
Graphiques (Données écologiques)

A.B. Dufour, S. Dray, D. Chessel, & J.R. Lobry

Avertissement. Dans la première partie de la fiche, toutes les commandes sont explicites. Puis, petit à petit, l'utilisateur est amené à réfléchir et à écrire lui-même les fonctions nécessaires à la réalisation du problème exposé.

Table des matières

1	Richesse faunistique	1
2	Chroniques par station	11
3	Annexe : Paramètres graphiques	18
3.1	Liste des points disponibles pour les graphiques	18
3.2	Liste des traits disponibles pour les graphiques	19

1 Richesse faunistique


Lire à <http://pbil.univ-lyon1.fr/R/donnees/ecrin.txt> le fichier de données `ecrin.txt`.

```
read.table("http://pbil.univ-lyon1.fr/R/donnees/ecrin.txt", h = TRUE)
```

Lorsque la lecture est affichée correctement rappeler l'ordre (flèche ↑), et ranger le résultat dans l'objet `ecrin` avec l'opérateur d'affectation `->` :

```
read.table("http://pbil.univ-lyon1.fr/R/donnees/ecrin.txt", h = TRUE) -> ecrin
```

```
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 l'aide de la fonction `help`. Cette documentation est extrêmement précise et riche. Tous les ouvrages fondamentaux de statistique y sont cités au bon endroit. Les pages `i` à `xxiv` du manuel de référence (`refman.pdf`) contiennent la liste des fonctions des dix bibliothèques de base de . 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 effectués dans le parc National des Écrins en 1991. Chaque relevé est effectué 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 Écrins.

```
is.vector(ecrin)
[1] FALSE
is.matrix(ecrin)
[1] FALSE
is.data.frame(ecrin)
[1] TRUE
class(ecrin)
[1] "data.frame"
```

`ecrin` est un data frame (tableau dont les colonnes ont des propriétés variées comme nom, type, ...). Dans un data frame, on peut mélanger les variables quantitatives, qualitatives, logiques et textuelles. Par opposition, une matrice - qui se présente sous la même forme qu'un data frame - ne contient qu'un type d'enregistrements. `ecrin` appartient à la classe des data frames 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] FALSE
as.data.frame(w0)
  V1 V2
1 a a
2 a a
3 a a
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] TRUE
```

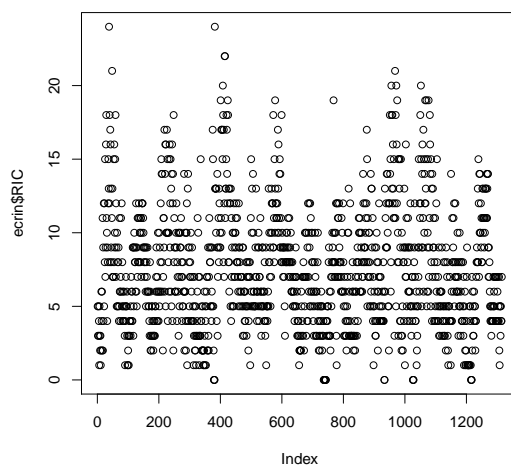
ecrin\$RIC ou ric sont strictement équivalents.

```
help(summary)
help(plot)
```

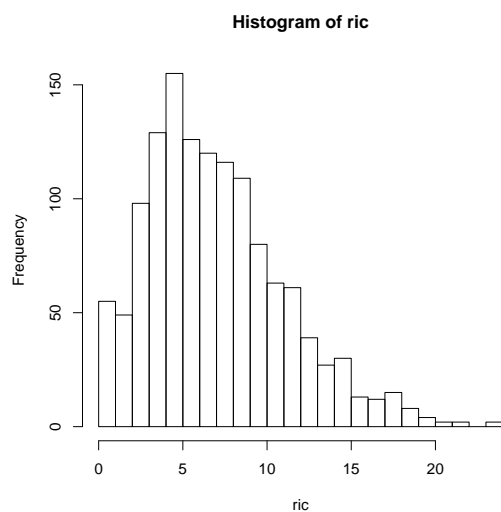
Les fonctions génériques s'appliquent à plusieurs types d'objets qui peuvent être très différents entre eux. Une fonction générique *func* fonctionne sur une classe d'objets *clas* si il existe une fonction *func.clas*. Par exemple, `summary` travaillent sur des facteurs, des tables de contingence, des modèles linéaires, ...

```
summary.factor(object, maxsum)
summary.table(object)
summary.lm(object, correlation=F, symbolic.cor=FALSE)
...
```

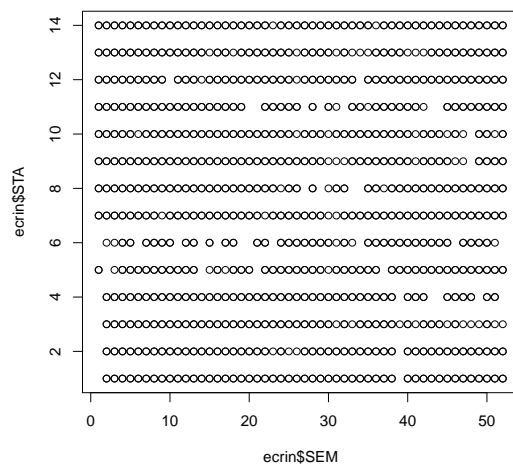
```
summary(ric)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.000  4.000   7.000   7.477 10.000  24.000
plot(ecrin$RIC)
```



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



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



Le plan d'observations a peu de " trous ".

```
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"
[17] "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30" "31" "32"
[33] "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45" "46" "47" "48"
[49] "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

```

Tabuler une variable :

```

table(ecrin$HEU)
 1  2
672 643
table(heu.fac)
heu.fac
Ma So
672 643

```

Les fonctions `summary` et `table` donnent les mêmes résultats lorsqu'il n'y a aucune donnée manquante dans la série statistique étudiée. Seule la fonction `table` élimine "naturellement" les données manquantes.

```

tempo <- c("Ma", "Ma", NA, "So", "Ma", "So", "So", "So")
tempo <- factor(tempo)
summary(tempo)
Ma So NA's
 3  4  1
table(tempo)
tempo
Ma So
 3  4

```

Tabuler deux variables :

```

table(heu.fac, sem.fac)
      sem.fac
heu.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
Ma     9 12 13 14 14 12 14 14 13 13 13 14 14 11 14 13 14 14 13 12 12 13 13 14 13
So     9 13 13 14 14 13 14 14 14 13 13 14 14 12 12 13 13 13 13 12 12 14 11 13 14
      sem.fac
heu.fac 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Ma    13 12 14 12 14 13 12 12 11 13 14 13 14 11 14 13 14 12 12 12 13 14 12 13 14
So    11 12 14 12  9  9 13 11 10 13 13 12 14 10 13 11 12 12 12 14 12 11 11 12 14
      sem.fac
heu.fac 51 52
Ma    14 12
So    11 11

```

Tabuler trois variables :

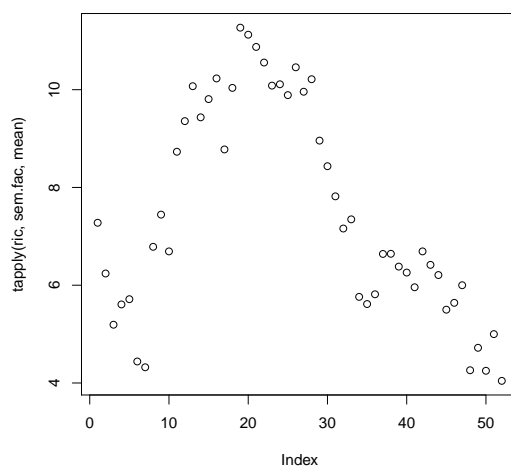
```
sta.fac <- factor(ecrin$STA)
table(sta.fac, heu.fac, sem.fac)
```

Dans ce dernier cas, nous montrons seulement les résultats de la semaine 1.

```
sta.fac <- factor(ecrin$STA)
table(sta.fac, heu.fac, sem.fac == 1)
, , = FALSE
      heu.fac
sta.fac Ma So
  1    50  50
  2    49  48
  3    49  43
  4    46  46
  5    47  43
  6    38  39
  7    48  50
  8    47  44
  9    49  45
 10    48  45
 11    43  42
 12    47  49
 13    51  41
 14    51  49
, , = TRUE
      heu.fac
sta.fac Ma So
  1     0  0
  2     0  0
  3     0  0
  4     0  0
  5     1  1
  6     0  0
  7     1  1
  8     1  1
  9     1  1
 10     1  1
 11     1  1
 12     1  1
 13     1  1
 14     1  1
```

`tapply` : *Appliquer une fonction à un vecteur par blocs*

```
tapply(ric, heu.fac, mean)
      Ma      So
8.971726 5.914463
plot(tapply(ric, sem.fac, mean))
```

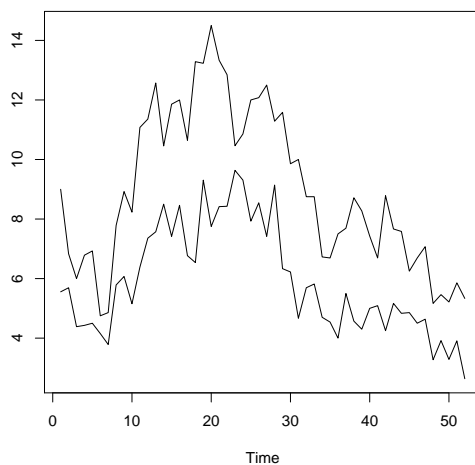


```
tapply(ric, list(sem.fac, heu.fac), mean)
```

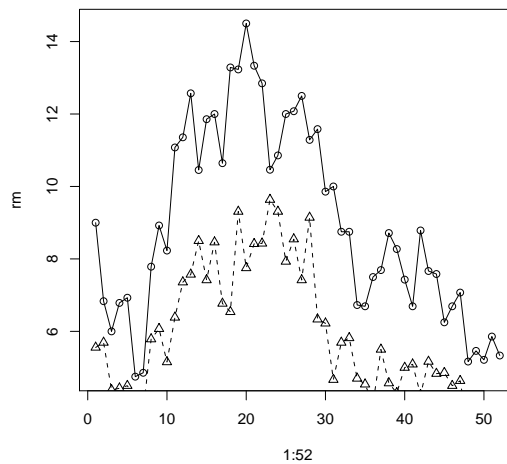
Seules les cinq premières et les trois dernières moyennes sont affichées.

```
tapply(ric, list(sem.fac, heu.fac), mean)[c(1:5, 50:52), ]
      Ma      So
1  9.000000 5.555556
2  6.833333 5.692308
3  6.000000 4.384615
4  6.785714 4.428571
5  6.928571 4.500000
50 5.214286 3.285714
51 5.857143 3.909091
52 5.333333 2.636364
```

```
ric.sem.heu <- tapply(ric, list(sem.fac, heu.fac), mean)
ts.plot(ric.sem.heu)
```

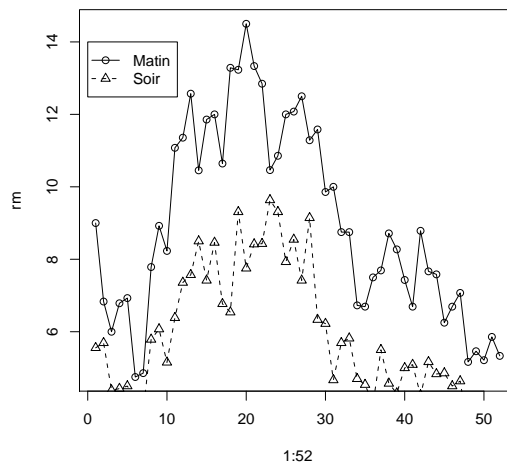


```
rm <- ric.sem.heu[, 1]
rs <- ric.sem.heu[, 2]
plot(1:52, rm, pch = 1)
points(1:52, rs, pch = 2)
lines(1:52, rm, lty = 1)
lines(1:52, rs, lty = 2)
```

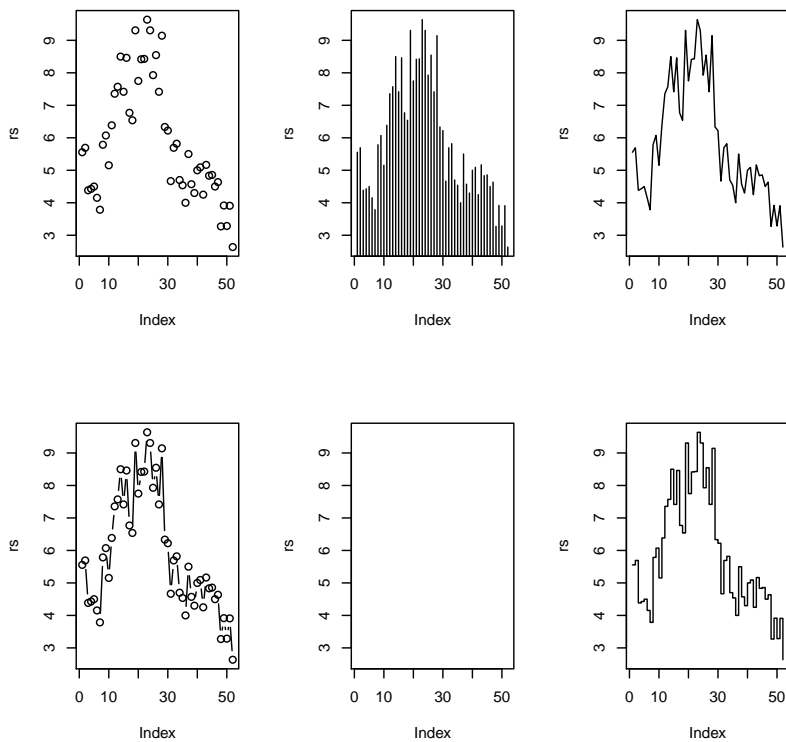


```
help(legend)
```

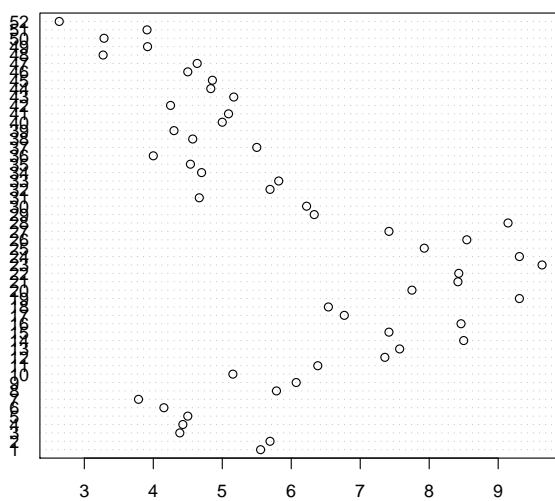
```
rm <- ric.sem.heu[, 1]
rs <- ric.sem.heu[, 2]
plot(1:52, rm, pch = 1)
points(1:52, rs, pch = 2)
lines(1:52, rm, lty = 1)
lines(1:52, rs, lty = 2)
legend(0, 14, c("Matin", "Soir"), lty = c(1, 2), pch = c(1, 2))
```



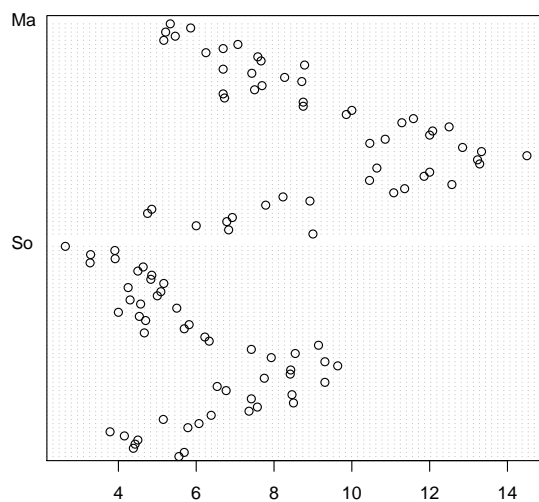
```
par(mfrow = c(2, 3))
plot(rs)
plot(rs, type = "h")
plot(rs, type = "l")
plot(rs, type = "b")
plot(rs, type = "n")
plot(rs, type = "s")
```



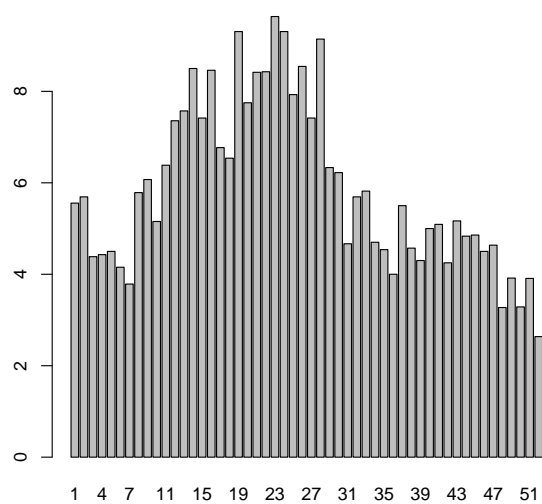
```
par(mfrow = c(1, 1))
dotchart(rs)
```



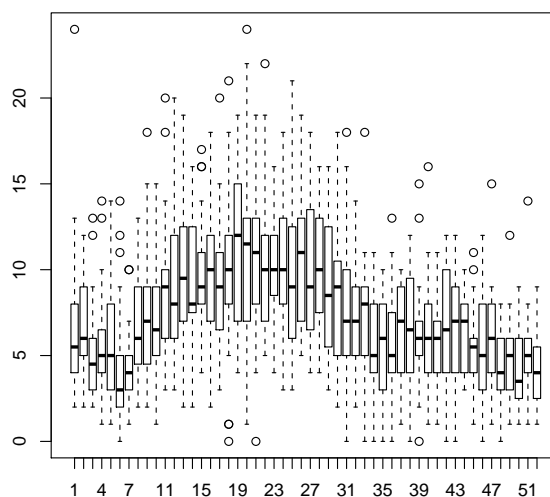
```
dotchart(ric.sem.heu, lab = "")
```



```
range(rs)  
[1] 2.636364 9.636364  
barplot(rs)
```



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



Etudier les fonctions `boxplot` et `boxplot.stats`.

2 Chroniques par station

```
ecrinsoir <- ecrin[heuf.fac == "So", ]
dim(ecrinsoir)
[1] 643 4

ecrinsoir <- ecrinsoir[, c("RIC", "SEM", "STA")]
dim(ecrinsoir)
[1] 643 3
```

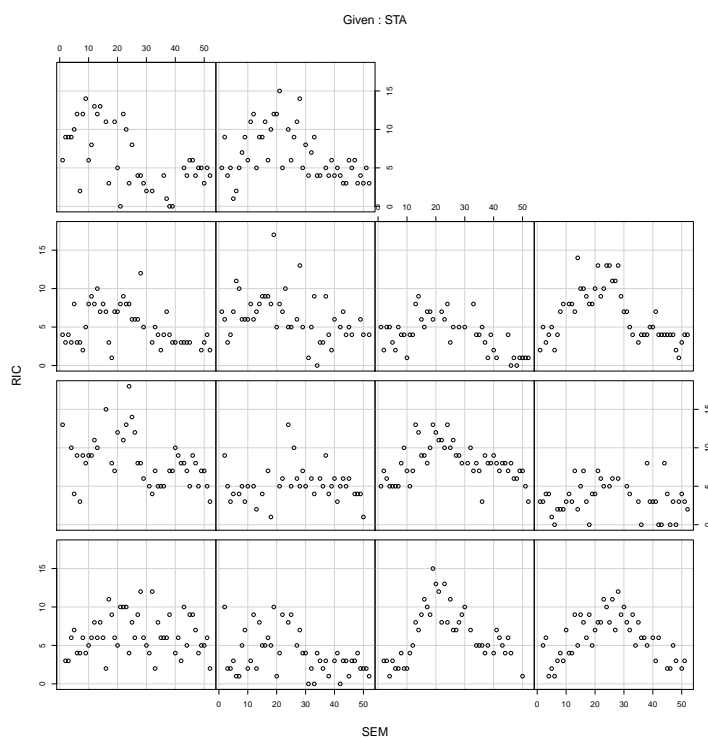
Transformer la variable `ecrin$STA` du data frame `ecrin` afin d'obtenir les résultats affichés ci-dessous :

```
summary(ecrinsoir)
  RIC          SEM          STA
Min. : 0.000  Min. : 1.00  7   : 51
1st Qu.: 4.000  1st Qu.:13.00 1   : 50
Median : 5.000  Median :25.00 12  : 50
Mean   : 5.914  Mean   :25.95 14  : 50
3rd Qu.: 8.000  3rd Qu.:39.00 2   : 48
Max.   :18.000  Max.   :52.00 4   : 46
      (Other):348

levels(ecrinsoir$STA)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14"
levels(ecrinsoir$STA) <- paste("Station", levels(ecrinsoir$STA),
  sep = "")
levels(ecrinsoir$STA)
[1] "Station1" "Station2" "Station3" "Station4" "Station5" "Station6"
[7] "Station7" "Station8" "Station9" "Station10" "Station11" "Station12"
[13] "Station13" "Station14"

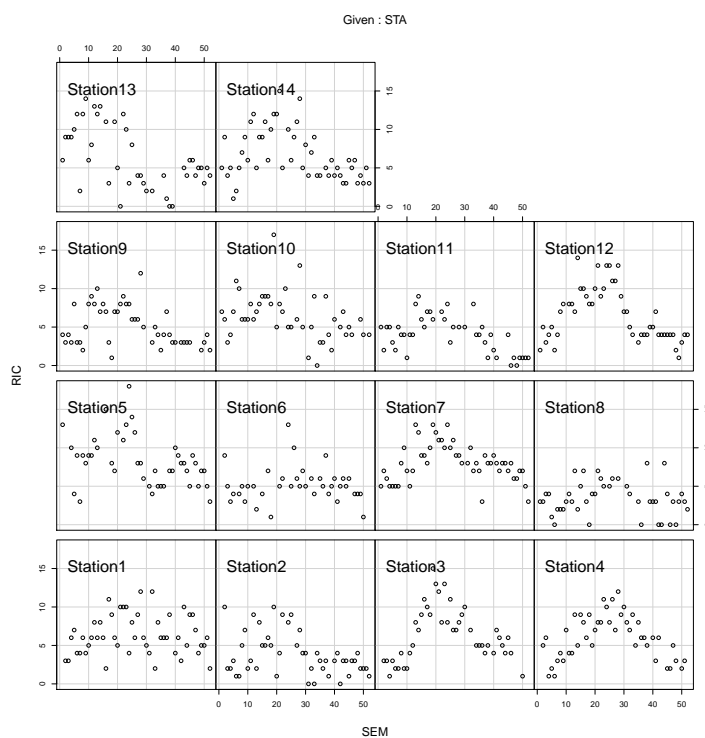
coplot(RIC ~ SEM | STA, data = ecrinsoir)

coplot(RIC ~ SEM | STA, data = ecrinsoir, show = F)
```



Etudier la fonction `coplot`. Implanter la fonction :

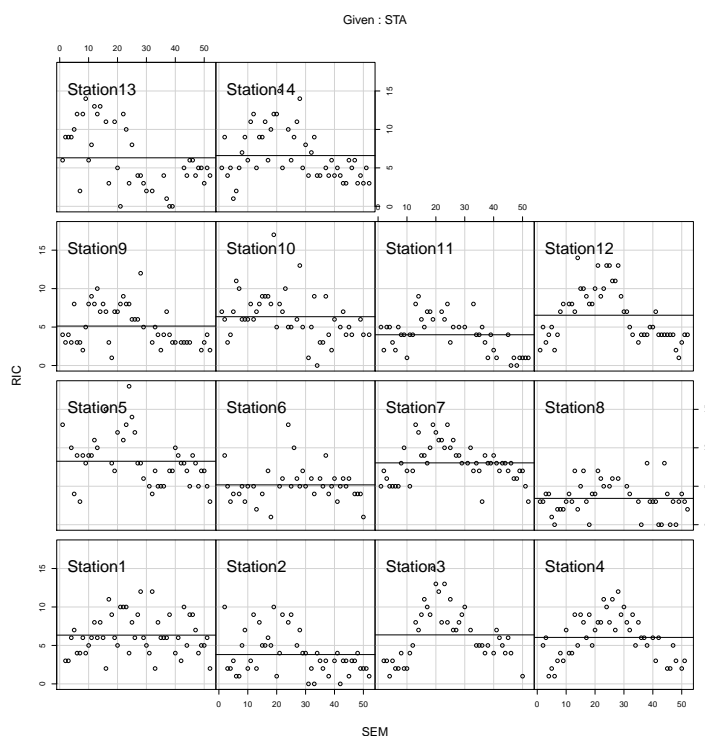
```
pan1 <- function(x, y, subscripts, col, pch) {
  points(x, y)
  text(1, 15, unique(ecrinsoir$STA[subscripts]), cex = 2, pos = 4)
}
coplot(RIC ~ SEM | STA, pan = pan1, show = F, sub = T, data = ecrinsoir)
```



Travailler directement depuis un éditeur (par exemple Blocnotes) en laissant ouvert une fenêtre d'édition. Ecrire la fonction ci-dessous et sauvegarder celle-ci dans nom.R :

```
pan1 <- function(x, y, subscripts, col, pch) {
  points(x, y)
  abline(h = mean(y))
  text(1, 15, unique(ecrinsoir$STA[subscripts]), cex = 2, pos = 4)
}
coplot(RIC ~ SEM | STA, pan = pan1, show = F, sub = T, data = ecrinsoir)
```

Puis, dans le menu *File*, choisir *SourceRCode* et sélectionner nom.R.



On peut changer totalement de stratégie.

```
ecrinsoir.list <- split(ecrinsoir, ecrinsoir$STA)
lapply(ecrinsoir.list, function(x) mean(x$RIC))
```

On écrit une nouvelle fonction

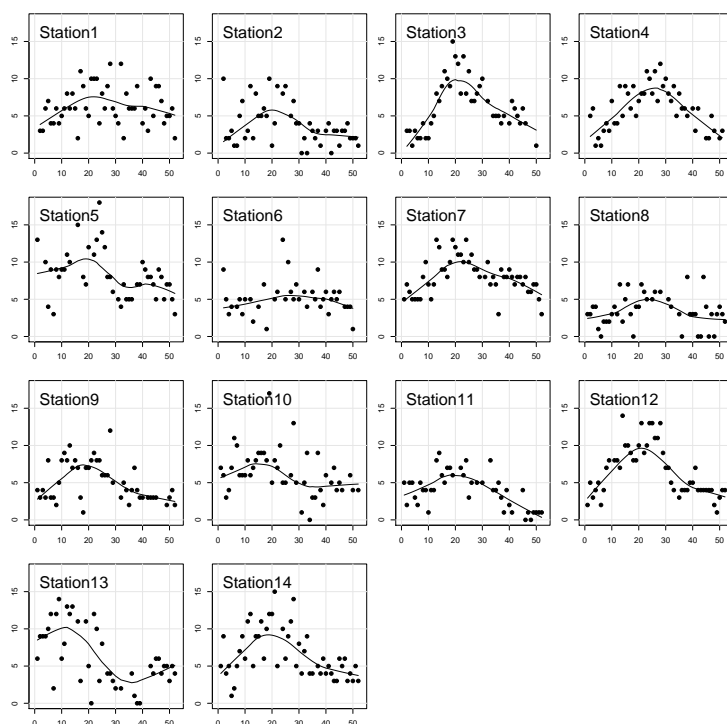
```
fun1 <- function(x) {
  plot(x$SEM, x$RIC, ylim = c(0, 18), xlim = c(0, 53), xlab = "",
       ylab = "", type = "n")
  grid(col = grey(0.9), lty = 1)
  points(x$SEM, x$RIC, pch = 20, cex = 1.5)
  text(0, 16, unique(x$STA), cex = 2, pos = 4)
}
par(mfrow = c(4, 4))
par(mar = c(3, 2, 1, 1))
lapply(ecrinsoir.list, fun1)
```

N'oubliez pas que les paramètres graphiques sont accessibles par l'une des deux expressions ci-dessous :

```
?par
help(par)
```

Faire varier les paramètres utilisés tout en consultant la documentation. Ajouter au bon endroit la ligne :

```
lines (lowess (x$SEM,x$RIC,f=0.5))
```



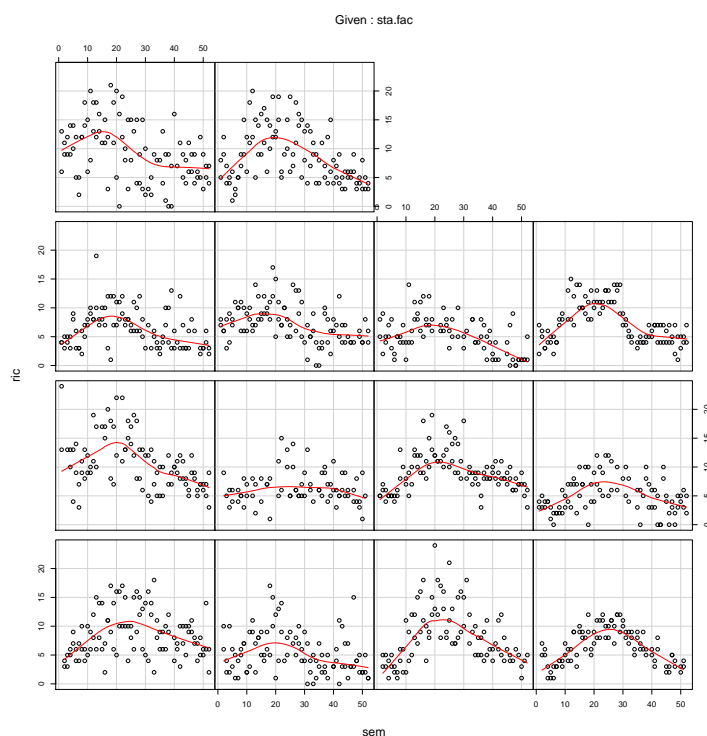
Merci Monsieur Cleveland !

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.

Pour aller plus vite :

```
coplot(ric ~ sem | sta.fac, show = F, panel = function(x, y, ...) panel.smooth(x,
y, span = 0.5, ...))
```



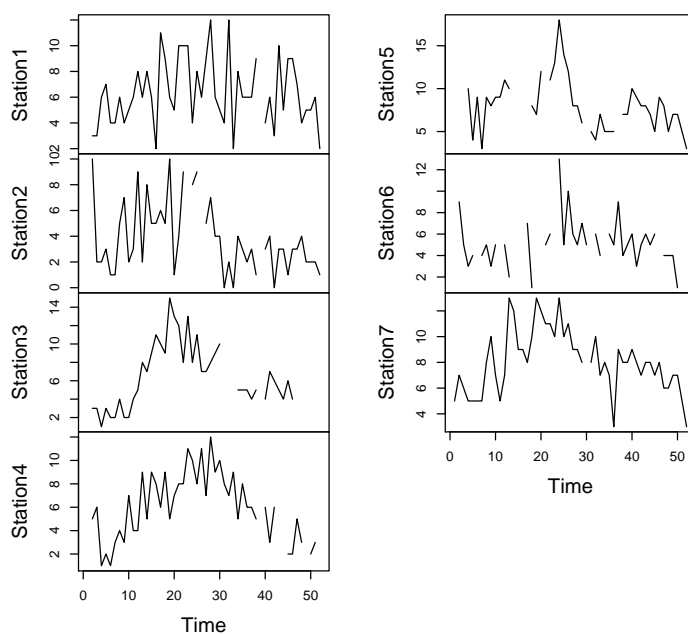
Ces premiers exemples peuvent vous inviter à consulter un ouvrage fabuleux :

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

Changer encore de stratégie :

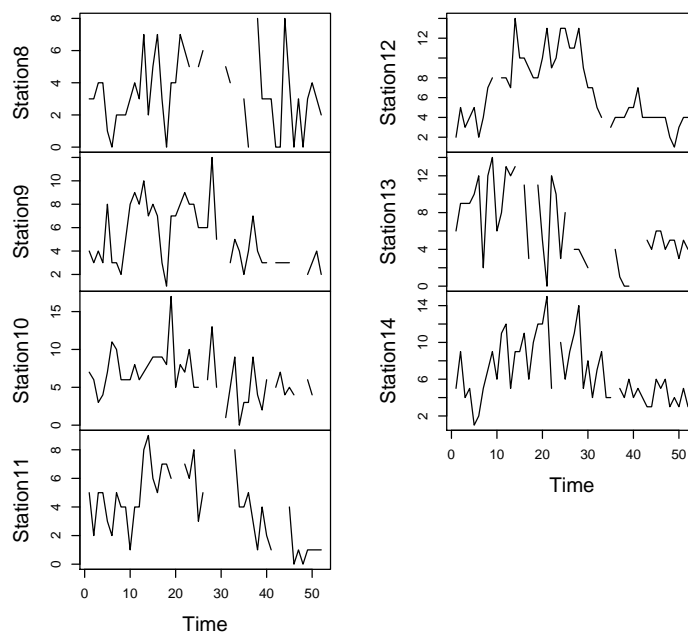
```
fun2 <- function(x) {
  w <- rep(NA, 52)
  w[x$SEM] <- x$RIC
  w
}
ecrinsoir.df <- as.data.frame(lapply(ecrinsoir.list, fun2))
ecrinsoir.df[1:4, ]
  Station1 Station2 Station3 Station4 Station5 Station6 Station7 Station8 Station9
1         NA         NA         NA         NA         13         NA         5         3         4
2         3         10         3         5         NA         9         7         3         3
3         3         2         3         6         NA         5         6         4         4
4         6         2         1         1         10         3         5         4         3
  Station10 Station11 Station12 Station13 Station14
1         7         5         2         6         5
2         6         2         5         9         9
3         3         5         3         9         4
4         4         5         4         9         5
plot(as.ts(as.matrix(ecrinsoir.df[, 1:7])))
```

```
as.ts(as.matrix(ecrinsoir.df[, 1:7]))
```

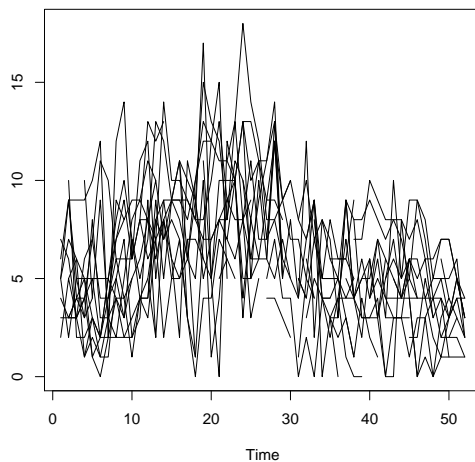


```
plot(as.ts(as.matrix(ecrinsoir.df[, 8:14])))
```

```
as.ts(as.matrix(ecrinsoir.df[, 8:14]))
```



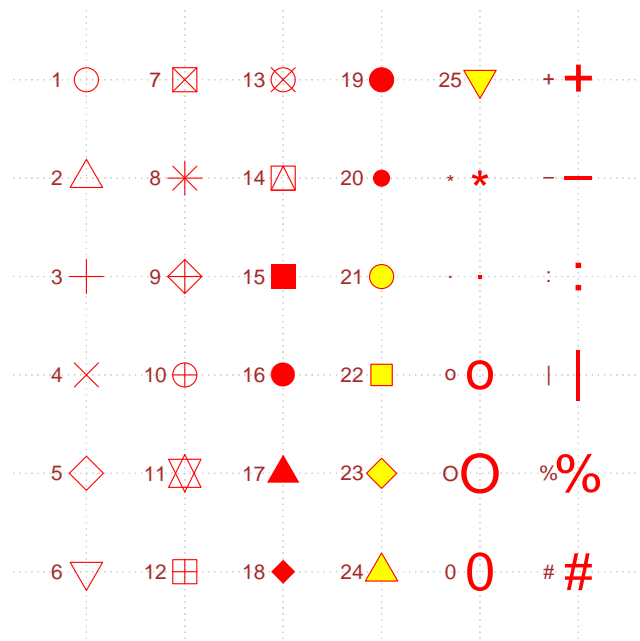
```
ts.plot(as.ts(as.matrix(ecrinsoir.df)))
```



Ah non ! Ici, ça ne vaut rien.

3 Annexe : Paramètres graphiques

3.1 Liste des points disponibles pour les graphiques

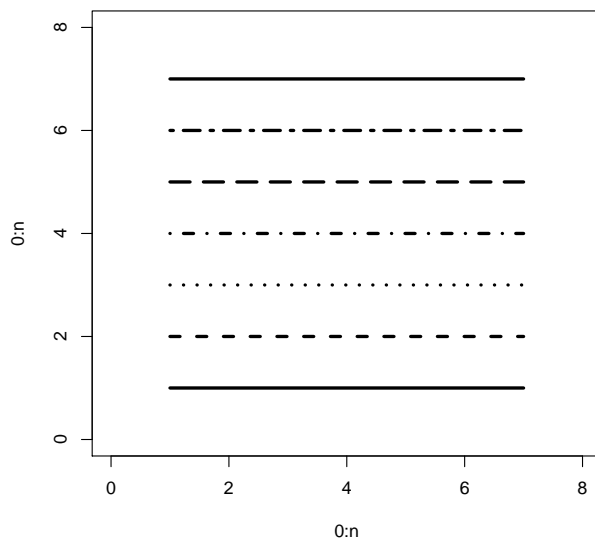


Le type de points est contrôlé avec le paramètre `pch`, par exemple pour avoir des cercles vides (l'option par défaut) on utiliserait `pch = 1`. La couleur externe est contrôlée avec le paramètre `col`, par exemple on a utilisé ici `col = "red"`. La couleur de remplissage, quand elle est disponible, est contrôlée par le paramètre

bg, par exemple on a utilisé `bg = "yellow"`. Pour plus de détails, consulter la documentation de la fonction `points()`

3.2 Liste des traits disponibles pour les graphiques

```
n <- 7
plot(0:n, 0:n, type = "n", xlim = c(0, n + 1), ylim = c(0, n + 1))
for (i in 1:n) {
  lines(1:n, rep(i, n), lwd = 3, lty = i)
}
```



Le type de traits est contrôlé avec le paramètre `lty`. Pour avoir, par exemple, un trait continu (l'option par défaut), on utiliserait `lty = 1`. Noter que l'épaisseur d'un trait, quelle que soit sa forme, s'obtient à l'aide de `lwd`.

Pour aller plus loin dans l'apprentissage des paramètres graphiques, lire le chapitre `Graphical procedures` de `An Introduction to R` que vous trouvez à partir du menu `Help, Manuals`.