Notes de Cours sur le logiciel R

Anne PHILIPPE

Université de Nantes, UFR des Sciences et Techniques
Laboratoire de Mathématiques Jean Leray
email : Anne.philippe@math.univ-nantes.fr

1er octobre 2008

Installation

Le logiciel R est un *freeware* disponible par exemple sur le site
http ://cran.r-project.org/
Il existe des versions

- Windows
- MacOS X
- Linux ...

Disponible :

- Le programme de "base"
- Des librairies complémentaires
- Site consacré aux graphiques
  http ://addictedtor.free.fr/graphiques/

Au démarrage

Le logiciel se lance en tapant R dans une fenêtre de commande Linux :

- informations sur le logiciel
- les commandes élémentaires :

```
Type "demo()" for some demos, "help()" for on-line help
     "help.start()" for a HTML browser interface to help.
Type "q()" to quit R.
>q()
Save workspace image? [y/n/c]:
```

- le symbole $>$ apparaît automatiquement en début de chaque ligne de commandes
- le symbole $+$ apparaît en début de ligne si la précédente est incomplète

Plan

Opérations élémentaires

- Opérations élémentaires sur les scalaires : $*$, $-$, $+$, $/$, $\hat{}$

```
>2+4
6
```

- Opérations avec affectation (avec ou sans affichage)

```
>x=2+4
>x
6
>(x=2+4)
6
```

Créer des vecteurs

- la fonction `c( )` concatène des scalaires ou des vecteurs :

```
> x=c(1,4,9)
> y=c(x,2,3)
> y
[1] 1 4 9 2 3
```

- Suites arithmétiques de raison 1 ou -1 : `c(a :b)`.

```
> c(1:4)     # a<b  raison 1   > c(4:1)     # a>b  raison -1
[1] 1 2 3 4                     [1] 4 3 2 1
> c(1.4:7)         # a-b n'est pas un entier
[1] 1.4 2.4 3.4 4.4 5.4 6.4
```

- Généralisation : `seq(a,b,t)` où `a` est premier terme, le dernier $\leq b$ et la raison `t`

```
seq(from, to)                  la raison est 1
seq(from, to, by= )            on fixe la raison
seq(from, to, length.out= )    on fixe le nombre de termes
```

- `x=rep(y ,n)` pour créer un vecteur constitué de l'élément `y` répété `n` fois. (y peut être un scalaire ou un vecteur)

Créer des matrices

Les matrices sont créées avec la fonction `matrix()` à partir d'un vecteur. On doit fixer le nombre de colonnes `ncol` et/ou le nombre de lignes `nrow`.

```
> x = matrix(c(2,3,5,7,11,13),ncol=2)
```

Par défaut la matrice est remplie colonne par colonne. Pour remplir ligne par ligne, on ajoute l'argument `byrow=T`

```
> y = matrix(c(2,3,5,7,11,13),ncol=2, byrow=T)
> x                    > y
     [,1] [,2]              [,1] [,2]
[1,]    2    7          [1,]    2     3
[2,]    3   11          [2,]    5     7
[3,]    5   13          [3,]    11   13
```

Attention : si la dimension du vecteur n'est pas égale au produit (ncol × nrow) alors l'opération effectuée est la suivante :

```
> matrix(c(1:3), ncol=2,nrow=3)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3

> matrix(c(1:3), ncol=2)
     [,1] [,2]
[1,]    1    3
[2,]    2    1
```

Extraire des éléments

```
> vect=c(1.1:10.1)
> vect[1]
[1] 1.1
>  mat[,1]     #1er colonne
[1] 1.1 2.1 3.1

> vect[c(2,4,8)]
[1] 2.1 4.1 8.1

>  mat[2:3,1:2]   # sous matrice
     [,1] [,2]
[1,]  2.1  5.1
[2,]  3.1  6.1
```

```
> mat=matrix(vect,ncol=3,nrow=3)
>  mat[2,1]
[1] 2.1
>  mat[3,]     # ligne n^^b0 3
[1] 3.1 6.1 9.1
```

Concaténer des vecteurs

1. rbind
2. cbind

```
> x=1:10
> y=x^2
> rbind(x,y)
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
x    1    2    3    4    5    6    7    8    9    10
y    1    4    9   16   25   36   49   64   81   100
> cbind(x,y)
       x   y
 [1,]  1   1
 [2,]  2   4
 [3,]  3   9
 [4,]  4  16
 [5,]  5  25
 [6,]  6  36
 etc
```

Opérations sur les Matrices/Vecteurs

• Les opérations + * - / entre 2 vecteurs ou matrices de même dimension sont des opérations terme à terme.

```
> x=c(1:5)
> y=c(rep(0,3),rep(1,2))

> x
[1] 1 2 3 4 5
> y
[1] 0 0 0 1 1

> x*y
[1] 0 0 0 4 5
```

Le produit matriciel est obtenu avec % ∗ %

ATTENTION

Si les vecteurs ne sont pas de même longueur, le plus court est complété automatiquement .

```
> x =c(1:5)
> x
[1]  1  2  3  4  5
> y =c(1,2)
> y
[1] 1 2
> x + y
 [1]  2  4  4  6  6
```

```
x  : 1 2 3 4 5
y  : 1 2 1 2 1
----------------
x+y : 2 4 4 6 6
```

Pour extraire les éléments dŠun vecteur, on peut utiliser des instructions logiques.

- extraire les composantes >8
  `vect[vect>8]` : `vect>8` est un vecteur de `TRUE` et `FALSE`, on extrait les composantes affectées à `TRUE`.
- extraire les composantes >8 ou <2
  `vect[(vect>8) | (vect<2)]`
- extraire les composantes >8 et <10
  `vect[(vect>8) & (vect<10)]`

Objets booléens et instructions logiques

Les opérations logiques : `< , > , <= , >= , != [différent], == [égal]` retournent `TRUE` ou `FALSE`. La comparaison entre deux vecteurs est une comparaison terme à terme. Si les vecteurs ne sont pas de même longueur, le plus court est complété automatiquement.

```
> a= 1:5   ;   b=2.5
> a<b
[1]  TRUE  TRUE FALSE FALSE FALSE
```

Il est possible de définir plusieurs conditions à remplir avec les opérateurs

- ET : &
- OU : |

Quelques fonctions usuelles

Si $A = (a_{i,j})$ est une matrice, alors `exp(A)` retourne une matrice constituée des éléments $e^{a_{i,j}}$.
Idem pour les fonctions :

```
sqrt               square root
abs                absolute value
sin   cos   tan    trigonometric functions (radians)
exp   log          exponential and natural logarithm
log10              common logarithm
gamma lgamma       gamma function and its natural log
```

Ces fonctions retournent un scalaire :

- soit x un vecteur de dimension n
- `sum()` (somme $\sum_i x_i$), `prod()` (produit $\prod_i x_i$), `mean()` (moyenne $\frac{1}{n}\sum_{i=1}^{n} x_i$)
- `max(), min()`
- `length(), ncol(), nrow()`

Ces fonctions retournent un vecteur :

- `cumsum()` (sommes cumulées $(x_1, x_1 + x_2, \ldots, \sum_{i=1}^{n} x_i)$, `cumprod()` (produits cumulés),
- `sort` (tri), `order, unique`
  remarque : `sort(x) = x[order(x)]`
- `fft()` (transformé de Fourier)

`which(condition)` retourne les indices des coordonnées égales à la valeur TRUE

```
> x=(1:10)^2
> x
 [1]   1   4   9  16  25  36  49  64  81 100
> which(x== 25)
[1] 5
> which(x > 21)
[1]  5  6  7  8  9 10
```

Cas particulier :

- `which.max(x)` (ou which.min) retourne `which(x==max(x))`

Construire un tableau croisé

La fonction `outer` retourne une matrice de la forme

$$M(i,j) = f(x_i, y_j)$$

où

- x et y sont des vecteurs
- f une fonction de deux variables $f : (x, y) \mapsto f(x, y)$

```
f=function(x,y) sin(x+y^4)
x=seq(0,2*pi,0.1)
y=seq(0,pi,0.1)
M=outer(x,y,'f')
```

Importer/exporter des données

1. Importer une suite : `x=scan("data.dat")` : pour créer un vecteur à partir de données stockées dans un fichier, ici `data.dat`.
2. Importer un tableau : `x=read.table("data.dat")` ou `x=read.table("data.dat", header=TRUE)` LŠinstruction header=TRUE permet de préciser que la première ligne du fichier contient le nom des colonnes du tableau.
3. Exporter : `write, write.table`

Définition d'une liste

C'est une structure qui regroupe des objets (pas nécessairement de même type).

- On crée les listes avec la fonction `list`
- Pour obtenir une liste (appelée *rnd*) contenant
  - un vecteur dans `serie`
  - un scalaire dans `taille`
  - une chaîne de caractères dans `type`
- la syntaxe est la suivante

```
>rdn=list(serie=c(1:100),taille=100,type="arithm")
```

- Pour créer des objets `R` à partir d'une liste (extraire les composantes d'une liste)

```
>attach(rdn)
 "serie"  "taille" "type"
```

- supprimer les objets créés avec la fonction `attach` :

```
>detach(rdn)
```

Opérations sur les listes

- Pour visualiser les composantes d'une liste

```
>names(rdn)
[1] "serie"  "taille" "type"
> summary(rdn)
       Length Class  Mode
serie  100    -none- numeric
taille   1    -none- numeric
type     1    -none- character
```

- Pour atteindre les composantes d'une liste

```
>rdn$taille   OU   >rnd[[2]]
[1] 100              [1] 100
```

Plan

Structure générale pour créer des fonctions

- La structure générale d'une fonction est

```
>myname=function(liste_des_param^^e9tres)
{
  commandes
  objets_retourn^^e9
  }
```

- Les accolades { et } définissent le début et la fin de la fonction.
- La dernière instruction contient le nom de lŠobjet retourné par la fonction.
- Exécuter la fonction : `>myname(...)`
- On peut donner des valeurs par défaut aux paramètres

Exemple

```
>exemple=function(n)
{ sample=runif(n)                    #
 m = mean(sample) ; v =var(sample)
 list(serie=sample,moyenne=m, variance=v)
}

>exemple(10)    OU  exemple(n=10)
$serie
[1] -1.1532544 -1.0295700 -0.9065778 ....
[7] -1.0301214  1.0690254 -0.2276646  0.1753657
$moyenne
[1]  -0.2267737
$var
[1]   0.7674006
>sortie = exemple(10)
>sortie$serie
[1] -1.1532544 -1.0295700 ...
```

Exemple : utilisation des paramètres par défaut

```
>exemple2=function(n=10)
{ sample=runif(n)                    #
m = mean(sample) ; v =var(sample)
list(serie=sample,moyenne=m, variance=v)
}
```

Les trois commandes suivantes

```
>exemple(10)    OU  exemple(n=10) OU exemple()
```

retournent le même résultat

Plan

Instructions conditionnelles

La syntaxe
`if (condition) {instructions}` permet de calculer les instructions uniquement si la condition est vraie.
`if (condition) { A } else {B}` calcule les instructions `A` si la condition est vraie et les instructions `B` sinon.
Par exemple,
`if (x>0) y=x*log(x) else y=0`
Remarque : Si les instructions se limitent à un seul calcul on peut utiliser `ifelse`
Par exemple,
`y=ifelse(x>0,x*log(x),0)`

Boucles

Les 3 structures suivantes sont équivalentes

```
> for ( i in 1:10) ou  while (i< 11)   ou  > repeat
+ {                    + {                  + {
+ print(i)             + print(i)           +  print(i)
+ }                    + i=i+1             + if (i <10 )
                       +}                   +  {i=i+1} else
                                            +  { break}
                                            + }
```

La fonction apply

La fonction `apply()` permet d'effectuer des opérations sur les lignes ou les colonnes d'une matrice.
Cette fonction retourne un vecteur.

```
> data(nottem)
> nottem
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
etc
> x=matrix(nottem,ncol=12,byrow=T)
> apply(x,1,mean)  # moyenne sur les lignes
 [1] 48.89167 50.73333 47.27500 47.81667 48.72500
etc
> apply(x,2,mean) # moyenne sur les colonnes
 [1] 39.695 39.190 42.195 46.290 52.560 58.040 61.900 60.
etc
```

La fonction sapply,lapply

On applique la même fonction sur chacune des coordonnées d'un vecteur ou sur chacune des composantes d'une liste.

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
etc
>pts <- list(x=cars[,1], y=cars[,2])
>lapply(x, mean)
$x                  $y
[1] 15.4            [1] 42.98
>lapply(x, quantile, probs = 1:3/4)
$x
  0%  25%  50%  75% 100%
   4   12   15   19   25
$y
  0%  25%  50%  75% 100%
   2   26   36   56  120
```

Quelques commandes

> ls() liste des variables et des fonctions
> source("nom.com") pour exécuter les commandes dans le fichier `nom.com`
> remove(a) supprimer l'objet `a`
> rm(list=ls()) pour effacer l'ensemble des objets créés
> # débute une ligne de commentaires
> readline() pour faire une pause
> browser() interrompre le programme pour inspecter les variables locales

Plan

Les lois

package: stats

Loi	nom	paramètres		
beta	beta	shape1	shape2	ncp
binomial	binom	size	prob	
Cauchy	cauchy	location	scale	
chi-squared	chisq	df	ncp	
exponential	exp	rate		
F	f	df1	df1	ncp
gamma	gamma	shape	scale	
geometric	geom	prob		
hypergeometric	hyper m	n	k	
log-normal	lnorm meanlog	sdlog		
logistic	logis location	scale		
negative	binomial	nbinom	size	prob
normal	norm	mean	sd	
Poisson	pois	lambda		
Student	t	t	df	ncp
uniform	unif	min	max	
Weibull	weibull	shape	scale	
Wilcoxon	wilcox	m	n	

package: stats

- Simulation de variables aléatoires suivant les lois disponibles :
  `runif(taille,min=0,max=1)`, `rpois(taille, a)`,
  `rnorm(taille, mean=0, sd=1)`, `rbinom(taille, n,p)`
  etc ..
- Simulation de va iid suivant une loi discrète à support fini :
  $p(X = x_i) = p_i$ avec $i = 1..k$
  `sample(x, size, replace=FALSE, prob)`
- En remplaçant `r` par `d` , `p` ou `q` on obtient respectivement
  - la densité
  - la fonction de répartition
  - son inverse

• Exemple : on simule des variables aléatoires iid suivant la loi normale standard (i.e. moyenne 0 et variance 1)

```
> x=rnorm(1000,0,1)
```

• Statistiques descriptives (sur les valeurs simulées `x`)

```
> median(x)
[1] -0.0584617
> mean(x)
[1] -0.04102906
> var(x)
[1] 0.9918595
>quantile(x, probs=c(5,50,95)/100)
```

```
> range(x)
[1] -3.427918  2.847309
```

Plan

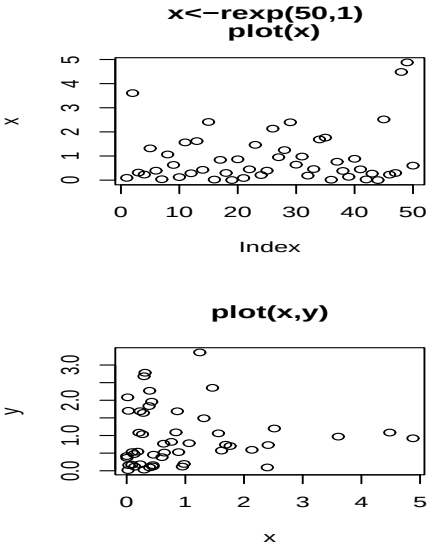
la fonction "centrale" : `plot()`

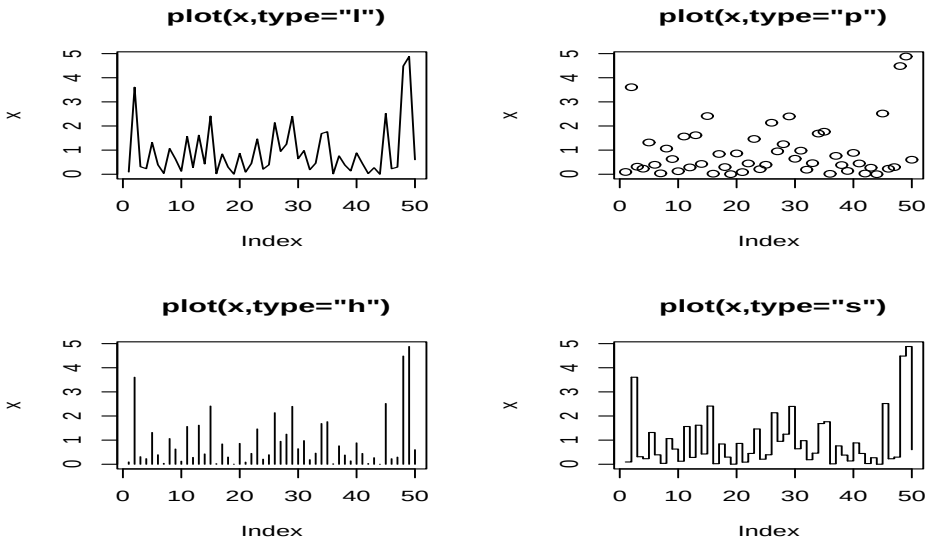
package: graphics

- `plot` : pour tracer la première "courbe"
- `points` ou `lines` : pour ajouter une ligne ou un nuage de points

Quelques arguments de la fonction `plot` :

- `type="p" (points) ou "l" (ligne)` : pour tracer une ligne ou un nuage de points.
- `pch` : type de points
- `lty` type de lignes.
- `col` : couleur
- pour fixer les limites des axes : `ylim=c(1,10)` et `xlim=c(1,10)`

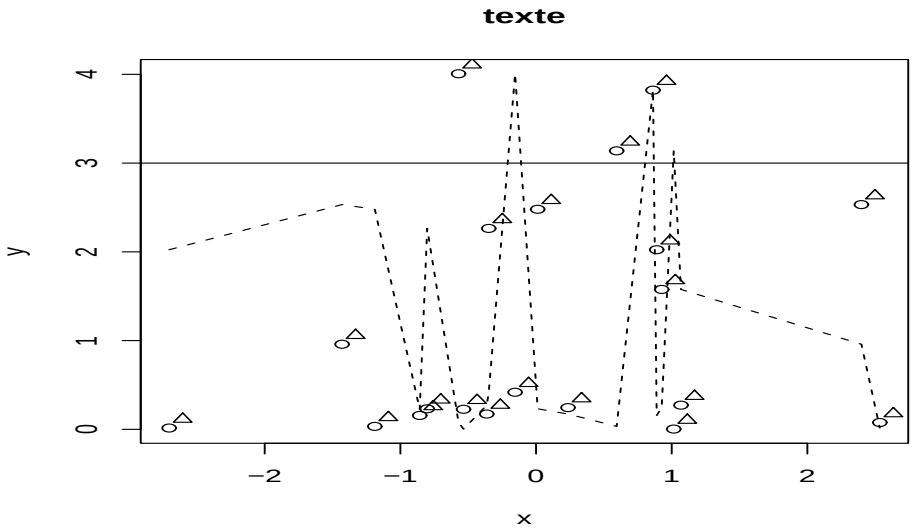




superposer des courbes

```
> x=rnorm(20)          > y=rexp(20)

> plot(x,y)                        #nuage de points
> points(x+.1,y+.1, pch=2)         #ajouter un nuage de points
> lines(sort(x),y, lty=2)          #ajouter une ligne
> text(1,5,"text")                 #texte
> abline(h=3)                      #ajouter une ligne horizontale
>title("texte")                    # ajouter un titre
```

Autour de la fonction plot




```
> methods(plot)
 [1] "plot.data.frame" "plot.default"
 [3] "plot.density"    "plot.factor"
 [5] "plot.formula"    "plot.function"
 [7] "plot.histogram"  "plot.lm"
 [9] "plot.mlm"        "plot.mts"
[11] "plot.new"        "plot.POSIXct"
[13] "plot.POSIXlt"    "plot.table"
[15] "plot.ts"         "plot.window"
[17] "plot.xy"
```

Quelques exemples

- sur une fonction (par ex sin)

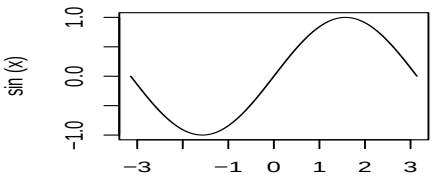
```
> plot(sin,xlim=c(-pi,pi))
```

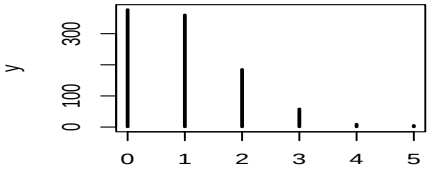
- sur un tableau

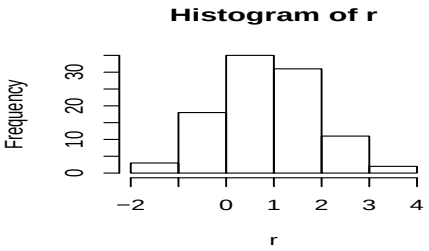
```
> x=rpois(1000,1)
> y=table(x)
> y
x
  0   1   2   3   4   5
352 371 203  55  18   1
> plot(y)
```

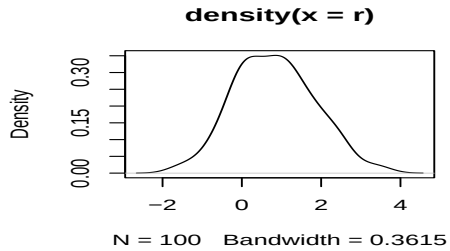
- sur un histogramme ou une densité

```
>r=rnorm(100,1)
> z=hist(r,plot=F)
> plot(z)
>w=density(r)
>plot(w)
```









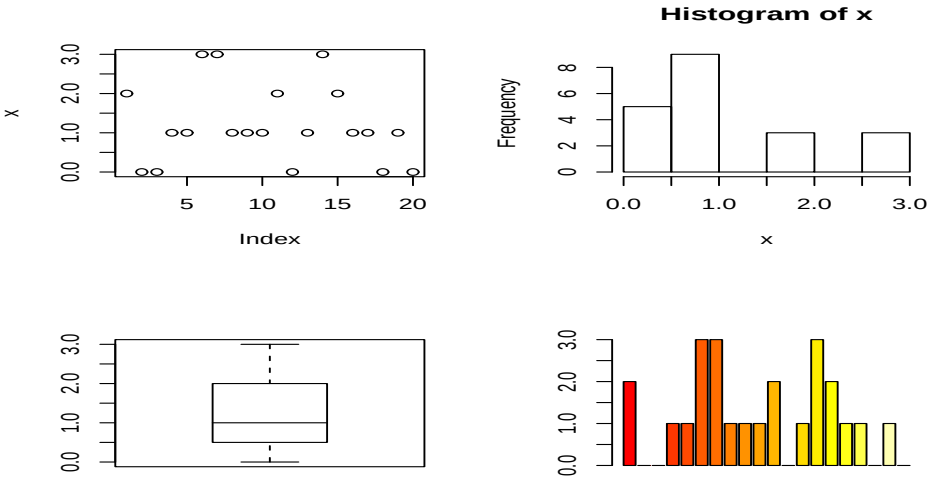
- Pour sauvegarder un graphique dans un fichier PostScript
  `dev.print(postscript, file="essai.ps")`
  voir aussi la fonction `postscript()`
- La fenêtre graphique peut être fractionnée en utilisant
  - `par(mfrow=c(n,m))`, on obtient alors n*m graphiques sur une même page repartis sur n lignes et m colonnes
  - `split.screen(m,n)` puis `screen(i) ou screen(i,FALSE)` pour ajouter, `erase.screen()`
    `close.screen(all = TRUE)`

Quelques graphiques

```
> x=rpois(20,1)
> par(mfrow=c(2,2))
>plot(x)
```

- `hist, boxplot`
  `> hist(x)` `> boxplot(x)`
- `barplot`
  `> barplot(x)`
- `pairs`

Les graphiques obtenus

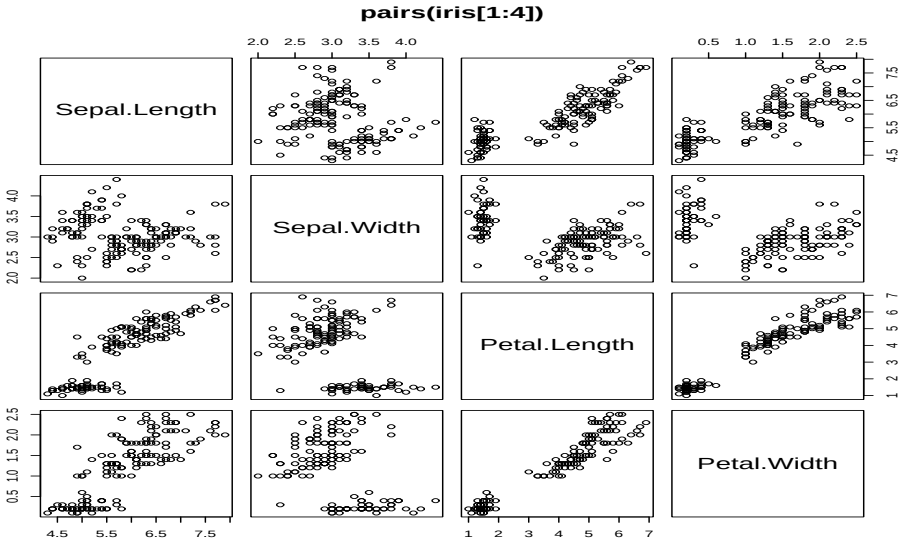


Utilisation des biplots

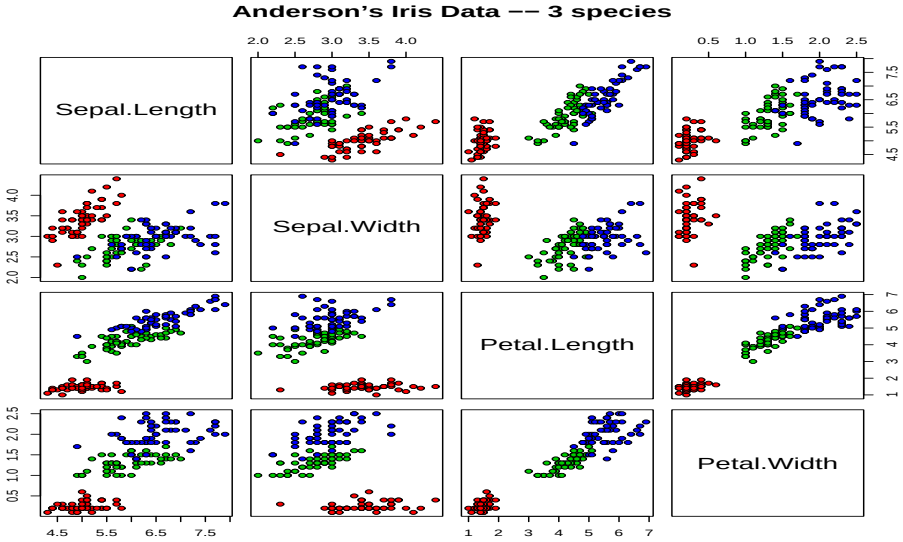
```
>data(iris)
>iris
```

	Sepal.Length	Sepal.Height	Petal.Length	Petal.Height	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4					
...					
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
...					
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica

```
> summary(iris)
  Sepal.Length    Sepal.Height     Petal.Length    Petal.Height
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa     :50
 versicolor:50
 virginica :50
```




```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("red", "green3", "blue")[codes(iris$Species)])
```

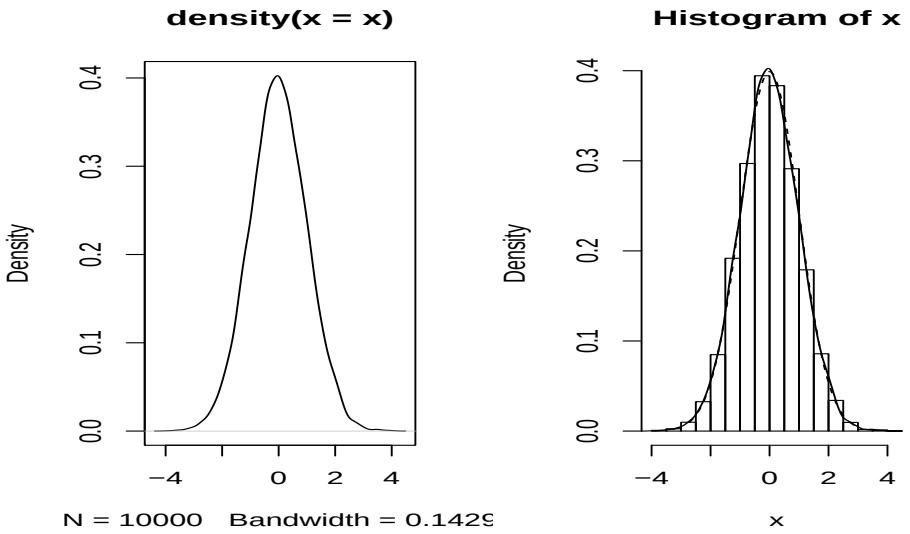


Quelques outils de la statistique classique

package: stats

- Estimation de la densité par l'estimateur à noyau (voir l'aide pour le choix du noyau , la vitesse de fermeture de la fenêtre etc.)

```
par(mfrow=c(1,2))
x_rnorm(10000)                  #echantillon N(0,1)
y_density(x)                    #estimateur ^^e0 noyau
plot(y)                         #repr^^e9sentation graphique
hist(x, proba=T)                #histogramme
lines(y)                        #superpose estimateur ^^e0 noyau
z_seq(min(x),max(x),0.01)
lines(z,dnorm(z,0,1),lty=2) #superpose densit^^e9 th^^e9orique
```



Exemple : mélange de lois

On considère la densité de probabilité suivante

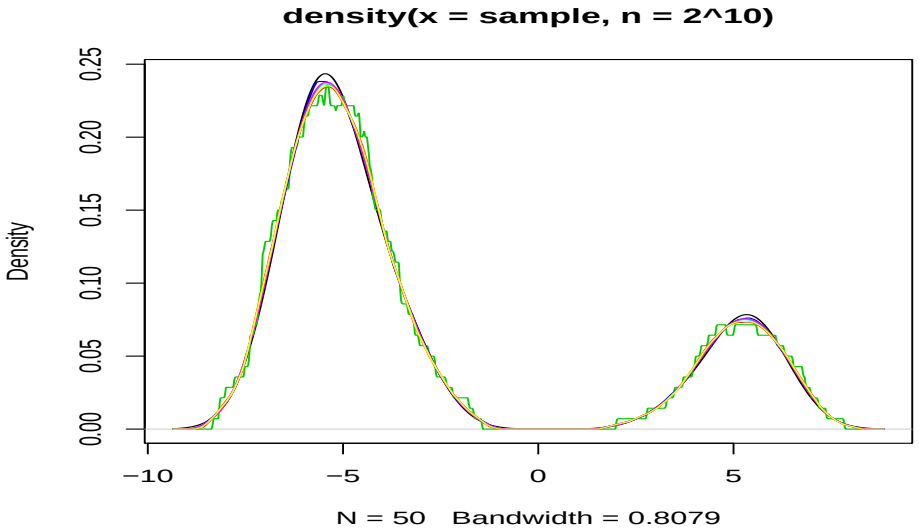
$$f(x) = \frac{1}{4}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x-5)^2} + \frac{3}{4}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x+5)^2}$$

- Pour simuler suivant f

```
w_rbinom(50,1,1/4)
sample _ w*rnorm(50,5) +(1-w) * rnorm(50,-5)
```

- Estimation de la densité f

```
>(kernels = eval(formals(density)$kernel))
[1] "gaussian"     "epanechnikov" "rectangular" etc
>plot(density(sample, n = 2^13))
>for (i in 2:length(kernels))
+ lines(density(sample, kern = kernels[i], n = 2^13), col = i)
```

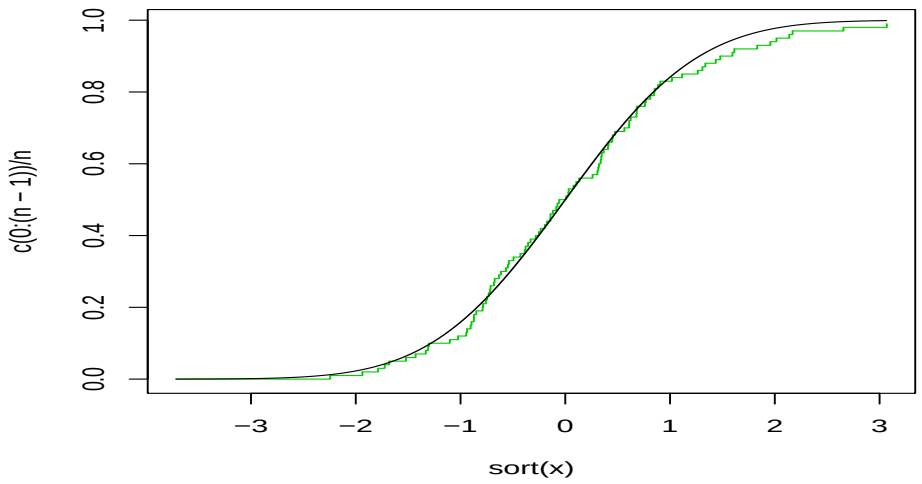


Fonction de répartition empirique (ecdf)

$$\hat{F}_n(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{]-\infty t]}(X_i) \to F(t)$$

```
> x= rnorm(100)
>Fn=ecdf(x)
>plot(Fn)                              #on trace la cdf empirique
> z=seq(min(x),max(x),0.01)           # ajoute la cdf theorique
> lines(z,pnorm(z), col=3,lwd=2)
```

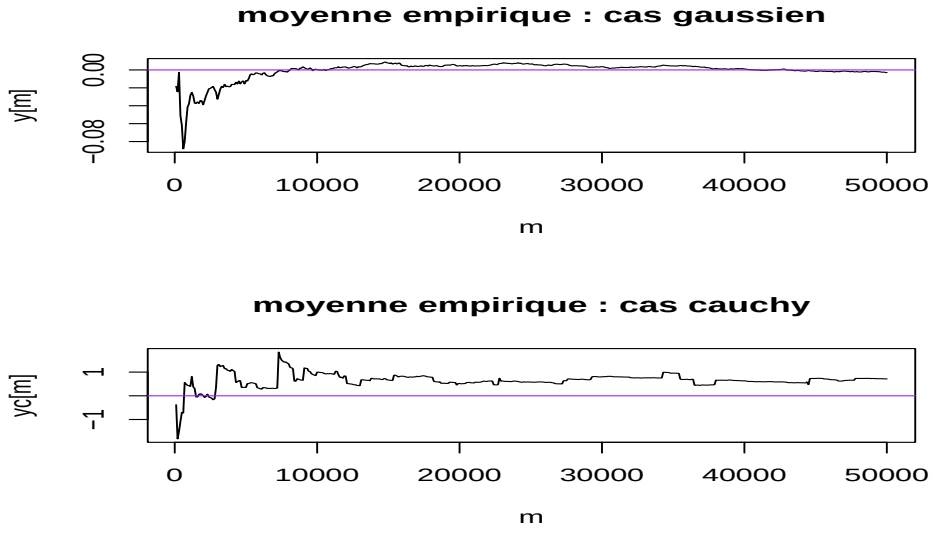
Illustration de la convergence p.s.



Soit $X_1, \ldots, X_n$ une suite de variables aléatoires iid, si $E|X_1| < \infty$ alors

$$\frac{1}{n}\sum_{i=1}^{n} X_i \to E(X_1)$$

```
n_50000
par(mfrow=c(2,1))
#loi gaussienne                  # loi d Cauchy
x_rnorm(n,0,1)                   xx_rt(n,0,1)
y_cumsum(x)/(1:n)                yc_cumsum(xc)/(1:n)
plot(y, type=''l'')              plot(yc, type=''l'')
abline(h=0)
```



Quelques librairies

Pour la liste des librairies `library()`
On charge une librairie avec la fonction `library(nom-de-la-librairie)`. Quelques librairies `base, stats, graphics, etc` sont chargées au lancement de R

base	The R Base Package	graphics	The R Graphics Package
boot	Bootstrap R (S-Plus)	grid	The Grid Graphics Package
	Functions (Canty)	MASS	Main Package of Venables
class	Functions for		and Ripley's MASS
	Classification	stats	The R Stats Package
foreign	Read data stored by	stats4	Statistical functions
	Minitab, S, SAS, SPSS,		using S4 classes
	Stata, ...		

Tests classiques :

package: stats

```
t.test(x,mu=5,alt="two.sided")      #student
t.test(x,y,alt="less", conf=.95)

var.text(x,y)                       # comparaison variance
cor.test(x,y)                       # non correlation
chisq.test(x,y)                     #independance
Box.test(z, lag = 1)                #non correlation

shapiro.test(x)                     #normalit^^e9
ks.test(x,"pnorm")                  #normalit^^e9 K-S
ks.test(x,y)                        # m^^eame distribution
```

Exemple : Test de Student `t.test()`

$X_1, \ldots, X_n$ iid $\mathcal{N}(1,1)$ et $Y_1, \ldots, Y_m$ iid $\mathcal{E}(1)$
Test $H_0 : E(X) = E(Y)$ vs $H_1 : E(X) \neq E(Y)$

```
> x = rnorm(100,1,1)
> y = rexp(200,1)
> t.test(x,y)

        Welch Two Sample t-test
data:  x and y
t = -0.2178, df = 178.446, p-value = 0.8278
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2648092  0.2121608
sample estimates: mean of x : 0.9544127  mean of y : 0.9807369
```

Test d'ajustement

On simule une loi normale en utilisant le TCL sur des variable iid suivant la loi uniforme.

Méthode de simulation non exacte ....

$U = 1, \ldots, U_n$ iid suivant la loi uniforme

$$\sqrt{\frac{n}{12}}(\bar{U}_n - \frac{1}{2}) \Rightarrow X \sim N(0,1) \qquad \bar{U}_n = \frac{1}{n}\sum_{i=1}^{n} U_i$$

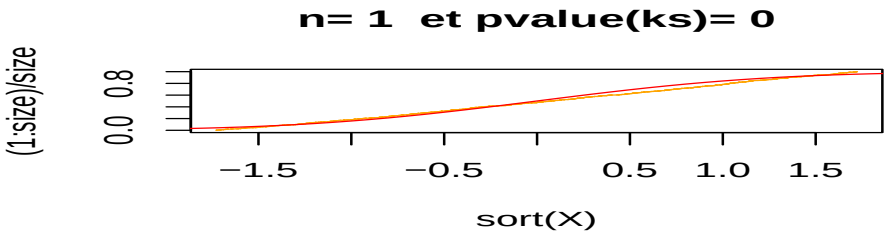
La générateur s'écrit

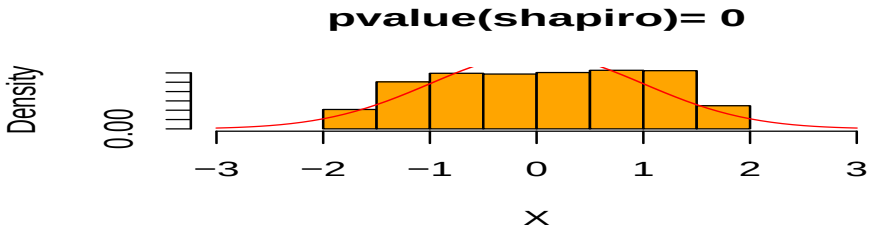
```
simU<-function(taille,size)
{
y = matrix(runif(taille*size),ncol=size)
(apply(y,1,mean)-1/2)*sqrt(taille/12)
}
```

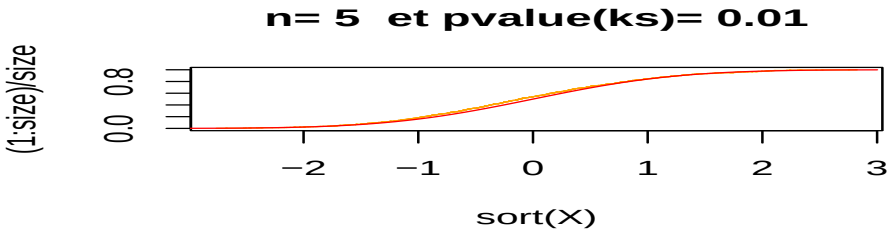
Performance de l'algorithme

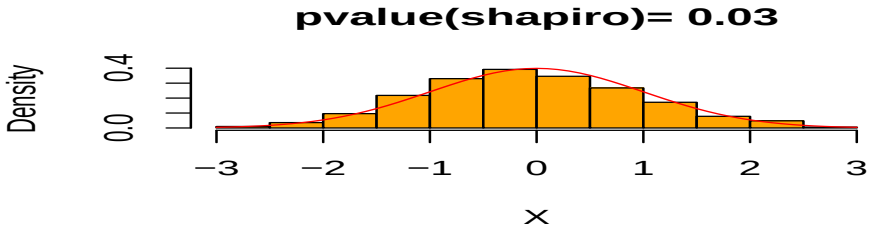
Pour différentes valeurs de $n = 1,\ 5,\ 10$

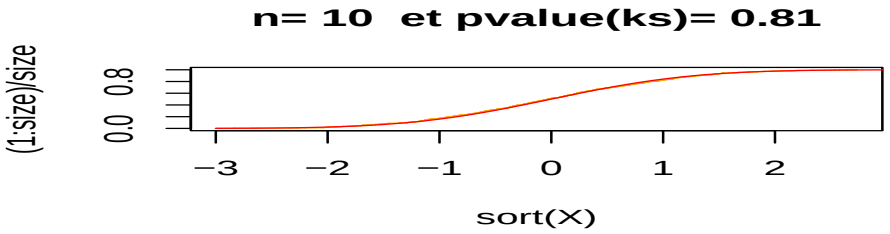
```
X = simU(1000,n)
test=ks.test(X,"pnorm",0,1)$p.value
test2 = shapiro.test(X)$p.value

plot(sort(X),(1:size)/size, type="s", col="orange")
lines(seq(-3,3,.1),pnorm(seq(-3,3,.1)),col="red")
title(paste("n=",n," et pvalue(ks)=",floor(c(test)*100)/100))

hist(X,col="orange", xlim=c(-3,3), proba=T,main="")
lines(seq(-3,3,.1),dnorm(seq(-3,3,.1)),col="red")
title(paste("pvalue(shapiro)=",floor(c(test2)*100)/100))
```

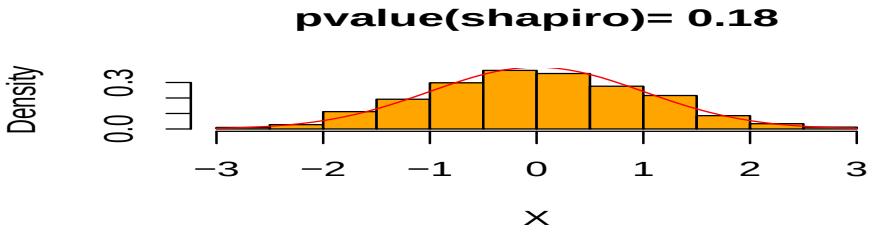












Plan

Régression linéaire

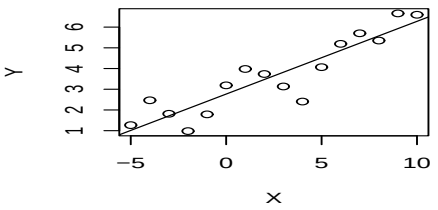
package: stats

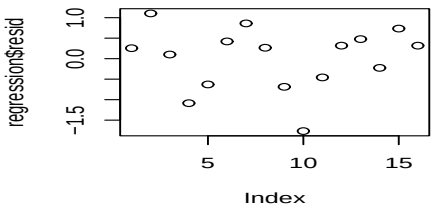
Le modèle le plus simple

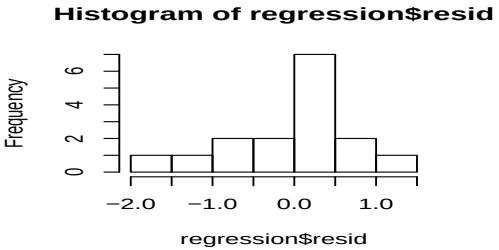
$$y_i = ax_i + b + \epsilon_i$$

où $(\epsilon_i)_i$ est une suite de variables aléatoires non corrélées.
Pour réaliser une régression linéaire, par la méthode des moindres carrés, on utilise la fonction `lm` :
Si les données sont sous la forme de 2 vecteurs `X` et `Y` (de même taille)

```
> regression=lm(Y~X)
> regression
Coefficients:
(Intercept)            X
     2.7679       0.3511
```





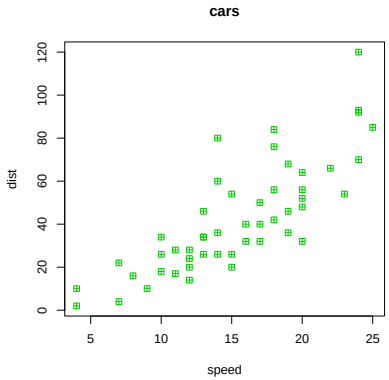


Visualisation :

```
> par(mfrow=c(2,2))
> plot(X,Y)                          # nuage
> lines(regression$fitted.values)
> plot(regression$residuals)         # residus
> hist(regression$resid)
```

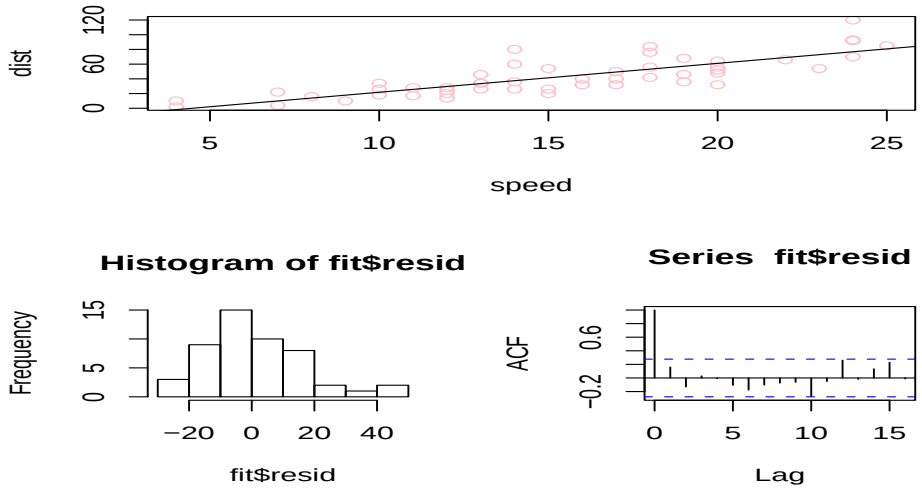
Si les données sont disponibles sous la forme d'une liste : par exemple la liste `cars` contient

- speed
- dist




```
> data(cars)
> fit = lm(dist ~ speed, cars)
```

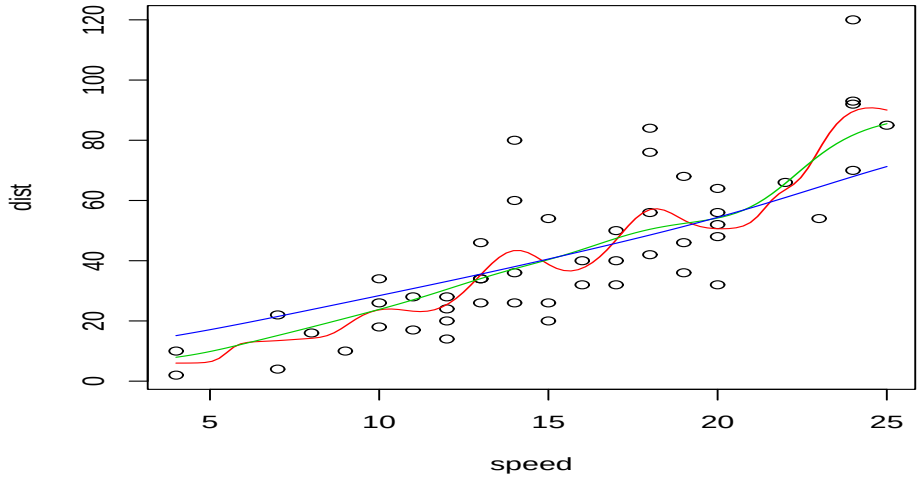
Visualisation




```
> split.screen(c(2,1))
[1] 1 2
> split.screen(c(1,2), screen = 2)
[1] 3 4
> screen(1)
> plot(cars, col="pink")
> abline(fit)
> screen(3)
> hist(fit$resid)
> screen(4)
> acf(fit$resid)
```

Lissage

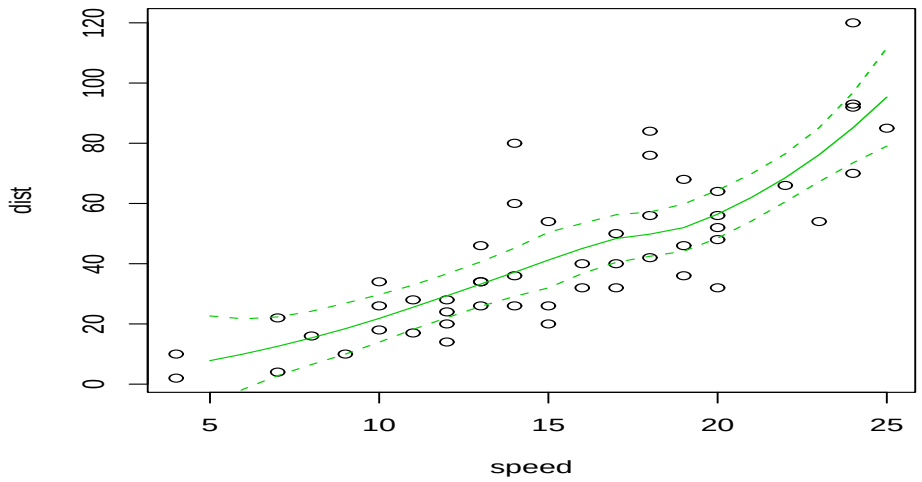
package: stats




```
>data(cars)
>attach(cars)
>plot(speed, dist)
>lines(ksmooth(speed, dist, "normal", bandwidth=2), col=2)
>lines(ksmooth(speed, dist, "normal", bandwidth=5), col=3)
>lines(ksmooth(speed, dist, "normal", bandwidth=10), col=4)
```

Polynômes Locaux

package: stats




```
>data(cars)
>cars.lo = loess(dist ~ speed, cars)
>p = predict(cars.lo, data.frame(speed = seq(5, 30, 1)), se = TRUE)
>plot(cars)
>lines(seq(5, 30, 1),p$fit,col="green3")
```

- régression multiple `lm(v ~ v1 + v2 + v3)`
- régression linéaire généralisée `glm`
- etc

Plan

Exemple

package: stats

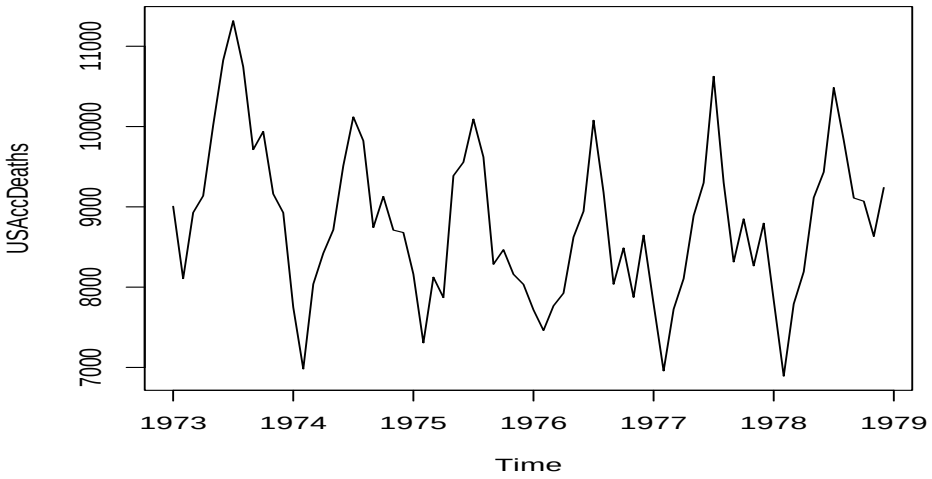
L'objet "série chronologique" est une liste qui contient

- les valeurs observées,
- la fréquence des observations,
- la date de la première observation
- la date de la dernière, etc...

```
>  data(USAccDeaths)
>  USAccDeaths
```

```
      Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   D
1973  9007  8106  8928  9137 10017 10826 11317 10744  9713  9938  9161  89
1974  7750  6981  8038  8422  8714  9512 10120  9823  8743  9129  8710  86
1975  8162  7306  8124  7870  9387  9556 10093  9620  8285  8466  8160  80
1976  7717  7461  7767  7925  8623  8945 10078  9179  8037  8488  7874  86
1977  7792  6957  7726  8106  8890  9299 10625  9302  8314  8850  8265  87
1978  7836  6892  7791  8192  9115  9434 10484  9827  9110  9070  8633  92
```

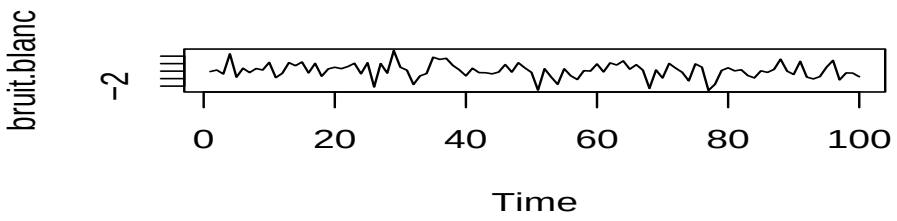
```
>plot(USAccDeaths)
```

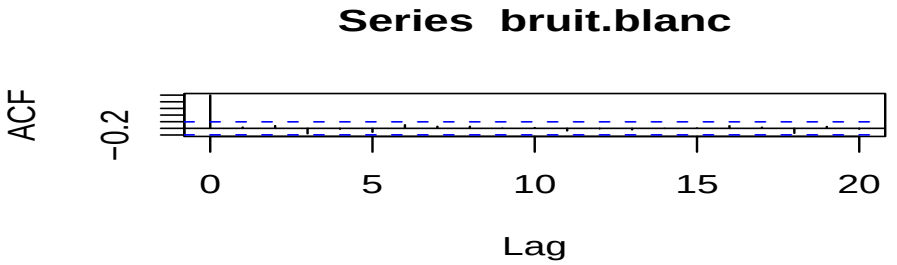


Pour créer une série chronologique, on utilise la fonction `ts`

```
>bruit.blanc=ts(rnorm(100), frequency = 1,
                            start = c(1), end=c(100))
>data=c(bruit.blanc)  #r^^e9cup^^e8re les observations
                       # dans le vecteur data
```

```
>plot(bruit.blanc)
>acf(bruit.blanc, type="correlation")
 #ou type="covariance" ou type="partial"
```





Filtre

- Premier exemple de filtre : $(I - L^p)^d$, on utilise la fonction `diff`

```
>diff(x,lag=p,differences=d)
```

- La fonction `filter` permet d 'appliquer des filtres linéaires :

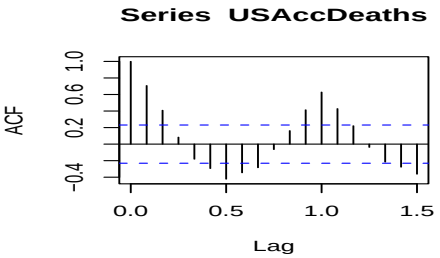
```
y=filter(x,sides= 1, method = ‘‘convolution’’ ,filter=c(2,3))
# y[i] = 2*x[i] + 3*x[i-1]
y=filter(x,sides= 2, method = ‘‘convolution’’ ,filter=c(2,3,4))
# y[i] = 2*x[i-1] + 3*x[i] +4]*x[i+1]
y=filter(x,method =’’recurcive’’ ,filter=c(2,3))
#y[i] = x[i] + 2*y[i-1] + 3*y[i-2]
```

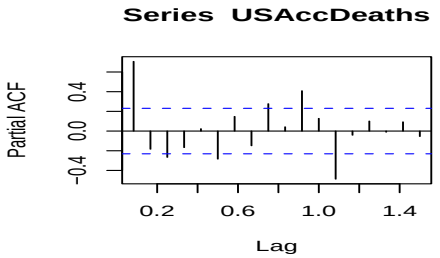
Ces fonctions permettent en particulier de créer un générateur de processus ARMA.

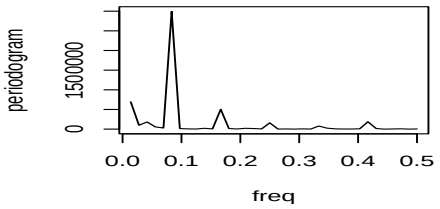
Étude préliminaire de la série USAccDeaths

```
    periodogram = function(traj)
{
        n = length(traj)
        freq = 1:(n %/% 2)/(n)
        periodogram = Mod(fft(traj))[2:(n %/% 2 + 1)]^
                2/(2 * pi * n)
        plot(freq, periodogram, type = "l")
}
```

```
>acf(USAccDeaths)
>pacf(USAccDeaths)
>periodogram(USAccDeaths)
```







modélisation et prévision AR

La fonction ar permet d'estimer les paramètres d'un modèle AR

$$A_p(L)X[t] = e[t]$$

Si l'ordre p n'est pas précisé, le meilleur modèle AR pour le critère AIC est sélectionné.

```
>data(lh)
>ar(lh)
Call:ar(x = lh)
Coefficients:
      1        2        3
 0.6534  -0.0636  -0.2269
Order selected 3  sigma^2 estimated as  0.1959
```

```
>ar(lh, aic = FALSE, order.max = 4) # on fixe p=4
Call:
ar(x = lh, aic = FALSE, order.max = 4)
Coefficients:
      1        2        3        4
 0.6767  -0.0571  -0.2941   0.1028
Order selected 4  sigma^2 estimated as  0.1983
```

ATTENTION x doit être une série chronologique (x=ts(x))

Autour des modèles ARMA

- ARMAacf pour le calcul des covariances théorique d'un modèle ARMA
- ARMAtoMA pour le développement en MA infinie d'un modèle ARMA
- arima.sim pour simuler des trajectoires d'un modèle ARMA ou ARIMA

Modèle ARMA : Estimation

• La fonction ar permet d'estimer les paramètres d'un processus AR.
• Pour les modèles ARMA d'ordre (p,q)

$$X[t] = a[1]X[t-1]+...+a[p]X[t-p]+e[t]+b[1]e[t-1]+...+b[q]e[t-q]$$

on utilise la fonction arima, la syntaxe est
out=arima(x,order=c(p,0,q))
la sortie out est une liste contenant :
out$coef : estimation des coefficients,
out$resid : estimation des résidus e[t]
ATTENTION x doit être une série chronologique (x=ts(x))

modélisation et prévision SARIMA

Plus généralement, la fonction `arima` permet d'estimer les paramètres d'un modèle SARIMA

$A_p(L)\alpha_P(L^s)Y[t] = \beta_Q(L^s)B_q(L)e[t]$ avec $Y[t] = (I-L)^d(I-L^s)^D X[t]$

la syntaxe est la suivante :

```
out=arima(x,order=c(p,d,q),
                    seasonal=list(order=c(P,D,Q),period=s))
```

la sortie est une liste contenant :

`out$coef` : estimation des coefficients,

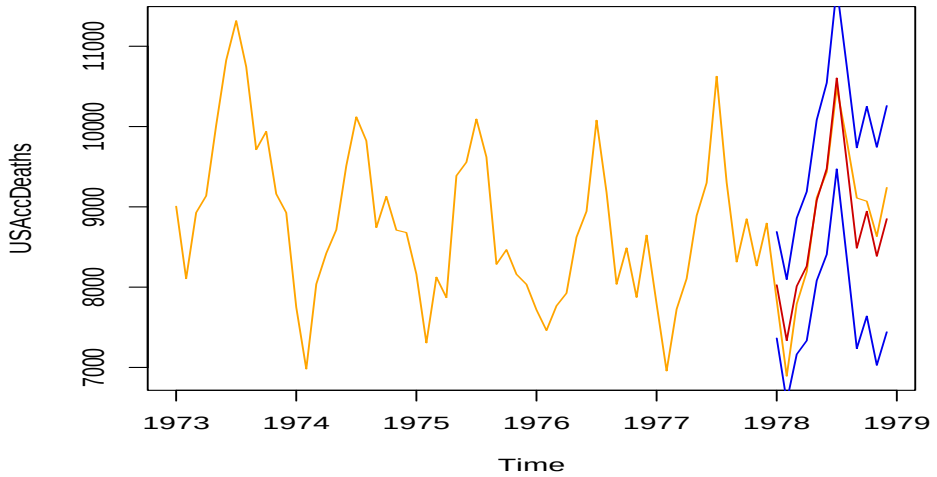
`out$aic` : critère AIC,

`out$resid` : estimation des résidus e[t]

option : include.mean=F ou T

Exemple :

```
data(USAccDeaths)
a = c(USAccDeaths)
USAcc= ts(a[1:60],frequency=12,start=c(1973,1))

fit = arima(USAcc, order=c(0,1,1), seasonal=list(order=c(0,1,1)))
Call:
arima(x = USAcc, order = c(0, 1, 1),
```



Prévision pour des modèles SARIMA

Pour prévoir à l'horizon h, on utilise la fonction `predict`

```
>out=arima0(...)
> p=predict(out,h)
p$pred   #contient les pr^^e9visions
p$se     #erreurs de pr^^e9vision ( ^^e9cart type )
```

Exemple :

```
p = predict(fit, n.ahead=12)
plot(USAccDeaths, col="orange")
lines(p$pred,col="red3")
lines(p$pred+1.96*p$se, col="blue2")
lines(p$pred-1.96*p$se, col="blue2")
```

Plus généralement : la fonction `predict`

package: stats

```
> methods(predict)
[1] "predict.ar"                "predict.arima"
[3] "predict.loess"             "predict.ppr"
[5] "predict.smooth.spline"     "predict.smooth.spline.fit"
[7] "predict.glm"               "predict.lm"
[9] "predict.mlm"
```

lissage exponentiel

La méthode de lissage de Holt & Winter est disponible sous R. Les fonctions s'appelle `HoltWinters` et `predict.HoltWinters`.
Exemple : sur la série `data(co2)`

```
data(co2)
m = HoltWinters(co2)                          #lissage
p = predict(m, 50, prediction.interval = TRUE)  # prevision
plot(m, p)
```

cette fonction peut aussi être utilisée pour

- des modèles multiplicatifs
- le lissage simple si l'on impose les coefficients.