Introduction au logiciel R

Camelia Goga et Catherine Labruère

IMB, Université de Bourgogne
camelia.goga@u-bourgogne.fr, clabruer@u-bourgogne.fr

Ecole Doctorale Dijon-2009

Plan du cours

1. Introduction et objectifs du cours
2. Créer et manipuler des données
3. Data-frame
4. Graphiques
5. Un peu de programmation : créer une fonction, boucles et exécutions conditionnels ;
6. Quelques analyses statistiques avec R :
   - statistique descriptive : représentations graphiques des données (boxplot, camembert, diagramme en barres et colonnes, histogramme)
   - simulation des observations suivant des lois classiques : Bernoulli, binomiale, normale ...
   - le modèle linéaire

1. Introduction et objectifs du cours

- Ce cours est une introduction au logiciel **R**.
- **R** est un logiciel libre distribué par 'GNU Public Licence' et dérivé du langage **S** (le logiciel S-PLUS).
- Il présente des caractéristiques remarquable comme la possibilité d'effectuer du calcul matriciel et d'autres opérations complexes, du stockage et de manipulation des données.
  R contient des nombreuses fonctions pour les analyses statistiques et des outils graphiques flexibles.
- **R** s'adresse à un large public formé de spécialistes et non-spécialistes en informatique.
- Bibliographie :
  - "An Introduction to R"
    (http ://cran.r-project.org/doc/manuals/R-intro.pdf.)
  - "R pour les débutants" , Emmanuel Paradis
    (http ://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf)
  - ...

Installation et démarrage de **R**

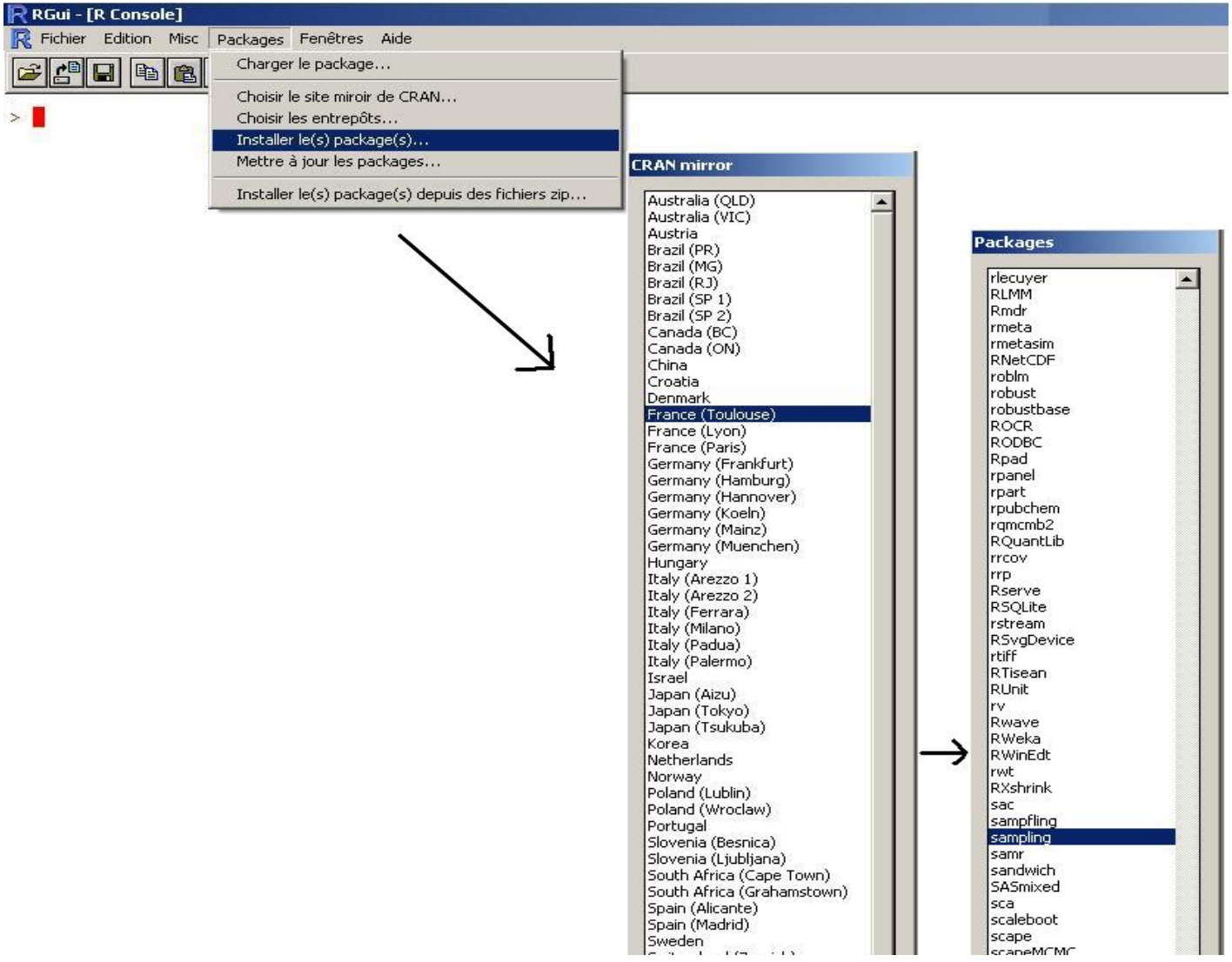
Sur le site : http ://cran.r-project.org/

Download and Install R en choisissant l'interface souhaitée (Windows, MacOS, Linux) et suivant les instructions ; le package de base est installé et souvent on a besoin de compléter avec des packages plus spécialisés.

Pour démarrer, on click sur le butons **R** et le prompteur $>$ apparaît (attente d'une commande).

Pour quitter, on tape `q()`.

2. Télécharger des packages



L'aide en ligne

Sur le site de **R** : http ://cran.r-project.org/

- l'aide en ligne sur une fonction avec `?fonction` or `help.search("function")` ;
  exemple : `?plot` ;
  `fonction` : nous donne les lignes du programme ;
- `find("  " )` and `apropos("  ")` ;

On peut avoir des exemples et démonstrations avec `exemple()` or `demo()`.
`exemple(lm)`, `demo(graphics)`
la fonction `ls()` pour lister les noms des objets de la mémoire ;
la fonction `rm()` pour effacer un objet de la mémoire.

2. Créer et manipuler des données

1. les objets
2. lire des données à partir d'un fichier
3. enregistrer les données
4. générer des données (de façon aléatoire ou non)
5. manipuler les objets
   - accéder à une valeur particulière d'un objet
   - les fonctions arithmétiques de base
   - calcul matriciel

2.1 Les objets

R travaille avec des `objets` qui ont deux attributs :

- *mode* : pour donées, on a numéric, character, complex, logical ;
  sinon, il existe aussi le mode function, expression, formula.
- *length* : nombre d'éléments de l'objet ;

objet	mode possibles	modes possibles ?
`vector`	numeric, character, complex ou logical	non
`factor`	numeric ou character	non
`matrix`	numeric, character, complex ou logical	non
`array`	numeric, character, complex ou logical	non
`data-frame`	numeric, character, complex ou logical	oui
`list`	numeric, . . ., logical, function, expression, formula	oui

2.2 Lire des données

Les fonctions

- `read.table()` (avec deux variantes `read.csv()` et `read.csv2()`)
  Ce sont les plus utilisées.

```
> rec=read.csv("rec99.csv") ##on lit le fichier en form
> rec ####pour faire defiler les donnees
```

- `scan()`
- `read.fwf()`

read.table(file, header = FALSE, sep = "", quote = "'"", dec = "."', row.names, col.names, as.is
= !stringsAsFactors,na.strings="NA")
read.csv(file, header = TRUE, sep = ",", quote="'"", dec="."', fill = TRUE, comment.char="", ...)
read.csv2(file, header = TRUE, sep = ";", quote="'"", dec=",", fill = TRUE, comment.char="", ...)

2.3 Enregistrer des objets

- Si on veut quitter **R**, le message suivant est affiché :

```
> q()
> Save workspace image? [y/n/c]:
```

alors, tous les objets sont sauvés dans un fichier spécifique **R**, .Rdata qui sera chargé automatiquement à l'ouverture d'une nouvelle session.

- enregistrer les data-frames avec `write.table()` or `write.csv()` :

write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "/n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"))

```
rec1=rec[,-3] ##on enleve la troisieme colonne
write.csv(rec1,file="rec1.csv")###on sauve dans un format csv;
read.csv("rec1.csv") ###pour la lire
```

- enregistrer les objets (data-frame, fonctions ...) avec `save()` `save.image()` :

save.image(file = ".RData", version = NULL, ascii = FALSE, compress = !ascii, safe = TRUE)

- Les données (ou `image`) sont chargées avec `load()` et listés avec `ls()` :

load(file, envir = parent.frame())

- On va créer et sauver un objet :

```
m <- stats::runif(20)
n <- list(a = 1, b = TRUE, c = "oops")
save(m, n, file = "mn.Rdata")
save.image()
```

- On quitte **R**, on relance et on liste les objets de la mémoire :

```
ls()
 [1] "A"          "B"          "BE"         "BE2"        "Bbase"      "Bs1"        "Fage"
 [8] "N"          "RB.ht"      "RB.spl"     "RMSE.ht"    "RMSE.spl"   "T"          "X"
[15] "Y"          "a"          "age"        "bin"        "breaks"     "classes"    "counts"
[22] "csp"        "delta"      "difage"     "distrcsp"   "e"          "ech.si"     "effect"
[29] "effectifs"  "fNpop"      "freq"       "freqcum"    "fypop"      "g"          "g1s"
[36] "g2s"        "g3s"        "g4s"        "g5s"        "gmod"       "hautpiec"   "i"
[43] "invT"       "knot"       "n"          "nb.simul"   "nknot"      "numeros"    "order"
[50] "pik"        "pop"        "s"          "s.be"       "s.be2"      "s1"         "sample"
[57] "si.veolia"  "sn"         "t"          "tXU"        "thetaNs1"   "thetays1"   "totxsample"
[64] "tx.ht"      "txu"        "ty.ht"      "ty.reg"     "u"          "uniforme"   "veolia"
[71] "w"          "x"          "xs1"        "xsample"    "y"          "yHT"        "ybar.ht"
[78] "ybar.spl"   "yhat"       "ys1"        "z"
> BE
function(N,pi)
{x=runif(N)
y=as.numeric(x<pi)
y
}
> m
Erreur : objet "m" non trouvé
####on change le repertoire de travail
> load("mn.Rdata")
>  m
 [1] 0.986927597 0.723471483 0.634258648 0.498725933 0.466320642 0.673727544 0.086547065
 [8] 0.099362703 0.377084813 0.007601145 0.558583773 0.089954030 0.559246394 0.470739615
[15] 0.395547755 0.699756759 0.969284647 0.902235428 0.118319487 0.653940664
```

2.4 Générer des données (non aléatoires)

On déclare nos données sous forme matricielle (vecteur ligne ou matrice) dans plusieurs façons :

- la fonction `c()` pour un vecteur (de type ligne) :

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

Pour un vecteur colonne, il faut utiliser la fonction `as.matrix()` :

```
> as.matrix( c(1,2,3,4))
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
```

- l'opérateur `:` (prioritaire sur les opérations au sein d'une expression)

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

- la fonction `seq` de différents arguments :
  `seq(from, to)` :from le début de la séquence, to pour la fin :

```
> seq(from=1, to=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=1.5, to=4.7)
[1] 1.5 2.5 3.5 4.5
```

- `seq(from, to, by= )` : la même chose et by pour le pas ;

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9
```

- `seq(from, to, length.out=)` : il créé une séquence de from jusqu'au to de la longueur length

```
> seq(from=1, to=10, length=3)
[1]  1.0  5.5 10.0
```

D'autres fonctions :

- `rep ()` :

```
> rep(1,5)
[1] 1 1 1 1 1
```

- `expand.grid()` : un data-frame avec toutes les combinaisons des arguments ;

```
>  expand.grid(h=seq(60,70,10),
 w=seq(100,300,100),
 sex=c("Male", "Female"))
```

	h	w	sex
1	60	100	Male
2	70	100	Male
3	60	200	Male
4	70	200	Male
5	60	300	Male
6	70	300	Male
7	60	100	Female
8	70	100	Female
9	60	200	Female
10	70	200	Female
11	60	300	Female
12	70	300	Female

Matrices

La fonction `matrix()` : remplissage par colonne par défault :

`matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)`

`data` : les données pour remplir la matrice ;

`nrow` : le nombre de lignes et `ncol` le nombre de colonnes ;

`dimnames` : une list de dimension 2 avec les noms des lignes et/ou colonnes

```
matrix(data=1:6,nr=2,nc=3,
dimnames=list(c("row1", "row2"), c("C.1", "C.2", "C.3")))
     C.1 C.2 C.3
row1   1   3   5
row2   2   4   6
```

Matrices (2)

Créer une matrice à partir des vecteurs déjà créés :
les fonctions `cbind()` (par colonne) et `rbind()` (par ligne)

```
> (alfa=seq(1,5))
[1] 1 2 3 4 5
> (beta=seq(1,10,2))
[1] 1 3 5 7 9
> (A1=cbind(alfa,beta))
     alfa beta
[1,]    1    1
[2,]    2    3
[3,]    3    5
[4,]    4    7
[5,]    5    9
> (A2=rbind(alfa,beta))
      [,1] [,2] [,3] [,4] [,5]
alfa     1    2    3    4    5
beta     1    3    5    7    9
```

2.5 Manipuler les objets : accéder à une valeur particulière d'un objet

Cette opération est réalisée à l'aide des crochets : `[ ]`

- pour un vecteur : `x[3]` la troisième composante de x ;
- pour une matrice ou data-frame : `A[2,3]` l'élément se trouvant sur la 2ème ligne et la 3ème colonne ;
  `A[2,]` : la 2ème ligne et `A[,3]` la 3ème colonne ;
- accéder à l'aide d'une opération logique : $>$, $<$, $<=$ (inférieur ou égal), $>=$ (supérieur ou égal), $==$ (égal), $!=$ (différent)

```
x=c(2,6,10,60,34)
x<=10
[1]  TRUE  TRUE  TRUE FALSE FALSE
x[x<=10]
[1]  2  6 10
```

- pour enlever un élément ou une ligne (resp. colonne) on utilise '-' : `A[-2,]` on enlève la 2ème ligne ;

Calcul arithmétique et fonctions simples

Les opérations suivantes sont effectuées composante par composante :

- la somme des deux vecteurs avec $+$;
- le produit avec * ;
- le rapport avec $/$;
- puissance avec ^ ;

Pour avoir le produit scalaire $x^t \cdot y = \sum x_k y_k$ entre deux vecteurs x et y, on déclare un des vecteurs comme une matrice :

```
y%*%as.matrix(x)
```

```
x=c(3,5,3)
y=c(5,2,8)
y%*%as.matrix(x)
     [,1]
[1,]   49
```

Calcul arithmétique et fonctions simples (2)

- `sum(x)` : sommes des composantes de x ;
- `prod(x)` : produit des composantes de x ;
- `max(x)` : maximum des composantes de x ;
- `which.max(x)` : retourne l'indice du maximum des composantes de x ;
- `range(x)` : idem que `c(min(x),max(x))` ;
- `length(x)` : nombre d'éléments dans x ;
- `mean(x)` : la moyenne des éléments dans x ;
- `median(x)` : la médiane des éléments dans x ;
- `var(x)` : la variance (corrigée) des éléments dans x ;
- `cor(x)` : matrice de corrélation si x est un data-frame et 1 sinon ;
- `cov(x,y)` : la covariance entre x et y ;

Le résultat est une valeur, sauf pour `range()`, `var()`, `cor()` et `cov()`.

Calcul arithmétique et fonctions simples (3)

- `round(x,n)` : arrondi les éléments de x à n chiffres après la virgule ;
- `rev(x)` : inverse l'ordre de x ;
- `sort(x)` : trie les éléments de x dans ordre croissante ;
- `rank(x)` : rangs des éléments de x ;
- `cumsum(x)` : un vecteur avec les sommes cumulées de composantes de x;
- `cumprod(x)` : idem pour le produit ;
- `table(x)` : retourne un tableau avec les effectifs de différentes valeurs de x;
- `which(x==a)` : retourne un vecteur des indices de x pour lesquels l'opération de comparaison est vraie ;

```
x=c(2,6,10,60,34)
which(x<35)
[1] 1 2 3 5 ##les indices pour lesquels on a vrai la comparaison
x[which(x<35)]
[1]  2  6 10 34
```

Calcul matriciel

- La somme de deux matrices est réalisée avec l'opérateur "+" ;
- Le produit matriciel usuel avec '%*%' mais il est possible de faire le produit composante par composante avec '*' :

```
> matrix(data=5,nr=2,nc=2)
     [,1] [,2]
[1,]    5    5
[2,]    5    5
> matrix(data=7,nr=2,nc=2)
     [,1] [,2]
[1,]    7    7
[2,]    7    7
```

```
> matrix(data=5,nr=2,nc=2)*
matrix(data=7,nr=2,nc=2)
     [,1] [,2]
[1,]   35   35
[2,]   35   35
> matrix(data=5,nr=2,nc=2)%*
%matrix(data=7,nr=2,nc=2)
     [,1] [,2]
[1,]   70   70
[2,]   70   70
```

Calcul matriciel (2)

- La transposé d'une matrice : la fonction `t()` ; ça marche aussi avec un data frame ;
- Pour extraire la diagonale d'une matrice : la fonction `diag()` ;
- Le déterminant d'une matrice : la fonction `det()` ;
- Pour calculer l'inverse d'une matrice : `solve(A)`
- Les valeurs et vecteurs propres d'une matrice : `eigen(A)`
- Pour résoudre un système d'équations : $Ax = b$ on utilise la fonction `solve()` comme suit :

```
solve(A,b)
```

Un peu plus sur les data frame

Un data frame est un objet avec des lignes et des colonnes, un peu comme une matrice, dont les éléments peuvent être de type numérique, qualitative, logique (une matrice contient que des nombres) ;
On peut avoir des data-frame :
par importation et sauvegarde des données externes (`read.table`, `read.csv`) or
par transformation avec la fonction `data.frame()` :

```
 x=runif(10)
 y=letters[1:10]
 z=sample(c(rep('a',5),rep('b',5)))
 new=data.frame(y,z,x)
new
   y z         x
1  a a 0.9585292
2  b a 0.4200847
3  c b 0.5208779
4  d a 0.7026489
5  e a 0.7759345
6  f b 0.0458354
7  g a 0.7018125
8  h b 0.7001671
9  i b 0.3831451
10 j b 0.2988668
```

Divers fonctions sur un data frame

Nous allons considérer le data frame obtenu après la lecture de `rec99.csv` :

```
> rec=read.csv('rec99.csv')
> rec[1:5,] ###les 5 premieres de rec
  CODE_N      COMMUNE BVQ_N POPSDC99 LOG LOGVAC stratlog
1  31014     ARGUENOS 31020       57  94      1        1
2  31131    CAZAUNOUS 31020       47  56      4        1
3  31348      MONCAUP 31020       26  57      2        1
4  31447  RAZECUEILLE 31020       37  89      6        1
5  31140 CHEIN-DESSUS 31020      184 174     28        2

> summary(rec)  ###un résumé du rec
 CODE_N             COMMUNE        BVQ_N          POPSDC99           LOG
 Min.   :31001   MONTESQUIEU-:  3   Min.   :31020   Min.   :   8.0   Min.   :  12.0
 1st Qu.:31152   MONTGAILLARD:  3   1st Qu.:31106   1st Qu.: 112.5   1st Qu.:  66.0
 Median :31302   CASTILLON-DE:  2   Median :31239   Median : 265.0   Median : 137.0
 Mean   :31299   LA SALVETAT-:  2   Mean   :31298   Mean   : 831.0   Mean   : 356.2
 3rd Qu.:31448   MARIGNAC-LAS:  2   3rd Qu.:31483   3rd Qu.: 858.2   3rd Qu.: 364.0
 Max.   :31593   SAINT-PIERRE:  2   Max.   :31584   Max.   :8733.0   Max.   :4490.0
                 (Other)     :540
> names(rec) ## les noms des variables
[1] "CODE_N"   "COMMUNE"  "BVQ_N"    "POPSDC99" "LOG"      "LOGVAC"   "stratlog"
> is.data.frame(rec)
[1] TRUE
> dim(rec)
[1] 554   7
```

- extraction des lignes ou colonnes de `rec` :

```
> rec[1,] ##extraction de la premiere ligne
  CODE_N  COMMUNE BVQ_N POPSDC99 LOG LOGVAC stratlog
1  31014 ARGUENOS 31020       57  94      1        1
############################
> rec[,"LOG"] ##extraction de la colonne ou la variable LOG
> rec[,5] ##la même chose
> rec[1:5, "LOG"] ###les cinq premieres lignes pour la colonne LOG
rec[1:5, "LOG"]
[1]  94  56  57  89 174
############################
> rec2=rec[,c(5,6)]###extraction des colonnes 5 et 6
> rec2[1:5,] ###extraction des 5 premiers lignes de rec2

LOG LOGVAC
1  94      1
2  56      4
3  57      2
4  89      6
5 174     28
################################elimination de colonne
rec1=rec[,-3]###pour enlever on doit utiliser le numero de colonne
> rec1[1:5,]
  CODE_N      COMMUNE POPSDC99 LOG LOGVAC stratlog
1  31014     ARGUENOS       57  94      1        1
2  31131    CAZAUNOUS       47  56      4        1
3  31348      MONCAUP       26  57      2        1
4  31447  RAZECUEILLE       37  89      6        1
5  31140 CHEIN-DESSUS      184 174     28        2
##on enleve la var BVQ_N
```

- extraction aléatoire des lignes

```
rec3=rec[sample(554,5),] ###selection aleatoire de 5 lignes
rec3
    CODE_N      COMMUNE BVQ_N POPSDC99  LOG LOGVAC stratlog
29   31256 LABRUYERE-DO 31033      143   57      4        1
164  31280      LATRAPE 31107      297  173     31        2
265  31333     MAUVEZIN 31239       48   36      6        1
421  31513 SAINT-PLANCA 31483      372  213     14        2
326  31253 LABASTIDETTE 31395     1328  475     10        3
```

- conditions logiques pour extraire des lignes

```
> attach(rec)
> rec[stratlog==3,] ### on garde que les lignes correspondantes au stratlog=3
> detach(rec)
############################
rec[-which(rec$LOG<100),] ###which est utile quand on veut enlever des lignes
############################
rec[,7]=as.factor(rec[,7])###on oblige la variable stratlog a etre une variable qualitative
rec4=rec[,sapply(rec,is.factor)]###on garde que la variable stratlog
rec4[1:5,]
       COMMUNE stratlog
1     ARGUENOS        1
2    CAZAUNOUS        1
3      MONCAUP        1
4  RAZECUEILLE        1
5 CHEIN-DESSUS        2
```

La fonction order pour exécuter des tris

```
> rec[1:10,]
 CODE_N      COMMUNE BVQ_N POPSDC99 LOG LOGVAC stratlog
1   31014     ARGUENOS 31020       57  94      1        1
2   31131    CAZAUNOUS 31020       47  56      4        1
3   31348      MONCAUP 31020       26  57      2        1
4   31447  RAZECUEILLE 31020       37  89      6        1
5   31140 CHEIN-DESSUS 31020      184 174     28        2
6   31174     ESTADENS 31020      425 274      1        2
7   31245 JUZET-D'IZAU 31020      193 153      6        2
8   31342       MILHAS 31020      140 150     10        2
9   31544  SENGOUAGNET 31020      220 239     15        2
10  31020        ASPET 31020      923 684     90        3

> orderstrat=order(rec$stratlog)
> rec.orderstrat=rec[orderstrat,]
> rec.orderstrat[1:12,]
   CODE_N      COMMUNE BVQ_N POPSDC99 LOG LOGVAC stratlog
1   31014     ARGUENOS 31020       57  94      1        1
2   31131    CAZAUNOUS 31020       47  56      4        1
3   31348      MONCAUP 31020       26  57      2        1
4   31447  RAZECUEILLE 31020       37  89      6        1
11  31039       BACHAS 31028       75  42      2        1
12  31063       BENQUE 31028      154  72      5        1
13  31086       BOUZIN 31028       69  35      5        1
14  31134 CAZENEUVE-MO 31028       54  33      2        1
15  31168         EOUX 31028      121  58      7        1
16  31172     ESPARRON 31028       36  29      5        1
17  31386 MONTOULIEU-S 31028      153  73      6        1
18  31414    PEYRISSAS 31028       82  42      5        1
```

Les fonctions `tapply` and `merge`

- La fonction `tapply` est utilisée pour faire moyennes, variance, ... par groupes.

```
tapply(LOGVAC,stratlog,sd)
        1         2         3         4
 3.326514  6.865199 21.441989 64.684940
```

  On calcule pour chaque modalité de la variable `stratlog`, l'écart-type de la variable `LOGVAC`.

- La fonction `merge` est utilisée pour coller deux data-frame.

4. Graphiques

Possibilité de voir des exemples de graphiques avec `demo(graphics)` ou `demo(persp)`.
Lorsqu'une fonction graphique est tapée sur la console, une fenêtre graphique va s'ouvrir avec le graphe demandé.

Partitionner une fenêtre grapfique

- `par(mfcol=c(nr,nc))` : on partitionne la fenêtre en une matrice de nr lignes et nc colonnes et le remplissage est réalisé par colonne ;
- `mfrow` : idem mais les graphiques sont dessinés en ligne ;
- `layout()` : pour des partitions plus complexes

```
> layout(matrix(c(1,2,3,4),2,2))
#### pour inserer dans un graphique
> layout(matrix(c(1,1,2,1),2,2),c(3,1),c(1,3))
> layout.show(2)###et visualiser la partition creee
```

Les fonctions graphiques

- `plot (x)` : graphe des valeurs de x (sur l'ordonnée) en fonction des valeurs de x ;
- `plot (x,y)` : graphe des valeurs de y (l'ordonnée) en fonction des valeurs de x ;
- `pie(x)` : "camembert" des valeurs de x ;
- `boxplot(x)` : boxplot de x ;
- `hist(x)` : histogramme de x (pour x quantitative) ;
- `borplot(x)` : diagramme en colonnes (pour x qualitative)

Pour chaque fonction, on a plusieurs options mais certaines sont communes :
type : "p" : points, "l" : lignes, "b" les deux, "h" : lignes verticales, "s" : escaliers ; for stair steps ;
xlab, ylab : noms des axes, variables caractères entre "" ;
main : title, variable de type caractère ; sub : sous-titre ;

Les fonctions graphiques (2)

- `points(x,y)` : ajoute des points ;
- `lines(x,y)` : idem mais avec des lignes ;
- `segments(x0,y0,x1,y1)` : trace une ligne entre les points (x_0, y_0) et (x_1, y_1)
- `abline(a,b)` : trace une ligne de pente b et ordonnée à l'origine a ;
- `legend(x,y,legend)` : : ajoute une légende au point de coordonnées (x, y) avec les symboles donnés par legend

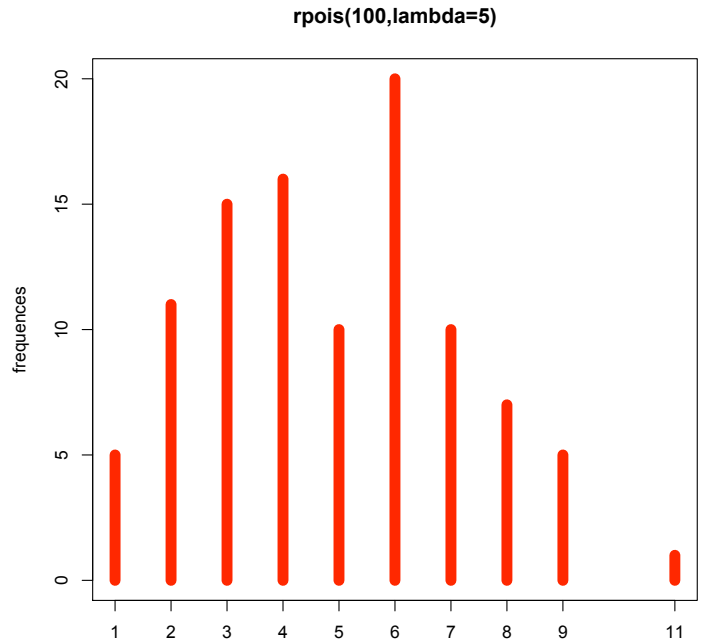
Les paramètres graphiques

Un petit résumé des paramètres graphiques les plus utilisés (68 en tout) ; la liste détaillée est donnée par `?par`.

- `bg` : couleur de l'arrière plan ;
- `cex` : pour contrôler la taille des caractères et des symboles ; options `cex.axis`, `lab`, `cex.title`, `cex.sub`
- `col` : pour contrôler la couleur des symboles ;
- `font` : un entier pour le style du texte ; 0 : normal, 1 : italique, 2 : gras, 3 : gras italique ;
- `lty` : pour contrôler le type de ligne tracée ; 1 : continue, 2 : tirets, 3 : points, 4 : points et tirets alternés, 5 : tirets longs, 6 : tirets courts et longs alternés ;
- `lwd` : une valeur pour contrôler la largeur des lignes.

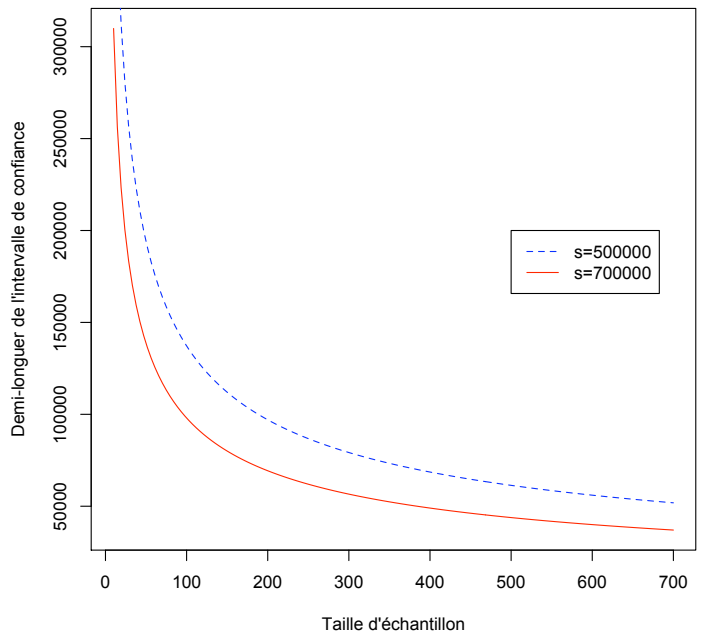
Exemples : 1

```
plot(table(rpois(100,5)), type = "h",
col = "red", ylab="frequences",lwd=10,
     main="rpois(100,lambda=5)")
```



Exemples : 2

```
n=seq(from=10,to=700,length=150)
s1=500000
s2=700000
f1=1.96*s1/sqrt(n)
f2=1.96*s2/sqrt(n)
plot(n,f1,type="l", xlim=c(min(n),max(n)),col="red",xlab="Taille d'échantillon"
lines(n,f2,ylim=c(min(f2),max(f2)),type="l",col="blue",lty=2)
legend(500,200000,c("s=500000","s=700000"),col=c("blue","red"),lty=c(2,1))
```



Un peu de programmation

- Boucles et exécutions conditionnels :
  if(cond) expr
  if(cond) cons.expr else alt.expr
  for(var in seq) expr
  while(cond) expr
  repeat expr

```
> N=10; pi=0.3
> x=runif(N)
> y=rep(1,N)
> for (i in 1:N) {if (x[i]<pi) y[i]=1  else y[i]=0}
```

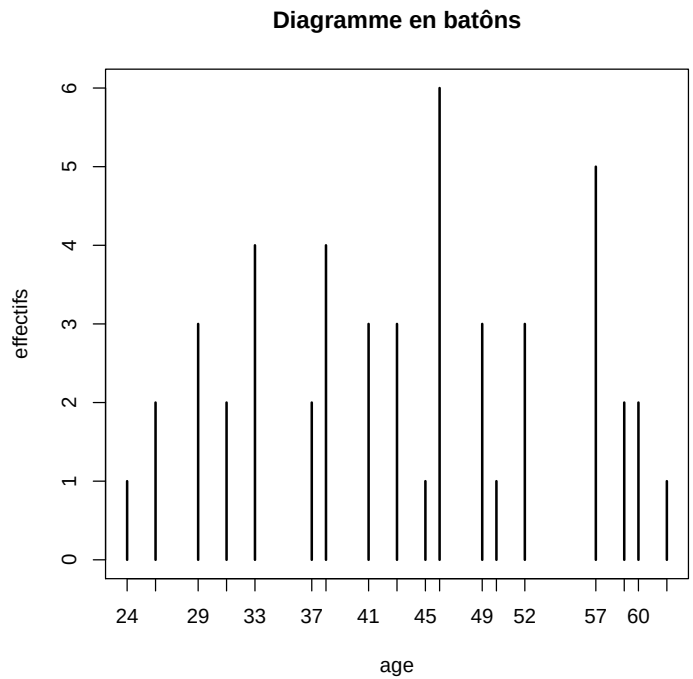
- Ecrire une fonction : function(), les paramètres de la fonction

```
> BE=function(N,pi)##########
{x=runif(N)
y=rep(1,N)
for (i in 1:N) {if (x[i]<pi) y[i]=1  else y[i]=0}
y
cat("echantillon Bernoulli est",y)
}
> BE(10,0.3)
echantillon Bernoulli est  0 1 0 0 0 0 0 0 0 0
```

Statistique descriptive : représentations graphiques

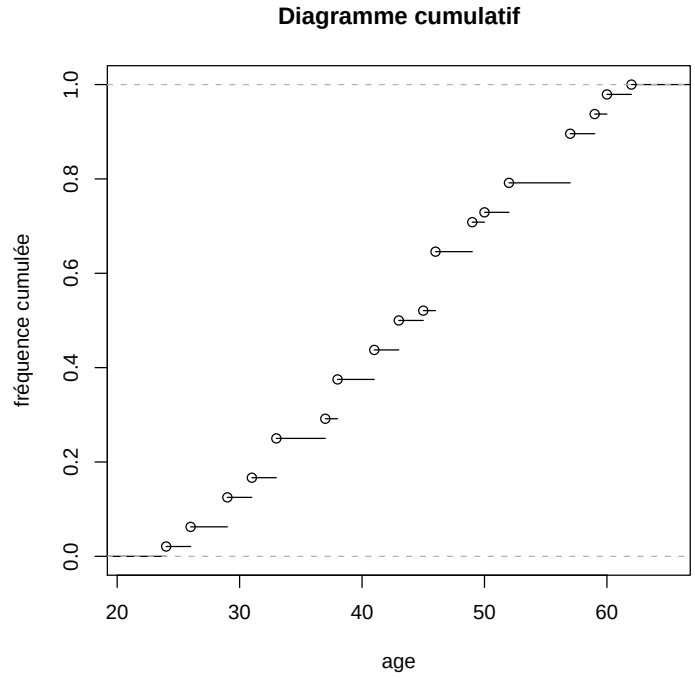
Variable quantitative discrète : une variable qui prend un petit nombre de valeurs ;

```
> age=c(43, 29, 57, 45, 50, 29, 37, 59, 46, 31, 46, 24, 33, 38, 49,
 31, 62, 60, 52, 38, 38, 26, 41, 52, 60, 49, 52, 41, 38, 26, 37, 59, 57,
 41, 29, 33, 33, 43, 46, 57, 46, 33, 46, 49, 57, 57, 46, 43)
####### les effectifs
> effectifs=table(age)
effectifs
> age
24 26 29 31 33 37 38 41 43 45 46 49 50 52 57 59 60 62
 1  2  3  2  4  2  4  3  3  1  6  3  1  3  5  2  2  1
> plot (effectifs, type='h')
```

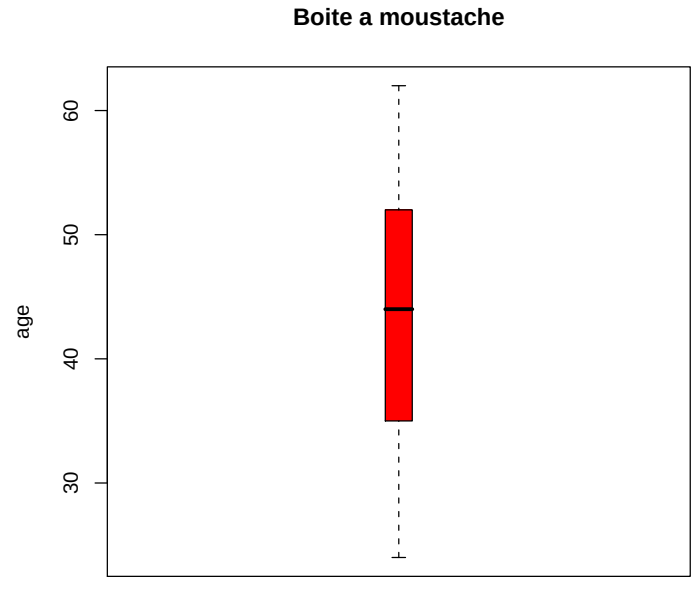


Fonction de répartition empirique et box-plot

```
plot (ecdf(age), xlab="age",
ylab="effectif cumulé",
main="Diagramme cumulatif")
```



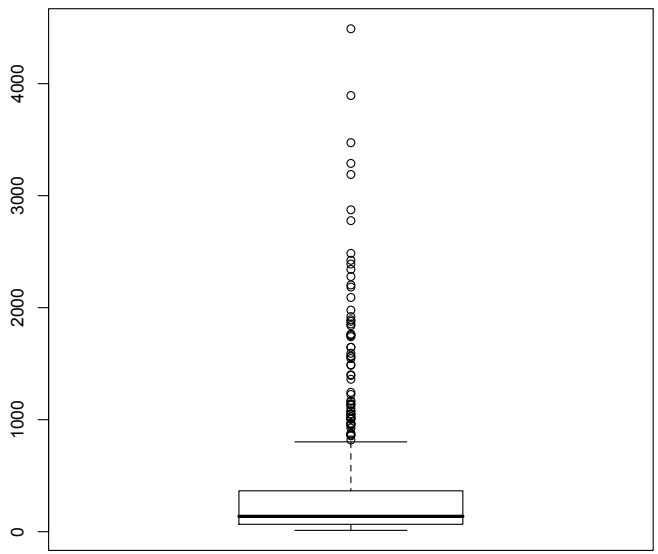

```
boxplot (age, ylab="age",
col="red", main="Boite a
moustache", boxwex=0.1)
```

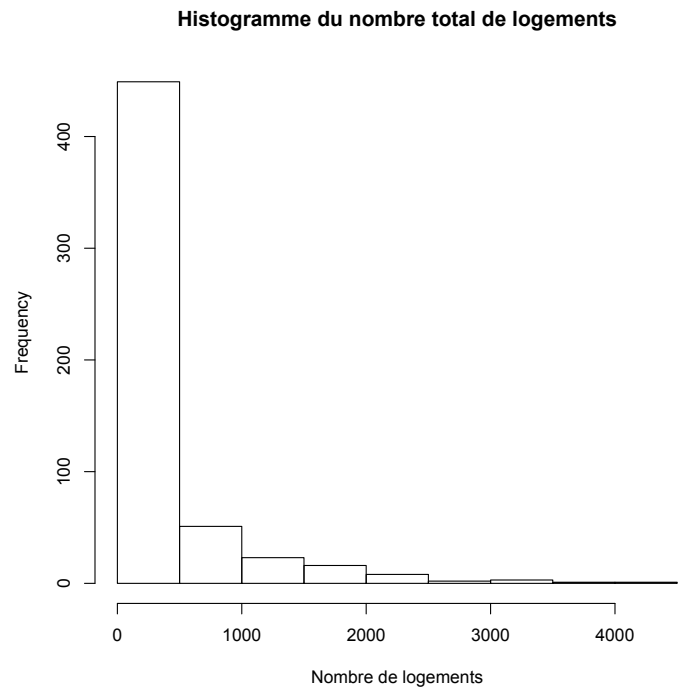


Variable quantitative continue

Une variable qui prend beaucoup de valeurs ; par exemple, la variable `LOG` de la table `rec` qui donne le nombre total de logements :

```
boxplot(rec[,"LOG]) ##boxplot pt var poidsb
hist(rec[,"LOG"],xlab="Nombre de logements", main="Histogramme du nombre total de logements")
```





Variable qualitative

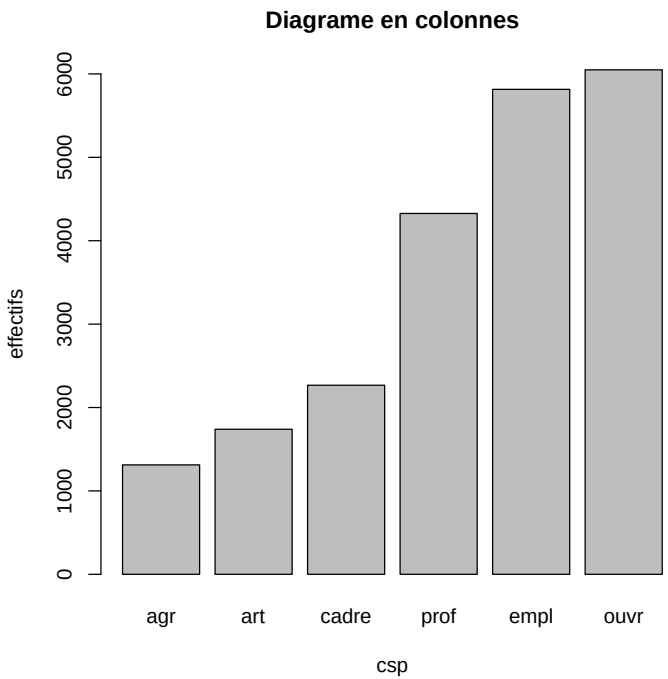
Une variable est considérée **qualitative** si ses valeurs ce sont des modalités.
Exemples :

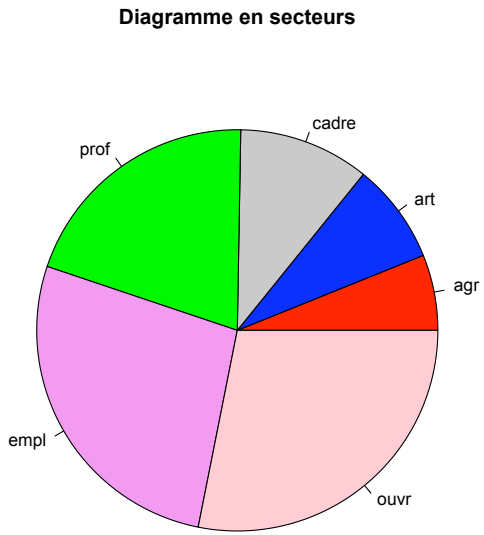
1. le sexe : H(omme), F(emme) ;
2. la catégorie socio-proféssionnelle (CSP) : ouvrier, cadre, ...
3. ...

Attention : on peut avoir une variable qualitative avec modalités définies par des nombres ;
`R` peut faire les tout types de calculs avec ces modalités alors qu'il n'y a pas de sens. Pour que `R` réconnaisse une variable qualitative, on utilise l'instruction `as.facteur`

Variable qualitative : représentation graphique

```
> csp=c("agr", "art", "cadre", "prof", "empl", "ouvr")
 ##les effectifs
> effect=c(1312,1739,2267,4327,5815,6049)
> barplot(effect, xlab="csp", ylab="effectifs", names.arg=csp)
> pie(effect, labels=csp, main="Diagramme en secteurs",col=c())
```





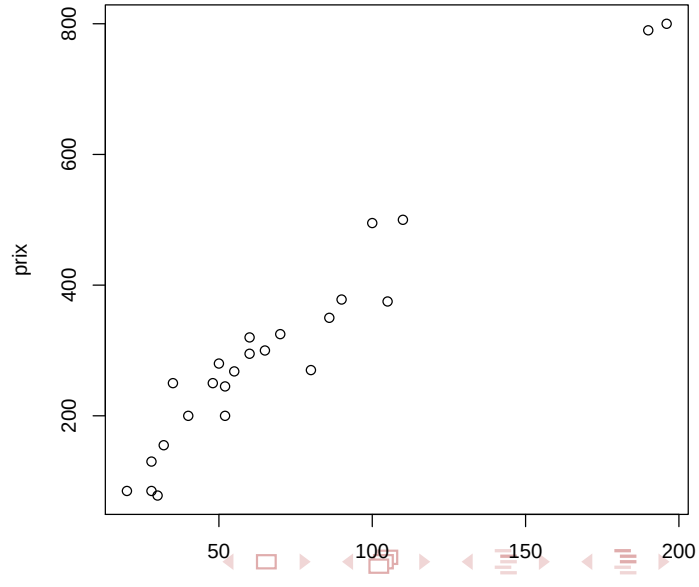
La régression simple

En général, le modèle de régression simple d'une variable Y qui prend les valeurs $y_1, \ldots, y_n$ sur une variable X de valeurs $x_1, \ldots, x_n$ est de la forme :

$$y_k = a + bx_k + \varepsilon_k$$

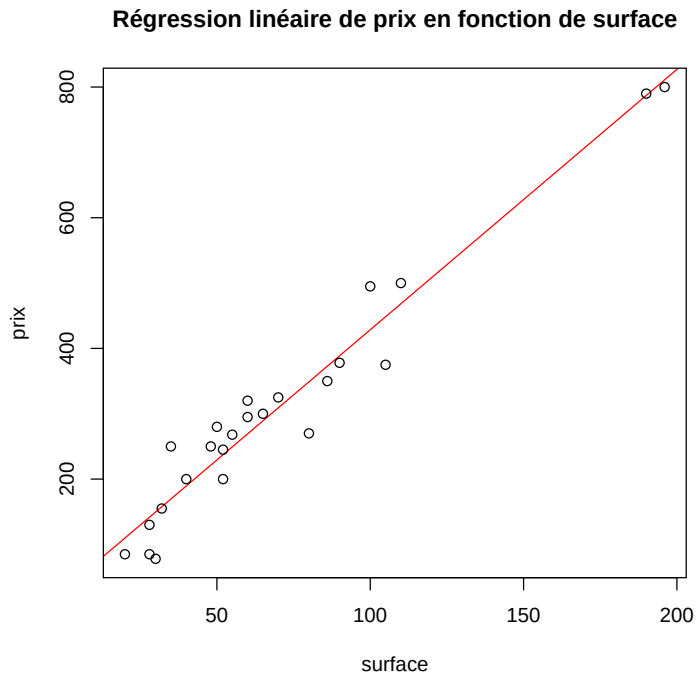
Les paramètres estimés (à partir des données) sont $\hat{a} = \overline{y} - \hat{b}\overline{x}$ et $\hat{b} = (\sum_{i=1}^{n} x_k^2)^{-1} \sum_{i=1}^{n} x_k y_k$

```
plot(surface,prix)
```



La régression simple (2)

```
###il realise le modele
> lm(prix~surface)
Call:
lm(formula = prix ~ surface)
Coefficients:

(Intercept)  surface
  30.0921  3.9844

###trace de la droite de \\
regression
> abline(lm(prix~surface))
```



La droite de régression estimée a l'équation

$$y = 30.0921 + 3.9844 * x$$

La régression simple(3)

Plusieurs fonctions permettent d'afficher des détails concernants le modèle utilisé :
`modele.reg=lm(prix surface)` : copie les résultats dans un objet ;
`summary(modele.reg)` : résultats sure les tests sur les paramètres du modèle
`residuals(modele.reg)` : pour les résidus
`predict(modele.reg)` : les valeurs prédites
`coef(modele.reg)` : les coefficients
`names(modele.reg)` : liste des resultats de l'objet `modele.reg`
`summary(modele.reg)["r.squared"]` : extraction de `r.squared`

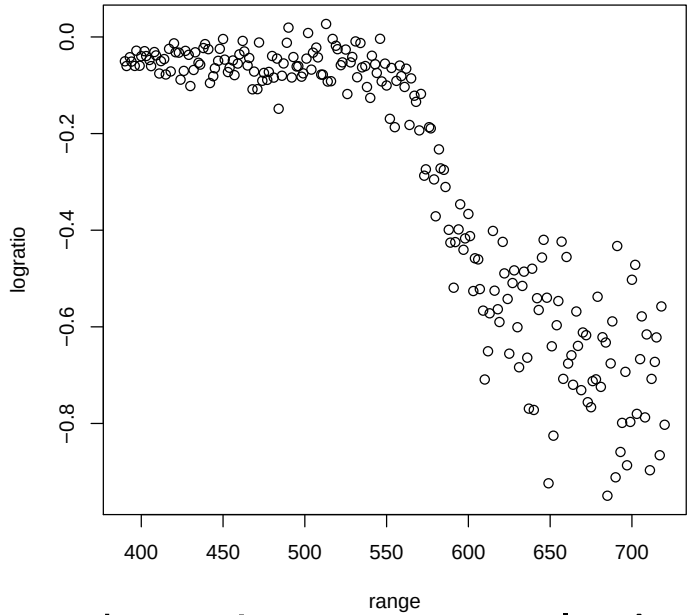
La régression simple : la fonction summary

```
summary(lm(prix~surface))
Call:
lm(formula = prix ~ surface)

Residuals:
    Min      1Q  Median      3Q     Max
-78.845 -23.259   5.293  26.546  80.453

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  30.0921    16.6455   1.808   0.0843 .
surface       3.9844     0.2003  19.896 1.49e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 43.84 on 22 degrees of freedom
Multiple R-Squared: 0.9474,Adjusted R-squared: 0.945
F-statistic: 395.9 on 1 and 22 DF,  p-value: 1.485e-15
```

La régression par un polynôme de degré 2

Considérons les données suivantes : `Lidar`

```
 ###lecture de la table et on l'attache
> lidar=read.table('lidar.txt',header=T)
> attach(lidar)
 ####le nuage des points
>  plot(range,logratio)
```

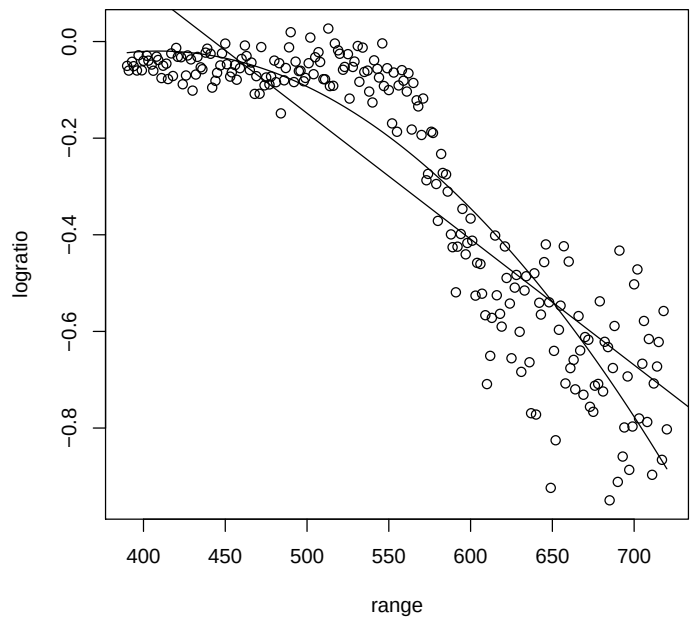


Nous voulons approcher le mieux ce nuage des points par un polynôme de degré 2 :

$$y_k = a + b_1 \cdot x_k + b_2 \cdot x_k^2 + \varepsilon_k$$

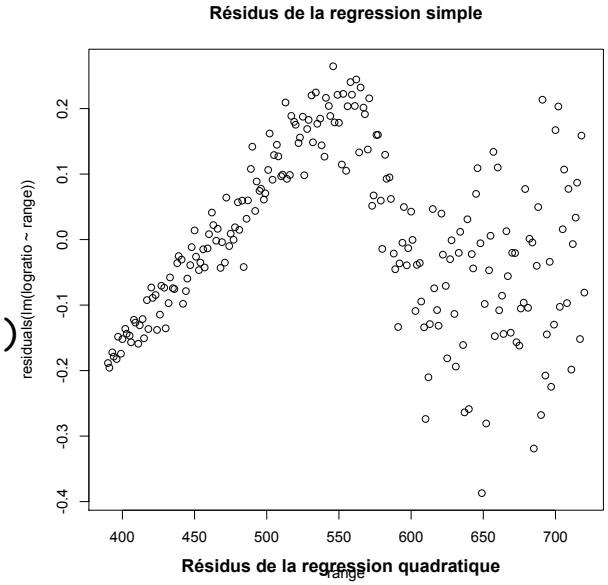
`poly(x,n): polynôme en x de degré n`

```
> lm(logratio~poly(range,2))
Call:
lm(formula = logratio ~ poly(range, 2))

Coefficients:
    (Intercept)  poly(range, 2)1  poly(range, 2)2
        -0.2912          -3.7068          -1.0916

modeldeux=lm(logratio~poly(range,2))

> plot(range,logratio)
> lines(range,predict(modeldeux),type="l")
> lines( abline(lm(logratio~range)))
```

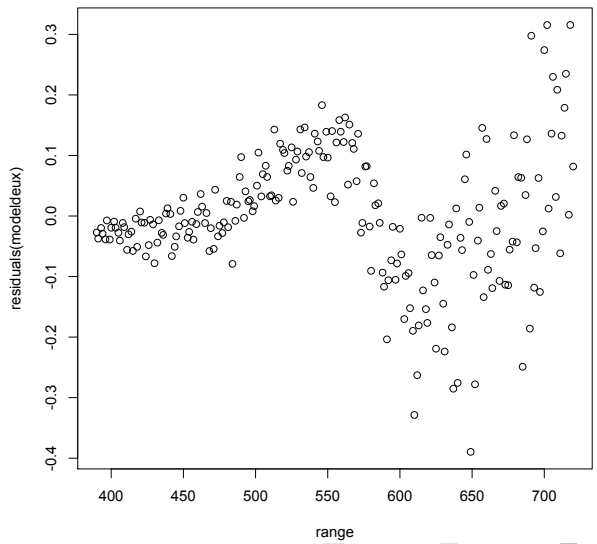


Comparaison des résidus de la régression simple et quadratique

```
>  plot(range,
residuals(lm(logratio~range)),
main="Résidus de la regression simple")
```




```
>plot(range,residuals(modeldeux),
main="Résidus de la
regression quadratique")
> detach(lidar)
```



Simulation de réalisations de lois classiques : Bernoulli, Poisson, normale ...

On génère des observations aléatoires avec
`rfunc(n,p[1],p[2], ...)` : func=loi de probabilité, n=nombre observations, p[1],p[2] paramètres de la loi

loi	**commande**
binomiale	`rbinom(n, size, prob)`
Poisson	`rpois(n, lambda)`
géométrique	`rgeom(n,prob)`
uniforme	`runif(n,min=0,max=1)`
Gauss	`rnorm (n, mean=0, sd=1)`

En plus, la loi de Bernoulli (ou "pile et face") avec la fonction
`sample(x,n, replace=TRUE)`
`qfunc` : pour obtenir les quantiles

Exemple

Loi binomiale et estimation de densité

On génère 1000 observations suivant une loi binomiale de paramètres 10 et 0.5.

```
> bin=rbinom(1000,10,0.5)
```

On trace l'histogramme

```
> hist(bin,xlim=c(min(bin), max(bin)),
probability=T, nclass=max(bin)-min(bin))
```

et on ajoute une estimation non-parametrique de la densité

```
> lines(density(bin),col="red",lwd=2)
```