Introduction à R

Julien JACQUES

22/10/2018

Le logiciel R

Le logiciel R (disponible sur *http://www.r-project.org/}) est un logiciel de Statistique libre ayant un certain nombre d'atouts:

- il permet l'utilisation des **méthodes statistiques classiques** à l'aide de fonctions prédéfinies,
- il permet de créer ses propres programmes dans un **langage de programmation** assez simple d'utilisation,
- il permet d'utiliser des **techniques statistiques innovantes** et récentes à l'aide de package développés par les chercheurs et mis à disposition sur le site du CRAN (*http://cran.r-project.org/}).

Interface R studio

Le logiciel R fonctionne initialement en ligne de commande, mais des interfaces permettent une utilisation plus conviviale.

Nous proposons ici de travailler avec l'interface RStudio, téléchargeable sur : \begin{center} *http://www.rstudio.com/} \end{center}

Interface R studio

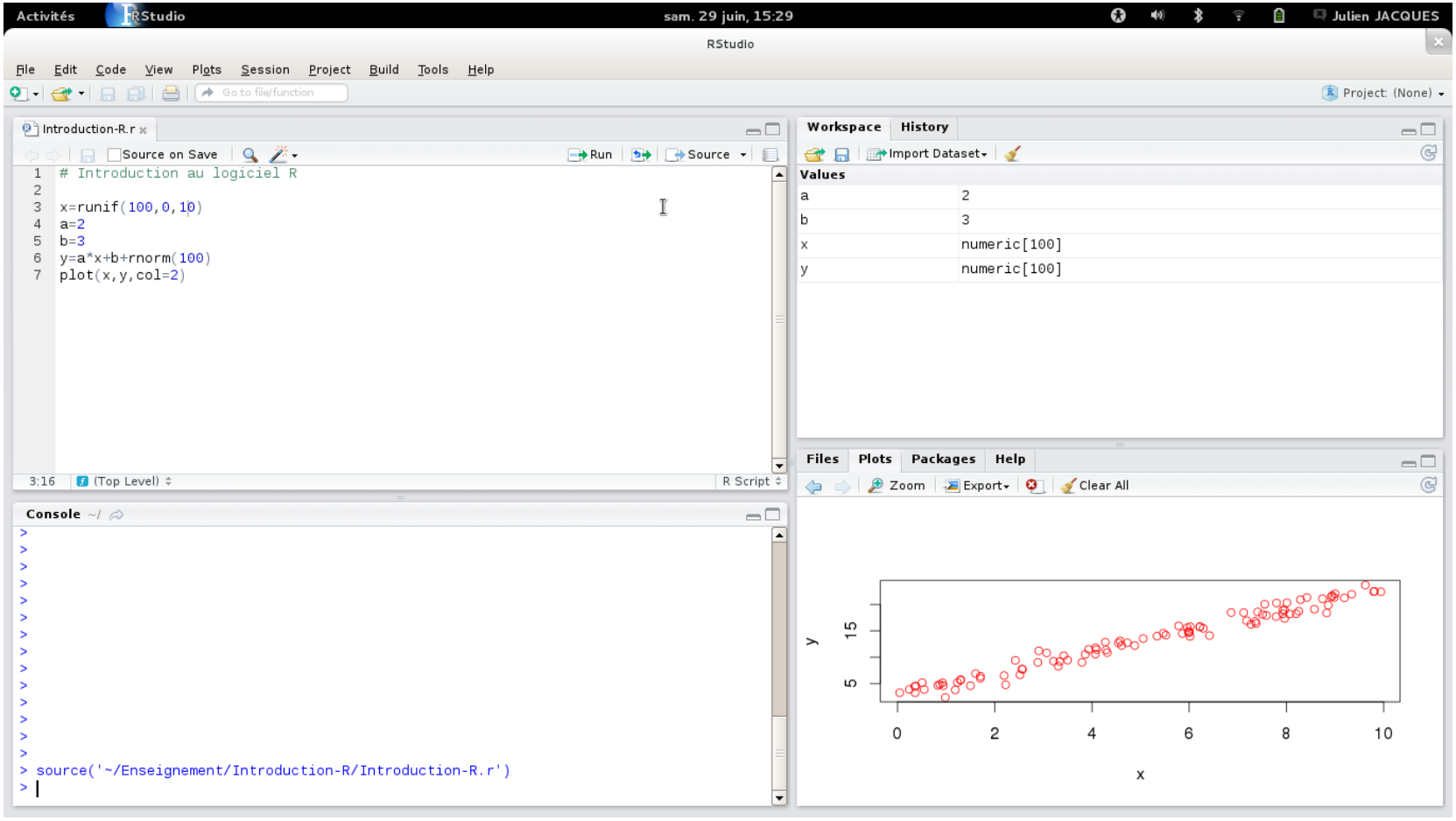


Figure 1:

Interface R studio

L'interface RStudio est généralement composée de quatre fenêtres:

- **Fenêtre d'édition** (en haut à gauche) : fichiers contenant les scripts R que l'utilisateur est en train de développer. Icônes permettent la sauvegarde, l'exécution d'une partie de code sélectionnée (*run*) ou de l'intégralité du code (*source*).
- **Fenêtre de commande** (en bas à gauche) : cette fenêtre contient une console dans laquelle les codes R sont saisis pour être exécutés.
- **Fenêtre espace de travail / historique** (en haut à droite) : contient les objets en mémoire, que l'on peut consulter en cliquant sur leur noms, ainsi que l'historique des commandes exécutées,
- **Fenêtre explorateur / graphique / package / aide** (en bas à droite) : l'explorateur permet de se déplacer dans l'arb, la fenêtre package montre les packages installés et actuellement chargés et la fenêtre d'aide contient la documentation sur les fonctions et packages.

Le répertoire de travail

Le répertoire de travail par défautest celui à partir duquel vous avez lancer l'interface RStudio.

Il sera pratique de se placer dans un répertoire de travail bien défini, celui par exemple contenant le fichier *.r} dans lequel vous tapez vos scripts R. Pour cela, utilisez le menu de l'interface :

- Session
  - Set Working Directory
    - To Source File Location

Par la suite, lorsque vous serez amené à charger des jeux de données, si ceux-ci sont placés dans le répertoire courant dans lequel vous vous êtes placé, vous n'aurez pas à saisir le chemin complet de ce répertoire.

Les packages

Un grand nombre de fonctions, contenus dans différents packages, sont installés dans la version de base du logiciel R.

Il est possible d'installer des packages supplémentaires, contenant d'autres fonctionnalités :

```
install.packages('FactoMineR')
```

Il faudra ensuite charger le package :

```
library('FactoMineR')
```

L'installation n'est à réaliser qu'une seule fois, alors que le chargement du package doit être fait au lancement de chaque nouvelle session.

Premières commandes R

R peut être utilisé pour réaliser des opérations élémentaires :

```
((1+sqrt(5))/2)
```

```
## [1] 1.618034
```

dont le résultat peut être stocké dans une variable

```
a=((1+sqrt(5))/2)
```

gardée en mémoire (*a} apparaît alors dans la fenêtre espace de travail), et qui peut être ré-utilisée par la suite :

```
nombredor = sqrt(a)
```

Pour effacer les variables en mémoire dans la session R, il faut taper la commande suivante (ou plus simplement utiliser l'icône *balai*) :

```
rm(list=ls())
```

Scalaires, vecteurs et matrices

La variable *a* est un scalaire

```
a=((1+sqrt(5))/2)
```

R gère aussi les vecteurs et matrices, et grand nombre d'opérations peuvent être effectuée direcement sur ces objets (sans recours à des boucles).

```
x = c(7,8,9)
y = 1:3
z = rep(x,y)
```

Matrices et array

La commande *matrix* permet de créer une matrice

```
M = matrix(z,2,3)
```

ce qui peut également être fait en concaténant des vecteurs en ligne (*rbind*) ou en colonne (*cbind*) :

```
M = rbind(x,y)
M = cbind(x,y)
```

Les tableaux à plus de 2 dimensions (*array*) sont également utilisables :

```
T = array(0,dim=c(2,3,4))
```

Listes

Une liste est une combinaison de structures de données de natures potentiellement différentes :

```
L=list(elt1=c(1,2,3),elt2=matrix(rnorm(9),3,3),
       elt3='tutu',elt4=seq(1,4,by=0.5))
```

Les éléments de la liste sont alors accessible par un '$', et les noms des éléments par la commande *names* :

```
L$elt4
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

```
names(L)
```

```
## [1] "elt1" "elt2" "elt3" "elt4"
```

Data Frame

L'objet le plus adapté au stockage des jeux de données est le **data.frame**, qui est un tableau dont :

- les **colonnes représentent les variables** (chaque colonne pouvant être d'un type différent), accessibles par le nom de la variable comme pour une liste
- les **lignes représentent les individus**

```
Mdf = as.data.frame(M)
str(Mdf)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: num  7 8 9
##  $ y: num  1 2 3
```

Toutes les fonctions d'analyse statistique sous R sont prévues pour travailler avec des données stockées sous la forme d'un data frame.

Les fonctions

R dispose d'un grand nombre de fonctions prédéfinies (outres les fonction d'analyses statistiques...) :

```
mean(x)
```

```
## [1] 8
```

```
rnorm(5)
```

```
## [1]  1.8908421 -0.5126003 -0.5886387 -0.6413107  0.9289923
```

```
rnorm(5,mean=1,sd=2)
```

```
## [1] -0.5077296  0.7489080  2.3613894  0.8063986  1.6545331
```

Astuce: lorsque vous commencez à taper le nom de la fonction, la touche *tabulation* permet de voir les différentes complétion possibles. Lorsque le nom de la fonction est totalement saisi, la tabulation permet de voir les arguments attendus par la fonction.

L'aide et la documentation

L'aide sur une fonction est accessible des deux façons suivantes :

```
help(rnorm)
?rnorm
```

Astuce: un bon moyen pour trouver de l'aide et des exemples sur une fonction consiste simplement à taper le nom de la fonction sous Google.

Les scripts

Vous n'êtes pas obligé de taper toutes les commandes R dans la fenêtre de commande. Il est possible de créer des scripts R (dans la fenêtre d'édition), en les enregistrant avec une extension '.r' et de les exécuter à l'aide de la commande source :

```
source('myscript.r')
```

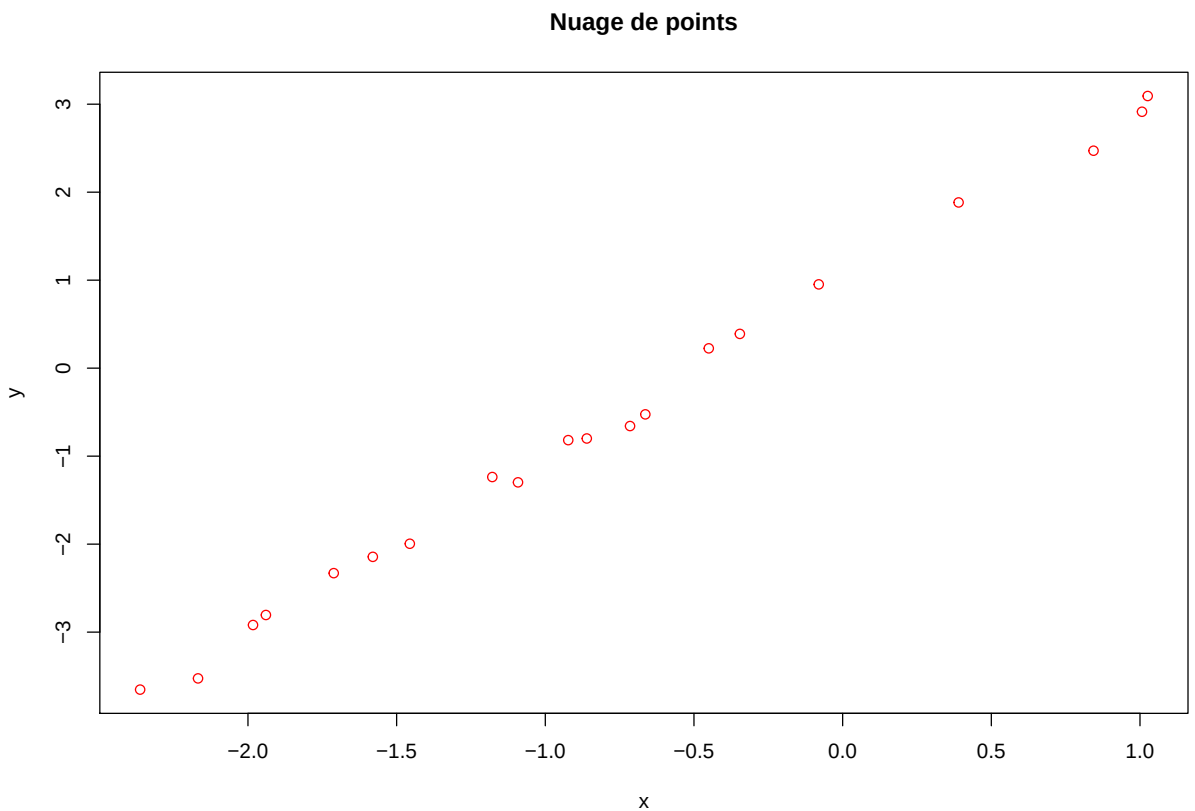
Les icônes *source* et *run* permettent d'exécuter tout ou partie du script R affiché dans la fenêtre d'édition.

Les graphiques

R permet de créer un grand nombre de graphiques.

La fonction *plot* permet de représenter un nuage de points :

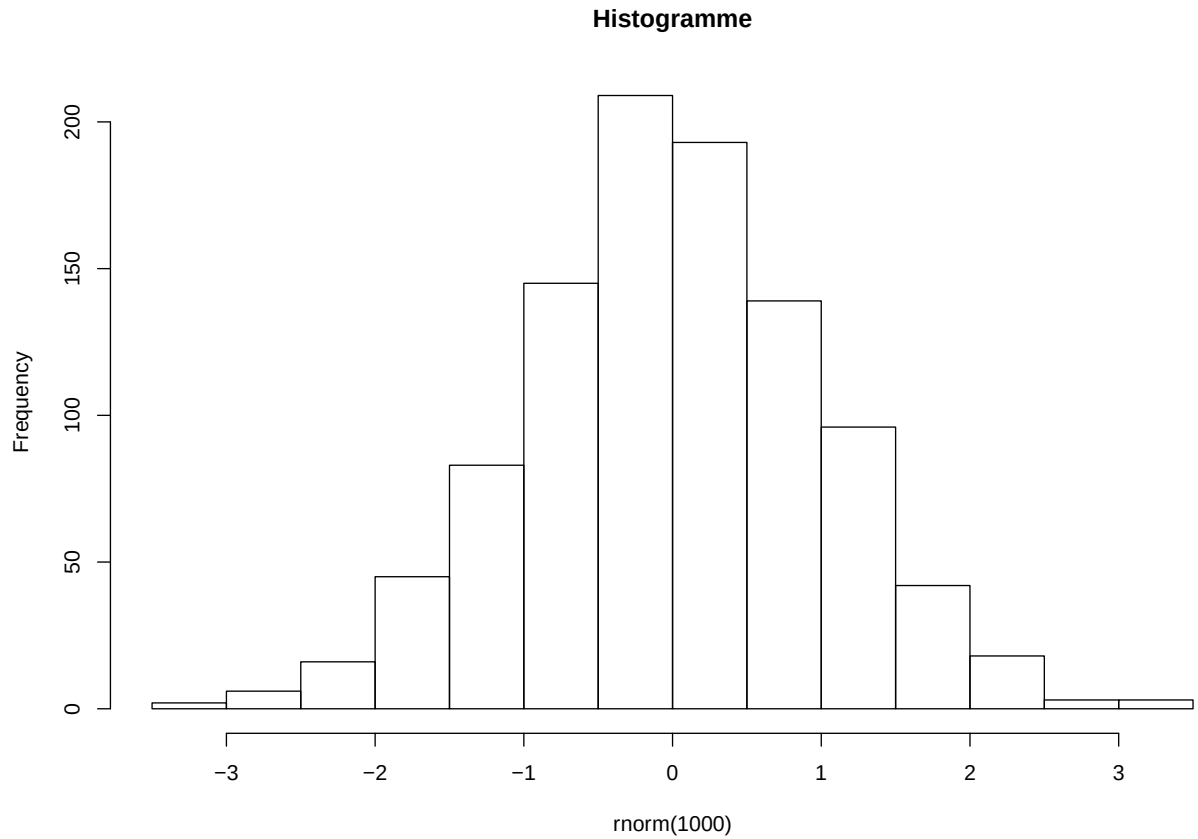
```
x=rnorm(20);y=2*x+1+rnorm(20,0,0.1)
plot(x,y,type='p',xlab='x',ylab='y',
     main='Nuage de points',col=2)
```



Les graphiques

La fonction *hist* permet de représenter un histogramme :

```
hist(rnorm(1000),breaks=20,main='Histogramme')
```



Astuce: le package *ggplot2* permet de créer des graphiques visuellement plus évolués

Importer et exporter des données

Il y a plusieurs façons d'importer et d'exporter des fichiers de données dans R. Les principales sont les fonctions **write.table** et **read.table** qui permettent respectivement d'exporter dans un fichier texte un data frame et d'importer un fichier texte (de type individus en ligne et variables en colonnes) dans un data frame.

Voici un exemple d'utilisation :

```
df=data.frame(x=c(11,12,14),y=c(19,20,21),z=c(10,9,7))
write.table(df,file='mydataframe.txt',row.names=FALSE)
newdf=read.table('mydataframe.txt',header=TRUE)
```

L'argument *row.names=FALSE* de *write.table* permet de ne pas sauvegarder de noms aux lignes. Par défaut l'option *col.names=TRUE* sauvegarde les noms des colonnes, qui sont ensuite ré-importées grâce à l'option *header=TRUE* de *read.table*.

Importer des données depuis Excel

La fonction **read.xls** (ou **read.xls**) permet d'importer des données directement depuis Excel.

```
library(gdata)
DataVoitures = read.xls("DataVoitures2010.xlsx")
```

```
str(DataVoitures)
```

```
## 'data.frame':    29 obs. of  13 variables:
##  $ Type         : Factor w/ 5 levels "4 x 4","Citadine",..: 2 3 3 5 3 2 2 3 5 3 ...
##  $ Marque       : Factor w/ 16 levels "Audi","Bmw","Citroen",..: 13 11 11 11 13 11 13 3 13 13 ...
##  $ Modele       : Factor w/ 29 levels "207","5008","528",..: 12 22 2 4 19 1 26 7 14 18 ...
##  $ Tarif        : int  10940 17250 24400 35500 24250 14600 12050 26350 33750 17650 ...
##  $ Cylindree    : int  1461 1560 1560 1997 1461 1398 1461 1560 1995 1461 ...
##  $ Puissance    : int  65 90 110 136 110 70 65 110 130 85 ...
##  $ Consommation: num  5.4 5.7 6.5 7.1 5 4.4 4.3 5.6 7.4 5.3 ...
##  $ CO2          : int  115 150 140 179 133 117 113 149 190 140 ...
##  $ Vitesse      : int  165 150 183 190 187 166 164 191 184 158 ...
##  $ Coffre       : int  255 675 679 830 508 270 230 439 650 660 ...
##  $ Poids        : int  1015 1407 1472 1600 1386 1194 980 1503 1757 1389 ...
##  $ Longueur     : num  3.82 4.38 4.53 4.73 4.8 4.05 3.6 4.78 4.66 4.21 ...
##  $ Hauteur      : num  1.42 1.87 1.64 1.75 1.45 1.47 1.47 1.46 1.73 1.8 ...
```

Attention : la paramétrisation de cette fonction est assez spécifique à chaque machine (système d'exploitation, version de Java, . . .). Quelques tests seront nécessaires. . .

Eléments de programmation

La syntaxe pour la condition **if** est la suivante (la condition *else* peut être omise) :

```
w=2
if (w>3) {res=2} else {res=4}
print(res)
```

```
## [1] 4
```

Une boucle **for** a la syntaxe suivante

```
vec=c()
for (i in 1:10){
  vec=c(vec,i)
  cat('iteration numero: ',i,'\n')
  }
```

```
## iteration numero:  1
## iteration numero:  2
```

Ecrire ses propres fonctions

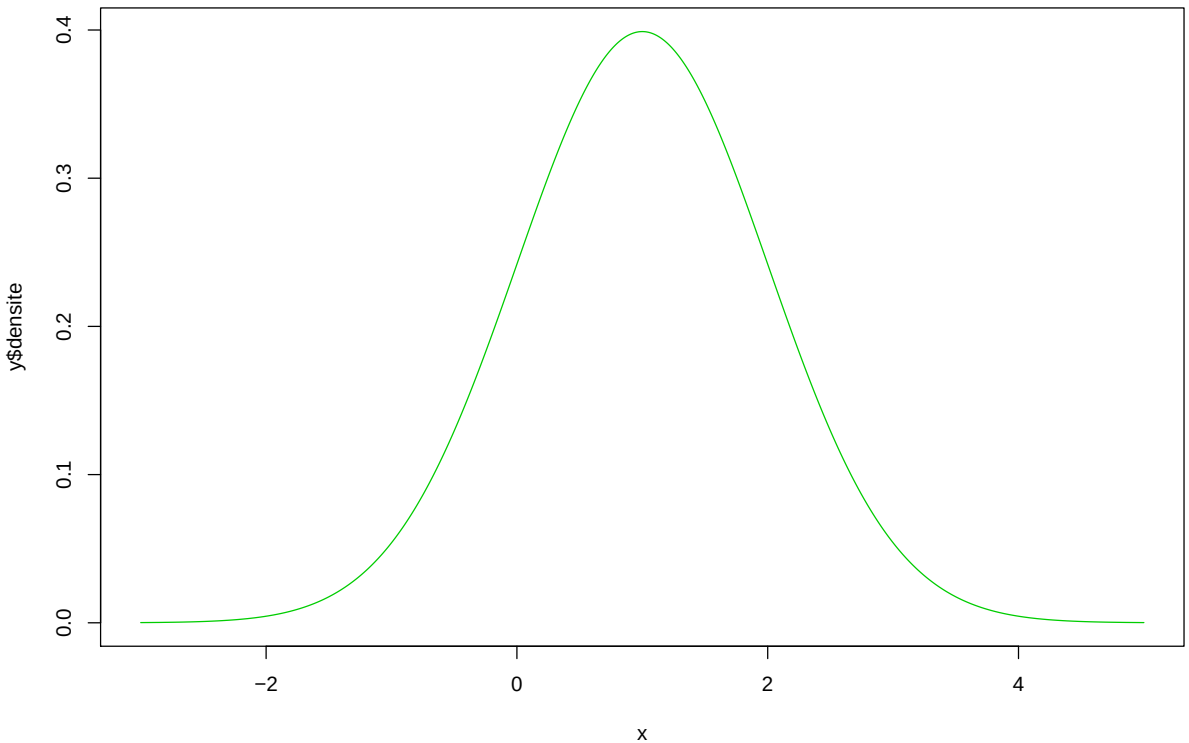
Un des grands intérêts du logiciel R est qu'il est possible de créer ses propres fonctions.

```
mafonction <- function(arg1,arg2=0,arg3=1){
tmp=exp(-1/2*((arg1-arg2)^2/(sqrt(arg3))))/sqrt(2*pi*arg3)
return(res=list(argument1=arg1,densite=tmp))
}
```

- les arguments de la fonction sont donnés après l'instance *function*
- une valeur par défaut à un argument peut être donnée en indiquant cette valeur lorsque les arguments sont définis
- le résultat de la fonction peut être de différente nature (scalaire, vecteur, matrice, liste...)

Ecrire ses propres fonctions

```
x=seq(-3,5,0.01)
y=mafonction(x,arg2=1)
plot(x,y$densite,type='l',col=3)
```



Principales fonctions : création de données

- **read.table** : lit un data frame à partir d'un fichier. Arguments: *header=TRUE* si la première ligne correspond aux intitulés des variables; *sep="* pour indiquer le séparateur de variables dans le fichier; *skip=n* pour ne pas lire les n premières lignes.
- **write.table** : sauvegarde un data frame dans un fichier.
- **c** : concatène des scalaires en un vecteur.
- **rbind, cbind** : concatène en ligne ou en colonne des vecteurs en une matrice.
- **list** : crée une liste.
- **matrix** : crée une matrice à *nrow* lignes et *ncol* colonnes.
- **data.frame** : crée un data frame.
- **array** : crée un tableau dont l'argument *dim* permet de préciser le nombre de dimensions ainsi que la taille de chaque dimension.
- **seq** : créer une séquence d'entiers.
- **rnorm, runif** : simule la génération d'une variable aléatoire normale, uniforme.

Principales fonctions : manipulation de données

- **x[n]** : n-ème élément du vecteur x.
- **x[n:m]** : n-ème au m-ème éléments du vecteur x.
- **x[c(k,l,m)]** : k-ème, l-ème et m-ème éléments du vecteur x.
- **x[x>m & x<n]** : éléments de x compris entre m et n.
- **l$x** ou **l[["x"]]**: élément x de la liste l.
- **M[i,j]** : élément ligne i et colonne j de la matrice M.
- **M[i,]** : i-ème ligne de la matrice M.
- **t(M)** : transposée de la matrice M.
- **solve(M)** : inverse de la matrice M.
- **M%*%N** : produit des matrices M et N.
- **sort(x)** : tri du vecteur x.

Principales fonctions : information sur les variables

- **length** : longueur d'un vecteur.
- **ncol**, **nrow** : nombre de colonnes et de lignes d'une matrice.
- **str** : affiche le type d'un objet.
- **as.numeric**, **as.character** : change un objet en un nombre ou une chaine de caractères.
- **is.na** : teste si la variable est de type 'NA' (valeur manquante).

Principales fonctions : statistiques

- **sum** : somme d'un vecteur.
- **mean** : moyenne d'un vecteur.
- **sd**, **var** : écart-type et variance d'un vecteur (dénominateur $n-1$)
- **rowSums**, **rowMeans**, **colSums** ou **colSums** : somme et moyenne en ligne ou en colonne d'une matrice.
- **max**, **min** : maximum et minimum d'un vecteur.
- **quantile(x,0.1)** : quantile d'ordre 10% du vecteur x.

Principales fonctions : graphiques

- **plot(x)** : représente une série de points (ordonnée x et numéro d'indice en abscisse).
- **plot(x,y)** : représente un nuage de points d'abscisse x et d'ordonnée y.
- **image(x,y,z)** : représente en niveau de couleur une image où z représente l'intensité au point (x,y) (z est une matrice dont le nombre de ligne est la longueur de x et le nombre de colonne celle de y).
- **lines, points** : ajoute une ligne ou des points sur un graphique existant.
- **hist** : histogramme.
- **barplot** : graphique en barre.
- **abline** : représente une ligne en précisant la pente b et l'ordonnée à l'origine a. Une ligne verticale d'abscisse x (v=x) ou horizontale d'ordonnée y (v=y)
- **legend** : ajoute une légende en précisant les symboles (*lty* ou *pch*et *col*), le texte (*text*) et l'emplacement (x=*'topright'*).

Principales fonctions : graphiques

- **axis** : ajoute un axe. Argument : *side* (1: bas, 2: gauche, 3: haut, 4: droite).
- **grid** : ajoute un quadrillage.
- **par(mfrow=c(n,p))** : partage la fenêtre graphique en n x p sous graphiques.

De nombreuses fonctions graphiques disposent des paramètres suivants :

- **type** : *'l'* pour ligne et *'p'* pour points.
- **col** : *'black'*, *'red, 'green'*, *'blue'* . . . (ou 1, 2, 3, 4. . .)
- **lty** : type de lignes (1: solide, 2: pointillée. . .).
- **pch** : type de points (1: cercle, 2: triangle. . .).
- **main** : titre principale.
- **xlab, ylab** : titre des axes.
- **log** : échelle logarithmique (*'x'* pour l'axe des abscisse, *'y'* pour l'axe des ordonnées, *'xy'* pour les deux axes).