- Licence Sciences et technologie
- Mention science de la vie – L2

SCIENCES SORBONNE UNIVERSITÉ

# INTRODUCTION A LA MODELISATION EN BIOLOGIE (BM1) - LU2SV382

## Modelisation statistique (Cours 2)

Martin LARSEN

www.immulab.fr

### Installing R

- Pour télécharger et installer R et Rstudio, accédez à https://rstudio.com/products/rstudio/download/#download
- Veuillez noter que vous devez d'abord installer R suivi de Rstudio.
- R et Rstudio sont des logiciels libres et peuvent être installés sur toutes les plateformes (Windows, Mac OS et Linux).

**Tutoriels vidéo pour installer R et Rstudio:**

- Installation R (https://www.youtube.com/watch?v=55a2drjIy-I) et Rstudio (https://www.youtube.com/watch?v=9GoswKwKDRE) (francais)
- Installez R et Rstudio sur MAC OS et premières étapes d'utilisation (anglais) (https://www.youtube.com/watch?v=orjLGFmx6l4)

- Absolument dernier solution uniquement pour ceux qui n'arrive pas a installé R et RStudio sur leur propre ordinateur (https://rstudio.cloud/)

# Prise en main de R et RStudio

LICENCE SCIENCES ET TECHNOLOGIE

MENTION SCIENCE DE LA VIE – L2

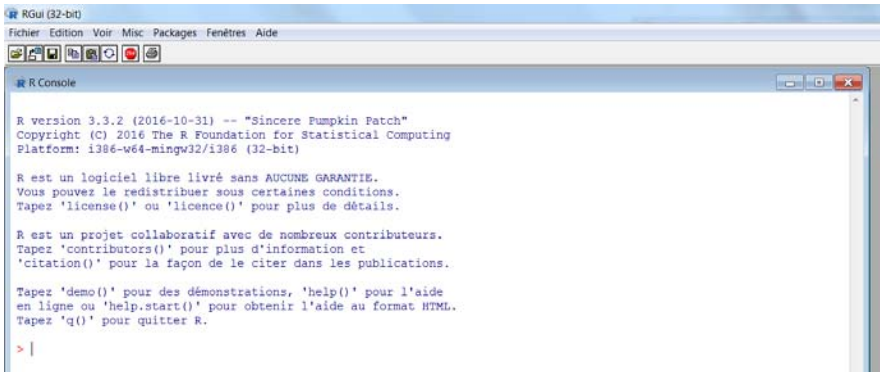
LARSEN MARTIN
DEMEYRIER VIRGINIE
RYBARCZYK HERVÉ

## Rappels sur R

- Il est construit autour d'un noyau de base contenant les principales fonctions. Le package nommé « base » est en quelque sorte le cœur de R.
- Il fait appel à des bibliothèques ou « packages » pour des utilisations plus spécifiques.
- Ces paquets sont en constant développement (communauté très dynamique).
- Il dispose d'une aide importante et pratiquement tous vos problèmes ou questions vont trouver une réponse sur un des innombrables forums.

## L'interface de R

- Une fois R lancé vous vous trouvez dans un fenêtre : la console de commande




## L'interface de R

- Du texte apparaît et un prompt « > » vous invite à saisir vos commandes.
- R manipule des objets
- MAIS ATTENTION : ceux-ci doivent IMPERATIVEMENT commencer par une lettre (A à Z ou a à z) et peuvent contenir des chiffres, des points ou des espaces soulignés.
- L'oublier est souvent source d'erreur et d'énervement.

## L'interface de R Studio

R Studio combine 4 éléments

- 1. Le terminal R
- 2. Un éditeur de texte / script lié au terminal R
- 3. Interface d'environnement et d'historique (par exemple, les valeurs de variables et de constantes)
- 4. Une interface pour les interactions entre R et les éléments externes (Fichiers, Graphiques, Packages, Aide et Viewer)

Pour zoomer sur des éléments individuels de R Studio, utilisez CTRL-SHIFT 0 à 9 (CTRL-SHIFT-0 fait référence à la disposition de base, tandis que les nombres de 1 à 9 zooment sur des éléments particuliers de Rstudio (1=Script, 2=Terminal, 3=Help, 4=History, 5=Files, 6=Plots, 7=Packages, 8=Environment and 9=Viewer).

R studio représente un environnement complet facilitant l'utilisation de R.

## Jouons un peu avec R – Calculatrice / Operation mathematique

| **Opérateur Arithmétiques:** | | **R comme calculatrice:** | |
|---|---|---|---|
| Addition: | + | > 5 + 3 | =8 |
| Soustraction: | - | > 5 – 3 | =2 |
| Multiplication: | * | > 5 * 3 | =15 |
| Division: | / | > 5 / 3 | =1.6 |
| Puissance: | ^ | > 5 ^ 3 | =125 |
| Modulo: | %% | > 5 %% 3 | =2, reste |
| Division euclidienne: | %/% | > 5 %/% 3 | =1, quotient |
| **Opérateurs relationnels:** | | | |
| ==, !=,<, >, <=, >= | | > 5 < 3 | =FALSE |
| **Opérateurs logiques:** | | | |
| Logical NOT | ! | > !FALSE | = TRUE |
| Element-wise logical AND | & | > c(FALSE,TRUE) & c(FALSE,TRUE) | = c(FALSE,TRUE) |
| Logical AND* | && | > c(FALSE,TRUE) && c(FALSE,TRUE) | = FALSE |
| Element-wise logical OR | \| | > c(FALSE,TRUE) \| c(FALSE,TRUE) | = c(FALSE, TRUE) |
| Logical OR* | \|\| | > c(FALSE,TRUE) \|\| c(FALSE,TRUE) | = FALSE |

*&& and || only evaluates the first element of each vector.

## Jouons un peu avec R – Initiation des Objets

**Initiation des objets:**

- Créons un objet « n » et donnons lui la valeur 5 (par exemple)

n <- 5 n = 5 5 -> n

Faire entrer pour valider la commande

Réponse de R : ?

- Voyons ce que R a fait :

> n

Réponse de R :

[1] 5

**Refait avec l'éditeur de script.**

**Commentaire:** **Pour expliquer le but du code (ajouter # au début de la ligne)**
**Pour exclure de(s) ligne(s) de code (CTRL+SHIFT+C)**

**Exécuter:** **Run line(s) of script: CTRL+ENTER (Windows) or CMD+ENTER (MAC)**

## Jouons un peu avec R – Type de Données

**Type de donnée:**

- **character**: "a", "swc"
- **numeric**: 2, 15.5
- **integer**: 2L (L fait stocker en tant qu'entier)
- **logical**: TRUE, FALSE
- **complex**: 1+4i (réel et imaginaire)
- **Factors** (catégorique)

**Structure des données:**

- **Atomic vector** (meme type pour touts elements)
- **List** (aucune restriction de type)
- **Matrix** (meme type pour touts elements)
- **Data.frame** (aucune restriction de type, liste de liste ou toutes liste on le même longueur.
- **Array:** Matrix able to have more than 2 dimensions.

**Caractéristiques des objets:**

**str()** – sommaire des genres d'objet!
**class()** - quel genre d'objet (de haut niveau)?
**typeof()** - quel est le type de données (de bas niveau)?
**length()** – longeur d'une vecteur OU nombre d'element dans les objets en deux dimensions?
**attributes()** - a-t-il des métadonnées?

| Data type | Example | class() | typeof() | Length |
|---|---|---|---|---|
| Valeur | n = 5 | Numeric | Double | 1 |
| Vecteur | v1 = c(2.5,3.5,4.5,5.5)<br>v2 = 1:10 | Numeric<br>Numeric | Double<br>Integer | 4<br>10 |
| Liste | l1 = list(2.5,3.5,4.5,5.5)<br>l2 = list(x="a", y=1:5, z=v2) | List<br>List | List<br>List | 4<br>3 |
| Matrix | m = matrix(v1,nrow=2,ncol=2) | Matrix | Double | 4 (elem) |
| Data.Frame | d = data.frame(x=v1, y=2*v1) | Data.frame | List | 2 (lists) |

## Les principaux types d'objets sous R

- Les vecteurs (variables avec valeurs assignées)

**un exemple de vecteur sous R :** longueur <- c(32,40,42,38,41.5,43.7,39.5,41.25,38.65,40.8)

- Les data frame (tableaux de variables)

**un exemple de data.frame sous R**

| | Dates | Fluo | Turbidity | Salinity | Temp | Chla | Pheo | PheoT | Tchla | NO2.NO3. | PO4 | SiO4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2009-12-15 | 1.6394600 | 1.5664000 | 35.60542 | 25.16604 | 0.9481284 | 0.1636948 | 14.72310 | 1.1118233 | 0.3931333 | 0.07053333 | 0 |
| 2 | 2010-01-15 | 1.7694375 | 1.4226875 | 35.26078 | 25.53967 | 0.5415489 | 0.1337643 | 19.80774 | 0.6753132 | 0.5184000 | 0.08123333 | 0 |
| 3 | 2010-02-15 | 0.5810062 | 1.1380000 | 35.34602 | 26.84876 | 0.6401712 | 0.1104446 | 14.71386 | 0.7506158 | 0.1988333 | 0.05093333 | 0 |
| 4 | 2010-03-16 | 0.0000000 | 0.5980000 | 35.30830 | 25.13370 | 0.6636906 | 0.1803576 | 21.36816 | 0.8440481 | 0.2657667 | 0.06903333 | 0 |
| 5 | 2010-04-15 | 0.0000000 | 0.6170000 | 35.39920 | 25.27000 | 0.6343510 | 0.1034000 | 14.01502 | 0.7377798 | 0.1065000 | 0.10760000 | 0 |
| 6 | 2010-05-17 | 0.7271211 | 0.6723684 | 35.43810 | 24.78147 | 0.4321119 | 0.0800000 | 15.61877 | 0.5122043 | 0.1167333 | 0.02980000 | 0 |

- Les tables (tableaux de contingences)
- Les matrices

**un exemple de table de contingence sous R**

| | A | B | Sum |
|---|---|---|---|
| C | 70 | 150 | 220 |
| D | 100 | 350 | 450 |
| Sum | 170 | 500 | 670 |

## Jouons un peu avec R – initiation des objets

- On peut entrer des « noms » dans des variables
  - Attention le point-virgule s'utilise pour séparer des commandes distinctes sur la même ligne

➢ name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5

- La fonction ls (pour list) permet d'afficher des objets en mémoire :

```
> ls()
[1] "m" "n1" "n2" "name"
```

- Seuls les « noms » des objets apparaissent ….

- Il peut être utile de « nettoyer » les variables en effaçant leur contenu.
- La commande rm() pour « remove » s'en charge :

```
> rm(n)
> n
Erreur : objet 'n' introuvable
```

## Jouons un peu avec R – action d'operateur sur des vecteurs et des matrices

```
v1 = c(2.5,3.5,4.5,5.5)
v2 = 1:10   # [1]  1  2  3  4  5  6  7  8  9 10
➢ 2 * v1
[1]  5  7  9 11
➢ v2 %% 2     #  [1] 1 0 1 0 1 0 1 0 1 0
➢ v2 %% 2 == 0
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
➢ mean(v1) # toutes éléments évalué ensemble
[1] 4
➢ sapply(v2,function(x){x %% 2 == 0}) # toutes éléments évalué individuellement
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE

m = matrix(v1,nrow=2,ncol=2)
➢ apply(m,1,mean)
[1] 3.5  4.5
➢ apply(m,2,mean)
[1] 3     5
```

| m | [,1] | [,2] | Mean |
|---|---|---|---|
| [1,] | 2.5 | 4.5 | 3.5 |
| [2,] | 3.5 | 5.5 | 4.5 |
| Mean | 3 | 5 | |

## R comme language de programmation

**Function:**

- foo <- function(Variable(s) indépendante(s), paramètre(s)){
  --- ligne de code ---
  algorithme applique sur les variable indépendante
  conditionnée par les paramètres
  --- ligne de code ---
  return(résultat)
  }

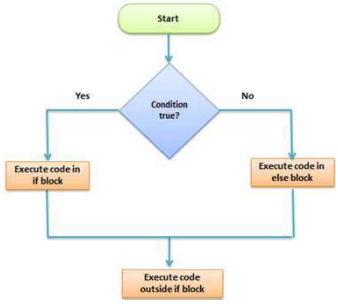
- foo1 <- function(x){2*x + 2}
- foo2 <- function(x, slope, intersect){slope*x + intersect} # paramètre: slope et intersect
  - foo1(5) == foo2(5, 2, 2)
- foo2_bis <- function(x, slope, intersect){
  res <- slope * x
  res <- res + intersect
  return(res)
  }
- foo3 <- function(x,y){x*y}

**Plot:** > curve(foo1)

## R comme language de programmation

### Sélection conditionnelle

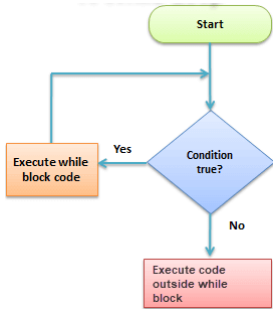
- if (n == 7){  # n une variable
  print('n=7')  # n partie d'une chaine de caractères
  } else {
  print('n!=7')
  }
- ifelse(n==7,print('n=7'),print('n!=7'))




### Boucle (While, For et Apply):

- i <- 1
  while (i <= 5){
  print(i)
  i <- i + 1
  }
- for (i in 1:5){
  print(i)
  }
- sapply(1:5,print) # boucle for optimiser

- for (i in <iter>){
  foo(i)
  }
- foo <- function(x){
  <algorithme>
  }
- sapply(<iter>, foo)




## R comme language de programmation

### If, else if, else

- Sélection conditionnelle multiple
- Inclus else (toutes autre car)

- foo4 <- function(x,y, param){
  if (param=='somme'){
  return(x + y)
  } else if (param=='prod') {
  return(x * y)
  } else {
  return(NULL)
  }

**Validation:**

> foo4(5, 4, 'somme') == 9
> foo4(5, 4, 'prod') == 20
> foo4(5, 4, 'sous') == NULL

### Switch()

- Switch est comme un dictionnaire (Python) avec clé et valeur associée, où la valeur peut être un algorithme.
- Pas de **else**

- foo4_bis <- funtion(x,y,param){
  switch( param,
  "somme" = {x+y},
  "prod" = {x*y,}
  )
  }

**Validation:**

> foo4_bis(5, 4, 'somme') == 9
> foo4_bis (5, 4, 'prod') == 20
> foo4_bis (5, 4, 'sous') == NULL

## R comme language de programmation

### Exercice de programmation

- Créer une fonction qui transforme °C (Tc) en Fahrenheit (Tf): Tf = 32 + 1.8 * Tc
- Créer une fonction qui transforme Fahrenheit (Tf) en °C (Tc)
- Crée une fonction f (a, b) qui résout l'équation: 0 = a * x + b
- Créez une fonction f (a, b, c) qui résout l'équation: 0 = a * x ^ 2 + b * x + c
- Créez une fonction f (secu, cle) qui valide votre numéro de sécurité sociale, sachant que 97 moins le reste d'une division euclidienne de 'secu' avec 97 doit être égal à la 'cléf'
  - Pour teste: secu=2850314025037, clef=50
- Create a function f(sense) that reverse complement a DNA code

Second order polynomial roots

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$D = b^2 - 4ac$

**D>0, Two real solutions**
**D=0, One real solution**
**D<0, Complex Solutions**

## R comme language de programmation - Solution de programmation

```
Tc_to_Tf <- function(Tc){32 + 1.8 * Tc}
Tf_to_Tc <- function(Tf){(Tf-32)/1.8}

firstOrder <- function(a,b){-b/a}

secondOrder <- function(a,b,c){
    dis = b^2-4*a*c
    if (a==0){
        sol = firstOrder(b,c) # -c/b
    } else {
        if (dis > 0){
            sol <- (-b+dis^0.5)/(2*a)
            sol <- c(sol, (-b-dis^0.5)/(2*a) )
            }
        if (dis==0){
            sol <- -b/(2*a)
        } else {
            sol = 'complex'
        }
    }
    return sol
}
```

```
secuVallid <- function(secu,clef){
    reste <- secu %% 97
    res <- 97 – res
    return(res==cle)
}

revcompbase <- function(base){
    res <- switch(base,'a'='t','t'='a','g'='c','c'='g')
    return(res)
}
revcompstr <- function(sense){
    i <- nchar(sense)
    res <- ""
    while (i>0){
        res <- paste(res,revcompbase(
                substr(sense,start=i,stop=i)),sep="")
        i <- i-1
    }
    return(res)
}

revcompstr_bis <- function(sense){
  sense_split <- strsplit(sense, "")[[1]]
```

## R comme language de programmation - Solution de programmation

```
revcompbase <- function(base){
        res <- switch(base,'a'='t','t'='a','g'='c','c'='g')
        return(res)
}
```

Contrairement à Python, R ne peut pas naturellement itérer sur une chaîne de caractères. En utilisant strsplit(), vous pouvez convertir une chaîne de caractères en une liste de caractères individuels:

```
> strsplit('atcg',"")[[1]]
[1] "a" "t" "c" "g"
```

**Boucle WHILE**

```
revcompstr <- function(sense){
        i <- nchar(sense)
        res <- ""
        while (i>0){
                e <- substr(sense,start=i,stop=i)
                res <- paste(res,revcompbase(e),sep="")
                i <- i-1
        }
        return(res)
}
```

**Boucle FOR**

```
revcompstr_bis <- function(sense){
  sense_split <- strsplit(sense, "")[[1]]
  res <- ""
  for (e in sense_split){
    res <- paste(res,revcompbase(e), sep="")
  }
  return(res)
}
```

## Matrix algebra avec R

### Valeurs et vecteurs propre

- M = matrix(c(1,4,2,3,nrow=2, ncol=2)

*Resoudre:*

$$X_1 + 2x_2 = \lambda * x_1$$
$$4x_1 + 3x_2 = \lambda * x_2$$

$$M * \vec{v} = \lambda * \vec{v}$$
$$\begin{bmatrix}1 & 2\\ 4 & 3\end{bmatrix} * \begin{bmatrix}x_1\\ x_2\end{bmatrix} = \lambda * \begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

```
> e <- eigen(M)
eigen() decomposition
e$values
[1]  5 -1
```

e$vectors # all vectors parallel to the column vectors are eigenvectors.

| | [,1] | [,2] |
|---|---|---|
| [1,] | -0.4472136 | -0.7071068 |
| [2,] | -0.8944272 | 0.7071068 |

- v = e$vectors[,1]
- l = e$values[1]
- M %*% v == l*v
- det(e$values[1]*diag(2)-M) == 0

### Résoudre un système d'équations

$$a_1x_1 + b_1x_2 = y_1$$
$$a_2x_1 + b_2x_2 = y_2$$

$$M * \vec{x} = \vec{y}$$
$$\begin{bmatrix}a_1 & b_1\\ a_2 & b_2\end{bmatrix} * \begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}y_1\\ y_2\end{bmatrix}$$

$$M^{-1} * M * \vec{x} = M^{-1} * \vec{y}$$
$$\vec{x} = M^{-1} * \vec{y}$$

```
> det(M) != 0      # pour resoudre: det(M) != 0
> solve(M,y)
Or
> solve(M) %*% y
```

- $M^{-1}$ = solve(M)

$$-x_1 + 2x_2 + x_3 = 0$$
$$-x_1 + x_2 + 2x_3 = 2$$
$$x_1 - 2x_2 + x_3 = 1$$

$$M * \vec{x} = \vec{y}$$
$$\begin{bmatrix}-1 & 2 & 1\\ -1 & 1 & 2\\ 1 & -2 & 1\end{bmatrix} * \begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} = \begin{bmatrix}0\\ 2\\ 1\end{bmatrix}$$

$x_1 = -2.5$  $x_2 = -1.5$  $x_3 = 0.5$

## Jouons un peu avec R

- Définissons un objet « longueur »
  - longueur <- c(32,40,42,38,41.5,43.7,39.5,41.25,38.65,40.8)

- Nous avons créé un objet, de type « VECTEUR », à une dimension contenant 10 valeurs. La commande suivante le vérifie :

➢length(longueur)

[1] 10

> longueur

[1] 32.00 40.00 42.00 38.00 41.50 43.70 39.50 41.25 38.65 40.80

- Les opérateurs mathématiques appliqués à un objet (e.g. vecteur, matrix ou data.frame) sont automatiquement appliqués à chaque élément de l'objet (contrairement à Python où il faudrait une boucle).

## Jouons un peu avec R - aide

- La commande > ?mean
  - Va vous fournir l'aide sur la fonction « mean » par exemple.
  - Valable pour toutes les fonctions !!

mean {base} R Documentation

### 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. Complex vectors are allowed for trim = 0, only. |

## Jouons un peu avec R – Des Fonctions intégré en R

**Appliquer un fonction:**

- mean(longueur)
- var(longueur)

> pnorm(1.96, mean=0, sd=1)
[1] 0.9750021

> qnorm(0.975, mean=0, sd=1)
[1] 1.959964

- La fonction pnorm : N(0,1) loi Normale centrée-réduite
- ATTENTION : Les décimaux TOUJOURS avec un « . »

## Quelques commandes

- Taille d'echantillon (N) : length(longueur) donne [1] 10
- L'étendue : range(longueur) donne [1] 32.0 43.7
- La moyenne : mean(longueur) donne [1] 39.74
- La médiane : quartile(longueur,0.50) donne ????
- L'écart-type : sd(longueur) donne [1] 3.189201
- L'erreur standard (L'écart-type de l'espérance):
- es=sd(longueur)/(sqrt(length(longueur)))
- Le coefficient de variation :
  - cv=sd(longueur)/mean(longueur)

$$\hat{\sigma}^2 = \frac{1}{n-1}\left(\sum x^2 - n\bar{x}^2\right)$$

$$es = \hat{\sigma}_{\bar{X}} = \frac{\hat{\sigma}_X}{\sqrt{N}}$$

$$cv = \frac{\hat{\sigma}_X}{\bar{x}}$$

Le coefficient de variation (cv) permet de comparer des évolutions de variables n'ayant pas la même échelle ex: croissance d'une souris et d'un éléphant …

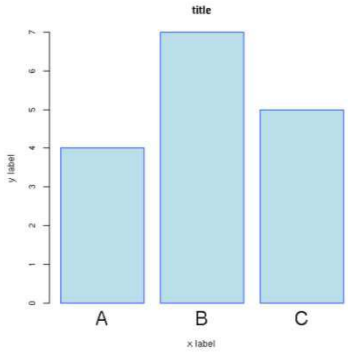
Créer votre propre fonction pour calculer la moyenne et la variance d'un ensemble de données.
F <- function(x,y,z){algorithm}

## Représentation graphique

- Variable aléatoire discrète : Diagramme en barre barplot()

vect <- c(a = 4, b = 7, c = 5)

**barplot(**vect, col = "lightblue", border = "blue", names.arg =toupper(names(vect)), cex.names = 2, cex.axis = 0.8, main = "title", xlab = "x label", ylab = "y label", axes = TRUE**)**




## Représentation graphique

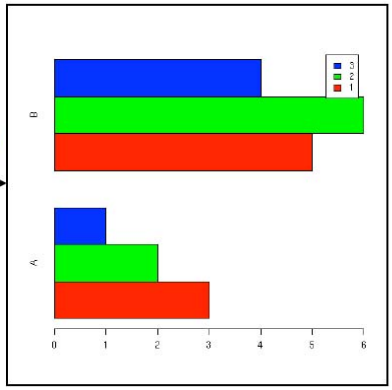
- mat <- matrix(c(3, 2, 1, 5, 6, 4), nrow = 3, dimnames = list(c("1", "2", "3"), c("a", "b")))

Ou

- mat <- matrix(c(3, 2, 1, 5, 6, 4), nrow = 3)
- row.names(mat) <- c("1", "2", "3")
- colnames(mat) <- c("a", "b")

| | a | b |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 2 | 6 |
| 3 | 1 | 4 |

barplot(mat)

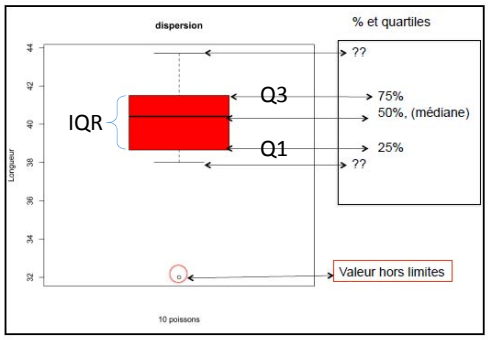



- barplot(mat, main = "Title", xlab = "x label", col = rainbow(3), beside=TRUE, horiz = T )
- legend("bottomright", row.names(mat), fill = rainbow(3) )

## Représentation graphique

- Variable aléatoire continue : Boîte à moustache

longueur <- c(32,40,42,38,41.5,43.7,39.5,41.25,38.65,40.8)
plotTmp <- boxplot(longueur,col="red")
Essai plotTmp$




Ajoutons la moyenne
> points(1,mean(longueur),col="black")
> points(1,median(longueur),col="green")

- Les valeurs des moustaches s'obtiennent par la formule suivante :

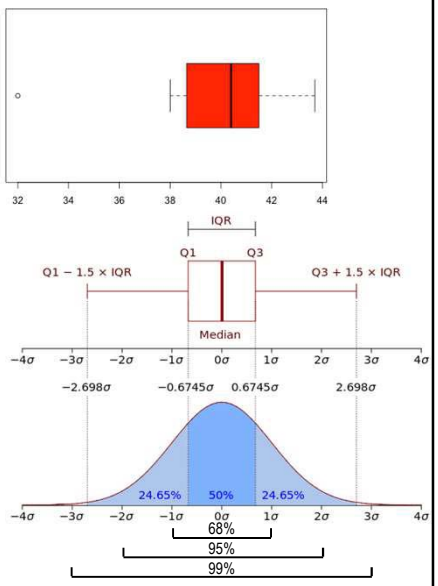
Upper whisker = min(max, max(non outliers), Q3 + IQR*1.5)
Lower whisker = max(min, min(non outliers), Q1 - IQR*1.5)

Une valeur « hors limite » on l'appelle un « outlier » (=plotTmp$out)

## Représentation graphique

- Variable aléatoire continue : Boîte à moustache

longueur <- c(32,40,42,38,41.5,43.7,39.5,41.25,38.65,40.8)
plotTmp <- boxplot(longueur,col="red« , horizontal=T)
Essai plotTmp$




Ajoutons la moyenne
> points(1,mean(longueur),col="black")
> points(1,median(longueur),col="green")

- Les valeurs des moustaches s'obtiennent par la formule suivante :

Upper whisker = min(max, max(non outliers), Q3 + IQR*1.5)
Lower whisker = max(min, min(non outliers), Q1 - IQR*1.5)

Une valeur « hors limite » on l'appelle un « outlier » (=plotTmp$out)
> En dehors de 1,5 fois IQR +/- quartile supérieur (+) ou inférieur (-)

## Représentation graphique

- Ces quantiles peuvent s'obtenir par la formule suivante :

> quantile(longueur,0.25)

où 0.25 correspond au 1er quartile soit 25% de la distribution
et cette commande retourne comme résultat :
25%
38.8625
Cela veut dire que 25% des valeurs de longueur la précédent

- Quelles bornes pour 10,50,75,90% de la distribution?

## Représentation graphique

- Une commande simple résume les différents paramètres de position d'une distribution des valeurs d'une V.A.C.

> summary(longueur)

qui donne

| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max |
|---|---|---|---|---|---|
| 32.00 | 38.86 | 40.40 | 39.74 | 41.44 | 43.70 |

## Représentation graphique

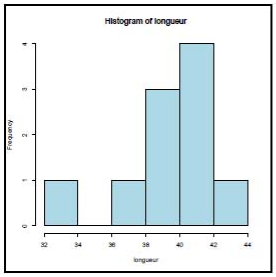
- On retire la valeur considérée comme aberrante et on recommence l'analyse des données.
- On forme un nouveau vecteur à partir de longueur auquel on a retiré la 1ère valeur
  - new <- longueur[-1]
  - new <- longueur[!(longueur %in% plotTmp$out)] # Alternative plus general que longueur != plotTmp$out (enlève 32.5 de « longueur »)
  - length(new)
- Résumons ce que nous venons de voir et enrichissons le:
  - new <- c(40,42,38,41.5,43.7,39.5,41.25,38.65,40.8)
  - boxplot(new,horizontal=T,col="red",main="dispersion",xlab="9 poissons", ylab="Longueur", add=FALSE)

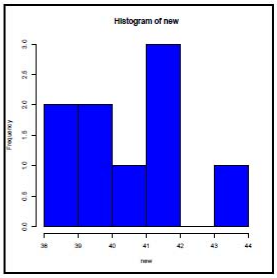
## Représentation graphique

- Traçons maintenant l'histogramme des données (effectifs ou fréquences):

> hist(longueur,col="lightblue",lwd=3,nclass=5)

> hist(new,col="blue",lwd=3,nclass=5)



## Représentation graphique

- Pour afficher plusieurs graphes dans la même fenêtre graphique, on peut la diviser en plusieurs sous parties avec la commande

par(mfrow=c(2,2))  # mfrow=c(nrows, ncols) crée une matrix de nrows x ncols remplis par ligne.

par exemple qui divise la fenêtre en 2 rangs et 2 colonnes (2x2) soit 4 sous graphes

- Essayez :

> par(mfrow=c(2,2))

> hist(longueur, col="lightblue", lwd=3)

> hist(new, col="blue", lwd=3)

## Représentation graphique

- L'histogramme (de longueur ou de new) n'est pas très représentatif d'une distribution spécifique.
- Il apparait plutôt incomplet par rapport à ce que l'on pourrait attendre d'une distribution de la taille (variable aléatoire continue dans la population de poissons de cette espèce).
- Simulons la distribution de plusieurs échantillons dont les valeurs suivent une **loi normale** avec n = 20, 50, 100, 1000 avec les mêmes paramètres :
  - moyenne de new = 40.6
  - écart-type de new =1.76

## Représentation graphique – distribution normale theorique

- On va utiliser la commande « rnorm(**a,b,c**) » qui permet de tirer au hasard

**a** : nombre de valeurs d'une distribution **normale** de paramètres
**moyenne** =**b**
et **écart-type** =**c**

- Ce qui dans notre exemple nous donne

> **set.seed(123) # pour rendre le hazard reproductible**
**> X=rnorm(9,40.6,1.7)**
**> hist(X)**

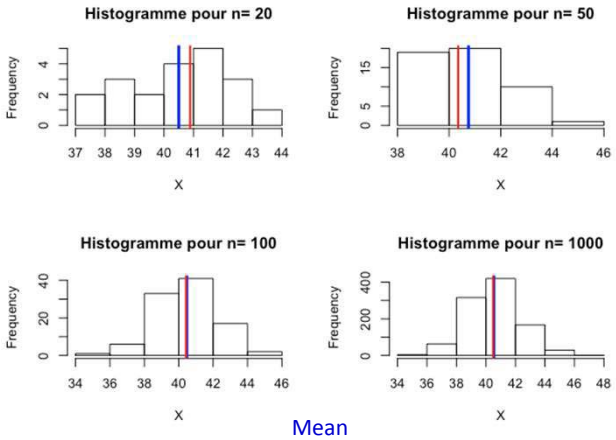
La commande **abline(v=mean(X))** qui permet de tracer une droite v pour verticale et de constante **mean(X)** et on recommence pour la médiane.

## Représentation graphique – distribution normale theorique

- # simulation de data Echantillon / Population

```
par(mfrow=c(2,2))

X=rnorm(20,40.6,1.76); hist(X,main="n=20")
abline(v=mean(X),lwd=3);abline(v=median(X),lwd=2,col="red")

X=rnorm(50,40.6,1.76);hist(X,main="n=50")
abline(v=mean(X),lwd=3);abline(v=median(X),lwd=2,col="red")

X=rnorm(100,40.6,1.76);hist(X,main="n=100")
abline(v=mean(X),lwd=3);abline(v=median(X),lwd=2,col="red")

X=rnorm(1000,40.6,1.76);hist(X,main="n=1000")
abline(v=mean(X),lwd=3);abline(v=median(X),lwd=2,col="red")
```

## Représentation graphique – distribution normale theorique



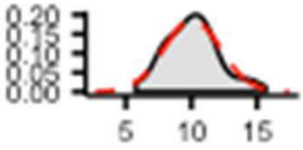

Mean
Mediane

Conclusion des simulations ?

Effectifs, Echantillons, Population, …

## Représentation graphique – distribution normale theorique

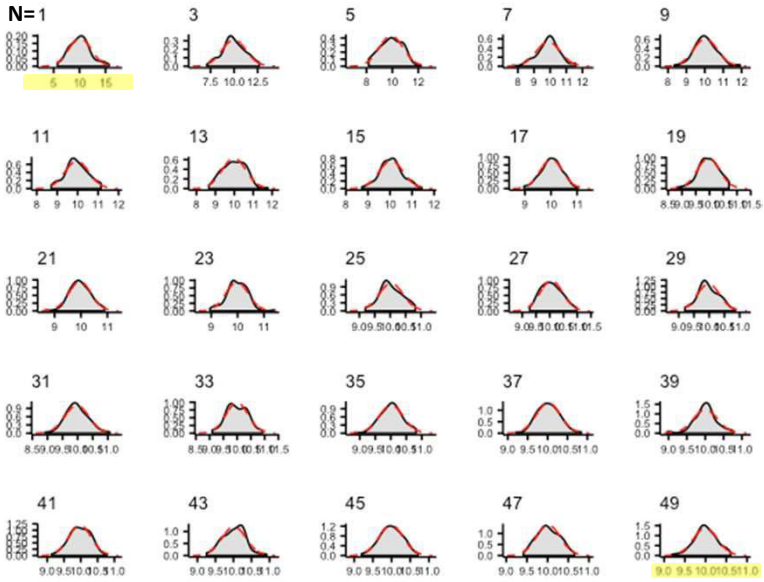
X ~ N(μ=10; σ=2)
N=100



Observed
Theoretical normal distribution

X ~ N(μ=10; σ=2)

**X_bar distribution: 100 échantillons a taille n éléments chaque**

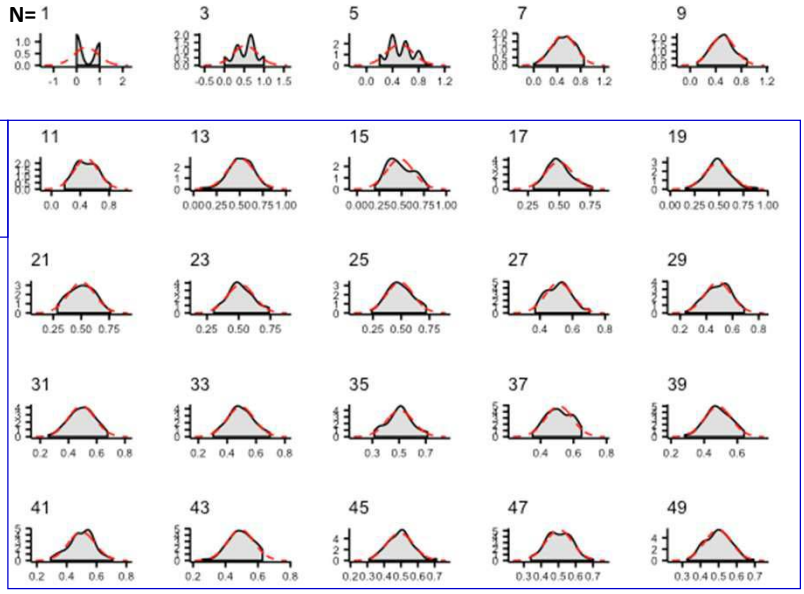



Distribution d'échantillonnage de la Moyenne

X_bar ~ N(μ; σ/√n)

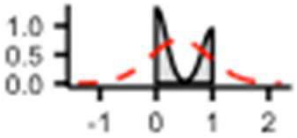
## Représentation graphique – distribution binomiale theorique

**F distribution: 100 échantillons a taille n éléments chaque**




X ~ B(n=1; π=0.5)

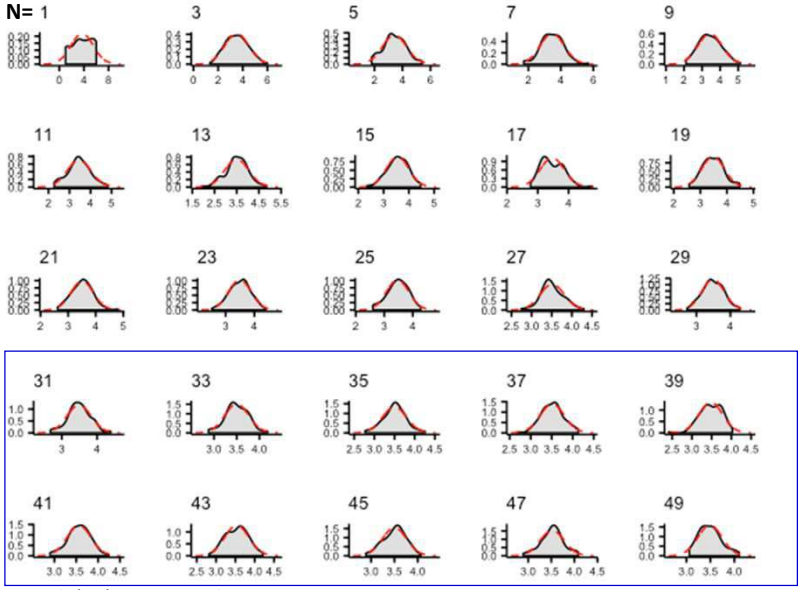
X ~ N(nπ, nπ(1-π))
F=X/n ~ N(π, π(1-π)/n)
si nπ>5 et n(1-π)>5



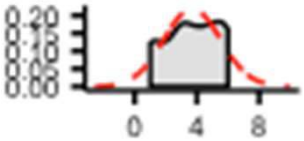
Observed
Theoretical normal distribution

## Représentation graphique – distribution uniforme theorique

**X_bar distribution: 100 échantillons a taille n éléments chaque**




X ~ Uniforme [1:6]
μ=3.5; σ=1.7



Observed
Theoretical normal distribution

X ~ Uniforme → Théorème Centrale Limite
Condition: N>30 → X_bar ~ N(μ; σ/√n)

## Représentation graphique

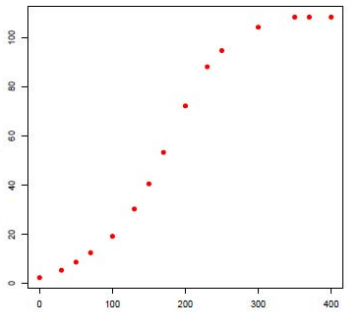
- Diagrammes binaires pour les variables continues : nuage de points ou « scatter plot » plot()

- **Exemple de croissance bactérienne Y en fonction du temps X**

**x <- c(0, 30, 50, 70, 100, 130, 150, 170, 200, 230, 250, 300, 350,370, 400)**

**y <- c(2.4, 5.4, 8.7, 12.5, 19.2, 30.3, 40.5, 53.3, 72.2, 88.1,94.7, 104.2, 108.3, 108.3, 108.3)**

**plot(y~x,pch=19,col="red")**




## Résumé des principales commandes

- > signifie que R attend vos instructions
- <-, -> ou = (correspond a <-) assigne une valeur ou du texte à un objet
- c(…,…) permet d'assigner plusieurs éléments dans un objet
- le . sépare les décimales des unités
- la , sépare les éléments d'un vecteur ou d'une commande
- le ; sépare les commandes dans une même ligne
- barplot()
- boxplot()
- hist()
- plot(x,y) ou plot(y~x)