

2 – Langage orienté objet

D. Chessel & J. Thioulouse

Résumé

Une fonction s'écrit $f(\text{objets}, \text{paramètres})$. Une partie d'un objet s'écrit $\text{objet}[\dots]$. Une fonction crée un objet $\text{objet } f(\text{objets}[\dots])$ ou un graphique. Les jeux du type $f(\text{objet}[f(\text{objet})])$ ou $\text{objet}[f(\text{objet}[f(\text{objet})])]$ et le nombre de fonctions disponibles donne un langage. Les fonctions, les données, les formules, les résultats, les graphiques sont des objets.

Plan

1.	LES VECTEURS	2
2.	LES TABLEAUX.....	3
3.	LES FONCTIONS	6
4.	LES FACTEURS	10
5.	ECRIRE UNE FONCTION.....	13
6.	LES GRAPHIQUES	14

1. Les vecteurs

L'objet de base est le vecteur :

_ Affectation

```
> w0<-7 affecte à l'objet de nom w0 la valeur 7
> w0_7 Le signe souligné a la même fonction et est plus simple à taper
> w0 Le nom de l'objet affiche son contenu
[1] 7
> > is.vector(w0)
[1] TRUE Wo est un vecteur
```

: Série d'entiers

```
> w0_1:12
> w0
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

c Combinaison

```
> w0_c(2,5,-3,8,"a")
> w0 w0 est un vecteur de chaînes de caractères
[1] "2" "5" "-3" "8" "a"
```

mode

```
> mode(w0)
[1] "character"

> w_c(1,5,-36,3.66)
> w0 w0 est un vecteur numérique
[1] 1.00 5.00 -36.00 3.66
> mode(w0)
[1] "numeric"

> w0_c(T,T,T,F,F)
> w0 w0 est un vecteur logique
[1] T T T F F
> mode(w0)
[1] "logical"
```

scan Saisie au clavier

```
> w0_scan()
1: 145
2: -1
3: 28.88
4: 2e-02
5: 3.45e1
6: Le premier retour-charriot après une chaîne vide met fin à la saisie
> w0
[1] 145.00 -1.00 28.88 0.02 34.50
```

[] Éléments des vecteurs

```
> w0[2:3] du second au troisième
[1] -1.00 28.88
> w0[4] le quatrième
[1] 0.02
> w0[-4] tous sauf le quatrième
[1] 145.00 -1.00 28.88 34.50
```

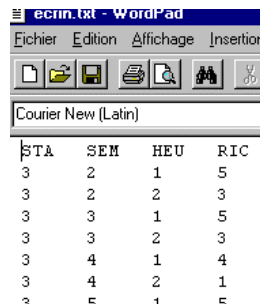
2. Les tableaux

Récupérer à <http://pbil.univ-lyon1.fr/Splus/SPLUSDEA/> le fichier de données ecrin.txt

read.table Saisie dans un fichier texte

Vérifier la réception :

	A	B	C	D
1	STA	SEM	HEU	RIC
2	3	2	1	5
3	3	2	2	3
4	3	3	1	5
5	3	3	2	3
6	3	4	1	4
7	3	4	2	1
8	3	5	1	5



The screenshot shows a WordPad window titled 'ecrin.txt - WordPad'. The menu bar includes 'Fichier', 'Edition', 'Affichage', and 'Insertion'. The text area contains the following data:

STA	SEM	HEU	RIC
3	2	1	5
3	2	2	3
3	3	1	5
3	3	2	3
3	4	1	4
3	4	2	1
3	5	1	5

```
STA SEM HEU RIC Séparateur tabulation entre les en-têtes de colonnes
3 2 1 5 Séparateur tabulation entre les nombres
3 2 2 3 idem
...
```

```
> read.table("ecrin.txt") Le fichier doit être dans le dossier de travail
  V1 V2 V3 V4
1  STA SEM HEU RIC
2    3  2  1  5
3    3  2  2  3
4    3  3  1  5
5    3  3  2  3
6    3  4  1  4
7    3  4  2  1
. . .
```

Observer que les noms de colonnes ont été pris pour le premier individu. Utiliser l'aide en ligne :

```
> help(read.table)
```

Usage:

```
read.table(file, header = FALSE, sep = "", dec = ".", quote = "\"",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           skip = 0)
```

header: header = FALSE est le paramètre par défaut. Les paramètres qui n'ont pas de valeurs par défaut doivent être affectés. Si le fichier contient les noms de colonnes utiliser header = TRUE, ou header = T ou h=T (les noms de paramètres peuvent être remplacé par une abbréviation non ambiguë).

sep: Le séparateur peut être choisi arbitrairement.

quote: Les chaînes de caractères peuvent être entre guillemets

dec: Le point décimal peut être paramétré

row.names: Utiliser row.names=1 si la première colonne du fichier contient les noms de ligne.

col.names: Les noms de variables peuvent être définies ici. Si h=T ils sont lus dans le fichier. Par défaut on aura V1, V2, ...

`as.is`: Par défaut les colonnes contenant des chaînes de caractères sont traitées comme des variables qualitatives.

`na.strings`: NA est la bonne manière d'entrer les données manquantes

`skip`: Le nombre de lignes à lire avant les données.

Toute abbréviation non ambiguë est acceptée pour les noms de paramètres (h pour header) :

```
> read.table("ecrin.txt",h=T)
  STA SEM HEU RIC
1    3  2  1  5
2    3  2  2  3
3    3  3  1  5
4    3  3  2  3
5    3  4  1  4
6    3  4  2  1
. . .
```

Lorsque la lecture est affichée correctement rappeler l'ordre (↑), déplacer le curseur vers la gauche (←) et définir l'affectation de la lecture :

```
> ecrin_read.table("ecrin.txt",h=T)
> ecrin[1:5,]
  STA SEM HEU RIC
1  3  2  1  5
2  3  2  2  3
3  3  3  1  5
4  3  3  2  3
5  3  4  1  4
. . .
```

Au moindre doute appeler le help de la fonction. Cette documentation est extrêmement précise et riche. Tous les ouvrages fondamentaux de statistique y sont cités au bon endroit. Les pages i à xiv du document `refman.pdf` (manuel de référence) contiennent la liste des fonctions des dix bibliothèques de base de R. On peut l'imprimer pour cocher les fonctions déjà rencontrées.

Les deux ordres suivants sont identiques :

```
> ?read.table
> help(read.table)
```

`ecrin` contient le nombre d'espèces d'Oiseaux (variable RIC) compté dans 1315 relevés ornithologiques exécutés dans le parc National des Ecrins en 1991. Chaque relevé est exécuté dans une station (1 à 14, variable STA), au cours d'une semaine de l'année (variable SEM de 1 à 52), soit le matin, soit le soir (variable HEU, 1-matin, 2-soir). Ces données sont extraites de la convention d'étude n° 228/92 du Parc National des Ecrins.

```
> is.vector(ecrin) ecrin n'est pas un vecteur
[1] F
> is.matrix(ecrin) ecrin est une matrice
[1] T
> is.data.frame(ecrin) ecrin est un data.frame
[1] T
> class(ecrin)
[1] "data.frame"
```

ecrin est un data.frame (tableau dont les colonnes ont des propriétés variées comme nom, type, ...). On peut mélanger dans un data.frame les variables quantitatives, qualitatives, logiques et textuelles. Une matrice ne contient qu'un type d'enregistrements. ecrin appartient à la classe des data.frame laquelle contient la classe des matrices. Une matrice n'est pas un data.frame mais peut le devenir :

```
> w0_matrix("a",nrow=3,ncol=2)
> w0
      [,1] [,2]
[1,] "a"  "a"
[2,] "a"  "a"
[3,] "a"  "a"
> is.data.frame(w0)
[1] F
> as.data.frame(w0)
  V1 V2
1  a  a
2  a  a
3  a  a
```

[] Éléments des matrices

```
> ecrin[1:4,]
  STA SEM HEU RIC
1   3   2   1   5
2   3   2   2   3
3   3   3   1   5
4   3   3   2   3
> dim(ecrin)
[1] 1315   4
> ecrin[1312:1315,]
  STA SEM HEU RIC
1312  12  51   1   5
1313  12  51   2   4
1314  12  52   2   4
1315  12  52   1   7
```

Les variables d'un data.frame sont directement accessibles par leur nom :

```
> ric_ecrin$RIC
> is.vector(ric)
[1] T
```

ecrin\$RIC ou ric sont strictement équivalents.

getwd Nom du dossier de travail

```
> getwd()
[1] "E:\\Pedago\\Rprojet"
```

object Liste des objets du dossier de travail

```
> objects()
[1] "ecrin" "w0"
```

Tout ce que fait R est consigné dans deux fichiers RData et RHistory qui sont mis à jour quand on sort de la session.

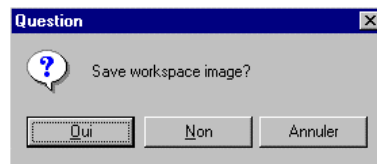
.Rhistory contient la liste des ordres envoyés :

```

.Rhistory - WordPad
Fichier Edition Affichage Insertion Format ?
w0
w0_1:15
w0
read.table(<< e:\pedago\rprojet\datatxt\ecrin.txt >>)
read.table(" e:\pedago\rprojet\datatxt\ecrin.txt")
read.table(" e:\\pedago\\rprojet\\datatxt\\ecrin.txt")
read.table("e:\\pedago\\rprojet\\datatxt\\ecrin.txt")
read.table("e:\\pedago\\rprojet\\datatxt\\ecrin.txt",h=T)
ecrin_read.table("e:\\pedago\\rprojet\\datatxt\\ecrin.txt",h=T)

```

.Rdata n'est pas accessible mais contient en code interne les objets de la session en cours.
 Quitter R :



Laisser faire la mise à jour. Consulter l'historique. Refermer l'historique. Relancer R et vérifier que tout est en place :

```

> getwd()
[1] "E:\\Pedago\\Rprojet"
> objects()
[1] "ecrin" "ric" "w0"

```

rm Détruire un objet

```

> rm(ric)
> objects()
[1] "ecrin" "w0"
> is.vector(ric)
Error in is.vector(ric) : Object "ric" not found
> ric_ecrin$RIC

```

3. Les fonctions

summary/plot Fonctions génériques

```

> ?summary

`summary' is a generic function used to produce result summaries
of the results of various model fitting functions. The function
invokes particular `methods' which depend on the `class' of the
first argument.

> ?plot

Generic function for plotting of R objects. For more details
about the graphical parameter arguments, see `par'.

```

Les fonctions génériques s'appliquent à plusieurs types d'objets qui peuvent être très différents entre eux. Une fonction générique *func* fonctionne sur une classe d'objets *clas* si il existe une fonction *func.clas*. Par exemple, plot et summary travaille sur des data.frames, des facteurs, des modèles linéaires généralisés, ...

```

plot.data.frame(data, labels = dimnames(data)[[2]], ...)
summary.data.frame(object, maxsum, ...)
plot.factor(x, y=NULL, style="box", rotate=<<see below>>, ...)
summary.factor(object, maxsum)
plot.glm(glm.obj, residuals = NULL, smooths = F, rugplot = F, , ...)
summary.glm(object, dispersion=NULL, correlation=T)
...

```

```

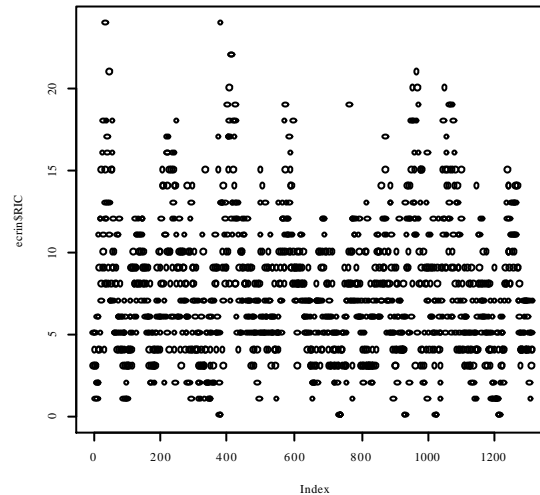
> summary(ric)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.000  4.000   7.000   7.477 10.000  24.000

```

```

> plot(ecrin$RIC)

```



```

> ric[2] la seconde composante de ric
[1] 3
> ric[c(1,10,100)] les composantes de rang 1, 10 et 100
[1] 5 3 2
> ric[ric<1] les composantes de ric pour lesquelles ric est < 1 !
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> length(ric)
[1] 1315

```

append/replace

Ajouter des valeurs, remplacer des valeurs

```

> ric[ric>20]
[1] 24 21 24 22 22 21
> (1:1315)[ric>20]
[1] 38 48 382 414 415 968
> ric[(1:1315)[ric>20]]
[1] 24 21 24 22 22 21

```

ric>20, 1:1315, (1:1315)[ric>20], ric, ric[ric>20] sont des vecteurs

unique/sort/rank/order

```

> x_ric[1:10]
> x
[1] 5 3 5 3 4 1 5 3 2 3 Le vecteur x
> unique(x)
[1] 5 3 4 1 2 les valeurs que peuvent prendre les composantes du vecteur

```

```

> sort(x)
[1] 1 2 3 3 3 3 4 5 5 5 le vecteur des composantes rangées par ordre croissant
> rank(x)
[1] 9.0 4.5 9.0 4.5 7.0 1.0 9.0 4.5 2.0 4.5 le vecteur des rangs des composantes
> order(x)
[1] 6 9 2 4 8 10 5 1 3 7 la permutation qui donne le rangement par
ordre croissant
> x[order(x)]
[1] 1 2 3 3 3 3 4 5 5 5
> sort(x)
[1] 1 2 3 3 3 3 4 5 5 5
> x[order(x)]==sort(x)
[1] T T T T T T T T T T le vecteur qui donne la comparaison élément par élément
des deux vecteurs
> x[order(x)]>sort(x)
[1] F F F F F F F F F F

```

sum/prod/cumsum/cumprod/diff

```

> sum(x)
[1] 34
> prod(x)
[1] 81000
> cumsum(x)
[1] 5 8 13 16 20 21 26 29 31 34
> cumprod(x)
[1] 5 15 75 225 900 900 4500 13500 27000 81000
> diff(x)
[1] -2 2 -2 1 -3 4 -2 -1 1
> diff(x,lag=2)
[1] 0 0 -1 -2 1 2 -3 0

```

Calcul sur les vecteurs

```

> 2*x
[1] 10 6 10 6 8 2 10 6 4 6
> x+10
[1] 15 13 15 13 14 11 15 13 12 13
> abs(2*x-10)
[1] 0 4 0 4 2 8 0 4 6 4
> log(x)
[1] 1.6094 1.0986 1.6094 1.0986 1.3863 0.0000 1.6094 1.0986 0.6931
[10] 1.0986
> sqrt(x)
[1] 2.236 1.732 2.236 1.732 2.000 1.000 2.236 1.732 1.414 1.732

```

rep/seq/rev

rep pour faire des répétitions, seq pour des suites, rev pour inverser :

```

> rep("a",3)
[1] "a" "a" "a"
> rep(1,3)
[1] 1 1 1
> rep(c(1,2),3)
[1] 1 2 1 2 1 2
> rep(c(1,2),c(3,3))
[1] 1 1 1 2 2 2
> seq(from=1,to=10,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5
[13] 7.0 7.5 8.0 8.5 9.0 9.5 10.0
> seq(from=1,to=10,le=3)
[1] 1.0 5.5 10.0
> seq(from=1,to=10,le=5.5)

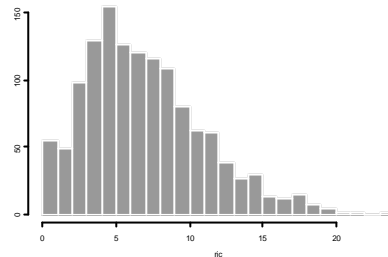
```



```
[1] 1 3 5 7 10
> seq(from=1,to=10,by=4)
[1] 1 5 9
> rep(1:3,1:3)
[1] 1 2 2 3 3 3
> rev(6:12)
[1] 12 11 10 9 8 7 6
```

Statistiques élémentaires

```
> hist(ric,nclass=30)
```



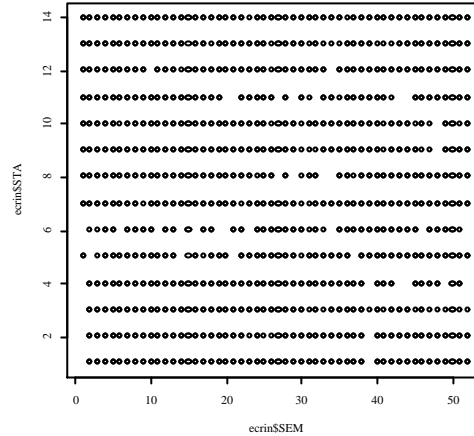
```
> min(ric)
[1] 0
> max(ric)
[1] 24
> range(ric)
[1] 0 24
> mean(ric)
[1] 7.477
> median(ric)
[1] 7
> quantile(ric, probs=seq(from=0, to=1,by=0.1))
 0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
 0   3   4   5   6   7   8   9  11  13  24
```

Les fonctions sont des objets

```
> ftoto_function(x) {return(2*x)}
> ftoto
function(x) {return(2*x)}
> ftoto(3)
[1] 6
> is.function(ftoto)
[1] TRUE
```

Voir aussi : ceiling/floor/trunc

```
> plot(ecrin$SEM,ecrin$STA)
```



Le plan d'observations a peu de « trous ».

4. Les facteurs

```
> sem_ecrin$SEM
> is.numeric(sem)
[1] TRUE
> summary(sem)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00  13.00   26.00   26.24  39.00   52.00
```

sem est un variable quantitative

```
> sem.fac_factor(sem)
> summary(sem.fac)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28
18 25 26 28 28 25 28 28 27 26 26 28 28 23 26 26 27 27 26 24 24 27 24 27 27 24 24
28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
24 23 22 25 23 21 26 27 25 28 21 27 24 26 24 24 26 25 25 23 25 28 25 23

> is.numeric(sem.fac)
[1] FALSE
```

sem.fac est un variable qualitative. Ses modalités sont :

```
> levels(sem.fac)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
[31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"
[46] "46" "47" "48" "49" "50" "51" "52"
```

```
> heu_ecrin$HEU
> heu.fac_factor(heu)
> levels(heu.fac)
[1] "1" "2"

> levels(heu.fac)_c("Ma", "So")
> levels(heu.fac)
[1] "Ma" "So"

> summary(heu.fac)
Ma  So
```

```
672 643
> table(ecrin$HEU)

  1  2
672 643
```

table *tables de contingence*

```
> table(ecrin$HEU) Tabuler une variable
```

```
  1  2
672 643
```

```
> table(heu.fac) Tabuler un facteur
```

```
heu.fac
  Ma  So
672 643
```

```
> table(heu.fac,sem.fac) Tabuler deux facteurs
```

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
Ma 9 12 13 14 14 12 14 14 13 13 13 14 14 11 14 13 14 14 13 12 12
So 9 13 13 14 14 13 14 14 14 13 13 14 14 12 12 13 13 13 13 12 12
...
  43 44 45 46 47 48 49 50 51 52
Ma 12 12 12 13 14 12 13 14 14 12
So 12 12 14 12 11 11 12 14 11 11
```

```
> sta.fac_factor(ecrin$STA)
```

```
> table(sta.fac,heu.fac,sem.fac) Tabuler trois facteurs
```

```
, , 1 par semaine
```

```
  Ma So
  1  0  0
  2  0  0
  3  0  0
  4  0  0
  5  1  1
...
14  1  1
```

```
, , 2
```

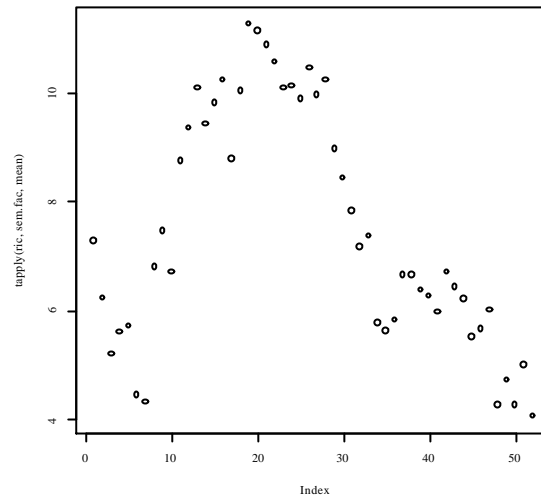
```
  Ma So
  1  1  1
...
  5  0  0
  6  0  1
  7  1  1
...
14  1  1
```

```
, , 3
```

```
  Ma So
  1  1  1
  2  1  1
...
```

tapply *Appliquer une fonction à un vecteur par blocs*

```
> tapply(ric,heu.fac,mean)
  Ma  So
8.972 5.914
> plot(tapply(ric,sem.fac,mean))
```



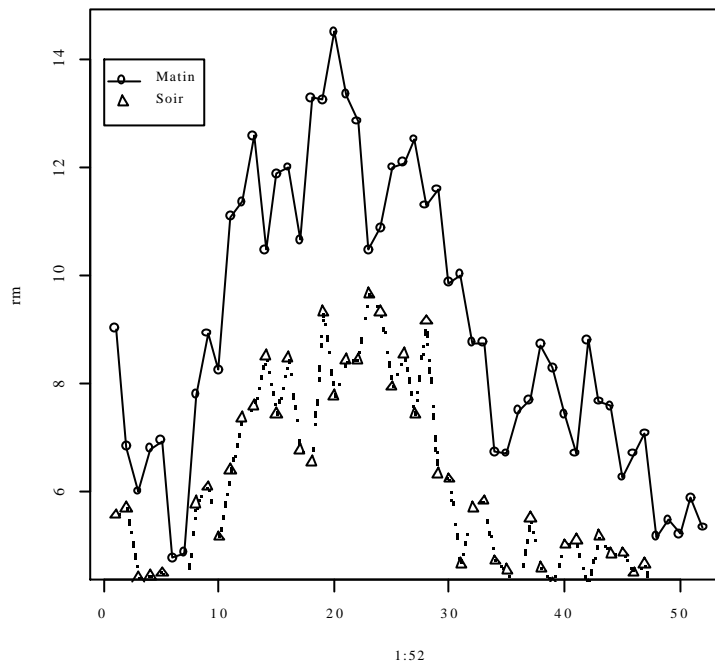
```

> tapply(ric,list(sem.fac,heu.fac),mean)
      Ma      So
1  9.000  5.556
2  6.833  5.692
3  6.000  4.385
...
50  5.214  3.286
51  5.857  3.909
52  5.333  2.636

> ric.sem.heu_tapply(ric,list(sem.fac,heu.fac),mean)
> ts.plot(ric.sem.heu)

> plot(1:52,rm,pch=1)
> points(1:52,rs,pch=2)
> lines(1:52,rm,lty=1)
> lines(1:52,rs,lty=2)
> help("legend")
> legend(0,14,c("Matin","Soir"),lty=c(1,2),pch=c(1,2))

```

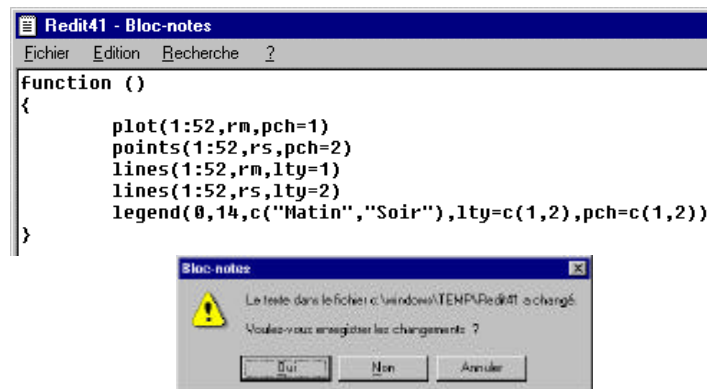


5. Ecrire une fonction

Utiliser une fonction pour tracer ce graphique ;

```
> fix(fgraph)
```

Taper dans la fenêtre de l'éditeur et sauvegarder le résultat :



```
> fgraph
function ()
{
  plot(1:52,rm,pch=1)
  points(1:52,rs,pch=2)
  lines(1:52,rm,lty=1)
  lines(1:52,rs,lty=2)
  legend(0,14,c("Matin","Soir"),lty=c(1,2),pch=c(1,2))
}
<environment: 02A329E8>
> fgraph()          refait le dessin
> dput(fgraph,"fgraph.txt")  garde la fonction dans un fichier texte
```

```

> dget("fgraph.txt")      lit le fichier sans affectation
function ()
{
  plot(1:52, rm, pch = 1)
  points(1:52, rs, pch = 2)
  lines(1:52, rm, lty = 1)
  lines(1:52, rs, lty = 2)
  legend(0, 14, c("Matin", "Soir"), lty = c(1, 2), pch = c(1,
  2))
}
> dget("fgraph.txt")()    refait le dessin

```

Ouvrir alors directement le fichier fgraph.txt :

Fermer et sauvegarder.

```

> dget("fgraph.txt")
Error in parse(file, n, text, prompt) : syntax error on line 8

```

Ouvrir à nouveau. Ajouter la) fermante. Sauvegarder.

```

> dget("fgraph.txt")
function ()
{
  plot(1:52, rm, pch = 1)
  points(1:52, rs, pch = 2)
  #lines(1:52, rm, lty = 1)
  #lines(1:52, rs, lty = 2)
  legend(0, 14, c("Matin", "Soir"), lty = c(1, 2))
}
<environment: 028CC5E0>
> toto_dget("fgraph.txt")
> toto()      Refait un dessin modifié
> dput(function() {}, "toto1")  Initialise l'écriture d'une fonction

```

Observer que l'écriture d'une fonction après fix gèle la console de R jusqu'à fermeture du fichier texte. La fonction est immédiatement analysée et disponible s'il n'y a pas d'erreur. Par contre avec dput la console de R est encore utilisable mais il faut relancer la lecture du fichier au clavier. Le fichier texte peut rester ouvert.

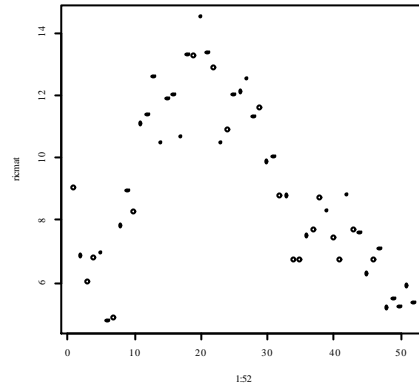
6. Les graphiques

```

> ricmat_ric.sem.heu[,1]
> ricmat
   1    2    3    4    5    6    7    8    9   10   11
9.000 6.833 6.000 6.786 6.929 4.750 4.857 7.786 8.923 8.231 11.077
  12   13   14   15   16   17   18   19   20   21   22
11.357 12.571 10.455 11.857 12.000 10.643 13.286 13.231 14.500 13.333 12.846
...
  45   46   47   48   49   50   51   52
6.250 6.692 7.071 5.167 5.462 5.214 5.857 5.333

```

```
> plot(1:52,ricmat)
```



```
> x_1:52
```

lowess

lowess

package:base

R Documentation

Scatter Plot Smoothing

Description:

This function performs the computations for the LOWESS smoother (see the reference below). `lowess` returns a list containing components `x` and `y` which give the coordinates of the smooth. The smooth should be added to a plot of the original points with the function `lines`.

Usage:

```
lowess(x, y, f=2/3, iter=3, delta=.01*diff(range(x)))
```

Arguments:

`x`, `y`: vectors giving the coordinates of the points in the scatter plot. Alternatively a single plotting structure can be specified.

`f`: the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness.

`iter`: the number of robustifying iterations which should be performed. Using smaller values of `iter` will make `lowess` run faster.

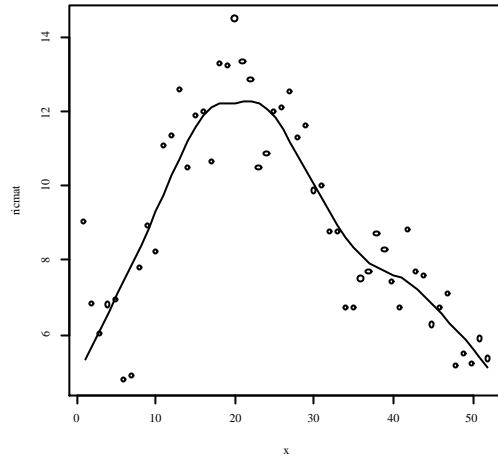
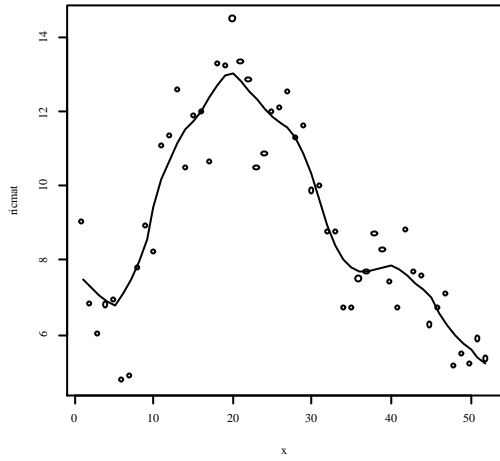
`delta`: values of `x` which lie within `delta` of each other replaced by a single value in the output from `lowess`.

References:

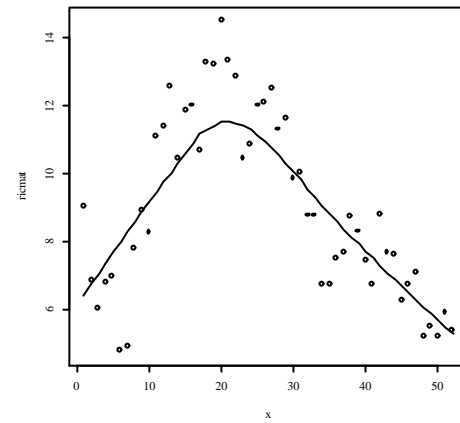
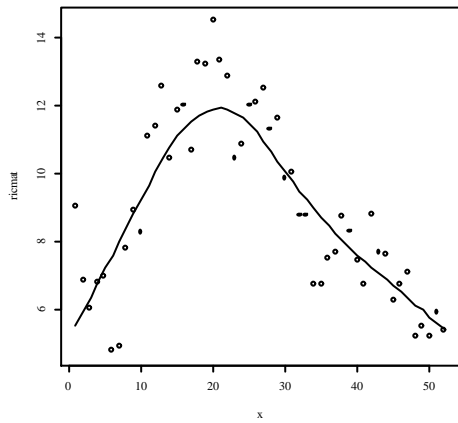
Cleveland, W. S. (1979) Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.* 74, 829-836.

Cleveland, W. S. (1981) LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35, 54.

```
> plot(x,ricmat)
> lines(lowess(x,ricmat,f=0.20))
> lines(lowess(x,ricmat,f=0.35))
```

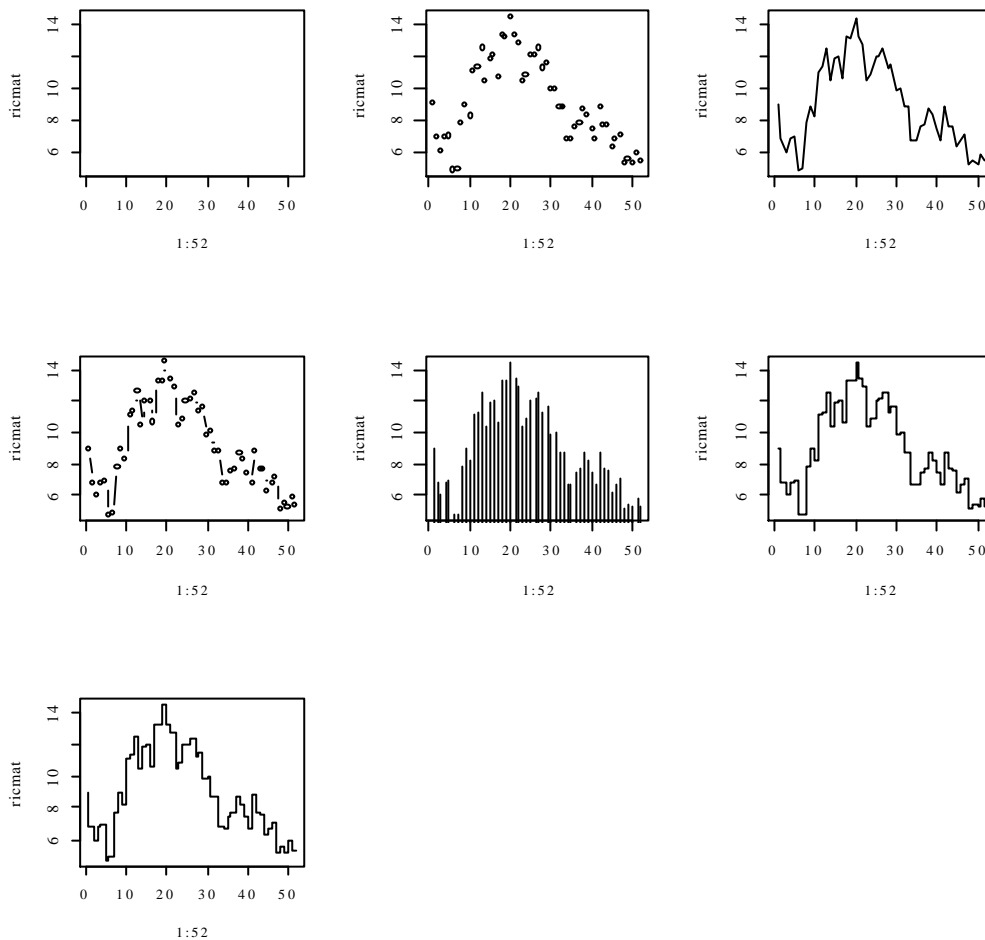


```
> lines(lowess(x,ricmat,f=0.50))
> lines(lowess(x,ricmat,f=0.65))
```



Difficile de choisir

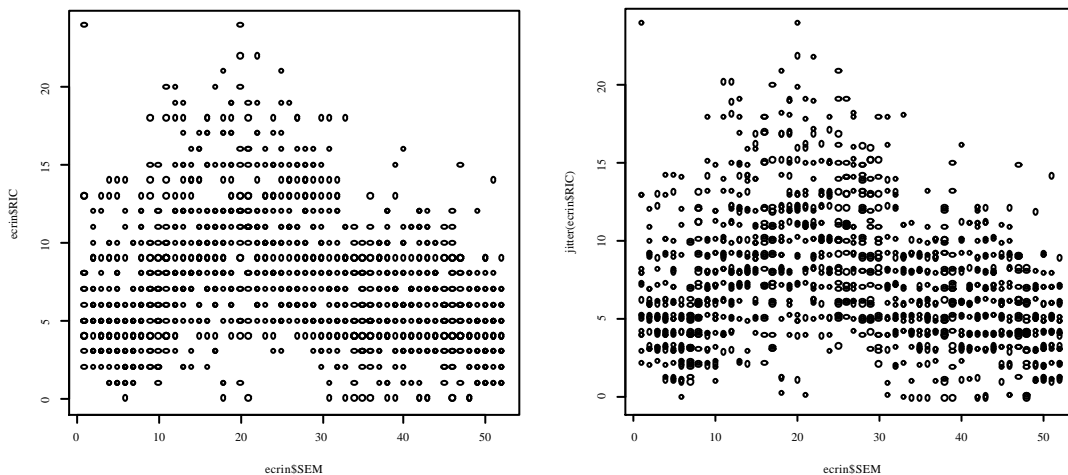
```
> par(mfrow=c(3,3))
> plot(1:52,ricmat,type="n")
> plot(1:52,ricmat,type="p")
> plot(1:52,ricmat,type="l")
> plot(1:52,ricmat,type="b")
> plot(1:52,ricmat,type="h")
> plot(1:52,ricmat,type="s")
> plot(1:52,ricmat,type="S")
```

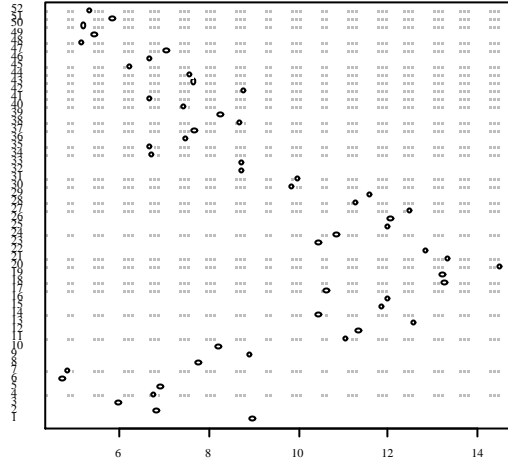
Au choix

Pour gérer les superpositions :

```
> plot(ecrin$SEM,ecrin$RIC)
> plot(ecrin$SEM,jitter(ecrin$RIC))
```



```
> dotplot(ricmat)
```



split

```
> split(ric,sta.fac)
$"1"
 [1] 3 4 3 5 5 6 7 9 4 6 7 4 10 6 4 6 10 5 9 6 8 12 6 15 14
 [26] 8 14 6 2 7 11 11 9 17 6 14 16 5 16 10 10 17 10 15 4 10 8 10 6 15
 [51] 16 9 12 15 6 13 5 14 16 4 12 14 2 18 8 11 9 6 9 6 6 10 12 9 4
 [76] 8 7 6 10 3 10 7 5 10 9 10 9 11 7 8 4 7 5 8 5 6 6 14 2 6

$"2"
 [1] 10 6 4 2 10 10 12 1 4 13 14 9 4 8 5 9 7 8 7 10 4 6 4 9 7
 [26] 4 5 4 1 3 7 5 2 3 2 5 1 3 3 4 6 11 3 10 7 3 1 7 3 3
 [51] 3 15 4 4 5 2 2 2 5 2 1 1 2 6 3 1 5 4 1 5 6 7 7 2 2
 [76] 9 3 2 7 11 9 8 8 5 9 5 4 6 9 5 17 15 5 2 0 0 0

...

$"14"
 [1] 5 8 9 12 7 4 4 5 6 1 2 3 5 5 9 7 15 9 6 12 11 18 12 20 15
 [26] 5 9 14 16 9 11 17 6 15 10 14 12 19 12 13 15 19 5 6 9 10 15 6 19 7
 [51] 9 11 18 16 14 5 15 8 13 4 14 7 13 9 11 9 4 8 4 11 5 7 4 12 6
 [76] 15 8 4 5 7 4 9 3 5 6 3 5 6 6 5 6 9 3 4 4 5 3 8 5 3
 [101] 4 3

> ricsplit_split(ric,sta.fac)
> is.list(ricsplit)
[1] TRUE
```

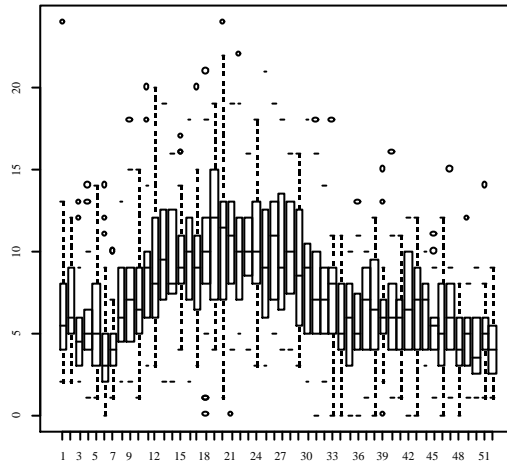
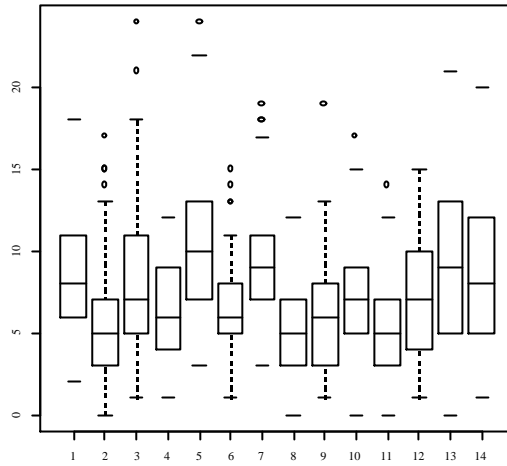
Le résultat est une liste. Cette structure de données est très importante. Chaque composante est un objet de type quelconque).

```
> unlist(lapply(ricsplit,mean))
 1      2      3      4      5      6      7      8      9     10     11
8.600 5.412 8.228 6.533 10.674 6.403 9.000 5.054 6.375 7.316 5.126
 12     13     14
7.378 9.362 8.706
```

On a la richesse moyenne par station.

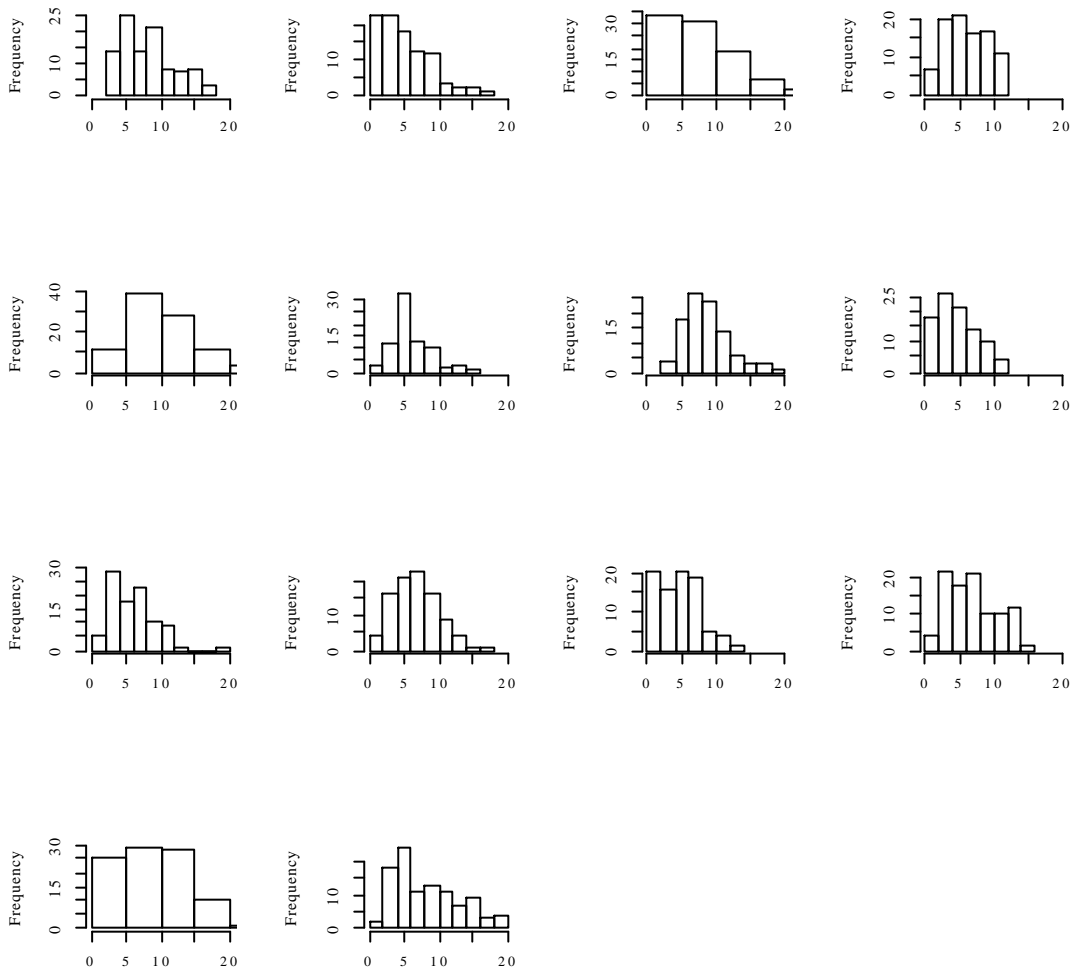
boxplot

```
> boxplot(ricsplit)
> boxplot(split(ric,sem.fac))
```



hist

```
> par(mfrow=c(4,4))
> lapply(ricsplit,hist,xlim=c(0,20),main="",xlab="")
```



coplot

coplot

package:base

R Documentation

Conditioning Plots

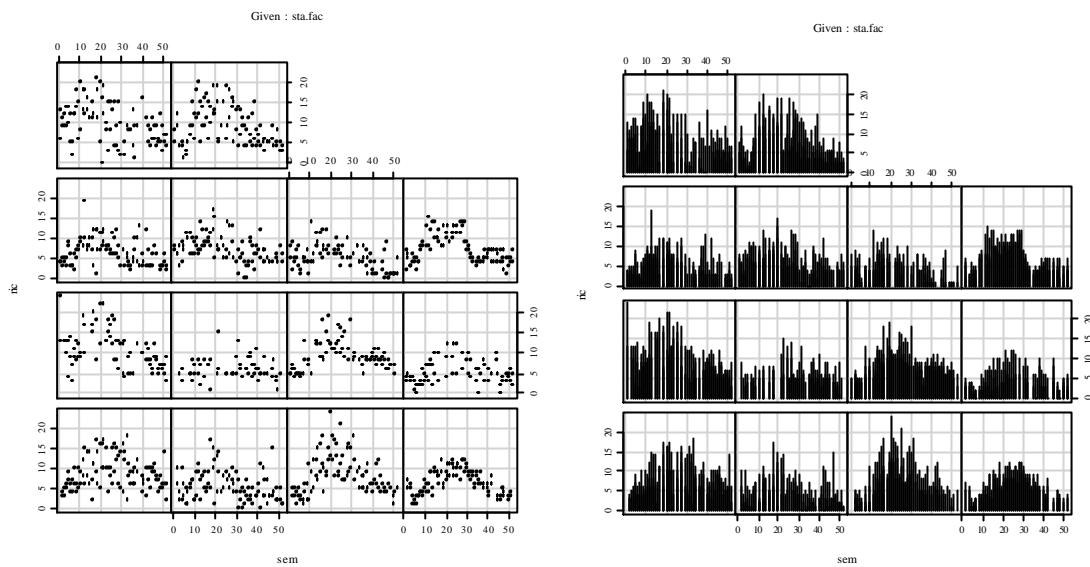
Description:

This function produces two variants of the conditioning plots discussed in the reference below.

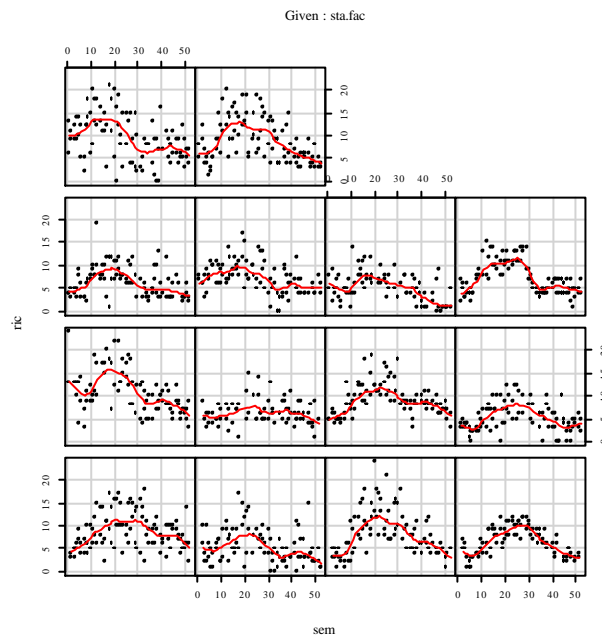
Usage:

```
coplot(formula, data, given.values, panel = points, rows, columns,
       show.given = TRUE, col = par("fg"), pch = par("pch"),
       xlab = paste("Given :", a.name),
       ylab = paste("Given :", b.name),
       number = 6, overlap = 0.5, ...)
co.intervals(x, number = 6, overlap = 0.5)
```

```
> coplot(ric~sem|sta.fac,show=F) Richesse fonction de la semaine par station
> coplot(ric~sem|sta.fac,show=F,type="h")
```



```
> coplot(ric~sem|sta.fac,show=F,panel=function(x, y, ...) panel.smooth(x, y, span = .25,
...))
```



Ces premiers exemples peuvent vous inviter à consulter un ouvrage fabuleux :

Cleveland, W.S. (1994) *The elements of graphing data*. AT&T Bell Laboratories, Murray Hill, New Jersey. 297 p.