Abims DIVCO

27/07/2016

Initiation

Formation 2016

Gildas Le Corguillé – Thomas Broquet

v 1.06

Plan

- Introduction
- Premières additions
- Importation de données tabulées
- Manipulation d'objets plus complexes
- Fonctions mathématiques
- Programmation
- Fonctions propres à R pour la manipulation de tableaux
- Les graphiques
- Exportation
- Un cas d'école

INTRODUCTION À L'ENVIRONNEMENT R

R, un outil utile en biologie ?

- R : pour quoi faire ?
  - traitement de données, analyses statistiques, graphs
  - très efficace pour traiter en routine les jeux de données les plus conséquents (par exemple, automatiser formatage des données, analyses et graphs en un seul script)
  - création de données (modélisation, simulation... par ex : simuler les résulats attendus sous des hypothèses alternatives)
  - toujours à la pointe en traitement statistique (ajouts permanents)
- R : pour *ne pas faire* quoi ?
  - ce que l'on sait faire mieux et plus vite ailleurs (e.g. Excel)
  - algèbre symbolique (e.g. Mathematica)
  - ...

Introduction

- R est un logiciel libre, gratuit et multiplateforme (windows, linux et mac) distribué par GNU Public Licence trés utilisé pour l'analyse statistique. La version de base dispose d'un grand nombre d'outils analytiques et graphiques permettant de manipuler, de traiter et de représenter des données de nature très différentes. Son développement met à contribution des utilisateurs qui peuvent créer de nouveaux paquets (« packages ») rendant les possibilités d'utilisation immenses dans des domaines d'études très différents (écologie, analyse sensorielle, psychologie, économie...) et faisant intervenir des techniques très diverses (analyse multivariée, modélisation linéaire et non linéaire, statistique spatiale, classification, tests statistiques...).
- Le partage grandissant de nouveaux paquets rend ce logiciel très dynamique et qui s'enrichit jour après jour.
- R est également un langage de programmation basé sur le calcul matriciel. La manipulation d'objets de type vecteur, liste, matrice permet une flexibilité de programmation d'algorithmes plus ou moins évolués répondant aux attentes de chacun.
- Parmi les paquets, il en existe permettant d'interfacer avec d'autres outils tels que PostgreSQL et MySQL pour les bases de données, le logiciel libre GRASS pour le SIG, RExcel pour excel ou encore Latex et OpenDocument pour l'exportation de résultats.

- **http://abcdr.2sigma.fr/**

Introduction

- Quelques bons tutoriels

  http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

  http://w3.jouy.inra.fr/unites/miaj/public/formation/initiationRv4.pdf

  http://cran.r-project.org/other-docs.html

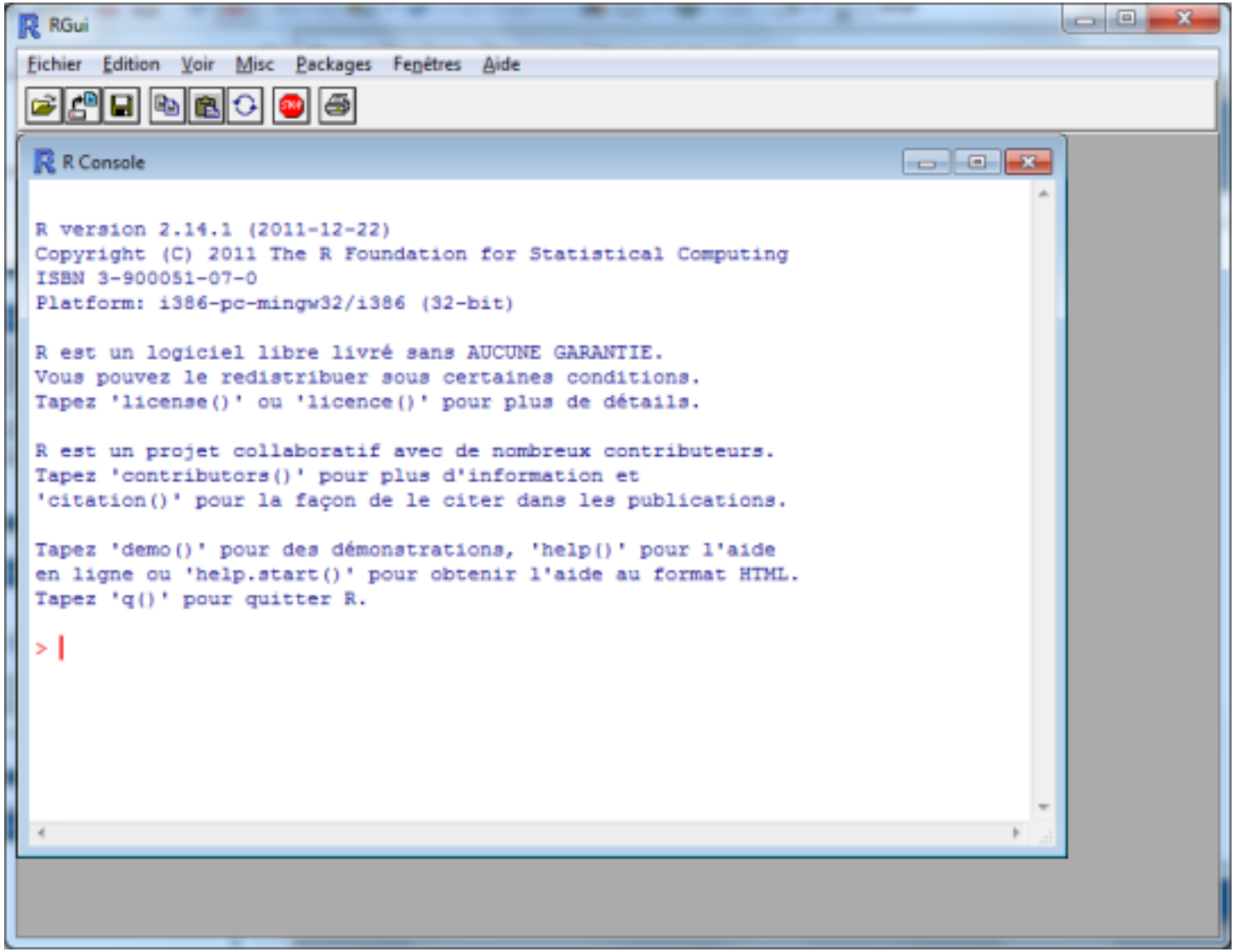
- R Reference Card

  http://cran.r-project.org/doc/contrib/Short-refcard.pdf

  http://cran.r-project.org/doc/contrib/YanchangZhao-refcard-data-mining.pdf

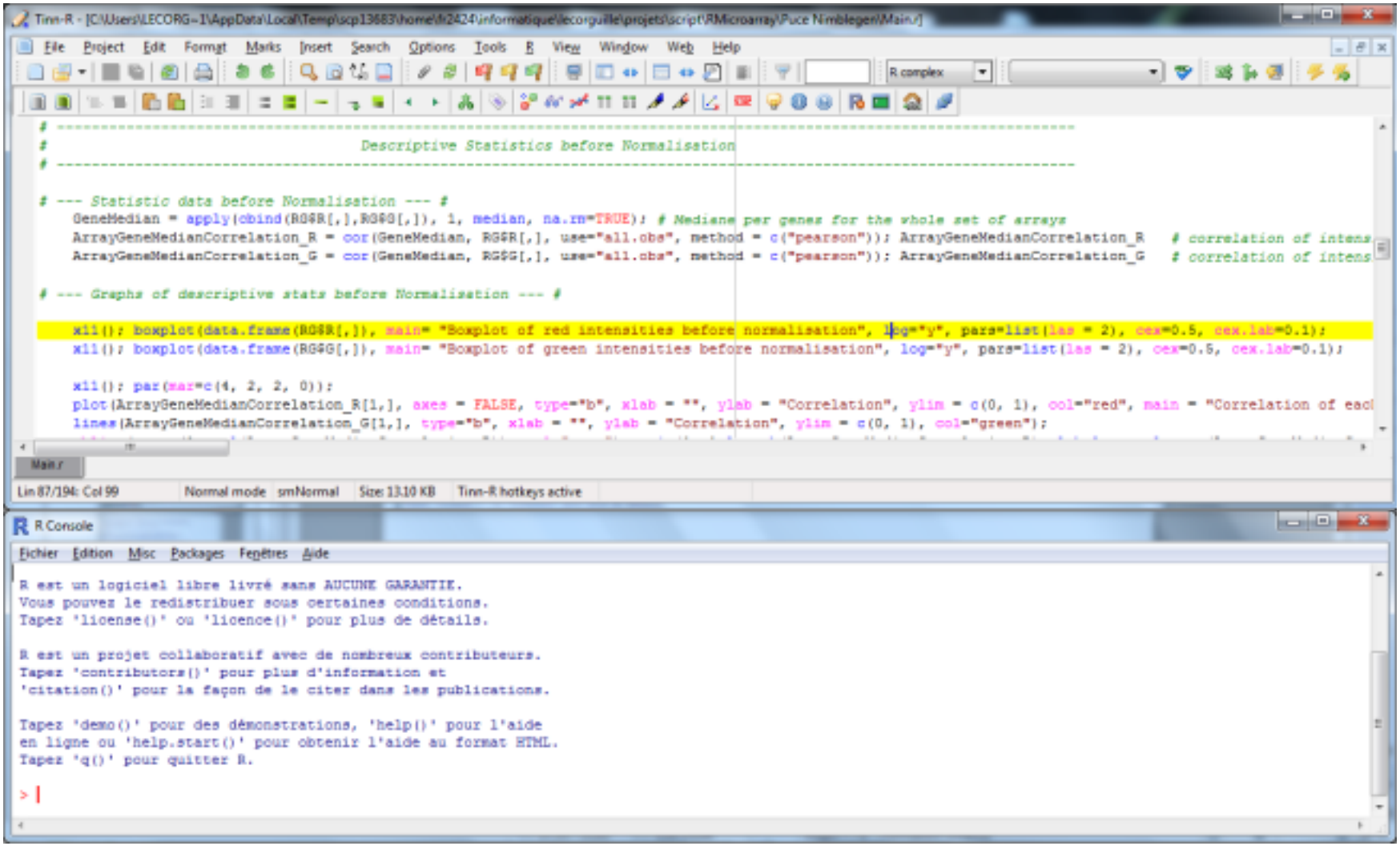
Introduction : IDE Windows

- RGui



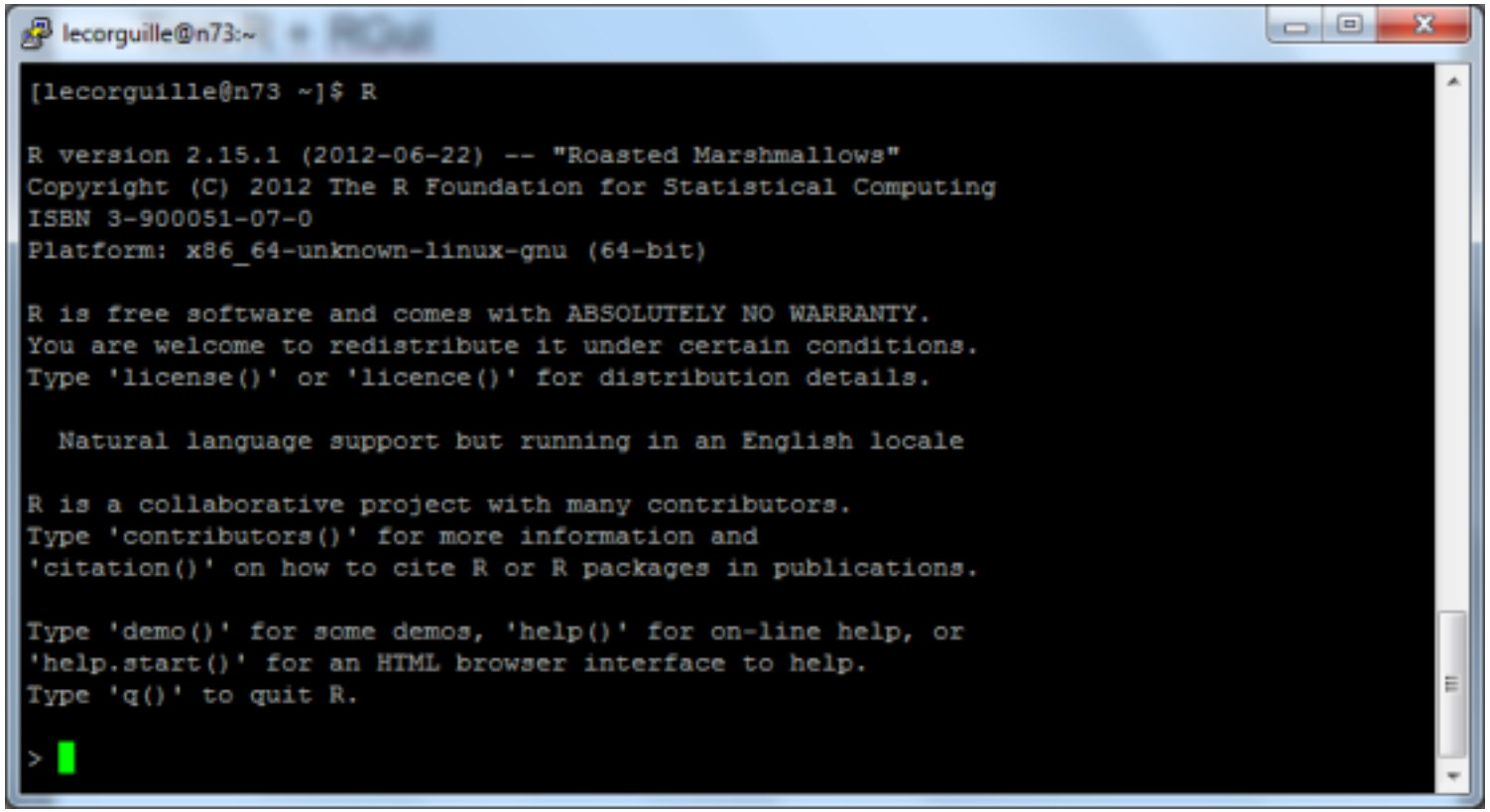
Introduction : IDE Windows

- Tinn-R + RGui



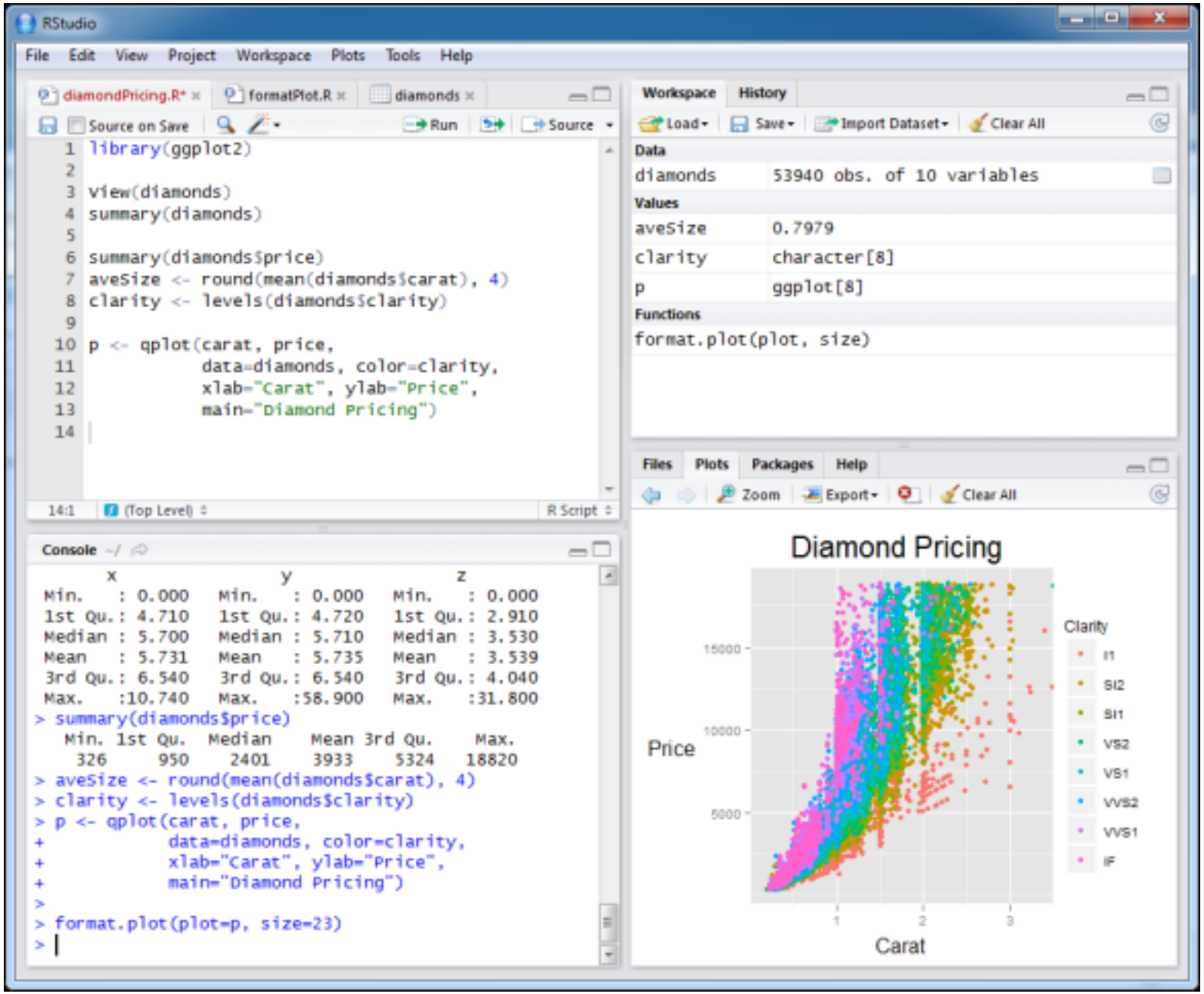
Introduction : IDE Unix

- R



Introduction : IDE en ligne

- RStudio



Introduction : Composants

- Library
  - Plusieurs bibliothèques de fonctions sont installées par défaut : base, graphics, stats, etc. . .
  - De nombreuses bibliothèques plus spécialisées doivent être installées depuis le CRAN ou via le projet Bioconductor (genomic, microarray).
    - CRAN : ~~install.packages("cluster")~~ → support.abims@sb-roscoff.fr
    - Bioconductor : ~~biocLite("affy")~~ → support.abims@sb-roscoff.fr
    - Ces library spécifiques doivent être chargées à chaque session avant leur utilisation

```
> library("cluster")
```

PREMIÈRES ADDITIONS

Premières additions : Hello World !

- Facile

```
> 1
[1] 1

> 1 + 1
[1] 2

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

Premières additions : Objets / variables

- Stockage dans des variables

```
> a = 5

> A <- "Hello World!"

> a
[1] 5

> A
[1] "Hello World!"



> b = a + 1

> b
[1] 6



> ls()
[1] "a" "A" "b"
```

Premières additions : Fonctions

- Les fonctions

  Toujours : `nom_de_la_fonction()`

- Les arguments de fonction
  - Parfois : `fct(arg1,arg2,arg3)`
  - Parfois : `fct(nom_arg1=arg1,nom_arg2=arg2)`
  - Parfois : `fct(arg1,nom_arg2=arg2)`

Bienvenue dans un monde du libre

Premières additions : Fonctions - Help

- help(mean)
- ?mean

Google

- R mean (649,000,000 results)
- GNU R mean (554,000 results)

R-Mean MySpace

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects, and for data frames all of whose columns have a meth allowed for trim = 0, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (in NA_real_ is returned, with a warning.

If trim is non-zero, a symmetrically trimmed mean is computed with a fraction of trim observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

weighted.mean, mean.POSIXct, colMeans for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Premières additions : Help

- pour résumer : comment trouver de l'aide ?
  - si on connait le nom de la fonction: help(fonction) ou utiliser l'aide dans R-studio
  - sinon, pour les bases : utiliser les tutoriaux et les listes de fonctions
  - pour aller plus loin : les livres (en particulier : R book)
  - le cas particulier qui m'intéresse n'est jamais exactement celui qui est décrit dans le bouquin... : écumer les forums de discussion sur internet.

IMPORTATION DE DONNÉES TABULÉES

Importation : La navigation

- La navigation
  - Le répertoire de travail est par défaut celui dans lequel vous lancez R
  - Il peut être changé de 2 manières :
    - Click-bouton via les menus (RGui ou Rstudio)
    - Fonction (plus efficace)

```
setwd("/projet/login/data/R")
```

```
setwd("../data/R")
```

Attention : à la syntaxe des chemins : "monrepertoire/monfichier"

pour windows : \ → \\ ou /

pour linux et mac: /

Importation : Fichier tabulé

- Qu'est-ce-qu'un fichier tabulé ?
  - Autorisé : .csv, .tab, .txt, .blast, … → format `ASCII text`
  - Pas autorisé : .xls, .xlsx, .ods

```
$ file data.tab
      data.tab: ASCII text
```

paternity.csv

```
ID;Mother;Father
1;S1A2;S1A3
2;S1A2;S1A3
3;S1A2;S1A3
4;S1A2;S1A3
5;S1A2;S1A3
6;S1A2;S1A3
7;S1A2;S1A3
8;S1A2;S1A3
9;S1A2;S1A3
10;S1A2;S1A3
```

paternity.tab

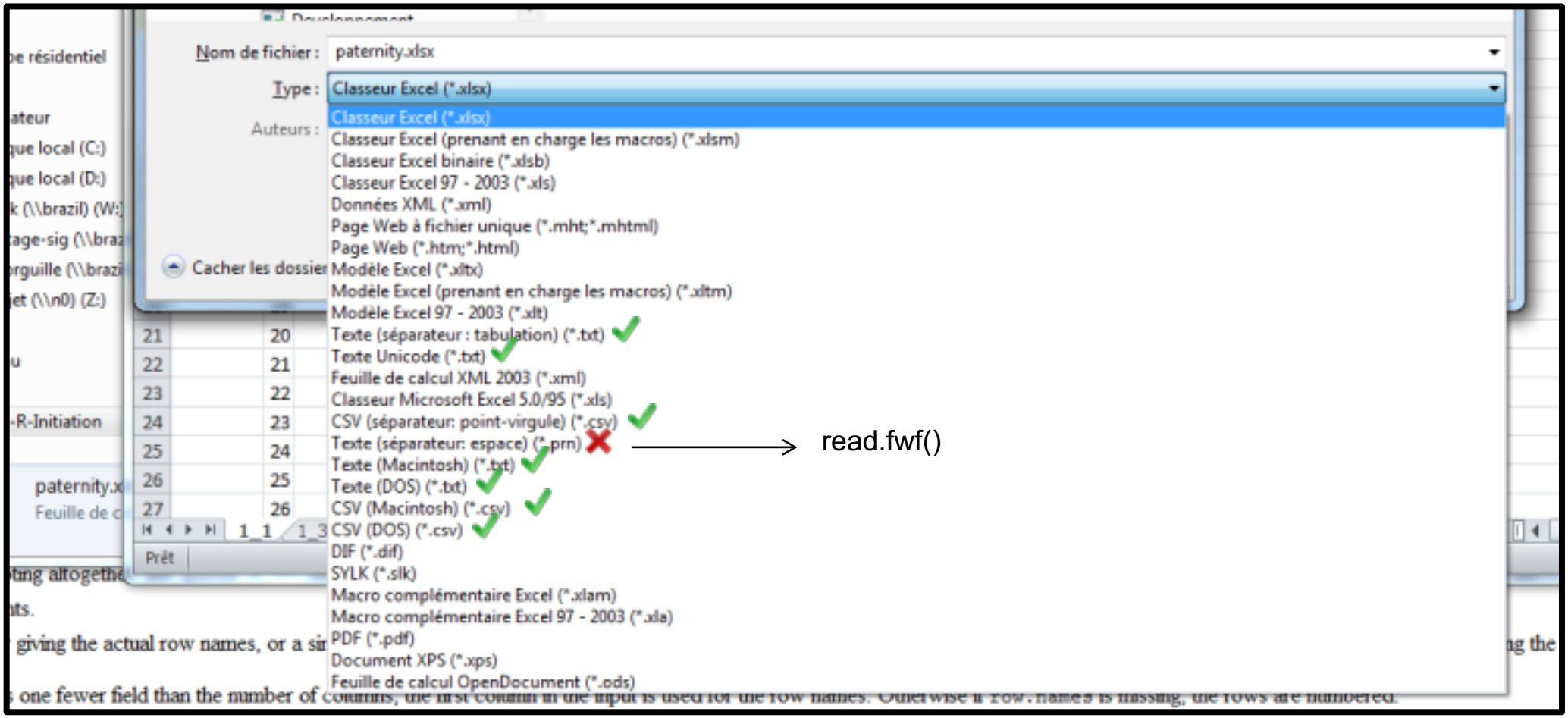
```
ID      Mother Father
1       S1A2   S1A3
2       S1A2   S1A3
3       S1A2   S1A3
4       S1A2   S1A3
5       S1A2   S1A3
6       S1A2   S1A3
7       S1A2   S1A3
8       S1A2   S1A3
9       S1A2   S1A3
10      S1A2   S1A3
```

paternity.txt

```
"ID"    "Mother"  "Fa
1       "S1A2" "S1A3"
2       "S1A2" "S1A3"
3       "S1A2" "S1A3"
4       "S1A2" "S1A3"
5       "S1A2" "S1A3"
6       "S1A2" "S1A3"
7       "S1A2" "S1A3"
8       "S1A2" "S1A3"
9       "S1A2" "S1A3"
10      "S1A2" "S1A3"
```

Importation : Fichier tabulé

- Export depuis Microsoft®©℗™ Office®©℗™ Excel®©℗™



®©℗™

Importation : Fichier tabulé

- Validation quand ça #%$&!!
  - Validation par nodepad++
    - Symbole spéciaux → afficher tous les caractères

ID	Mother	Father
1	S1A2	S1A3
2	S1A2	S1A3
3	S1A2	S1A3
4	S1A2	S1A3
5	S1A2	S1A3
6	S1A2	S1A3
7	S1A2	S1A3
8	S1A2	S1A3
9	S1A2	S1A3
10	S1A2	S1A3

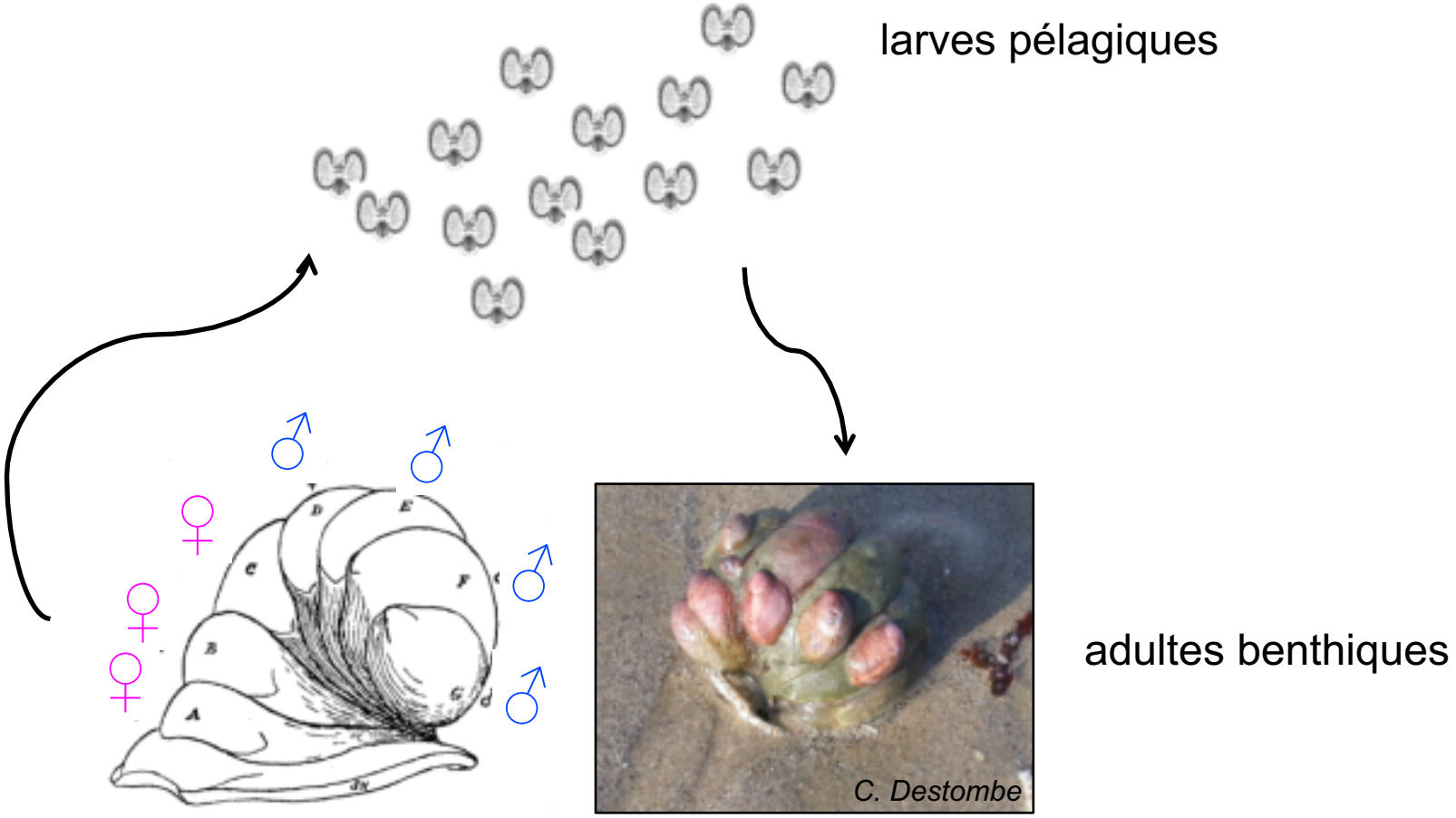
ID→Mother→FatherLF
1→S1A2→S1A3LF
2→S1A2→S1A3LF
3→S1A2→S1A3LF
4→S1A2→S1A3LF
5→S1A2→S1A3LF
6→S1A2 →S1A3 LF
7→S1A2→S1A3LF
8→S1A2→S1A3LF
9→S1A2→S1A3LF
10→S1A2 →S1A3LF

Excercice

IMPORTATION DE DONNÉES TABULÉES

Importation : Exemple de données

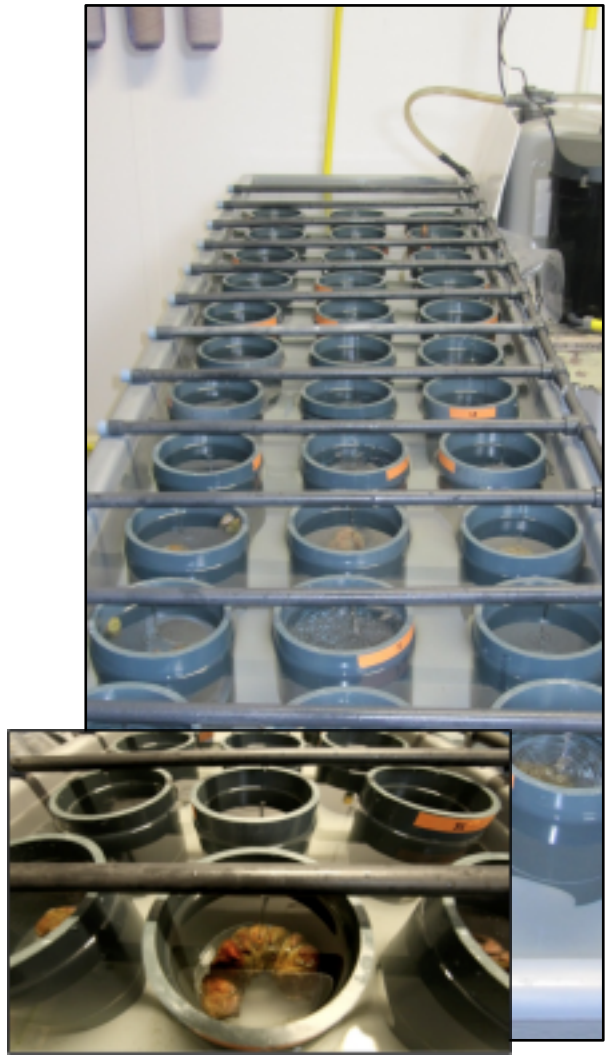
- Etude du succès reproducteur chez la crépidule



Importation : Exemple de données

- Etude du succès reproducteur chez la crépidule

  1. élevage des adultes
  2. échantillonnage des larves émises par chaque chaîne d'individus
  3. mesure des larves et génotypage
  4. assignation génétique de parenté



→ - système de reproduction
- variance de succès reproducteur
- dérive génétique
- compromis croissance / reproduction

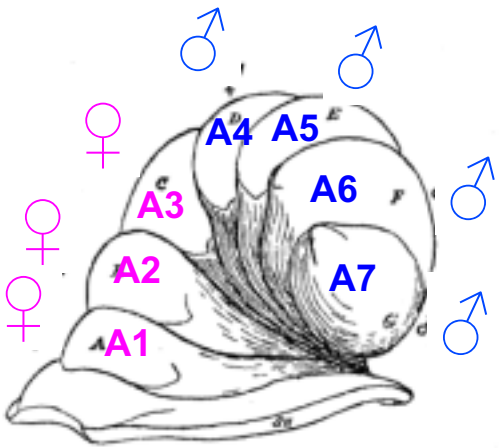
...

Importation : Exemple de données

- Les données : fichier excel “paternity.xlsx”

identifiants des parents : **S**tack ***x*** Adult ***y***

identifiant des larves



Stack 1 (S1)

ID	Mother	Father
1	S1A2	S1A3
2	S1A2	S1A3
3	S1A2	S1A3
4	S1A2	S1A3
5	S1A2	S1A3
6	S1A2	S1A3

...

Rq: parfois pères inconnus : X1, X2

Importation : Exemple de données

- Les données : fichier excel “offspring.xlsx”
  - nombreuses colonnes plus ou moins utiles accumulées au cours du travail ...

stack	clutch	label_clutch	day	ID	label_seq	label_plate	label_manue	plate	size	exp
1	1	1_1	0	1	AEB_P1-1	ind02	indiv01	P	404	Audrey
1	1	1_1	0	2	AEB_P1-1	ind03	indiv02	P	444	Audrey
1	1	1_1	0	3	AEB_P1-1	ind04	indiv03	P	424	Audrey
1	1	1_1	0	4	AEB_P1-1	ind05	indiv04	P	404	Audrey
1	1	1_1	0	5	AEB_P1-1	ind06	indiv05	P	404	Audrey
1	1	1_1	0	6	AEB_P1-1	ind07	indiv06	P	444	Audrey
1	1	1_1	0	7	AEB_P1-1	ind08	indiv07	P	384	Audrey
1	1	1_1	0	8	AEB_P1-1	ind09	indiv08	P	424	Audrey
1	1	1_1	0	9	AEB_P1-1	ind10	indiv09	P	424	Audrey
1	1	1_1	0	10	AEB_P1-1	ind11	indiv10	P	384	Audrey
1	1	1_1	0	11	AEB_P1-1	ind12	indiv11	P	444	Audrey

...

identifiant des larves

Exercice

- Etape 1 : formater les données sous Excel
  - attention aux espaces, aux accents, caractères spéciaux, etc.
  - enregistrer les différentes feuilles des classeurs “paternity.xlsx” et “offspring.xlsx” sous un format tabulé

Exercice

- Etape 2 : commencer un script sous R-studio

```
# A script to analyse Crepidula paternity data
# Cours initiation à R - juin. 2016

# working directory
setwd("/projet/unity/team/login/projet_crepidula/tmp/")
```

→ Ctrl + Enter pour envoyer les commandes à R (ligne active ou sélection)

→ Enregistrer ce script (e.g. “myscript.r”) pour réutilisation ultérieure

Exercice

- Etape 3 : création de tableaux de données
  - la fonction d'import sous R : `read.table()`

```
> ?read.table

> paternity = read.table("paternity.csv", header=TRUE, sep=";", quote="")

> paternity = read.table("paternity.tab", header=TRUE, sep="\t", quote="")

> paternity = read.table("paternity.txt", header=TRUE, sep="\t", quote="\"")

> head(paternity)
ID  Mother Father
1   S1A2   S1A3
2   S1A2   S1A3
3   S1A2   S1A3
4   S1A2   S1A3
5   S1A2   S1A3
6   S1A2   S1A3
> View(paternity)# commande RStudio pour visualiser les données (sinon cliquer)
```

paternity.csv

```
ID;Mother;Father
1;S1A2;S1A3
2;S1A2;S1A3
3;S1A2;S1A3
```

paternity.tab

```
ID      Mother Father
1       S1A2   S1A3
2       S1A2   S1A3
3       S1A2   S1A3
```

paternity.txt

```
"ID"    "Mother"   "Fa
1       "S1A2" "S1A3"
2       "S1A2" "S1A3"
3       "S1A2" "S1A3"
```

Exercice

- Etape 3 : création de tableaux de données
  - la fonction d'import sous R : `read.table()`

```
# A script to analyse Crepidula paternity data
# Cours initiation à R - sept. 2012

# working directory
setwd("/projet/.../.../login/projet_crepidula/tmp/")

#### import raw data
# offspring
offspring <- read.table("offspring.txt", header=T, sep="\t")

# parents: 3 files
parents1 <- read.table("stack1.txt",header=T,sep="\t")
parents2 <- read.table("stack2.txt",header=T,sep="\t")
parents25 <- read.table("stack25.txt",header=T,sep="\t")
```

Importation : Les retours d'erreur

- Quand R essaye de communiquer : les messages d'erreur

```
> setwd("/projet/.../.../login/projet_crepidule/tmp/")
 Error in setwd("/projet/.../.../login/projet_crepidule/tmp/") :
 cannot change working directory
```

```
> offspring <- read.table("offspring.txt",header=T,sep="\t")
 Error in make.names(col.names, unique = TRUE) :
  invalid multibyte string 11
```

Importation : read....

- Autre fonction d'import sous R : scan ( )

```
> ?scan
```

OBJETS PLUS COMPLEXES

Objets plus complexes : Mode et Class

- Les modes
  - numerique : 1 -1 0.5 2.1e23
  - caractère : "a" "Hello World" 'toto' "cas \"pas simple\""
  - logique : TRUE / FALSE

```
> 1 == 1
[1] TRUE
```

  - complexe :

```
> complex(real = 0, imaginary = 1)
[1] 0+1i
```

- Pour obtenir le mode : mode()

```
> mode(a)
[1] "numeric"
> mode(A)
[1] "character"
> mode(1==1)
[1] "logical"
```

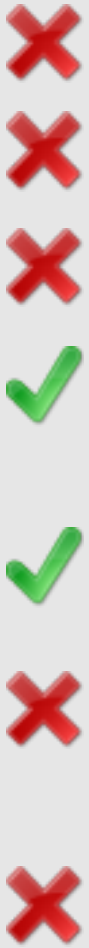
Objets plus complexes : Mode et Class

- Les classes (ou types)
  - vecteur (`vector`) : suite d'éléments une dimension
  - matrice (`matrix`) : tableau à 2 dimensions
  - tableau (`array`) : tableau à k dimensions
  - tableau de données (`data.frame`) :

    ensemble de vecteurs de même longueur
  - liste (`list`) : liste d'objets
  - facteur (`factor`)

    suite d'éléments catégoriels avec niveaux
  - séries temporelles (`ts`) : fréquence, dates …

Plusieurs modes autorisés dans le même objet ?



Objets plus complexes : vector

- vecteur (`vector`) : suite d'éléments une dimension
  - 1 mode autorisé dans un même objet

```
> myvector = c(1,3,5,7,9)
> myvector
 [1] 1 3 5 7 9

> length(myvector)
 [1] 5

> myvector2 = c(myvector,2,4,6,8)
> myvector2
 [1] 1 3 5 7 9 2 4 6 8

> myvector2 - 1
 [1] 0 2 4 6 8 1 3 5 7



> myvector2 - myvector
 [1] 0 0 0 0 0 1 1 1 1
Message d'avis :
In y - x :
  la taille d'un objet plus long n'est pas multiple de la taille d'un objet plus court
```

Objets plus complexes : vector

- Générateur de vecteurs

```
> 1:25
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> seq(1,10,by=0.5)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5
[19] 10.0

> seq(1,25,2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25

> rep(1,25)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> sequence(4:5)
 [1] 1 2 3 4 1 2 3 4 5


> rnorm(25, mean=0, sd=1)
 [1]  1.861223234  0.179097312  0.193908771  0.447167513  0.490801348  1.718215675
 [7] -0.008796725  1.808822710  0.055496719 -0.761208517 -0.504335108 -0.824040103
[13] -1.471801732  2.003031214  1.313190462  1.505939057  1.098118465 -0.695082609
[19] -0.338180962 -0.116440754 -1.242136683 -1.048167354  1.492289109  1.432021299
[25]  1.494668508

> # quelques autres : rexp, rgamma, rpois, ...
```

Objets plus complexes : vector

- Accéder aux valeurs d'un vecteurs

```
> myvector = 1:25
> myvector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> myvector[8]
 [1] 8

> myvector[-8]
 [1]  1  2  3  4  5  6  7  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> myvector[2:4]
 [1] 2 3 4

> myvector[seq(1,25,by=2)]
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25

> myvector[10] = 100

> myvector > 10
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE

> myvector[myvector > 10]
 [1] 100 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Contrairement aux autres langages de développement, les indices de tableaux commence à 1

Objets plus complexes : matrix

- matrice (`matrix`) : tableau à 2 dimensions
  - 1 mode autorisé dans un même objet

```
> matrix(data=1:6, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6,2,3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

> cbind(c(1,2),c(3,4),c(5,6))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Objets plus complexes : matrix

- matrice (`matrix`) : tableau à 2 dimensions
  - 1 mode autorisé dans un même objet

```
> dim(mymatrix)
[1] 2 3

> ncol(mymatrix)
[1] 3

> nrow(mymatrix)
[1] 2
```

```
> mymatrix = 1:12                    # nous avons pour le moment un vector

> mymatrix
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

> dim(mymatrix)
NULL

> dim(mymatrix) = c(4,3)             # ici, nous forcons pour creer une matrix

> mymatrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Objets plus complexes : matrix

- Accéder aux valeurs d'une matrice

```
> mymatrix = matrix(data=1:12, nrow=4, ncol=3)
> mymatrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> mymatrix[3,2]
[1]  7

> mymatrix[2,]
 [1] 2 6 10

> mymatrix[,2]
 [1] 5 6 7 8

> mymatrix[3:4,1:2]
     [,1] [,2]
[1,]    3    7
[2,]    4    8
```

Objets plus complexes : matrix

- Accéder aux valeurs d'une matrice

```
> mymatrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> dimnames(mymatrix)
NULL

> colnames(mymatrix) = c("condition1","condition2","condition3")
> rownames(mymatrix) = c("sample1","sample2","sample3","sample4")
> dimnames(mymatrix)
[[1]]
[1] "sample1" "sample2" "sample3" "sample4"

[[2]]
[1] "condition1" "condition2" "condition3"

> mymatrix
        condition1 condition2 condition3
sample1          1          5          9
sample2          2          6         10
sample3          3          7         11
sample4          4          8         12

> mymatrix["sample2","condition3"]
[1] 10
```

Objets plus complexes : matrix

- Accéder aux valeurs d'une matrice

```
> mymatrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> mymatrix[mymatrix >= 10]
[1]  10 11 12

> mymatrix[mymatrix >= 10] = NA
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   NA
[3,]    3    7   NA
[4,]    4    8   NA
```

Objets plus complexes : array

- tableau (`array`) : tableau à k dimensions
  - 1 mode autorisé dans un même objet

```
> myarray = array(data=c(11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,35,36), dim=c(3,2,3))
> myarray
, , 1

     [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16

, , 2

     [,1] [,2]
[1,]   21   24
[2,]   22   25
[3,]   23   26

, , 3

     [,1] [,2]
[1,]   31   34
[2,]   32   35
[3,]   33   36

> myarray[1,2,3]
[1] 34
```

Objets plus complexes : data.frame

- tableau de données (`data.frame`) : ensemble de vecteurs de même longueur
  - plusieurs modes autorisés dans un même objet

```
> array1 = c(-1.5585350,-0.5669521,-0.7483982,-0.5685524,0.5566560,-0.3465147)
> array2 = c(-0.7348447,-1.1037727,-0.3216453,0.1248720,-0.6403694,-1.0225939)
> gene = c("typ1","droML","furA","sufB","rpo42","rrr")

> mydataframe = data.frame(array1,array2,gene)
> mydataframe
     array1     array2  gene
1 -1.5585350 -0.7348447  typ1
2 -0.5669521 -1.1037727 droML
3 -0.7483982 -0.3216453  furA
4 -0.5685524  0.1248720  sufB
...

> mydataframe = cbind(mydataframe, core=c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE))
> mydataframe
     array1     array2  gene  core
1 -1.5585350 -0.7348447  typ1  TRUE
2 -0.5669521 -1.1037727 droML  TRUE
3 -0.7483982 -0.3216453  furA FALSE
4 -0.5685524  0.1248720  sufB FALSE
...

> mydataframe = cbind(mydataframe, rank=c(2,1,4,3,5))
Erreur dans data.frame(..., check.names = FALSE) :
  les arguments impliquent des nombres de lignes différents : 6, 5
```

Objets plus complexes : data.frame

- tableau de données (`data.frame`) : ensemble de vecteurs de même longueur

```
> mydataframe
      array1     array2  gene  core
1 -1.5585350 -0.7348447  typ1  TRUE
2 -0.5669521 -1.1037727 droML  TRUE
3 -0.7483982 -0.3216453  furA FALSE
4 -0.5685524  0.1248720  sufB FALSE
5  0.5566560 -0.6403694 rpo42  TRUE
6 -0.3465147 -1.0225939   rrr FALSE

> mydataframe[4,2]
[1] 0.124872

> mydataframe[4,"array2"]
[1] 0.124872

> mydataframe$array2[4]
[1] 0.124872

> mycolumn = "array2"
> mydataframe[[mycolumn]][4]
[1] 0.124872

> mydataframe["array2"][4]
Erreur dans `[.data.frame`(mydataframe["array2"], 4) : undefined columns selected
> mydataframe["array2"][4,]
[1] 0.124872
```

Exercice

- Retour au jeu de données Crepidula
  - fusionner les 3 data.frames contenant les parents

```
# parents: 3 files
parents1 <- read.table("stack1.txt",header=T,sep="\t")
parents2 <- read.table("stack2.txt",header=T,sep="\t")
parents25 <- read.table("stack25.txt",header=T,sep="\t")

# group all parentage data wihin one single dataframe
parents = rbind(parents1,parents2,parents25)
```

Exercice

- Retour au jeu de données Crepidula
  - sélectionner les champs utiles du tableau offspring

```
# sélectionne certaines données du tableau offspring :
# uniquement les champs ID, day, size
temp <- subset(offspring, select=c(stack,ID, size))
```

  - former un seul tableau contenant toutes les informations (e.g. taille des larves et identité des parents)

```
# combine les jeux de données en se basant sur la seule colonne en
# commun aux deux tableaux : ID
data <- merge(temp,parents)
```

Exercice

- Retour au jeu de données Crepidula
  - vérifications

```
> dim(offspring)
[1] 1221   11

> dim(parents)
[1] 959   3

> dim(data)
[1] 959   5
```

La fonction merge() n'a conservé que les individus (ID) présents à la fois dans le tableau offspring et le tableau parents

```
> head(data)
   ID stack size Mother Father
1   1      1 404.0   S1A2   S1A3
2   2      1 444.4   S1A2   S1A3
3   3      1 424.2   S1A2   S1A3
4   4      1 404.0   S1A2   S1A3
5   5      1 404.0   S1A2   S1A3
6   6      1 444.4   S1A2   S1A3

> data$ID
```

Objets plus complexes : list

- liste (`list`) : liste d'objets
  - plusieurs objets autorisés dans un même objet

```
> mylist = list(a=a,A=A,myvector=myvector,mymatrix=mymatrix,myarray=myarray)
> mylist
$a
[1] 5

$A
[1] "Hello World!"

$myvector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

$mymatrix
        condition1 condition2 condition3
sample1          1          5          9
sample2          2          6         10
sample3          3          7         11
sample4          4          8         12

$myarray
, , 1

     [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16

, , 2

     [,1] [,2]
[1,]   21   24
...

> mylist$mydataframe = mydataframe
```

Objets plus complexes : list

- liste (`list`) : liste d'objets

```
> mylist
$a
[1] 5

$A
[1] "Hello World!"

$myvector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

$mymatrix
        condition1 condition2 condition3
sample1          1          5          9
sample2          2          6         10
...

> names(mylist)
[1] "a"           "A"           "myvector"    "mymatrix"    "myarray"     "mydataframe"

> mylist[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> mylist[["myvector"]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> mylist[3]
$myvector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Objets plus complexes : factor

- facteur (`factor`) : suite d'éléments catégoriels avec niveaux
  - 1 mode autorisé dans un même objet

```
> dataVector = 1:10
> dataVector
 [1]  1  2  3  4  5  6  7  8  9 10


> sampleVector = sample(dataVector, 10,
replace = TRUE)
> sampleVector
 [1]  5  6  5  4  9  8 10  1 10  3


> table(sampleVector)
sampleVector
 1  3  4  5  6  8  9 10
 1  1  1  2  1  1  1  2


> sampleVector = factor(sampleVector,
levels = 1:10)
```

```
> dataFactor = as.factor(1:10)
> dataFactor
 [1] 1  2  3  4  5  6  7  8  9  10
Levels: 1 2 3 4 5 6 7 8 9 10

> sampleFactor = sample(dataFactor, 10,
replace = TRUE)
> sampleFactor
 [1] 5  6  5  4  9  8  10 1  10 3
Levels: 1 2 3 4 5 6 7 8 9 10

> table(sampleFactor)
sampleFactor
 1  2  3  4  5  6  7  8  9 10
 1  0  1  1  2  1  0  1  1  2
```

Objets plus complexes : factor – ARGG !!!

- facteur (`factor`) : suite d'éléments catégoriels avec niveaux

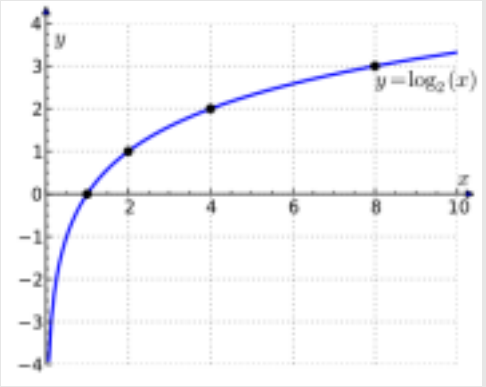
```
> mydataframe$gene
[1] typ1  droML furA  sufB  rpo42 rrr
Levels: droML furA rpo42 rrr sufB typ1

> class(mydataframe$gene)
[1] "factor"

> c(mydataframe$gene,"yo")
[1] "6"  "1"  "2"  "5"  "3"  "4"  "yo"

> c(as.vector(mydataframe$gene),"yo")
[1] "typ1"  "droML" "furA"  "sufB"  "rpo42" "rrr"   "yo"
```

```
> options(stringsAsFactors = FALSE)

> mydataframe$gene = c("typ1","droML","furA","sufB","rpo42","rrr")
> mydataframe$gene
[1] "typ1"  "droML" "furA"  "sufB"  "rpo42" "rrr"

> class(mydataframe$gene)
[1] "character"

> c(mydataframe$gene,"yo")
[1] "typ1"  "droML" "furA"  "sufB"  "rpo42" "rrr"   "yo"
```

Objets plus complexes : Valeur manquantes

- Les valeurs manquantes :

```
> c(1.2604103,1.4802003,NA,1.6124621,1.6718345,0.5794688,-1.7442904)
[1]  1.2604103  1.4802003         NA  1.6124621  1.6718345  0.5794688 -1.7442904

> log(0)
[1] -Inf

> log(1e-1000) - log(1e-1000)
[1] NaN

> mydataframe$annotation
NULL
```



- Les fonctions de test (return TRUE/FALSE)
  - Pour NA → is.na()
  - Pour Inf et -Inf → is.infinite() (à l'opposé : is.finite())
  - Pour NaN → is.nan()
  - Pour NULL → is.null

FONCTIONS MATHÉMATIQUES

Fonctions mathématiques

- Les opérateurs arithmétiques
  - + - * / : les classiques
  - ^ : puissance
  - %% : modulo
  - %/% : division entière

Fonctions mathématiques

- Les opérateurs de comparaison
  - == : égalité != : différence
  - > < <= >=

```
> (1 - 0.3) == 0.7
[1] TRUE

> (1.1 - 0.4) == 0.7
[1] FALSE

> all.equal(1.1 - 0.4, 0.7)
[1] TRUE

> isTRUE(all.equal(1.1 - 0.4, 0.7))
[1] TRUE
```

Il faut se méfier des erreurs numériques dues aux calculs numériques sur l'ordinateur (le epsilon machine).

all.equal() compare l'égalité approximative avec un seuil de tolérance dicté par la machine

*

* source http://www-irma.u-strasbg.fr/~salmon/polyananum_salmon.pdf

Analyse Numérique

$$x = \pm 0.a_1 \dots a_N . b^E,$$

$$\begin{cases} -\mathcal{T}u''(x) = f(x) & \forall x \in]0,1[\\ u(0) = 0. \\ u(1) = 0. \end{cases}$$

0.2 Pourquoi un ordinateur fait-il des calculs faux ?

Tout simplement parce qu'il ne connaît qu'un nombre fini de nombres ! Par exemple ceux qui possèdent un nombre fini donné de chiffres non nuls après la virgule, or ce n'est pas le cas de 1/3 ou de √2 qui ont un nombre infini de chiffres non nuls après la virgule.

[...]

$$\underbrace{0.p\cdots 1^{-d}p^d p}_{\text{ou}} \equiv {}_{\text{ou}} q^1 p \sum_{d}^{0=!} = u$$

$$x = x - \frac{f(x)}{\alpha} = g(x), \ \alpha \neq 0$$

$$\frac{|x - Ar(x)|}{|x|} \leq \underbrace{\frac{b}{2} b^{-N}}_{\text{précision machine}}$$

$$x = \sum_{i=-\infty}^{p} d_i b^i \overset{\text{noté}}{=} \overline{d_p d_{p-1} \dots d_0 , d_{-1} \dots d_{-q} \dots}^{b}$$

0.2.3 Erreurs

La première source d'erreur dans les calculs faits par un ordinateur provient donc d'abord des erreurs d'arrondi sur les données. Puis des opérations effectuées sur les réels en virgule flottante. Nous ne nous étendrons pas sur ce sujet (voir TAN licence) mais il faut être conscient que certaines opérations, telles la soustraction de deux réels voisins par exemple, peut être une source d'erreurs non négligeables !

Exemple

Soit x = 0, 124322.104 et y = 0, 123171.104. Le calcul de x − y sur une machine à 4 chiffres significatifs donne Ar(x)−Ar(y) = 0, 1243.104 −0, 1231.104 = 0, 11.102, il ne reste alors plus que deux chiffres significatifs ! (c'est ce que l'on appelle l'extinction de chiffres).

Fonctions mathématiques

- Les opérateurs logiques
  - !x : NON logique
  - x&y : ET logique
  - x|y : OU logique

```
> mylogique1 = c(TRUE,TRUE,FALSE)
> mylogique2 = c(FALSE,TRUE,TRUE)

> !mylogique1
[1] FALSE FALSE  TRUE

> mylogique1&mylogique2
[1] FALSE  TRUE FALSE

> mylogique1|mylogique2
[1] TRUE TRUE TRUE

> mydataframe[mydataframe$array1 < 0 & mydataframe$array2 < 0 & mydataframe$core,]
      array1     array2  gene core
1 -1.5585350 -0.7348447  typ1 TRUE
2 -0.5669521 -1.1037727 droML TRUE
```

Fonctions mathématiques

- Les fonctions simples
  - sum(x), prod(x), min(x), max(x)
  - which.min(x), which.max(x) : indice de la valeur min/max

```
> sum(c(1,3,5))
[1] 9
> prod(c(1,3,5))
[1] 15
> sum(mymatrix)
[1] 78
> prod(mymatrix)
[1] 479001600

> max(sampleVector)
[1] 10
> which(sampleVector == max(sampleVector))
[1] 7 9

> sum(c(1,NA,5))
[1] NA
> sum(c(1,NA,5), na.rm = TRUE)
[1] 6
```

Beaucoup de fonctions ont cette option

Fonctions mathématiques

- Les fonctions simples
  - rev(x)
  - sort(x)
  - choose(n, k) : combinatoire de k élément parmi n
  - sample(x, n) : tirage aléatoire sans remise de n éléments parmi x

```
> sampleVector
 [1]  5  6  5  4  9  8 10  1 10  3
> rev(sampleVector)
 [1]  3 10  1 10  8  9  4  5  6  5
> sort(sampleVector)
 [1]  1  3  4  5  5  6  8  9 10 10
> sort(sampleVector, decreasing=TRUE)
 [1] 10 10  9  8  6  5  5  4  3  1

> choose(100,10)
[1] 1.731031e+13

> sample(1:100, 10)
 [1] 26 74 43 96 97 29 90 85 60 17
```

Fonctions mathématiques

- Les fonctions "statistiques"
  - mean(x) → moyenne, median(x) → médiane
  - var(x) → variance
  - cor(x,y) → corrélation
  - cov(x,y) → covariance

$$\sigma_{XY} = \operatorname{cov}(X,Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} x_i y_j \mathbb{P}(X = x_i \text{ et } Y = y_j) - \mathbb{E}[X]\mathbb{E}[Y].$$

```
> sampleVector
 [1]  5  6  5  4  9  8 10  1 10  3

> mean(sampleVector)
[1] 6.1
> median(sampleVector)
[1] 5.5
> var(sampleVector)
[1] 9.433333

> cor(sampleVector, sort(sampleVector))
[1] 0.03415783
> cor(sort(sampleVector), sort(sampleVector, decreasing=TRUE))
[1] -0.9434629

> cov(sampleVector, sort(sampleVector))
[1] 0.3222222
> cov(sort(sampleVector), sort(sampleVector, decreasing=TRUE))
[1] -8.9
```

Fonctions mathématiques

- Les fonctions matricielles
  - x%*%y : produit de deux matrices
  - t(x) : transposée de x
  - diag(x) : extrait la diagonale d'une matrice x

```
> mymatrix = matrix(1:9, 3,3)
> mymatrix
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> mymatrix %*% mymatrix
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
> diag(mymatrix)
[1] 1 5 9
> t(mymatrix)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Fonctions mathématiques

- %in%

```
> 12 %in% data$ID
[1] TRUE

> 1324 %in% data$ID
[1] FALSE

> 1324 %in% data
[1] FALSE
```

- apply

```
> tapply(data$size,data$stack,mean)
       1        2       25
429.0630 405.8439       NA


> tapply(data$size,data$stack,mean,na.rm=T)
       1        2       25
429.0630 405.8439 404.6942
```

Exercices

1) Ajouter au jeu de données “data” une colonne “pair” contenant un identifiant par couple de parents (ex: S1A1_S1A2).

→ utiliser la fonction paste()

2) Combien de larves produites par le couple S1A2_S1A3 ont-elles été mesurées ?

→ utiliser la fonction length()

3) Calculer la taille moyenne des larves du couple S1A2_S1A3

4) Obtenir automatiquement ce résultat pour l'ensemble des couples

→ en utilisant la fonction tapply()

→ puis en utilisant la fonction aggregate()

5) Même chose (avec la fonction aggregate), mais en ayant préalablement sous-sélectionné seulement les chaines 1 et 2 (stacks)

→ utiliser la fonction “subset()”

6) Obtenir un tableau contenant le nombre et la taille moyenne des larves pour chaque couple des chaines 1 et 2

→ combiner les fonctions subset(), aggregate(), et tapply()

Exercices – correction

```
# ajouter une colonne "pair"
data$pair <- paste(data$Mother,data$Father,sep="_")


# nombre de larves produites par le couple S1_A2_S1_A3
length(data$size[data$pair=="S1A2_S1A3"])

# moyenne de la taille des larves produites par ce couple
mean(data$size[data$pair=="S1A2_S1A3"])

# ou encore
with(data[data$pair=="S1A2_S1A3",],
     mean(size)
     )


# moyenne pour chaque couple (avec la fonction tapply)
tapply(data$size,data$pair,mean)
```

Exercices – correction

```
# avec la fonction "aggregate"
aggregate(data$size,
          by=list(data$Mother,data$Father),
          FUN=mean)



# même chose en conservant les noms de colonnes "Mother" et "Father"
aggregate(data$size,
          by=list(Mother=data$Mother,Father=data$Father),
          FUN=mean)


# plus compact
aggregate(size~Mother+Father,data,mean)
```

Exercices – correction

```
# mêmes calculs effectués seulement pour certaines chaînes de crépidules

stack_list <- c(1,2)
sub_data <- subset(data,data$stack %in% stack_list)
summary <- aggregate(size~Mother+Father,sub_data,mean)


# Finalement, si on veut ajouter les tailles d'échantillons
summary$n <- tapply(sub_data$size,sub_data$pair,length)
```

PROGRAMMATION

- If / else if / else

```
> up = 1; down = -1
> a = 0.5

> if (a > up) {
+     b="ok";
+ } else if (a < down) {
+     b="bof";
+ } else {
+     b="null";
+ }

> b
[1] "null"
```

Programmation Boucle

- for

```
for(i in values){
  ... do something ...
}
```

```
> for (i in seq(1:5)) { print (i) }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5

> for (num_line in 1:nrow(mymatrix)) {
+       if (num_line %% 10) { … } }

> datas = list()
> for (infile_i in infiles) {
+   datas[[infile_i]] = read.table(infile_i, header=TRUE, check.names=FALSE)
+ }
```

Programmation Boucle

- while

```
while(condition){
  ... do something ...
}
```

```
> j=1
> while (j < 5) { print(j); j=j+1 }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Attention aux boucles infinies

```
while (j < 5) { print (j); }
```

FONCTIONS PROPRE À R POUR LA MANIPULATION DE TABLEAUX

Fonctions pour les tableaux

- apply : apply fonction over array margins

```
> head(mymatrix)
            [,1]        [,2]       [,3]         [,4]        [,5]
[1,]  0.57177836 -0.09623617 -1.2877098 -0.999702657 -1.83084206
[2,]  0.36411961 -0.69660756  0.3213590 -0.065069318 -0.09920370
[3,] -0.69590786  0.20113220  1.4770141  0.198917245 -0.08979680

> dim(mymatrix)
[1] 100   5

> apply(mymatrix, 1, mean)
  [1] -0.728542456 -0.035080387  0.218271782  0.158465563  0.257838960  0.327600
 [16]  0.282106984  0.269398490  0.093404380 -0.230242816  0.653728937 -0.37 [...]
 [91]  0.447504533 -0.905960081 -0.542304096  0.175122985 -0.510900002 -0.074437

> apply(mymatrix, 2, mean)
[1] -0.088757464  0.017911582  0.097586186 -0.008795336 -0.019285733
```

Fonctions pour les tableaux

- lapply → sapply → vapply : Apply a Function over a List or Vector

```
> ?lapply
```

- mapply : Apply a Function to Multiple List or Vector Arguments

```
> ?mapply
```

- tapply : Apply a Function Over a Ragged Array

```
> ?tapply
```

Exemple graph par fichier infiles

Fonctions pour les tableaux

- apply avec une fonction maison

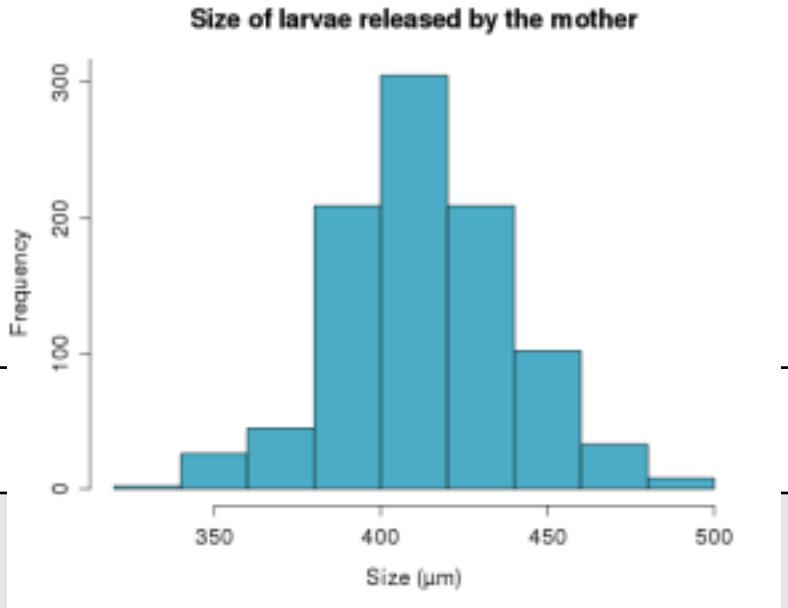
```
> threshold = function(x, up = 0, down = 0) {
+       if (mean(x) > up) {
+               return("ok");
+       } else if (mean(x) < down) {
+               return("bof");
+       } else {
+               return("null");
+       }
+ }

> threshold(1.5)
[1] "ok"
> threshold(1.5, 2, -2)
[1] "null

> apply(mymatrix, 1, threshold, 1, -1)
  [1] "ok"   "null" "null" "null" "null" "null" "null" "null" "null" "null" "null"
 [20] "null" "null" "null" "null" "null" "null" "null" "null" "null" "null" "null"
 [39] "null" "bof"  "null" "null" "null" "null" "null" "null" "null" "null" "null"
 [58] "null" "null" "null" "null" "null" "ok"   "ok"   "null" "null" "null" "null"
 [77] "null" "null" "null" "null" "null" "null" "null" "null" "null" "null" "null"
 [96] "null" "null" "null" "null" "ok"
```

FONCTIONS GRAPHIQUES

Fonctions graphiques

- Quelques exemples : histograms, boxplots, scatterplots

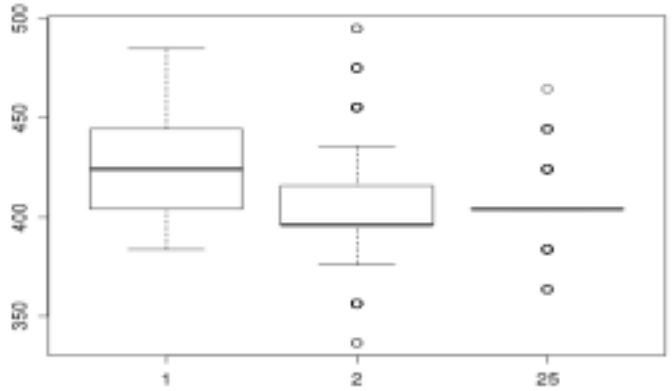
```
### Exploration graphique

#histogramme de fréquences de taille pour toutes les larves
hist(data$size)

# quelques améliorations graphiques
col1 <- rgb(75,172,198,maxColorValue=255)
hist(data$size,
     main="Size of larvae released by the mother",
     yaxp=c(0,300,3),
     xlab="Size (µm)”,
     col=col1,
     )
```



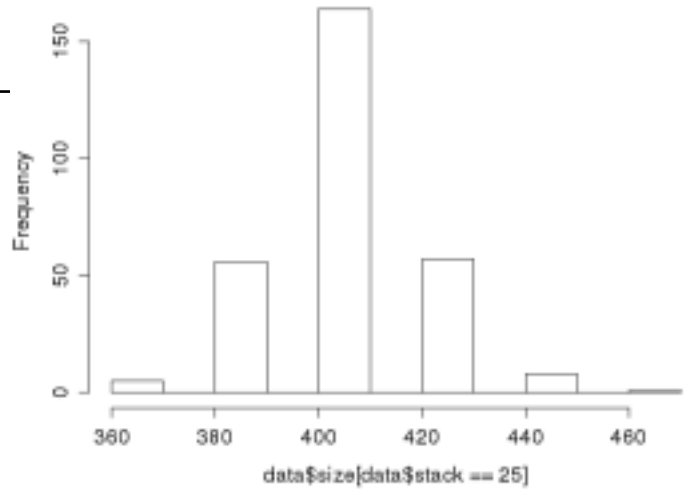

```
> help(par)
```

Fonctions graphiques

```
# Différences de tailles larvaires émises par les différentes mères ?
# Différents pères ? # Différentes chaînes ?
boxplot(size~Mother,data)
boxplot(size~Father,data)
boxplot(size~stack,data)
```

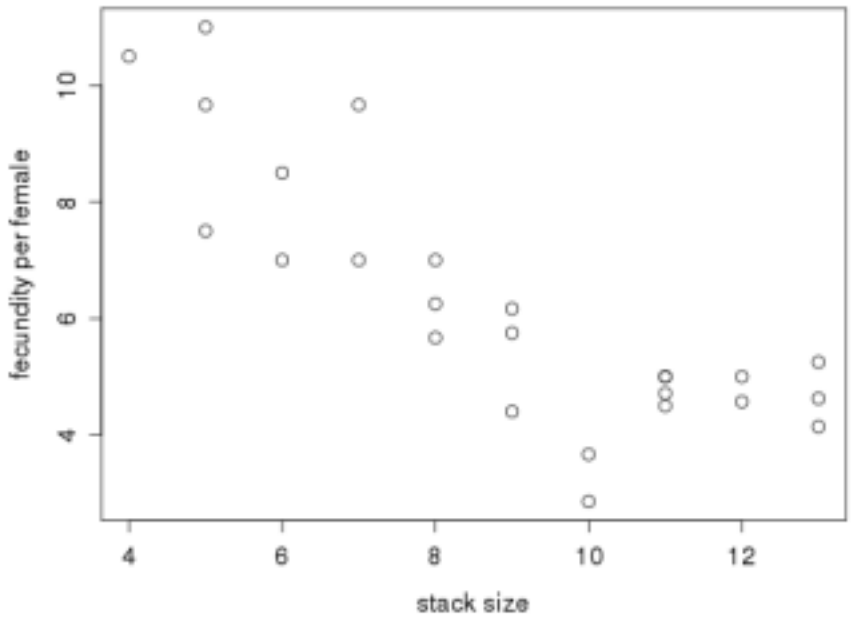



```
# Pour comprendre le boxplot bizarre de la chaine 25
hist(data$size[data$stack==25]))
```



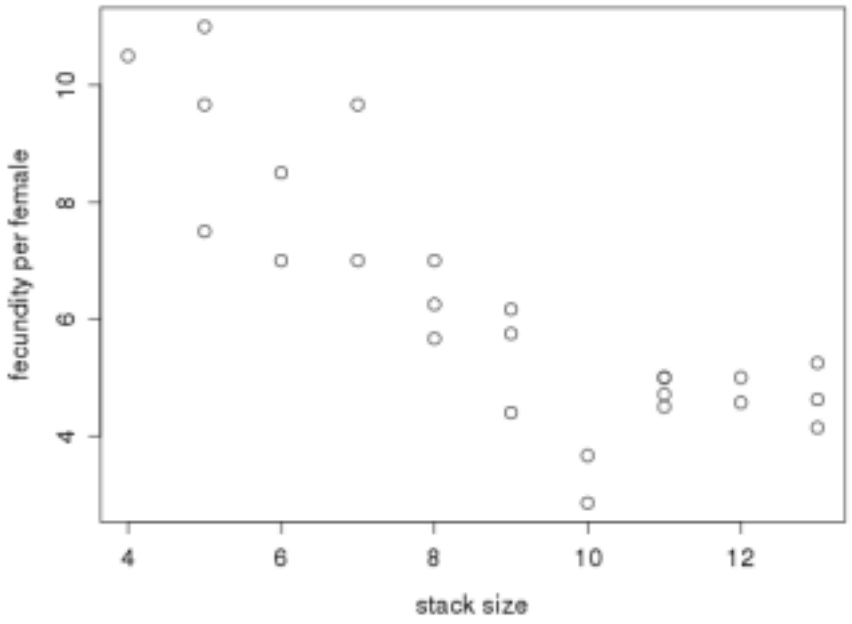
Fonctions graphiques

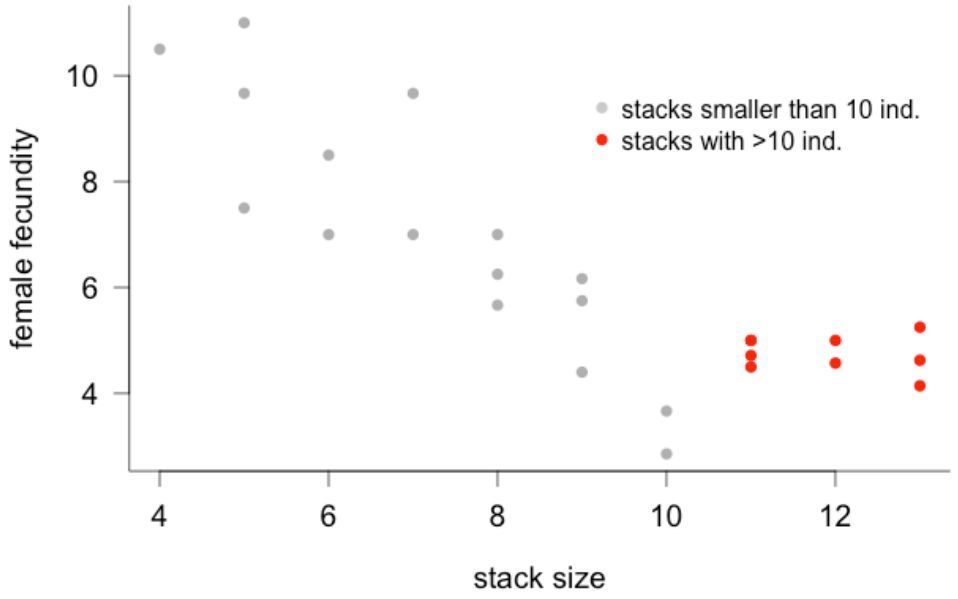
```
# Fécondité des femelles au sein des différentes chaines de crépidules

data2 <- read.table("data/fecundity.txt",sep="\t",header=T)
data2$fecundity <- data2$nclutch/data2$nfemales
plot(fecundity~stack_size,data=data2,
     xlab='stack size',ylab='fecundity per female')
```



Exercice

```
# Fécondité des femelles au sein des différentes chaines de crépidules

data2 <- read.table("data/fecundity.txt",sep="\t",header=T)
data2$fecundity <- data2$nclutch/data2$nfemales
plot(fecundity~stack_size,data=data2,
     xlab='stack size',ylab='fecundity per female')
```





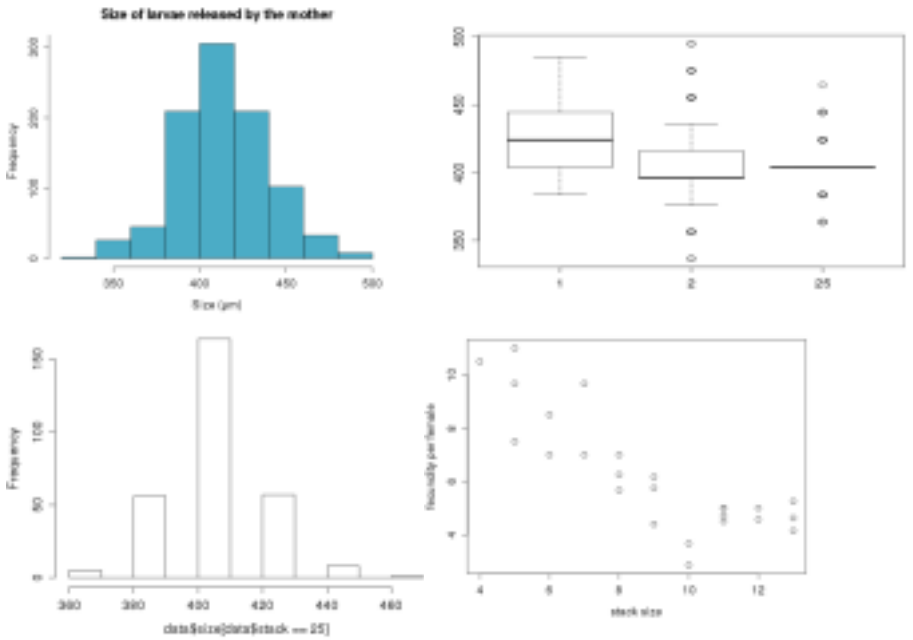
Exercice – Correction

```
# Fécondité des femelles au sein des différentes chaines de crépidules


plot(fecundity~stack_size,data2,
     xlab="stack size",
     ylab="female fecundity",
     pch=16,
     col=grey(0.7),
     cex.lab=1.2,
     cex.axis=1.2,
     bty="l”,
     las=1)

points(fecundity~stack_size,subset(data2,stack_size>10),
       pch=16,
       col="red")

legend(9,10,c("stacks smaller than 10 ind.","stacks with >10 ind."),
       pch=16,col=c(grey(0.8),"red"),
       bty="n")
```

- Juxtaposition de plusieurs graphiques : version simple

```
# c(nr, nc) : number of row x number of column
par(mfrow(c(2,2))

hist(data$size,yaxp=c(0,1000,10),col=col1,main="Size of larvae released […])
boxplot(size~stack,data)
hist(data$size[data$stack==25]))
plot(nclutch/nfemales~stack_size,data=data2,xlab='stack size',ylab='fec […])
```

  - mfcol() procède à un remplissage par colonne

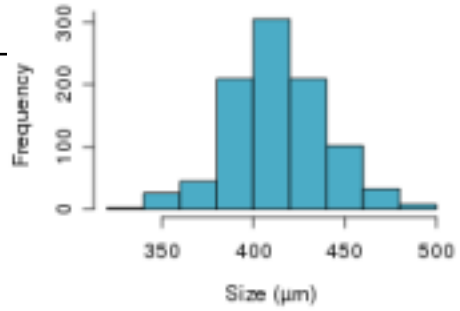


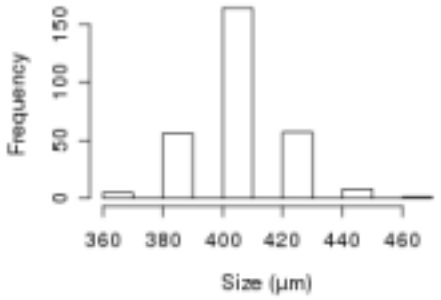
Fonctions graphiques Juxtaposition

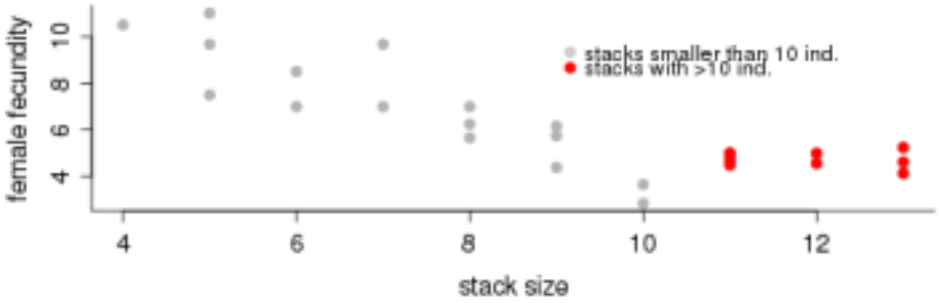
- Juxtaposition de plusieurs graphiques : version étendue

```
# Créer une matrice pour définir les régions du graphique
mat <- matrix(data=c(1,2,3,3),nrow=2,ncol=2,byrow=TRUE)

# Fonction layout
layout(mat)
hist(data$size,[...])
hist(data$size[data$stack==25],[...])
plot(fecundity~stack_size,[...])
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    3
```







EXPORTATION

- Les sauvegardes R
  - RHistory : fichier texte contenant les commandes lancées
  - RData : fichier binaire contenant les objets générés

    – Des sauvegardes sont parfois générées automatiquement à la fin de la session : .RHistory et .RData (Cf cours unix : ls -a)

Attention : le .RData est chargé automatiquement si vous relancez R dans un répertoire qui en contient un.

    – Manuellement :

```
> savehistory("sauvegarde.RHistory")                    # sauver

> source("sauvegarde.RHistory")                         # charger
```

```
> save.image("sauvegarde.RData")                        # sauver toute les objets
> save(objet1, objet2, file="objets.RData")             # sauver 2 de mes objets

> load("sauvegarde.RData"))                             # charger
```

Exportation De fichier tabulé

- Rappel de l'importation d'un fichier tabulé

```
> paternity = read.table("paternity.tab", header=TRUE, sep="\t", quote="")
```

- Exportation vers un fichier tabulé

```
> write.table(paternity, "paternity.tab", header=TRUE, sep="\t", quote=FALSE)
```

Exportation Graphique → pdf

- L'export en pdf se fait en 3 étapes
  - Ouverture d'un fichier pdf
  - Ecriture du (des) graphique(s)
  - Fermeture/Finalisation du fichier pdf

```
pdf(file="graph.pdf")

plot(nclutch/nfemales~stack_size,data2,xlab="stack size",ylab="female […])
points(nclutch/nfemales~stack_size,subset(data2,stack_size>10),pch=16,[…])
legend(9,10,c("stacks smaller than 10 ind.","stacks with >10 ind."),pc[…])

dev.off()
```

  - l'option onefile de pdf() créera une page par plot

Exportation Graphique → png

- L'export en png se fait en 3 étapes
  - Ouverture d'un fichier png
  - Ecriture du graphique
  - Fermeture/Finalisation du fichier png

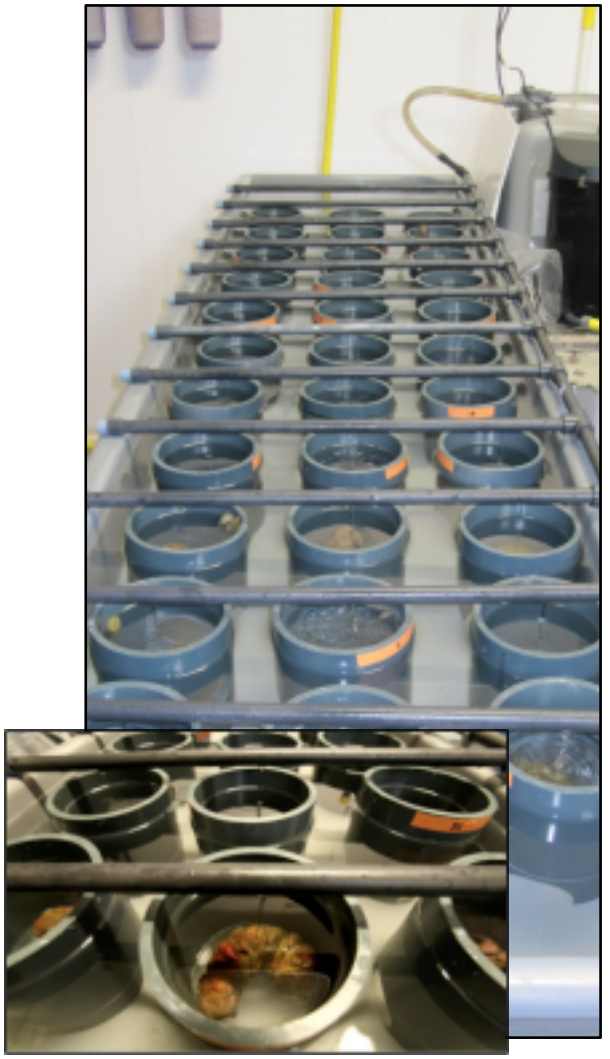
```
png(file="graph.png", width = 960, height = 960)

plot(nclutch/nfemales~stack_size,data2,xlab="stack size",ylab="female  […])
points(nclutch/nfemales~stack_size,subset(data2,stack_size>10),pch=16, […])
legend(9,10,c("stacks smaller than 10 ind.","stacks with >10 ind."),pc […])

dev.off()
```

  - l'option bg="transparent" de png() peut-être utile

Exercice

Objectif : représenter graphiquement les courbes de croissance des adultes en cours de manip

(Croissance de la coquille pendant 251 jours)



Exercice

1) Importer le jeux de données contenu dans le fichier growth.xlsx

2) Ajouter une colonne contenant la croissance absolue des adultes

3) Représenter graphiquement la croissance absolue en fonction de la taille initiale et du sexe

→ en utilisant la fonction plot() pour l'un des deux sexes

→ puis en utilisant la fonction points() pour l'autre sexe

4) Optimiser la qualité de présentation du graphique pour publication

→ utiliser les paramètres pch, col, xlim, ylim, xlab, ylab, cex.lab, cex.axis, las, …

→ ajouter une légende (mâles, femelles)

6) La croissance est bien représentée par le modèle polynomial suivant :

```
m1 <- lm(growth~sex*size+I(size^2), data)
```

→ utiliser la fonction "predict" pour prédire la variable réponse (growth) en fonction des variables explicatives (sex et size)

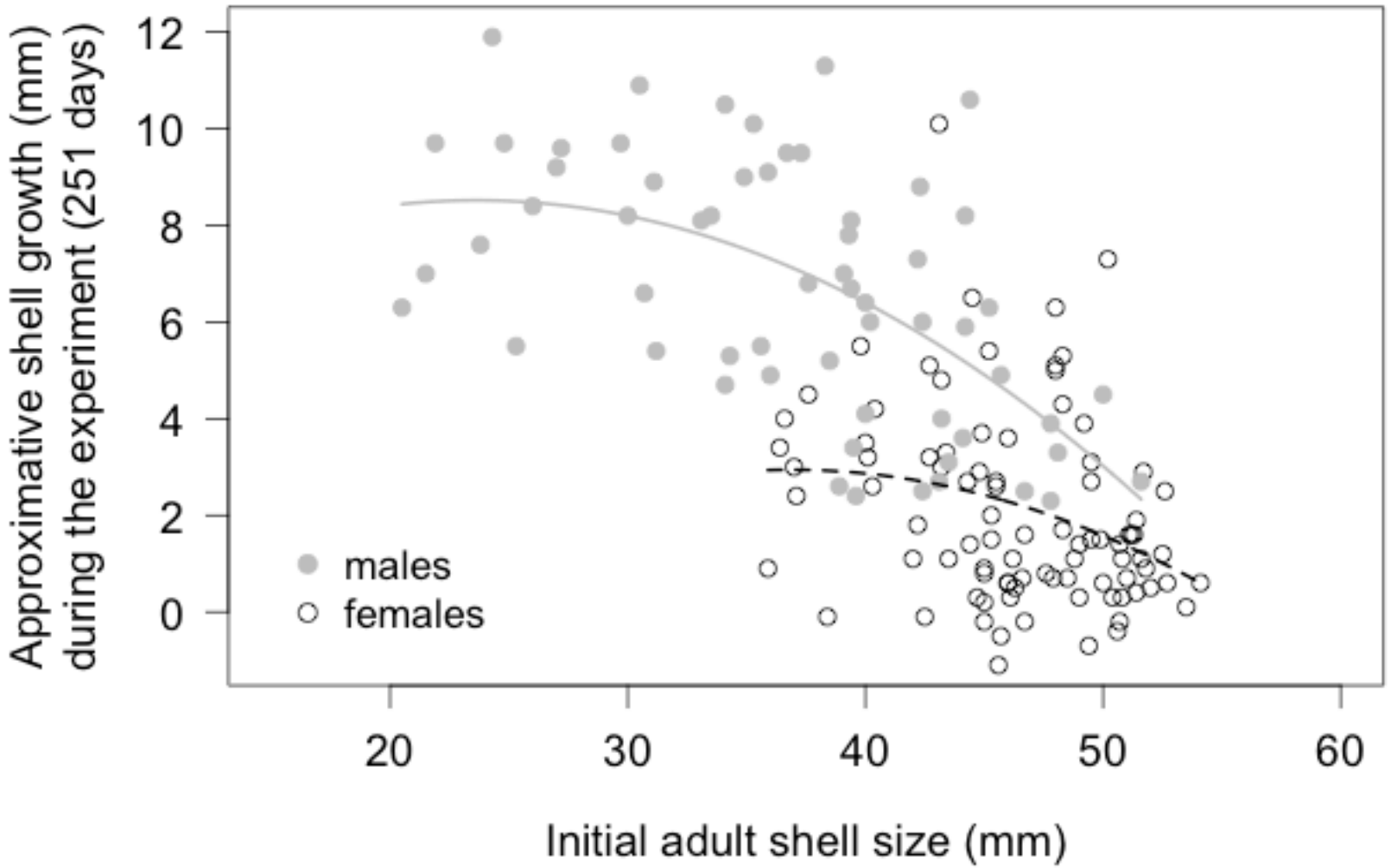
→ predict(modèle, data.frame(variable explicative1=…, variable explicative2=…))

→ utiliser la fontion "points" pour représenter les courbes de croissance mâle et femelle

```
# A plot of Crepidula fornicata growth in an experimental population

setwd("/Users/tbroquet/Documents/Enseignement/Intro_R_Form_int")

# load a (correctly formatted) dataset
data <- read.table("data/growth.txt", sep="\t", header=T)

# calculate growth
data$growth <- data$size2-data$size
```

```
par(mar=c(5, 5, 4, 2) + 0.1)  # give me some more room on the left

# Female-first
plot(data$growth[data$sex=="F"]~data$size[data$sex=="F"],
     xlim=c(15,60),  # this will allow smaller males to be visible as well
     ylim=c(-1,12),  # same idea for the y-axis
     xlab="Initial adult shell size (mm)",
     ylab="",        # I will draw my own axis title on the left
     cex.lab=1.2,    # Publication requires bigger labels
     cex.axis=1.2,
     cex=1,
     las=1)

# Here come the males, shown in grey
points(data$size[data$sex=="M"],data$growth[data$sex=="M"],
       col='grey',pch=19,cex=1)

# Then the y-axis label and legend
mtext("Approximative shell growth (mm)",side=2,line=4,cex=1.2)
mtext("during the experiment (251 days)", side=2,line=2.8,cex=1.2)

legend(15,2,c("males","females"),pch=c(19,1),col=c("grey","black"),bty="n",
       cex=1.1)
```

```
# Now I want to add predictive growth curves

# Polynomial model with a quadratic effect of initial size
m1 <- lm(growth~sex*size+I(size^2),data)

min(data$size)
max(data$size[data$sex=="M"])

# I wish to predict growth for males between 20.5 and 51.6 mm
x <- seq(20.5,51.6,0.1)
y <- predict(m1,data.frame(size=x,sex="M"))
points(x,y,type='l',col="grey",lwd=2)

# Now female growth
min(data$size[data$sex=="F"])
max(data$size[data$sex=="F"])
x <- seq(35.9,54.1,0.1)
y <- predict(m1,data.frame(size=x,sex="F"))
points(x,y,type='l',col="black",lty=2,lwd=2)
```


Approximative shell growth (mm)
during the experiment (251 days)
12
10
8
6
4
2
0
males
females
20
30
40
50
60
Initial adult shell size (mm)

UN CAS D'ÉCOLE

- Cas classique :
  - Un fichier d'annotation mal formaté (…)

Gene ID	Contig	Strand	Start	End	GeneType	description	GO-labels	protein domains	KOGid
Esi0000_0001	sctg_0	+	302	821	mRNA	expressed unknown protein	NA	NA	NA
Esi0000_0004	sctg_0	+	6364	11109	mRNA	hypothetical protein	NA	NA	NA
Esi0000_0005	sctg_0	-	13479	16792	mRNA	conserved unknown protein	NA	IPR002563: Flavin reductase-like, FMN-binding; IPR012349: FMN-bi	
Esi0000_0006	sctg_0	+	17040	21812	mRNA	conserved unknown protein	NA	NA	NA
Esi0000_0008	sctg_0	-	26488	30077	mRNA	expressed unknown protein	NA	NA	NA
Esi0000_0011	sctg_0	-	30876	40025	mRNA	Sphingosine kinase	GO:0004143;GO:0007205	IPR001206: Diacylglycerol kinase, catalytic region;	
Esi0000_0012	sctg_0	+	41042	48256	mRNA	expressed unknown protein	NA	NA	NA
Esi0000_0013	sctg_0	+	50756	60867	mRNA	expressed unknown protein	NA	NA	NA
Esi0000_0014	sctg_0	+	62273	68202	mRNA	conserved unknown protein	NA	NA	NA
Esi0000_0015	sctg_0	-	71243	76115	mRNA	bile Acid:Na+ symporter family	NA	IPR001763: Rhodanese-like	NA
Esi0000_0016	sctg_0	-	78257	78731	mRNA	expressed unknown protein	NA	NA	NA
Esi0000_0017	sctg_0	+	79806	81103	mRNA	calmodulin 2	GO:0005509	PTHR10891: CALMODULIN; IPR011992: EF-Hand type; SSF47473: EF-hand	
Esi0000_0018	sctg_0	-	81873	83822	mRNA	hypothetical protein	NA	NA	NA
Esi0000_0020	sctg_0	-	86250	91784	mRNA	conserved unknown protein	GO:0015031	PTHR22761: SNF7 - RELATED; IPR005024: Snf7	NA
Esi0000_0021	sctg_0	+	91943	100070	mRNA	conserved unknown protein	GO:0016491;GO:0055114	IPR001395: Aldo/keto reductase; IPR020471:	

  - Une sélection de gènes faite sur la base de transcriptomique

Condition	Gene ID
WT-GA DW	Esi0002_0097
WT-GA DW	Esi0005_0036
WT-GA DW	Esi0009_0041
[...]	
WT-GA DW	Esi0414_0009
WT-GA DW	Esi0448_0016
WT-GA DW	no gene found
WT-GA UP	Esi0000_0099
WT-GA UP	Esi0002_0166
[...]	
WT-GA UP	Esi1543_0001
WT-GA UP	Esi1667_0001

- Cas classique :
  - Un fichier d'annotation **très mal** formaté (…)

```
otein→GO:0004340;GO:0005524;GO:0006096—
5404: Potassium channel, voltage dependen
→n/a;n/a;n/a→NA→NALF
→GO:0004252;GO:0006508;n/a→IPR000595:
```

```
→mRNA→Alanyl-tRN
→mRNA→→Solubl
→mRNA→conserved
```

```
Esi0115_0039→sctg_115→-→189614→202049→mRNA→putative phosphatidylinositol 3-kinaseCRLF
→GO:0016303;GO:0016773;GO:0046854;GO:0048015→IPR001263: Phosphoinositide 3-kinase accessory region P
```

- La problématique
  - Mettre en évidence les catégories fonctionnelles (GO) sur/sous exprimées suivant les conditions testées

Un cas d'école La démarche

- La démarche
  1. importer le fichier d'annotation
  2. "reformater" pour rendre plus pratique l'exploitation des GO terms
  3. importer le fichier de transcriptomique
  4. fusionner les informations
  5. faire les effectifs de termes GO pour chaque condition

1. importer le fichier d'annotation

```
# -- annotation file -

# import

annotation = read.table(infileAnnotation, sep="\t", quote="", header=TRUE)

#head(annotation[!is.na(annotation[,colGO]),])
```

2. "reformater" pour rendre plus pratique l'exploitation des GO

```
# formatage

annotation2 = as.data.frame(matrix(ncol=ncol(annotation), nrow=0))

colGONum = which(colGO == colnames(annotation))
for (row_i in 1:nrow(annotation)) {
  myrow = as.character(annotation[row_i,])

  if (is.na(myrow[colGONum])) {
    annotation2 = rbind(annotation2, myrow)
  } else {
    for (go_i in unlist(strsplit(myrow[colGONum],";"))) {
      myrow2 = myrow
      myrow2[colGONum] = go_i
      annotation2 = rbind(annotation2, myrow2)
    }
  }

  # compteur
  if ((row_i%%100) == 0) {
    cat (row_i,"/",nrow(annotation),"\n")
  }
}

colnames(annotation2) = colnames(annotation)

#head(annotation2[!is.na(annotation2[,colGO]),])
```

3. importer le fichier de transcriptomique

```
# -- transciptomic file : up/down genes -

# import

transciptomic = read.table(infileTransciptomic, sep="\t", quote="", header=TRUE)
```

4. fusionner les informations

```
# -- merge info -

transciptomicAnnotated = merge(transciptomic, annotation2, by.x = 2, by.y = 1)
```

5. faire les effectifs de termes GO pour chaque condition

```
# -- counting -

#allGOs = levels(as.factor(transciptomicAnnotated[,colGO]))
allGOs = unique(transciptomicAnnotated[,colGO])

allConditions = unique(transciptomicAnnotated[,"Condition"])

for (condition_i in allConditions) {
  cat(condition_i,"\n")
  transciptomicAnnotatedCondition = factor(transciptomicAnnotated[transciptomicAnnotated["Condition"]
          ==condition_i,colGO], levels=allGOs)
  print(table(transciptomicAnnotatedCondition))
}
```

Est-ce qu'il y a moyen de faire avec le ~ ?

The End