-project.org`

⇒ **Vous pouvez démarrer R tout seul sans fenêtre graphique.** (Si nécessaire, après avoir téléchargé, par exemple ici : `http://cran.univ-lyon1.fr`, puis installé le logiciel R.)

1 Présentation de RStudio

RStudio est un environnement de développement pour R que nous allons utiliser. Il permet de travailler plus efficacement avec R que la simple ligne de commande dans une console, en utilisant différentes informations rangées dans des fenêtres graphiques.

- Démarrer RStudio, si nécessaire après avoir téléchargé puis installé :
  1) le logiciel R (comme indiqué ci-dessus),
  2) puis l'environnement Rstudio ici : `https://www.rstudio.com/products/rstudio/download/`).

1.1 Agencement de RStudio

- En bas à gauche, la "console". Elle attend vos commandes R après le prompt >. Comme dans le cas précédent, vous pouvez lui demander ce que vaut 3+5.

- En haut à gauche, la fenêtre "éditeur". Chaque fichier est dans un onglet. Cela permet de taper des commandes R, de les enregistrer dans des fichiers (appelés fichiers de script) et de les retrouver ensuite. Vous pouvez ensuite les envoyer dans la console R, voir plus loin.

- En haut à droite, la fenêtre "workspace" qui liste les données dans la mémoire de R, et un onglet "History" où vous voyez l'historique des commandes que vous avez envoyées à R.

- En bas à droite, dans quatre onglets,
  - le système de fichiers (Files)
  - les graphiques produits par R (Plots)
  - les bibliothèques de commandes disponibles (Packages), mais nous ne les utiliserons pas dans ces TP.
  - les pages d'aide (Help)

1.2 Exécution d'une commande directement dans la console ($\rightsquigarrow$ R comme une calculatrice !)

Il suffit de taper une **commande** dans la fenêtre console (en bas à gauche), puis d'appuyer sur la touche d'entrée. L'exemple de commande le plus simple est celui d'une opération arithmétique. Taper dans la fenêtre console :

```
3 + 5
```

```
((3+6+9)/3 -(2*8)^2)/4
```

On peut également utiliser des prédicats logiques (opérateurs de comparaison, par exemple) dont le résultat est une valeur booléenne (TRUE ou FALSE).
Taper dans la fenêtre console :

```
3 < 2
```

Il est souvent plus intéressant de stocker l'objet résultant d'une commande dans une variable, afin de pouvoir le manipuler ultérieurement. Pour cela nous allons voir comment affecter un objet à une variable.

Garder des valeurs en mémoire : Affectation d'un objet à une variable

On peut utiliser la flèche d'affectation `<-` (attention `<-` obtenu avec le tiret sous la touche 6) ou le signe `=` pour affecter le résultat d'une commande à une variable `x`. Après affectation, `x` désignera l'objet résultat de la commande, jusqu'à ce qu'une autre affectation ait lieu qui remplace le contenu de la variable `x`.

Taper dans la console :

```
> x<-2
> x
> 2->x
> y
> x+x
> z=x+x
> z
```

Taper :

```
x <- 8
x > 3 (qui donne \verb+TRUE+ car la valeur contenue dans x est supérieure à 3)
```

Opérateurs

Voici quelques exemples d'opérateurs : +,-,*,/,`sqrt()`, `log()`, `exp()`, `abs()`, ...

Exercice : Taper :
```
> a<-2
> b=3
```
puis affecter à c la valeur $(\exp(b)\sqrt{\log(a)})^{-\frac{b}{a}}$.

2 Scripts R

2.1 Répertoire de travail

Le répertoire de travail (ou `"Working Directory"`) est l'endroit du système de fichiers de l'ordinateur où R va lire et écrire des informations par défaut.

- La commande `getwd()` permet de savoir quel est le répertoire de travail dans lequel on se trouve.
- **Lorsque vous démarrez un TP sous R, pensez à vous placer dans le bon répertoire de travail.** Il y a deux façons de procéder :

  - La commande `setwd` permet de le faire. Par exemple, de la façon suivante :
    `setwd("~/L3S6stat/TP1")`
    permet de choisir le sous-répertoire `TP1` du répertoire `L3S6stat` de votre compte `~` (à condition bien sûr d'avoir au préalable créé le répertoire `L3S6stat` et le sous-répertoire `TP1` dans votre compte).
  - La seconde méthode consiste à utiliser la commande `"Set Working Directory"` puis `"Choose Directory"` qui se trouve dans le menu `"Session"` (ou dans le menu `"Tools"` suivant les versions de RStudio).

- Pour chaque fiche de TP, nous utiliserons un script R. Il est conseillé de créer un répertoire `L3S6stat` dans lequel vous pourrez stocker :
  - les fichiers de données,
  - le ou les scripts R qui contiennent les commandes que vous aurez exécutées au cours du TP,
  - les fichiers de graphiques que vous aurez enregistrés (au format png ou pdf).
- Enfin, notez que tout ce qui suit un `#` dans un fichier script de R est interprété comme un commentaire.

2.2 Premier script R

Vous allez utiliser un premier script R. L'intérêt est de pouvoir sauvegarder votre code pour le retrouver et l'utiliser ultérieurement. Le script permet aussi d'exécuter simultanément plusieurs instructions au lieu de procéder ligne par ligne dans la console. Rappelons que tout ce qui suit un `#` dans une ligne du script est du commentaire. Vous pouvez ensuite envoyer les commandes de ces scripts dans la console R à l'aide des boutons :

- `"Run"` pour envoyer uniquement la ligne où se trouve le curseur, ou toute la sélection si une partie du script est sélectionnée.
- `"Source"` pour envoyer tout le script à R.

2.3 Exercice 1 (à rendre)

- Créer un fichier `memo.R` dans lequel vous testerez les principales commandes présentées dans les sections 1 et 3.
- Pour chacune des commandes testées, indiquer rapidement ce qu'elle effectue sous la forme d'un commentaire en le précédent par : #.
  (Ce fichier pourra vous servir d'aide-mémoire.)

3 Les Objets

Les entités que R crée et manipule sont des **objets**. Il peut s'agir de valeurs numériques, de vecteurs, de listes, de chaînes de caractères, de tableaux de données, de fonctions, ou de graphiques, par exemple. La structure des objets dépend de leur type. Ils comportent généralement plusieurs composantes auxquelles on peut accéder facilement. Le résultat d'une commande est, de façon générale, un objet.

3.1 Les vecteurs

Le vecteur est l'objet de base dans R. Un vecteur est un ensemble d'éléments de même type, le plus souvent des **valeurs numériques** mais aussi des **chaînes de caractères** ou des **booléens**. Un objet à un seul élément sera considéré comme un vecteur de taille 1.

- Pour créer un vecteur, on utilise la commande `c()` comme suit :
  `z <- c(1.25, 0.25, 5.34)`
  Taper ensuite `z` pour visualiser le contenu de `z`.

- Autre exemple : `s <- c("bleu","vert","marron")`
  puis taper `s` pour visualiser le contenu de `s`.

- Si l'on applique à un vecteur une fonction qui devrait s'appliquer à un nombre, celle-ci s'applique à chaque composante du vecteur.
  Taper :
  `z <- c(1.25, 0.25, 5.34)`
  `z*2`
  Le résultat est : `2.5 0.5 10.68`
  Chaque composante du vecteur `z` a été multipliée par 2.

- Pour accéder à la k-ème composante d'un vecteur `z`, il suffit de taper `z[k]`.
  Taper :
  `z[2]`
  Le résultat est : `0.25`

- Pour sélectionner un sous-vecteur d'un vecteur `z` allant de la k-ème à la ℓ-ème composante, il suffit de taper : `z[k:`ℓ`]`.
  Taper :
  `z[2:3]`
  Le résultat est : `0.25 5.34`

Opérations sur les vecteurs

- **Aller plus loin dans la construction de vecteurs ...**

```
> d=1:6
> x1<-rep(1,4) # rep : replicate
> x2<-rep(1:4,2)
> x3<-rep(1:4,each=2) # que constatez-vous ?
> d<-c(7,9,13)
> d1<-rep(d,3)
> d2<-rep(d,1:3)
> d3<-rep(1:2,c(10,15))
> length(d3) # longueur d'un vecteur
> d3>2  # logical
> length(d3>2)
> y1<-seq(1,10,0.5)
> y2<-seq(1,2,length=20) # seq : sequence : par rapport à y1
> y<-paste("X",1:10,sep="")
```

  N.B. : Vous l'aurez remarqué : les chaines de caractères doivent être entourées de guillemets, les valeurs logiques sont codées TRUE ou FALSE abrégées T et F. Enfin les données manquantes sont codées par la chaine de caractère NA.

- **Opérations sur les vecteurs numériques**
  Essayer d'utiliser +,-,*,/,sqrt(), log(), exp(), abs(), t()
  Ou encore :

```
> z<-3*x1+y1
> z<-x1%*%t(y1)  # multiplier x par transpos\'{e}e de y1
```

  Finalement, c'est quoi z ?

- **Opérations logiques sur les vecteurs**
  Avec par exemple

```
> x<-1:6
> y<-c(1,4,2,5,4,3)
```

  vous pouvez tester $x < y$, $x == y$, $x! = y$, $(x <= 3)\&(y > 3)$.

- **Autres fonctions utilisables sur les vecteurs :**
  `min(x)`, `max(x)`, `length(x)`, `sum(x)`, `prod(x)`, `sort(x)`, `mean(x)`, `cumsum(x)`, `cumprod(x)`, `summary(x)`, ...

```
> x<-c(2,4,6,5,3,1)
> rev(x)
> sort(x)
> sort(x,decreasing=TRUE)
> rev(sort(x))

> summary(x)
```

- **Remarque :** Pour en savoir plus sur une fonction, ne pas hésiter à utiliser la fonction d'aide : `help()`.

3.2 Les matrices : `matrix`

Cet objet est semblable à la notion mathématique de matrice : c'est un tableau contenant le plus souvent des **valeurs numériques**, mais aussi des caractères, ou des objets booléens. Cependant, **contrairement au dataframe** (voir plus loin), tous les éléments de la matrice doivent être du même type.

1) Création de matrices

- **avec l'ordre "matrix"**

Par exemple, pour créer une matrice de 3 lignes et 5 colonnes contenant les nombres de 1 à 15, on tape l'instruction suivante :

```
A <-matrix((1:15), nrow=3, ncol=5,byrow=T)
dim(A)
colnames(A)<-c("a","b","c","d","e") # donner un nom aux colonnes de A
rownames(A) <-c("l1","l2","l3") # donner un nom aux lignes de A
```

Les arguments `nrow` et `ncol` indiquent respectivement le nombre de lignes et de colonnes. L'argument `byrow` permet de remplir la matrice horizontalement (byrow=T) ou verticalement (byrow=F, qui est l'option par défaut).
La commande `dim(A)` : donne les dimensions (nombre de lignes et colonnes) de la matrice A.

Examiner aussi :

```
> Z1 <-matrix(1:12,nrow=3,byrow=T)
> str(Z1)
> x<-c(2,4,6,5,3,1)
> M2 <-matrix(x,nrow=2)
> M3 <-matrix(x,nrow=2,byrow=T)

> Z<-matrix(1:12,ncol=3)
> summary(Z)
```

- **par "collage" de vecteurs lignes ou colonnes**

```
> A<-1:4
> B<- seq(5,8)
> C<- 9:12
> M1 <-cbind(A,B,C) # collage des colonnes
> M2 <-rbind(A,B,C) # collage des lignes
> dim(M1)
> dim(M2)
> M3=cbind(M1,rep(3,4))
```

- **à partir d'un vecteur existant**

```
> x<-1:12
> dim(x)<- c(3,4)
```

2) Extraction d'éléments

```
> Z2[2,3]
> Z2[2,]
> Z2[,3]
> Z2[-2,]
```

3) Opérations courantes sur les matrices

```
> diag(1:5)
> diag(4)
> M <-matrix(1:12,nrow=3,byrow=T)
> dim(M)
> MM<-M[c(1:2),c(1:2)]
> diag(MM)
> MMtrans <-t(MM)
> MMinv<-solve(MM)
> MMinv%*%MM  #identite ?
```

Les opérations $+, -, *, /,$ *log*() s'effectuent élément par élément :

```
> Z1=matrix(1:12,ncol=3)
> 3*Z1
> Z2=matrix(1:12,ncol=3,byrow=TRUE)
> Z1*Z2
```

Mais ça ne correspond pas au produit matriciel ! Essayez alors :

```
> Z1%*%Z2
```

4) Opération avec la fonction *apply*

```
> apply(Z1,1,sum)
> apply(Z1,2,sum)
> apply(Z1,2,sum)/4
> apply(Z1,2,mean)
```

Voici quelques opérations sur les matrices :

- On peut multiplier la matrice A par un nombre réel en utilisant le signe `*` :
  `2*A`
  donne le résultat suivant :

```
    a  b  c  d  e
l1  2  4  6  8 10
l2 12 14 16 18 20
l3 22 24 26 28 30
```

- On peut multiplier 2 matrices A et B si leur taille le permet en utilisant la commande de produit matriciel `%*%` . Par exemple, si on veut faire le produit matriciel suivant :

$$AB = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix} \begin{pmatrix} 1 & 6 \\ 2 & 7 \\ 3 & 8 \\ 4 & 9 \\ 5 & 10 \end{pmatrix} = \begin{pmatrix} 55 & 130 \\ 130 & 330 \\ 205 & 530 \end{pmatrix}$$

  On tape les commandes :
  ```
  B=matrix((1:10),nrow=5)
  A%*%B
  ```

- Encore quelques commandes :
  ```
  > A[-c(1,3)] # supprime les lignes 1 à 3 de $A$.
  > diag(3) # crée la matrice identité 3x3.
  > diag(c(1,1,9)) # crée une matrice diagonale 3x3
                   # dont les éléments diagonaux sont 1,1,9.
  ```

N.B. Une matrice peut être transformée en dataframe (afin de lui appliquer plus naturellement les traitements statistiques) par la commande : `as.data.frame(A)`.

3.3 Tableau de données ou `data.frame`

Le `data.frame` est le format privilégié de stockage des données sur lesquelles on souhaite faire des traitements statistiques. Ses colonnes sont des vecteurs représentant des variables (qui peuvent être de types différents : **quantitatives** ou **qualitatives**) décrivant les mêmes individus ou observations **en ligne** : ce qui est plus souple qu'une matrice dont tous les éléments doivent être de même type. C'est donc un tableau qui se présente sous la forme (Individu x Variable). On crée un `data.frame` en spécifiant ses vecteurs-colonnes (les variables). La commande ci-dessous :

```
> D <- data.frame(c(10,20,25,15),c("bleu","bleu","vert","bleu"),c(F,F,T,F))
> D
```

donne :

```
1 10 "bleu" FALSE
2 20 "bleu" FALSE
3 25 "vert" TRUE
4 15 "bleu" FALSE
```

Le data.frame `D` possède 3 variables en colonne et 4 individus en ligne.

On peut nommer les variables codées en colonne de la façon suivante :

```
> names(D) <- c("age","couleur","fille")
```

ou bien encore leur donner un nom au moment même de la création du data.frame de la façon suivante :

```
> D <- data.frame(age = c(10,20,25,15),couleur=c("bleu","bleu","vert","bleu"),
fille = c(F,F,T,F))
```

N.B. Si un seul nombre est mis à la place d'un vecteur-colonne, une colonne de dimension adéquate est créée par R et le nombre est répliqué autant de fois que nécessaire :

```
> M <- data.frame(18,c(F,F,T))
> M
```

donne :

```
1 18 FALSE
2 18 FALSE
3 18 TRUE
```

On accède aux variables composant un data.frame à partir du nom de la variable (en colonne). Par exemple, pour obtenir la variable `couleur` du data.frame `D` :

```
D$couleur
```

ou bien comme avec les matrices par le numéro de colonne :

```
D[,2]
```

Pour extraire le sous-tableau composé des lignes i à j et des colonnes k à ℓ d'un data.frame `D`, on peut procéder comme avec les matrices de la façon suivante :

```
D[1:2,2:3]

1  "bleu" FALSE
2  "bleu" FALSE
```

qui donne uniquement les lignes 1 et 2 ainsi que les colonnes 2 et 3 de `D`.

3.4 Caractéristiques des objets

```
> class(x) # indique la classe de l'objet \texttt{x}.
> length(x) # indique le nombre de composantes de l'objet \texttt{x}.
> dim(x) # donne la taille d'une matrice ou d'un data.frame.
```

3.5 Exercice 2 (à rendre)

A effectuer à la suite du fichier nommé `memo.R`, après avoir ajouté le commentaire :

```
### Exercice 2
```

1) Créer deux vecteurs : `poids`, `taille` de composantes respectives $(60, 72, 57, 90, 95, 72)$ et $(1.75, 1.80, 1.65, 1.5, 1.74, 1.91)$.
2) Calculer `bmi=poids/taille^2`.
3) Créer un vecteur ayant seulement les composantes `>20` de `bmi`. Combien a-t-il d'éléments ?
4) Créer une matrice nommée `data` de type (individus $\times$ variables) avec les variables `poids`, `taille` et `bmi`.
5) Donner la commande permettant d'afficher la dimension de cette matrice.
6) Transposer cette matrice.

4 Importer, exporter des données

4.1 La commande `read.table()`

R peut lire n'importe quel fichier texte. La commande de base est `read.table()`. Elle va créer un *data.frame*.

Exemple :
Le fichier `"loisirs.txt"` contient les lignes suivantes (Notons que la première ligne contient le nom des variables) :

```
CSP      Circuit Mer Montagne Campagne Ville
Agriculteurs 7 56 23 11 3
Artisans 14 46 15 16 10
Cadres         13 45 11 20 12
```

Pour lire ce fichier et le stocker dans *l'objet* `vacances`, taper :

```
> vacances=read.table("loisirs.txt", header=T)
```

On peut ensuite afficher le contenu de la table `vacances` ou d'une de ses colonnes :

```
> vacances
> vacances$Circuit
```

Pour mesurer l'effet de l'option `header=T`, comparer le résultat des commandes suivantes :

```
> vacances=read.table("loisirs.txt")
> vacances
> dim(vacances)
> vacances=read.table("loisirs.txt", header=T)
> vacances
> dim(vacances)
```

4.2 Lire un fichier Excel

Pour lire un fichier issu d'un tableur (Excel ou Openoffice), il faut tenir compte que l'on a des colonnes *(séparé par des tabulations, des virgules, des point-virgules, ...)* et que les nombres décimaux s'écrivent par exemple $3,24$ dans le tableur alors que R ne reconnaît que 3.24.

Le mieux est d'enregistrer votre fichier tableur sous la forme d'un fichier texte (option enregistrer sous). On a alors 2 types d'extension classiques :
- les fichiers avec l'extension *.txt* où le séparateur est un espace,
- les fichiers avec l'extension *.csv* (comma separated values) où le séparateur est "," ou " ;".

A titre d'exemple, le fichier `"apparts.csv"` comporte 3 colonnes (Surface ; Prix au m^2 ; Prix Total). Comparer le résultat des commandes suivantes pour charger ce fichier sous R :

```
> apparts = read.table("apparts.csv", sep=";", header=T)
> apparts = read.table("apparts.csv", sep=";", dec=",", header=T)
```

Attention : la commande précédente suppose que le fichier `"apparts.csv"` est dans le répertoire courant. Dans le cas contraire, il faut :

- soit donner le chemin pour accéder au fichier, par exemple :
  `> read.table("C:\\Data\\apparts.csv",sep=";",dec=",",header=T)`
- soit changer de répertoire de travail :
  ⇒ Utiliser la commande *Changer le répertoire courant* de l'onglet *Fichier* de **R** pour se placer dans le bon répertoire.
- soit taper : `> setwd("C:\\Data")` et **R** travaillera dans ce répertoire directement.

(Pour rappel, la commande `getwd()` vous donne le répertoire actuellement utilisé.)

4.3 Sauvegarder des données sous forme de table

- Pour sauvegarder des données **dans un fichier texte**, la fonction de base est `write.table()` :
  `> write.table(apparts,"monFichierApparts.csv",sep=";")`
  les arguments sont les mêmes que pour la lecture.
- Pour sauvegarder et charger des fichiers **au format R** :
  `> save(apparts,file="my_apparts.rda")`
  `> load("my_apparts.rda")`

4.4 Exercice 3 (à rendre)

A effectuer à la suite du fichier nommé `memo.R`, après avoir ajouté le commentaire :

```
### Exercice 3
```

1) Donner les commandes permettant de lire les fichiers `"loisirs.txt"` et `"apparts.csv"` (après avoir enregistrés ces fichiers sur votre ordinateur).
2) Que donne la commande `summary(apparts)` ? (répondre dans le fichier sous la forme d'un commentaire #)
3) Saisir le tableau ci-dessous dans un tableur (Excel ou Openoffice) :

Num	Tree	Age	Circumference
1	1	118	30,12
2	1	484	58,13
3	2	664	87,59
4	2	1004	115,26

   puis enregistrer ce tableau dans un fichier au format .txt.
   - Ecrire une ligne de commande permettant d'importer ce fichier sous R.
   - Vérifier la dimension de la table ainsi obtenue.
   - Pouvez-vous ajouter `+1` à tous les éléments de la colonne `Age` ?
   - Pouvez-vous ajouter `+1` à tous les éléments de la colonne `Circumference` ?
4) Répéter la question précédente en enregistrant le tableau sous le format .csv.

5 Les graphiques

On distingue dans R :

- des commandes graphiques dites « haut-niveau » qui vont créer un graphe ou la base d'un graphe : `plot, pairs, hist, pie, ...`
- des commandes graphiques dites « bas-niveau » qui se rajoutent à un graphe existant : `points, lines, abline, legend, locator, ...`

Les paramètres graphiques (taille des caractères, couleur, axes, ...) peuvent être entrés en option des commandes de haut niveau ou être gérés globalement par la fonction `par()`, qui possède 68 paramètres différents, visibles en tapant :

```
>?par
```

Dans ces paramètres, il y a notamment : titre (`main`), couleurs (`col`), taille des points (`cex`), forme des points (`pch`), relier des points par des lignes (`type`), limites des axes (`xlim`, `ylim`) ... et bien d'autres choses encore !

```
> x<-seq(0,10,length=50)
> y<-sqrt(x)
> plot(x,y)
> plot(x,y,type="l")
> plot(x,y,type="s")
## essayez aussi :
> par(mfrow=c(2,2))
> plot(x,y)
> plot(x,y,type="l")
> plot(x,y,type="s")
> plot(x,y,pch="s")
> abline(v=4,col=2,lty=2)
```

1) **Histogramme** : fonction `hist()`
Si x désigne un vecteur d'observations, hixt(x) permet de tracer l'histogramme associé à x.

```
> x<- rnorm(1000,5,2) #simuler 1000 r\'{e}alisations d'une loi normale
> hist(x)
```

Remarque :
Arguments de `hist` :
`freq` =T si les hauteurs des rectangles représentent des effectifs, =F pour des fréquences ;
`breaks` = entier : nbre de classes demandé ;
`labels` =T si "étiquetage" des rectangles ;
`main` = 'chaine caractères' pour mettre un titre à l'histogramme ;
`xlab` = 'chaine caractères' pour mettre une légende à l'axe des x ;
`ylab` = 'chaine caractères' pour mettre une légende à l'axe des y ;
`col` = vecteur chaine caractères couleurs des rectangles ;
etc ...

```
> hist(iris$Petal.Length)
```

2) **Diagramme en barres** : fonction `barplot()` La fonction `barplot()` permet de tracer des diagrammes en barres (séparées ou juxtaposées).

faire un diagramme en barres séparées (pour variable nominale) :

```
> barplot(c(10,15),names=c("H","F"))
```

faire un diagramme en barres juxtaposées (pour variable ordinale) :

```
> barplot(c(10,15),space=0,names=c("Petit","Grand"))
```

3) **Diagramme en bâtons** : on utilise la fonction `plot()` conjuguée à la commande `table`. La fonction `table()` permet de construire le tableau de distribution. On peut ensuite tracer le diagramme en bâtons sur la base de ces effectifs ou après transformation en fréquence.

```
> x=c(rep(1,15),rep(2,5),rep(3,10),rep(4,2))
> x
> table(x)
> plot(table(x),type="h")
```

4) **Graphique à 2 variables** : fonction `plot()`.

Exemple 1 :

```
> data(iris) # iris est un jeu de données disponible sous R
> head(iris) # affiche les 6 premières lignes de la table iris
> plot(iris$Sepal.Length,iris$Petal.Length)
> plot(iris$Sepal.Length,iris$Petal.Length,main="Iris",xlab="Sepale",
ylab="Petale",pch=22,bg="yellow",col="red",xlim=c(0,8),ylim=c(0,8),
bty="l",cex=1.5,las=1,col.axis="blue")
```

Exemple 2 :

```
> x=runif(100)
> bruit=rnorm(100,sd=0.5)
> y=1+2*x+bruit
> plot(x,y,cex=1.5) # représentation graphique du nuage de points
> lines(x,1+2*x,lwd=2,col="blue") # ajout de la droite de régression
> segments(x,y,x,1+2*x) # ajout des segments
> text(0.3,3.5,"La régression linéaire",col="blue",cex=2)
# ajout d'un texte en (0.3,3.5)
```

Exemple 3 : que fait la commande ci-dessous ?

```
> plot(iris$Petal.Length~iris$Species)
```

Une commande utile pour positionner ou connaître les coordonnées d'un point. Après avoir saisi la commande :

```
> locator()
```

Cliquez sur un point dont vous souhaitez connaitre les coordonnées, puis sur le bouton "`finish`" (en haut à droite de votre graphique). Les coordonnées du point s'affichent alors dans la console.

5.1 Quelques fonctions graphiques

Commande	Description
`plot`	Permet de faire des nuages de points, des courbes en donnant une liste de points serrés par lesquels passe la courbe,
`hist`	Permet de faire des histogrammes en densité d'effectifs ou densité de fréquences
`boxplot`	Permet de tracer un ou plusieurs boxplot en parallèle,
`points`	Permet d'ajouter à un graphique ouvert des points,
`legend`	Permet d'ajouter une légende à un graphique ouvert,
`title`	Permet d'ajouter un titre à un graphique ouvert

Exercice

- Faire les histogrammes des deux distributions observées de la `taille` et de `pb` avec les options par défaut (choix des classes automatiques, le choix manuel des classes fera l'objet du TP2).

- Taper la commande suivante qui crée un `data.frame` contenant de nombreuses informations :

```
DOtaille=hist(taille)
DOtaille
```

Observer et dire ce que contiennent les vecteurs suivants du `data.frame DOtaille` :

- `DOtaille$breaks`
- `DOtaille$counts`
- `DOtaille$intensities`
- `DOtaille$density`
- `DOtaille$mids`
- `DOtaille$xname`
- `DOtaille$equidist`

6 Quelques fonctions statistiques élémentaires

Quelques fonctions statistiques pour résumer une distribution observée codée sous forme de vecteur (ou de `data.frame`) :

Commande	Description
`summary`	Calcule des statistiques résumées sur un vecteur d'observations (s'adapte aux vecteurs "numeric", "factor", ...) ou sur une table de données, colonne par colonne.
`mean`	Calcule la moyenne d'un vecteur
`sd`	Calcule l'écart-type d'un vecteur
`median`	Calcule la médiane d'un vecteur
`quantile`	Calcule les quantiles d'un vecteur d'observations.

Exercice Utiliser les fonctions `mean`, `quantile`, `summary`, `sd` pour calculer les résumés numériques des deux distributions observées des variables `taille` et `pb`.