Guide sur l'utilisation du logiciel S-PLUS

Normand Ranger
CIRANO
Centre Interuniversitaire de Recherche en ANalyse des Organisations
2020 rue University, 25e étage
Montréal, Qué. H3A 2A5

Guide # 7 - Version 1.0 - 18 novembre 1996

Ce document est disponible à l'adresse suivante:
http://www.cirano.umontreal.ca/publication/guides/page1.html

Table des matières

Avant-propos

Ce petit guide n'a pas pour but de présenter tout le logiciel (langage?) S-PLUS. Ce document ne présente qu'un nombre restreint de commandes et d'instructions. De plus, une certaine connaissance de UNIX pourra aider les utilisateurs de S-PLUS.

Historique

La version originale de S-PLUS s'appelait S et avait été produite par *AT&T Bell Laboratories*. Par la suite, une version *améliorée* fut commercialisée sous le nom S-PLUS par la compagnie *Statistical Sciences Inc. (StatSci)* qui fut achetée par le fournisseur actuel *MathSoft*.

Manuels de référence, librairies de programmes publiques

Le fabricant de S-PLUS fournit avec le logiciel une série de livres de référence: *S-PLUS User's Manual, S-PLUS Reference Manual, S-PLUS Gentle introduction, S-PLUS Crash Course, S-PLUS Programmer's Manual, S-PLUS Guide to Statistics and Mathematical Analysis, Treillis Graphics User's Manual*, etc. De plus, plusieurs auteurs ont écrit des livres sur S-PLUS ou sur un domaine scientifique utilisant S-PLUS (ex.: en statistique, graphisme). Mentionnons en particulier (cette liste provient du site WWW de S-PLUS et date du 27 février 1996):

```
"An Introduction to S and S-Plus"
by Phil Spector,
Duxbury Press, Belmont, CA, 1994

"Statistical Models In S"
by John Chambers and Trevor Hastie
Chapman and Hall, 1992

"A Handbook of Statistical Analyses using S-PLUS"
by Brian Everitt,
Chapman & Hall, London, UK, 1994.

"Modern Applied Statistics with S-PLUS"
by William N. Venables and Brian D. Ripley,
Springer, New York, NY, 1994

"Data analysis by using S"
by M. Sibuya and R. Shibata

"Graphics"
by Richard A. Becker, John M. Chambers and Allan R. Wilks

"Smoothing Techniques with Implementation in S"
by W. Hardle,
Springer, 1991

"Algorithms, Routines and S Functions for Robust Statistics"
by A. Marazzi,
Wadsworth & Brooks/Cole, Pacific Grove, CA, 1992.
```

```
"Visualizing Data"
by William S. Cleveland,
Hobart Press, Summit, NJ, 1993.
```

Notons que le site WWW du logiciel est: `http://www.mathsoft.com/splus.html` .
Il existe aussi un site WWW (`http://lib.stat.cmu.edu/S`) qui offre des libraires de programmes S-PLUS produites par des usagers du logiciel. Sur ce même site, certains autres guides d'usagers sont disponibles dont:

```
"Une introduction à S-Plus"
Marcel Baumgartner (french.intro)

"Frequently Asked Questions about
S/S-PLUS"
by S. Gomatam and B. Narasimhan (faq)

"Cheatsheet: Simple Summaries of S Syntax and Expressions"
by Barry Brown (cheatsheet)

"Introductory Guide to S-Plus"
by Brian Ripley (sguide.ps1, sguide.ps2)

"Notes on S-PLUS"
by William Venables and Dave Smith (splusnotes)

"Notes on S-PLUS (student edition)"
by William Venables and Dave Smith
```

Environnement

Ce document traite de la version UNIX sous Solaris. Les versions sous d'autres environnements (dont *Windows*) présentent certaines différences, en particulier sous l'aspect graphique.

Remarque et remerciements

Ce guide est une nouvelle version d'un document produit antérieurement:
Mini-guide S et S-PLUS , Normand Ranger, Laboratoire de statistique-informatique, Dépt. d'informatique et de recherche opérationnelle, Université de Montréal, septembre 1993, version 3.0 .
L'auteur tient à remercier Christian Léger et Michel Lamoureux du département de Mathématiques et Statistique de l'Université de Montréal pour leurs commentaires lors de la révision du document mentionné ci-dessus. De plus, Christian Boudreau et Nancy Forget du même département m'ont permis de plagier quelques paragraphes de leur documentation sur S-PLUS

Améliorations

Enfin, toutes erreurs, remarques, commentaires et suggestions quant au contenu et/ou à la présentation de ce document seront recueillis avec joie, amour et tendresse par l'auteur...

Chapitre 1

Introduction

1.1 Version et commande d'appel

Version

La version actuelle de S-PLUS sur BERGERAC est 3.4 .

Appel de S

Il y a 2 façons d'appeler S-PLUS:

- Interactive avec la commande:
  Splus

- En lot (*batch*):
  Splus BATCH *fichier_input fichier_output*
  où:
  fichier_input contient les instructions S-PLUS à exécuter;
  fichier_output contient les résultats des instructions S-PLUS exécutées.

1.2 Préliminaires

Sous-répertoire de travail

Notons que S-PLUS a besoin d'un sous-répertoire de travail nommé obligatoirement **.Data** . Si l'usager 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.
On peut créer des sous-répertoires **.Data** dans n'importe quel répertoire.
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...)

Note: Sous Windows, le sous-répertoire de travail se nomme **_DATA** et se retrouve dans le répertoire défini par la variable d'environnement **HOME** .

1.3 Syntaxe

- une fonction s'utilise toujours avec une paire de parenthèses (pouvant ne rien contenir)
- une fonction utilisée sans parenthèses donnera le "source" de la fonction en S-PLUS. Malheureusement plusieurs fonctions ne font qu'appeler une fonction interne S-PLUS non visible...
- 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.

1.4 Remarques

1. Le symbole d'incitation à donner une instruction (*prompt symbol*) est le caractère *plus grand* (>).
2. Enfin, on termine une session S-PLUS avec la fonction **q()** .
3. Au CIRANO, une librairie locale de fonctions est automatiquement incluse lors de l'appel de S-PLUS. Cette librairie contient actuellement les fonctions suivantes:

```
accent                      accent.fin.fonte.1335
accent.postscript           accent.ps.setfont.std.latin1
accent.table                afm
attach.DSE                  fquad
ichar                       is.file
last.dump                   mixed.mtext
mixed.mtext.vector          mixed.text
mixed.text.vector           ps.fonts
stringwidth                 unix.true
```

Ces fonctions sont décrites à l'appendice A.

Chapitre 2

Objets S

2.1 Introduction

- Un nom d'objet est formé de caractères alphanumériques et du point. Il doit débuter 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.

2.2 Assignation

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'usager. Les fonctions **assign** et **get** permettent de manipuler des objets ayant des noms identiques (dans le répertoire de l'usager et dans S-PLUS). Malgré l'existence des deux fonctions nommées précédemment, on suggère d'essayer de trouver des noms "originaux" à ses objets. Dans certaines situations (mais pas toujours!...), S-PLUS indiquera cette duplication de noms par le message suivant:
Warning messages:
assigning `"nom_de_l'objet"` *masks an object of the same name on database ...*

2.3 Manipulations

On peut manipuler les objets de plusieurs façons:

- liste du contenu (les éléments) de l'objet
- transformation arithmétique dont:
  - les 4 opérations traditionnelles (+, -, *, /), ∧ (exponentation), %/% (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'usager) sur l'objet

2.4 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
```

2.5 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 suivant 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 $mat1_{21}$

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 2.7 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 (**r***ow*) 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 (**c***olumn*) supplémentaire de 1

2.6 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, consulter le *S-PLUS User's Manual*.

2.7 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

2.8 Fonction

2.8.1 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'usager 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). La valeur de la fonction est toujours la valeur représentée par la valeur de la dernière variable assignée dans la fonction. Exemple:

```
function(a,b)
{
c <- a + b
d <- a - b
}
```

Cette fonction retourne la valeur de la variable `d` .

- 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($mean$, 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)

2.8.2 Écrire ses propres fonctions

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

- un éditeur (voir section 3.8)
- certaines fonctions S-PLUS utiles en programmation (voir chapitre 3)

2.8.3 Règles de construction d'une fonction

L'exemple ci-dessous est présenté avec la fonction **fix** . Pour l'utilisation d'éditeurs, consultez la section 4.8 .

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. On suggère, lors de la création initiale d'une fonction, de créer la fonction sans aucune instruction et d'éditer ensuite cette fonction. Exemple:
   toto _ function(x, valeur = 4){}
   fix(toto)

3. L'éditeur se charge de présenter et d'*indenter* correctement les lignes de programmation de la fonction.

4. Si, en sortant de l'éditeur-fonction, il y a des erreurs de syntaxe, S-PLUS permet de retourner à la fonction modifiée avec la fonction **fix()** .
   Exemple:
   fix(toto)
   ...
   ... édition de la fonction puis sortie de l'éditeur

   Si un message d'erreur semblable à celui-ci apparaît:
   Syntax error: ... used illegally at line ..., file /tmp/fix...
   Dumped
   Errors occurred; Use fix() to re-edit this object.
   Il suffit de retourner dans la fonction en faisant: fix()

5. 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.

6. 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'usager. Les fonctions **assign** et **get** permettent d'employer des noms identiques (dans le répertoire de l'usager et dans S-PLUS). Malgré l'existence des deux fonctions nommées précédemment, on suggère d'essayer de trouver des noms "originaux" à ses objets. Au moment de la création initiale de la fonction personnelle, S-PLUS indiquera cette duplication par le message suivante:
   Warning messages:
   assigning `"nom_de_la_fonction"` *masks an object of the same name on database ...*

2.9 Objets de type *modèle*

Dans les nouvelles versions de S-PLUS, le concept "orienté-objet" apparaît avec la notion de "modèle". En fait, un des aspects intéressants est que les fonctions reconnaissent dans un objet le genre de fonction qui a créé celui-ci et produisent un résultat en conséquence. L'usager intéressé à ce concept de modèle manipulé par certaines fonctions aurait intérêt à consulter le livre *Statistical Models in S* de Chambers et Hastie...

Exemple

L'exemple ci-dessous utilise le modèle de régression linéaire associé à la fonction **lm** .

```
> bmdp[1:5,]
       Id Age V1  V2 V3  V4 V5  V6 V7
[1,] 2381  22 67 144  1 200 43  98 54
[2,] 1610  25 62 128  1 243 41 104 33
[3,] 1797  25 68 150  2  50 38  96 30
[4,]  561  19 64 125  1 158 41  99 47
[5,] 2519  19 67 130  2 255 45 105 83
> resul1.lm <- lm(V7 ~ V1 + V2, data = data.frame(bmdp))
> resul1.lm
```

```
Call:
lm(formula = V7 ~ V1 + V2, data = data.frame(bmdp))

Coefficients:
 (Intercept)         V1         V2
    35.71445 -0.1646386 0.1704361

Degrees of freedom: 181 total; 178 residual
Residual standard error: 10.81639
> par(mfrow=c(6,1))
> plot(resul1.lm)
```

Remarques

1. l'instruction `bmdp[1:5,]` permet de montrer que l'objet `bmdp` est une matrice de données N x 9. Seules les 5 premières lignes sont présentées ici.

2. La fonction **lm** permet de fournir un modèle à analyser en utilisant les noms des colonnes. Notons que l'objet à analyser doit être de type *data frame* d'où l'utilisation de la fonction **data.frame** dans le paramètre `data` .

3. L'instruction **resul1.lm** indique que l'objet contient toutes les références nécessaires et certains résultats pour se rappeler comment cet objet a été créé.

4. Après avoir redéfini la fenêtre graphique avec l'instruction **par**, l'exécution de l'instruction **plot** produira automatiquement (sans autres paramètres) 6 graphiques reliés à l'analyse de régression: graphique des résidus, des quantiles, de la distance de Cook, etc.

On remarque donc que l'utilisation de fonctions de type "modèle" permet de pouvoir manipuler et analyser des résultats dans un contexte bien défini (la rgression dans l'exemple ci-dessus)

Chapitre 3

Quelques fonctions élémentaires

3.1 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)

3.2 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

3.3 Fonctions d'aide

ls() lister les objets de l'usager; certains paramètres peuvent être mentionnés dans des cas particuliers (voir le chapitre 4)

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'usager 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(densite)` pourrait produire ceci:

```
1: lpr(densite)
2: fix(densite)
function(reper, beta, gamma, omega, borne)
{
        basef <- paste("/export/home2/ghyselse/ggs/", reper, "/", reper, sep =
                "");        nlgr <- 3;        parmfrow = c(nlgr, 2), oma = c(0, 0, 2, 0))
3: densite
4: densite
5: densite
6: densite("bt2", "a", "l", 1:6, 1.96)
7: densite("bt2", "a\",\"l\",1:6,1.96)\n\")\n\"\n)\n")
8: args(densite)
9: densite
Selection:
```

Il suffirait de choisir, s'il y a lieu un nombre entre 1 et 9 (il n'y avait que 9 occurences de la chaîne spécifiée dans cet exemple) pour que l'instruction soit exécutée à nouveau.
Note: cette fonction possède d'autres arguments.

3.4 Fonctions graphiques

3.4.1 Introduction

Parmi les fonctions graphiques, ce document présente les plus simples et les plus utiles. L'explication des paramètres y est très simplifiée. L'usager 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.

3.4.2 Fenêtres graphiques et impression

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'utliser la fonction **postscript()**. Les boutons de menu de cette fenêtre permettent de modifier l'impression, copier la fenêtre, etc.

postscript() diriger les graphiques vers l'imprimante par défaut NEUVILLETTE

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 le *S-PLUS User's Manual*.

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'usager réexécute une nouvelle fonction d'impression.

dev.off() "ferme" le plus récent *device* activé. La liste des *devices* forme une pile: dernier activé, premier désactivé.

3.4.3 Fonctions graphiques de base

Introduction

S-PLUS contient un ensemble de fonctions graphiques très intéressantes.
S-PLUS offre des fonctions graphiques qui "ajoutent" à un graphique (ex.: title, lines, abline, etc.) tandis que d'autres réinitialisent ("haut niveau") la fenêtre graphique (ex.: plot, hist, etc.).
Un usager intéressé à utiliser le graphisme dans S-PLUS aurait intérêt à consulter les multiples sections traitant de ce sujet dans les manuels de référence...

Fonctions de haut niveau

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

matplot superposition de plusieurs nuages de points à partir d'une matrice (chaque colonne de la matrice est un nuage de points)

tsplot graphique d'une série chronologique (**t***ime* **s***series*)

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

boxplot graphique de type *boxplot*

Fonctions "ajoutantes"

title permet d'ajouter des titres et sous-titres à un graphique

lines, matlines, tslines similaires à **plot, matplot** et **tsplot** mais ajoutent au graphique existant

points, matpoints, tspoints similaires aux fonctions précédentes mais ajoutent des points au graphique existant

abline permet d'ajouter, entre autres, des lignes verticales ou horizontales dans un graphique

legend permet d'ajouter une légende à un graphique

text permet d'ajouter du texte dans un graphique

mtext permet d'ajouter du texte dans les marges du graphique

axis permet de redéfinir les axes d'un graphique. Il faut noter que cette fonction s'utilise lorsqu'on trace un graphique sans la construction automatique des axes. Exemple:

```
plot(vect1, vect2, type = ''l'', axes = F) ; axis(2) ; box() ;
axis(1, seq(100, 150, length = 25), label = vect.nom)
```

3.4.4 Paramètres graphiques (fonction*par()*)

La fonction **par()** permet de modifier des dizaines de paramètres graphiques. Ces paramètres sont aussi diversifiés que: largeur des marges, fontes, type de lignes, subdivision de la fenêtre en plusieurs graphiques, etc. Un appel à cette fonction modifiera les paramètres graphiques par défaut pour tout graphique qui suivra. Seuls une fermeture d'un *device* ou un nouvel appel à la fonction **par** pourra modifier les paramètres affectés.
De plus, certains des paramètres graphiques de cette fonction peuvent être modifiés temporairement à l'appel d'une focntion. Exemples:
1) `par(lty = 2) ; plot(x,y) ; lines(a,b)`
2) `plot(x,y) ; lines(a,b,lty = 1)`
Ci-dessus, les 2 courbes de l'exemple 1 seront en pointillé tandis que dans l'exemple 2, seule la courbe produite par **lines** sera en pointillé.

3.4.5 Famille des fonctions Trellis

Il existe aussi dans S-PLUS une (plus ou moins) récente famille de fonctions graphiques nommée Trellis. Un usager intéressé par le graphisme aurait intérêt à consulter le manuel *Trellis Graphics User's Manual* qui présente cet autre ensemble de fonctions graphiques.

3.5 Fonctions statistiques

Il existe une grande variété de fonctions statistiques avancées dans différents domaines: régression, ANOVA, multidimensionnel, séries chronologiques, non paramétrique, etc. Ce document ne présente que 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 valeurs 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 regénérer 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'histogramme 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, _binom, _geom, _hyper, _nbinom, _pois, _weibull et plusieurs autres...: 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, Binomiale, Géométrique, Hypergéométrique, Binomiale négative, Poisson et Weibul, etc..
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^2_5:
   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 Exp(1):
   rexp(5)
   [1] 1.2752478 2.3055170 1.6694114 0.1327236 0.6108058

3.6 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")** (c'est stupide mais c'est comme ça...)

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 =
                "lol", 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")
}
```

Cette instruction doit être utilisée avec grande modération. S-PLUS est reconnu comme très lent avec l'utilisation de l'instruction **for** . L'approche matricielle de S-PLUS permet d'éviter très souvent la boucle **for** ...
La section suivante traite des problèmes de S-PLUS raltivement à cette fonction et à la gestion de la mémoire. Cette section a été originalement écrite par Christian Boudreau et Nancy Forget.

```
GESTION DE LA MÉMOIRE
     Un des défauts de Splus, est sa manière de gérer sa mémoire.
     L'utilisation des boucles "for" est un exemple fréquemment rencontré.

     Voici une boucle "for" traditionnelle:

          for (i in 1:5000) {
          v <- var...
          w <- sqrt...
          x <- apply....
          y <- apply....
          z <- mean....

     # affichage du "i-ème" numéro de la boucle et du temps à tous les 250
     # tours... (tout ce qui est mis en "#" est un commentaire dans Splus).

          if (i%%250==0) {cat(i, ") ", date(), "\n")}
          }

     Le problème avec ce type de fonction "for" est au niveau de
     l'allocation de la mémoire. Cette allocation est effectuée à chaque
     itération, en fonction du travail effectué lors d'une itération. Ainsi,
     le processus associé grossit en espace mémoire plus le nombre
     d'itérations augmente. Comme Splus ne se débarasse pas très bien de la
     mémoire qui n'est plus utilisée, on se retrouve avec de l'espace
     mémoire rendue inutile par Splus, que personne ne peut utiliser.

     Pour faire un exemple, supposons qu'une itération de la fonction "for"
     précédente a besoin de "1024 bytes = 20 K" par tour de boucle. On se
     retrouve donc à la fin du "for", avec 5000 x 20K de mémoire => 100 Mb
     utilise par Splus. Le problème avec cette méthode est que la mémoire
     n'est rendue disponible qu'à la fin de toutes les itérations. Donc,
     pour des boucles "for" avec un grand nombre d'itérations, la mémoire
     utilise peut facilement grossir, ce qui ralentit Splus ainsi que les
     autres usagers sur la station ou le serveur utilisé. Effectivement,
     lorsqu'il n'y a plus de mémoire "vive" disponible, la machine utilise
     sa mémoire virtuelle qui est beaucoup plus lente d'accès, ralentissant
     toutes les applications. Une borne sur la mémoire maximum sans trop
     ralentir les autres processus serait autour de la moitié de la mémoire
     vive disponible sur la station ou le serveur. Pour régler ce problème,
     on peut utiliser la boucle "For" (F majuscule). Ceci a pour avantage de
     ne considère qu'un certain nombre d'itérations à la fois au lieu d'un
     bloc total d'itérations, réduisant la perte de mémoire. Par exemple, au
     niveau de notre exemple précédent, voici l'équivalent avec
     une boucle "For" :

          For(i=1:5000,
          {
```

```
v <- var...
w <- sqrt...
x <- apply....
y <- apply....
z <- mean....
if (i%%250==0) {cat(i, ") ", date(), "\n")}
}, grain.size=10)
```

```
Le grain.size définit le nombre d'itérations à effectuer , dans des
processus indépendants de la fonction "For". Ceci veut dire que les
5000 itérations seraient divisées en 500 processus de 10 itérations,
avec chaque processus utilisant sa propre mémoire allouée. De cette
manière on obtient possiblement une économie sur la mémoire, augmentant
la rapidité d'exécution du programme Splus. À noter qu'il est possible
que l'"object.size" soit trop petit lorsqu'on utilise ce type de
boucle. C'est qu'avec un "For", la mémoire est allouée dès le départ du
processus Splus associé, et non pas au fur et à mesure. Donc, il est
possible que la taille maximale d'un objet (object.size) a besoin
d'être ajustée.
```

```
options(object.size= 15000000)
```

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.

3.7 Fonctions de manipulations de caractères

paste fonction permettant de "coller" plusieurs objets ensemble et de produire un vecteur ou une chaîne de caractères. Très utile pour l'impression et la fabrication de chaîne de caractères (dans les graphiques entre autres). Exemples:
1) `paste(1:3,c("A","B","C"), sep="*-*")` produira le vecteur de 3 éléments suivant:
`[1] "1*-*A" "2*-*B" "3*-*C"`
2) `paste(1:3,c("A","B","C"), collapse="*-*")` produira la chaîne suivante:
`[1] "1 A*-*2 B*-*3 C"`
3) `title(main=paste(``Graphique pour B =''',total),sub=paste("Version",date(),"No",n))`

substring extrait une chaîne de carctères. Exemples:
1) `substring("Bonjour",2,4)` produit la chaîne `"onj"`
2) `substring("Bonjour",2)` produit la chaîne `"onjour"`
3) `substring("Bonjour",2,2:4)` produit le vecteur suivant: `[1] "o"  "on"  "onj"`

3.8 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'usager (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

.First() L'utilité de cette fonction se compare à celle du fichier **.login** dans UNIX. La fonction **.First** sera exécutée au moment du démarrage d'une session S-PLUS. Cete fonction peut contenir toute instruction S-PLUS. Exemple, avec commentaires à droite:

```
function()
{
cat("Bonjour!!\n")                        # Imprime un message...
cat(paste("Aujourd'hui:", date(), "\n")) # Imprime un message incluant l'execution
                                          #     de la fonction date()
options(editor = "pico -t")               # Redefinir l'editeur par defaut de la fonction fix()
invisible()                               # Indique que la fonction ne produit aucun resultat
}
```

3.9 Fonctions d'édition

La fonction S-PLUS **fix** permet d'employer un éditeur UNIX pour éditer les objets S-PLUS. Malheureusement(?), l'éditeur par défaut est `vi` ... Par contre, il est possible de se définir un autre éditeur par défaut. Voici la marche à suivre:

1. Exécuter l'instruction suivante:
   options(editor = "*nom_de_l'éditeur*"**)**
   où "*nom_de_l'éditeur*" pourrait être: `"pico -t", "emacs", "xemacs"` ou tout autre éditeur

2. Créer un objet nommé obligatoirement **.First** à l'aide de la commande:
   fix(.First)
   et y inclure les lignes suivantes (pour l'éditeur xemacs, par exemple):
   ```
   function() {
   options (editor = ``xemacs'') }
   ```

3. Lors des prochaines sessions de S-PLUS, l'exécution initiale de la fonction **.First** définira l'éditeur mentionné comme éditeur par défaut avec l'instruction **fix** .

Par la suite, il suffira d'éditer l'objet désiré de la façon suivante:
fix(*nom_de_l'objet***)**
Si, en quittant l'édition, S-PLUS détecte une erreur de syntaxe, le message suivant apparaîtra: `Error in ...: Can't interpret...`
`Dumped`
`Errors occurred; Use fix() to re-edit this object.`
Tel qu'indiqué, il suffit alors de faire **fix()** pour revenir à l'édition de l'objet syntaxiquement erroné.

Il existe aussi une fonction **data.ed** qui permet d'éditer un objet (matrice, vecteur, *data frame*) à l'aide d'une fenêtre de type X11 et dans la philosophie de manipulation d'un chiffrier électronique...

Chapitre 4

Librairies locales

Malgré l'existence de nombreuses fonctions dans S-PLUS, il existe au CIRANO 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, 3 librairies maisons sont disponibles:

librairie locale de CIRANO librairie de fonctions générales. Voir l'appendice A;

DSE voir l'appendice B;

domouSe librairie de fonctions pour la détection de données multivariées aberrantes. Voir l'appendice C.

Remarques:

1. La librairie locale de CIRANO est automatiquement attachée lors d'une session S-PLUS
2. Les autres librairies peuvent être attachées à l'aide de la fonction **attach()** mais il existe une fonction locale spécifique à chaque librairie pour l'appeler plus facilement:
   attach.DSE()
   attach.domouSe()
3. La fonction **search()** permet de lister toutes les librairies attachées.
4. On peut lister les noms des objets d'une librairie attachée en tapant:
   ls(pos=i) $i > 1$
   où i représente la i^e librairie (répertoire) tel que définie par la fonction **search()**.

Appendice A

Librairie locale au CIRANO

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;

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);

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

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

fquad fonction permet de calculer la la fonction cumulative d'une somme de variables aléatoires χ^2. I.e.:
$P(\sum_{i=1}^{n} a_i X_i < x)$
où $X_i \sim \chi^2_{d_i}(\delta_i)$, i.e. de loi χ^2 avec d_i degrés de liberté et paramètre de non-centralité δ_i .
(Algorithme de Hoerts et Abrahamse selon la méthode d'Imhof)
Notons que cette fonction exige de charger d'abord dynamiquement une sous-routine FORTRAN ainsi:
dyn.load("/usr/local/cirano/splus-3.4/librairies_locales/CIRANO/fquad.o")

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.;

attach.DSE, attach.domouSe fonctions permettant d'appeler les autres librairies locales

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

Appendice B

Librairie DSE

La librairie DSE (**D***ynamic* **S***ystems* **E***stimation*) contient plusieurs centaines de fonctions d'estimation et de transformation de séries chronologiques. On suggère de consulter le guide d'usager disponible au laboratoire. On peut "attacher" cette librairie à l'aide de la fonction **attach.DSE()** .

Appendice C

Librairie locale domouSe

Note: le nom de cette librairie provient de **d***etection* **o***f* **m***ultivariate* **o***utliers* **u***sing* **S**-*PLUS* **e***nvironment*...
Cette librairie contient des fonctions relatives à la détection de données multivariées aberrantes.
On peut "attacher" cette librairie à l'aide de la fonction **attach.domouSe()** .
Cette librairie contient les fonctions suivantes:

dom.dyn.load2 fonction semblable à dyn.load (inutile sous le système d'exploitation au CIRANO);

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***)**.

Appendice D

Exemples

Le genre de logiciel comme S-PLUS "s'apprivoise" souvent en consultant des programmes (fonctions) déjà écrits. Cet appendice contient donc une série de programmes S-PLUS plus ou moins commentés...

Exemple 1

```
function(nvar = 7)
{
par(mfrow = c(5, 1))
donnees <- scan("donnees.S_avant_oct94", list(toux = 0, sibilance = 0,
tirage = 0, sommeil = 0, activite = 0, b2 = 0, steroides = 0,
autre = 0), flush = T)
names(donnees)[6:7] <- c("B2 (Ventolin) (fois/jour)",
        "Steroide (Beclovent)")
for(i in 1:nvar)
        tsplot(ts(donnees[[i]], start = 1991 + 300/366, freq = 365),
                main = names(donnees)[i])
}
```

Exemple 2

```
function()
{
vect <- matrix(scan("donnees_du_120395"), ncol = 2, byrow = T)
vect <- vect[, 2]
nv <- length(vect)
par(oma = c(0, 0, 2, 0), mfrow = c(3, 1))
plot(vect, type = "h", xlab = "Lag", ylab = "ACF")
title(main = "DEM_USD Market", cex = 0.5)
abline(h = 1/sqrt(18792), lty = 2)
mtext("ACF for squared returns over twenty business minutes - O&A activity scale",
        3, 0, T, cex = 1)
invisible()
}
```

Exemple 3

```
function()
{
donnees <- scan("donnees", list(0, 0, 0, 0), flush = T)
#b2 <- ts(donnees[[1]], start = 1994 + 267/366, freq = 365)
#steroide <- ts(donnees[[2]], start = 1994 + 267/366, freq = 365)
par(mfrow = c(4, 1))
maxd <- max(donnees[[3]], donnees[[4]], na.rm = T)
mind <- min(donnees[[3]], donnees[[4]], na.rm = T)
for(i in 1:4) {
        plot(c(0, 1), c(mind, maxd), type = "n", xlab = "", ylab = "",
                ax = F)
        axis(2)
        axis(1, at = seq(1, 12, length = 12)/12, labels = c("Jan.",
                accent("F'{e}v."), "Mars", "Avril", "Mai", "Juin",
                "Juil.", accent("Ao^{u}t"), "Sept.", "Oct.", "Nov.",
                accent("D'{e}c.")))
        abline(h = c(50, 100, 150, 200), col = 3, lty = 3)
        abline(v = seq(1, 12, length = 12)/12)
        box()
}
for(i in 1:3) {
        par(mfg = c(i, 1, 4, 1))
        title(paste(accent("Ann'{e}e"), 1993 + i))
}
noms <- c("A", "B", "C", "D")
par(mfg = c(4, 1, 4, 1))
title(accent("Ann'{e}es cumul'{e}es"))
for(iserie in 1:2) {
        serie <- ts(donnees[[iserie + 2]], start = 1994 + 267/366, freq
                 = 365)
        time.serie <- time(serie)
        for(i in 1:3) {
#                   3 annees a date...
                condition <- time.serie < 1994 + i & time.serie >= 1993 +
                        i
                xl <- time.serie[condition] - (1993 + i)
                yl <- serie[condition]
                par(mfg = c(i, 1, 4, 1))
                lines(xl, yl, lty = iserie)
                par(mfg = c(4, 1, 4, 1))
                lines(xl, yl, lty = iserie)
        }
}
invisible()
}
```

Remarque:
On notera que la fonction se termine par **invisible()**. Sinon, la fonction produirait une "valeur" `NULL` non nécessaire (puisque la fonction se termine par la fonction **lines**)

Exemple 4

```
function(lambda = 0.3, nu = 0.75, beta.inf = 1, K = 30, alpha = 0.8, delta = NULL)
{
W.sup.P <- function(d, lambda, nu, beta.inf, beta.sup, delta.beta, K)
{
        (-0.5) * d^2 - (1 + lambda) * (nu * beta.inf + (1 - nu) *
                beta.sup) * (K - d) - nu * lambda * delta.beta * (K - d
                )
}
W.sup.S <- function(d1, d2, nu, lambda, beta.inf, beta.sup, K,
        delta.beta)
{
        V <- function(d)
        {
                0.5 * d * d
        }
        nu * ( - V(d1) - (1 + lambda) * beta.inf * (K - d1) - lambda *
                delta.beta * (K - d2)) + (1 - nu) * ( - V(d2) - (1 +
                lambda) * beta.sup * (K - d2))
}
par(mfrow = c(1, 1))
beta.sup <- 3
delta.beta <- beta.sup - beta.inf
d11 <- (1 + lambda) * beta.inf
d21 <- (1 + lambda) * beta.sup + ((1 + lambda) * nu * delta.beta)/(1 -
        nu)
d12 <- (1 + lambda) * beta.inf
d22 <- (1 + lambda) * beta.sup + ((1 + lambda - 1/alpha) * nu *
        delta.beta)/(1 - nu)
d.sup.p <- (1 + lambda) * (nu * beta.inf + (1 - nu) * beta.sup) +
        lambda * nu * delta.beta
Wp <- W.sup.P(d.sup.p, lambda, nu, beta.inf, beta.sup, delta.beta, K)
Ws <- 0.5 * W.sup.S(d11, d21, nu, lambda, beta.inf, beta.sup, K,
        delta.beta) + 0.5 * W.sup.S(d12, d22, nu, lambda, beta.inf,
        beta.sup, K, delta.beta)        # Tracer Wp-Ws.......
# 2e partie....
d11.chapo <- (1 + lambda) * beta.inf
d.etoile <- ((1 + lambda - nu) * beta.sup - lambda * nu * beta.inf)/(1 - nu)
idelta <- seq(0, 2, by = 0.1)
d21.chapo <- d.etoile + (d21 - d.etoile)/(1 + idelta)
d12.chapo <- (1 + lambda) * beta.inf
d22.chapo <- d.etoile + (d22 - d.etoile)/(1 + idelta)
Ws.chapo <- 0.5 * W.sup.S(d11.chapo, d21.chapo, nu, lambda, beta.inf,
        beta.sup, K, delta.beta) + 0.5 * W.sup.S(d12.chapo, d22.chapo,
        nu, lambda, beta.inf, beta.sup, K, delta.beta)
plot(idelta, Wp - Ws.chapo, type = "l", xlab = "", ylab = "")
abline(h = 0)        #        title(paste("delta = ", idelta))
invisible()
```

Remarque:
On remarque dans cette fonction qu'il y a des fonctions locales. Les fonctions `W.sup.P` et `W.sup.S` sont des fonctions définies localement dans la fonction. Elles ne pourraient être appelées dans une autre fonction. Et il y a même une fonction locale (`V`) définie localement dans la fonction locale `W.sup.S` !!!

Exemple 5

```
function(prefixe, borne)
{
# fonction calorlando calquee sur la fonction densitep
liste <- unix("ls /export/home2/ghyselse/ggs/orlando")
long <- 1:500
nlong <- 500
resul <- paste(prefixe, ".resul", sep = "")
iliste <- grep(paste("^", prefixe, sep = ""), liste)
cat(date(), ";", "Prefixe:", prefixe, "Borne:", borne, "\n", append = T, file = resul)
for(i in iliste) {
        ft <- scan(paste("/export/home2/ghyselse/ggs/orlando/", liste[i], sep = ""))
        ft <- ft[!is.na(ft)]
        lft <- length(ft)
        if(lft >= nlong) {
                ft <- ft[long]
                lft <- length(ft)
                complet <- T
        }
        else complet <- F
        prop.sup <- round((sum(ft >= borne) * 100)/lft, 2)
        prop.inf <- round((sum(ft < borne) * 100)/lft, 2)
        cat(liste[i], " ; ", ifelse(complet, "", paste("Incomplet (",
                lft, ") ; ", sep = "")), " % >= a borne:", prop.sup,
                " ; % < a borne:", prop.inf, "\n", append = T, file = resul)
}
invisible()
}
```

Remarque:
On notera l'utilisation de la fonction S-PLUS **grep** qui ressemble à la fontion du même nom dans UNIX... Par contre, la fonction **unix(...)** est la fonction S-PLUS qui exécute une fonction UNIX (**ls** dans cet exemple).

Exemple 6

```
function(var, objet)
{
ensemble <- c("DEM_USD", "JPY_DEM", "JPY_USD")
var.vect <- c("Quotes", "Bid-Ask Spreads", "Absolute Returns")
indice <- switch(var,
        quote = 1,
        pbpa = 2,
        dxt = 3,
        stop("Erreur de parametre"))
n <- objet[[1]]$n.used
borne <- 1/sqrt(n)
par(oma = c(0, 0, 2, 0), mfrow = c(3, 1))
for(graph in (1:3)) {
        plot(c(objet[[graph]]$acf[-1,  ,  ], borne,  - borne), type =
                "n", xlab = "Lag", ylab = "ACF", main = paste(ensemble[ graph], "Market"))
        lines(objet[[graph]]$acf[-1,  ,  ], type = "h")
        abline(h = c(borne,  - borne), lty = 2)
}
mtext(paste("Figure 3.", indice + 3, " Deviations from Averages: ",
        var.vect[indice], " 20 Min. Intervals - ACF"), 3, 0, T, cex = 0.8)
invisible()
}
```

Remarque:
On notera l'utilisation de la fonctions **switch** qui ressemble à une instruction **case** fréquente dans d'autres langages de programmation. Cette instruction aurait s'écrire plus "lourdement" avec l'instruction **if**:

```
if( var == "quote" )
   indice <- 1
else if ( var == "pbpa" )
   indice <- 2
else if ( var == "dxt")
   indice <- 3
else stop("Erreur de parametre")
```

Exemple 7

```
function(annee, mois, long.mois)
{
options(object.size = 10000000)
fich.temp <- paste("/tmp/S_extrait.", mois, ".dat", sep = "")
cat(unix(paste("grep '", annee, "-", mois,
        "' /usr/local2/donnees/HFDF93/DEM_USD.dat", sep = ""), output
         = T), file = fich.temp, sep = "\n")
table <- scan(fich.temp, list(date = "", temps = "", bid = 0, ask = 0,
        country = 0, city = 0, bank = 0, filter = 0))
resul <- paste("mois", mois, ".dat", sep = "")
cat("", file = resul)
table2 <- list()
for(jour in 1:long.mois) {
        print(jour)
        j <- as.numeric(substring(table[[1]], 9, 10))
        select <- (1:length(j))[j == jour]
        if(length(select) == 0)
                select <- max((1:length(j))[j == (jour - 1)], na.rm = T)
        else if(jour != 1)
                select <- c(select[1] - 1, select)
        print(select[1:10])
        for(i in 1:8)
                table2[[i]] <- table[[i]][select]
        for(heure in 0:23)
                for(minute in seq(0, 40, 20)) {
                        h <- as.numeric(substring(table2[[2]], 1, 2))
                        m <- as.numeric(substring(table2[[2]], 4, 5))
                        s <- as.numeric(substring(table2[[2]], 7, 8))
                        if(jour != 1) {
                          s[1] <- s[1] - 60
                          m[1] <- m[1] - 59
                          h[1] <- h[1] - 23
                        }
                        diff <- s + (m - minute) * 60 + (h - heure) * 3600
                        t <- max((1:length(diff))[diff <= 0], na.rm = T)
                        cat(as.numeric(mois), jour, heure, minute,
                          table2[[1]][t], table2[[2]][t], table2[[3]][t
                          ], table2[[4]][t], table2[[5]][t], table2[[6
                          ]][t], table2[[7]][t], table2[[8]][t], "\n",
                          file = resul, sep = " ", append = T)
                }
}
unix(paste("/bin/rm", fich.temp))
invisible()
}
```

Exemple 8

```
function()
{
# Fonction verif.volume
#
#Fonction locale julien
julien <- function(a)
{
        an <- trunc(a/10000)
        mois <- trunc(a/100) - an * 100
        jour <- a - (an * 10000 + mois * 100)
        a <- julian(mois, jour, an + 1900, origin = c(1, 1, 1984))
        a
}
mat1 <- matrix(scan("totvol1.out"), ncol = 2, byrow = T)
tempo <- matrix(scan("totvol2.out"), ncol = 2, byrow = T)
print(sum(mat1[, 1] - tempo[, 1]))
mat1[, 2] <- mat1[, 2] + tempo[, 2]
mat1[, 1] <- julien(mat1[, 1])
mat2 <- matrix(scan("spadjust.yr2887"), ncol = 9, byrow = T)
mat2 <- mat2[, c(1, 8)]
mat2[, 2] <- exp(mat2[, 2])
mat2[, 1] <- julien(mat2[, 1])
mat1 <- cbind(mat1, NA)
l <- dim(mat2)[1]
print(sum(mat1[1:l, 1] - mat2[1:l, 1]))
mat1[1:l, 3] <- mat2[1:l, 2]
print(mat1[c(1:11, 1000:1020),  ])
plot(mat1[1:l, 1], mat1[1:l, 2], type = "l", axes = F)
axis(1)
axis(2)
par(new = T)
plot(mat1[1:l, 1], mat1[1:l, 3], type = "l", axes = F, lty = 2)
axis(4)
box()
legend(0, 600000, legend = c("CRSP", "SPAD"), lty = 1:2)
invisible()
}
```

Remarque:
Cet exemple superpose deux courbes dans le même graphique (en utilisant la fonction **par(new = T)**) et associe aux axes des valeurs différentes (utilisation de la fonction **axis**).