

Mini-guide S-PLUS

Miguel Chagnon, Normand Ranger et Didier Garriguet
Laboratoire de statistique
Département de mathématiques et statistique
Université de Montréal

Juin 1999
Version 4.1

Contents

1	Avant-propos	1
1.0.1	Références	1
2	Introduction	3
2.1	Utilisation des gestionnaires d'écran	3
2.2	Disponibilité et appel de S-PLUS	3
2.3	Preliminaires	4
2.4	Syntaxe	4
3	Objets S-PLUS	5
3.1	Introduction	5
3.2	Assignation	5
3.3	Manipulations	6
3.4	Vecteur	6
3.5	Matrice et tableau	8
3.6	Data frame	10
3.7	Liste	10
3.8	Fonction	11
3.8.1	Principes fondamentaux	11
3.8.2	Écrire ses propres fonctions	11
4	Quelques fonctions élémentaires	15
4.1	Fonctions de manipulation	15
4.2	Fonctions de transformations numériques	15
4.3	Fonctions d'aide	16
4.4	Fonctions graphiques	16
4.5	Fonctions statistiques	17
4.6	Fonctions utiles à la programmation de ses propres fonctions	19
4.7	Autres fonctions	20
5	Exemple d'une session S-PLUS	23
5.1	Organisation d'une session S-PLUS	23
5.2	Exemple	24

6	Librairies maisons	29
A	Librairie maison Stat.S	I
B	Librairie maison pour l'analyse de variance	III
C	Librairie maison pour le bootstrap	V
D	Librairie maison Chrono pour les séries chronologiques	VII
E	Librairie maison domouSe	XIII
F	Librairie maison Gee pour solutionner des équations généralisées	XV
G	Librairie maison pour la régression	XVII
H	Librairie maison Venables	XIX
I	Librairie maison tests sur les tests statistiques	XXI
J	Librairie maison wavetresh sur la théorie des ondelettes	XXIII

Chapter 1

Avant-propos

Ce petit guide n'a pas pour but de présenter tout le logiciel S-PLUS. Spécialement conçu pour des étudiants suivant des cours où l'apprentissage de S-PLUS est nécessaire, ce document ne présente qu'un nombre restreint de commandes et d'instructions. De plus, une certaine connaissance de SunOS (ou IRIX) et UNIX pourra aider les utilisateurs de S-PLUS. Le logiciel S-PLUS est une version améliorée du logiciel S de *AT&T Bell Laboratories*, commercialisée par *Statistical Sciences Inc. (StatSci)*.

1.0.1 Références

Les concepteurs de S publient un manuel de référence sur S ¹. Un utilisateur *régulier* de S-PLUS a intérêt à consulter ce livre.

Pour la version S-PLUS, plusieurs manuels du fabricant sont disponibles ².

De plus, il existe un manuel avec une approche *statistique* du logiciel:

Chambers J.M., Hastie T.J., Statistical Models in S, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, 1992.

¹Becker R.A., Chambers J.M., Wilks A.R., The New S Language, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, 1988.

²S-PLUS User's Manual, Vol. 1-2, Version 3.2;

S-PLUS Reference Manual, Version 3.2;

S-PLUS Programmer's Manual, Version 3.2;

S-PLUS User's Manual Supplement, Version 3.2.

Chapter 2

Introduction

2.1 Utilisation des gestionnaires d'écran

À moins d'avis contraire, nous suggérons d'utiliser le gestionnaire d'écran **X11** ou **OpenWindows**¹ pour l'utilisation de S-PLUS quelle que soit la procédure d'appel présentée à la section suivante.

2.2 Disponibilité et appel de S-PLUS

Environnements informatiques

Le logiciel S-PLUS est disponible à l'Université de Montréal sur (au moins) 3 réseaux:

- réseau ESI avec la version S-PLUS 3.4 ;
- réseau DMS (serveur *cramer*) avec la version S-PLUS 3.4 et 5.03 ;
- réseau JSP avec la version S-PLUS 3.4 .

Notons que ces 3 réseaux sont indépendants. Il est donc possible de posséder un "compte" sur chaque réseau.

Appel de S-PLUS

Pour appeler S-PLUS faire la commande suivante:

Splus

¹Ces gestionnaires d'écran s'appelle avec les commandes **x** (note: lettre minuscule!), **startx** ou **openwin**. Sur un terminal X, l'appel au gestionnaire d'écran est habituellement automatique.

Consultez le responsable de l'installation.

2.3 Préliminaires

Notons que S-PLUS a besoin d'un sous-répertoire de travail nommé obligatoirement **.Data** . Si l'utilisateur n'a pas de sous-répertoire **.Data** dans le répertoire où il se trouve, S-PLUS en créera un à moins qu'il en existe un dans le répertoire principal.

S-PLUS est un logiciel orienté sur le concept de manipulation d'objets et de fonctions. Il utilise son propre répertoire (**.Data**) qui contient tous les fichiers (objets, fonctions, etc.) créés durant une session interactive. De plus, ce répertoire contient d'autres fichiers créés par S-PLUS dont:

- **.Audit** : journal de toutes les commandes antérieures, utile pour la fonction **history**
- **.Random.seed** : utilisé lors de génération de valeurs aléatoires
- **last.dump**, **.Last.fixed**, **.Last.value** (objets utilitaires...)

Le symbole d'incitation à donner une instruction (*prompt symbol*) est le caractère *plus grand* (>). Enfin, on termine une session S-PLUS avec la fonction **q()** .

2.4 Syntaxe

- une fonction s'utilise toujours avec une paire de parenthèses (pouvant ne rien contenir)
- une fonction utilisée sans parenthèses donnera la définition (en langage S-PLUS) de celle-ci.
- une commande UNIX peut-être exécutée durant une session S-PLUS en la précédant d'un point d'exclamation. Par contre, il faudra être prudent dans l'utilisation de certaines commandes UNIX.
- tout texte à la droite du caractère # est un commentaire. Ceci est particulièrement utile lorsqu'on écrit ses propres fonctions.

Exemples:

```
# Ceci est une ligne de commentaire
mean(x[-1])          # moyenne sur x sauf l'element 1
```

- il est permis de faire un Return durant l'énoncé d'une instruction; S-PLUS produira automatiquement le caractère + au début de la ligne suivante si la syntaxe de ce qui a déjà été tapé ne peut pas être considéré comme une instruction S-PLUS complète.
- S-PLUS différencie les minuscules des majuscules.

Chapter 3

Objets S-PLUS

3.1 Introduction

- Un nom d'objet est formé de caractères alphanumériques et du point. Il doit débiter par une lettre.
- S-PLUS manipule des objets qui peuvent être, entre autres, des vecteurs, matrices, tableaux, catégories, séries (chronologiques), listes et fonctions. S-PLUS offre aussi un objet de type *data frame*.
- un objet peut posséder des attributs:
 - un mode: logique, numérique, complexe et caractère;
 - une longueur: souvent le nombre d'éléments dans l'objet;
 - des noms associés aux éléments dans l'objet.

3.2 Assignment

La forme générale de l'assignation est:

objet *symbole_d'assignation* *expression*

- *objet* : peut être tout objet; si le reste de l'assignation est absente, on aura le contenu de l'objet
- *symbole_d'assignation* : peut être le caractère `_` (souligné) ou la paire de caractères `<-`
- *expression* : peut être un objet, une partie d'objet ou une transformation arithmétique, logique de plusieurs objets.

Remarque:

Si un nom d'objet créé (par l'assignation) est un nom déjà existant dans S-PLUS, le logiciel utilisera toujours, par défaut, l'objet de l'utilisateur. Les fonctions **assign** et **get** permettent de manipuler des objets ayant des noms identiques

(dans le répertoire de l'utilisateur et dans S-PLUS). Malgré l'existence des deux fonctions nommées précédemment, on suggère d'essayer de trouver des noms "originaux" à ses objets.

3.3 Manipulations

On peut manipuler les objets de plusieurs façons:

- liste du contenu (les éléments) de l'objet
- transformation arithmétique dont :
 - les 4 opérations traditionnelles (+, -, *, /), ^ (exponentiation), %/% (division entière), %% (modulo), %*% (produit matriciel), etc.
- opération logique dont :
 - >, <, >=, <=, == (égal), != (différent), & (et), | (ou), ! (négation), etc.
- extraction d'une partie d'objet; ceci se fera par une manipulation logique ou par la mention d'un indice
- fonction (pré-définie ou de l'utilisateur) sur l'objet

3.4 Vecteur

L'objet de type vecteur est très important dans S-PLUS. Un objet d'un seul élément sera considéré comme un vecteur de longueur 1.

Exemples de manipulations de vecteurs

1. `data1 = c(12.1,8.4,0.1,3)` : la fonction `c()` crée un vecteur dont la longueur est le nombre d'éléments donné
2. `data2 = c(6.02,2.8,0.025,.6)`
3. `data1 / 2`
`[1] 6.05 4.20 0.05 1.5`
 Remarques:
 - 1) Les 2 objets n'ayant pas la même longueur, l'objet 2 est *répété* pour compléter l'opération.
 - 2) Le symbole `[1]` indique que le résultat est un vecteur et sert à numéroter les éléments lorsque ceux-ci n'ont pas de noms (voir la fonction `names`).
4. `data1 + c(2,4)`
`[1] 14.1 12.4 2.1 7.0`
5. `resul = data1 / 2` : création de l'objet `resul` qui sera un vecteur

6. `data1 < 1.0`
[1] F F T F : manipulation logique dont le résultat est un vecteur d'éléments logiques (mode logique); F signifie False et T True
7. `data2[3]`
[1] 0.025 : élément 3 du vecteur data2 qui devient un vecteur de longueur 1
8. `data1[2:4]`
[1] 8.4 0.1 3.0 : les éléments 2 à 4 du vecteur data1 qui deviennent un vecteur de longueur 3
9. `data1[-1]`
[1] 8.4 0.1 3.0 : les éléments de data1 sauf l'élément 1
10. `data1[data1 > 1]`
[1] 12.1 8.4 3 : les éléments de data1 qui *appartiennent* à la condition [...]; note : ceci est différent de `data1 > 1`
11. `data2[data1 > 1]`
[1] 6.02 2.8 .6 : les éléments de data2 qui sont *vrais* sous la condition faite sur data1
12. `(1:33)`
[1] 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] 26 27 28 29 30 31 32 33 : générer une séquence de nombres; forme abrégée de la fonction **seq**
13. `(1:3)[data1 < 1]`
[1] 3 : le(s) numéro(s) de séquence parmi les 3 premiers éléments soumis à la condition
14. `(1:4)[data1 < 1 & data2 < 1]`
[1] 3 : le(s) numéro(s) de séquence des éléments de valeur True du vecteur logique créé par la condition
15. `mean(data1)`
[1] 5.9 : appel de la fonction pré-définie mean (moyenne arithmétique)
16. `length(data1)`
[1] 4 : appel de la fonction pré-définie length (taille du vecteur)
17. `data1.moy - mean(data1)`
`(1:length(data1))[data1 > data1.moy]`
[1] 1 2 : le(s) numéro(s) de séquence des éléments de data1 supérieurs à leur moyenne
18. `names(data1) - c("Christian","Normand","Robert","Roch")` : associe des noms à chaque élément du vecteur. Ainsi:

```
data1
  Christian Normand Robert Roch
      12.1      8.4      0.1      3
```

3.5 Matrice et tableau

L'objet matrice est évidemment semblable à la notion bien connue en mathématique. On peut associer des noms aux lignes et colonnes de cet objet. De plus, il est facile de manipuler des sous-matrices d'une matrice. Enfin, l'objet matrice fait référence uniquement à des tableaux à 2 dimensions; c'est l'objet tableau (**array**) qui représente les tableaux à plus de 2 dimensions. Dans ce document, on ne traite que de l'objet matrice. Les exemples qui suivent présentent des matrices dont les entrées sont des nombres mais S-PLUS peut manipuler des matrices de caractères ou d'éléments logiques.

Exemples de manipulations de matrices

1. `mat1 = matrix((1:15),nrow=3,ncol=5,byrow=T)` : création d'un objet matrice à l'aide de la fonction **matrix**. Ainsi:

```
mat1
  [,1] [,2] [,3] [,4] [,5]
[1,]   1   2   3   4   5
[2,]   6   7   8   9  10
[3,]  11  12  13  14  15
```

2. `mat1[2,1]`
`[1] 6` : élément `mat121`
3. L'utilisation d'une ligne ou colonne d'une matrice est possible de plusieurs façons:
 - (a) `mat1[,2]` : vecteur contenant la deuxième colonne de la matrice `mat1`
 - (b) `mat1[,2,drop=F]` : une matrice 3×1 contenant la deuxième colonne de la matrice `mat1`
 - (c) `mat1[1,drop=F]` : une matrice 1×3 contenant la première ligne de la matrice `mat1`
4. `mat1[,2] - 2`
`[1] 0 5 10` : cette soustraction produit ce vecteur
5. `mat2 = mat1[,-2]` : produit une nouvelle matrice 3×4 ayant ôté la deuxième colonne de la matrice `mat1`
6. `t(mat1)` : appel de la fonction pré-définie **t** (transposée)

7. `mat1 %*% t(mat1)` : produit matriciel de la matrice `mat1` par sa transposée
8. `dimnames(mat1) _ list(c("r1","r2","r3"),c("c1","c2","c3","c4","c5"))` : fonction pré-définie **dimnames** qui associe aux lignes et aux colonnes de la matrice des noms; voir à la section 3.6 pour la notion de liste (**list**). De plus, lors de transformations arithmétiques de cette matrice *identifiée* les noms associés seront conservés si possible. Ainsi:

```
mat1
  c1 c2 c3 c4 c5
r1  1  2  3  4  5
r2  6  7  8  9 10
r3 11 12 13 14 15
t(mat1)
  r1 r2 r3
c1  1  6 11
c2  2  7 12
c3  3  8 13
c4  4  9 14
c5  5 10 15
```

9. `mat1["r3","c2"]`
 [1] 12 : élément de la matrice retracé par ses noms aux indices de ligne et colonne
10. `ncol(mat1)`
 [1] 5 : fonction pré-définie indiquant le nombre de colonnes de la matrice
11. `nrow(mat1)`
 [1] 3 : fonction pré-définie indiquant le nombre de lignes (rangées) de la matrice
12. `dim(mat1)`
 [1] 3 5 : fonction pré-définie produisant un vecteur de longueur 2 contenant le nombre de lignes et de colonnes. Ainsi:
nrow(mat1) est équivalent à **dim(mat1)[1]** et **ncol(mat1)** est équivalent à **dim(mat1)[2]**
13. `mat3 _ rbind(mat1,c(-2,-1,0,1,2))` : création d'une nouvelle matrice formée de la matrice `mat1` et d'une rangée (**row**) supplémentaire indiquée par le vecteur nommé
14. `mat4 _ cbind(mat1,rep(1,row(mat1)))` : création d'une nouvelle matrice formée de la matrice `mat1` et d'une colonne (**column**) supplémentaire de 1

3.6 Data frame

S-PLUS offre un autre type d'objet: *data frame*. Celui-ci est en fait semblable au type matrice. Dans le cas d'une matrice toutes les valeurs sont du même type tandis que le cas du *data frame*, on peut mettre dans chaque colonne des types différents. Dans l'exemple suivant:

```
nouvo _ data.frame(data1,data2,data1<1)
l'objet nouvo contiendra:
```

	data1	data2	data1...1
Christian	12.1	6.020	FALSE
Normand	8.4	2.800	FALSE
Robert	0.1	0.025	TRUE
Roch	3.0	0.600	FALSE

dont les colonnes sont de type numérique et logique. De plus, S-PLUS a attribué automatiquement des noms aux lignes et aux colonnes de ce *data frame* en utilisant, pour identifier les lignes, les noms d'un des vecteurs (*data1*) utilisés pour la création de cet objet et, pour identifier les colonnes, les noms des arguments. Pour plus de détails, voir le chapitre 5 de *S-PLUS 5 User's Guide*.

3.7 Liste

Un objet de mode liste est un ensemble pouvant contenir plusieurs éléments (composantes) qui peuvent être de différents modes. Ce concept est très versatile et permet des manipulations d'objets pouvant devenir très sophistiqués.

Exemples de manipulations de listes

1. `liste1 _ list(data1,mat1,c(1.1,2.2,3.3),c("Allo","Bonjour"))` : création de l'objet `liste1` qui contient 4 composantes (2 objets précédemment créés (un vecteur et une matrice), un vecteur numérique et un vecteur caractère)
2. `liste1[[3]]`
`[1] 1.1 2.2 3.3` : composante 3 de la liste `liste1`
3. `liste[[4]][2]`
`[1] "Bonjour"` : deuxième élément de la composante 4 (qui est un vecteur) de la liste
4. `liste1 _ list(vect1=data1,mat=mat1,vect2=c(1.1,2.2,3.3),nom=c("Allo","Bonjour"))`
: similaire à l'exemple 1 mais en associant des noms à chacune des composantes de la liste
5. `liste1$vect2` : similaire à l'exemple 2 en supposant l'association des noms définis à l'exemple 4

6. `t(liste1[[2]])` ou `t(liste$mat)` : transposée de la matrice `mat1`; toute composante d'une liste est un objet utilisable dans toute manipulation d'objet
7. `liste2 _ list(listeune=liste1,logique=c(T,T,F,F,T))` : création d'une liste contenant 2 composantes dont la première est une liste et la seconde un vecteur logique; ainsi, `liste2[[1]][[3]][1]` ou `liste2$listeune$vect2[1]` sera l'élément 1 de la composante 3 de la composante 1 de la liste `liste2` i.e. 1.1

3.8 Fonction

3.8.1 Principes fondamentaux

S-PLUS est principalement basé sur l'utilisation de fonctions qui sont des objets très puissants. Il existe 2 types de fonctions : celles définies par l'utilisateur et celles pré-définies (dans S-PLUS). La syntaxe d'une fonction S-PLUS est :

nom(paramètres)

- une fonction peut n'avoir aucun paramètre
- s'il y a plus d'un paramètre, ceux-ci doivent être séparés par des virgules
- (le résultat d'une fonction ne vaut qu'une valeur mais celle-ci peut être complexe (ex.: une liste)
- les valeurs des paramètres de la fonction peuvent être transmises par position ou par mot-clé; il y a très souvent une valeur par défaut pour les paramètres

Exemple :

la fonction **rnorm** qui génère des nombres d'une loi $N(\text{mean}, \text{sd}^2)$ est définie:

`rnorm(n,mean=0,sd=1)`

- *n* nombre de valeurs à générer
- *mean* moyenne; valeur par défaut: 0
- *sd* écart-type; valeur par défaut: 1

Ainsi :

`rnorm(5)` : génère 5 nombres d'une $N(0,1)$

`rnorm(5,sd=4)` : génère 5 nombres d'une $N(0,16)$

`rnorm(mean=2,n=4)` : génère 4 nombres d'une $N(2,1)$

3.8.2 Écrire ses propres fonctions

Pour écrire ses propres fonctions S-PLUS, l'utilisateur doit connaître:

- un éditeur
- certaines fonctions S-PLUS utiles en programmation (voir section 4.6)

Éditeur

Il y a 2 éditeurs de texte possibles dans Splus: l'éditeur **vi** et **emacs**.

Par défaut, l'éditeur de texte est vi. Pour changer votre éditeur emacs, vous devez ajouter la ligne suivante à votre fichier .cshrc:

```
setenv S_CLEditor emacs
```

Par la suite, lorsque vous voulez entrer dans S-Plus, faites la commande:

Splus -e

et ce que vous voulez travailler avec l'éditeur vi ou emacs. La commande Splus tout court n'ouvre aucun éditeur de texte (seulement la fenêtre de travail).

Voici certaines commandes que vous pouvez utiliser suite à ces changements:

COMMAND	emacs	vi
Reculer d'un espace	Ctrl-B	Esc, h
Avancer d'un espace	Ctrl-F	Esc, l
Ligne précédente	Ctrl-P	Esc, k
Ligne suivante	Ctrl-N	Esc, j
Dbut de la ligne	Ctrl-A	Esc, ^ (Shift-6)
Fin de la ligne	Ctrl-E	Esc, \$ (Shift-4)
Avancer d'un mot	Esc, f	Esc, w
Reculer d'un mot	Esc, b	Esc, b
Couper un caractère	Ctrl-D	Esc, x
Couper une ligne	Ctrl-K	Esc, Shift-d
Effacer un mot	Esc, d	Esc, dw
Recherche arrière	Ctrl-R	Esc, ?
Coller	Ctrl-Y	Esc, Shift-y
Inverser 2 caractères	Ctrl-T	Esc, xp

Pour éditer une fonction, nous conseillons d'utiliser un éditeur de votre choix, sauvegarder votre fonction par exemple sous le nom "fonction.splus", et ensuite de faire, dans S-PLUS:

```
source("fonction.splus")
```

Règles de construction d'une fonction (Exemple avec vi)

1. Une fonction S-PLUS comporte: un nom, une paire de parenthèses pouvant contenir des arguments, une accolade gauche, des lignes d'instructions et se termine par une accolade droite.

2. Si la fonction contient des arguments, on associe à chacun d'eux un mot-clé (nom) et possiblement une valeur par défaut.

Exemples:

La fonction suivante ne contient aucun argument:

```
function()
{
    print("Bonjour")
    print("Normand")
}
```

La fonction suivante contient deux arguments qui n'ont pas de valeurs par défaut:

```
function(mot1, mot2)
{
    print(mot1)
    print(mot2)
}
```

La fonction suivante contient deux arguments qui possèdent des valeurs par défaut:

```
function(mot1 = "Bonjour", mot2 = "Normand")
{
    print(mot1)
    print(mot2)
}
```

Cette dernière fonction est la plus intéressante puisqu'elle permet de passer n'importe quelles valeurs aux deux arguments sans que ce soit obligatoire.

3. Si on crée une fonction avec un nom qui existe déjà dans l'ensemble des fonctions S-PLUS, le logiciel utilisera toujours, par défaut, la fonction de l'utilisateur. Les fonctions **assign** et **get** permettent d'employer des noms identiques (dans le répertoire de l'utilisateur et dans S-PLUS). Malgré l'existence des deux fonctions nommées précédemment, on suggère d'essayer de trouver des noms "originaux" à ses objets.

Chapter 4

Quelques fonctions élémentaires

4.1 Fonctions de manipulation

rep(x,times,length) : répéter l'objet **x** pour **times** fois pour une longueur de **length** valeurs

Exemples : (avec `data1` défini en 3.3.2)

```
rep(data1,2)
```

```
[1] 12.1 8.4 0.1 3 12.1 8.4 0.1 3
```

```
rep(data1,length=7)
```

```
[1] 12.1 8.4 0.1 3 12.1 8.4 0.1
```

seq(from,to,by,length,along) : générer une séquence de nombres

Exemples :

```
seq(-1,4,length=9)
```

```
[1] -1.000 -0.375 0.250 0.875 1.500 2.125 2.750 3.375 4.000 : générer 9  
nombres entre -1 et 4 (intervalles égaux)
```

```
seq(1,10,1) est équivalent à (1:10)
```

sort(data) : trier en ordre croissant un vecteur **data**

rm(...) : détruire des objets (du répertoire `.Data`)

Exemples :

```
rm(data1)
```

```
rm(data1,data2,mat1)
```

4.2 Fonctions de transformations numériques

Il existe une multitude de fonctions transformant les objets dont :

- **sqrt**, **abs**, **exp**, **log**, **log10**, fonctions trigonométriques, etc.

- **ceiling**, **floor**, **trunc**, **round**, **signif** : arrondi, *tronquation*, nombre de chiffres significatifs, etc.
- **min(...)**, **max(...)** : valeur minimum ou maximum d'un vecteur numérique

4.3 Fonctions d'aide

ls() : lister les objets de l'utilisateur; certains paramètres peuvent être mentionnés dans des cas particuliers (voir le chapitre 5)

help(fonction) : documentation interactive sur la **fonction** nommée
 Note: `help(nom_de_la_fonction, T)` enverra à l'imprimante par défaut la documentation de cette fonction.

help.start() : fonction de S-PLUS qui génère une fenêtre où l'utilisateur peut chercher interactivement à l'aide de la souris, la documentation sur une fonction. La recherche se fait par thèmes (ou catégories). Particulièrement utile lorsqu'on ne connaît pas le nom de la fonction dont on cherche la documentation...

history(pattern=".",max=10) : relevé (à partir du fichier .Audit) des **max** (valeur par défaut:10) dernières instructions S-PLUS où la chaîne de caractères **pattern** est apparue; à la suite de ce relevé, il est possible d'en choisir une et de la réexécuter automatiquement. Si **max=1** ou si la fonction ne retrace qu'une instruction, celle-ci sera automatiquement réexécutée.

Exemples :

`history(dat,7)` : liste des 7 dernières commandes S où les caractères **dat** sont apparus

`history()` : liste des 10 dernières instructions

4.4 Fonctions graphiques

Parmi les fonctions graphiques, ce document présente les plus simples et les plus utiles. L'explication des paramètres y est très simplifiée. L'utilisateur devra consulter le manuel ou utiliser interactivement la fonction **help(nom_de_la_fonction)**. Pour qu'un graphique soit dirigé vers un support d'impression, une fonction d'impression (**X11()**, **motif()**, **postscript()**, etc.) doit être active. De plus, la fonction d'impression demeure active tant qu'une autre fonction d'impression n'a pas été appelée.

X11(), **motif()** : pour l'impression à l'écran des graphiques sous le gestionnaire d'écran X11.

La fenêtre produite comporte un menu permettant à tout moment de commander l'impression vers l'imprimante par défaut du graphique tel qu'il apparaît à l'écran. Mais pour une meilleure qualité d'impression,

on suggère d'utiliser la fonction **postscript()**. Le menu de cette fenêtre permet de changer la destination de l'imprimante par défaut de l'option *Printgraph*, de permettre l'impression en format 11 X 8 1/2 (*landscape*) plutôt que 8 1/2 X 11 (*portrait*) et de copier la fenêtre graphique actuelle pour permettre la comparaison de plusieurs graphiques.

postscript() : diriger les graphiques vers l'imprimante par défaut (sur le réseau DMS, l'imprimante par défaut est *labostat* au 4218).

dev.cur(), **dev.list()**, ... : dans S-PLUS, il existe plusieurs fonctions permettant de travailler simultanément dans plusieurs fenêtres graphiques et/ou imprimante. Il n'y a toujours qu'une fenêtre (imprimante) "courante" à la fois mais avec les fonctions de type **dev.---** il est possible de changer la fenêtre courante. Pour plus de détails, voir *help(dev.cur)*.
Pour le "néophyte" du graphisme dans S-PLUS, il est recommandé d'utiliser la fonction suivante:

graphics.off() : permet de s'assurer que tous les *devices* (destinations, support d'impression) graphiques sont "fermés", s'assurant ainsi le début de l'impression (particulièrement dans le cas de la fonction **postscript()**). Il faut noter qu'un graphique n'est physiquement imprimé que lorsque S-PLUS est certain qu'il n'y a plus rien d'autres à ajouter à ce graphique. La fonction exige que l'utilisateur réexécute une nouvelle fonction d'impression.

identify(2_vecteurs_ou_une_liste, labels=vecteur_des_noms_des_éléments)
: permettre lors d'une session avec graphiques à l'écran d'identifier des points sur un graphique à l'aide de la souris

title(main, sub, xlab, ylab, axes=F) : permet d'ajouter des titres à un graphique

hist(x, nclass, breaks, plot=T, angle, density, col, inside) : trace un histogramme des valeurs du vecteur **x**

plot(x, y, type="p", log="") : produit un nuage de points entre les vecteurs **x** et **y**

4.5 Fonctions statistiques

Ce document présente quelques fonctions statistiques élémentaires. Pour une documentation plus complète, faire **help(nom_de_la_fonction)** ou consulter le manuel de référence.

cor(x, y, trim=0) corrélation ou matrice de corrélations (si **x** ou **y** sont des matrices).

mean(x, trim=0) : moyenne (tronquée ou non) des éléments d'un vecteur

median(x) : médiane des éléments d'un vecteur

prod(...) : produit des éléments d'un vecteur

quantile(x, probs=seq(0,1,.25)) : quantile d'une distribution d'éléments d'un vecteur. Par défaut, le résultat est un vecteur dont les éléments sont: la valeur minimum, les quartiles et la valeur maximum.

sample(x, size, replace=F) : générer un échantillon aléatoire

set.seed(i) : permet de fixer le germe avant une génération de nombres aléatoires. Utile pour régénérer un même ensemble de valeurs.

stem(x, nl, scale, twodig=1, fence, head=T, depth=F) : produit un histogramme de type "*stem-and-leaf*". Même si le résultat est un histogramme, ce n'est pas une fonction graphique car l'historgramme n'apparaît pas dans la fenêtre graphique.

sum(...) : somme des éléments d'un vecteur

var(x, y) : variance des éléments d'un vecteur ou matrice de covariances si les arguments sont des matrices. L'argument **y** n'est pas nécessaire pour la variance d'un vecteur **x**.

_beta, _cauchy, _chisq, _exp, _f, _gamma, _logis, _lnorm, _norm, _t, _unif

: noms de code relatifs aux distributions suivantes:

Beta, Cauchy, χ^2 , Exponentielle(1), *F* de Fisher, Gamma, Logistique, Log-normal, Normale, *T* de Student, Uniforme.

De plus, il existe 6 autres noms de code possibles:

_binom, _geom, _hyper, _nbinom, _pois, _weibull]

qui représentent les distributions suivantes:

Binomiale, Géométrique, Hypergéométrique, Binomiale négative, Poisson et Weibull.

Ces noms de code doivent être précédés d'un des 4 préfixes suivants:

d : fonction de densité

p : fonction de répartition

q : quantiles

r : génération de nombres aléatoires

Chaque fonction possède ses propres arguments.

Exemples:

1. Valeur de la fonction de densité d'une Beta(4,5) au point 0.1:
`dbeta(.1,4,5)`
`[1] 0.183708`
2. Valeurs de la fonction de répartition d'une Cauchy(0,20) aux trois points nommés:
`pcauchy(c(2,4,6),0,20)`
`[1] 0.5317255 0.5628330 0.5927736`

3. Déciles d'une χ_5^2 :
`qchisq(seq(.1,.9,.1),5)`
`[1] 1.610308 2.342534 2.999908 3.655500 4.351461 5.131868 6.064431`
`7.289279`
`[9] 9.236354`
4. Génération de 5 valeurs aléatoires d'une $\text{Exp}(1)$:
`rexp(5)`
`[1] 1.2752478 2.3055170 1.6694114 0.1327236 0.6108058`

4.6 Fonctions utiles à la programmation de ses propres fonctions

Un usager qui doit écrire une fonction, peut utiliser toutes les fonctions du logiciel (et ses propres fonctions...) pour construire celle-ci. En plus des fonctions mentionnées dans les autres sections, mentionnons quelques autres fonctions. Note: pour plus d'informations, voir le manuel de référence ou faire `help(nom_de_la_fonction)`. Pour les fonctions `if` et `for`, il faut faire: `help("if")` et `help("for")` ...

`apply(X, MARGIN, FUN, ...)` : permet d'appliquer aux lignes ou colonnes (argument `MARGIN`) d'une matrice (argument `X`), la même fonction (argument `FUN`).

Exemples:

`apply(mat,2,mean)`

produit un vecteur de moyennes des colonnes de la matrice

`t(apply(mat,1,sort))`

produit une matrice dont chaque ligne est triée en ordre croissant. L'utilisation de la fonction `t` (transposée) est nécessaire pour produire une nouvelle matrice aux mêmes dimensions que l'originale.

`for` : semblable à l'instruction `do` ou `for` dans d'autres langages de programmation.

La syntaxe générale est:

```
for(variable in objet) {
... instructions ...
}
```

Si le bloc d'instructions ne contient qu'une instruction, la paire d'accolades est facultative.

Exemple:

```
for(i in (1:norigine)) {
  matplot(resul[[i]][, 1], resul[[i]][, 2:4], type =
    "l", lty = c(2, 1, 2), pch = "p", col = 1)
  title(main = paste("Origine: temps", origine[i]), cex = 0.7)
  lines(resul[[i]][, 1], resul[[i]][, 5], type = "o", pch = "r")
}
```

if : semblable à l'instruction `if` dans d'autres langages de programmation. La syntaxe générale est:

```
if(expression_logique) {
  ... instructions ...
}
else {
  ... instructions ...
}
```

Dans l'*expression_logique*, on utilise les opérateurs logiques décrits à la section 3.3 . Si le bloc d'instructions ne contient qu'une instruction, la paire d'accolades est facultative.

Exemple:

```
if(lin[i] == 0) {
  nncol <- iy
}
else {
  nncol <- lin[i]
}
```

stop(message) : imprime un **message** et arrête l'exécution de la fonction.

warning(message) : imprime un **message** mais l'exécution de la fonction se poursuit.

4.7 Autres fonctions

print(x, digits, quote=TRUE) : permet l'impression (à l'écran...) de l'objet `x`

lpr(x, command = "lpr", width = 80, length = 66, wait = F, rm.file = T, ...)
: permet d'envoyer à l'imprimante par défaut le contenu d'un objet

attach(file=NULL,pos=2) : permet d'attacher d'autres répertoires de travail (**.Data**)

sink("nom_de_fichier_UNIX") : permet de diriger les résultats dans un fichier du répertoire de l'utilisateur (pas **.Data**) plutôt qu'à l'écran; **sink()** ramène les résultats à l'écran

scan(file="",what=numeric(),n,sep,multi.line=F,flush=F,append=F)
: permet de copier un fichier UNIX d'un usager dans le répertoire **.Data** en le transformant en objet **S-PLUS**. Il y a, entre autres, 2 façons d'utiliser cette fonction. Si on suppose que le fichier UNIX de données est composé de *nl* lignes et de *nc* colonnes de nombres séparés par au moins un blanc chacun, on peut créer un objet de type vecteur, matrice ou liste. Exemples:

1. Création d'un vecteur de n éléments où $n = nl \times nc$:
`resul = scan(file="nom_du_fichier_UNIX")`
2. Création d'une matrice:
`resul = matrix(scan(file="nom_du_fichier_UNIX"),nrow=nl,ncol=nc,byrow=T)`
Cet exemple crée la matrice `resul` de dimension nl par nc . On pourra évidemment compléter la création de cette matrice avec la fonction `dimnames` (voir section 3.5).
3. Création d'une liste à 2 éléments (`var1` et `var2`):
`resul = scan(file="nom_du_fichier_UNIX",list(var1=0,var2=0))`

unix(command, input, output=TRUE) : permet d'exécuter une commande UNIX comme une fonction S-PLUS.

Exemples:

unix("ls .Data") est équivalente à `ls()`

unix("wc -l toto") : permet de calculer le nombre de lignes du fichier UNIX `toto`

Chapter 5

Exemple d'une session S-PLUS

Pour ce chapitre, nous nous sommes inspirés du document suivant:

Une introduction à S-Plus
Marcel Baumgartner
juillet 1994

Une copie de ce document est disponible au local 4249 du pavillon mathématiques et informatique.

5.1 Organisation d'une session S-PLUS

Les commentaires suivants concernent l'organisation d'une session S-PLUS avec un environnement UNIX. On suppose que le lecteur possède quelques connaissances de base de UNIX.

Suivez les points ci-dessous si vous allez utiliser S-PLUS pour la première fois:

- Créer tout d'abord, au niveau de votre répertoire principale, un nouveau répertoire; par exemple intro:

```
pivele% mkdir intro
```

- Puis allez dans ce répertoire et créer un nouveau sous-répertoire .Data, qui va contenir tous les objets S-PLUS que vous allez définir:

```
pivele% cd intro  
pivele% mkdir .Data           (obligatoirement .Data)
```

- Restez dans le répertoire intro et **n'allez pas** dans le sous-répertoire .Data. Démarrez S-PLUS avec la commande suivante:

```
pivele% Splus
```

Vous devriez voir 4 lignes similaires aux suivantes qui indiquent que S-PLUS est prêt.

```
-----
S-PLUS : Copyright (c) 1988, 1996 MathSoft, Inc.
S : Copyright AT&T.
Version 3.4 Release 1 for Sun SPARC, SunOS 5.3 : 1996
Working data will be in .Data
>
```

- Le logiciel attend maintenant vos commandes. Pour quitter le programme:

```
> q()
```

Si vous utilisez à nouveau S-PLUS dans ce répertoire, vous n'avez pas besoin de recréer le répertoire .Data. Remarquez que tous les objets que vous définirez lors d'une session avec S-PLUS, seront conservés dans ce répertoire .Data.

Si vous n'avez pas de répertoire .Data dans le répertoire à partir du quel vous démarrez S-PLUS, le logiciel va en créer un sous votre répertoire principale.

Les deux fonctions suivantes sont aussi importantes en ce qui concerne l'organisation du travail dans S-PLUS: `ls()` et `objects()`. La commande `ls()` affiche toutes les variables du répertoire .Data *courant*. La commande `objects()` est une version plus sophistiquée de `ls()`. Elle permet entre autres de n'afficher qu'une partie des objets.

5.2 Exemple

- On appelle le logiciel S-PLUS

```
ceres 17%
S-PLUS : Copyright (c) 1988, 1996 MathSoft, Inc.
S : Copyright AT&T.
Version 3.4 Release 1 for Sun SPARC, SunOS 5.3 : 1996
Working data will be in .Data
>
```

- On définit un vecteur x de composantes (2,5,6,9,3,10,12,18)

```
> x <- c(2,5,6,9,3,10,12,18)
```

- On affiche le contenu de l'objet x

```
> x  
[1] 2 5 6 9 3 10 12 18
```

- On affiche la longueur du vecteur x

```
> length(x)  
[1] 8
```

- On affecte la moyenne de x dans la variable $moy.x$

```
> moy.x <- mean(x)
```

- on affiche la variable $moy.x$

```
> moy.x  
[1] 8.125
```

- On calcule la variance du vecteur x

```
> var(x)  
[1] 27.83929
```

- On définit un vecteur y de composante 1 à 8 et on affiche le contenu de y

```
> y <- c(1:8)  
> y  
[1] 1 2 3 4 5 6 7 8
```

- On fait la somme des vecteurs x et y

```
> x+y  
[1] 3 7 9 13 8 16 19 26
```

- On construit une matrice $matX$ avec colonnes x et y et on affiche le contenu de $matX$

```
> matX <- cbind(x,y)  
> matX  
      [,1] [,2]  
[1,]    2    1  
[2,]    5    2
```

```
[3,] 6 3
[4,] 9 4
[5,] 3 5
[6,] 10 6
[7,] 12 7
[8,] 18 8
```

- On ouvre une fenêtre graphique

```
> X11()
```

- On fait le graphique de y en fonction de x

```
> plot(x,y,type="b")
```

- On lit le fichier de données /JSP/C2/usagers/chagnonm/intro/olympique et on affecte le résultat dans l'objet olymp et on affiche le contenu de l'objet olymp

```
> olymp<-matrix(scan("/JSP/C2/usagers/chagnonm/intro/olympique"),ncol=3,byrow=T)
> olymp
      [,1] [,2] [,3]
[1,] 100  9.9  0
[2,] 200 19.8  0
[3,] 400 43.8  0
[4,] 800 103.0  0
[5,] 1500 212.5  0
[6,] 5000 785.6  0
[7,] 10000 1658.4  0
[8,] 42195 7761.0  0
[9,] 100  11.0  1
[10,] 200  21.8  1
[11,] 400  48.8  1
[12,] 800 113.5  1
[13,] 1500 236.6  1
[14,] 3000 516.0  1
[15,] 42195 8692.0  1
```

- on affecte la première colonne de olymp dans l'objet olymp1 et on affiche olymp1

```
> olymp1 <- olymp[,1]
> olymp1
[1] 100 200 400 800 1500 5000 10000 42195 100 200 400 800
[13] 1500 3000 42195
```

- On liste les objets du répertoire .Data

```
> ls()
[1] "matX"  "moy.x"  "olymp"  "olymp1" "x"      "y"
```

- On efface les objets matX et olymp

```
> rm(matX,olymp)
> ls()
[1] "moy.x"  "olymp1" "x"      "y"
```

- On quitte S-PLUS

```
> q()
ceres 21%
```


Chapter 6

Librairies maisons

Malgré l'existence de nombreuses fonctions dans S-PLUS, le laboratoire de statistique du DMS a créé un petit ensemble de nouvelles fonctions pour utiliser S-PLUS dans différents contextes. Les fonctions sont réparties dans des librairies maisons selon leur utilité. Présentement, 13 librairies maisons sont disponibles:

Stat.S : librairie de fonctions générales. Voir l'appendice A;

Anova : voir l'appendice B;

Bootstrap : Librairie des fonctions Splus et des jeux de données tirées du livre "An Introduction to the Bootstrap" par B. Efron and R. Tibshirani, 1993, Chapman and Hall. Voir l'appendice C

Chrono : librairie de jeux de données et de fonctions relatives aux séries chronologiques développées par Normand Ranger. Pour plus d'informations, voir l'appendice D.

Chrono3 : importante librairie pour l'estimation des systèmes dynamiques (séries chronologiques). Plus d'informations dans les menus d'aide de Splus une fois la librairie attachée.

DomouSe¹ : librairie de fonctions pour la détection de données multivariées aberrantes. Voir l'appendice E.

gee : librairie contenant des fonctions permettant de solutionner des équations généralisées. Voir l'appendice F.

Regression : voir l'appendice G;

Rice : Jeux de données provenant du livre "MATHEMATICAL STATISTICS AND DATA ANALYSIS", 2nd Edition, par John A. Rice.

Rousseeuw.Leroy : Jeux de données provenant du livre de Rousseeuw et Leroy, "Robust regression and outlier detection", Wiley, 1987.

¹domouSe est l'abréviation de : `detection of multivariate outliers using S environment...`

Appendix A

Librairie maison Stat.S

Cette librairie contient des fonctions générales. Actuellement cette librairie contient (entre autres) les fonctions suivantes:

accent : fonction permettant d'imprimer des lettres accentuées dans les graphiques;

S+

accent.fin.fonte.1335 fonction qui édite un fichier UNIX PostScript qui contient un graphique avec des caractères accentués et des caractères des fontes 13 (mathématique, grecque) ou 35 (ZapfDingbats);

S+

accent.postscript : fonction postscript modifiée pour permettre l'impression des caractères accentués dans les graphiques;

S+

dyn.load2 : chargement dynamique d'un fichier objet. Il en existe une autre version (légèrement différente) dans S-PLUS;

ichar fonction qui retourne la valeur numérique d'un caractère dans la table ASCII;

S+

mixed.mtext, **mixed.mtext.vector**, **mixed.text**, **mixed.text.vector** fonctions permettant d'imprimer dans un graphique des caractères avec des indices, exposants et en combinant différentes fontes, orientations, tailles, etc.;

S+

public : fonction identique à **postscript** sauf qu'elle imprime par défaut au U-211 du DIRO;

text.optim : fonction similaire à la fonction originale **text** mais optimisée pour limiter le nombre de points superposés dans un graphique de type "nuage de points";

S+

xgobi : fonction dynamique graphique d'analyse de données.

Des descriptions détaillées des fonctions sont disponibles en appelant la fonction **help**(*nom_de_la_fonction*).

Appendix B

Librairie maison pour l'analyse de variance

... À documenter ...

IV APPENDIX B. LIBRAIRIE MAISON POUR L'ANALYSE DE VARIANCE

Appendix C

Librairie maison pour le bootstrap

Cette librairie contient des fonctions relatives au bootstrap. Elles sont tirées du livre “An Introduction to the Bootstrap” par B. Efron and R. Tibshirani, 1993, Chapman and Hall. Dans le répertoire `\home\stat\lib\Lib.splus\Bootstrap` on retrouve également les jeux de données et les erreurs contenues dans ce livre. Actuellement, cette librairie contient les fonctions suivantes:

abcnon : fonction permettant de calculer des intervalles de confiance ABC non paramétriques.

abcpar : fonction similaire à **abcnon**, mais pour le bootstrap paramétrique.

bcanon : fonction similaire à **abcnon**, mais pour les intervalles BCa non paramétriques.

bootpred : fonction permettant de calculer l'estimation bootstrap de l'erreur de prédiction.

bootstrap : fonction permettant d'estimer un estimé bootstrap non paramétrique.

boott : fonction similaire à **abcnon** mais pour l'intervalle de confiance bootstrap.

crossval : fonction permettant d'effectuer la validation croisée de K-fold.

Appendix D

Librairie maison Chrono pour les séries chronologiques

Cette librairie contient des jeux de données et des fonctions relatives aux séries chronologiques. En voici quelques unes.

Fonction graphique

Cette fonction permet d'imprimer le graphique d'une variable (série). La syntaxe de cette fonction est:

```
graphique(fichier, nocol, nom, titre, s1, d1, s2, d2, lambda, ecran,  
impression)
```

fichier nom du fichier de données ou de résultats contenant les valeurs à mettre en graphique. Ce nom doit être entre guillemets. De plus, si le fichier n'est pas un fichier dans le répertoire où l'utilisateur est en train de travailler, on devra fournir le nom au complet (avec tous les sous-répertoires car la forme *~/sous-répertoires* n'est pas acceptée) . Aucune valeur par défaut.

nocol numéro de colonne de la variable considérée, i.e. quelle colonne de valeurs du fichier. Si le fichier de données ne contient qu'une seule série (variable) et que plusieurs valeurs sont sur une même ligne, on affectera la valeur 0 à cet argument. Aucune valeur par défaut.

nom nom de la variable à l'impression du graphique. Ce nom doit être entre guillemets. Valeur par défaut: du fichier **fichier** . De plus, si des arguments **s1, d1, s2, d2** ou **lambda** sont non nuls, l'impression du nom sera modifiée en conséquence.

titre titre à l'impression du graphique. Ce titre doit être entre guillemets.
Valeur par défaut:

Graphique de la serie **nom**

s1 d1 s2 d2 lambda arguments permettant de transformer la série originale (disons X) ainsi:

$$(1 - B^{s1})^{d1}(1 - B^{s2})^{d2}X^{lambda}$$

Si **lambda** vaut 0, la transformation logarithmique sera effectuée. Les valeurs par défaut de ces arguments ne produisent aucune transformation de la série originale.

ecran clé logique valant **T** (true) ou **F** (false) indiquant si on désire le graphique à l'écran (**T**) ou à une imprimante (**F**). Valeur par défaut: **T**.

impression argument dont la valeur dépend de l'argument **ecran**.

Si **ecran** est **T** (true), **impression** indique sous quel type de gestionnaire d'écran se déroule la session de S. Dans la plupart des situations, la valeur par défaut **X11()** (i.e. en utilisant **X**) sera la plus appropriée. Rappelons que sous **X** à l'appel de cette fonction, il faudra cliquer la patte gauche de la souris pour créer la fenêtre graphique.

Si **ecran** est **F** (false), **impression** indique le nom de l'imprimante où le graphique sera imprimé. La valeur la plus probable serait: **labostat** (4218). Le nom de l'imprimante doit être entre guillemets. Aucune valeur par défaut.

Fonction imprime

Cette fonction permet de diriger à une imprimante le contenu d'un objet S-PLUS. La syntaxe de cette fonction est:

imprime(objet, impression)

objet indique l'objet dont le contenu sera dirigé à une imprimante. Tout objet (même le source d'une fonction) sauf un graphique peut être imprimé. Cet objet sera très souvent l'appel d'une fonction (voir l'exemple 7.). Aucune valeur par défaut.

impression indique le nom de l'imprimante où l'objet sera imprimé.

Les valeurs les plus probables seraient: **labostat** (au 4218) ou **dms4191** (au 4426-1). Le nom de l'imprimante doit être entre guillemets. Aucune valeur par défaut.

Fonction nuage

Cette fonction permet d'imprimer un nuage de points de la variable (série) désirée en fonction de cette même variable avec un délai k désiré. De plus, ce graphique indique le coefficient de corrélation et la droite de régression des moindres carrés de ce nuage de points. La syntaxe de cette fonction est:

nuage(fichier, nocol, nom, titre, delai, ekran, impression)

fichier nocol nom ecran impression arguments identiques à la fonction **graphique**.

titre argument identique à celui de la fonction **graphique** sauf que la valeur par défaut est:

Nuage de points $X(t+k)$ vs $X(t)$ de la serie **nom** avec $k = ,delai$

delai indique la valeur du délai de la variable considérée.

Fonction prevision

Cette fonction permet de calculer les prévisions et les limites de confiance pour la série originale. La syntaxe de cette fonction est:

prevision(fichier, lambda, const, alpha, graph, ecran, impression)

fichier nom du fichier de résultats produit par la macro locale *prevas* du logiciel SCA (voir description de cette macro à l'appendice B). Cette fonction ne peut accepter aucun autre fichier que celui produit par la macro mentionnée. Ce nom de fichier doit être entre guillemets. Aucune valeur par défaut.

lambda valeur de la transformation λ de Box-Cox. Une valeur 0 indique la transformation logarithmique. Valeur par défaut: aucune transformation.

const constante à ajouter, s'il y a lieu, aux valeurs des séries de prévisions. Valeur par défaut: 0.

alpha niveau de signification pour les limites de confiance. Valeur par défaut: 0.05.

graph clé logique valant T (true) ou F (false) indiquant si on désire le graphique des prévisions et des limites de confiance. Avec la valeur F (false), on obtient un tableau des résultats. Valeur par défaut: F .

ecran impression arguments identiques à la fonction **graphique**.

Fonction tboxcox

Cette fonction permet d'imprimer un graphique du logarithme de la fonction de vraisemblance par rapport au paramètre λ de la transformation de Box-Cox pour différentes valeurs de λ . La syntaxe de cette fonction est:

tboxcox(fichier, nocol, nom, titre, s1, d1, s2, d2, ecran, impression, graph, intervalle, alpha)

fichier nocol nom s1 d1 s2 d2 ecran impression arguments identiques à la fonction **graphique**.

titre argument identique à celui de la fonction **graphique** sauf que la valeur par défaut est:

Logarithme de la fonction de vraisemblance (transformation Box-Cox)
de la serie **nom** (alpha = 0.05)

XAPPENDIX D. LIBRAIRIE MAISON CHRONO POUR LES SÉRIES CHRONOLOGIQUES

graph clé logique valant T (true) ou F (false) indiquant si on désire le graphique du logarithme de vraisemblance des sommes des carrés des écarts SCE (T) ou une liste des valeurs graphiques (F). Valeur par défaut: T .

intervalle vecteur de 3 valeurs numériques indiquant l'ensemble (intervalle) de valeurs de λ . Ce vecteur doit être obligatoirement sous la forme suivante: $c(\text{inf}, \text{sup}, \text{saut})$ où:

- *inf* valeur inférieure de λ ;
- *sup* valeur supérieure de λ ;
- *saut* valeur du saut (*incrément*) dans l'intervalle mentionné. Les valeurs de λ seront: $\text{inf}, \text{inf}+\text{saut}, \text{inf}+2*\text{saut}, \dots, \text{sup}$

Valeur par défaut: **c(-2,2,.1)** .

alpha valeur pour la région de confiance à $(1 - \alpha)$ pour λ . Valeur par défaut: **0.05**.

Exemples

1. graphique("toto.dat",3,"PNB")
2. graphique("/home/stat/test/ex.res",1,"Var. 3",ecran=F,impression="labostat")
Voir remarque 2.
3. graphique("dat.tp1",1,"Cout", "Graphe du Cout",F,"labostat")
4. nuage("toto.bis.dat",2,"Masse monetaire",delai=1,ecran=F,impression="labostat")
Voir remarque 2.
5. tboxcox("w3.dat",0,"W3",s1=1,d1=1,s2=12,d2=1)
6. prevision("prev.g",0,graph=T,alpha=0.01)
7. imprime(tboxcox("w3.dat",0,"W3",graph=F),"labostat")

Remarques:

1. Il est à remarquer que dans ces exemples, les guillemets sont notés par le caractère " .
2. Dans un appel d'une fonction S, tous les arguments non mentionnés prennent la valeur par défaut. Il n'est donc pas nécessaire de les mentionner si cette valeur par défaut est désirée. Mais si on ne désire que les valeurs par défaut de certains arguments, il faut indiquer le nom des arguments avec leurs valeurs pour tous les autres arguments de la fonction qui suivent ceux non mentionnés.

Aide

Durant une session de S, il est possible d'avoir accès à la documentation relative aux fonctions maison décrites ci-dessus. Il suffit de taper la fonction suivante:

help(*nom_de_la_fonction*)

XII APPENDIX D. LIBRAIRIE MAISON CHRONO POUR LES SÉRIES CHRONOLOGIQUES

Appendix E

Librairie maison domouSe

Cette librairie contient des fonctions relatives à la détection de données multivariées aberrantes. Actuellement cette librairie contient les fonctions suivantes:

dom.dyn.load2 : fonction identique à `dyn.load2` (voir appendice A);

eigen2 : fonction qui calcule les valeurs et vecteurs propres d'une matrice réelle (pas nécessairement symétrique);

fquad : approximation de la fonction de distribution $F(x) = P(Q \leq x)$ où Q est une combinaison linéaire de variables χ^2 ;

gen.comb : fonction générant la “prochaine” (au sens lexicographique) combinaison (permutation sans ordre) d'une combinaison donnée;

infl.A : fonction utilitaire dans le calcul de la fonction **influence**;

infl.mult : fonction d'influence de tous les sous-ensembles possibles d'une matrice de données ou d'un sous-ensemble fixé de données;

influence : fonction d'influence d'une matrice de données selon différents coefficients de corrélation vectorielle;

robuste : estimation robuste d'une matrice de covariance et d'un vecteur moyenne;

rohlf : Test de Rohlf (*Generalised Gap Test*) pour la détection de valeurs aberrantes;

rv : calcul d'un coefficient de corrélation vectorielle;

trace.mat : trace d'une matrice;

varinfl : fonction utilitaire calculant la variance de la fonction d'influence;

wilks : test de Wilks et distance de Rohlf pour la détection de valeurs aberrantes;

Des descriptions détaillées des fonctions sont disponibles en appelant la fonction **help**(*nom.de.la.fonction*).

Appendix F

Librairie maison Gee pour solutionner des équations généralisées

... À documenter ...

Appendix G

Librairie maison pour la régression

Cette librairie contient des fonctions relatives à la régression linéaire multiple. Actuellement cette librairie contient les fonctions suivantes:

addplot : graphique d'une variable ajoutée

avas : additivité et stabilisation de la variance (pas encore opérationnelle...)

bootreg : bootstrap de la régression linéaire multiple

boxcox : graphique de la vraisemblance logarithmique pour la transformation paramétrique dans le modèle de Box et Cox

influence : graphiques sur des diagnostics d'une régression

linreg : fonction de régression linéaire multiple

qr.r : décomposition QR d'une matrice

resplot : graphique des résidus

seqtable : table ANOVA séquentielle d'une régression linéaire multiple

tables : tables d'une régression

Des descriptions détaillées des fonctions sont disponibles à l'aide de la fonction **help**(*nom_de_la_fonction*) .

Appendix H

Librairie maison Venables

... À documenter ...

Appendix I

Librairie maison tests sur les tests statistiques

... À documenter ...

XXII APPENDIX I. LIBRAIRIE MAISON TESTS SUR LES TESTS STATISTIQUES

Appendix J

Librairie maison wavetresh sur la théorie des ondelettes

... À documenter ...