R en 30 minutes!

Gauthier Ligot

11 mars 2011

1. Introduction
   - Objectifs de la présentation
   - R, c'est quoi?
   - Premiers pas
2. Les bases du language R
   - Assignation et gestion des objets
   - Opérateurs et fonctions
   - Les objets R
   - Programmation
3. Gestion des données
4. Exemples
   - Graphiques et régressions linéaires
   - Régressions non-linéaires
   - Graphiques en treillis
   - Bootstrap
   - Divers
5. Documentation sur le Web

Objectifs

- Introduction à R en 30 min !
- Manipulation de données
- Illustration de certaines fonctions
- Peu d'info sur les fonctions statistiques et graphiques
- Peu d'info pour programmer et faire ses propres fonctions

R, c'est quoi ?

- Environnement Open Source pour les calculs statistiques et la réalisation de graphiques
- Download : `http://cran.r-project.org/`
- Un language de programmation qui vient du S et du S-plus (Unix)
- Extensions par l'utilisation de packages (plus de 1800 actuellement)
- Fonctionnement en ligne de commande (principalement)

R, c'est quoi ?

- La fenêtre principale de R est la console
- Les lignes de commande peuvent y être directement inscrites
- Mais il est préférable de les écrire d'abord dans un éditeur
- Plusieurs éditeurs (et debugeurs) spécialisés ont également vu le jour : TinnR (Windows), Rkward (Linux), statet sous éclipse, Emacs et ESS...

Première utilisation

- > est le prompt
- # pour introduire une ligne de commentaire
- si l'expression est incomplète, alors le symbole + apparait à la ligne suivante
- [1] indique le premier élément d'un vecteur

```
> 1 + 3
[1] 4
> pi
[1] 3.141593
> exp(32)
[1] 7.896296e+13
> sin(2 * pi)
[1] -2.449213e-16
```

L'aide et les packages

- `library(lme4)` ou `require(lme4)` pour charger le package
- `install.packages("lme4")` pour l'installer
- Chaque fonction et chaque package possèdent un fichier d'aide (+ nombreux tutoriels sur Internet)
- `?` pour avoir l'aide d'une fonction précise
- `??` pour chercher l'aide dans les packages qui ne sont pas chargés
- `citation()` pour obtenir les références de R et `citation("lme4")` pour les références du package lme4

Assignation

- l'opérateur est <- (ou =).
- `ls()` donne la liste des objets en mémoire
- `rm()` suprime des objets, `rm(list=ls())` supprime tous les objets en mémoire

```
> x <- 5
> y <- x
> x
[1] 5
> ls()
[1] "x" "y"
> rm(x)
> ls()
[1] "y"
```

Opérateurs

Opérateurs

Arithmétique		Comparaison		Logique	
`+`	addition	`<`	inférieur à	`! x`	NON logique
`-`	soustraction	`>`	supérieur à	`x & y`	ET logique
`*`	multiplication	`<=`	inférieur ou égal à	`x && y`	idem
`/`	division	`>=`	supérieur ou égal à	`x | y`	OU logique
`^`	puissance	`==`	égal	`x || y`	idem
`%%`	modulo	`!=`	différent	`xor(x, y)`	OU exclusif
`%/%`	division entière				

Opérateurs - Exemples

```
> 9%%2
[1] 1
> 9%/%2
[1] 4
> 10 == 10
[1] TRUE
> "gauthier" != "Gauthier"
[1] TRUE
> 12 < 11 && 10 >= 2
[1] FALSE
> 12 < 11 || 10 >= 2
[1] TRUE
> 12 %in% c(10, 12, 15)
[1] TRUE
```

Les objets R

Objet	**modes**	**plusieurs modes ?**
vector	numérique, caractère, complexe, logique	non
factor	numérique, caractère	non
array	numérique, caractère, complexe, logique	non
matrix	numérique, caractère, complexe, logique	non
dataframe	numérique, caractère, complexe, logique	oui
ts	numérique, caractère, complexe, logique	non
liste	numérique, caractère, complexe, logique fonction, expression	oui

Le mode des objets R

Les données peuvent être numériques, caractères, complexes ou logiques.

- `mode()` permet d'identifier le mode

```
> x <- 12/0
> mode(x)
[1] "numeric"
> x
[1] Inf
> x - x
[1] NaN
> x <- "Hello"
> mode(x)
[1] "character"
> x <- (12 < 14)
> mode(x)
[1] "logical"
```

Vecteurs I

- Construire des vecteurs avec : `c()`, `:`, et `seq()`.

```
> c(1, 2, 10)
[1]  1  2 10
> 1:4
[1] 1 2 3 4
> seq(5, 10, by = 1)
[1]  5  6  7  8  9 10
> x <- c(1, 2:4, seq(5, 10, by = 1))
```

- Les parenthèses `[]` permettent de spécifier les indices à utiliser

```
> x[5]
[1] 5
> x[-5]
[1]  1  2  3  4  6  7  8  9 10
> x[c(1, 5, 6)]
[1] 1 5 6
```

Vecteurs II

- La règle du recyclage !

```
> x <- 1:10
> y <- c(0, 10)
> x + y
 [1]  1 12  3 14  5 16  7 18  9 20
```

Quelques fonctions toujours utiles

`sum(x)`	somme des éléments de `x`
`prod(x)`	produit des éléments de `x`
`max(x)`	maximum des éléments de `x`
`min(x)`	minimum des éléments de `x`
`which.max(x)`	retourne l'indice du maximum des éléments de `x`
`which.min(x)`	retourne l'indice du minimum des éléments de `x`
`range(x)`	idem que `c(min(x), max(x))`
`length(x)`	nombre d'éléments dans `x`
`mean(x)`	moyenne des éléments de `x`
`median(x)`	médiane des éléments de `x`
`var(x)` ou `cov(x)`	variance des éléments de `x` (calculée sur $n-1$) ; si `x` est une matrice ou un tableau de données, la matrice de variance-covariance est calculée
`cor(x)`	matrice de corrélation si `x` est une matrice ou un tableau de données (1 si `x` est un vecteur)
`var(x, y)` ou `cov(x, y)`	covariance entre `x` et `y`, ou entre les colonnes de `x` et de `y` si ce sont des matrices ou des tableaux de données
`cor(x, y)`	corrélation linéaire entre `x` et `y`, ou matrice de corrélations si ce sont des matrices ou des tableaux de données

Dataframe I

Les dataframes sont des ensembles de vecteurs.

```
> x <- 1:10
> y <- x^2
> donnee <- data.frame(absice = x, Ordonnee = y)
> head(donnee)
   absice Ordonnee
1       1        1
2       2        4
3       3        9
4       4       16
5       5       25
6       6       36
> str(donnee)
'data.frame':   10 obs. of  2 variables:
 $ absice   : int  1 2 3 4 5 6 7 8 9 10
 $ Ordonnee: num  1 4 9 16 25 36 49 64 81 100
> dim(donnee)
[1] 10  2
> summary(donnee)
```

Dataframe II

```
     absice          Ordonnee
 Min.   : 1.00   Min.   :  1.00
 1st Qu.: 3.25   1st Qu.: 10.75
 Median : 5.50   Median : 30.50
 Mean   : 5.50   Mean   : 38.50
 3rd Qu.: 7.75   3rd Qu.: 60.25
 Max.   :10.00   Max.   :100.00
> donnee[, 1]
 [1]  1  2  3  4  5  6  7  8  9 10
> donnee$absice
 [1]  1  2  3  4  5  6  7  8  9 10
> donnee[2, ]
  absice Ordonnee
2      2        4
> donnee[5, 1]
[1] 5
> subset(donnee, absice > 7)
```

Dataframe III

```
   absice Ordonnee
8       8        64
9       9        81
10     10       100
> donnee[donnee$absice > 7, ]
   absice Ordonnee
8       8        64
9       9        81
10     10       100
> donnee$absice > 7
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
     TRUE
```

Organiser les données

Le bon fonctionnement de certaines fonctions (e.x. boxplot, aov) requiert une bonne organisation des données.

```
> donnee$nom <- c("tintin", "Milou", "tintin", "Milou", "
    Capitaine")
> str(donnee)
'data.frame':   10 obs. of  3 variables:
 $ absice   : int  1 2 3 4 5 6 7 8 9 10
 $ Ordonnee: num  1 4 9 16 25 36 49 64 81 100
 $ nom      : chr  "tintin" "Milou" "tintin" "Milou" ...
> donnee$nom <- as.factor(donnee$nom)
> is.factor(donnee$nom)
[1] TRUE
> str(donnee)
'data.frame':   10 obs. of  3 variables:
 $ absice   : int  1 2 3 4 5 6 7 8 9 10
 $ Ordonnee: num  1 4 9 16 25 36 49 64 81 100
 $ nom      : Factor w/ 3 levels "Capitaine","Milou",..: 3 2
    3 2 1 3 2 3 2 1
```

Liste

```
> donnee2 <- list(Liste1 = data.frame(x = 1:5, y = (1:5)^2),
      liste2 = 1:6)
> donnee2
$Liste1
  x  y
1 1  1
2 2  4
3 3  9
4 4 16
5 5 25

$liste2
[1] 1 2 3 4 5 6
> donnee2$liste2
[1] 1 2 3 4 5 6
> donnee2[[1]]$x
[1] 1 2 3 4 5
> donnee2[[1]][2, 2]
[1] 4
```

Les valeurs manquantes

- Le symbole d'une valeur manquante est `NA`.
- Les `NA` se propagent...
- `is.na()` permet de les détecter

```
> (x <- c(10, 20, NA, 4, NA, 2))
[1] 10 20 NA  4 NA  2
> sum(x)
[1] NA
> is.na(x)
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE
> sum(x[!is.na(x)])
[1] 36
> sd(x)
[1] NA
> sd(x, na.rm = TRUE)
[1] 8.082904
```

Structures de programmation

Les conditions :

```
> x <- 10
> if (x < 5) {
+     print("1")
+ } else if (x < 9) {
+     print("2")
+ } else {
+     print("3")
+ }
[1] "3"
```

Structures de programmation

Les boucles :

```
> for (i in 1:3) {
+     print(i)
+ }
[1] 1
[1] 2
[1] 3
> i <- 1
> while (i <= 10) {
+     print(i)
+     i <- i + 1
+     if (i >= 3) {
+         break
+     }
+ }
[1] 1
[1] 2
```

Working directory

- définir un dossier de travail
- permet de charger et de sauver des objets sans repréciser l'adresse du dossier

```
> setwd("/media/Data/MyDocu/boulot/presentationR/")
> getwd()
[1] "/media/Data/MyDocu/boulot/presentationR"
```

Lire un fichier de données I

Lire un fichier csv/txt/dat.. :

```
> semis <- read.table("semis.csv", header = T, sep = ";",
    dec = ",")
> str(semis)
'data.frame':   281 obs. of  5 variables:
 $ Plot   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Subplot: int  1 1 1 1 1 1 1 2 2 2 ...
 $ Year   : int  7 7 7 7 7 7 7 7 7 7 ...
 $ species: Factor w/ 4 levels "Fagus sylvatica",..: 4 4 4 4
     1 1 1 4 4 4 ...
 $ Hauteur: num  16.5 13 8.5 7.5 60 128 119 36 34 31 ...
```

Lire un fichier de données II

Il existe une multitude d'autres commandes possibles qui diffèrent surtout par les valeurs par défaut de leurs paramètres par défaut : `read.csv`,`read.csv2`, `read.table("clipboard")`... Ma préférée sous Windows est :

```
> myconn <- odbcConnectAccess("Inventaire_semis_GL_v24.mdb")
> semis <- sqlQuery(myconn, "SELECT * FROM tblSemis;")
```

Enregistrer un fichier de données

Sous forme de fichiers classiques :

```
> write.table(semis, "semis.csv", sep = ";", dec = ",", row.
    names = F)
```

ou directement sous le format R :

```
> save(semis, file = "semis")
> load("semis")
```

Enregistrer un graphique

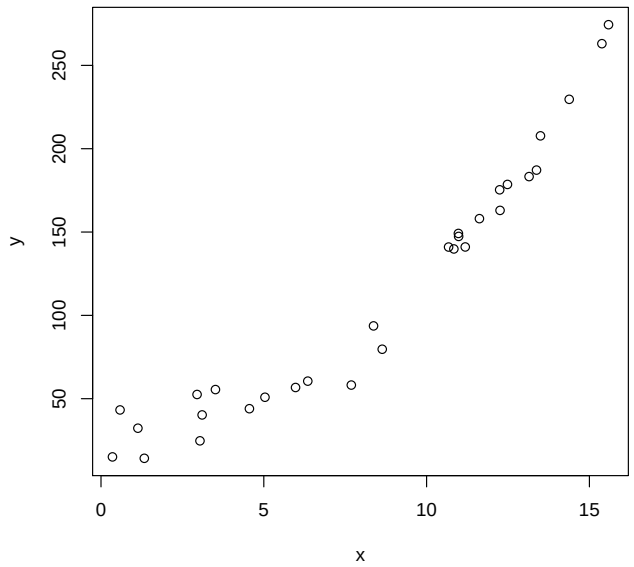
Soit via l'interce graphique, soit en ligne de commande avec les fonctions :

- `jpeg()` utilisable partout mais sans redimensionnement
- `png()` utilisable partout mais sans redimensionnement
- `win.metafile()` meilleur choix avec MS Office
- `pdf()` meilleur choix avec LaTeX

```
> jpeg("rplot.jpg")
> plot(x <- 1:10, y <- x^2)
> dev.off()
```

Premiers graphiques et régressions I

```
> x <- runif(30, 0, 16)
> y <- x^2 + rnorm(n = 30, mean = 25, sd = 10)
> plot(x, y)
```



Premiers graphiques et régressions II

```
> fm1 <- lm(y ~ x)
> fm1

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      -10.94         15.08
```

Premiers graphiques et régressions III

```
> summary(fm1)
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-46.788  -13.489   -4.714   14.724   50.331

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -10.9398     8.7073  -1.256    0.219
x            15.0784     0.9048  16.665 4.56e-16 ***
---
Signif. codes:  0     ***     0.001     **     0.01     *     0.05
        .     0.1         1

Residual standard error: 23.49 on 28 degrees of freedom
Multiple R-squared: 0.9084,     Adjusted R-squared: 0.9051
F-statistic: 277.7 on 1 and 28 DF,  p-value: 4.565e-16
```

Premiers graphiques et régressions IV

```
> summary(fm2 <- lm(y ~ I(x^2)))
Call:
lm(formula = y ~ I(x^2))

Residuals:
    Min      1Q  Median      3Q     Max
-24.367  -8.050   1.244   5.766  19.296

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 24.73152    3.03329   8.153 7.09e-09 ***
I(x^2)       0.97842    0.02533  38.631  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05
        . 0.1  1

Residual standard error: 10.53 on 28 degrees of freedom
Multiple R-squared: 0.9816,     Adjusted R-squared: 0.9809
F-statistic:  1492 on 1 and 28 DF,  p-value: < 2.2e-16
```

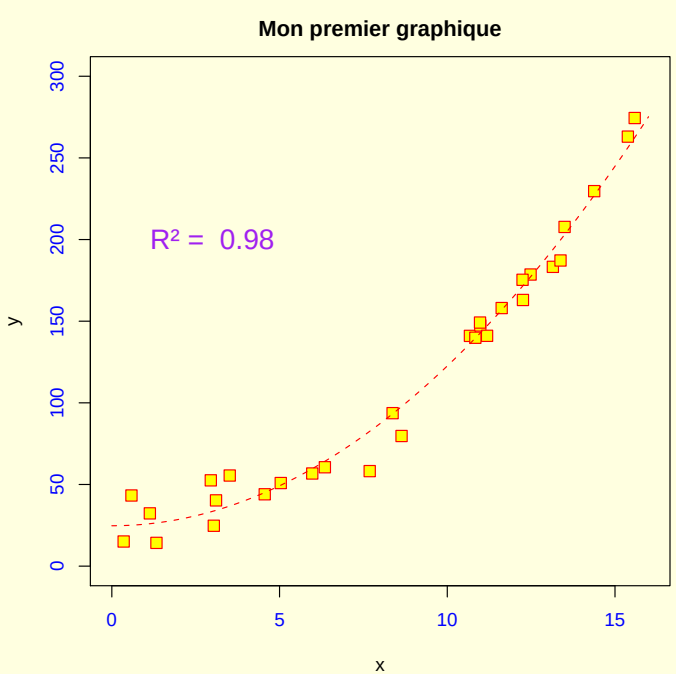
Premiers graphiques et régressions V

```
> fm3 <- lm(y ~ x + I(x^2))
> anova(fm2, fm3)
Analysis of Variance Table

Model 1: y ~ I(x^2)
Model 2: y ~ x + I(x^2)
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     28 3106.9
2     27 2829.9  1    277.05 2.6433 0.1156
```

Premiers graphiques et régressions VI

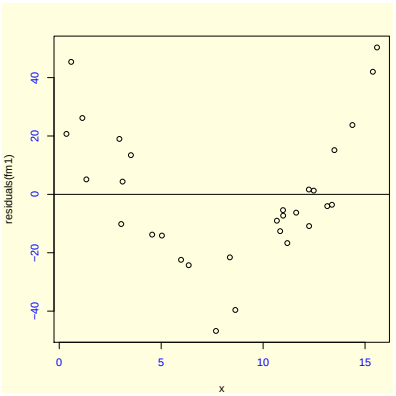
```
> plot(x, y, xlab = "x", ylab = "y", main = "Mon premier
    graphique",
+     xlim = c(0, 16), ylim = c(0, 300), pch = 22, col = "
    red",
+     bg = "yellow", cex = 1.5)
> xtemp <- seq(0, 16, length = 100)
> lines(xtemp, coef(fm2)[1] + coef(fm2)[2] * xtemp^2, col =
    "red",
+     lty = 2)
> text(3, 200, paste("R    = ", round(summary(fm2)$r.squared,
     2)),
+     col = "purple", cex = 1.5)
```

Premiers graphiques et régressions VII

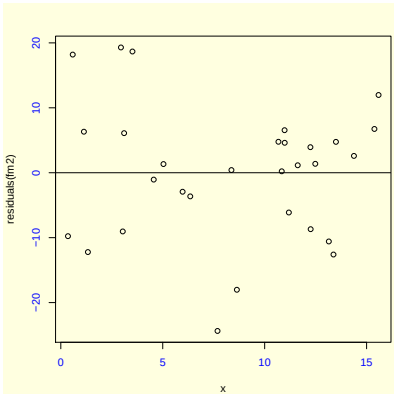


Premiers graphiques et régressions VIII

```
> plot(residuals(fm1) ~ x)
> abline(0, 0)
```

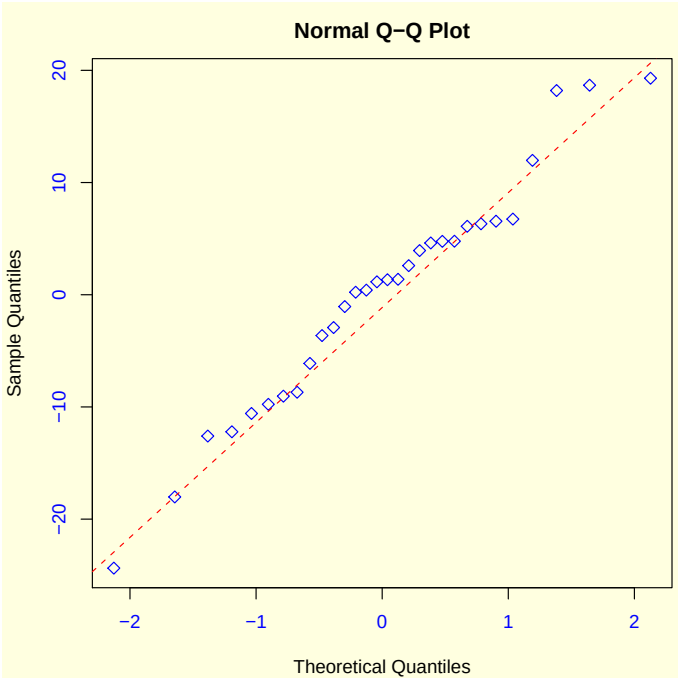



```
> plot(residuals(fm2) ~ x)
> abline(0, 0)
```

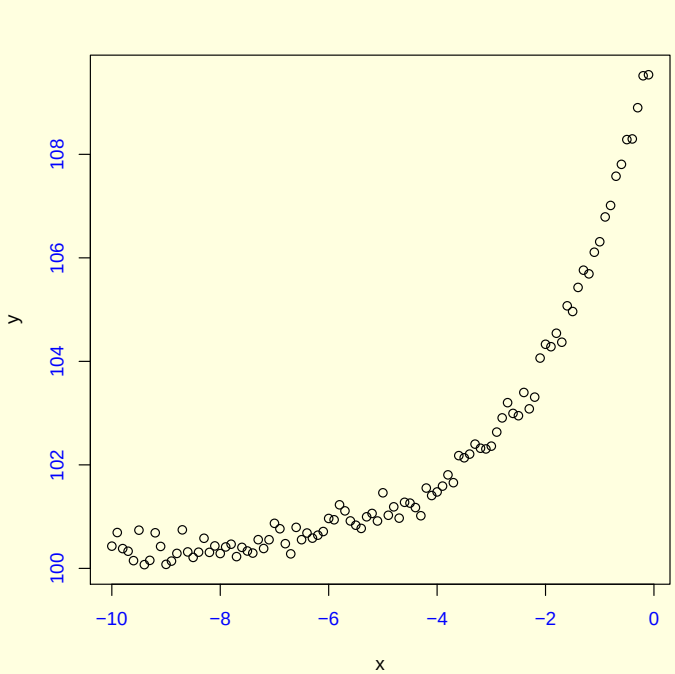


Premiers graphiques et régressions IX

```
> qqnorm(residuals(fm2), col = "blue", pch = 5)
> qqline(residuals(fm2), col = "red", lty = 2)
```



Régression non linéaire I



Soit l'ajustement d'une fonction puissance

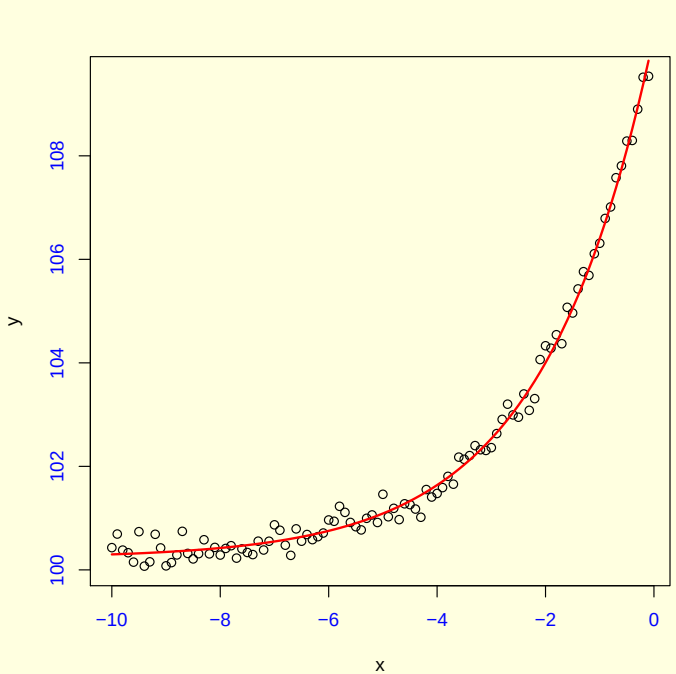
```
> fm1 <- nls(y ~ c + a * exp(b * x), start = list(a = 5, b =
+     0.4,
+     c = 120))
```

Régression non linéaire II

```
> summary(fm1)
Formula: y ~ c + a * exp(b * x)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
a 1.010e+01  8.972e-02  112.55   <2e-16 ***
b 4.916e-01  9.305e-03   52.84   <2e-16 ***
c 1.002e+02  3.829e-02 2617.57   <2e-16 ***
---
Signif. codes:  0     ***     0.001     **     0.01     *     0.05
        .     0.1         1

Residual standard error: 0.1922 on 97 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 5.511e-07
```

Régression non linéaire III

```
> plot(x, y)
> lines(x, predict(fm1), col = "red", lwd = 2)
```



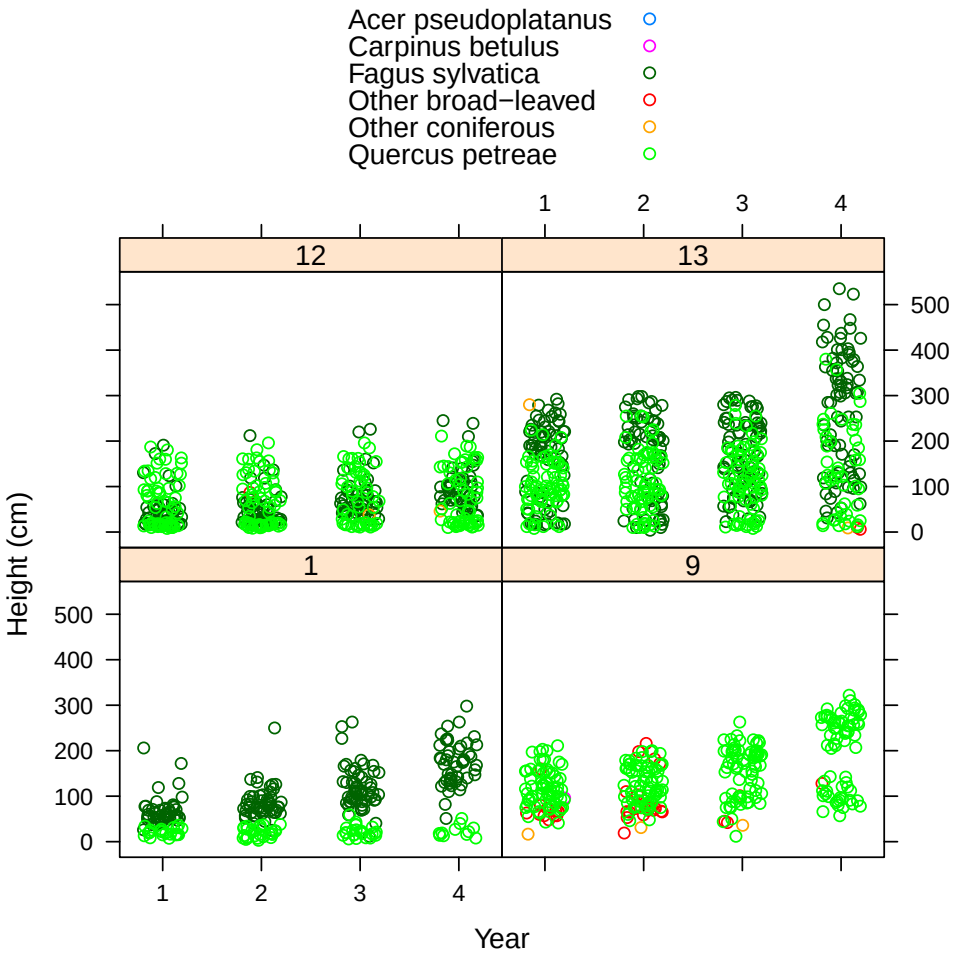
Graphiques treillis I

```
> semis <- read.table("semisDominants.txt", header = T, sep
    = " ",
+     dec = ".")
> str(semis)
'data.frame':   15471 obs. of  5 variables:
 $ Plot   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Subplot: int  1 1 1 1 1 1 1 2 2 2 ...
 $ Year   : int  7 7 7 7 7 7 7 7 7 7 ...
 $ species: Factor w/ 6 levels "Acer pseudoplatanus",..: 6 6
     6 6 3 3 3 6 6 6 ...
 $ Hauteur: num  16.5 13 8.5 7.5 60 128 119 36 34 31 ...
> semis$Plot <- as.factor(semis$Plot)
> semis$Subplot <- as.factor(semis$Subplot)
> semis$Year <- factor(semis$Year, labels = c("07", "08", "
    09",
+     "11"), ordered = T)
> library(lattice)
```

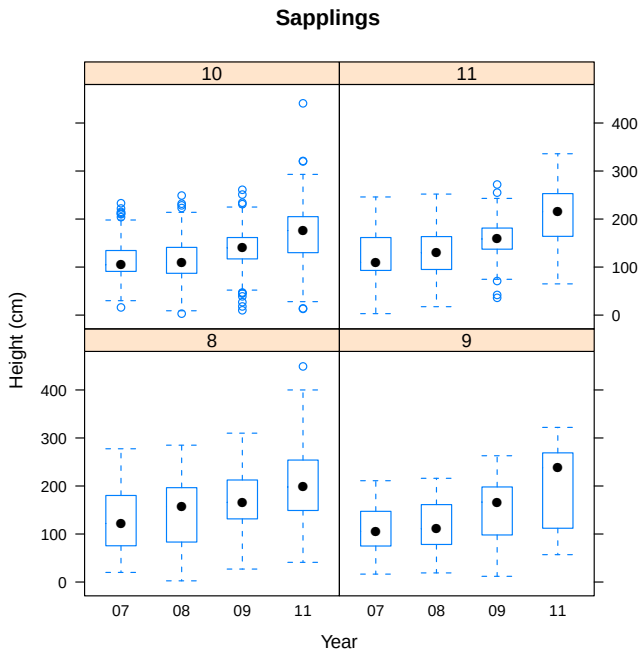
Graphiques treillis II

```
> library(lattice)
> print(xyplot(Hauteur ~ jitter(as.numeric(Year)) | Plot,
    ylab = "Height (cm)",
+     xlab = "Year", group = species, data = semis, subset =
    Plot %in%
+         c(1, 9, 12, 13), type = "p", jitter = T, auto.key
    = T))
```

Graphiques treillis III



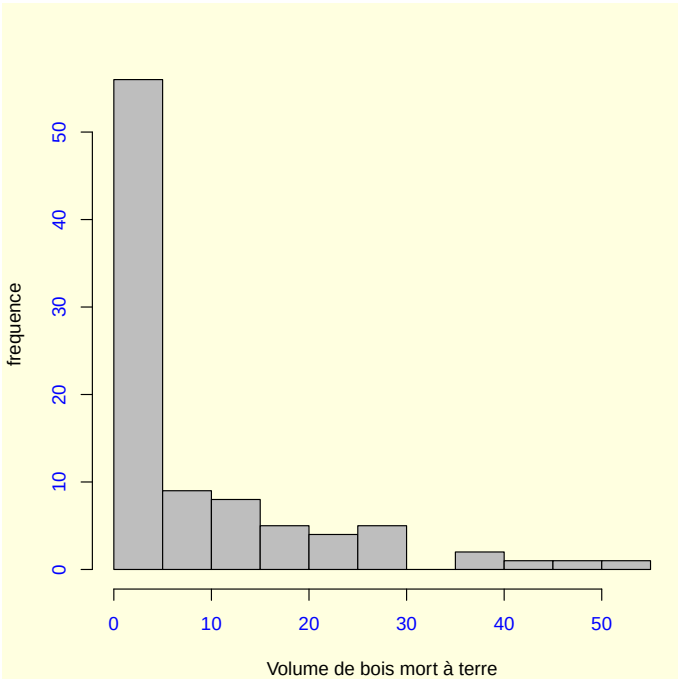
```
> print(bwplot(Hauteur ~ Year | Plot, subset = Plot %in%
    8:11,
+     ylab = "Height (cm)", xlab = "Year", main = "Sapplings
    ",
+     data = semis))
```



Calculer un intervale de confiance bootstrap I

```
> library(boot)
> placette <- read.table("WithoutBMATEST.txt", sep = ";",
    dec = ".",
+     header = T)
> hist(placette$BMATmes, xlab = "Volume de bois mort
    terre",
+     ylab = "frequence", main = NA, col = "grey")
```

Calculer un intervale de confiance bootstrap II

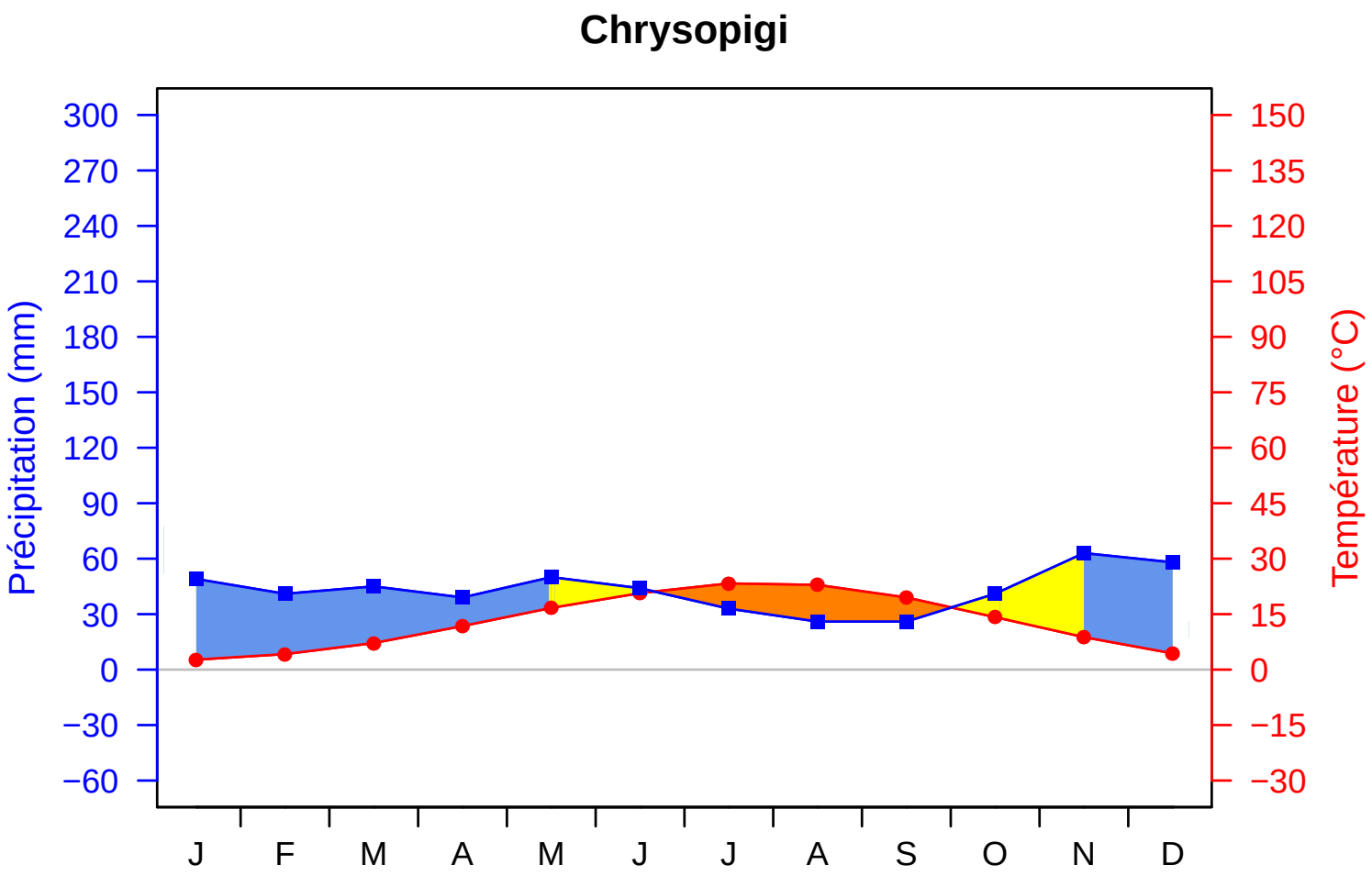



```
> mean(placette$BMATmes)
[1] 8.576304
> samplemean <- function(x, d) return(mean(x[d]))
> bmean <- boot(placette$BMATmes, samplemean, R = 1000)
> boot.ci(bmean)
```

Calculer un intervale de confiance bootstrap III

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = bmean)

Intervals :
Level      Normal              Basic
95%   ( 6.298, 10.986 )   ( 6.158, 10.857 )

Level      Percentile            BCa
95%   ( 6.296, 10.994 )   ( 6.605, 11.363 )
Calculations and Intervals on Original Scale
```

Graphiques ombrothermiques



R, Sweave, LaTeX, OOO

Ce document à été rédigé sous LaTeXavec les packages Beamer et Sweave. (Sweave fonctionne aussi avec Open Office mais pas avec MS Office) :

```
\frame[containsverbatim]{
 \frametitle{Première utilisation}
 \begin{itemize}
    \item \verb>@ est le prompt @
 \item ...
 \end{itemize}
 << >>=
 1+3
 pi
 exp(32)
 sin(2*pi)
 @
 }
```

Documentation sur le Web

- Documentation officielle `http://cran.r-project.org/`
- Quick R : `http://www.statmethods.net/`
- R by examples : `http://www.mayin.org/ajayshah/KB/R/index.html`
- R pour les débutants : `http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf`