Un peu de SAS dans un grand verre...

Gilles Hunault

Université d'Angers, 1999

Table des matières

Préface

Chapitre 1.

Présentation de SAS

1.1 Qu'est-ce que SAS ?

SAS est à la fois un système de gestion de fichiers, un outil de calcul statistiques, un atelier de développement de programmes multi-plateformes (dos, unix, mac mais aussi vm/sp, mvs etc.). Pour qu'on puisse efficacement communiquer avec *SAS*, les développeurs de SAS ont créé un langage de commandes et de programmation, le **SASL** (ou "langage de *SAS*").

Nous présentons ici les interfaces de *SAS*, le langage de *SAS* et comment réaliser des calcus de Statistique Descriptive en *SAS*. Nous laissons le soin au lecteur, à la lectrice de vérifier tout ce que nous écrivons en effectuant les travaux pratiques correspondant sur ordinateur.

On trouvera aussi en fin de document quelques remarques sur ce qu'est aussi *SAS*, à savoir un logiciel "monstre", qui fait "tout" et qui peut "tout" ou presque – mais à condition d'y mettre le prix et d'investir dans la programmation *SAS*. Par exemple, quel logiciel est utilisé par la Banque de France depuis 30 ans pour gérer ses fichiers et ses données ? Certainement pas *Microsoft Excel*, mais peut-être *SAS*...

1.2 Les trois interfaces de SAS

On communique avec *SAS* au moyens d'instructions qui se termine obligatoirement par un point-virgule. Les interfaces de base sont respectivement les fichiers, le clavier et la souris.

1.2.1 Interface Ligne

Les instructions *SAS* peuvent être écrites dans un fichier et soumises en mode ligne à Sas. L'interface est alors minimale : *SAS* se contente d'exécuter le fichier, disons CALC.SAS et met dans les deux fichiers CALC.LOG et CALC.LST le résultat de l'exécution. CALC.LST contient les calculs effectués ; CALC.LOG, quant à lui, décrit ce que *SAS* a fait, ce qu'il a compris, combien de temps a duré chaque étape...

1.2.2 Interface fenêtrée

L'interface traditionnelle se compose de trois fenêtres : la fenêtre *Program editor* ou plus simplement *Editor* (accessible par *F5* ou en activant la commande *Program Editor* du menu *Fenêtre*), la fenêtre *Log* (*F6*) et la fenêtre *Output* (*F7*). On écrit les instructions dans la fenêtre Editor et on soumet (par *F3* ou par la commande Exécuter du menu *Général*) ces instructions à *SAS*. On visualise alors les résultats dans la fenêtre *Output* et en cas de problème on vient lire le détail des actions effectuées ou ratées par *SAS* dans la fenêtre *Log*. Une fois soumis, le texte dans Editor disparait mais on peut le rappeler par la touche *F4* (ou la commande *Rappeler le texte* du menu *Local*).

1.2.3 Interface assistée

L'interface d'assistance de *SAS* se lance par la commande *Sas/Assist* du menu *Général*. On dispose alors d'icones et de panneaux à cliquer qui réduisent au maximum le temps de frappe (mais masquent les instructions). On exécute alors le calcul par la commande *Exécuter* du menu *Local*. Il est possible de sauvegarder le texte des instructions correspondant aux "clics" en activant la commande *Visualiser le code source* du menu *Local*.

On sort du mode *Assist* en activant les différentes commandes *Retour* jusqu'au moment où on arrive au panneau principal de *Sas/Assist* dont on sort en activant la commande *Sortie*. Cette interface est utile pour débuter, pour se faire une idée de l'ensemble des modules. Par exemple, si dans le *Panneau Principal* on active le panneau *Environnement* puis la ligne *Tables SAS d'essai*, *SAS* propose de créer (ou de recréer) des exemples de fichiers de données de façon à pouvoir tester des calculs sur des fichiers ad hoc.

Au lieu de taper des instructions dans la fenêtre *Editor*, on peut également faire du copier/coller à partir d'un autre éditeur, ou charger un ensemble d'instructions par la commande *Open* du menu *File*, ou encore, et c'est la solution que nous préférons, inclure le texte du programme par l'instruction

```
%include 'nom_du_fichier'.sas;
```

De même, nous conseillons d'exécuter la commande *Nouveau* dans la fenêtre *Output* avant de relancer une suite de commandes de façon à être bien sûr(e) de visualiser les résultats des commandes et non pas ceux de la série précédente.

Lors de l'inclusion de fichiers, il est conseillé de mettre l'option globale Source2 à On pour les lignes incluses avec leur numéro apparaissent dans le *Log*.

1.3 Comment apprendre SAS ?

Tout d'abord en s'armant de patience !

SAS est un système logiciel complet, avec une mécanique très lourde mais efficace. Le lanagage de sas, qui se veut langage exhaustif de traitement de données, permet de tout faire, à condition d'utiliser la bonne procédure (ou plutot le bon paramétrage de la bonne procédure).

Il faut en général du temps pour se familiariser avec la philosophie de *SAS*, sa façon de parcourir les données, sa gestion ligne par ligne des variables.

Par exemple, pour cumuler des ventes, on pourrait croire que le programme

```
data ventes ;
input ventes ;
if _n_ = 1 then cumul = ventes ;
           else cumul = cumul + ventes ;
datalines ;
100
300
200
;
proc print data=ventes ;
run ;
```

est suffisant. C'est une erreur : la variable cumul "disparait" d'une ligne de données à l'autre et l'affichage des résultats à savoir :

```
OBS    VENTES    CUMUL

 1       100      100
 2       300        .
 3       200        .
```

semble donc incompréhensible au profane...

Pour apprendre *SAS*, il faut commencer par connaitre le nom des procédures de base, puis lire leur syntaxe complète, la liste de leurs options. Cela demande de nombreux essais (parfois un essai pour chaque option), avec un apprentissage parfois des difficultés conceptuelles. Ainsi keep ne retient que les variables citées, drop ne retient pas que les variables passées en paramètres. Et si on effectue[1] un keep et un drop de la même variable? ou si on "droppe" tout, que fait *SAS*? Essayer sur un exemple n'est parfois pas suffisant et il faut tout le *feeling* du gourou *es SAS* pour parfois trouver la solution.

1. seul un enseignant se poserait la question aussi directement ; en fait, par le jeu de substitution de variables, il peut arriver de faire un keep et un drop d'une même variable...

Chapitre 2.

SAS, langage de commandes

Le langage de *SAS* est à la fois un langage de commandes et de programmation. Commençons par les commandes. Il y en a trois fondamentales : DATA, PROC et RUN. DATA permet de décrire les données et les fichiers associés, PROC définit la procédure de calcul à utiliser, RUN ne sert qu'à exécuter le traitement. Elle est obligatoire en fin de texte avant exécution. L'instruction DATA n'est pas obligatoire. Par exemple, si les tables d'essais de *SAS* ont été créées comme indiqué au paragraphe précédent, les deux lignes

```
* tut01.sas ;
proc print data=sasuser.habitat ; run ;
```

affichent le contenu du fichier d'exemple nommé habitat.

2.1 L'instruction DATA

Le mot DATA indique en fait le début de toute une "étape des données". Cette étape ne s'arrête que lorsqu'on rencontre l'instruction PROC ou une autre instruction DATA ou l'instruction RUN. Pour de petits jeux de données ou des calculs "jetables", on utilise une étape DATA/INPUT/DATALINES qui se prépare comme suit. On met après DATA le nom du fichier à créer sous forme d'un seul mot d'au plus 8 caractères. On met ensuite un point-virgule pour indiquer que la ligne DATA est terminée et on décrit par la ligne INPUT le format des données (voir exemple). Après l'instruction DATALINES terminée elle-aussi par un point-virgule, on met les données à utiliser à raison d'une ligne de données par ligne physique et enfin on termine l'étape par un dernier point-virgule.

Par exemple, pour jouer à la marchande, nous voulons comptabiliser des ventes. On dispose du prix unitaire PU et du nombre d'articles NA. On veut simuler les deux ventes NA=10, PU=8 et NA=3, PU=11. On écrira pour la partie définition des données:

```
* tut02.sas ;
data marchan ;
 input na pu ;
datalines ;
10 8
3 11
;
run ;
```

Lorsqu'on dispose de données plus conséquentes, il vaut mieux laisser les données dans un fichier externe, les référencer par l'instruction FILENAME et le charger par l'instruction INFILE. Si les donnés ne sont pas seulement délimitées (comme dans l'exemple précédent) mais bien cadrées, dizaine sous dizaine, unité sous unité, on donne à *SAS* les positions physiques des variables dans la ligne des données à la suite du nom de la colonne. Ainsi avec le fichier-programme

```
* tut03.sas ;
filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data essai ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

** pour vérification ;

proc print data=essai ;
run ;
```

SAS comprend que nos données sont dans le fichier z:\elf.dat, que les quatre premiers caractères de la ligne forment la colonne ID et que cette colonne (à cause du symbole $) doit être traitée comme du texte et pas comme du numérique, que la colonne SEXE correspond au caractère numéro 9 dans la ligne etc.

Les fichiers créés sont temporaires et n'existent pas lorsqu'on termine la session *SAS*. Il est possible de créer des fichiers dits permanents mais nous ne le détaillons pas ici, pas plus que le vocabulaire propre à *SAS* (librairie, référence, comme avec libname, libref, fileref, filename...).

Dans l'étape DATA, on peut également donner un libellé (LABEL) à une colonne, lui associer un format de description (mais il faut définir ces formats par la procédure FORMAT). FORMAT définit de nouvelles variables (en général texte) qui font la correspondance entre une valeur (ou une plage de valeurs, comme 12-20 pour indiquer entre 12 et 20, bornes comprises) et une chaine de caractères. Attention : il faut mettre un point après la variable de FORMAT dans l'option FORMAT de DATA car plusieurs variables peuvent partager le même format et *SAS* a besoin du point pour s'y retrouver...).

Voici un exemple de telles opérations :

```
* tut04.sas ;
proc format ;
 value Sexf 0='Homme' 1='Femme' ;

data work.elf ;
 infile 'l:\gh\crs\stat\trysas\elf.dat' ;
 input ID $ 1-4 SX 9 ;
 label SX='Sexe de la personne' ;
 format SX Sexf. ;

** pour vérification ;

proc print data=work.elf ;
run ;
```

Enfin, dans l'étape DATA, on peut changer de nom base avec SET, recoder les données ou construire de nouvelles colonnes, détruire des lignes via DELETE. Par exemple, pour la marchande, nous voulons calculer les sommes gagnées pour les plus grosses ventes. On décide pour cela de calculer le prix de vente PV égal au produit du nombre d'articles par le prix unitaire et de ne pas retenir les lignes telles que pu $<$ 5 ou pv $<$ 30. On veut nommer VENTES la table *SAS* correspondante.

Voici les instructions.

```
* tut05.sas ;
data work.marchan ;
 input na pu ;
 pv  = na*pu ;
 if pu <  5 then delete ;
 datalines ;
 10 8
 3 11
 1  1
 7  2
 1  7
 10 50
 50 12.5
;

data work.ventes  ;
 set work.marchan ;
 if pv < 30 then delete ;

** pour vérification ;

proc print data=marchan ;
run ;

proc print data=ventes ;
run ;
```

2.2 L'instruction PROC

Le mot PROC indique le début d'une étape-procédure. Le mot suivant est le nom de la procédure, suivi du mot DATA= pour indiquer quelles sont les donnés. Suivent alors les paramètres éventuels de la procédure avec éventuellement un / pour spécifier des options. Les autres options de l'étape procédure sont définies par des mots propres à la procédure, par exemple VAR pour spécifier la liste (et surtout l'ordre) des variables à utiliser, BY pour afficher de facçon groupée. Les procédures usuelles sont PRINT, MEANS, CORR, FREQ, PLOT, CHART.

Elles permettent respectivement d'imprimer, d'effectuer des calculs statistiques de base sur des données quantitatives et des comptages, de tracer des courbes et histogrammes.

La première procédure à savoir utiliser est certainement PRINT. Elle affiche les données. Ainsi, après l'étape DATA, on met souvent (au moins en début de session *SAS*) une instruction PROC PRINT pour vérifier que les données ont été correctement lues. Une étoile en début de ligne indique un commentaire, c'est à dire une ligne ignorée par *SAS*. Par exemple :

```
* tut06.sas ;
* création du jeu d'essai pour la marchande ;
data marchan ;
 input na pu ;
datalines ;
10 8
3 11
;
* affichage des données ;
proc print data=marchan ;
run ;
```

Pour imprimer de façon ordonnée, il faut utiliser le mot BY après avoir trié via la procédure SORT les données. Ainsi dans le dossier Elf, nous voulons utiliser l'affichage de politesse, à savoir les femmes d'abord, les hommes ensuite, les personnes d'age élevé avant les plus jeunes. Décidant de n'afficher que l'identificateur, le sexe et l'age des personnes dans cet ordre, nous écrirons les lignes suivantes

```
* tut07.sas ;

* définition des données ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

run ;
```

```
/* tri selon l'ordre de politesse ;
   femme d'abord, homme ensuite
   les vieux d'abord, les jeunes ensuite
   code sexe : 0=homme, 1=femme           */

proc sort data=elf  out=elf_Poli;
  by descending sexe descending age ;

* affichage sélectif dans l'ordre ;

proc print data=elf_Poli ;
  var id sexe age ;

* exécution ;

run ;
```

2.3 Quelques procédures utiles

La procédure CONTENTS permet de connaitre ce que contient une table sas. Exemple d'utilisation :

```
* tut08.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

* vérification du stockage de données ;
proc contents data=marchan ;

proc contents data=sasuser.habitat ;

proc contents data=elf ;

run ;
```

En particulier, pour savoir ce que contient tout la librairie utilisateur :

```
/*
   +=================================================+
   +                                                 +
   +  attention : cela produit des tonnes de pages, +
   +  mais au moins on a tout le détail !           +
   +                                                 +
   +=================================================+

*/

proc contents data=sasuser._all_ ;
run ;
/*
*
* au mieu de sasuser, on peut mettre aussi
* library, maps, sashelp, work
*
*/
```

2.4 Fichiers temporaires et permanents

Une référence de fichier (*fileref*) au sens de *SAS* est le nom attribué aux données en cours de traitement. Si ce nom se compose d'un seul mot, comme par exemple ventes alors le fichier est temporaire et disparaitra lorsqu'on quitte *SAS*. Si ce nom est composé de deux mots reliés par un point, alors le fichier est permanent. Le premier mot fait référence à une "librairie" (*libref*) et le second mot est le nom du fichier. Le fichier sera écrit à l'endroit correspondant pendant la session. On associe un nom de répertoire à une libraire par la commande libname.

Supposons par exemple qu'on exécute les commandes suivantes dans une session *SAS* :

```
libname enquet 't:\stat\enq\99\surv' ;
data testUn ;
...
data cours.testDeux ;
```

...

alors le *dataset* référencé par testUn est temporaire alors que alors le *dataset* référencé par enquet.testDeux est permanent et est stocké dans le fichier t:\stat\enq\99\surv\testDeux.sd2.

A l'usage, ce mécanisme de référence se révéle souple : d'un ordinateur à l'autre, il suffit de modifier le *libname* pour que tous les fichiers correspondent. Sur un même ordinateur, changer de *libname* permet d'utiliser des fichiers de même nom situés dans des répertoires différents.

Chapitre 3.

Calculs statistiques en SAS

3.1 Traitement des Variables Quantitatives

La procédure de base est MEANS. *SAS* l'appique à toutes les colonnes numériques (même si les nombres correspondent à des codes). Par défaut, *SAS* affiche le nombre de valeurs (N), la moyenne (MEAN), l'écart-type (STD), le minimum (MIN) et le maximum (MAX). Ainsi pour notre dossier Vins qui ne contient que des variables numériques quantitatives, il suffit d'écrire

```
* tut10.sas ;

filename vins 'l:\gh\crs\stat\trysas\vins.dat' ;
data vins ;
 infile vins ;
 input TYPEV $ BELGIQ NEDERLD RFA    ITALIE
              UK     SUISSE  USA    CANADA ;
 list ;

proc means data=vins ;
run ;
```

Pour n'afficher que certaines caractéristiques, on les indique après la spécification de données. Si on veut certaines variables seulement, on utilise VAR et si on veut traiter des sous-populations, on écrit BY (il est nécessaire d'avoir trié auparavant).

Ainsi pour avoir le nombre de personnes, la moyenne et le coefficient de variation (CV=σ/m) de l'age pour chaque sexe du dossier Elf, on écrit

```
* tut11.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc sort data=elf out=elf_Tri ;
 by sexe ;

proc means data=elf_Tri n mean cv maxdec=1;
 var age ;

proc means data=elf_Tri n mean cv maxdec=1;
 var age ;
 by  sexe ;

run ;
```

Il y a d'autres paramètres disponibles pour MEANS dont NMISS, nombre de valeurs manquantes, MAXDEC pour préciser le nombre de décimales à l'affichage, RANGE pour l'étendue (c'est la différence maximum moins minimum), SUM pour la somme des valeurs, VAR pour la variance, SKEWNESS, KURTOSIS, SUMWGT, CSS, STDERR... La seconde procédure à utiliser est CORR. Elle calcule les divers coefficients de corrélation (le ρ classique de Pearson, mais aussi celui de Spearman et celui de Kendall pour les rangs, les coefficients de Cronbach, de Hoeffding etc.). Par défaut, *SAS* calcule les caractéristiques m, σ etc. de toutes les variables numériques puis affiche une matrice de corrélation avec en-dessous de ρ, la probabilité de dépasser $|R|$ dans l'hypothèse H_0 (hum !) et le nombre d'observations sur lesquelles ρ a été calculé.

On peut bien sur avec CORR spécifier des corrélations partielles, faire des corrélations entre une variable et une liste d'autres variables, etc., n'afficher que les n meilleures corrélations (avec BEST=n mais attention la machine donne toujours $\rho(X,X) = 1$ donc BEST doit être toujours supérieur à 2) etc.

Pour obtenir la matrice des corrélations classiques, on prend les options NOPROB et PEARSON, soit, pour le dossiers Vins :

```
* tut12.sas ;

filename vins 'l:\gh\crs\stat\trysas\vins.dat' ;
data vins ;
 infile vins ;
 input TYPEV $ BELGIQ NEDERLD RFA    ITALIE
               UK     SUISSE  USA    CANADA ;

proc corr data=vins pearson noprob ;

run ;
```

Le complément de MEANS est UNIVARIATE, celui de CORR est REG. Nous renvoyons le lecteur, la lectrice au manuel de référence, à l'aide "en ligne" pour en lire les détails et les options, fort nombreuses...

3.2 Traitement des Variables Qualitatives

La procédure de base est ici FREQ. Avec l'option TABLE, elle calcule les tris à plat en indiquant au passage les effectifs absolus et relatifs, le cumul des fréquences. Par exemple :

```
* tut13.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc freq data=work.elf  ;
  tables sexe etud prof ;
run ;
```

On peut bien sur choisir de ne pas afficher le cumul avec NOCUM, afficher par ordre décroissant d'effectifs avec ORDER=FREQ.

Par exemple on peut écrire :

```
* tut14.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc freq data=elf order=freq ;
  tables sexe etud prof / nocum ;

run ;
```

Pour un tri croisé, on écrit la même chose, sauf que dans la ligne TABLES on donne les couples de variables à croiser reliés par une étoile.

Ainsi pour le tri croisé sexe / études dans le dossier Elf, on écrira :

```
* tut15.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc freq data=elf  ;
  table sexe*etud ;

run ;
```

Pour obtenir un tri croisé "bien beau, bien propre", on utilisera des labels et des formats comme suit :

```
* tut16.sas ;

proc format ;
 value sexf  0='Homme' 1='Femme' ;
 value etudf 0='NR' 1='Primaire' 2='Bepc' 3='Bac' 4='Sup' ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
```

```
data   elf_beau ; infile elf ;
```

```
input  ID $ 1-4 SX 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;
label  SX='Sexe de la personne' ;
label  ETUD='Niveau d''études' ;
format SX sexf. ETUD etudf. ;

proc freq data=elf_beau ;
 table etud*sx / norow nocol nopercent ;

run ;
```

Signalons que la procédure TABLES calcule aussi des chi-deux, les coefficients de Fisher, de Stuart, Somer, Cramer, Mantel-Haenszel...

3.3 Tracé de courbes

On trace avec la procédure PLOT. L'instruction PLOT (si, si, c'est bien le même mot) est alors l'option de tracé. La syntaxe du choix des variables est, comme pour FREQ, y*x qui vient tracer y en fonction de x. Par exemple :

```
* tut17.sas ;

data ventes ;
input an vente ;
datalines ;
1995 127000
1996 150000
1997 134000
1998 186000
;
proc plot data=ventes ;
 plot vente*an ;
run ;
```

SAS met alors un A comme symbole de tracé, un B si deux points sont superposés sur le graphique. On peut choisir son symbole (ou faire mieux, si on n'imprime pas dans un fichier mais si on utilise un "grpahique écran"...). Avec les options hpercent et vpercent, on peut même plusieurs graphiques sur une même page.

3.4 Tracé d'histogrammes

Les histogrammes se tracent avec la procédure CHART, que ce soit en barres (ou histogrammes verticaux, option VBAR) ou en lignes, (ou histogrammes horizontaux, option HBAR) ou en secteurs angulaires (option PIE). *SAS* se charge de calculer les fréquences.

Si ces histogrammes sont faciles à produire, on se méfiera du fait qu'il sont en standard en mode texte, c'est à dire que ce sont des lignes de caractères "normaux" et non pas des images, comme avec *Excel* ou *Gnuplot*.

Pour l'histogramme des niveaux d'étude dans le dossier Elf, on écrira :

```
* tut18.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc chart data=elf ;  vbar sexe ;

proc chart data=elf ;  hbar sexe ;

run ;
```

La ventilation d'un histogramme se fait avec l'option GROUP. Signalons qu'on peut réaliser des graphiques en trois dimensions avec BLOCK plutot que HBAR, VBAR ou PIE.

```
* tut19.sas ;

filename elf 'l:\gh\crs\stat\trysas\elf.dat' ;
data elf ;
 infile elf ;
 input ID $ 1-4 SEXE 9 AGE 12-13 PROF 17-18
       ETUD 23 REGI 28 ACTIO 33 ;

proc chart data=elf ;  vbar sexe / group=etud ;

run ;
```

Finissons maintenant par un exemple plus conséquent. Reprenant le dossier Vins, nous décidons d'afficher les résultats statistiques par moyenne décroissante puis par ordre de coefficient de variation décroissant. Pour cela, nous stockons avec OUT les résultats de la procédure MEANS sans rien afficher (NOPRINT). Le fichier de résultat ne se présentant pas sous une forme convenable pour notre tri, nous le transposons avec la procédure TRANSPOSE. Ensuite, nous renommons les colonnes avec un identifiant plus explicite (*moy* plutot que *col*4, etc.). et nous calculons à la main le *cdv*. Il ne reste plus qu'à trier par ordre décroissant (SORT... BY DESCENDING...) et à afficher pour obtenir ce que l'on veut ! Voici donc le programme complet :

```
* tut20.sas ;

** 1. Lecture des données ;

filename vins 'l:\gh\crs\stat\trysas\vins.dat' ;
data work.vins ;
 infile vins ;
 input TYPEV $ BELGIQ NEDERLD RFA    ITALIE
              UK     SUISSE  USA    CANADA ;

** 2. Calculs ;

proc means  data=work.vins  noprint ;
     output out=work.vins_gsa ;

** 3. Transposition du fichier de résultats ;

proc transpose data=work.vins_gsa out=work.vins_asg;

** 4. Suppression des lignes "parasites"
      et calcul du cdv ;

data work.vinsasg ;
     set work.vins_asg ;
     if col1=0 then delete ;
     if _name_='_FREQ_' then delete ;
     min = col2 ; max = col3 ;
     moy = col4 ; ect = col5 ;
     cdv =100.0*ect/moy ;
```

```
** 5. Tri par moyenne décroissante ;

proc sort  data=work.vinsasg ;
 by descending moy ;

proc print data=work.vinsasg ;
 var _name_ moy ect cdv min max ;

** 6. Tri par coefficient de variation décroissant ;

proc sort  data=work.vinsasg ;
 by descending cdv ;

proc print data=work.vinsasg ;
 var _name_ moy ect cdv min max ;

** Exécution ;

run ;
```

Et dans la foulée, faisons l'AFC de ce même tableau :

```
proc corresp   data=work.vins   ;
     var BELGIQ NEDERLD RFA    ITALIE
         UK     SUISSE  USA    CANADA ;
run ;
```

vous voulez plus simple? Y'a pas !

3.5 Traitement des Variables Textuelles

3.6 Autres calculs

Chapitre 4.

SAS, programmation avancée

4.1 Bonjour chez vous avec la table !

Prenons notre fameux exemple traditionnel de "bonjour" qui demande un nom, affiche la date et l'heure et qui dit au revoir à la personne en traduisant le nom en majuscule. La version minimale qui convertit le nom en majuscule peut être

```
* bonjour1.sas ;

data bonjour ;
     input   nom  $   ;
     nomMaj  = upcase(nom) ;
     drop    nom ;
     datalines             ;
paul
;
```

mais pour inclure la date et l'heure, c'est un peu plus délicat car il faut utiliser des fonctions comme *day* et *date* et concaténer les chaines de caractères avec l'opérateur ||.

De plus, pour qu'il n'y ait rien d'autre sur la page, il faut mettre un titre vide avec `title ' ';` et activer l'option `nodate`. Notre programme crée des variables autres que celle définie dans la partie `input` de l'instruction `data` et on ne garde que la variable intéressante avec l'instruction `keep`.

D'où la deuxième version de bonjour.sas :

```
* bonjour2.sas ;

title ' ' ;
options nodate ;

data bonjour ;
 input   nom  $   ;
 lejour    = day(date()) || '/'  ;
 lemois    = month(date()) || '/' ;
 lannee    = year(date()) || ',' ;
 dateheur  = 'Le ' ||
             compress(lejour || lemois || lannee) ||
            ' au revoir, ' ;
 msg       = compbl(dateheur||compress(upcase(nom)|| '.')) ;
 keep msg ;
 * put écrit dans le log  ;
 put  msg ;
 datalines ;
paul
;
```

Passons maintenant à la traditionnelle table de multiplication. Un premier essai donne :

```
* tablemul1.sas ;

data tablemul ;
 input n ;
 do i = 1 to n ;
    msg1 = ' fois ' ;
    l    = n ;
    msg2 = ' = ' ;
    k    = i*n ;
    output ;
 end ;
 drop n i ;
 datalines ;
6
;
```

Au lieu de mettre les données dans la partie datalines de l'instruction data, il est possible de lire les données d'un fichier externe (nommé ici t:table.mul) grâce à la commande infile.

```
* tablemul2.sas ;

options nodate nocenter nonumber linesize=256 pagesize=32767 ;
filename tablemul 't:\table.mul' ;

     data tablemul ;
      infile tablemul ;
      input n ;
      title ' Table de Multiplication de ' n  ;
      do i = 1 to 10 ;
         msg1 = ' fois ' ;
         l    = n ;
         msg2 = ' = ' ;
         k    = i*n ;
         output ;
      end ;
      drop n i ;
```

Il peut être parfois intéressant de mettres les résultats dans le *Log* avec l'instruction put, de format les résultats... C'est ce que nous faisons avec le programme suivant :

```
* tablemul3.sas ;

data tablemul ;
  nombre = 10 ;
  fois   = ' fois ' ;
  egale  = ' = ' ;
  do i = 1 to 10 ;
     j = i * nombre ;

     * résultats dans le log ;
     put i 2. fois $ 6. nombre egale j 7. ;

  end ;
run ;
```

4.2 La classe, cumuls et variables automatiques

Finalement, le problème "classe" qui calcule les moyennes des élèves est assez simple, car SAS est "étudié pour".

```
* classe.sas ;
filename classe 't:\classe.dat' ;
data classe ; infile classe ;
   input  nom $  not1 not2 not3 ;
   moy =(not1+not2+not3)/3 ;
   put nom $ 1-40 not1 not2 not3 moy ;
proc print ; run ;
```

On notera qu'en sortie on force la taille de nom à 40 pour obtenir un "bon" cadrage. Dans le même genre d'idées, si l'on connait l'instruction retain, rien de plus simple que de cumuler le total des ventes dans la colonne cumul, à l'aide de la variable automatique _n_ qui correspond au numéro d'enregistrement, que l'on n'affiche pas dans la procédure print grâce à l'option noobs.

Nous en profitons au passage pour montrer l'utilisation de

data a ; set b ;

qui transfère dans l'ensemble de données (*dataset* dans la terminologie *SAS*) nommé a les données de b.

```
 data ventes ; input fruit $ nbkilo prixunit ;
   datalines ;
   oranges  3  5.5
   pommes  10  2
   oranges  5  5.5
   ;
data ventcum ; set ventes ;
   vente  = _n_ ;
   retain cumul ;
   if _n_ = 1
      then cumul = nbkilo*prixunit ;
      else cumul = cumul + nbkilo*prixunit ;
   keep vente cumul ;
```

```
proc print data=ventcum noobs ; run ;
```

4.3 Combien de lignes, combien de colonnes...

Pour connaitre le nombre de lignes, de colonnes d'un fichier, rien ne vaut quelques bonnes fonctions de derrière les fagots ! Ce sera l'occasion ici d'introduire le langage des macros de *SAS*.

SAS dispose d'un deuxième langage de programmation, complémentaires du premier, dont les structures ne sont plus nommées programmes mais macros-commandes. Ce deuxième langage, plus souple, plus interprété, utilise des variables repérées par &, des mots-clés repérés par %. Une macro commence par le mot clé macro, se termine par le mot-clé mend (contraction de *macro end*). Elle s'utilise par %nom_macro(paramètres). Par exemple, pour utiliser la macro suivante nommée combien sur le *dataset* référencé par cours.vins, il suffit d'écrire

%combien(cours.vins) ;

après avoir éventuellement chargé la macro par

%include 'combien.sas' ;

On appréciera le fait qu'il n'y ait pas à écire run ; ici. Le résultat (la phrase dans le fichier...) est mis dans le *Log*.

```
/******************************
 *                            *
 * macro SAS : combien        *
 *                            *
 ******************************/

    %macro combien(ds) ;
    %let dset  = &ds ;
    %let dsid  = %sysfunc(open(&dset)) ;
    %let nblig = %sysfunc(attrn(&dsid,NOBS)) ;
    %let nbcol = %sysfunc(attrn(&dsid,NVARS)) ;
```

```
%let rc    = %sysfunc(close(&dsid)) ;
%put dans le fichier &ds il y a
     &nblig ligne(s) et &nbcol colonne(s) ;
%mend asg ;
```

4.4 Somme en ligne, sieste en colonne

La macro suivante permet de rajouter une colonne à un *dataset*, nommée somme et qui contient la somme de toutes les valeurs numériques de la ligne. L'instruction array valr(*) _numeric_; crée un tableau nommé valr de dimension non fixée par avance, qui correspond aux valeurs numériques de la ligne. L'instruction nbcoln = dim(valr) ; affecte le nombre de ces valeurs à la variable nbcoln et une simple boucle pour suffit alors à calculer la valeur de somme.

```
%macro somlig(fid) ;
data somlig ; set &fid ;
array valr(*) _numeric_ ;
nbcol = dim(valr) ;
somme = 0 ;
do i = 1 to nbcol ;
   somme = somme + valr(i) ;
end ;
drop i nbcol ;
;
%mend somlig ;
```

Pour combiner des statistiques *de modalité* avec les données originales, par exemple pour indiquer à coté de la colonne quel pourcentage du total elle reprente, il faut procéder en général en quatre étapes :

1 - définir les données de base avec data
2 - trier les données par modalité sort...by
2 - effectuer les calculs avec means
4 - effectuer la fusion avec data option merge..by

Par contre, pour intégrer des données sans variable commune (comme un total général) il faut utiliser des techniques de set suivant le modèle

```
data  Nouveau;
```

```
if _n_ = 1 then set Stat ;
set Ancien ;
```

où Ancien est le *dataset* original, Stat le *dataset* des résultats (produit par proc means ... out = et Nouveau le *dataset* qu'on veut obtenir. On pourra consulter le chapitre 6 du *The little sas book* conseillé dans la bibliographie pour plus d'informations.

Essayons maintenant de calculer les totaux par colonne. Nous prendrons une approche radicalement différente de celle utilisée pour les sommes en ligne. On effectue le travail en trois étapes :

1 - on calcul les sommes avec proc means ...sum
2 - on transpose les résultats avec proc transpose
2 - on concatène les fichiers via date AetB ; set A B ;

Bien sûr il vaut mieux ne pas afficher les numéros d'enregistrements dans le nouveau *dataset*, soit les instructions

```
proc means data=<dataset> sum ; var _numeric_ ; output out=stat sum=  ;
proc transpose data = stat  out = tats ;
data ajout ; set stat ; id = 'somme ' ;
data global ; set <SATASET> ajout ; drop _type_ _freq_ ;
proc print data=global noobs;
```

4.5 Matrice des corrélations triangulaire

La procédure corr de *sas* générère une matrice des corrélations rectangulaire. L'habitude en statistiques veut que l'on ne donne que la matrice triangulaire inférieure.

Nous en profitons au passage pour montrer

comment tester si un *dataset* existe	avec sysfunc(open(
comment renommer une variable	avec rename
comment extraire des lignes	avec where
comment typer explicitement des variables	avec attrib
comment mettre les données dans un fichier externe	avec file

```
/*******************************************
 *                                         *
 * Matrice des correlations triangulaire   *
 *                                         *
 *******************************************/

  %macro matcor(ds) ;

  %let donnees = &ds ;
  %let dsid    = %sysfunc(open(&donnees)) ;
  %if &dsid=0 %then %do ;
      %put Le fichier de données &donnees est inaccessible !? ;
  %end ;
  %else %do ;

  %let nbcol = %sysfunc(attrn(&dsid,NVARS)) ;

  proc corr data=&ds noprint out=scorr ;
  data mcor ; set scorr ; where _type_ = 'CORR' ;
       rename  _name_ = nom ; drop _type_ ;

  data mcortr ; set mcor ;
  array cor(*) _numeric_ ;
  attrib rho length = $200 ;
  attrib v   length = $200 ;
  do i = 1 to _n_ ;
     if i = 1 then
        do  ;  w = int(cor(1)*100+0.5)/100 ;
               v = put(w,6.3) ;
        end ;
        else  do ;  w = int(cor(i)*100+0.5)/100 ;
                    t = put(w,6.3) ;
                    v = trim(v) || ' ' || t ;
        end ;
  end ;
  rho = v ;
  keep nom rho ;
```

```
data lnom ; set scorr ; where _type_ = 'CORR' ;
     rename  _name_ = ndl ; drop _type_ ;

data lnoml  ; set lnom ;
attrib noml length = $015 ;
attrib ldv  length = $200 ;
noml = trim(ndl) || '_______' ;
noml = substr(noml,1,6) ;
if _n_ = 1 then ldv = noml ;
           else ldv = trim(ldv) || ' ' || noml ;
ndl = ' '  ;
retain ldv ;
keep   ndl ldv ;

data premlig ;
     set lnoml ;
     rename ndl = nom ;
     rename ldv = rho ;
     if _n_ = &nbcol - 1 ;

data matcor ; set premlig mcortr ;

data _null_ ; set matcor ;
     file 't:matcor.sor' ;
     rhot = ' : ' || rho ;
     put nom @10 rhot $ ;

proc print data=matcor noobs ;

run ;
%put Vous pouvez consulter le fichier 'matcor.sor' ;

%end ; /* fin de si le fichier de données existe */
run ;

%mend matcor ;
```

Le *dataset* particulier _null_ évite à *sas* de sauvegarder le *dataset* utilisé, ce qui peut économiser de la mémoire et de l'espace disque...

Chapitre 5.

La galaxie SAS

Comme ont du le montrer certaines listes d'options, *SAS* semble tout savoir en statistiques. C'est normal, il est "créé pour". On trouve donc dans les manuels de *SAS*, et c'est indispensable, les pages de description des options, des paramètres, des exemples d'utilisation mais aussi les formules de calcul et les références bibliographiques des auteurs de ces formules ou des articles significatifs quant à leur compréhension.

SAS comprend aussi des systèmes de cartographies, de la gestion de fichiers à travers les réseaux, un système complet de développement d'applications, des gestionnaires de fichiers etc. etc. Par contre, ce qu'on peut lui reprocher c'est d'être parfois trop complet et d'obliger à lire 10 pages d'option avant de comprendre comment "filtrer" les affichages, d'être "lourd" quant à la syntaxe et la façon de procéder, de ne pas chercher la simplicité mais l'efficacité, et aussi son prix, car *SAS* ne s'achète pas, il se loue...

BIBLIOGRAPHIE

SAS dispose de sa propre maison d'édition, de ses propres ouvrages, sans compter les nombreux manuels "standards". Notre bibliographie est donc forcément incomplète.

R. P. Cody, J. K. Smith
Applied statistics and the sas programming language
Prentice-Hall, 1997.

L. D. Delwiche, S. J. Slaughter
The little sas book, a primer
Sas Institute Inc., 1995.

Sas Language
Sas Institute Inc., 1997.

Sas Stat User's Guide vol. 1 et 2
Sas Institute Inc., 1997.

Sas Procedures Guide
Sas Institute Inc., 1997.

Sas Macro Language Reference
Sas Institute Inc., 1997.