SAS : manuel d'introduction

et

aide-mémoire

document rédigé par J.Worms (UVSQ, julien.worms@uvsq.fr) ; version Septembre 2016

Ce document a deux objectifs :

- ↪ permettre un ***apprentissage*** relativement autonome des bases de la programmation SAS (vocabulaire, structure, syntaxe, intéractions avec l'hôte) et de sa pratique
- ↪ constituer un ***aide-mémoire*** des commandes et procédures les plus courantes, assorti d'indications détaillées pour l'utilisation du système d'aide de SAS

Ce poly est conçu à l'attention d'étudiants niveau fin de licence ou master, suivant une formation en statistique mathématique et analyse de données [1]. Il ne présuppose en aucune façon des connaissances préalables en SAS ou autre logiciel statistique, mais en statistique, si ! Il est toutefois rédigé de manière assez compacte, et le premier chapitre est fait pour être lu en continu : le "picorage" d'information ne peut raisonnablement avoir lieu qu'en seconde lecture, par souci d'efficacité et pour ne pas risquer de trouver l'ensemble indigeste. Il est évidemment vivement recommandé d'effectuer une lecture active, clavier et logiciel à disposition, pour effectuer ses propres tâtonnements et s'approprier les bases. L'orientation de ce poly est volontairement l'apprentissage de la *programmation* en SAS, l'angle le plus rentable à moyen terme [2]. La lecture du chapitre I est un préalable indispensable à tout premier TP.

1. si tant est que cela ait un sens de distinguer les deux...
2. Remarque : il existe d'autres manières d'aborder (voire d'utiliser) SAS, dans des environnements agréables au sein desquels on clique et on sélectionne par ci par là à tout va, et où tout semble plus facile (modules SAS Insight, SAS Enterprise Miner, SAS Assist, the Analyst Application...). Si de tels environnements peuvent s'avérer pratiques à l'usage, il ne s'agit pourtant que d'**enrobages** des procédures SAS, et au final l'utilisateur se verra rapidement contraint d'aller mettre le nez dans le manuel d'aide et donc dans la syntaxe du langage SAS sous-jacent.

I. Survol de SAS & Fondamentaux

1. Le systeme SAS en bref

SAS est un logiciel commercial qui permet (pour ce qui nous intéresse ici) le traitement statistique des données, en particulier issues d'autres sources (tableurs, gestionnaire de bases de données,...). Il est doté d'un langage de programmation propre qu'il est préférable d'apprendre dès ses premiers pas dans SAS. Il s'est répandu dans le monde professionnel et fait désormais parfois plus office de logiciel de gestion de données que de logiciel d'analyse statistique des données.

Au sein de l'UFR de Sciences de l'UVSQ, SAS n'est accessible que sous sa version serveur Linux à infinité de jetons, pour des raisons (essentiellement financières) qui pourront vous être détaillées en cours. Cependant une version gratuite sous Windows pour les ordinateurs personnels des étudiants est disponible (licence gratuite à domicile, à télécharger, ne durant qu'un an, et fournie par moi même). La version Windows est un peu plus conviviale, mais il y a essentiellement assez peu de différences. Ce polycopié traitera de la version Windows.

Des alternatives à SAS sont SPSS, StatGraphics, SPAD, Minitab, Splus, et surtout le logiciel libre (mais terriblement complet) R. SAS est un logiciel très présent dans le monde de l'entreprise (dans le monde de la recherche, R est d'un usage quasi hégémonique pour réaliser des analyses statistiques).

Une *session SAS* consiste en une succession d'*étapes*, qui peuvent être de 2 types :

— Les etapes DATA

— Les fenêtres **Results** et **Explorer** sont très utiles, mais d'un intérêt secondaire pour une toute première approche. La fenêtre Results permet d'avoir accès, par le biais d'une *arborescence*, à tous les résultats préalablement obtenus, qui ont été affichés dans la fenêtre Output. Elle permet également de *sauvegarder* tout ou partie de ces résultats dans un fichier (`.lst` ou fichiers graphiques s'il s'agit de graphiques), ou de les soumettre à l'*impression*. La fenêtre Explorer permet de gérer l'interaction entre SAS et le système d'exploitation sous-jacent (Unix à l'UVSQ, Windows à domicile). En fait SAS est une *surcouche* du système d'exploitation : par exemple, quand une table SAS s'appelle *don* pour SAS (pour ainsi dire son "*nom SAS*"), le fichier physique auquel elle correspond se nomme *donnees.sas7bdat* (son nom de fichier) ; il en est de même pour les *répertoires SAS*. Pour des détails sur ce sujet, voir plus loin le paragraphe "4. Système interne de fichiers SAS, notion de nom SAS".

— D'autres fenêtres pourront apparaître en cours de session : des fenêtres graphiques, un tableur (Viewtable = d'interface conviviale ressemblant à un tableur pour entrer ou visionner ses données),...

Comment se deroule concrêtement un TP sous SAS ? On lance l'application SAS et on tape des commandes dans l'éditeur, on sélectionne les commandes que l'on veut soumettre à SAS avec la souris, on clique sur le bouton de soumission ou sur F3, on regarde si l'instruction s'est bien déroulée dans la fenêtre Log/Journal, et si c'est le cas alors on visionne l'éventuel résultat affiché dans la fenêtre Results Viewer. Puis on procède comme cela par étapes successives pendant toute la session...

Passons maintenant à la description concrête des étapes DATA et PROC.

2. L'etape DATA en bref

Une étape DATA commence par le mot clef `DATA`, suivi du *nom* donné à la table SAS créée au terme de cette étape DATA, et par *un point virgule*. Après, il existe de nombreuses syntaxes possibles pour *le corps* de l'étape DATA (qu'on appelera quelquefois simplement *corps DATA*), utilisant de nombreux mots-clefs généralement spécifiques aux étapes DATA, tels que `cards, input, do...end, output, set, put, infile,` etc... Dans ce paragraphe on se contentera de cas simples et on ne fera qu'effleurer le sujet. Un peu plus loin le (volumineux) paragraphe II.1 est dédié au fonctionnement d'un concept important sous SAS, *la boucle DATA*, et aux principaux mots-clés de l'étape DATA.

Supposons par exemple qu'on veuille rentrer les données suivantes, qu'on peut qualifier de ***données simples (ou brutes)*** :

Nom	Sophie	Amir	Paul	Anne	Stéphanie	Serge
Age	12	6	42	31	35	15
Poids	34	20	70	53	64	65
Sexe	F	H	H	F	F	H

On va alors créer la table SAS nommée *don1*, qui contient 6 observations et 4 variables :

```
data don1;
input nom $ age poids sexe $;
cards;
Sophie 12 34 F
Amir 6 20 H
...
Serge 15 65 H
;
run;
```

Il y a plusieurs choses à noter dans cet exemple basique. D'abord, c'est un exemple dans lequel les données sont entrées directement, entre le mot-clef `cards` et le point virgule (qui, soit dit en passant, *doit absolument se trouver seul sur la dernière ligne*, avant le mot-clef `run;`) : cette partie-là doit se trouver en fin d'étape DATA [3]. Ensuite, parmi les 4 variables, deux d'entre elles sont alphanumériques : il est indispensable de le notifier à SAS, sous peine d'erreur, ce qui est fait avec le dollar `$` qui suit le nom des variables *nom* et *sexe* ; les autres variables, qui sont numériques, n'ont besoin de rien après l'annonce de leur nom (ce que l'on place après un nom s'appelle un *format*, ou plutôt *informat* ; voir plus loin dans ce poly). Ensuite, les observations sont "rentrées" à raison d'une par ligne.

Regardons maintenant un autre type de données couramment rencontrées, qu'on peut appeler ***données pondérées*** : cela permet de réaliser combien la structure d'une table SAS est à la fois souple et adaptée à de nombreuses situations, mais aussi pas si facile que ça à appréhender. Supposons que l'on veuille "disposer" des données du tableau de contingence suivant, par exemple [4] dans le but de mettre en œuvre un test d'homogénéité du χ^2 :

3. dans la pratique cependant,les données sont plus volumineuses et à aller chercher dans un fichier externe à l'aide d'une instruction `infile`, voire sont dans un format autre que celui de SAS.

4. à ce sujet, voir la description de la procédure `FREQ` plus loin dans ce document

	Guéri	Décédé
Homme	234	147
Femme	451	339

Il s'agit bien de données "pondérées". Mais mathématiquement parlant, il y a $n = 1171$ individus, et 2 variables.

Il y a donc deux tables SAS que l'on peut imaginer créer pour pouvoir travailler avec ces données :

* une première, conforme à la vision "mathématique" du modèle (les données dites brutes), aurait bien 1171 observations, et 2 variables, mais il n'est pas évident de trouver un moyen de la fabriquer sans avoir à écrire 1171 lignes après la commande `cards` !
* une seconde, plus naturelle mais a priori "moins commode" quand il s'agira de s'en servir, est la suivante :

```
data don2;
input sexe $ etat $ effectif;
cards;
Homme Gueri 234
Homme Decede 147
Femme Gueri 451
Femme Decede 339
;
```

(on peut noter que la programmation n'est pas forcément optimale [5])
La table SAS `don2` a cette fois 3 variables et 4 observations, *et il faudra tenir compte de cette structure lorsque l'on appliquera une procédure statistique à ces données.*

Même si l'on n'a pas vu encore comment fonctionnent les procédures SAS, regardons en une quand même à l'œuvre pour les besoins du propos. Si l'on considère la seconde table, `don2`, et que l'on veut calculer les effectifs selon les modalités de la variable `etat`, on a recours à la procédure `FREQ` qui (entre autres choses) calcule les effectifs des modalités de variables catégorielles (ou plus généralement réalise un tableau de contingence à partir de données brutes). La procédure

```
proc freq data=don2;    ← lancement de la procedure appliquee a la table SAS don2
tables etat;    ← demande de calcul d'e  ectifs pour chaque modalite de la variable etat
run;
```

va donner comme résultat que 2 individus sont guéris, et 2 sont décédés... Cette syntaxe ne serait donc utile que pour une table SAS "de données brutes". Il faut donc procéder autrement puisqu'il s'agit de données pondérées, *et que dans cette table SAS une observation ne vaut plus 1 individu !* Il faut donc plutôt soumettre

```
proc freq data=don2;
tables etat;
weight effectif;    ← demande de compter chaque observation de la table SAS
run;                      avec un poids egal a la valeur de la variable effectif
```

Cette fois, le résultat est bien 486 décédés et 685 guéris (dans l'ordre lexicographique).

On vient donc de voir deux manières d'exploiter la structure de table SAS, mais bien entendu il y en a beaucoup d'autres, et celles-ci interviennent assez rapidement en pratique : ce sont celles qui sont *issues de procédures statistiques*. On n'en parlera pas plus ici, mais ci-dessous se trouve un code qui montre l'une de ses tables, dans laquelle chaque observation correspond à différentes modalités d'une variable catégorielle [6] :

```
proc means data=don1 mean std;
var age poids;
class sexe;
output out=donout mean=ma mp std=sa sp;
proc print data=donout; run;
```

Remarque Importante :

SAS ne demande jamais l'autorisation d'écraser des tables SAS existantes (tout du moins avec les options par défaut). Par exemple, si l'on soumet une étape `DATA don;...;` et que `don` est un nom (*fileref*, cf + loin) déjà attribué à une table SAS, alors à la fin de cette étape le remplacement aura eu lieu sans autre forme de procès... Le plus souvent ça ne pose pas de problème, mais il vaut mieux le savoir.

5. on peut être moins redondant en utilisant le code `data don3; do sexe='Homme','Femme'; do etat='Guéri','Décédé'; input effectif @@; output; end; end; cards` (↪) `234 147 451 339` (↪) `; run;`

6. ces tables de sorties de procédures peuvent avoir une observation par combinaisons de variables catégorielles, ou même être un véritable fourre-tout, qui ne garde un semblant de structure que par la présence d'une variable importante, `_TYPE_`, comme par exemple la sortie `outstat=` d'une procédure PRINCOMP réalisant une ACP

3. Etape de procedure PROC

Il existe de nombreuses procédures SAS, dont beaucoup d'entres elles sont de nature statistique : `PRINT, SORT, FREQ, MEANS, REG, ANOVA, PRINCOMP, CLUSTER, LOGISTIC,` etc... Ce paragraphe décrit la structure et le vocabulaire de base de toutes les procédures (qu'il est indispensable de maîtriser pour ne pas perdre son temps plus tard), et contient un exemple illustratif, mais le détail procédure par procédure figure plus loin dans ce document (et en constitue la plus grande partie d'ailleurs) [7].

Une *étape de procédure* commence par le mot clef `PROC` suivi par le *nom* de la procédure, puis par les *options* de procédures, puis par un *point-virgule* annonçant le début du corps de la procédure. Ce *corps* se termine par le mot clef `run;` (qui ordonne l'exécution de la procédure), et il est constitué d'une succession de *instructions* (de procédures), elles-mêmes éventuellement affublées options de instructions.

Vocabulaire "fondamental"

— instructions de procedures :

Il s'agit des différentes *actions* que l'on désire voir être effectuées lors du déroulement de la procédure : réalisation de graphiques, enregistrement de résultats sous une certaine forme, indication des différents arguments de la procédure qui ne sont pas indiqués en options, et toutes sortes de tâches spécifiques à la procédure... L'ensemble des instructions forme *le corps de l'étape PROC*. Chaque instruction commence par un mot-clef, son *nom* (`var, class, model, plot, output, by, tables, format,`...), et se termine par un *point-virgule*. Entre les deux figurent d'abord les paramètres ou *arguments de l'instruction*, puis éventuellement (voire souvent) les *options* pour cette instruction (à ne pas confondre avec les options de la procédure elle-même) qui sont quelquefois séparées des arguments par une *barre oblique*.

— Options de procedures :

Il s'agit de tout ce qui est indiqué *entre le nom de la procédure et le premier point virgule venant après celui-ci*. Les options sont indiquées les unes à la suite des autres, *séparées par de simples blancs*. Le plus souvent on utilise l'option data=*nom_table_SAS*, qui sert à indiquer quelle table SAS est traîtée par la procédure.

— Options de nom de table SAS :

Il s'agit de tout ce qui est indiqué *entre parenthèses à la suite immédiate du nom d'une table SAS*; ces options de nom vont indiquer quelle partie de la table SAS en question est censée être conservée pour la tâche courante. Par exemple, `don(obs=13)` désigne la sous table constituée uniquement des 13 premières observations de la table `don`; de même, `don(firstobs=3)` est la table `don` tronquée de ses 2 premières observations. Dans un autre registre, `don(keep=`x y`)` désigne la table `don` mais dont on n'a conservé que les variables x et y, et `don(drop=`z`)` est la table `don` d'origine, hormis la variable z. Enfin,

```
don(where=(x le 3 and y = 'homme'))
```

désigne la table constituée des observations de `don` pour lesquelles la variable `x` est ≤ 3 et la variable `y` vaut 'homme'. Noter qu'il est indispensable d'utiliser les parenthèses après `where=`, sinon cela occasionne des erreurs.

Toutes ces options de nom sont combinables (dans ce cas elles doivent figurer les unes après les autres entre les 2 parenthèses, simplement séparées par des blancs). Pour avoir la liste de toutes les options de nom, voir l'onglet *SAS Products→ Base SAS→ SAS Language Reference : Dictionary→ Data Set Options* de l'aide de SAS.

Exemple : `proc means data=`*don*`(where=(`*var*`='H') obs=20) noprint;` (*a* : `means`, *b* : `data=`...`noprint;`)

`var` x y`;` } c

`output out=`*donout*`;` } d

`run;` } e

Dans cet exemple, la procedure a pour nom means (a), et a sa suite les options de cette procedure (b) : la table SAS traitee sera la table constituee des observations de la table *don* pour lesquelles la variable *var* vaut 'H', et qui sont parmi les 20 premieres. Une autre option (noprint) stipule de ne pas a cher a l'ecran le resultat de la procedure. Ensuite une premiere instruction (c) indique que seules les variables x et y seront traitees, et une seconde instruction (d) fait sauvegarder le resultat de la procedure dans une table SAS nommee *donout*. Aucune de ces instructions n'est a ublee d'option de instruction. En n la procedure est lancee (run;).

7. Autrement, le manuel de référence de toutes les procédures figure dans l'aide de SAS : par exemple, pour celles de SAS Base, dans l'onglet *SAS Products→SAS Base→SAS Procedures→Procedures*, et pour SAS/Stat dans *SAS Products→SAS/Stat User's Guide*.

4. Systeme interne de chiers SAS, notion de nom SAS, tables SAS permanentes

Quelque soit le système d'exploitation sur lequel SAS tourne, les fichiers SAS (tables, programmes, répertoires, catalogues,...) sont des fichiers encodés qui sont physiquement présents sur le disque dur. Ils sont donc contenus quelquepart dans l'arborescence et portent donc un *nom*. Considérons par exemple un fichier de données brutes dont le nom Windows est `donnees2609.dat`, situé dans le répertoire `'F:\sas\travaux_pratiques1'`. Ce nom Windows aura une signification au sein du système Windows, mais il ne sera pas celui utilisé par SAS dans les programmes, par ex, pour charger ces données (cf page 13), on n'écrira pas `infile donnees2609.dat;` mais

```
infile 'F:\sas\travaux_pratiques1\donnees2609.dat';
```

SAS peut attribuer un ***nom SAS*** à chaque fichier ou répertoire auquel il fait appel :

* dans le cas d'un fichier, on parle alors d'un ***fileref***, qui sera attribué soit à la souris par des clics en utilisant l'Explorateur SAS (voir plus bas), soit en soumettant à SAS l'instruction (globale)

  ```
  FILENAME donbrutes 'F:\sas\travaux_pratiques1\donnees2609.dat';
  ```

  Dans ce cas, `don` est le nom SAS (fileref) du fichier et c'est celui qui sera utilisable par SAS, par exemple dans `data donnees; infile donbrutes; input....;` .
* dans le cas d'un répertoire, on parle de ***libref*** (*library reference*), qui sera également attribué via l'Explorateur SAS ou par l'instruction

  ```
  LIBNAME tp1 'F:\sas\travaux_pratiques1';
  ```

Tables SAS permanentes

Les librefs et filerefs fonctionnent ensemble via le système de *nom SAS à 2 niveaux*, consistant en l'identification possible d'une table SAS par le libref du répertoire la contenant, suivi d'un point, suivi du fileref du fichier. Par exemple, si on a soumis la ligne de code (instruction globale)

```
LIBNAME tp1 'F:\sas\travaux_pratiques1';
```

alors le code `data tp1.doncopie ; set don; run;` va créer une table SAS de nom `doncopie` dans le répertoire dont le *libref* est `tp1`, table qui est une copie de la table `don` qui est située, elle, dans le répertoire dont le *libref* est `work` (répertoire qui est un peu enfoui dans votre arborescence et qui est vidé à la fin de chaque session SAS). En effet, se référer à une table SAS par `don` est une abréviation pour le nom complet SAS `work.don`.

Ce système est intéressant car il permet de créer des tables SAS à différents endroits de votre arborescence Windows, dans des répertoires pérennes, et permet également de s'abstenir de devoir réécrire à chaque fois l'adresse physique complète d'un fichier.

Il existe par défaut un certain nombre de librefs prédéfinis, qui sont par exemple `Sashelp, Work, Sasuser`. On les voit dans la sous-fenêtre "Explorateur" de le fenêtre SAS.

Utilisation de l'Explorateur SAS (SAS Explorer)

Si l'on ne veut pas utiliser les commandes `libname` et `filename`, il faut utiliser l'Explorateur (Explorer) SAS :

* pour accéder à un répertoire SAS existant, cliquer sur *Libraries/Bibliothèques*, puis cliquer sur celui-ci.
* pour créer un nouveau répertoire SAS (càd assigner un libref à un répertoire n'en possédant pas encore un), cliquer sur *Libraries*, puis cliquer droit en arrière plan et choisir *New/Nouveau*. Il est important de noter que, à la création d'un libref, on peut cocher une case *"Enable at startup"* qui créera le libref à chaque début de session SAS (on peut le faire aussi en configurant un fichier autoexec convenable...)

On a aussi toutes les informations sur un répertoire SAS (notamment son emplacement "cible") en cliquant droit sur son icone.

Maintenant, pour associer un fileref à un fichier sur son disque (contenu dans un répertoire SAS ou non) sans avoir recours à la commande `filename`, et surtout de manière permanente (à chaque session), il suffit de cliquer droit sur l'icone *File Shortcuts* (Raccourcis de Fichiers) en haut de l'arborescence de l'Explorateur : une fenêtre s'ouvre, dans laquelle on peut indiquer le fileref, parcourir l'arborescence pour indiquer l'adresse physique du fichier en question, et cocher si nécessaire la case "Assign at startup", qui évite d'avoir à le refaire à chaque session.

Pour aller plus loin : il y aurait encore beaucoup à dire au sujet du lien entre les fichiers externes et le système SAS, mais il n'en sera plus question ailleurs dans ce document. Pourtant savoir bien interagir entre SAS et Windows ou entre SAS et Unix est primordial dans le travail quotidien du statisticien régulier ou occasionnel. Il ne faudra donc pas hésiter à approfondir le sujet, avec l'aide de SAS ou d'ouvrages compétents.

II. Étape DATA, boucle implicite DATA, et notions importantes

⟶ *Ce qui a été vu jusqu'à maintenant (étape DATA, étape PROC, options de noms de tables SAS, liaison avec le système) constitue le b-a-ba de la programmation SAS, et est suffisant dans un premier temps pour pouvoir commencer à travailler en TP et tirer profit des descriptions de procédures figurant à partir de la page 26. Néanmoins il reste beaucoup à apprendre, et les paragraphes qui suivent vont évoquer quelques uns des sujets à savoir pour progresser dans l'apprentissage de SAS ; ils concernent essentiellement la programmation au sein des étapes DATA, et renvoient tous à un onglet pertinent de l'aide de SAS permettant de préciser ou développer les idées... Attention cependant : ce chapitre n'est pas rédigé en style télégraphique, et une lecture continue donnera de meilleurs résultats que des tentatives de pêche aux informations. D'un autre côté, si ce style "condensé" vous convient peu, il est alors recommandé de compléter par d'autres lectures (autres polys ou livre).*

1. Fonctionnement de la boucle DATA et principales instructions de l'etape DATA

La notion de ***boucle DATA*** est très importante à comprendre, et pour la présenter, commençons par regarder de nouveau l'exemple de la page 3, que nous reportons ici par commodité.

```
data don1;
 input nom $ age poids sexe $;
 cards;
  Sophie 12 34 F
  Amir 6 20 H
  Paul 42 70 H
  Anne 31 53 F
  Stephanie 35 64 F
  Serge 15 65 H
  ;
run;
```

Cas Simple

Dans le cas présent, on parle de boucle DATA car le *corps* de l'étape DATA (ici réduit à la seule ligne `input nom $ age poids sexe $;`) *va être exécuté autant de fois qu'il y aura d'observations à lire* après la commande `cards`, c'est-à-dire 6 fois ici. Autrement dit, SAS considère la première ligne ou ***record*** (`Sophie 12 34 F`) et va écrire dans une zone mémoire appelée PDV (*Program Data Vector*) les valeurs des variables `nom, age, poids, sexe`, comme lui intime l'instruction `input`. Comme il est arrivé à la fin de l'instruction `input` de l'étape DATA (et aussi à la fin du *record* d'ailleurs), il va *aller à la ligne* dans la zone de données (*input buffer*), et tombe sur un nouveau *record* (`Amir 6 20 H`). Cela tombe bien, car le curseur de lecture est arrivé à la fin du corps DATA (ici, à `cards`), et cela met fin à la lecture de cette première observation : comme conséquence, il y a écriture du contenu du PDV dans la table SAS `don1`, c'est-à-dire enregistrement de la première observation. Le curseur remonte alors au début du corps DATA, et une nouvelle *itération* de la boucle DATA commence, c'est-à-dire le stock de valeurs des variables dans le PDV puis leur enregistrement dans la table SAS comme nouvelle observation à la fin de l'itération. La boucle continue ainsi jusqu'à ce qu'il n'y ait plus de *record* (*i.e.* SAS est arrivé à la fin de la zone de données / *input buffer*), ce qui met fin à l'étape DATA [8].

Variations possibles

A la lecture de ce qui précède, on imagine bien qu'il existe de multiples façons d'*altérer le déroulement standard de la boucle DATA*, les facteurs étant : la façon dont on accède aux données (usage de `infile, set`), la façon dont sont présentées les données (usage des sigles `@, @@`, permettant de retarder l'enregistrement de l'observation et la fin d'itération de boucle, usage de `retain`), la possibilité d'écrire plusieurs observations dans la table au sein d'une même itération de la boucle (usage de `do` et `output`), de changer d'avis en cours de boucle (usage de `delete, abort, error, stop`), de conditionner le déroulement de l'itération (usage de `where, if, select`), le contrôle des variables à conserver (instructions ou options `keep, drop, rename`), le contrôle de la destination de la sortie de l'étape DATA (commandes `put` et `file`), le contrôle de la lecture et de la restitution des données (instructions `informat, format, length, label`), le contrôle du "déplacement" au sein de la boucle (`go to, label, link, return`).

Ce (long) paragraphe va évoquer certaines de ces commandes, pour les autres se référer à l'onglet suivant de l'aide :

SAS Products→ SAS Base→ SAS Language Dictionary→ Statements

8. on notera qu'en réalité il y a **deux curseurs de lecture** : un pour la lecture du corps DATA, et un pour la lecture de la zone de données ; pour plus de détails, voir l'onglet *SAS Products→ Base SAS→ Step-by-Step Programming...→ Getting your data into shape*

Variables supplémentaires, instruction `IF ... THEN; ... ELSE`

Pour l'instant, on n'a vu que des corps d'étape DATA très courts, réduits à une seule instruction `input`. Cependant ils sont le plus souvent constitués de plus d'une instruction, comme nous le verrons tout le long de ce chapitre.

Tout d'abord, il faut noter que toutes les variables d'une table SAS ne proviennent pas forcément d'une lecture de données dans l'*input buffer*, puisqu'on peut définir de nouvelles variables à partir de variables existantes.

Par exemple, si le corps de l'étape DATA étudié plus haut était

```
input nom $ age poids sexe $;
poidslb=poids*2.2;
if age >= 18 then statut='Majeur'; else statut='Mineur';
```

alors la table `don1` aurait deux nouvelles variables, `poidslb`, égale au poids en livres (*pounds*) des individus, et `statut`, qui indique si les individus sont majeur ou mineur. On remarquera les apostrophes dans `'Majeur'` et `'Mineur'`, indispensables pour expliciter des valeurs alphanumériques, et également le point-virgule avant le `else` et l'absence de virgule avant le `then`. Important : si on avait voulu faire plusieurs opérations après le `then` ou le `else` (par exemple une nouvelle instruction `if ... then`) , il aurait fallu encadrer ces instructions par les mots-clefs

```
do; ......; end;
```

en n'omettant pas le point virgule après le `do`. On notera que des le `end;` est strictement réservé à la cloture d'une instruction `do` (ou `select`), et qu'il n'y en a pas besoin à la fin d'un `then`.

Sigle `@@` (plus d'une observation par ligne)

Supposons maintenant que, dans la zone de données, il n'y ait pas une ligne complète réservée par observation, mais plusieurs observations sur une même ligne de la zone de données. Ceci peut être imposé par la structure du fichier de données dont on dispose et qu'on ne peut pas modifier, ou alors parce que l'on veut créer des données ne comportant qu'une ou deux variables, et qu'on ne veut pas réserver une ligne complète par observation. Une instruction `input` simple ne convient pas à la lecture de telles données, et il faut recourir à ce qu'on appelle un *line-hold specifier* : le sigle `@@` (*double trailing sign*). Celui-ci se place à la fin d'une instruction `input`, juste avant le point virgule, et sa signification est la suivante : *SAS continue de lire la ligne courante de l'input buffer même si l'on a atteint la fin du corps DATA* [9].

Par exemple,

```
data donsimple;
input x @@;
cards;
1 2 3 4
;
```

donne une table à 4 observations

x
1
2
3
4

Autre exemple, on peut écrire, pour rentrer un échantillon bivarié (plus une nouvelle variable),

```
data regress;
input X Y @@;
logY=log(Y);
cards;
3.4 12  2.9 15.3  4.8 8.4  3.7 11.2
;
```

ce qui donne

X	Y	logY
3.4	12.0	2.48491
2.9	15.3	2.72785
4.8	8.4	2.12823
3.7	11.2	2.41591

Sigle `@`, différence entre les sigles `@` et `@@`

Les sigles `@` et `@@` sont des *line-hold specifiers*, c'est à dire des sigles qui *empêchent SAS de mettre fin à la lecture de l'observation courante dès que la fin d'une instruction* `input` *survient* : cette possibilité de contrôle est capitale pour ne pas se contenter de corps d'étapes DATA trop basiques. Supposons par exemple que l'on dispose de données indiquant (dans cet ordre) le sexe, le prénom, le nom de personnes, le nom de jeune fille dans le cas des femmes mariées, et leur âge. La zone de données est la suivante :

```
F rym worms ramdani 31
H julien worms 30
F baya hamdani chaouche 47
```

Que faire pour pouvoir lire ces données ? Voilà la programmation adéquate du corps DATA :

```
data don;
input sexe $ prenom $  nom $ @;
if sexe='F' then input nomjf $ @;
input age;
```

Le résultat est alors le suivant :

Obs	sexe	prenom	nom	nomjf	age
1	F	rym	worms	ramdani	31
2	H	julien	worms		30
3	F	baya	hamdani	chaouche	47

9. et tant que l'on ne rencontre pas de instruction `input` ne se terminant pas par `@@`.

On remarque que l'observation 2 a une valeur manquante pour la variable `nomjf`, et que l'on a bien lu la variable `age`. Cela n'aurait pas été le cas si l'on avait omis le sigle `@` aprés `input nomjf`, ou plutôt cela aurait occasioné une erreur (`Lost Card`). Il y aurait aussi eu une erreur bien sûr si on avait omis le sigle `@` à la fin de la première commande `input` (voir plus loin pour avoir plus d'explications sur ce procédé de retenue de passage à la ligne).

Différence entre `@` *et* `@@`

On a vu que le rôle du sigle `@` est d'empêcher le retour a la ligne dans la zone de donnees lorsque l'on atteint la n d'une instruction `input` (on dit qu'il y a eu *maintien de la ligne courante*). Mais lorsque l'iteration courante se termine, c'est-a-dire que SAS est arrive a la n du corps DATA, il y a egalement retour a la ligne automatique dans cet *input buffer* a n de commencer a lire le nouveau *record* et debuter la nouvelle iteration. *Le sigle* `@` *n'empèche pas cela.* Par exemple si la premiere ligne de la zone de donnees est

```
F rym worms ramdani 31 mcf
```

et que le corps reste inchange, alors la donnee `mcf` ne sera pas lue par SAS (et n'occasionera pas d'erreur d'ailleurs) car SAS sera alle a la ligne juste avant dans l'*input buffer*, puisque la n du corps DATA aura ete atteinte. Il en est de même si l'on rajoute un `@` dans la derniere ligne du corps DATA : `input age $ @;` .

Par contre, l'utilite de `@@` au bout d'une instruction `input` est d'empecher le retour a la ligne dans la zone de donnees ayant normalement lieu *soit* pour cause de n de corps DATA (n d'iteration de la boucle DATA), *soit* pour cause de n de instruction `input` : pour être plus precis, *le sigle* `@` *maintient la ligne courante tant que le corps DATA n'est pas terminé ou que l'on n'a pas atteint de instruction* `input` *ne se terminant pas par* `@`*, tandis que le sigle* `@@` *maintient la ligne courante tant que l'on n'a pas atteint de instruction* `input` *ne se terminant pas par* `@@`. Mais dans les deux cas, on passe quand même a la ligne si la ligne courante est elle-même nie (en tout etat de cause !).

instruction `DO; ...; END;` et mot-clef `OUTPUT`

On a vu plus haut une première utilité de l'instruction `DO`, pour délimiter un ensemble de instructions, par exemple au sein d'une structure `if ... then...;` . Maintenant on peut utiliser l'instruction `DO;` de manière plus "conventionnelle" dans le but de faire réaliser une même action plusieurs fois dans la même itération de boucle DATA, à un paramètre près. Par exemple, pour créer un échantillon de variables gaussiennes centrées réduites, on écrira

```
data gauss;      do i=1 to 5;   x=rand('normal'); output;    end;
```

On a utilisé ici une instruction importante qui est `OUTPUT;` : elle a pour effet de *vider le contenu du PDV et d'enregistrer ainsi l'observation courante* au stade où elle en est, dans la table SAS mais cela n'a PAS pour effet de finir l'itération de la boucle et revenir au début du corps DATA [10]). On remarque aussi qu'il s'agit d'une étape DATA sans zone de données ni instruction `input`, et que le résultat est une table SAS de 5 observations et 2 variables, `x` et `i`, car SAS conserve (logiquement en fait !) la variable d'itération `i` de la boucle, comme une variable ordinaire. Pour ne pas la conserver, il faut utiliser une instruction `DROP`

```
data gauss;  do i=1 to 5; x=rand('normal'); output; end;  drop i;
```

ou bien une option de nom `DROP`

```
data gauss(drop=i);  do i=1 to 5; x=rand('normal'); output; end;
```

les deux syntaxes n'étant pas les mêmes.

Autre exemple : pour voir une utilisation un peu plus évoluée des boucles `DO` en combinaison avec le sigle `@@`, on a la programmation plus efficace suivante de l'étape DATA vue en paragraphe I-2 :

```
data don3;
 do sexe='Homme','Femme';
  do etat='Gueri','Decede';
   input effectif @@;
   output;
  end;
 end;
cards;
234 147 451 339
;
```

Il existe d'autres instructions utilisant le mot-clef `DO`, que sont `DO ... WHILE` et `DO ... UNTIL`; voir le paragraphe correspondant page 16.

10. pour s'en convaincre, il n'y a qu'à regarder ce que donne le code suivant : `data gauss;  do i=1 to 5;  x=rand('normal'); output; y=x+1; end;  proc print; run;` On constatera le phénomène "important" de **retenue automatique** des valeurs des variables, d'une itération de la boucle à la suivante...

instruction SELECT

l'instruction `select` a comme but d'éviter des imbrications interminables de `IF ... THEN`. Pour le coup, des exemples (même idiots) valent mieux qu'un long discours :

```
data bidon;
 input x c @@;
 select;
  when (x=1 and c=1) y=x*10;
  when (x=2 or c=0) ;
  when (3 LE x) y=x*100;
  when (c=.) y=.;
  otherwise y=1;
 end;
cards;
1 1 2 1 3 0 7 0 7 1 0 1 1 .
;
```

ou encore

```
data couleurs;
 length coul $8;
 input c $ @@;
 select(c);
  when ('be') coul='bleu';
  when ('bc') coul='blanc';
  when ('r') coul='rouge';
  when ('v') coul='vert';
  otherwise;
 end;
cards;
v c be bc r bc f
;
```

instructions LENGTH et LABEL

Dans l'exemple `couleurs` proposé ci-dessus, a été utilisée une instruction `length`, qui a indiqué à SAS la taille en caractères (ici 8) que l'on voulait donner à la variable littérale `coul`. En effet, cette variable qui est définie directement et non pas par une lecture de données (instruction `input`), va se voir attribuer une longueur égale à celle de la *première* modalité que va rencontrer SAS lors de la compilation, c'est-à-dire `bleu`, donc une longueur de 4 : il y aura alors troncature automatique des modalités plus longues, telle `rouge` ou `blanc`.

Par ailleurs, les variables littérales se voient attribuer par défaut une longueur de 8 caractères : toute modalité, même lue dans l'*input buffer*, de longueur supérieure, va être tronquée à 8 caractères. C'est exactement ce qui arrive avec la table `don1` du début de ce chapitre : la 4ème observation voit sa valeur de `nom` tronquée à "`Stéphani`" (voir plus bas ce qu'il faut faire pour éviter cela).

C'est là toute l'utilité de la commande `length`, qui force à attribuer telle ou telle longueur aux variables alphanumériques. Les longueurs doivent toutefois être précédées d'un dollar, sans signe =, et la déclaration de longueurs va avoir un effet sur l'ordre dans lequel les variables vont figurer dans la table SAS [11].

Pour remédier au problème dans la table `don1`, il suffit donc d'ajouter la commande `length nom $9;` en début de corps DATA (avant l'instruction `input` en tout cas) ; toutefois des complications peuvent survenir si certains noms comportent des blancs, et d'autres solutions doivent être trouvées (voir paragraphe suivant sur le *column-input*).

La commande `label` est une commande d'étape DATA qui permet de donner des "surnoms" aux variables, qui seront utilisées dans les procédures et produiront une meilleure présentation. Tout comme la longueur `length`, un *label* est ce qu'on appelle un *attribut de variable*. Par exemple ci-dessus, la variable `coul` peut être affublée du *label* "Couleur de l'embout" en utilisant la commande

```
label coul="Couleur de l'embout";
```

dans le corps de l'étape DATA. Ceci aura comme conséquence que ce sera ce surnom qui apparaitra dans les sorties de procédures, au lieu du vrai nom `coul`, si on le demande : par exemple dans la procédure `print` avec option `label`

```
proc print data=couleurs label; run;
```

On remarquera que dans la chaîne de caractères `"Couleur de l'embout"`, l'apostrophe est acceptée car la chaîne est définie avec des guillemets (voir le paragraphe III-1 de conseils à ce sujet).

Lecture des données et types d'*input*

Il y a plusieurs façons de présenter les données dans l'*input buffer* (ou le fichier externe, cf plus bas), et par conséquents plusieurs façons de les lire dans une étape DATA : considérons par exemple des données semblables à celles proposées comme illustration en début de document

```
data don1;                                  |  data don1;
 length nom $14;                            |   length nom $14;
 input nom $ age poids sexe $;              |   input nom $ @15 age @18 poids @21 sexe $;
cards;                                      |  cards;
Anne-Sophie 12 34 F                         |  Anne-Sophie   12 34 F
Amir 6 20 H                                 |  Amir          6  20 H
Serge 15 65 H                               |  Serge         15 65 H
;                                           |  ;
```

On parle à gauche de *list input*, et à droite de *column input* : dans le premier cas les blancs sont considérés comme des séparateurs entre modalités de variables distinctes, et dans le second on déclare à l'avance où (*i.e.*

11. on peut d'ailleurs l'observer sur la table `couleurs`, où la variable `coul` est située avant `c`, ce qui est un peu surprenant au premier abord ; il faut utiliser la procédure `datasets` pour modifier cet ordre, soit dit en passant.

dans quelles colonnes) les modalités sont censées être lues, variable par variable.

Le grand avantage de la première méthode, est la simplicité mais celui de la seconde méthode est qu'il suffit de ne rien mettre dans les colonnes appropriées pour indiquer à SAS qu'il y a là une valeur manquante [12] (en outre, la présentation, bien que plus rigide, est plus nette à lire et à vérifier, et il est fréquant que les données réelles soient présentées dans ce format). Le sigle `@` suivi d'un numéro est en fait une commande qui indique à SAS où placer le curseur de lecture pendant la lecture de l'observation courante dans l'*input buffer*. On remarquera enfin la commande `length $14`, indispensable ici sous peine que la modalité `Anne-Sophie` soit tronquée à 8 caractères (même en *list input*).

D'autres possibilités de syntaxe sont les suivantes : (à droite *column input*, et à gauche une variante utilisant un *informat*, *cf* plus bas)

```
data don1;
 input nom $12. age poids sexe $;
cards;
Anne-Sophie 12 34 F
Amir        6 20 H
Serge       15 65 H
;
```

```
data don1;
 input nom $1-14 age 15-16 poids 18-19 sexe $;
cards;
Anne-Sophie   12 34 F
Amir          6  20 H
Serge         15 65 H
;
```

Maintenant, nous allons parler de la gestion des espaces dans la lecture de données. Supposons que dans les deux codes ci-dessus, il y a dans la zone de données `Anne Sophie` (sans tiret) au lieu de `Anne-Sophie`. Cela pose-t-il un problème ? Ici non, car nous utilisons du *column-imput*. Mais si l'instruction de lecture avait été en *list input* (c'est-à-dire `input nom $ age poids sexe $;`) alors cela aurait occasionné une erreur, car SAS aurait lu `Anne` pour la valeur de `nom` de la première observation, puis `Sophie` pour la valeur de `age`, ce qui n'est pas possible car `age` est une variable numérique. Pour pallier à cela, il faut utiliser le sigle `&`, qui indique à SAS que les données dans la zone de données sont censées être espacées de *deux* espaces ("blancs") et non plus d'un seul, et qu'il faut donc attendre deux espaces consécutifs avant de charger une valeur dans une variable. Par exemple

```
data don2;
 length nom $20;
 input nom & $ date;
cards;
Francois Mitterand  1981
Jacques Chirac  1995
Nicolas Sarkozy  2007
;
```

Un autre sigle potentiellement utile est le sigle `#`, qui sert à lire une observation qui s'étend sur plus d'une ligne (contrairement au sigle `@@` qui sert à gérer plusieurs observations sur une même ligne). Supposons que la zone de données soit

```
Anne Sophie
12 34 F
Amir
6 20 H
Serge
15 65 H
```

Pour pouvoir gérer une telle structure de données, voilà le code à utiliser :

```
data don1;   input #1 nom $  #2 age poids sexe $;
```

En effet, `#1` indique qu'il faut se placer au début de la "première" ligne pour lire les données, et `#2` qu'il faut passer à la "deuxième" ligne. Une alternative est d'utiliser l'instruction de *saut de ligne* (dans la zone de données) qu'est la barre oblique `/`

```
data don1;   input nom $  /  age poids sexe $;
```

Il y a beaucoup d'autres commandes de ce style qui existent et qui, couplées à des instructions `IF` par exemple, donnent une grande souplesse dans la lecture des fichiers de données que l'on est obligé d'utiliser tels quels. Voir l'aide de SAS pour poursuivre sur ce sujet (*SAS Products→ Base SAS→ SAS Language Concepts→ DATA Step Concepts→ Reading Raw Data→ Reading Raw Data with the INPUT Statement*).

12. voir page 17 pour plus de détails sur la gestion des valeurs manquantes dans l'étape DATA

Variables numériques et alphanumériques, formats et informats

Il existe très peu de types de variables sous SAS : les variables numériques, alphanumériques (littérales), et les dates (en réalité, dans SAS une date n'est autre qu'un entier qui correspond au nombre de jours séparant la date en question et une certaine année de référence, le 1er janvier 1960=année 0 ; mais nous ne nous étendrons pas sur le sujet des dates dans ce document). Sans rentrer trop dans les détails, les variables ont de nombreux attributs, dont deux sont parfois importants en pratique : le ***format*** et l'***informat***. En fait, les attributs tels la longueur et le *label* (qu'on a vu plus haut) sont des attributs "intrinsèques" d'une variable, déclarés pendant l'étape DATA qui a créé la table SAS, tandis que les formats sont des attributs qui sont susceptibles de changer fréquemment, et peuvent être attribués de manière permanente ou temporaire.

> *Un **format** est un code qui indique comment doivent être **affichées** les modalités d'une variable. Un **informat** est un code qui indique comment doivent être **lues** les modalités d'une variable dans le fichier de données brutes.* Un *informat* est donc un format de lecture, tandis qu'un *format* est un format d'écriture. En outre, dès à présent, il est important de réaliser la différence entre un *label*, manière d'afficher le *nom* d'une variable, et un *format*, manière d'afficher les *modalités* d'une variable...

Les codes à utiliser pour créer ou manipuler des formats et informats sont assez similaires, à la différence que les informats sont souvent déjà disponibles par défaut (et déclarés au sein d'une instruction `input` d'une étape DATA), tandis que les formats sont les plus souvent créés par l'utilisateur. On crée un format à l'aide de la procédure `FORMAT`, puis on y fait appel au sein des procédures suivantes (`print`, `freq`, ...). Cependant, si le format est associé à une variable dès la création de la table et de la variable (via une instruction `format` *nomvar format;* de l'étape DATA), alors le format sera automatiquement appelé et utilisé quand cela s'avèrera possible.

Exemple de création de format

Si nos observations sont des individus ayant eu un accident de voiture, et qu'une variable `accident` a été codée 0 ou 1 selon que la personne est indemne ou a été blessée, on peut avoir envie de voir s'afficher autre chose que des 0 et des 1 pendant notre étude. On utilisera donc un format pour la variable `accident`, en utilisant la syntaxe suivante :

```
proc format;
value accid  0='Indemne' 1='Blessé';
run;
```

```
data don;
 length nom $9;
 input nom $ age accid;
 format accid accid. ;
cards;
 Dupont    34 0
 Durand    57 1
 Duschmoll 21 1
;
```

Avec une telle déclaration de format au sein même de l'étape DATA initiale, partout où devraient s'afficher les modalités 0 et 1 de la variable accident

et celles de `age` : "`$12.0`", "`$6.0`", et "`$15.0`". Les formats les plus simples sont les suivants :

- $w.d$: format numérique dans lequel w est le nombre de caractères *tout compris* (càd incluant les éventuels virgule et décimales si $d \neq 0$, signe $-$) et d est le nombre de décimales. Un exemple vient d'en être donné.
- `$`$w.$: format littéral de base, où w est le nombre de caractères [13].
- Tous les formats s'écrivent sous la forme `<$>`*nomformat*`<w>.<d>`, càd incluent un dollar s'ils sont littéraux, portent un nom, une taille (`w`) (*en nombre de colonnes tout compris*), un point (**absolument indispensable**, notamment pour les formats définis par l'utilisateur), et éventuellement un nombre de décimales dans le cas d'un format numérique. Des exemples courants sont les suivants : `$QUOTE`$w.$ (affiche entre guillemets), `DOLLAR`$w.d$ ou `EURO`$w.d$ (ajoute un dollar ou un `E` en tête), `DDMMYYYY10.` (un des innombrables formats de dates, ici de la forme 26/09/2005), `E`$w.d$ (numérique scientifique), `PERCENT`$w.d$ (en pourcentages), `ROMAN`$w.d$ (en chiffres romains), `NUMX`$w.d$ (numérique qui remplace le point anglo-saxon de la décimale par la virgule),... Voir l'onglet suivant de l'aide pour une description courte et le lien aux descriptions longues de tous les formats de SAS Base :

  SAS Products→ SAS Base→ SAS Language Dictionary→ Dictionary of... → Formats→ Formats by Category.

Lecture de fichiers externes, commande `INFILE`

Jusqu'à maintenant, on a toujours écrit les données dans un champs `cards` cloturant l'étape DATA : *toutefois, dans la plupart des cas, les données sont disponibles dans un fichier externe.*

— lorsque les données proviennent d'Excel ou d'un tableur, il faut avoir recours à la procédure `PROC IMPORT` ou bien à l'*Import Wizard* (menu *Fichier→Importer des données*)

— lorsque les données sont sous format texte, avec ou sans séparateur (extensions `.txt` ou `.csv`), il faut utiliser la commande `INFILE` d'une étape DATA.

Commençons par parler du deuxième cas. Les données ne sont plus contenues entre le mot-clef `cards` et le point-virgule final de l'étape data, elles sont contenues dans un fichier externe qu'il va falloir désigner soit par son adresse physique précise, entre apostrophes, soit suivie d'un *fileref* si le fichier en question en a été affublé (*cf* page 6). Ce nom de fichier doit être donné en paramètre de la commande `INFILE` de l'étape data, pour dire à SAS où aller chercher le fichier. Par exemple, dans l'étape `couleurs` donnée plus haut (en page 10), si un fichier texte contenant

```
v c be bc r bc f
```

a comme adresse physique `F:\sas\tp1\ficcouleurs.txt`, ou comme *fileref* `ficcoul`, alors l'une ou l'autre des commandes suivantes suffit à remplacer le bloc `cards` dans l'étape DATA (elle est à placer en *début* de cette étape d'ailleurs, contrairement à `cards`) :

```
infile 'F:\sas\tp1\ficcouleurs.txt';    ou bien    infile ficcoul;
```

Si maintenant le fichier est au format CSV (*comma separated values*), alors il faut rajouter des options dans la commande `infile`. Un exemple : le fichier *données1.txt* est constitué des lignes suivantes

```
"Sophie",12,34,F
"Amir",,20,H
"Paul",42,70,
```

On remarque qu'ici les données sont séparées par des virgules, certaines modalités alphanumériques sont munies de guillemets, et il y a des données manquantes (âge d'Amir et sexe de Paul). Si ce fichier *données1.txt* a comme *Fileref* `donnees1`, alors la syntaxe à utiliser est

```
data don1;  infile donnees1 dlm=',' dsd missover;  input nom $ age poids sexe $;
```

l'instruction `infile donnees1` indique à SAS qu'il faut aller lire les données dans le fichier dont le nom SAS (*Fileref*) est `donnees1`. L'option `dlm=','` indique que, dans ce fichier, les différentes données sont *délimitées* par une virgule (`dlm` est le diminutif de `delimiter`). L'option `dsd` (*delimiter-sensitive data*) indique qu'il faut interpréter des délimiteurs successifs (ici, des virgules qui se suivent) comme signe qu'il y a valeur manquante ; en outre, la présence de cette option fait que les guillemets entourant les noms seront supprimées (c'est-à-dire que la valeur de la variable `nom` pour la première observation est *Sophie* et non *"Sophie"*). Enfin, l'option `missover` fait comprendre à SAS que des virgules en fin de ligne signifient des valeurs manquantes. D'autres options pertinentes existent, une couramment utilisée étant `firstobs=2`, qui indique que la première ligne du fichier ne contient pas de données.

13. Mais attention, il est très important de distinguer l'effet d'un *informat* (indication du nombre **exact** de colonnes réservées à la lecture d'une donnée) de l'effet d'une commande `length` (indication du nombre **maximum** de caractères que peut comporter une donnée littérale). Par exemple, si l'on indique le format `$11.` pour la variable `nom`, que les données sont celles du code de gauche (non-adaptées à un *column-input*) et que l'on *omet l'instruction* `length $14` (sans point, soit dit en passant : ce n'est pas un format), alors pour la deuxième observation "`amir 6 20 H`" cela occasionne un erreur car SAS attribue la valeur (11 carac. de long) "`amir 6 20 H`" à `nom`, puis se retrouve en fin de record : il n'a plus de grain à moudre, ne trouve rien à attribuer à la variable `age`, va à la ligne histoire de peut-être trouver la suite ("*SAS went to a new line...*") mais trouve "`Serge`", qui n'est pas numérique, et là l'étape DATA plante vraiment. On voit donc là tout l'intérêt du style *column input*...

Si maintenant les données sont issues d'un tableur (ex. Excel) ou issues d'un gestionnaire de bases de données, alors une solution est d'utiliser la procédure d'***importation*** `proc IMPORT` (ou recours à l'*Import Wizard*, disponible dans l'onglet *Fichier* → *Importer Données*), décrite en page 32 . Une autre possibilité est de convertir d'abord le fichier Excel en fichier CSV. Quoi qu'il en soit, ce sujet (oh combien redondant et important) ne sera pas vraiment détaillé dans ce poly et je renvoie à d'autres sources.

L'aide pour la lecture des données "externes" est située au noeud :

Base SAS → *SAS Language Dictionary* → *Dictionary of Language Elements* → *Statements* → *INFILE Statement*

L'indispensable instruction `SET` (premiers pas)

La commande `SET` est une commande *très importante* de l'étape DATA et il est grand temps de la présenter. Elle permet de créer une nouvelle table SAS à partir d'une table SAS existante, *ce qui est une tâche extrêmement fréquente en pratique*. Couplée avec d'autre mots-clef de l'étape DATA, l'instruction `SET` permet de nombreuses finesses.

Commençons par un exemple très simple, en supposant que `don` soit une table SAS que le système connait déjà, et qui comprend au moins une variable numérique de nom `x`. Alors le code suivant va créer une deuxième table, `donbis`, égale en tout point à `don`, mais avec une variable supplémentaire `x2` qui vaut le carré de `x` :

```
data donbis;
 SET don;
 x2=x**2;
run;
```

On notera que si on avait écrit `data don; set don;` au lieu de `data donbis; set don;`, alors il n'y aurait aucune nouvelle table créée, mais que `don` aurait été "écrasée" par sa nouvelle version contenant la variable `x2`. Autre exemple : supposons que la table SAS `repert` contienne les variables `nom, categ, age, sexe`. Alors l'étape DATA suivante va créer une nouvelle table SAS, à partir de `repert` [14] :

```
data repert2;
 SET repert;
 length statut $12;
 select(categ);
  when (1,2) do; statut='bon'; score=age**2; end;
  when (3)   do; statut='sous-reserve'; score=age; end;
  otherwise  do; statut='mauvais'; score=0; end;
 end;
```

La table `repert2` va contenir toutes les variables et observations de la table `repert`, plus les variables `statut` et `score` qui viennent d'être définies. Concrètement, et pour faire simple, lorsque la commande `SET repert;` est soumise, SAS va regarder successivement [15] chacune des observations de la table `repert` et effectuer ce qui est indiqué dans le corps DATA ci-dessus pour chacune de ces observations. En quelque sorte, le retour à la ligne dans l'*input buffer* dont il était question en début de chapitre, revient maintenant à passer à l'observation suivante de la table SAS `repert`.

Encore un exemple : le code suivant crée la table `repert2` qui conserve, parmi les 15 premières observations de la table `repert`, celles qui sont des garçons de moins de 10 ans, et supprime les variables `categ` et `sexe`

```
data repert2; set repert(where=(sexe='H' & age<10)); drop categ sexe; run;
```

La commande `set` utilisée seule accepte une poignée d'options, dont une est souvent utile : `nobs=`. Elle permet de pouvoir faire appel au nombre d'observations de la table lue dans la commande `set`. Par exemple

```
data don; set don nobs=n; nombre=n; prop=_N_/n; run;
```

va ajouter à la table `don` une variable `nombre` qui vaut constamment le nombre d'observations n de la table, et une variable `prop` qui vaut i/n où i est le numéro de l'observation. Par contre, si `nobs=` est utilisée dans l'exemple qui précédait, alors le nombre d'observations sera celui de la table `repert2`, et non celui de la sous-partie des observations "garçons de moins de 10 ans" de la table `repert2` (ce qui peut paraître curieux a priori). On remarquera aussi que, dans l'expression `nobs=n`, `n` n'est pas une variable, ce qui explique pourquoi on doit rajouter `nombre=n`, si l'on veut créer une telle variable en tant que telle.

L'option `end=` fonctionne de la même façon, en créant une "variable" qui vaut toujours 0 sauf pour la dernière observation (valant 1 alors) : ceci permet de réaliser certaines actions uniquement lorsque la fin du fichier de données est atteint. Les options `key=` et `point=` sont dignes d'intérêt également, mais sont d'utilisation plus subtile et nous renvoyons à l'aide de SAS pour les détails.

Pour en terminer avec cette introduction à l'instruction `set`, citons une utilisation très fréquente, consistant à concatener des tables en les listant à la suite d'une même instruction `set` : par exemple,

14. il s'agit d'un exemple jouet, car il y a beaucoup de manières de procéder, avec les Formats par exemple

```
data table1; input nom $ x1 x2;
cards;
Worms    2 3
Gautier  4 5
Jones    6 4
;
```

```
data table2; input nom $ x1 y;
cards;
Silver  4 10.2
Kast    2 12.8
;
```

```
data concat;  set table1 table2;
```

va donner comme résultat

nom	x1	x2	y
Worms	2	3	·
Gautier	4	5	·
Jones	6	4	·
Silver	4	·	10.2
Kast	2	·	12.8

Il y aurait encore beaucoup de choses à dire sur la commande `set`, et sur les autres façons de combiner ou mettre à jour des tables SAS, aussi nous renvoyons à un paragraphe en fin de document sur le sujet (commandes `set` successives, instructions `merge` et `update`... *cf* page 61).

Mots-clefs `KEEP, DROP, RENAME`

Les mots-clefs `keep, drop` et `rename` servent à manipuler les variables que l'on souhaite garder, retirer, ou bien renommer. Ils sont utilisés

— soit comme *instructions d'étape data*, par exemple

```
data don;   do i=1 to 5; x=rand('normal',0,1); output; end;   drop i;   run;
```

(la table `don` en sortie n'aura qu'une variable, `x`, car `i` aura été supprimée à chaque passage de boucle donc pour chaque observation)

— soit comme *options de nom de table SAS*, par exemple

```
data don(drop=i);   do i=1 to 5; x=rand('normal',0,1); output; end;   run;
```

(ce qui aboutit au même résultat)

Un autre exemple : si dans la table `repert2` créée ci-dessus on ne veut conserver que les variables `nom, statut, score`, alors on peut soit rajouter en fin de corps DATA l'instruction `keep nom statut score;` (ou bien l'instruction `drop age sexe categ`), soit remplacer la déclaration d'étape DATA `data repert2;` par `data repert2(keep=nom statut score);` (sans virgules ni parenthèses après le signe égal, *même en présence d'autres options de nom SAS*) ou bien par `data repert2(drop=age sexe categ);`.

Si maintenant on veut garder toutes les variables dans la table `repert2`, mais qu'on ne veut considérer que `nom, statut`, et `score` dans une certaine procédure (par exemple `REG`), alors on utilisera le mot-clef `keep` comme option de nom de table SAS dans la déclaration suivante de la procédure `REG` :

```
proc reg   data=repert2(keep=nom statut score);  .....  run;
```

La commande `rename` s'utilise de manière semblable dans une étape DATA, la syntaxe étant

`rename` *nom1*`=`*nouveau_nom1* *nom2*`=`*nouveau_nom2* `...`

Mot-clef `WHERE` et options de nom de tables SAS

On a vu ci-dessus comment sélectionner une partie des *variables* d'une table SAS : voyons maintenant comment ne sélectionner que certaines *observations* d'une table SAS. Si par exemple on ne veut afficher que les observations femmes et majeures de la table `don1`, alors on utilisera la syntaxe

```
proc print data=don1(where=(sexe='F' and age GE 18)); run;
```

Dans la ligne de code ci-dessus, ce qu'on place entre paranthèses après le nom `don1`, ce sont des options de nom de table SAS, dont il a déjà été question, mais qui est un concept important du langage SAS permettant une grande flexibilité et évitant de créer des doublons inutiles de données... Les autres options de nom utiles que sont `keep`, `drop`, `firstobs`$= k$ et `obs`$= k$, ont déjà été évoquées au paragraphe précédent et en page 5.

Si plusieurs options de nom sont utilisées, celles-ci doivent n'être séparées que par de *simples espaces*; en outre, comme le montre l'exemple ci-dessus, si l'on veut mettre plusieurs conditions impliquant plusieurs variables dans l'option `where=`, alors il faut les séparer par des séparateurs logiques `and`. Si l'on veut par contre demander si telle variable vaut une valeur parmi un petit nombre de valeurs précises, il faut avoir recours au mot-clef `IN` et à ce qu'on appelle une *liste de modalités*, par exemple [16]

```
proc means data=don1(where=(nom in ('Amir','Serge'))); run;
proc print data=don1(where=(age in (17,18,19,20))); run;
```

Pour résumer : → sélectionner une partie des *variables* d'une table = option de nom `DROP` ou `KEEP`
→ sélectionner une partie des *observations* d'une table = option de nom `WHERE`

16. notez dans cet exemple la présence des différentes parenthèses : aucune d'entre elles ne peut être omise...

Variables automatiques _N_

Dans ce paragraphe nous évoquons un objet assez curieux, et pourtant très utile, qui est la variable automatique _N_, disponible dans toute étape DATA : cette variable est automatiquement créée et vaut *le nombre de fois que la boucle DATA a été entamée* (c'est-à-dire le plus souvent le numéro de l'observation). Par exemple, si l'on veut créer une nouvelle table SAS `donbis` qui ne contienne que les observations multiples de 3 de la table `don1`, alors on écrira

```
data don1bis;  set don1;  if mod(_N_,3) NE 0 then delete; else numobs=_N_;  run;
```

Oon remarquera que le numéro de l'observation dans la table d'origine `don` a été enregistré au sein d'une nouvelle variable qu'on a notée `numobs`, mais que la variable _N_ n'apparait jamais lorsque l'on veut afficher les données, et en fait elle n'existe plus dès que l'étape DATA s'achève.

Il existe une autre variable automatique intéressante, _ERROR_, qui vaut 1 dès qu'une erreur survient (division par 0, types de variables incompatibles,...) ; voir la documentation à son sujet.

Boucles `DO WHILE` et `DO UNTIL`, instruction `GOTO`

Pour terminer ce très long chapitre sur les principales commandes des étapes DATA, parlons maintenant de quelques instructions de programmation classiques que `goto` et les boucles `do while` et `do until`.

Une instruction `goto` permet de casser le flot habituel de la lecture du corps DATA, pour aller à un endroit particulier du code communément appelé *label*. Par exemple,

```
if x <= 0 then goto bip;
y=log(x);
bip error("Logarithme d'un nombre <=0");
```

équivaut à

```
if x > 0 then y=log(x);
else error("Logarithme d'un nombre <=0");
```

Ici le nom du *label* est `bip`, et pour le définir on lui ajoute un ":" à la fin de son nom. Il faut cependant prendre garde à ne pas placer ce *label* AVANT l'instruction `goto` (cela peut être problématique), et à préférer le plus possible d'autres moyens que l'utilisation de telles instructions `goto` ; ici dans l'exemple précédent, l'usage du `if then else` est de loin préférable !

Les boucles `do while ... end` (faire tant que) et `do until ... end` (faire jusqu'à ce que) sont ce qu'on appelle des *boucles conditionnelles*. Par exemple

```
data don; n=1; do while(n<=3); x=rand("normal",1,2); if x>4 then do; output; n=n+1; end; end;
```

va simuler 3 gaussiennes $\mathcal{N}(1,4)$ conditionnées à être supérieures à 4 (on sort de la boucle quand `n` vaut 4, ce qui advient après 3 simulations convenables).

Attention ! Si on avait remplacé `do while(n<=3)` par `do until(n>3)`, on aurait abouti à la simulation de 3 ou 4 variables, car dans une boucle `do until` le test a lieu en FIN de boucle (càd après que ce que contient la boucle a été exécuté) alors que dans une boucle `do while` le test a lieu en DEBUT de boucle. Voir l'aide de SAS pour d'autres exemples.

Pour conclure, évoquons les instructions `ABORT` et `STOP`, qui ont comme effet de terminer immédiatement l'étape DATA en cours pour passer à la prochaine étape (DATA ou PROC) du code soumis ; toutefois, la table SAS en l'état est sauvegardée sur le disque.

2. Autres aspects importants de la programmation SAS

Valeurs manquantes

La présence de valeurs manquantes constitue un sujet délicat en statistique, et SAS ne pourra pas pallier à ce problème. Tout ce qu'il pourra faire, c'est essayer de simplifier la vie à l'utilisateur et être transparent dans la gestion des valeurs manquantes. Dans ce paragraphe, nous nous contenterons de parler de celles-ci dans le cadre des étapes DATA [17].

Le minimum est de savoir qu'une donnée manquante numérique s'écrit avec un point (`.`), tandis qu'une donnée manquante littérale s'écrit `""` (sans espace, càd pas `" "`). Par exemple

```
data donplus;  set don;  if depenses > recettes then do; decouv=recettes-depenses; statut="A decouvert"; end;
                                                 else do; decouv=. ; statut=""; end;
```

Cependant dans l'exemple ci-dessus, ce qui suit le `else` est en fait inutile, car les variables `decouv` et `statut` sont de toute façon créées, et si on ne leur affecte aucune valeur elles sont automatiquement affublées d'une valeur manquante pour l'observation concernée.

Pour demander s'il y a une valeur manquante, on écrit `if x is missing` ou alors `if x=.` ou `if x=""` (selon le type de la variable `x`). Pour demander si ce n'est pas une valeur manquante, on écrit `if x is not missing`. Une instruction fréquente est par exemple `if x is missing then DELETE;` qui supprime l'observation en cours si la variable `x` présente une valeur manquante [18]. On notera que l'instruction précédente est généralement écrite de manière plus compacte avec ce qu'on appelle un IF implicite :

```
if x is missing then delete;   equivaut a   if x is not missing;
```

Un `if` implicite est dépourvu de `then`, et fait que seules les observations satisfaisant à la condition suivant le `if` seront conservées dans la table SAS en question [19].

On notera que les valeurs manquantes numériques sont, par convention, strictement inférieures à toute valeur numérique valide.

Fonctions SAS

SAS dispose comme tout language d'une liste importante de fonctions, notamment des fonctions mathématiques et probabilistes (fonctions de répartition, fonctions quantiles, générateurs aléatoires,...), ainsi que de nombreuses fonctions agissant directement sur des fichiers. Il y en a plus de 600, dont beaucoup sont très pratiques mais on ne se doute pas de leur existence ! La description de toutes ces fonctions, et leur présentation par catégories, se trouve dans l'onglet suivant de l'aide :

SAS Language Dictionary→Dictionary of Language Elements→ Functions and CALL Routines

Nous n'en détaillerons aucune ici, mais nous listons quand même, pour mémoire, les noms de quelques fonctions : d'abord, probabilistes

`rand('uniform')`	*gener. de* $\mathcal{U}nif(0,1)$	`probnorm(`x`)`	$\mathbb{P}(\mathcal{N}(0,1) \leq x) = \Phi(x)$
`rand('bernoulli',`p`)`	*gener. de* $\mathcal{B}(p)$	`probit(`p`)`	$\Phi^{-1}(p)$
`rand('binomial',`p`,`n`)`	*gener. de* $\mathcal{B}(n,p)$	`quantile('normal',`p`,`μ`,`σ`)`	$p^{ème}$ *quantile de* $\mathcal{N}(\mu,\sigma^2)$
`rand('normal',`μ`,`σ`)`	*gener. de de* $\mathcal{N}(\mu,\sigma^2)$	`quantile('gamma',`p`,`α`,1/`β`)`	$p^{ème}$ *quantile de* $\mathcal{N}(\alpha,\beta)$
... etc avec `'T','F','exponential', 'chisquare','weibull'...`		`quantile('exponential',`p`,1/`λ`)`	$p^{ème}$ *quantile de* $\mathcal{E}xp(\lambda)$
`cdf('normal',`x`,`μ`,`σ`)`	$\mathbb{P}(\mathcal{N}(\mu,\sigma^2) \leq x)$	`cdf('binomial',`k`,`n`,`p`)`	$\mathbb{P}(\mathcal{B}(n,p) \leq k)$
`cdf('exponential',`x`,1/`λ`)`	$\mathbb{P}(\mathcal{E}xp(\lambda) \leq x)$	`sdf('normal',`x`,`μ`,`σ`)`	$\mathbb{P}(\mathcal{N}(\mu,\sigma^2) \geq x)$

puis usuelles (`sign, sqrt, int, ceil, floor, round, dif, lag, max, min, sum, mean, geomean, harmean, median, rms, std, mad, iqr, ordinal`), de manipulation de caractères (`trim, cat, catx, quote, count, verify, anyalpha`, etc...), de manipulation de dates et heures (`today, weekday, datetime, hms, dhms, day, week, year, date`), et enfin mathématiques (`exp, log, cos, sin, tan, cosh, sinh, tanh, arcos, arsin, artan, gamma, digamma, beta, airy, mod`,...).

Exemple : probability plot d'un échantillon aleatoire de gaussiennes

```
data gauss;  do i=1 to 40;  x=rand('normal',3,4); y=cdf('normal',x,3,4); output;  end;
proc sort; by y; run;
data gauss;  set gauss;  u=_N_/40; run;
proc print data=gauss(obs=20); run;
proc gplot data=gauss;
 symbol1 I=none value=circle cv=red width=1;    plot u*y ;
 symbol1 I=rl;  plot u*y;
run;
```

17. dans l'aide de SAS, un onglet parle de ce sujet : *SAS Products→ SAS Base→ SAS Language Concepts→ SAS System Concepts→ Missing Values* ; pour la gestion des valeurs manquantes au sein de procédures, se référer à l'aide de chacune séparément.

18. en termes techniques, l'instruction `delete` fait vider le contenu du PDV et commencer une nouvelle itération de la boucle DATA, en revenant au début du corps DATA

19. Remarque : les instructions `if` implicites n'ont rien à voir avec la notion de valeurs manquantes, et fonctionnent avec toute condition booléenne : `if x=0;`, ou `if sexe='Femme';` sont tout à fait valables aussi bien entendu.

Syntaxe des opérateurs usuels

- l'élévation à une puissance se fait avec `**` (par exemple x^4 s'écrit `x**4`).
- les opérateurs de "comparaison" sont `EQ` ($=$), `NE` ($\neq$), `LE` ($\leq$), `GE` ($\geq$), $>$, $<$. Ils servent le plus souvent au sein des commandes `where, if` (implicite), `if... then`. On notera que des équivalents à `EQ, NE, LE, GE` sont `=, ^=, <=, >=`.
- les connecteurs logiques sont `and` et `or` (qui peuvent être écrits `&` et `|`).
- l'expression `0<x<1` est valide et équivaut à `x>0 & x<1`, et `x in (0,1,2)` équivaut à `x=0 | x=1 | x=2` (voir le concept de liste de modalités page 15 et dans le paragraphe suivant).
- on peut écrire `x<>y` au lieu de `max(x,y)`, et `x><y` au lieu de `min(x,y)`.
- l'opérateur de concaténation est `||` (càd `'abc' || 'def'` vaut `'abcdef'`). Cependant, les variables alphanumériques étant par défaut de longueur 8, il est souvent nécessaire d'"élaguer" les blancs inutiles, et c'est là que la fonction `trim(.)` est utile : elle enlève les blancs inutiles aux chaînes de caractères, comme le démontre le code suivant

```
data don; input x $ y $;
z1=x||y; z2=trim(x)||trim(y); cards;
abc def
;
```

qui donne comme résultat

x	y	z1	z2
abc	def	abc	def	abcdef

- les opérateurs `=:`, `<=:` et `>=:` sont intéressants, ils permettent de comparer le début d'une chaîne de caractères à une suite de caractères donnée. Par exemple, `if var =: "F"` teste si la valeur de la variable (supposément alphanumérique) commence par F, et `if var <=: "F"` teste si la valeur de la variable commence par une lettre plus petite que F dans l'ordre lexicographique. Cela fonctionne de la même façon avec `if var =: : "Fra"`, qui teste si la valeur commence par Fra (casse prise en compte).
- l'opérateur `substr` permet de sélectionner une portion d'une chaîne de caractères. A titre d'exemple `substr(nom,2,3)` aura comme résultat la chaîne de 3 caractères partant du second caractère de la valeur courante de la variable *nom*, autrement dit les caractères numéros 2,3, et 4. Dans le même style, la fonction `index(`*char*`,"abc")` renverra le numéro de l'emplacement où commence la première occurence de la chaîne *abc* dans la valeur courante de la variable *char*.
- terminons en parlant de 2 opérateurs assez utiles, `ifn` et `ifc`, qui permettent de définir des variables dichotomiques suivant une condition à vérifier, sans utiliser de instruction `if ... then` : si `cond` est une expression booléenne quelconque (impliquant généralement une ou plusieurs variables) et `x` et `y` sont deux valeurs numériques, alors `var=ifn(cond,x,y)` va définir une variable *numérique* `var` qui vaut `x` si `cond` est vraie, et `y` sinon. Et si `c1` et `c2` sont deux chaînes de caractères, alors `var=ifc(cond,c1,c2)` va définir une variable *alphanumérique* `var` qui vaut `c1` si `cond` est vraie, et `c2` sinon. Par exemple

```
moyenne=ifn(mean(of note1-note6) >= 10,0,1)
```
et
```
moyenne=ifc(mean(of note1-note6) >= 10,"Oui","Non")
```

Tous les détails sur la syntaxe des différents opérateurs se trouvent dans l'onglet (particulièrement enfoui) suivant :

SAS Products→ SAS Base→ SAS Language Concepts→ SAS System Concepts→ Expressions→ SAS Operators in Expressions

De l'utilisation des booléens

Une expression booléenne telle que `x>0` est utilisée dans les instructions de type `if ... then` ou `where`. Cependant, il y a des circonstances où il y a une transformation automatique d'un booléen en valeur numérique [20], vrai étant transformé en 1, faux en 0. Par exemple, dans une étape DATA l'instruction

```
positif=(x>0);
```

va créer une variable numérique `positif` qui vaut 1 si `x` est strictement positif, et 0 sinon. Une telle transcription booléens-nombres peut être utilisée de plusieurs façons, par exemple dans une addition implicite : l'instruction

```
nombre + (x>0);
```
équivaut à
```
if _N_=1 then nombre=0;  if x>0 then nombre=nombre+1;
```

va créer une variable `nombre` qui vaudra la *fréquence cumulée* de l'événement `x>0`. Un autre exemple intéressant est

```
groupe = 1 + (x>0) + (x>1);
```

qui équivaut à `groupe=1; if x>0 then groupe=2; if x>1 then groupe=3;` .

Listes de variables, liste de modalités

Il est possible de gagner du temps et éviter des fautes de frappe en ayant recours à des *listes de variables*, par exemple quand on veut appliquer la procédure `means` à toutes les variables numériques, ou seulement à

20. on notera que le type de variable "booléen" n'existe pas dans SAS, il n'y a que les types numérique, alphanumérique, et date.

certaines. Voila les différentes possibilités qu'offre SAS :

— `_all_` (`_numeric_ , _character_`) : toutes les variables (ou toutes les numériques, ou alphanumériques)

— `x1-xn` : équivaut à la liste des variables de noms consécutifs `x1, x2, ... xn`

— `n--m` : équivaut à la liste de toutes les variables comprises (inclusif) entre les variables `n` et `m`, l'ordre considéré étant enregistré dans la table SAS, qui est généralement celui issu de la création de la table, et qui est visionable via la procédure `contents`.

— `n-numeric-m` et `n-character-m` : idem que ci-dessus, mais seulement les variables numériques ou alphanumériques, selon le cas.

— `var` (lisez bien var 2 points) : toutes les variables dont le nom commence par `var`

Exemple : `m=max(of var1-var7)` génère une variable `m` dont la valeur est le max des variables `var1,var2,...,var7`.

Remarque : il ne faut pas confondre les listes de variables avec les *listes de modalités*, qui ne sont pas le même objet, et ne partagent pas la même syntaxe. Si par exemple dans une procédure `means` on ne veut traiter que les observations de la table SAS `don` pour lesquelles la variable `var` vaut `'bleu'`, `'vert'`, ou `'rouge'`, on écrira

```
proc means data=don(where=(var in ('bleu','vert','rouge'))); ... run;
```

Tableaux (arrays) de variables

Il s'agit d'un concept tout à fait différent du précédent, et que l'on retrouve dans la plupart des langages de programmation (car presque indispensable dans certaines situations). La création d'un tableau de variables consiste à assigner des sortes d'alias et un indice à un certain nombre de variables, le temps d'une étape DATA, afin par exemple de faire réaliser la même "tâche" à ces variables, via une boucle le plus souvent. Il existe des tableaux uni- ainsi que multi-dimensionnels : nous ne parlerons ici que des uni-dimensionnels.

La syntaxe pour définir un *array* dans une étape DATA est simple : il faut utiliser l'instruction `array`, suivie du nom de cet *array*, immédiatement suivi (sans espace) entre accolades du nombre de variables que compte l'*array* (on peut aussi mettre une étoile {∗}, SAS comptant alors tout seul le nombre de variables déclarées), suivi de la *liste de variables* définissant l'*array* (voir précédemment pour la notion de liste). Par exemple,

```
data don(drop=i);
input nom $ age taille poids;
array tab{3} age taille poids;   array tabar{3} agear taillear poidsar;
do i=1 to 3; tabar(i)=round(tab(i)); end; cards;
pauline 12 67.4 7.4
dounia  11 70.7 9.3
;
proc print noobs; run;
```

Quelques remarques :

– cet exemple est purement illustratif, et en aucun cas il ne s'agit d'une solution de programmation optimale ; par ailleurs, l'usage des tableaux dans la programmation SAS va bien au delà de cet exemple "jouet", et il est très judicieux d'aller voir quelques exemples de son utilité dans l'aide de SAS (voir références ci-dessous)

– plus haut, on aurait pu définir le tableau `tabar` avec `array tabar{*} agear ...`, ce qui revenait au même (la déclaration de dimension de tableau est plus subtile pour les tableaux à plusieurs dimensions)

– on peut utiliser l'expression `dim(tabar)` dans la boucle `do i=1 to dim(tabar)`

– on peut utiliser toutes les listes de variables que l'on veut, par exemple `array tab{*} _numeric_` incluera toutes les variables numériques dans l'*array*.

– les variables définissant l'*array* n'ont pas besoin d'avoir ete declarees au prealable (voir les références de l'aide SAS ci-dessous pour des exemples) ; il y a à ce sujet deux choses utiles à dire. D'abord, si elles ne l'ont pas été et qu'il s'agit de variables littérales, il est nécessaire de faire précéder d'un dollar `$` la liste des variables. Ensuite, on peut très bien affubler les variables définissant l'*array* de valeurs initiales, en les faisant figurer entre parenthèses à la suite de la liste de variables ; par exemple,

  ```
  array niveaux{4} t1-t4 (0.5 1 5 10);
  ```

  va créer un *array* de dimension 4 dont les variables associées (a priori inexistantes auparavant) sont `t1, t2, t3, t4`, et sont affublées des valeurs initiales 0.5, 1, 5, et 10.

– si par exemple les variables `var1-var6` existent déjà dans la table SAS, alors la déclaration suivante est licite : `array var{6};`, car SAS comprendra tout de suite de quoi il s'agit (cf aide).

– une utilisation possible d'un tableau est par exemple `x=mean(of tabar);` qui a comme effet d'affecter à la variable `x` la moyenne des variables constituant l'*array* `tabar`.

Les onglets de référence sur le sujet des *arrays* dans l'aide de SAS sont les suivants :

SAS Products→ SAS Base→ SAS Language Concepts→ DATA Step Concepts→ Array Processing
SAS Products→ SAS Base→ SAS Language Dictionary→ Dictionary of Language Elements→ Statements→ ARRAY Statement / Array Reference Statement

3. Macro-commandes et macro-variables

Comme de nombreux logiciels, SAS a un langage permettant de créer des macro-commandes, ou plus communément *macros*. Une macro est en fait un morceau de code qui a besoin d'être réutilisé de manière intensive : elle est affublée d'un nom, et peut-être paramétrable (en fait, elle l'est le plus souvent). Il existe aussi des *macro-variables*, qui jouent plus ou moins le rôle de *variables globales*, et leur usage facilite énormément la vie de l'utilisateur de SAS (macro-commandes et macro-variables sont communément appelées des alias dans d'autres contextes).

Le chapitre VII de ce poly est consacré au langage macro de SAS, et en présente la syntaxe essentielle, avec illustrations.

4. Impression, Sauvegarde des resultats

Pour enregistrer dans un fichier une partie des sorties des procédures SAS, il n'y a pas mieux que l'utilisation de la fenêtre **Results**, bien que des commandes permettent de le faire sans clics ni souris au sein même des programmes SAS. Pour faire court, on peut sélectionner des portions de l'arborescence que contient la fenêtre Results, et sélectionner l'impression en cochant une case telle que *"Print to File"* : s'il s'agit d'un résultat de procédure, la sauvegarde se fera sur fichier texte (extension `.lst`), et s'il s'agit d'un graphique, on peut par exemple sous Unix sélectionner le format *postscript* (`.ps`) puis gérer l'impression via un visualiseur postscript tel que *gv* (sous Windows, si une imprimante est bien configurée et reliée à l'ordinateur, il y a moyen d'imprimer directement sans sauvegarder l'image dans un fichier).

Il est aussi possible de changer de destination le tracé d'un graphique, c'est-à-dire au lieu de le faire afficher à l'écran, l'enregistrer dans un fichier postscript. L'intérêt est que cela ne nécessite pas de manipulations à la souris [21]. Voici un exemple, faute de développer plus le sujet dans ce document :

```
filename gsasfile "F:\sas\essaigraph.ps";
goptions ftext=zapf device=ps gaccess=gsasfile border rotate=landscape;
proc gplot data=don; plot y*x; run;
```

Avec ce code, le graphique n'est plus affiché à l'écran, mais dans le fichier postscript d'emplacement physique F: \sas\essaigraph.ps . Pour rentrer dans les détails, `goptions` est une instruction globale qui permet de spécifier des options graphiques, et les deux options principales sont ici `device=` et `gaccess=` : `gaccess=gsasfile` indique que la destination du graphique devient un fichier externe, dont le nom est la valeur du *fileref* `gsasfile` (attention, on ne peut choisir un autre *fileref*...), et `device=ps` indique que le graphique doit être enregistré au format postscript (lisible avec Ghostview, sous Windows, ou gv, sous Unix). L'option `ftext=` stipule la fonte (du titre par exemple), et `rotate=` l'orientation...

Enfin, il y a possibilité d'utiliser les capacités très intéressantes de l'**Output Delivery System (ODS)** pour générer des versions sous format `rtf, html, ps` ou `pdf` de certaines sorties SAS [22], ou encore plus directement via un clic droit dans l'arborescence de la fenêtre Results pour obtenir des sorties sous format `xls` ou `html` [23] ; voir le paragraphe IX-2 page 59 à ce sujet.

5. Di erents acces a SAS (version Unix seulement)

Sous Unix, la plupart du temps, l'accès à SAS se fait en lançant la commande `sas &`. Cependant, ce n'est pas le seul moyen d'utiliser SAS. D'abord, on peut adjoindre des paramètres à cett commande unix `sas` : par exemple, `sas -work ~/sas/ &` va lancer SAS en demandant que le répertoire *Work* soit défini comme un des répertoire de votre compte Unix (en l'occurence ici `~/sas/`). Très nombreux sont les aspects de SAS qui peuvent être configurés ainsi dès le lancement, nous voulions ici seulement mettre le doigt sur un aspect "configuration personnelle" auquel on ne pense pas toujours...

Il est également possible d'exécuter des programmes SAS "en mode batch" (*i.e.* en arrière plan) càd sans demander le lancement de toute l'interface graphique usuelle. Par exemple, si `fic.sas` contient des commandes SAS se suffisant à elles-mêmes, alors la commande unix `sas fic.sas &` va les exécuter et enregistrer les résultats (normalement affichés dans la fenêtre Output) dans un fichier texte `fic.lst` lisible avec n'importe quel éditeur. Ceci est surtout utile pour faire tourner de gros programmes (la nuit par exemple) sans accaparer trop de ressources (graphiques notamment) ; cela montre l'utilité de maîtriser la programmation SAS plutôt qu'une

21. si, si ! Se passer de la souris est très intéressant !
22. voir les onglets suivants de l'aide de SAS : *SAS Products→ SAS Base→ Step-by-Step Programming...→ Designing your own output*, ou *SAS Products→ SAS Base→ SAS Language Concepts→ SAS System Concepts→ SAS Output*, ou encore *SAS Products→ SAS Base→ Output Delivery System* (manuel de référence sur le sujet)
23. voir l'onglet suivant de l'aide de SAS : *SAS Products→ SAS Base→ Using Base SAS software→ Managing your Files*

approche "je clique un peu partout". Pour sauvegarder des graphiques en procédant ainsi, il suffit d'avoir recours à un code comme celui qui est fourni dans le paragraphe précédent.

III. Conseils & Astuces, conclusion provisoire, et description de l'aide de SAS

1. Remarques diverses, conseils & astuces :
   - Quitte a radoter un peu, il est toujours utile de repeter qu'une lecture attentive du journal (fenêtre Log) est indispensable pour resoudre les problemes de syntaxe qui se posent toujours dans toute activite de programmation...
   - Pour inserer des remarques/notes dans un code SAS, ou pour isoler une certaine portion du code, il y a deux moyens. D'abord, tout ce qui est compris entre une etoile * et un point virgule ; sera ignore. Ensuite, il en est de même pour tout ce qui est compris entre `/*` et `*/`.
   - L'option de procedure `data=`*nom de table sas* est en un certain sens facultative : si elle est omise, alors la procedure prendra en entree la derniere table SAS cree, soit via une etape DATA, soit en sortie d'une procedure. Par exemple, dans

     `proc sort data=`*don* `out=`*donout*`; run; proc print; run;`

     la procedure `print` va a cher a l'ecran le contenu de la table SAS *donout*, qui vient juste d'être creee (d'ailleurs, on peut noter que la commande `run;` du milieu est facultative, cf plus bas).
   - commandes globales : il existe un certain nombre de commandes globales, c'est-a-dire qui ne sont inclues ni dans une etape DATA ni dans une etape de procedure. Un certain nombre d'entres elles sont decrites a la n de ce document (`TITLE, OPTIONS, FILENAME, LIBNAME, GOPTIONS`, etc...)
   - modules de SAS : SAS comprend de nombreux modules, dont BASE, STAT, ETS (series chronologiques), GRAPH (graphiques haute-resolution), OR (recherche operationnelle), etc... Dans ce document ne seront evoques que les modules BASE, STAT, et GRAPH.
   - dans le monde anglo-saxon et donc dans SAS, \frequency" signi e *effectif*, ce qui est trompeur et doit être garde a l'esprit !
   - du fonctionnement de la commande `run;` : il n'est pas obligatoire de terminer une procedure systematiquement par une commande `run;`, il su t de ne l'indiquer qu'apres une succession de procedures.
   - majuscules et minuscules : hormis dans des valeurs alphanumeriques (cha^nes de caracteres), SAS ne tient pas compte de la casse et tra^te indi eremment majuscules et minuscules la plupart du temps.
   - par defaut, les noms de variables ou de tables sas peuvent être de taille jusqu'a 32 caracteres, les noms d'informats 7, et les noms de *libref, fileref*, et formats, 8 caracteres.
   - il est indispensable d'employer une double apostrophe pour avoir une apostrophe dans une cha^ne de caracteres, sous peine de voir se terminer prematurement celle-ci (et occasionner a tous les coups des erreurs penibles [24]). Par exemple, pour avoir le titre *Résultats de l'ACP*, il faut utiliser, dans la procedure `princomp`, l'instruction

     `title 'Resultats de l''ACP';` ou bien `title "Resultats de l''ACP";`

     Autre exemple : l'instruction `if rep='Oui' then x=1;` va assigner la valeur 1 a x si la variable litterale `rep` vaut 'Oui' (et une valeur manquante a x sinon). Par contre, l'instruction `if rep=Oui then x=1;` va cherche une variable de nom `Oui` pour comparer sa valeur a celle de la variable `rep`, ce qui occasionera une erreur ! (du genre *Variable Oui not known*...)
   - il parait futile d'adjoindre un titre (avec l'instruction `title '.....';` , disponible dans toutes les procedures) *à chaque fois* qu'une procedure cree une sortie a l'ecran, mais en realite c'est tres utile car les titres que l'on prend sont repris dans la fenêtre **Results** de SAS, qui s'avere on ne peut plus pratique a l'usage pour retrouver ses resultats anterieurs et travailler dans de bonnes conditions. Moralite : faites des titres systematiquement (au depart on trouve ca lourd et inutile, mais a la longue, c'est tres pratique).
   - le point virgule : son absence sournoise vous empoisonnera souvent la vie, tout comme le fera un oubli d'apostrophe...
   - il est deconseille d'utiliser les ascenseurs (scrollbars verticales) dans la fenêtre de sortie notamment : l'utilisation des touches *pageup* et *pagedown* s'avere beaucoup plus e cace et moins exasperante.
   - la fenêtre Results s'avere a l'usage d'une grande utilite pratique, a condition d'y faire regner un minimum d'ordre et d'organisation. Ses potentialites sont grandes, notamment en ce qui concerne la sauvegarde ou l'impression de parties de sorties ou de graphiques.
   - il est facile de passer d'une fenêtre a une autre, que l'on soit dans la version Unix ou bien Windows de SAS. Pour cela il su t de cliquer dans l'onglet *View* (ou *Affichage* ou encore *Fenêtres* suivant la version) puis selectionner la fenêtre de son choix, qui viendra se placer en avant-plan a l'ecran. C'est extrêmement pratique a l'usage.
   - la macro `%include` permet de charger le contenu d'un chier d'instructions dans un autre. Par exemple, `%include "fichier.sas";` va lancer tout le code SAS contenu dans le chier `fichier.sas`. Ceci est evidemment tres utile pour charger des macros personnelles dans ses propres chiers SAS...

24. IMPORTANT : une erreur fréquemment rencontrée en pratique est l'oubli de cette règle de la double apostrophe ; cela a comme conséquence que tout ce qui suit la dernière apostrophe tapée est traité par SAS comme une chaîne de caractère et en particulier tout le reste des commandes et des instructions n'est pas soumis à SAS. En pratique, on se retrouve avec une session SAS qui ne répond plus à aucune des commandes et des instructions qu'on lui soumet, le programme faisant "la sourde oreille". La solution pour résoudre le problème est de soumettre exactement la ligne `';run;` puis, en ignorant tout ce que dit le fenêtre du log, et après avoir identifié l'emplacement de l'apostrophe manquante, relancer tout à partir de l'étape où tout semblait avoir chaviré... Si malgré tout cela ne fonctionnait toujours pas, il faut sortir la grosse artillerie et soumettre la "chaîne magique" suivante `;*';*";*/;quit;*%mend;` .

– la gestion des attributs de variables que sont les `label, format, length`, peut quelquefois s'averer laborieuse quant elle est e ectuee separement pour chaque attribut et chaque variable. L'instruction `attrib` de l'etape DATA permet de tout gerer en même temps, variable par variable ; voir l'aide de SAS pour sa description.

– de nombreux outils sont disponibles dans les onglets *Outils/Tools* et *Solutions* : en se servant de l'aide de SAS, il est possible d'en explorer les potentialites, même si l'aide se cantonnera aux aspects pratiques et le plus souvent ne decrira pas le fond "scienti que" de ces outils.

– desormais des ouvrages sur SAS existent en francais, certains sont particulierement reussis, d'autres sont une mine d'information mais trop brouillon, d'autres encore se limitent a des aspects trop basiques. Cependant vous ne couperez pas a la necessite d'utiliser l'aide de SAS : bien entendu, il est dommage de devoir lire de la documentation a l'ecran, et peu ecologique d'imprimer a tout va les pages d'aide d'un logiciel, donc il est utile d'apprivoiser le plus tôt possible l'arborescence de l'aide de SAS (voir le paragraphe suivant "Conclusion et Perspectives").

– il existe des raccourcis clavier : la commande SAS `KEYS` appelle la fenêtre les listant, et dans laquelle on peut en ajouter ou en modi er certains.

– dans SAS, par defaut toutes les variances (et covariances) sont calculees avec le denominateur "$n-1$". Il est tres important de le savoir et d'en tenir compte. Cette convention peut toujours être changee, dans de nombreuses procedures, par l'utilisation de l'option de procedure `vardef=N` (denominateur n). Voir l'aide pour plus de details...

– il ne faut jamais oublier qu'une etape DATA `data don; ....` va ecraser l'eventuelle table SAS don qui pouvait exister, et ce, sans autre forme de proces. Il est donc absolument fondamental que le statisticien conserve precieusement ses donnees de depart (sa matiere premiere !), sous une forme ou sous une autre. De toute facon, comme SAS est un langage de programmation, le code peut toujours être resoumis, n'entrainant qu'une perte de temps de calcul (ce qui, il est vrai, peut être dommageable, si les ressources en calcul sont limitees ou a partager).

– SAS est un logiciel hautement con gurable, bien que sans aide ou documentation appropriee cela soit un peu ardu de prime abord. Il y a de nombreux endroits dans l'aide de SAS ou il est question de ces possibilites de con guration (options de systemes modi ees de maniere \permanente", pro l d'utilisateur, registre SAS personnel, chier(s) de con guration, chier autoexec lance au debut de chaque session, etc...). Voila ci-dessous certains emplacements ou l'on peut trouver de telles informations :

Using SAS Software in your operat. env.→ Using SAS in Unix→ Running SAS Software under Unix
→ Customizing the SAS Windowing Environment in UNIX Environments
Using SAS Software in your operat. env.→ Using SAS in Windows→ Running SAS under Windows
→ Interacting with SAS under Windows
SAS Products→ Base SAS→ Using Base SAS Software→ Using the SAS Windowing environment
→ Customizing the SAS windowing environment

2. Conclusion a cette premiere partie et Perspectives

SAS est une usine à gaz, alors à partir de quand peut-on considérer avoir un niveau de base en SAS ?

(i) quand on sait exploiter les tables issues de procédures dans d'autres procédures

(ii) quand on sait trouver une information rapidement

(iii) quand on commence à écrire et utiliser des macros

(iv) quand on arrive rapidement à débugger ses programmes

Dans ce cas, on peut considérer qu'une étape est franchie dans la compréhension et l'autonomie, mais il ne faut pas se leurrer :

impossible de progresser sans comprendre et ma^triser les notions statistiques sous-jacentes

En attendant que ce stade soit atteint, voila quelques conseils en vue de l'acquisition rapide d'une autonomie avec le logiciel SAS et son fonctionnement dans l'environnement Unix :

— Inutile de rechigner avec l'anglais : la quasi totalité de la documentation sur SAS est en anglais, et il faut faire avec, et ne pas se contenter de ce document ou de bribes en français trouvées ici ou là.

— L'aide de SAS (accès via l'onglet *Aide/Help* de n'importe quelle fenêtre) est particulièrement volumineuse, et ce serait une erreur de penser qu'il s'agit d'un sac de nœuds inextricable : prendre du temps pour découvrir ce que cache l'arborescence à première vue aride qui s'offre à vous, c'est faire un pas décisif vers une autonomie très productive et efficace qui vous rendra de grands services et est la seule garante de progrès dans l'apprentissage de SAS, en vue de son utilisation professionnelle. Pour débuter dans votre exploration de l'aide, vous trouverez ci-après un (court !) extrait de l'arborescence mettant le doigt sur des nœuds intéressants, et qui permet de s'orienter (l'ordre des branches au sein de chaque nœud n'a pas toujours été respecté, et il s'agit de l'aide de SAS 9.1 version Windows ; pour la version Unix l'aide diffère en plusieurs endroits).

⊖ Learning to Use SAS
  ⊢ Sample SAS Programs (*source très importante de programmes-types*)
  ⊢ Tutorial : Getting Started with SAS Software (*Endroit idéal pour débuter ou s'y remettre doucement*)

⊖ Using SAS Software in Your Operating Environment

(*informations speci ques au fonctionnement de SAS dans chaque systeme d'exploitation : Unix, Windows... voir feuillets separes*)

⊖ **SAS Products → Base SAS**

⊖ **Step-by-Step Programming with Base SAS Software**

↪ *Cet onglet vaut un livre à lui tout seul pour l'apprentissage des bases de la programmation SAS*

⊕ Introduction to the SAS system (*mise en jambe*)

⊕ Getting your Data into Shape (*notion de boucle DATA, types d'input (list,column), subtilites de lecture des donnees, usage de @ et @@, multiples input, options de noms de tables SAS*)

⊕ Basic Programming

(*usage de LENGTH, IF-THEN-DELETE, manipulations et operations sur les variables numeriques et alphanumeriques, usage de OUTPUT, calculs de totaux, notion de retenue, usage de RETAIN, usage de DO-END, notion de tableaux/arrays de variables, manipulation de dates*)

⊕ Combining Data Sets (*chapitre sur le delicat sujet de la* **combinaison de tables SAS** *: usage des mots-clef SET, UPDATE, MERGE, MODIFY, subtilites d'utilisation de BY*)

⊕ Understanding your SAS Session (*b-a-ba du* **débuggage**)

⊕ Producing Report (*par l'utilisation des procedures REPORT ou TABULATE*)

⊕ Designing your own Output (*usage des mots-clef PUT et FILE, diverses options de sortie, introd. a l'*ODS)

⊕ Storing and Managing Data in SAS Files (*mise en pratique du* **système de noms SAS**, *manip. de tables SAS*)

⊕ Understanding you SAS Environment (*divers conseils d'utilisation du systeme de fenêtres SAS commandes du Program Editor, et indications de* **configuration** *de SAS*)

⊕ Glossaire de nombreux termes techniques

⊖ **SAS Language Concepts**

↪ *Cet onglet présente, développe, et illustre* **tous** *les éléments et concepts du language SAS*

⊖ SAS System Concepts

– Essential Concepts (*introduit le reste du chapitre, et evoque les chiers de con guration et autoexec*)
– Rules for Words and Names
(*regles generales de syntaxe pour les* **noms** *dans SAS ; voir aussi + bas onglet sur les expressions*)
– SAS Language elements
(**options de tables, (in)formats, fonctions SAS, routines CALL, et options de système** (très détaillé))
– SAS Variables *(types, leurs* **attributs**, *variables* _N_ *et* _ERROR_, *listes, instructions/options* `keep, drop, rename`*)*
– SAS Expressions *(syntaxe generale, conversion automatique numerique↔litteral par SAS, et paragraphe tou u et important sur les* **opérateurs** *en general : arithm., manip de booleens, double statut des booleens, precisions sur l'ordre lexicographique)*
– Missing Values *(quelques generalites sur l'origine et la gestion des* **valeurs manquantes***)*
– Dates, Times, and Intervals *(chapitre de reference sur le sujet important des* **dates**, *et leur gestion)*
– Error Processing and Debugging *(traite du* **débuggage** *; pour la lecture du Log, voir ci-dessous)*
– SAS Output *(gestion et routage des* **sorties**, *lecture et manip du* **Log**/*Journal, grosse introduction a l'*ODS*)*
– WHERE-Expression processing *(syntaxe detaillee et conseils dans l'utilisation de instructions* `WHERE`*)*
– The SAS Registry *(manuel pour manipuler et con gurer son registre SAS, quand on est assez aguerri)*
– Printing with SAS *(manuel de reference sur l'***Universal Printing**, *methode de base d'impression)*
– *(...Autres paragraphes, notamment sur l'optimisation et la performance dans l'utilisation de SAS...)*

⊖ DATA Step Concepts

– DATA Step Processing
(description tres precise du **déroulement des étapes DATA**, *et des moyens d'***influencer** *le deroulement standard de la boucle DATA par l'usage de nombreuses commandes, et instructions pour utiliser SAS comme moyen de generer des* **fichiers textes** *en serie, via la commande* `PUT`, *comme des rapports par exemple)*
– Reading Raw Data *(traite des di erentes sources de donnees, et des* **types d'input***)*
– Reading, Combining, and Modifying SAS Data Sets
(manuel d'utilisation des instructions `SET, MERGE, MODIFY,` *et* `UPDATE`, *assorti de conseils et d'exemples)*
– Array Processing
(chapitre de reference concernant l'utilite, l'utilisation, la syntaxe, des **tableaux/arrays de variables** *)*

⊖ Windowing Environment Concepts

(Instructions moderement detaillees des capacites qu'o re l'environnement de fenêtres, et concernant aussi la gestion des repertoires SAS, la gestion des tables SAS via VIEWTABLE, et l'importation et exportation des donnees via les **Wizards** *correspondants)*

⊖ SAS Files Concepts

(ce chapitre aborde vraiment tous les sujets relatifs aux chiers SAS : repertoires, tables SAS, data views (d'etapes DATA, de SQL, d'ACCESS...), data les, catalogues, programmes compiles,... ; il aborde egalement la protection de chiers SAS par mots de passe, la reparation de chiers endommages, les di erents moteurs existants, la manipulation de chiers externes...)

⊕ **SAS Language Dictionnary → Dictionary of Language Elements**

↪ *à la différence de l'onglet précédent, détaille exhaustivement les différents éléments de langage SAS et leur syntaxe :*
- *Options de noms de tables SAS*
- *Options de système SAS*
- *instructions globales ou d'étapes DATA*
- *Formats (formats d'écriture)*
- *Informats (formats de lecture)*
- *Fonctions SAS et routines CALL (>600 !)*

⊖ **SAS Procedures**

↪ *Cet onglet incontournable est le manuel de référence de toutes les procédures de SAS Base*

⊖ Concepts
(*parle de nombreux sujets : presente d'abord rapidement les di erentes procedures de SAS Base, classees par categories, et donne les* **définitions** *precises de ce que SAS calcule dans ses procedures statistiques de base ; ensuite presente les options de systeme ou de tables sas utilisables au sein des procedures ; en n (dans "Procedure Concepts") aborde divers sujets, tels les subtilites des* **instructions BY**, *la notion de* **listes de variables**, *la pratique des* **formats**, *et un petit chapitre sur le tres utile* **ODS**-*Output Delivery System*)

⊕ Procedures
(*le detail procedure par procedure, avec illustrations et syntaxe precise et complete*)

⊕ Appendices
(*contient un chapitre sur les* **statistiques de base** *calculees par SAS, ainsi que toutes les* **tables** *SAS utilisees dans les exemples du manuel ci-dessus*)

⊕ **SAS Command Reference**

↪ *Cet onglet explique et liste toutes les* **commandes SAS** *telles SUBMIT, PRINT, PRTFILE, X,... et aussi celles qui sont "host-specific"*

⊖ **Using SAS Base Software**

↪ *Cet onglet est consacré à une description détaillée de la pratique de SAS*

⊕ Using the SAS Windowing Environment
(*utilisation e cace et optimale des di erentes* **fenêtres**, *ainsi qu'un chapitre sur l'editeur de* **registres**, *dans l'onglet "Customizing the SAS Windowing Environment"*)

⊕ Working with SAS Libraries
(*pratique de la creation de* **filerefs et librefs**, *catalogues SAS, et problemes frequemment rencontres*)

⊕ Using the VIEWTABLE window
(*manuel complet decrivant le fonctionnement de l'outil VIEWTABLE de visualisation de tables SAS*)

⊕ Managing your Files
(*visualisation en* `.html` *ou* `.xls` *de tables SAS, modifs a posteriori sur tables SAS, ...*)

⊕ Working with Programs
(*contient un gros chapitre tres interessant et totalement cache sur le* **débuggage** *des etapes DATA*)

⊖ **Output Delivery System**

↪ *manuel de l'ODS, système permettant de produire des sorties de procédures SAS en format ps,pdf,html,rtf...*

⊕ Introduction (*commandes de base et presentation des di erents fonctionnements d'ODS*)

⊕ Concepts (*Explications plus detaillees de l'utilisation de l'ODS, et son insertion dans une etape DATA*)

⊕ ODS Language Statements (*Glossaire precis du systeme ODS, et liste exhaustive de toutes ses commandes*)

⊕ **SAS SQL Procedure User's Guide**

↪ *Manuel détaillé pour l'utilisation de la procédure SQL, implémentation SAS du SQL*

⊕ **SQL Query Window**

↪ *Manuel de cet utilitaire qui facilite l'écriture de instructions SQL, appliquées à des données sous divers formats*

⊖ **SAS Macro Reference**

↪ *Manuel de référence des capacités Macro de SAS*

⊕ Understanding and Using the Macro Facility
(*manuel introduisant et décrivant en détails l'écriture de Macros SAS et l'utilisation de macro-variables*)

⊕ Macro Language Dictionary
(*liste, description, et syntaxe pour toutes les commandes et fonctions de Macros, et Macros pré-définies*)

⊕ Appendices
(*inclut un aide-mémoire de toutes les fonctions utilisables en conjonction avec la fonction %SYSFUNC*)

⊖ **SAS Products → SAS/STAT**

⊖ **SAS Products → SAS/GRAPH**

IV. Principales Procédures de SAS Base

- `proc CONTENTS`

  La procédure `contents` donne toutes les informations utiles sur la table SAS en entrée, notamment taille du fichier, emplacement SAS ou dans l'environnement, et surtout toutes informations utiles sur l'ensemble des variables composant la table, notamment type, format, informat, label, longueur, ... Très pratique voire indispensable dans de nombreuses situations (notamment quand on essaye de résoudre un problème récalcitrant et que l'on cherche une piste).

  Les options utiles sont `OUT=`*table_sas*, `NOPRINT`, `SHORT`. Une utilisation potentiellement utile est la suivante : elle permet de rajouter dans une table SAS une variable, `nobs`, qui contiendra la valeur du nombre de données que contient cette table sas [25] :

  ```
  proc contents data=table_sas noprint out=contents;
  data table_sas; set table_sas; if _N_=1 then set contents;
  proc print data=table_sas; run;
  ```

  Cependant, une manière plus élégante et rapide est d'utiliser l'option `nobs=` de l'instruction `set` :

  ```
  data table_sas; set table_sas nobs=n; nobs=n; proc print data=table_sas; run;
  ```

- `proc PRINT`

  Cette procédure élémentaire affiche tout ou partie du contenu d'une table SAS. Ses principales *instructions* sont :

  — `VAR` *var1 var2 ...* : indique les variables à afficher, et l'ordre dans lequel le faire (*par défaut toutes*!).

  — `ID` *nomvar* : par défaut, le résultat de la procédure comportera une première colonne (`OBS`) contenant le numéro de l'observation. l'instruction `ID` *nomvar* remplacera `OBS` par la variable *nomvar* en première colonne, ce qui est souvent plus agréable ; il est alors inutile de faire figurer cette variable dans la liste des variables en entrée de l'instruction `VAR`.

  — `BY` *varby* : affiche les données par modalité de la variable *varby*, voire par combinaison des modalités dans le cas où `BY` a comme argument une liste de variables ; nécessite *impérativement* un tri préalable suivant la ou les variables en question (*cf* `proc SORT` ci-dessous), sous peine de se faire gronder dans le Journal/Log.

  — `SUM` : calcule des sommes par groupes générés par `BY` pour une ou plusieurs variables données en argument).

  Des options utiles de la procédure `PRINT` sont : `NOOBS` (supprime l'affichage de la colonne `OBS` ; inutile en cas de instruction `ID`), `LABEL` (fait utiliser les labels, car par défaut ils ne sont pas pris en compte), `ROUND` (arrondit toutes valeurs numériques non préalablement formatées à deux décimales) ; par exemple :

  ```
  proc print data=table_sas label;  var age taille;  by sexe;  id nom; run;
  ```

  La procédure `PRINT` est très souvent utilisée en conjonction avec des *options de nom* pour la table SAS qui est en entrée. Par exemple, si on ne veut afficher que les 10 premières observations, toutes les variables sauf *var1* et *var5* de la table *table_sas*, et seulement les observations pour lesquelles *var2* vaut 'A' ou 'C', on écrira

  ```
  proc print data=table_sas(obs=10 drop=var1 var5 where=(var2 in ('A','C'))) noobs; run;
  ```

- `proc SORT`

  Avec l'instruction `BY` *var1*, la procédure `SORT` va trier les observations de la table en entrée par ordre croissant des modalités de la variable *var1* (ordre lexicographique dans le cas d'une variable alphanumérique) ; pour un ordre *décroissant*, utiliser l'instruction `BY descending` *var1*. Pour trier *suivant plusieurs variables en même temps*, il faut mettre la liste des variables en question à la suite de l'instruction `BY` (exemple : `BY departement descending population;` va faire trier d'abord dans l'ordre croissant suivant le département, "puis" dans l'ordre décroissant de la population).
  Pour enregistrer la sortie, il faut utiliser l'option de procédure `OUT=`*table_sortie*. Par exemple,

  ```
  proc sort data=table_sas out=table_en_sortie;  BY var1 var2 ...; run;
  ```

25. Explication de ce code : la table en sortie de `proc contents`, *contents*, contient de nombreuses variables, et une observation par variable de la table d'origine *table_sas*. Une des variables de *contents* est *nobs*, valant constamment la taille de la table d'origine pour chaque ligne. L'etape `data` qui suit va lire les observations de la table *table_sas*, mais aussi la 1ere observation de la table *contents(keep=nobs)*, autrement dit la table *contents* reduite a une seule variable, *nobs*. Ceci n'a lieu qu'au premier passage de la boucle `data`, et il y a alors, pour toutes les autres observations de la table *table_sas*, **retenue automatique** de la valeur de la nouvelle variable *nobs*. La table *table_sas* est donc ecrasee (car sortie de l'etape `data`) et remplacee par elle-même + une nouvelle variable valant constamment la taille de la table.

Une alternative est d'utiliser la procedure `means` au lieu de `contents`, avec `proc means data=`*table_sas* `noprint N; output out=nombre N=ntotal; run; data` *table_sas*`; set` *table_sas*`; if _N_=1 then set nombre; run;`

Attention ! si l'option `OUT` est omise, alors la table *en entrée* est automatiquement triée et modifiée/écrasée !!

Important : toute instruction `BY` *var1* dans une procédure SAS (telles `PRINT, MEANS, CHART,` etc...) requiert un tri prealable de la table en question suivant la variable *var1*. Quelquefois, une telle instruction `BY` peut être "remplacée" par une instruction `CLASS`, mais `BY` et `CLASS` n'ont pas vraiment le même effet : en particulier, `CLASS` va produire les résultat pour chaque modalité ou combinaison de modalités de la ou des variables, mais va aussi fournir les résultats pour chaque "loi marginale".

Exemple : pour la table sas *chaussures* (comptant les variables numeriques `ventes`, `benefices`, et les variables categorielles `zone` et `ville`), voila 2 exemples de tri, dont l'un avec le mot clef `descending` pour un tri decroissant :

```
proc sort data=chaussures; by zone;
title 'Proc SORT; BY zone; sans instruction OUTPUT, ecrase la table de depart !'; proc print; run;
proc sort data=chaussures out=chausstemp; by zone descending ville;
title 'Proc SORT out=chausstemp; BY zone descending ville; '; proc print; run;
```

- `proc UNIVARIATE`

Cette procédure peut calculer à peu près tous les indicateurs classiques d'une distribution numérique univariée, et réaliser les graphiques de base (histogrammes, boxplots, branches et feuilles, qq-plots, graphes des estimations paramétriques ou non-paramétrique de la distribution en question). Elle calcule également certains intervalles de confiance (moyenne, variance, quantiles théoriques) et les valeurs et p-valeurs des statistiques de certains tests d'adéquation (dont le test de Student).

La syntaxe de base comporte

```
proc univariate data=nom_de_la_table_SAS;
var variable ou liste_de_variables;
```

l'instruction `var` indiquant la ou les variables à analyser (si l'instruction `var` est omise, toutes les variables numériques sont traîtées, une à une). Les autres instructions non-graphiques possibles sont ensuite :

- `freq` : quand les observations ne sont pas brutes, mais que chacune est munie d'un poids contenu dans une variable *var_poids*, alors il faut utiliser l'instruction `freq` *var_poids*`;` pour indiquer que chaque observation doit être comptabilisée proportionnellement à la valeur correspondante de la variable *var_poids*.
- `by` ou `class` : suivi du nom d'une variable catégorielle, elle fait effectuer tous les calculs pour chaque groupe défini par les modalités de cette variable (par exemple, pour les femmes puis pour les hommes, si les observations sont des individus et que la variable en question est le sexe) ; dans le cas de `by`, les données doivent être préalablement classées, mais plusieurs variables classifiantes peuvent être indiquées, auquel cas les calculs se feront pour chaque combinaison de modalités.
- `output` *table_sortie* : cette instruction permet, au delà du simple affichage des résultats à l'écran, d'en sauvegarder certains dans une table SAS (la table *table_sortie*). Voir les détails plus bas.
- `id` *var* : cette instruction (beaucoup plus anecdotique) permet d'afficher les valeurs de la variable *var* donnée en argument, dans la table contenant les observations extrêmes (en effet, par défaut, seuls les numéros des observations extrêmes sont indiqués, ce qui n'est guère parlant).

les instructions graphiques seront décrites plus loin dans ce document : `HISTOGRAM, BOXPLOT, PROBPLOT, INSET` (paragraphe VII-6-b page 49). La procédure `univariate` compte également un grand nombre d'options, dont nous évoquerons certaines après avoir décrit les calculs de base qu'elle effectue par défaut.

La procédure calcule de très nombreux indicateurs statistiques relatifs à la ou les variables numérique stipulée dans l'instruction `var` (la plupart, mais hélas pas tous [26], sont enregistrables dans une table SAS à l'aide de l'instruction `output`, et possèdent donc un code, que nous indiquons entre parenthèses) : nombre [27] d'observations (N), nombre de valeurs manquantes (NMISS), moyenne (MEAN), mode (MODE), variance (VAR [28]), coefficient de variation (CV), écart-type (STD), somme totale (SUM), somme des carrés corrigée/non corrigée (CSS / USS), amplitude/portée (RANGE), min et max (MIN, MAX), coefficients d'asymétrie et d'aplatissement (KURTOSIS, SKEWNESS), médiane (MEDIAN), nombreux quantiles (P1,P5,P10,Q1,Q3,P90,P95,P99, et autres par le biais de l'option `pctlpts` de l'instruction `output`), intervalle inter-quartiles (QRANGE), déviation absolue moyenne (MAD, avec option `robustscale`), etc...

Test d'adéquation à une moyenne

Par défaut, la procédure fait calculer des statistiques de test de l'hypothèse $H_0 : \ \mu = \mu_0$ où $\mu_0 = 0$ et μ désigne l'espérance commune supposée de l'échantillon. Par exemple, le test de Student basé sur la statistique $T = \sqrt{n}(\bar{X}_n - \mu_0)/S_n$: SAS fournit la valeur observée T_{obs} ainsi que la p-valeur associée au test bilatéral (alternative $H_a : \ \mu \neq \mu_0$). La statistique T est notée t par SAS, et la p-valeur, qui vaut $\mathbb{P}_{H_0}(|T| > |T_{obs}|)$, est placée à côté de l'abbréviation `Pr > |t|`. ***Une telle expression, que l'on trouve dans bien d'autres***

26. comme les bornes de l'intervalle de confiance pour la variance par exemple.
27. le mot *e ectif* se dit *frequency* an anglais, ce qui est troublant...
28. attention ! Il s'agit de la variance empirique, dite variance corrigée et communément notée S_n^2, càd avec le dénominateur $n-1$; utiliser l'option `vardef=n` pour avoir le dénominateur n dans tous les calculs de ce type.

procédures, désignera toujours une p-valeur. Il est bien entendu possible de spécifier une autre valeur de μ_0, à l'aide de l'option de procédure `mu0=`μ_0 ; dans le cas où plusieurs variables sont analysées, on peut spécifier plusieurs valeurs différentes de μ_0 les unes après les autres.

Intervalles de confiance usuels

L'option `CIBASIC` ordonne le calcul d'intervalles de confiance gaussien (*i.e.* sous hypothèse de normalité) pour la moyenne, la variance, l'écart-type : par défaut les deux bornes sont calculées et le niveau est 95% (*i.e.* `CIBASIC` équivaut à `CIBASIC ALPHA=0.05`). L'option `ALPHA=`α sert à spécifier un niveau $1-\alpha$ pour l'intervalle. On notera que les bornes de ces intervalles ne sont pas récupérables dans une table de sortie (instruction `OUTPUT`), ce qui est plutôt contrariant : il faudra alors utiliser plutôt la procédure `MEANS`. Dans le même registre, les options `CIPCTLDF` et `CIPCTLNORMAL` génèrent des intervalles de confiance pour quantiles, suivant une méthode libre ou sous hypothèse de normalité.

Autres options utiles

L'option `FREQ` fait calculer les effectifs (souvent en conjonction avec une instruction `BY`), `NORMAL` fait réaliser des tests de normalité numériques ou graphiques (test de Shapiro-Wilks), `TRIM=`α ou `WINSORIZED=`α font raboter ou "Winsoriser" la moyenne (trimmed mean ou winsorized mean de proportion α), `NOPRINT` annule l'affichage des résultats de la procédure, `ROUND=`n arrondit à n décimales (on peut aussi utiliser `ROUND=0.5`...), `ROBUSTSCALE` fait calculer l'ensemble des indicateurs robustes (tels que la MAD, l'indice de Gini,...), `MODE` fait calculer le ou les modes, `NEXTROBS=`n indique combien de valeurs extrêmes afficher (inférieures et supérieures), et enfin `PLOT` fait tracer des graphiques branche & feuilles, boxplot (plusieurs côte-à-côte si instruction `by`), et prob-plot gaussien pour la distribution étudiée. L'option `ALL` réclame le calcul et l'affichage de tous les indicateurs possibles.

Affichage et sauvegarde des résultats (instruction `OUTPUT`*)*

Contrairement à la procédure `MEANS`, par défaut `UNIVARIATE` déverse une cascade de résultats à l'écran. Il est possible de ne lui en faire afficher qu'une partie précise, via l'instruction globale `ods select` (voir paragraphe VIII-2 page 59), et par ailleurs limiter les informations qui figureront dans le fichier de sortie généré par l'instruction `OUTPUT`.

La syntaxe de cette instruction est `OUTPUT` suivi de `out=`*table_de_sortie*, puis d'un certain nombre de mots-clefs, qui correspondent aux indicateurs que l'on veut voir figurer dans la table de sortie. Par exemple

```
output out=donout mean=moyenne std=ecart T=stud probT=pvalT P1=P1 P99=P99;
```

va créer la table `donout` contenant les variables `moyenne, ecart, stud, pvalT, P1, P99`. Comme il n'y a pas de instruction `by` ou `class`, cette table contiendra une seule observation, pour laquelle par exemple la valeur de la variable `moyenne` sera la moyenne empirique, celle de `P99` sera le quantile d'ordre 99%, ou encore celle de `probT` sera la p-valeur du test de Student d'adéquation de la moyenne à la valeur μ_0 qui aura été spécifiée. Les mots-clefs utilisés à gauche du signe d'affectation sont ceux qui ont été listés plus haut.

S'il y a une instruction `by` ou `class`, alors grosso-modo il y aura une observation par modalité et/ou combinaison de modalités de la ou des variables classifiantes (la structure de la table de sortie sera assez différente suivant que `by` ou `class` est utilisée). Si en outre plusieurs variables sont analysées dans la même procédure `univariate`, alors il faudra spécifier plusieurs noms de variables à la suite des signes `=` (voir exemple ci-dessous).

La procédure `UNIVARIATE` est donc une gigantesque usine à gaz : elle est une des principales procédures statistiques à savoir maîtriser, et à ce titre son manuel d'utilisation figure en bonne place, assorti de *très* nombreux exemples, dans l'aide de SAS :

SAS Products→ SAS Base→ SAS Procedures : CORR, FREQ, UNIVARIATE→ The UNIVARIATE Procedure

Regardons un exemple de syntaxe pour cette procédure avec la table SAS *chaussures* :

```
proc univariate data=chaussures cibasic alpha=0.01 location=500000 20000 round=2 robustscale;
 by zone;
 var ventes benefices;
 output out=univventes  mean=moyventes moybenef  median=medventes
                std=stdventes  mad=madventes  T=ttestventes ttestbenef;
proc print data=univventes;
 title 'R\'esultat de la proc\'edure Univariate; var ventes; (valeurs test : 500000 et 20000)' ;
run;
```

La procedure calcule les indicateurs habituels pour deux variables, *ventes* et *benefices*. Le resultat se fera a l'ecran avec une page par variable ainsi traitee, mais aussi par modalite de la variable *zone*. Avec l'option `robustscale` on obtiendra encore d'autres indicateurs (dont l'ecart absolu moyen), et l'option `cibasic alpha=0.01` fait calculer un intervalle de con ance de niveau 99% pour la moyenne (pour chaque variable), et avec l'option `location=500000 20000`, est realise le test de Student de conformite, a la moyenne 500000 pour *ventes*, et a la moyenne 20000 pour *benefices*. Maintenant, pour ce qui est de la sauvegarde, **seuls les indicateurs spécifiés seront inscrits dans la table de sortie** *univventes* : moyennes des ventes et des bene ces, medianes, ecarts-types et ecarts absolus moyens des ventes (aucun nom n'etant speci e pour la mediane, l'ecart-type, et le MAD des bene ces, ils ne sont pas sauvegardes), et valeurs de la statistique du test de conformite de Student (`T`). Il est interessant de voir par soi-même la structure de cette table de sortie (elle contient autant d'observations que de modalites de la variable classi ante *zone*).

- proc FREQ

Cette procédure génère ou gère des tables de contingence, pour calculer des distributions marginales ou/et réaliser des tests du χ^2 d'homogénéité. Elle permet également de mettre en œuvre le test du χ^2 d'ajustement à une distribution de référence. Elle est d'une utilisation extrémement fréquente.

La principale utilisation de la procédure `freq` consiste à calculer des effectifs croisés (c'est ce qui est appelé un *tri à plat...*). Considérons par exemple la table `don1` introduite en début de chapitre II, à laquelle on rajoute une variable `majeur` qui vaut 'oui' si l'individu est majeur(e) et 'non' sinon. Le table `don1out` issue du code suivant

```
data don1; set don1; if age >18 then majeur='oui'; else majeur='non';
proc freq data=don1;   tables sexe*majeur / out=don1out; run;
```

va être

sexe	majeur	count	percent
F	non	1	16.6667
F	oui	2	33.3333
H	non	2	33.3333
H	oui	1	16.6667

l'instruction `TABLES` fait construire la table de contingence 2×2 croisant les variables `sexe` et `majeur`, l'affiche à l'écran avec les effectifs, fréquences absolues, et fréquences marginales dans chaque cellule.

l'instruction principale de la procédure `FREQ` est donc l'instruction `TABLES var1*var2 </options>`, qui fait établir une table de contingence [29] avec en lignes les modalités de la variable *var1*, et en colonnes celles de *var2*; bien entendu ces variables sont censées être catégorielles ! Si elles ne le sont pas, il est possible d'exploiter une instruction `format` pour classer en catégories des données numériques (voir Exemple 2 plus bas).

Attention ! Il est important de comprendre que SAS va traîter les lignes de la table en entree de la procedure `FREQ` comme des *individus*, ayant chacun un poids (e ectif) de 1. Si l'on veut que SAS prenne comme poids d'une ligne la valeur d'une certaine variable (de comptage) *varcompte* pour cette ligne, il faut speci er l'instruction `WEIGHT` *varcompt*; en debut de procedure `FREQ`! Sinon chaque ligne pesera 1 et la table produite n'aura aucun sens... (voir Exemple 1 ci-dessous).

Test du χ^2 d'homogénéité/indépendance (option `CHISQ`*)*

Si l'on veut que SAS réalise le test du χ^2 d'homogénéité/indépendance sur la table, il suffit simplement d'utiliser l'option `CHISQ` de l'instruction `TABLES` : la valeur de la statistique du χ^2 et la p-valeur associée sont fournies, ainsi que d'autres mesures d'association du même acabit. Dans le cas d'une table 2×2, le test exact de Fisher est
entrepris (voir aussi l'instruction `EXACT` [30]).

Options intéressantes de l'instruction `TABLES`

`NOFREQ` annule l'affichage (mais pas le calcul !) des effectifs dans les cellules, et `NOCOL, NOROW, NOPERCENT` annulent respectivement l'affichage des proportions marginales selon *var1*, marginales selon *var2*, ou globales (la signification est rappelée en haut à gauche de la table) ; `CELLCHI2` va faire afficher la contribution de chaque cellule à la statistique du χ^2 (très utile pour l'analyse) ; `DEVIATION` va faire afficher pour chaque cellule l'écart d'effectif par rapport à l'effectif attendu en cas d'homogénéité ; `EXPECTED` va faire afficher dans chaque cellule cet effectif attendu ; `NOPRINT` annule l'affichage de la table, pour ne laisser s'afficher que les résultats des procédures statistiques.

Sauvegarde des résultats, option `OUT=`

Revenons à l'option `OUT=`*table_out* de l'instruction `tables` évoquée plus haut. Elle peut être affublée des options supplémentaires `OUTPCT` et/ou `OUTEXPECT`. La table *table_out* contient toujours les valeurs des modalités des variables définissant la table, ainsi que les effectifs (*frequencies* en anglais) correspondant aux différents croisements de modalités observés. A ceci s'ajouteront les *effectifs attendus* (en cas d'indépendance des deux caractères) si l'option `OUTEXPECT` est utilisée, et les proportions marginales colonnes et lignes avec l'option `OUTPCT` (les variables ainsi créées sont respectivement nommées `expected, pct_row, pct_col`).

Test du χ^2 d'ajustement

La procédure `FREQ` est également utile pour l'analyse statistique des tables à un caractère, car elle permet de mettre en œuvre le *test du χ^2 d'ajustement*, soit à des effectifs cibles, soit à des proportions cibles. Ce test se réalise en utilisant l'une des 2 instructions `tables` *var* `/ TESTF=`*effectifs* ou bien `tables` *var* `/ TESTP=`*props*, où *effectifs* désigne la liste des effectifs cibles, et *props* celle des probabilités/proportions cibles (dans les 2 cas, les valeurs peuvent être séparées par des blancs, et l'option `chisq` est sous-entendue).

29. Il faut préciser dès maintenant qu'on peut faire de même avec 3 variables ou plus, mais les tables sont de dimension > 2 et on n'en parlera pas plus ici ; par exemple pour 3 variables il faut utiliser l'instruction `tables` x`*`y`*`z. Il est aussi possible de croiser une variable avec plusieurs autres, sans croiser ces dernières entre elles : par exemple x`*(`y z`)` va croiser x avec y et x avec z.

30. Ce test de Fisher peut être demandé avec l'option `fisher` de `tables`, pour des tables de dimension plus grande que 2×2.

La procédure FREQ est une des principales procédures statistiques à savoir maîtriser, et à ce titre son manuel d'utilisation figure en bonne place, assorti de nombreux exemples, dans l'aide de SAS :

SAS Products→ SAS Base→ SAS Procedures : CORR, FREQ, UNIVARIATE→ The FREQ Procedure

Exemple 1 : on veut appliquer le test du χ^2 a la table de contingence ci-dessous

	Malade	Sain
Homme	21	125
Femme	18	187

alors il faut creer la table sas suivante

```
data don; do sexe='Homme','Femme'; do etat='Malade','Sain'; input nb @@; output; end; end;
cards; (↪) 21 125 18 187 (↪); run;
```

(formulation equivalente elegante de `data don; input sexe $ etat $ nb @@; cards; Homme Malade 21 Homme Sain 125 Femme Malade 18 Femme Malade 187 (↪) ;`)

et soumettre le code suivant :

```
proc FREQ data=don; weight nb; tables sexe*etat / nopercent nocol chisq; run;
```

Exemple 2 : Pour la table sas *chaussures*, nous voulons determiner la table croisee des variables *zone* et *ventes*, ou la variable *ventes* sera repartie en 4 categories : moins de 200000$, entre 200000 et 500000 $, entre 500000 et 1M $, et plus que 1M $. On procede comme suit avec une procedure FORMAT prealable (de nition d'un format a utiliser pour la variable *ventes*) :

```
proc format; value ventes low-100000='0-100m' 100000-500000='100m-500m' 500000-1000000='500m-1M' 1000000-high='Plus que 1M';
proc freq data=chaussures; format ventes ventes.; tables zone*ventes / out=nbventes chisq nocol nopercent;
proc print data=nbventes; run;
```

Voir a ce sujet la rubrique sur les formats (page ??).

- proc MEANS

La procédure MEANS réalise des calculs que réalise également proc UNIVARIATE, mais donne en quelque sorte une moindre impression de fouillis et va à l'essentiel. En l'occurence, on peut demander à ce que seulement certains indicateurs statistiques soient calculés mais pas les autres (ce n'est pas possible avec proc univariate), et la présentation est différente, en particulier lorsqu'il est fait usage des instructions CLASS ou BY. En revanche, certains indicateurs ne sont pas calculables par proc MEANS, comme par exemple l'écart absolu moyen (MAD).

Un exemple de syntaxe standard est, dans le cas du traitement de 2 variables avec affichage et sauvegarde des moyennes, variances corrigées, bornes d'IdC pour la moyenne, et selon les modalités d'une variable *categ* : est

```
proc MEANS data=don mean var clm ;
var var1 var2;   class categ;
output out=donout mean=moy1 moy2 var=var1 var2 lclm=lclm1 lclm2 uclm=uclm1 uclm2;
```

Dans cet exemple, on voit que les statistiques que l'on désire voir calculées sont indiquées en option de procédure (ici après le champ data=) et codées : ici moyenne (mean), variance (var), et bornes de confiance pour la moyenne (clm). Les codes pour les autres statistiques sont indiqués plus bas.

Les statistiques calculées par défaut sont N, MEAN, STD, MIN, MAX. La procédure means peut aussi calculer : CV (coef variation), STDERR (standard error = écart type de la moyenne empirique = $\sigma/\sqrt{n}$), CLM (confidence limits = interv. de conf. pour la moyenne, de niveau $1-\alpha$ où α vaut par défaut 5%, mais peut être spécifié avec l'option de procédure ALPHA=α), LCLM/UCLM (borne de confiance à gauche/à droite pour la moyenne, de niveau $1-\alpha$), RANGE (étendue), CSS (somme cumulée des carrés), SUM (somme), VAR (variance corrigée), NMISS, MEDIAN, Q1, Q3, P1, ... , P99, QRANGE (étendue interquartiles). L'option VARDEF= permet de spécifier le mode de calcul de la variance ; par défaut c'est DF (càd $n-1$, le nombre de degrés de liberté), mais on peut aussi prendre N (n), ou WDF (pour des observations pondérées). On peut aussi mettre en œuvre des tests de Student pour la moyenne, avec les options T et PROBT= (dans ce cas le niveau est $1-\alpha$, où α est la valeur spécifiée dans le champ ALPHA= (0.05 par défaut).

La syntaxe pour sauvegarder le résulat de proc MEANS dans une table sas est l'instruction OUTPUT OUT=*tableout*, suivie des noms des indicateurs qu'on veut y sauvegarder, ainsi que des noms des variables correspondantes. C'est exactement le même principe que pour proc univariate ; dans l'exemple plus haut, deux variables sont traîtées, donc il faut fournir deux noms de variables à la suite de chaque code (par exemple mean=moy1 moy2 pour les moyennes).

Il faut utiliser les instructions BY ou CLASS pour faire les calculs relativement aux modalités d'une ou de plusieurs variables classifiantes. Les tables en sortie (*i.e.* issues de output) ont alors des structures que je ne détaille pas ici (elles comptent notamment des variables importantes _TYPE_ et _STAT_ par exemple, la première

indiquant le niveau auquel est fait le calcul, et la seconde indiquant la statistique concernée), ces structures étant différentes suivant que `BY` ou `CLASS` est utilisée [31]. Avec l'instruction `BY`, il y a un écran de résultats par modalité de la variable classifiante, tandis qu'avec l'instruction `CLASS` tout figure sur un même écran, et en outre le fichier de sortie va contenir tous les calculs non seulement par modalités mais aussi hiérarchiquement pour les "marginales" (dans l'exemple plus bas, la table `meanchauss2` contiendra les statistiques par zone mais aussi pour la réunion de toutes les zones, càd le niveau au dessus : la variable `_TYPE_` vaut 0 dans ce dernier cas, et 1 sinon).

`MEANS` est une procédure de base qu'il est important de savoir maîtriser, s'entend par là qu'il faut savoir la lancer et produire une table en sortie qui contienne les résultats et qui soit exploitable au sein d'une autre procédure SAS. Un minimum de pratique est nécessaire pour savoir quoi attendre de `univariate` et quoi attendre de `means`. Dans l'aide de SAS, le manuel pour `proc means` est situé ici :

SAS Products→ SAS Base→ SAS Procedures→ Procedures→ The MEANS Procedure

Exemple : pour les données fil rouge *chaussures*

```
proc means data=chaussures mean std; var ventes; output out=meanchauss1;
  title 'proc MEANS mean std; VAR ventes; '; run;
proc means data=chaussures mean std; var ventes; class zone; output out=meanchauss2;
  title 'proc MEANS; class zone;'; run;
proc means data=chaussures mean std; var ventes; by zone; output out=meanchauss3;
  title 'proc MEANS; by zone; (préalablement classé selon zone)'; run;
proc print data=meanchauss2; title 'Table en sortie de proc MEANS; class zone';
proc print data=meanchauss3; title 'Table en sortie de proc MEANS; by zone;'; run;
proc means data=chaussures(where=(zone eq 'Etats Unis')); var ventes;
  title 'proc means data=chaussures(where=(zone eq ''Etats Unis''))'; run;
```

- `proc CORR`

  Cette procédure calcule les corrélations de Pearson classiques, croisées pour des listes de variables données en entrée, mais aussi les corrélations de Spearman et de Kendall (options `SPEARMAN` et `KENDALL`), et des statistiques plus simples comme les moyennes et les variances de chaque variable (à moins d'utiliser l'option `NOSIMPLE`). Pour chaque coefficient de corrélation, la procédure fournit également les p-valeurs associées au test de non-corrélation (dans le cadre de l'hypothèse d'échantillons gaussiens).

  Les variables à traîter doivent figurer en liste en argument de l'instruction `VAR`, et deux commandes intéressantes sont `WITH` et `BY`, la seconde effectuant classiquement les calculs suivant les modalités de *varby*; un exemple d'utilisation de `WITH` est `PROC CORR data=donnees; VAR var1 var2 var3; WITH varI varII; RUN;` dans laquelle seront seulement étudiées les corrélations de var1, var2, var3 avec varI puis avec varII, mais ni var1-var2 ni var1-var3 ni var2-var3 ni varI-varII ne seront étudiées)

  Des options utiles sont `BEST=`b (affiche les b plus forts coefficients de corrélation pour chaque variable, très utiles lorsque les variables sont nombreuses), `OUTP=` (création d'une table de sortie munie d'une variable `_TYPE_`, et contenant la matrice des corrélations), `FISHER` (réalisation de tests d'adéquation de la forme $\rho = \rho_0$ ou unilatéraux), `NOPROB`, `NOMISS`, et `COV` (demande l'affichage de la matrice de covariance, qui par défaut, rappelons le, est calculée avec dénominateur $n-1$)...

  La procédure `CORR` est une procédure statistique fondamentale, et à ce titre son manuel d'utilisation figure en bonne place, assorti de nombreux exemples, dans l'aide de SAS :

  SAS Products→ SAS Base→ SAS Procedures : CORR, FREQ, UNIVARIATE→ The CORR Procedure

- `proc TABULATE`

  Cette procédure, à la syntaxe pas toujours évidente à cerner, permet de réaliser des tableaux présentant de manière synthétique et esthétique des résultats. En combinaison avec le système ODS, il permet d'avoir des tableaux plus jolis et insérables dans des rapports (il suffit pour cela d'inclure, avant `proc tabulate`, la commande `ODS HTML BODY='nom.html'` (où `nom.html` est le nom du fichier html qui contiendra la ou les tables produites par `proc tabulate`, mais en format html), et, après `proc tabulate`, la commande `ODS html close;` (voir chapitre IX à ce sujet).

  Il y a de très nombreuses manières d'utiliser cette procédure, qui possède elle-même des capacités de calculs statistiques intégrés, et le mieux est d'aller voir directement l'aide de SAS et les exemples qu'elle contient

  SAS Products→ SAS Base→ SAS Procedures→ Procedures→ The TABULATE Procedure

- `proc TRANSPOSE`

  Cette procédure, assez difficile à cerner, peut par exemple "remplacer" dans une table SAS les modalités d'une variable *var* par des noms de colonnes (càd des variables ; instruction `ID` *var*) ; l'instruction `BY` *var* permet une réorganisation d'une table SAS, en rassemblant toutes les modalités de la variable *var*... Une page intéressante

31. et dans les deux cas différentes de celle d'une table en sortie d'une procédure `univariate`.

de la toile traîtant de cette procédure est la suivante :

`http://www.ats.ucla.edu/stat/sas/modules/wtol_transpose.htm`

- proc DATASETS

  Cette procédure gère, déplace, copie les tables SAS (voir aussi `proc APPEND`). Des commandes de cette procédure sont `CHANGE` *ancien nom=nouveau nom*, `DELETE` *nom_table_sas*, `RENAME, COPY, MODIFY, LABEL, FORMAT,...` Elle est certainement indispensable pour une utilisation intensive de SAS.

- proc FORMAT

  Un *format* est une manière d'*écrire* des données en sortie de procédure, alors qu'un *informat* est une manière de *lire* des données dans une instruction `INFILE` d'une étape `DATA` ; la procédure `FORMAT` permet de définir un ou plusieurs formats (une instruction `VALUE` *nomformat* par format qu'on veut créer) qu'il faudra attribuer à une (ou plusieurs) variable(s) SOIT globalement au sein d'une commande `FORMAT` d'une étape `DATA` (ou `proc DATASETS`, etc...), soit de manière locale dans une procédure, toujours avec une commande `FORMAT`, par exemple dans les procédures `PRINT, FREQ, TABULATE, GCHART, GPLOT`,...) (dans toute procédure produisant une sortie contenant des valeurs de modalités de variables).

  Exemple : la table `logement` comporte les variables `superficie` et `dept`, la premiere etant continue et la seconde alphanumerique. On veut realiser une table de contingence en rassemblant `superficie` en 4 classes (donc la variable numerique `superficie` devient en quelque sorte categorielle), et `dept` en Paris, petite et grande couronnes.

```
PROC FORMAT;
VALUE $zoneIdF  'PA'='Paris'  'HS','SD','VM'='Petite Couronne'  'SM','YV','ES','VO'='Grande Couronne';
VALUE superf  low-20='0 a 20m2'  20-50='20 a 40m2'  50-80='50 a 80m2'  80-high='plus de 80m2';
*Remarquez que superf est un format numérique;
PROC FREQ data=logement;  TABLES superficie*dept / OUT=freqout;
       FORMAT superficie superf.; FORMAT dept $zoneIdF.;  run;
PROC PRINT data=freqout; run;
```

- proc LABEL

  Un label (libellé) est une sorte d'alias pour nom de variable, un libellé qui remplacera le nom SAS d'une variable donnée dans les sorties de procédures, sans pour autant modifier ce nom SAS ; celui-ci étant souvent trop peu explicite, la possibilité d'utiliser des labels est donc bien utile pour une présentation plus claire ; ces labels peuvent être définis dès le départ au sein d'une étape `DATA`, ou alors de manière éphémère lors d'une procédure, à l'aide d'une instruction `LABEL` *nomvar='labelnomvar'* ; de tels labels peuvent être utilisés dans des procédures `PRINT` (avec l'option `label`), `TABULATE`, et de nombreuses procédures graphiques.

  IMPORTANT ! un label concerne les NOMS de variables, et non pas les modalités des variables, pour lesquelles c'est la notion de FORMATS ou d'INFORMATS qui est adéquate.

exemple :
```
data logement;  input superficie dept @@;  label dept='Département'; cards;
          40 PA  115 SM  ..................... 67 SD;
```

- proc IMPORT

  La procédure `IMPORT` permet de lire des fichiers dans des formats tels Excel ou CSV (*comma-separated values*) et d'en faire des tables SAS utilisables sous SAS. Elle va en fait générer automatiquement une étape `DATA` (lisible dans le log) en : détectant automatiquement les types de variables et la plupart des formats de dates, assignant des longueurs adéquates aux variables littérales, lisant les noms de variables sur la première ligne du fichier (sinon, utiliser l'option `getnames=no`), interprétant deux délimiteurs consécutifs comme une donnée manquante, quelquefois lisant les valeurs entre guillemets en ôtant celles-ci...

  La syntaxe classique comprend les éléments suivant :

```
PROC IMPORT datafile=nomfichier_en_entrée
   OUT=nomtablesas_en_sortie
   DBMS=csv/dlm/excel/access...
   REPLACE=yes/no;
GETNAMES=yes/no;
SHEET=nom de la feuille ciblée;
run;
```

  Cette procédure, indispensable en pratique, est capricieuse et quelquefois délicate à utiliser [32]. L'option `replace` indique à SAS qu'il a le droit d'écraser la table dont le nom est indiqué dans l'option `out=`, si cette table existe déjà (pour une fois, SAS est prudent). L'option `dbms=` permet d'indiquer le type de fichier qui va être importé : CSV, EXCEL, ACCESS,... Suivant la valeur de ce champ `dbms` (*data base management*

32. ce qui n'est guère étonnant, se situant à l'interface de SAS avec des logiciels "étrangers".

system), certaines options ou instructions seront disponibles, voir l'aide à ce sujet. Si `replace` vaut *yes*, alors la table cible écrasera l'éventuelle ancienne. l'instruction `getnames=`*yes/no* indique s'il faut ou non considérer la première ligne du fichier en question comme contenant les noms des variables (ce qui abstient alors d'une instruction `input`). La commande `sheet=...` dans laquelle vous pouvez spécifier le nom de la feuille de calcul que vous souhaitez importer (dans le cas où le fichier excel possède plusieurs feuillets).

On peut aussi faire appel à l'*Import Wizard* (dans l'onglet *File*) : il s'agit d'une manière plus conviviale de préciser ce que l'on désire réaliser comme importation (l'avantage de l'écriture d'un code étant que celui-ci peut être resoumis automatiquement, sans l'aide d'une souris !). Normalement il est possible de récupérer le code qui est produit par l'Import Wizard.

L'importation de fichiers externes pour les transformer en tables SAS est une activité des plus communes dans la vie d'un analyste de données, et n'est qu'une tâche élémentaire parmi celles qui peuvent s'avérer nécessaires. Il n'en sera pas vraiment question ici, et nous renvoyons à l'aide de Base SAS ou SAS/ACCESS, et à la toile, pour approfondir le sujet...

- `proc EXPORT`

  Elle a pour but d'exporter des tables SAS sous d'autres formats ; là encore un *Export Wizard* existe sous Windows, et l'on renvoie au manuel officiel ou à la toile pour plus de détails.

- proc REG

  Procédure très importante et très instructive pour bien comprendre comment les procédures statistiques de SAS sont exploitables, `PROC REG` concerne l'analyse de modèles linéaires par la méthode des moindres carrés. Cette procédure a de très nombreuses potentialités, et nous ne rentrerons pas dans le détail de toutes ici : instructions graphiques intégrées, différentes méthodes de sélection de modèles, calcul de critères d'information, tests linéaires sur paramètres du modèle, régression ridge, utilisation intéractive,...

  Il y a deux instructions importantes dans la procédure `REG` : `MODEL` indique à SAS le modèle linéaire à étudier, et `PLOT` fait tracer des graphiques (il peut y avoir plusieurs instructions `PLOT` par procédure). Bien entendu une instruction `BY` peut être utilisée. Nous revenons plus en détails sur les instructions `model` et `plot` plus loin.

  l'instruction `VAR` permet d'indiquer les variables que l'on souhaite pouvoir utiliser (pour un graphique, ou pour les rajouter plus tard à l'étude) sans qu'elles figurent d'emblée dans le modèle linéaire étudié. l'instruction `WEIGHT` est là pour spécifier l'éventuelle variable contenant le poids des observations, et `OUTPUT OUT=`*table_out ... ;* est la commande permettant de *sauvegarder les résultats de l'étude* dans une table SAS (*la syntaxe est la même que pour* `proc means` *ou* `proc univariate`, voir exemple ci-dessous) : cette table contiendra des informations comme les variables entrant dans l'étude, les valeurs des résidus (`R=`), des prédictions (`P=`), les bornes des intervalles de confiance pour la moyenne du modèle $\mu_i =< x_i, \theta >$ (`LCLM=` et `UCLM=`), les bornes des intervalles de prédiction (`LCL=` et `UCL=`), les résidus studentisés (`STUDENT=`), *leverages* (`H=`),...

  Des options de procédure utiles sont : `OUTEST=` (voir ci-dessous), `ALPHA=` (stipule la valeur de α qui sera utilisée dans tous les calculs statistiques, pour les intervalles de confiance et les tests de signification), `SIMPLE` (fait calculer des statistiques de base pour les variables faisant partie du modèle), `CORR` (idem, mais fait afficher la matrice des corrélations), `NOPRINT` (annule l'affichage des résultats statistiques de la procédure), `GOUT=` (indique où enregistrer les graphiques), `ANNOTATE=` (indique le nom de la table d'annotations graphiques que l'on souhaite voir appliquer aux graphiques générés par les instructions `PLOT`).

  Attardons nous seulement sur l'option `OUTEST=`*table_out* : elle fait *sauvegarder les résultats des estimations des paramètres* dans la table SAS indiquée, mais pour pouvoir inscrire plus d'informations dans cette table il faut utiliser les options `TABLEOUT` (écrit les erreurs-standards, bornes des intervalles de confiance pour les paramètres, valeur du t de Student qui teste la nullité du paramètre, et p-valeur associée), et/ou `COVOUT` (écrit l'estimation de la matrice de covariance du vecteur des paramètres). En outre, tout indicateur dont on demanderait le calcul dans une instruction `MODEL` aurait comme conséquence l'ajout des résultats dans cette table de sortie *table_out* (comme `R2, AIC, MSE`...).

  Attention : il ne faut pas confondre la table de sorties issue de l'option `OUTEST=` (contenant les informations sur les paramètres estimés) avec celle issue de l'instruction `OUTPUT OUT=` (contenant les valeurs des résidus, des prédictions, des moyennes estimées, des bornes des IdC pour les moyennes ou des intervalles de prédiction,...).

  Venons en maintenant à la syntaxe de la procédure, en regardant le cas d'une régression linéaire simple :

  ```
  proc reg data=don <outest=tableoutest><alpha=α> ;
  model y=x </ options>;
  plot y*x </ options>;
  output out=tableout P=predictions R=résidus LCLM=biIdC UCLM=bsIdC LCL=biIdP UCL=bsIdP;
  run;
  ```

  l'instruction `model` y=x indique que le modèle étudié est $y = a+bx+\epsilon$. S'il y avait eu plusieurs variables explicatives (donc régression multiple), la syntaxe aurait été `model y=x1 x2 ... xp` . La variable avant le signe `=` est appelée *variable dépendante* ou *à expliquer*, et les autres *indépendantes* ou *explicatives*. l'instruction `model` peut être munie d'options parmi lesquelles les plus simples sont :

  — `P` : fait afficher les valeurs attendues pour la variable dépendante, autrement dit le vecteur colonne $\hat{Y} = X\hat{\theta}$ ou $\hat{Y} = \hat{a} + \hat{b}x$ dans le cas simple
  — `R` : fait afficher les valeurs des résidus observés, autrement dit le vecteur colonne $\hat{\epsilon} = Y - \hat{Y}$
  — `CLB` : fait calculer et afficher les intervalles de confiance pour les paramètres du modèle
  — `CLM` : fait afficher les intervalles de confiance pour la moyenne de la variable dépendante, pour chaque valeur observée des variables explicatives, càd pour les $\mu_i = (X\theta)_i (= a + bx_i$ dans le cas simple)
  — `CLI` : fait afficher les intervalles de *prédiction* de la variable dépendante, pour chaque valeur observée des variables explicatives
  — `ALPHA=` : permet d'indiquer le niveau des intervalles de confiance, de prédiction, ou celui des tests
  — `SELECTION=`*type* : permet de faire fonctionner un processus de selection de modeles, suivant la méthode indiquée : *forward* (`F`), *backward* (`B`), *stepwise* (`stepwise`),...

D'autres options de la commande `model` complètent utilement cette option `selection` : `maxstep`, `include=`n, `sle, sls`, `best=`, `groupnames=`, `details`...

l'instruction `plot` fait tracer un graphique au cours du traitement de la procédure `REG` : elle a le même fonctionnement qu'une instruction `plot` de `proc gplot` par exemple, mais avec quelques spécificités néanmoins, qui s'avèrent bien pratiques. Il n'est pas question ici d'en explorer toutes les capacités, une balade sérieuse dans l'aide de SAS sera beaucoup plus efficace à cet effet. Les spécificités sont essentiellement l'usage de mots-clefs tels que `P.` ou `R.` pour désigner les variables "prédiction" et "résidus". Par exemple, l'instruction

```
plot (y P.)*x / overlay;
```

va faire tracer sur un même graphique (option `overlay`) les points d'abscisse `x` et d'ordonnée `y`, puis d'ordonnée `P` (valeur de la prédiction $\hat{Y}$ en ce x) ; si on avait écrit quelquechose de la forme `(y1 ... yk)*(x1 ... xm)`, celà revient à demander le tracé des nm graphiques `x`i *vs* `y`j où $1 \leq i \leq m$ et $1 \leq j \leq k$.

D'autres mots-clefs graphiques sont par exemple : `u95.` et `l95.` (bornes sup et inf des intervalles de prédiction de niveau 95%), `u95m.` et `l95m.` (bornes sup et inf des intervalles de confiance de niveau 95% pour les $a+bx_i$), `ucl.`, `lcl.`, `uclm.`, `lclm.` (idem, mais pour le niveau correspondant à la valeur de l'option de procédure `alpha=` ou celle de l'option `alpha=` de la dernière instruction `model` ; attention aux conflits possibles de niveau, SAS ne veut pas traîter plusieurs niveaux dans la même instruction `plot`), `student.` (résidus divisés par leur erreur standard), `npp.` (l'instruction `plot R.*npp.` fait tracer un PP-plot des résidus observés, très utile), `nqq.` (l'instruction `plot R.*npp.` fait tracer un QQ-plot des résidus observés, très utile aussi), `obs.` (le numéro de l'observation), etc...

Des options basiques de l'instruction `plot` sont :

— `overlay` : permet de ne pas changer de cadre graphique si plusieurs tracés sont demandés en même temps : autrement dit on demande la superposition de ces graphiques

— `conf` : fait tracer les courbes correspondant aux bornes supérieure et inférieure des intervalles de confiance pour les moyennes $\mu = X\theta (= a + bx$ dans le cas simple) ; c'est une alternative pratique à l'utilisation des mots-clefs présentés ci-dessus.

— `pref` : idem pour les intervalles de prédiction

— `annotate=` : nom du fichier d'annotations à utiliser

— `description=` et `name=` : mots-clefs utilisables lorsque tous les graphiques sont enregistrés dans un catalogue à l'aide de l'option de procédure `GOUT=` ; la valeur du champ `name` est le nom du graphique au sein du catalogue en question (par défaut, *REG*), et celle du champ `description` est une phrase descriptive qui sera associée au graphique. L'intérêt d'enregistrer les graphiques est de pouvoir les réexploiter plus tard, à l'aide de la procédure `GREPLAY`, par exemple pour en afficher plusieurs côte à côte... Voir l'aide de SAS et le paragraphe VII-5 de ce document.

— des options graphiques spécifiques à `REG` telles que `MSE` (affichage de la MSE en sus de l'équation du modèle et autres statistiques affichées par défaut), `STATFONT, AIC,`...

— les options graphiques classiques, parmi lesquelles `axis, symbol, href, vref, caxis, chref, cvref, cframe,...`

Enfin, `proc REG` peut être utilisée en **mode interactif**, c'est-à-dire, après le lancement de la procédure via l'instruction `run;` , tant qu'une autre procédure n'est pas lancée ou qu'une instruction `quit` n'est pas soumise, on peut soumettre à SAS d'autres instructions de `proc REG` : par exemple demander le tracé d'autres graphes (instruction `plot ...; run;`), ôter ou bien ajouter une variable à l'étude (et donc demander implicitement de relancer l'analyse et les instructions précédentes), ... Voir l'onglet *Details→ Interactive Analysis* du chapitre sur `proc REG`.

Pour clore le sujet, il n'est vraiment pas un luxe de passer du temps à étudier les exemples fournis dans l'aide de SAS au chapitre de `proc REG` (*SAS/STAT user's guide*), car ces instructions se retrouvent dans une forme identique ou très proche dans plusieurs autres procédures statistiques, telles `proc LOGISTIC, GLM,`...

- `proc ANOVA`

Cette procédure permet de réaliser l'analyse de la variance *pour les modèles équilibrés seulement*, càd comportant le même nombre d'observations par modalité du facteur (si un seul facteur), ou par combinaison possible des facteurs (si plusieurs facteurs) ; sinon utiliser plutôt `PROC GLM`. Attention, un certain ordre dans les instructions de cette procédure est requis, par exemple l'instruction `CLASS` doit précéder l'instruction `MODEL`.

l'instruction `CLASS` indique le nom du ou des facteurs de l'ANOVA (on les note dans la suite `A, B, C`,...). l'instruction `MODEL` indique quelle est la variable numérique dépendante qui est étudiée (c'est-à-dire la réponse, dont on étudie les différences de moyennes entre les groupes définis par le ou les facteurs), ainsi que la structure du modèle d'ANOVA qui va être considérée : par exemple `MODEL` varnum = A pour une ANOVA à un facteur A, `MODEL` varnum = A B ou `MODEL` varnum = A B A*B (un synonyme étant `MODEL` varnum = A | B) pour une ANOVA à deux facteurs A et B sans ou avec interaction/effet croisé, ou encore, pour 3 facteurs

avec intéraction, `MODEL` varnum = A | B | C (synonyme de *A B A*B B*C A*C A*B*C*). Enfin, l'instruction `MEANS` *liste des facteurs*, va faire calculer les moyennes de la variable *varnum* pour chacune des modalités ou combinaison de modalités du ou des facteurs ; pour résumer, un exemple d'ANOVA à deux facteurs fact1 et fact2 avec interaction

```
PROC ANOVA data=tablesas;  CLASS fact1 fact2;  MODEL reponse=fact1|fact2;  MEANS fact1|fact2;  run;
```

On peut aussi faire une telle ANOVA avec une instruction supplémentaire `BY`...

- `proc TTEST`

  La procédure `TTEST` effectue des tests de Student pour comparer les moyennes d'échantillons supposés de lois normales (test à 2 échantillons, appariés ou non), ou bien pour tester l'égalité de la moyenne à une valeur de référence donnée (test à 1 échantillon) ; noter que `proc univariate` et `proc means` calculent également la p-valeur d'un tel test. Les syntaxes sont relativement différentes dans les 3 cas (voir plus bas).

  les instructions principales sont les suivantes : `VAR` *var* (indique la variable numérique dont on va étudier la moyenne), `CLASS` *varclass* (indique quelle est la variable (nécessairement à deux modalités !), selon laquelle on essaye de comparer la moyenne de la variable *var*), `BY` *varby* (va faire effectuer un t-test par modalité d'une troisième variable discrète *varby*). Voir aussi les instructions `FREQ, WEIGHT, CI` (intervalles de confiance pour σ),...

  Pour les échantillons appariés, ce n'est pas l'instruction `VAR` qu'il faut utiliser, mais l'instruction `PAIRED` *X*Y*. Dans ce cas il n'y a pas de instruction `CLASS` (voir syntaxe plus bas). L'option de procédure `H0=`m va, dans le cas d'un t-test à un échantillon (*i.e.* une seule variable renseignée dans le champs `var`), indique quelle est la moyenne qu'on essaye de tester ; par défaut, c'est 0.

  Important ! Il ne semble pas exister de commandes permettant de sauvegarder les résultats d'une procédure `TTEST` (ni option `out=`, ni instruction `output out=`).

  La syntaxe standard pour le t-test à 1 échantillon, de conformité à une moyenne μ_0, est

  `proc ttest data=`*don* `H0=`μ_0`; var` X`;  (by` *varclass*`;)`

  La syntaxe standard pour le t-test à 2 échantillons *appariés*, comparant les valeurs de 2 variables *avant* et *après*, est

  `proc ttest data=`*don*`;  PAIRED` *avant*`*`*après*`;  (by` *varclass*`;)`

  La syntaxe standard pour le t-test à 2 échantillons *indépendants*, pour la variable X, est

  `proc ttest data=`*don*`;  var` X`;  class` *categ*`;  (by` *varclass*`;)`

  où la table *don* contient au moins 2 variables, qui sont X (variable numérique d'intérêt), et *categ*, variable ne présentant uniquement que 2 modalités (numérique ou non), qui indique auquel des 2 "échantillons" appartient chaque observation.

- `proc NPAR1WAY`

  La procédure `NPAR1WAY` oc´on

*t*L'abb(o)-28(c)28(viaption)-32[(N)83(PAR1V)111(A)84Y.signdie.(ffnion)-333(paam(tercn)-
procédurnsteriqu262-334(ne)-333(mettan)28(t)-334eon qd'une seule variableóritelle clasrnfiant0.
aacune option PROC NPAR1WAY, lors comreusu262-347(sttistriqu262-3-3es)18erdontcalcunés.62-34

(ltas ea
novalee pour lr uilaitŕal, avc, approauss(enne)-349(ou)-346p-(v)56(ale)-1(e)1rt),

`proc NPAR1WAY data=`*don* `wilcoxon; class` *varclass*`; var` *X*`; exact wilcoxon; output out=`*donout*`; run;`

où *varclass* n'a que 2 modalités, et X est la variable numérique étudiée.

La syntaxe standard pour un test de Kolmogorov Smirnov à 2 échantillons, avec sauvegarde en sortie, est :

`proc NPAR1WAY data=`*don* `edf; class` *varclass*`; var` *X*`; output out=`*donout*`; proc print; run;`

- `proc RANK`

  Avec l'option `OUT=`*tableout* pour la table dans laquelle on veut voir la sortie de la procédure enregistrée, cette procédure détermine les rangs des observations suivant la (ou les) variable(s) numérique(s) en paramètres de l'instruction `VAR`, et peut les assigner à des variables de rangs avec l'instruction `RANKS` ; on peut aussi utiliser une instruction `BY` *varby* pour faire ces calculs pour chaque modalité de la variable *varby*. Exemple :

```
PROC RANK DATA=concours OUT=resultats;
VAR scorejeu1 scorejeu2;
BY categorie;
RANKS rangjeu1 rangjeu2;
RUN;
```

- `proc SCORE`

  Cette procédure calcule des fonctions pondérées à partir d'un fichier de pondérations/scores et d'un fichier de données. C'est une procédure très précieuse en analyse de données : si elle n'existait pas, il faudrait la programmer !

  Supposons par exemple que `don` soit une table SAS contenant des variables numériques $x_1, \ldots, x_K$, et que `scores` soit une table SAS contenant au moins une variable `_TYPE_` (dont une modalité est `SCORE`), une variable `_NAME_` valant toujours la même modalité (disons, "`Facteur1`"), et des variables de même nom $x_1, \ldots, x_K$: une telle table `scores` est par exemple issue d'une procédure telle que `princomp, factor, reg`... La ligne pour laquelle la variable `_TYPE_` vaut `'SCORE'` contient des coefficients associés à chacune des variables x_j : la valeur de la variable x_j pour cette ligne vaut le coefficient qui sera multiplié aux valeurs de la variable x_j pour chaque observation de la table `don`. Appelons $(c_1, \ldots, c_n)$ ce vecteur de coefficients. La procédure `SCORE` va produire une table SAS à une variable dont le nom sera "`Facteur1`", et de même nombre d'observations que `don` : la valeur de cette variable pour la $i$ème observation aura été calculée comme

$$\sum_{j=1}^{K} c_j \, \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

  où $(x_{i1}, \ldots, x_{in})$ sont les valeurs des x_j pour la $i$ème observation, et $\bar{x}_j$ et σ_j les moyenne et écarts-type empiriques de la variable x_j. La procédure renvoie plutôt $\sum_{j=1}^{K} c_j x_{ij}$ (c'est-à-dire sans centrer ni réduire) quand on indique l'option de procédure `NOSTD` (par défaut, la plupart du temps elle centre et réduit implicitement).

  La syntaxe la plus courante est la suivante :

```
proc SCORE data=don score=scores out=scoreout;
var x1-xn;
run;
```

  où `data=` est le champ où indiquer la table des données brutes, `score=` celui pour la table contenant les scores, et `out=` celui pour la table qui contiendra les résultats de la procédure (il n'y a pas de sortie à l'écran pour `proc SCORE`).

  Si la variable `_NAME_` présente plusieurs modalités au lieu d'une seule (ce qui est souvent le cas), alors la procédure `SCORE` va produire une table SAS à autant de variables que le nombre de ces modalités, dont les noms seront égales à celles-ci : la table `scores` comptera en effet plusieurs lignes pour lesquelles `_TYPE_` vaudra `'SCORE'`, chacune contenant les coefficients permettant de calculer le "facteur" de nom la valeur de `_NAME_`.

- `proc STANDARD`

  Cette procédure standardise des données, càd les centre puis les réduit. Une telle manipulation est quelquefois faite implicitement au sein de procédures statistiques plus évoluées (comme en ACP par exemple, avec `PRINCOMP` par défaut). Un exemple d'utilisation est `proc standard data=`*don* `out=donstand mean=0 std=1; run;` pour centrer et réduire, mais on peut prendre d'autres valeurs pour `mean=` et `std=` .

- `proc PRINCOMP`

  La procédure `princomp` (*principal component*) réalise l'analyse en composantes principales de données numériques continues. Sa syntaxe est assez simple :

`proc PRINCOMP data=`*don* `vardef=n out=`*donout* `outstat=`*donoutstat*`;`

`var` *liste_des_variables_à_traîter*`;`

```
run;
```

l'option `vardef=n` signifiant que les variances et covariances seront calculées avec dénominateur n (ce qui est traditionnel en ACP ; par défaut, `vardef=df` comme d'habitude), et les options `out=` et `outstat=` permettant de sauvegarder les résultats de l'ACP :

— la table *donout* spécifiée après `out=` contiendra la table SAS de départ (contenant les individus) *avec en plus les nouvelles variables issues de l'ACP*. Ces variables s'appellent [33] `prin1, prin2,...` et contiennent donc les coordonnées de chaque individu dans le nouveau système de coordonnées défini par les composantes principales.

— la table *donoutstat* spécifiée après `outstat=` contiendra toutes les statistiques calculées à l'occasion de cette ACP : valeurs propres (`_TYPE_=eigenval`), coordonnées des vecteurs propres (`_TYPE_=score`), matrice de corrélation [34] (`_TYPE_=corr`), moyennes, écarts-types.

Il est important de ne pas confondre ces deux tables de sortie, qui n'ont rien à voir au niveau de la nature et de la structure.

On notera que la procédure `princomp` réalise par défaut l'ACP normée (càd en centrant et réduisant les variables de départ), sauf si l'option `cov` est activée, auquel cas il y a centrage mais pas réduction (donc c'est la matrice de covariance qui est alors diagonalisée). Et comme d'habitude, si l'instruction `var` est omise, alors SAS traitera toutes les variables numériques de la table *don* dans l'ACP.

On notera aussi qu'il est possible de donner en entrée, au lieu d'une table SAS de données brutes, plutôt une table SAS qui contient une matrice de corrélation : une telle table doit avoir une variable `_TYPE_`, et les observations de cette table pour lesquelles cette variable vaut 'corr' doivent être les lignes de la matrice de corrélation à traîter.

La procédure `princomp` réalise vraiment le strict minimum de l'ACP (mais inclus le graphe en ébouli des valeurs propres depuis SAS 9.4), et il est le plus souvent indispensable de disposer d'une macro réalisant de nombreuses tâches de manière automatique (tracé du nuage dans les 2 premiers plans factoriels, avec annotation des points si possible, tracé de cercles de corrélation...). Une telle macro sera fournie en TP, après un temps d'adaptation à la procédure de base.

- `proc CLUSTER`

Cette procédure permet de réaliser une classification ascendante hiérarchique (CAH) de données multivariées, disponibles soit sous forme brute, soit sous forme de matrice de distance (ce dernier cas ne sera pas considéré ici). C'est une procédure pour laquelle le temps de calcul explose avec la taille des données, et surtout dont l'utilisation nécessite pas mal de précautions au stade de l'interprétation, car les méthodes de CAH peuvent être sensibles aux différences d'échelles (un prétraitement des données par la procédure `ACECLUS` est préconisé par le manuel de `CLUSTER`). Il est d'usage courant de combiner l'utilisation de méthodes de CAH avec des méthodes agrégatives, mises en œuvre à l'aide de la procédure `FASTCLUS` [35].

Il y a deux champs à remplir dans l'appel à `CLUSTER` :

– le champ `outtree=`, qui indique le nom de la table de sortie qui contiendra les informations permettant de tracer l'arbre issu de la CAH (le dendrogramme), avec la procédure `proc tree data=... <horizontal>; run;`.

– le champ `method=`, qui stipule la méthode à utiliser ; des méthodes disponibles sont `single` (saut minimal), `complete` (saut maximal), `average` (saut moyen), `centroid` , `ward` (méthode de Ward de variance minimum), `eml` (CAH par maximisation de la vraisemblance, sous l'hypothèse de données gaussiennes multivariées sphériques et de même matrice de covariance ; nécessite les options supplémentaires `K=` et `R=`), `twostage`,...

La syntaxe est donc

```
proc CLUSTER data=don  method=méthode  outtree=table_de_sortie ... ;
var liste_des_variables_à_traîter;
copy liste_des_variables_à_recopier;
id variable_identificatrice;
run;
```

où l'instruction `copy` indique le nom des variables (alphanumériques la plupart du temps) que l'on veut voir recopiées dans la table de sortie, mais n'entrant pas dans l'analyse, et l'instruction `id` indiquant la variable dont les modalités identifient chaque individu (ce qui facilite grandement la lecture des résultats, car sans instruction `id` les individus sont identifiés par leur numéro d'observation dans la table d'entrée).

Les options principales sont `STD` (qui centre et réduit préalablement les variables numériques traîtées), `PRINT=`n

33. à moins qu'on spécifie un autre préfixe avec l'option de procédure `prefix=` : par exemple, si `prefix=Y`, alors les nouvelles variables s'appelleront `Y1, Y2,`...

34. ou de covariance si l'option `cov` est utilisée

35. on rappelle que *cluster* signifie *amas* en anglais, et les méthodes de classification non supervisée sont souvent appelées *clustering methods*.

($n = 15$ par défaut, la procédure détaillant les n *dernières itérations* de l'algorithme, disant qui s'agrège avec qui...), et `TRIM=`p (qui retire $p\%$ des observations avant de commencer la classification, les observations ôtées étant sélectionnées en raison d'une estimation faible de la densité supposée ; voir le manuel pour des détails). Les options `NONORM`, `CCC`, `RMSSSTD`, et `NOPRINT` méritent aussi l'attention.

- `proc FASTCLUS`

  La procédure `FASTCLUS` met en œuvre les méthodes agrégatives de classification (dites de type "nuées dynamiques") en utilisant une classification initiale permettant, dans de nombreuses situations, de ne nécessiter ensuite qu'une poignée d'itérations pour atteindre un stade proche de celui "de la convergence". Ce type d'algorithme est préconisé pour le traîtement de grandes quantités de données (il est assez connu que, s'il y a moins d'une centaine d'observations, la classification obtenue a de grandes chances d'être très dépendante de l'ordre des individus dans la table d'entrée).

  La syntaxe usuelle est la suivante :

  ```
  proc FASTCLUS data=don  maxclusters=n  out=sortie1  mean=sortie2  list ;
  var liste_des_variables_à_traîter;
  id variable_identificatrice;
  run;
  ```

  L'option `maxclusters=`n indique le nombre de classes désirées dans la partition, et `out=` et `mean=` permettent d'écrire dans des tables SAS les résultats de la procédure : celle issue de `out=` contiendra les variables d'origine plus 2 nouvelles variables, `cluster` et `distance` (représentant respectivement le numéro de la classe à laquelle a été affectée chaque observation, et la distance qui la sépare du centre de la classe en question), tandis que celle issue de `mean=` contiendra les informations concernant les classes obtenues (coordonnées du centre, nombre d'éléments,...). L'option `list` fait afficher pendant chaque étape les valeurs des variables `cluster` et `distance` sus-citées, et d'autres options peuvent encore s'avérer utiles : `maxiter` borne le nombre maximum d'itérations de l'algorithme (par défaut, 10), `seed=` permet de fournir une table contenant la liste des centres initiaux de l'algorithme, et `least=`p stipule qu'on utilise la distance L^p (par défaut, $p = 2$, c'est-à-dire c'est la distance euclidienne qui est utilisée), `vardef=` indique le mode de calcul des variances, et il existe encore d'autres options dignes d'intérêt suivant la situation.

- `proc CANDISC`

  Cette procédure réalise l'analyse factorielle discriminante (*canonical discriminant analysis*) de données multivariées, en déterminant les variables canoniques, le système de coordonnées qui maximise l'écart entre les centres des classes tout en minimisant les variances intra-classes.

  Sa syntaxe est très simple :

  ```
  proc CANDISC data=don  out=donout  outstat=donoutstat;
  class variable_classifiante;
  var liste_des_variables_à_traîter;
  run;
  ```

  l'instruction `class` stipule le nom de la variable dont les modalités constituent les classes à discriminer ; elle doit être la première instruction de la procédure. La table en sortie de `out=` aura comme observations les observations d'origine, avec les variables d'origine ainsi que les nouvelles variables canoniques nommées par défaut [36] `can1, can2,`... La table en sortie de `outstat=` est d'une toute autre nature, puisqu'elle contiendra toutes les autres informations de l'AFD (en particulier, les coefficients canoniques, les coefficients canoniques standardisés, qui permettent de tirer des conclusions quantifiées de l'AFD).

- `proc DISCRIM`

  La procédure `DISCRIM` est une procédure relativement complexe permettant de mettre en œuvre diverses méthodes de classification supervisée, dont les analyses discriminantes linéaire et quadratique classiques, et la classification par plus proches voisins ou méthodes à noyau. Elle réalise en outre les mêmes calculs que la procédure `CANDISC` décrite précédemment, en proposant les mêmes possibilités de sauvegarde dans une table de sortie. Il ne faut toutefois pas la voir comme la boite à outils ultime pour l'analyse discriminante, car ce sujet majeur a donné lieu à l'élaboration de très nombreuses méthodes plutôt efficaces, qui ne sont en aucun cas couvertes par `proc discrim`. Il faut donc raison garder et son enthousiasme tempérer, même si les méthodes gérées par `proc discrim` ont l'avantage d'être assez simples et plutôt bien connues des utilisateurs.

  La syntaxe de base est exactement la même que pour `proc candisc`, à l'exception de l'option `canonical` qui fait réaliser les calculs de l'AFD et rajouter les variables `can`i à la table de sortie issue de `out=` :

  ```
  proc discrim data=don  out=donout  outstat=donoutstat  canonical;  class varclass;  var ...;  run;
  ```

36. L'option `prefix=` permet de spécifier le nom des variables canoniques qui apparaitront dans la table issue de `out=`, par défaut `prefix=can`.

Cependant, la plupart du temps, une ou plusieurs options supplémentaires seront rajoutées à la syntaxe de base, les plus "importantes" étant décrites ci-dessous :

- `method=` : indique la méthode utilisée, par défaut c'est l'analyse discriminante gaussienne (`normal`), l'autre possibilité étant `method=npar`, une méthode non-paramétrique.
- `k=, r=, kernel=` : ce sont les options qu'il faut préciser dans le cas où `method=npar`, `K=`k est le nombre de plus proches voisins à considérer dans la méthode éponyme, tandis que `r=`r est la fenêtre (*bandwidth*) utilisée en cas de discrimination par estimation à noyau et `kernel=` indique le noyau utilisé (les noyaux possibles sont : `uniform, normal, epanechnikov, biweight, triweight`).
- `pool=yes` *ou* `no` : par défaut, `pool=yes`, c'est-à-dire la matrice de covariance utilisée pour réaliser l'analyse est la matrice globale (analyse discriminante *linéaire*, tandis que si `pool=no`, pour le calcul de la distance à une classe donnée, c'est la matrice de covariance interne à cette classe qui est utilisée (analyse discriminante *quadratique*).
- `list, listerr` : l'option `list` ordonne l'affichage de la liste de toutes les observations, de leur classe d'origine et de la classe à laquelle elles sont réaffectées (si elles ne concordent pas, l'observation est munie d'une astérisque) ; l'option `listerr` ne demande que l'affichage des observations mal réaffectées.
- `crossvalidate, crosslist, crosslisterr` : cette option demande que la méthode de *validation croisée* soit appliquée en plus de la méthode usuelle, et les options `crosslist` et `crosslisterr` sont les analogues de `list` et `listerr` pour cette méthode. Les taux d'erreur par validation croisée sont aussi fournis.
- `outcross=` : permet d'indiquer une table de sortie qui contiendra les observations de départ ainsi que les numéros des classes auxquelles elles sont réaffectées par la méthode de validation croisée.

- `testdata=` : permet d'inclure dans l'analyse un échantillon test, sur lequel sera testé la règle d'affectation obtenue, ce qui fournira le taux d'erreur test dont la différence avec le taux d'erreur apparent permettra de juger de la pertinence de la méthode [37].
- `outd=` : sauvegarde les variables et classes initiales ainsi que les classes de réaffectation et les estimations "*group-specific*" des densités.
- `simple` : affiche moyennes et écarts-types par modalité de la variable classifiante.
- `anova / manova` : réalise une analyse de la variance (resp. multivariée) par variable d'origine.
- `PCOV / TCOV` : fait afficher les matrices de covariance "pooled within-class" ou totale, `PCORR / TCORR` faisant de même pour les matrices de corrélation, et `WCOV` ou `WCORR` le faisant pour chaque classe.

Il y a une instruction importante pour la procédure `DISCRIM`, c'est l'instruction `priors proportional;` (par défaut, `priors equal;` est activée) qui indique que les probabilités *a priori* (dans le modèle d'affectation bayésien) pour la variable de classe sont les proportions de chaque classe dans l'échantillon (voir un cours de statistique pour les détails).

Avant de terminer, il n'est pas inutile de préciser le contenu de la table de sortie issue de `OUT=` : il y a autant d'observations que dans la table de départ, et elle contient la variable classifiante, les variables initiales, les probabilités *a posteriori* (dans le modèle d'affectation bayésien), la variable `_INTO_` contenant les affectations par la règle de resubstitution [38], et enfin les variables canoniques `can1`, `can2`, ... (si l'option `canonical` est utilisée).

- **Quelle procedure SAS pour telle ou telle tâche statistique ?**

Cette question survient souvent lorsque l'on est face à son ordinateur, désemparé devant la machine à gaz que constitue SAS, qui s'est développé par extensions successives, de sorte qu'une tâche donnée peut très bien soit être effectuée par une multitude de moyens possibles, soit seulement d'une et unique façon (soit bien entendu ne peut pas être effectuée, car les fonctionnaités de SAS ne recouvrent pas tout ce que la recherche en statistique permet de faire... en tout cas pas de manière directe).

L'objet de ce paragraphe est juste d'indiquer la marche à suivre pour conduire certains tests usuels ou calculer les statistiques les plus classiques, sachant que la page internet suivante (en anglais) dresse une liste très complète des tâches que peut effectuer SAS :

`http://www.ats.ucla.edu/stat/SAS/faq/whatproc.htm`

sans oublier le manuel du module SAS/STAT, qui inclut des introductions à l'implémentation sous SAS de plusieurs familles de méthodes (régression, analyse discriminante, classification, stat. non paramétrique,

37. Cependant, si l'échantillon test n'est pas fourni initialement dans le lancement de la procédure, cette instruction `testdata=` peut être indiquée dans une procédure `discrim` ultérieure pour laquelle la table d'entrée (dans le champ `data=`) est la table de sortie issue de `outstat=` : SAS reconnait immédiatement qu'il s'agit d'une table SAS spéciale, et choisit le type de l'analyse, linéaire ou quadratique, suivant qu'une observation de cette table a comme valeur `LINEAR` ou `QUAD` pour la variable `_TYPE_`.

38. pour les affectations par validation croisée, c'est la table issue de `outcross=` qu'il faut visionner.

ANOVA, analyse de données catégorielles ou longitudinales, analyse de survie, sans compter les manuels des autres modules statistiques de SAS).

On se contente donc ici de lister les analyses statistiques les plus classiquement croisées lors d'un cursus de statistique :

— *intervalle de confiance pour une moyenne ou une variance* (`proc means` ou `univariate`)
— *test et intervalle de confiance pour une proportion* (instruction `binomial` de `proc freq`)
— *test de Student d'adéquation à une moyenne* (`proc univariate`)
— *test de Student de comparaison de moyennes* (`proc ttest`, cas échantillons appariés ou indépendants)
— *test du signe ou des rangs signés* (calculer la variable "différence" puis traîter cette variable avec `univariate`)
— *test de Wilcoxon-Mann-Whitney de la somme des rangs* (`proc npar1way`)
— *analyse de la variance* (`proc anova` ou bien, pour le *test de Kruskal Wallis*, `proc npar1way`)
— *test du* χ^2 *d'adéquation* (`proc freq`)
— *test du* χ^2 *d'indépendance/d'homogénéité* (`proc freq`)
— *test de Kolmogorov-Smirnov d'adéquation* (calcul au sein `proc univariate`, affichable via une instruction `inset`, voir paragraphe VII-6-b ; dans le cas de l'adéquation à une loi normale, utiliser l'option `normal` de cette procédure)
— *régression linéaire simple ou multiple* (`proc reg`)
— *régression logistique ou polytomique ordinale* (`proc logistic`)
— *régression binômiale ou poissonienne* (`proc genmod`)
— *test de bruit blanc* (`proc spectra` et d'autres instructions de procédures de SAS/ETS)
— *test d'homogénéité des variances* (option `hovtest=` de l'instruction `means` de `proc glm`, voir l'onglet "Comparing Groups" des pages "Details" du manuel de `proc glm`)
— *estimation à noyau de la densité ou de la régression* (`proc kde` de SAS/STAT)

- Autres procedures statistiques avancees

Dans l'état actuel de ce document, il n'a pas été possible d'inclure des informations sur la mise en œuvre et la syntaxe des procédures statistiques avancées ci-dessous ; cependant, pour référence, j'indique en deux mots la signification des noms de procédures suivants, procédures couramment utilisées en pratique ; voir directement l'aide de SAS/STAT à leur sujet.

⋆ `LOGISTIC` (régression logistique pour réponses binaires ou ordinales)
⋆ `CORRESP` (analyse des correspondances)
⋆ `FACTOR` (analyse factorielle)
⋆ `GENMOD` (modélisation par *modèles linéaires généralisés*, incluant modèles logit et probit, log-linéaires pour données multinomiales...)
⋆ `GLM` (modélisation par *modèles linéaires généralisés* par la méthode des moindres carrés)
⋆ `MIXED` (généralisation des capacités de modélisation de `PROC GLM`, pour prendre en compte des effets mixtes)
⋆ `CATMOD` (modélisation linéaire, log-linéaire, régression logistique, analyse de mesures répétées ; de manière générale, gère la modélisation de *données catégorielles*)
⋆ `KDE` (réalise des estimations de densités uni- ou bi-variées par la méthode des noyaux)
⋆ `LIFEREG` (analyse de données de survie censurées, sous modèle linéaire)
⋆ `NLIN` (modélisation additive non-linéaire par moindres carrés)
⋆ `PROBIT` (modéles de régression probit, logit, logistique discrête, et gompit (fonction de lien de type "loi des valeurs extrêmes"))

VI. Utilisation basique du module SAS/ETS

(Ce chapitre est censé décrire la syntaxe de base de la procédure `ARIMA`, qui permet de mettre en œuvre la méthodologie initiée par Box & Jenkins dans l'étude des séries chronologiques. J'ai rédigé un document spécifique plus détaillé (6 pages), que je n'ai pas inclus dans ce manuel car son "ton" est différent de celui de ce manuel et il traite d'un sujet bien spécialisé. Ce document spécifique SAS/ETS est téléchargeable sur ma page web.)

VII. Réalisation de graphiques sous SAS

ATTENTION ! Ce chapitre est un peu daté, il ne traite pas des procédures modernes de tracé de graphes que sont les procédures `SGPLOT, SGPANEL,`... Il contient cependant des informations toujours intéressantes.

Les graphes dont nous allons surtout parler ici sont ceux qui peuvent être tracés :

— soit à l'aide de **procedures graphiques** telles que `GPLOT` (la plus courante, pour graphes en lignes brisées, nuages de points, splines interpolants,...), `G3D` (graphes en 3 dimensions) ou `GCHART` (diagrammes en bâtons pour tableaux de bord),

— soit à l'aide de **instructions graphiques** telles que l'instruction `PLOT` de la procédure `REG`.

On peut aussi tracer des **histogrammes** (instruction `HISTOGRAM` de la procédure `UNIVARIATE`), des **boites de dispersion** (instruction `BOXPLOT` de cette même procédure, ou bien procédure `BOXPLOT`, ou encore via l'utilisation d'une certaine option de tracé), des graphes en étoile ou en sections, des colorations de cartes ...

Ces graphes peuvent être embellis et annotés de diverses manières, ils peuvent aussi être *enregistrés* pour être reproduits en association avec d'autres graphes (procédure `GREPLAY`). Dans ce document nous expliquerons notamment comment adjoindre des labels/étiquettes aux divers points d'un nuage : ce procédé d'**annotation** est souvent nécessaire en stats, mais sa mise en place est bizarrement un peu lourde, ce qui est frustrant (une meilleure prise en main de cette simple fonctionnalité aurait été la bienvenue).

Par ailleurs, certaines procédures graphiques (ou instructions graphiques de procédures statistiques) sont compatibles avec le système ODS (cf paragraphe IX-2), ce qui s'avère très pratique à l'usage [39].

1. Concepts et options graphiques

Commençons par expliquer comment SAS trace des graphes, en nous limitant ici aux nuages ou graphes plans.

Supposons que l'on veuille tracer un graphe plan ou un nuage de points impliquant deux variables d'une table SAS, que nous appellerons respectivement `x`, `y`, et `don`. Le graphe SAS proprement dit n'est en fait rien d'autre que le placement de points dans une zone graphique :

* pour chaque observation de la table `don`, un point est placé aux coordonnées indiquées par les valeurs de `x` et `y` pour cette observation (par exemple via une instruction `plot y*x` d'une procédure graphique ou statistique).
* selon ce que l'on indique à SAS, ces points sont reliés (par un segment ou interpolés d'une autre manière) ou non (dans le cas d'un nuage de points), et à leur emplacement figure un sigle (rond, carré, triangle, croix, étoile,...) ou non. La couleur et l'aspect des points et de la courbe interpolante sont stipulés à SAS à l'aide d'une commande globale `SYMBOL`, qui est décrite en détails plus loin.

D'autres commandes globales [40] permettent de définir l'aspect et les noms des axes (`AXIS`), ajouter des titres ou des notes aux graphiques (`TITLE` et `FOOTNOTE`), des légendes (`LEGEND`), des ombrages (`PATTERN`), et d'autres options (procédure ou instruction `GOPTIONS`). La connaissance de ces commandes globales est absolument nécessaire pour obtenir un résultat final satisfaisant.

Il ne serait pourtant pas logique de décrire la syntaxe de ces commandes avant de montrer comment on trace un graphique, aussi cette description est reportée au dernier paragraphe de ce chapitre.

2. La procedure `GPLOT`

Il s'agit de la procédure de base pour tracer des courbes et des nuages de points en deux dimensions. Elle n'a que très peu d'options :

— option `gout=`*nomdecatalogue* : permet d'enregistrer nommément le graphe produit par la procédure, pour une exploitation future dans une procédure `greplay` (voir paragraphe VI-5.).

— option `uniform` : permet notamment d'avoir la même étendue dans l'axe des abscisses lorsque plusieurs graphes différents sont réalisés au sein de la même procédure (par exemple en raison d'une instruction `BY`).

— option `annotate=`*table_sas_d'annotation* : fait appliquer la table SAS d'annotation spécifiée à tous les graphes réalisés par la procédure ; voir le paragraphe VI-4 sur l'annotation pour plus de détails.

Il n'existe quasiment qu'une seule instruction dans la procédure `GPLOT`, l'instruction `PLOT` : elle peut être invoquée avec diverses syntaxes :

39. voir l'onglet de l'aide *SAS Products → SAS/GRAPH → SAS/GRAPH Reference → Bringing SAS/GRAPH Output for the Web*
40. *SAS/GRAPH Statements*

```
proc gplot data=don;   plot y*x;  run;
```

va tracer le nuage de points de coordonnées (x_i, y_i) (remarquez l'ordre *variable ordonnée* $*$ *variable abscisse*) ;

```
proc gplot data=don;   plot y*x=categ;  run;
```

va faire la même chose, mais en colorant les points suivant la valeur que prend la variable `categ` (considérée comme catégorielle) pour chaque observation, et en insérant une légende ;

```
proc gplot data=don;   plot (y1 y2 y3)*(x1 x2)=categ;  run;
```

va tracer les 6 nuages de points d'abscisses et d'ordonnées respectives `y1*x1`, `y1*x2`, `y2*x1`, `y2*x2`, `y3*x1`, et `y3*x2`, avec coloration suivant les modalités de la variable `categ`.

Par défaut, les points sont tracés avec des gros points noirs, et aucune interpolation n'est réalisée. Si l'on veut changer cela, il faut utiliser la commande globable `SYMBOL`, décrite plus en détails plus loin, mais dont on fournit ici un exemple d'utilisation : si l'on veut tracer une trajectoire de série chronologique (variables `date` en abscisse et `production` en ordonnée), en pointillé rouge assez épais avec des gros carrés bleus en guise de "points", il faut soumettre

```
symbol i=join l=2 w=3 c=red v=square h=3 cv=blue;
proc gplot data=don;   plot production*date;  run;
```

On remarque donc que *tout* ce qui concerne l'aspect du tracé (couleur, type de ligne, type de tracé, épaisseur, type et aspect des points) se spécifie en utilisant une option `SYMBOL`, *hors* de la procédure `GPLOT` proprement-dite, qui elle, se contente de spécifier quelles données et quelles variables utiliser pour tracer les graphiques.

Nous évoquons maintenant les principales options de l'instruction `PLOT`.

— option `overlay` : permet de tracer sur la même fenêtre graphique tous les graphes commandés par l'instruction `plot` (par exemple `plot (x1 x2 x3)*y` ou `plot x1*y x2*z x3*t`). Noter que cela ne fonctionne pas avec une instruction du style `plot (x1 x2 x3)*y=categ`, et qu'aucune légende n'est tracée par défaut.

— option `annotate=`*table_sas_d'annotation* : fait appliquer la table SAS d'annotation spécifiée au(x) graphe(s) réalisé(s) par cette instruction `plot` (voir paragraphe VII-4).

— option `cframe=`*couleur* : spécifie la couleur de l'arrière plan du ou des graphiques en question. Remarquer que cet arrière plan est entouré d'*un cadre*, par défaut, mais que celui-ci peut être supprimé via l'option `noframe`.

— option `href=`*valeur(s)* (resp. `vref=`*valeur(s)*) : fait tracer une ou plusieurs lignes de "référence" verticale(s) (resp. horizontale(s) [41]). Si l'on veut faire tracer plusieurs lignes, les valeurs suivant le signe `=` sont seulement à séparer par des espaces (par exemple `vref=10 12 15`). La couleur de la ou des ligne(s) de référence est par défaut noire, mais on peut la spécifier à l'aide des options `chref=` et `cvref=`, suivies d'une couleur. En outre, l'option `grid` fait tracer des lignes de référence pour chaque graduation de chaque axe du graphe (traçant par conséquent une grille ; on peut faire des tracer de telles lignes uniquement horizontales ou uniquement verticales avec, respectivement, les options `autohref` et `autovref`).

Le type de ligne (continue ou en pointillés) est à spécifier avec les options `lhref=`n et `lvref=`n, la valeur $n = 1$ signifiant trait continu, et toutes les valeurs $n \geq 2$, divers traits en pointillés.

— options `haxis=`*liste_de_valeurs* (resp. `vaxis=`) : permet de stipuler les valeurs de l'axe au niveau desquelles seront tracées les graduations. Par liste on entend une liste de valeurs séparées par des espaces, ou alors une syntaxe du genre `0 to 10000 by 2000`. Le nombre de sous-graduations entre deux graduations est à stipuler avec les options `hminor=` et `vminor=`.

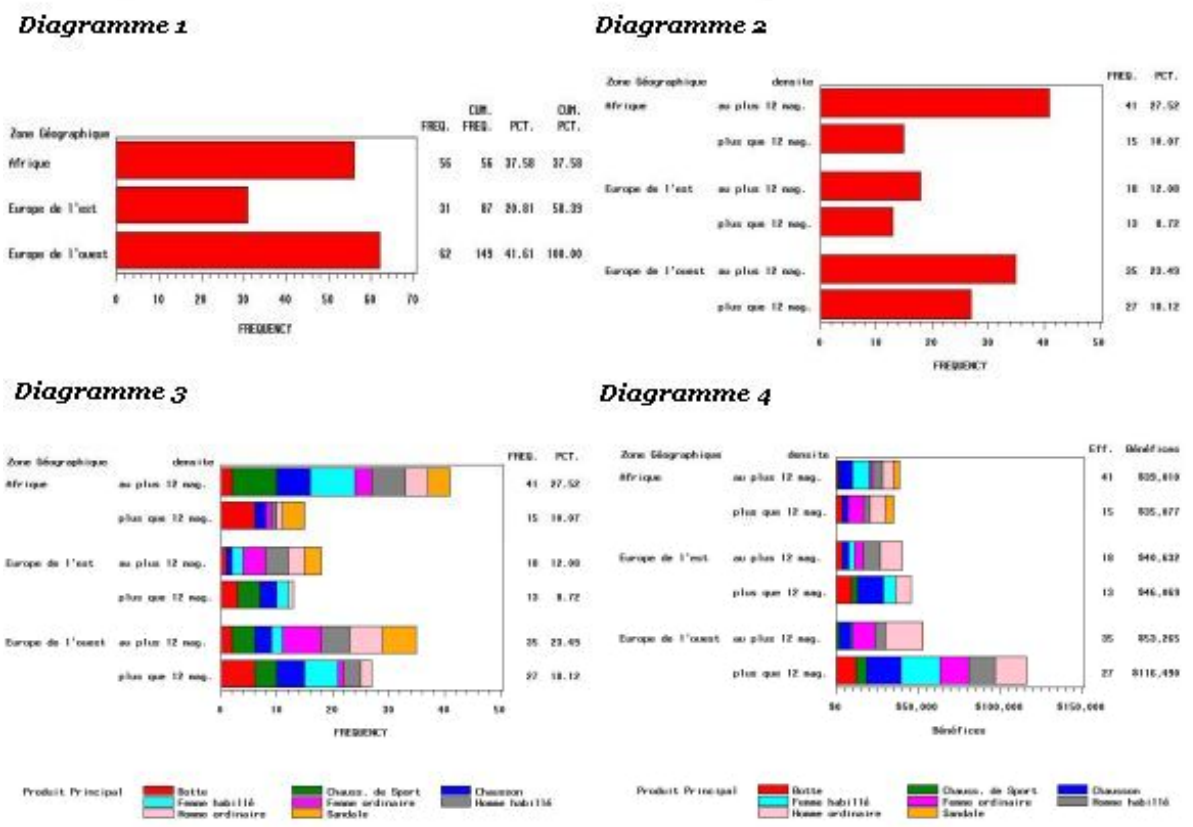
— options `haxis=AXIS`j (resp. `vaxis=`) : c'est en fait la syntaxe la plus courante, dans laquelle tous les détails concernant les axes sont indiqués au sein d'une commande globale `AXIS`. Voir le paragraphe VII-7-b.

3. La procedure `GCHART`

La procédure `GCHART` permet de réaliser des diagrammes (en bâtons [42]), verticaux ou horizontaux, avec un nombre confortable d'options permettant de les mettre en forme de la manière la plus adaptée à chaque situation. Dans ce paragraphe, nous nous concentrerons sur les instructions `hbar` (surtout) et `block` ; les autres instructions fonctionnent de manière très similaire (`hbar3d, vbar, vbar3d, block3d, pie, star`...).

Pour découvrir les potentialités et la syntaxe de la procédure `gchart`, des illustrations préalables sont indispensables. Voici par exemple 4 diagrammes produits à partir de la table SAS *chauss* :

41. si si ! dans ce sens là...
42. que les gens non avisés (et le tableur Excel !) qualifient d'histogrammes, ce qui est un terme totalement inadapté...



Ces diagrammes ont été obtenus via le code suivant :

```
proc gchart data=chauss(where=(zone in ('Europe de l''ouest' 'Afrique' 'Europe de l''est')));
hbar zone;
hbar densite / group=zone freq percent;
hbar densite / group=zone subgroup=produit freq percent;
hbar densite / group=zone subgroup=produit sumvar=benefices type=sum freqlabel='Eff.';
run;
```

On rappelle que chaque observation de la table `chauss` est notamment définie par une zone géographique (`zone`), un type de produit (`produit`), une variable indiquant si le nombre de magasins est > 12 ou non (`densite`), et la variable numérique `benefices`. Ici on s'est contentés de considérer les continents africain et européen.

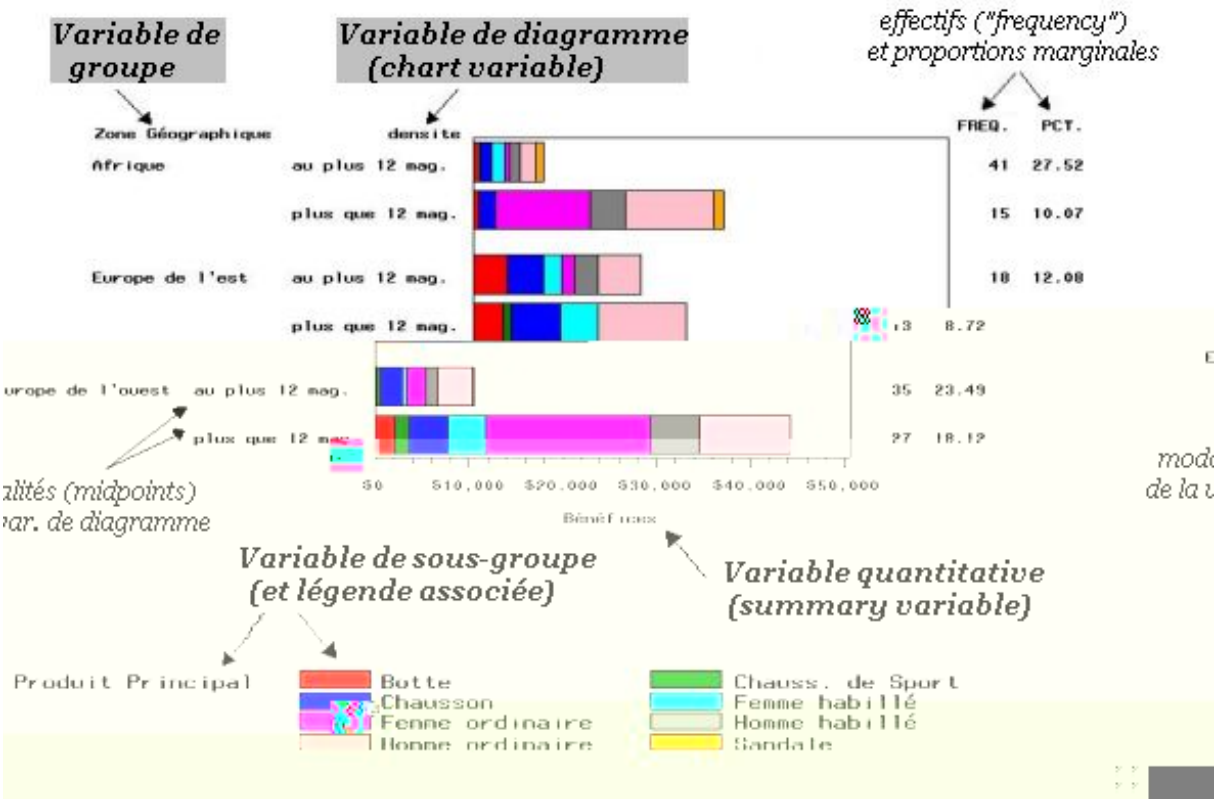
Le premier diagramme représente le nombre d'observations (l'effectif, appelé *frequency* en anglais) par zone. La variable de diagramme (chart variable) est alors `zone`, et ses modalités sont appelées *midpoints* par SAS. A droite du diagramme sont rappelés les effectifs, ainsi que les effectifs cumulés, les proportions (et proportions cumulées) correspondantes.

Le second diagramme fait de même, mais cette fois en différenciant suivant la quantité de magasins (> 12 ou non). La variable de diagramme devient alors `densite`, et la variable `zone` est dite variable de groupe ; elle est la plus à gauche sur le diagramme, et il y a alors une barre par modalité (*midpoint*) de la variable de diagramme et par modalité de groupe. On remarquera que l'ajout des options `freq` et `percent` a fait disparaître l'affichage des effectifs et proportions cumulés (qui a lieu par défaut).

Le troisième diagramme fait intervenir une troisième variable catégorielle, la variable `produit`, qui est qualifiée de variable de sous-groupe. Chaque barre est alors saucissonée en fonction des modalités de cette variable, et une légende des couleurs apparait sous le diagramme. On interprète alors la longueur de chaque morceau de barre comme le nombre d'observations qui satisfont aux modalités correspondantes des 3 variables catégorielles en question (sous-groupe, diagramme, et groupe).

Le quatrième diagramme a la même structure, mais cette fois les longueurs des barres ne sont plus des effectifs mais correspondent à une 4ème variable, numérique cette fois ci, appelée par SAS summary variable. Ici il s'agit de la variable `benefices`, et la longueur est soit la somme, soit la moyenne, des valeurs que prend la variable `benefices` parmi toutes les observations réalisant les modalités correspondantes des variables catégorielles. Pour notre cas, c'est la somme qui a été calculée (option `type=sum`), et la somme totale est reportée en colonne de droite à la place des proportions (à condition que l'option `percent` ait été ôtée).

L'image ci-dessous résume les différents éléments de vocabulaire de la procédure `GCHART` que nous venons de voir :



Voyons maintenant comment créer de tels diagrammes avec une variable *continue* en guise de *chart variable* (ce qui a comme résultat une sorte d'"histogramme") ? Pour un diagramme à barres verticales, une syntaxe possible est d'utiliser l'option `midpoints=` suivie par exemple de la liste `25 35 45` ou de `20 to 50 by 10` (dans ce cas, seules les observations pour lesquelles la variable continue en question vaut entre 20 et 50 seront concernées) ; une autre syntaxe possible est de spécifier le nombre de *midpoints* avec l'option `levels=`*n*, SAS se chargeant de la définition des classes. On peut spécifier (pour un `vbar`) la largeur des barres, avec l'option `width=`, toutefois, ce n'est pas une bonne idée de tracer des histogrammes avec `gchart`, utiliser plutôt l'instruction `histogram` de la procédure `univariate` (cf plus bas).

Pour terminer, parlons d'un problème qui se pose souvent, celui de l'ordre des modalités de la variable diagramme. Par exemple, si la variable a comme modalités les jours de la semaine, alors SAS va présenter les modalités dans l'ordre alphabétique (soit *Dimanche, Jeudi, Lundi,...*). Pour lui imposer un ordre, il faut soit l'indiquer explicitement avec une option `midpoints` (`midpoints='Lundi' 'Mardi' 'Mercredi'...`) soit que la variable en question soit formatée, auquel cas l'ordre des valeurs dans le format est conservé (on peut aussi utiliser le paramètre `order=` dans une instruction `axis`).

Il existe de nombreuses autres options (d'ordre esthétique pour la plupart), se référer à l'aide de SAS/GRAPH.

4. Le procede d'annotation

Il est très souvent utile de pouvoir adjoindre aux points d'un nuage de points, des étiquettes. Par exemple si une table SAS fournit les poids, tailles, et noms d'enfants, on aimerait que dans le nuage de points, chaque point puisse être immédiatement identifié par le nom de l'enfant ; mais comme chaque enfant a sans doute un nom qui lui est propre, il n'est pas envisageable de légender le graphe avec une couleur de point par enfant via l'instruction `plot poids*taille=nom;` . On a envie que le nom de l'enfant soit écrit à côté du point qui lui correspond, et ceci est rendu possible par le biais du systeme d'annotation de SAS (remarque : celui-ci ne se borne pas à cette problématique, mais nous nous limiterons à elle dans ce paragraphe).

Le principe est le suivant : lorsqu'un graphe est tracé, il va être possible d'y superposer des *éléments graphiques supplémentaires* (des traits, flèches, du texte, des cercles...), dont le détail sera contenu dans une table SAS particulière appelée table d'annotation : celle-ci aura été créée *préalablement* au tracé du graphe proprement dit, et sera appelée au moyen de l'option `annotate=`.

Pour en revenir à l'exemple des enfants et du nuage de points, la table d'annotation va avoir autant d'observations que la table SAS des enfants, et aura comme variables : le nom de l'enfant à apposer, les coordonnées des points où il faudra l'apposer (en l'occurence ici, ses poids et taille), le nom de l'action d'annotation (ici, apposer du texte), la position relative du texte par rapport au point (au dessus, à gauche, en bas à droite,

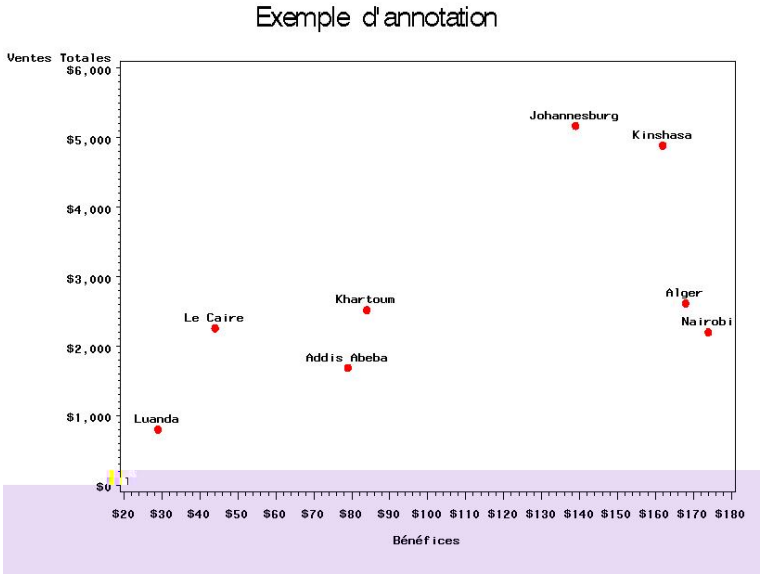
etc...),...

Nous allons maintenant voir un exemple de ce type, mais avec les données `chauss` utilisées jusqu'ici. Le but est de représenter le total des ventes et les bénéfices de l'ensemble des magasins de chaussures de sport pour chaque ville d'afrique, en indiquant le nom de la ville sur le graphe. Le code est le suivant :

```
data annotchauss;
set chauss(where=(zone='Afrique' and produit='Chauss. de Sport'));
 xsys='2'; ysys='2';        * code pour déterminer les axes... ;
 x=benefices; y=ventes;     * ce sont les positions des points ;
 function='label';          * indique que l'action à effectuer est d'apposer du texte au graphe;
 text=ville;                * indique la variable contenant le texte à apposer ;
 position='2';              * position autour du point, ici '2'= au dessus ;
 size=1;                    * taille de ce qui est trace, ici du texte ;


symbol v=dot i=none cv=red;
proc gplot data=chauss(where=(zone='Afrique' and produit='Chauss. de Sport'));
 title 'Exemple d''annotation';
 plot ventes*benefices / cframe=brown annotate=annotchauss;
run; quit;
```

et le résultat est comme ci-dessous :



Attention cependant, le système d'annotation de SAS est aussi flexible et intéressant que particulièrement délicat (et parfois pénible) à manipuler, sa syntaxe étant très sensible à la moindre erreur.

5. La procedure `GREPLAY`

Cette procédure permet de réexploiter des graphes qui ont été préalablement tracés ET sauvegardés : l'utilisation de base de cette procédure relativement complexe consiste à rassembler plusieurs graphes sur une même "fenêtre" graphique. L'idée est la suivante : créer un **catalogue graphique SAS** dans lequel stocker les graphiques tracés et sauvegardés (avec les options `gout=` et `name=` des procédures ou instructions graphiques), puis définir un "*template*" graphique qui va accueillir les différents graphes en question, via la procédure `greplay`.

Nous nous contenterons ici de montrer un exemple d'enregistrement puis rassemblement de 4 graphiques.

```
proc reg
 data=chauss(where=(zone in ('Europe de l''ouest' 'Afrique' 'Europe de l''est')))
 gout=goutchauss;                                    * creation du catalogue graphique "goutchauss" ;
 title 'Regression lineaire de benefices contre ventes totales';
 model benefices=ventes / alpha=0.10; run;           * realisation de l'ajustement lineaire;
 output out=regchauss R=R P=pred lclm=lclm uclm=uclm;  * sauvegarde des resultats dans une table SAS;
run; quit;                                           * on ferme la procedure interactive REG;

symbol;
proc gplot data=regchauss gout=goutchauss;          * declaration d'utilisation du catalogue via gout= ;
```

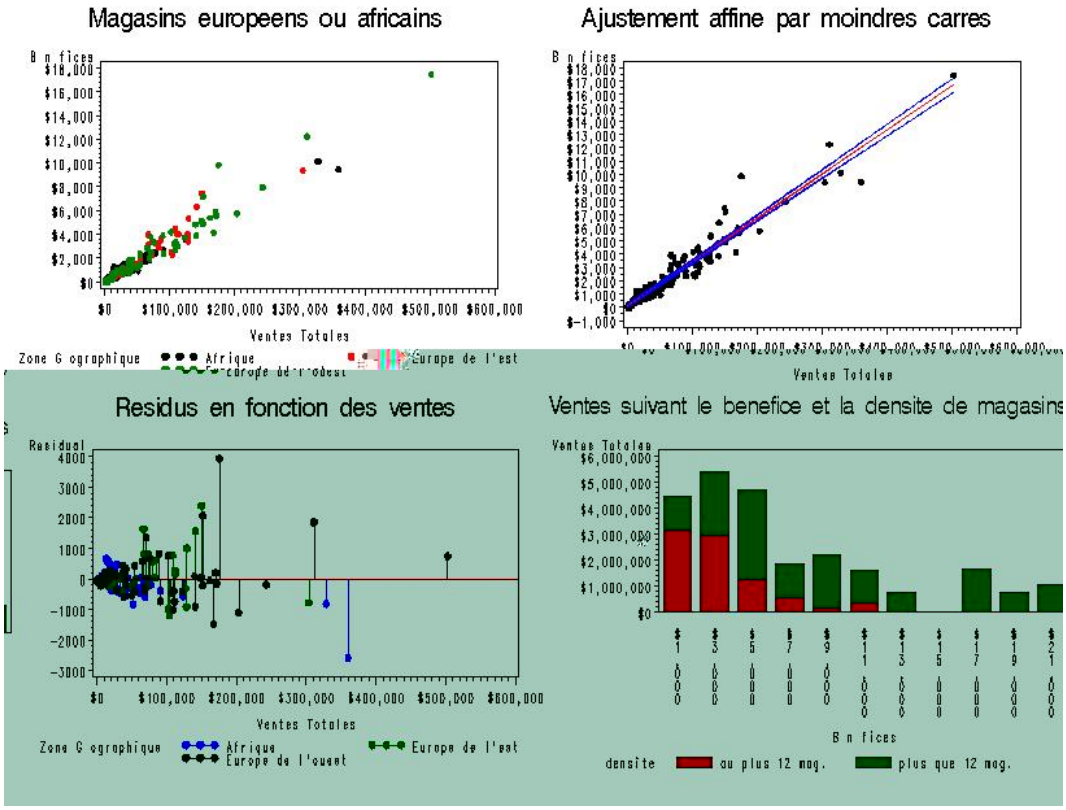
```
 symbol i=none v=dot h=0.8;
 title 'Magasins europeens ou africains';
 plot benefices*ventes=zone / name='nuage';            * premier trace de graphique et sauvegarde sous le
run;                                                   * nom "nuage" dans le catalogue declare precedemment;
 symbol1 i=none v=dot c=black;                         * on remarque que la commande "run;" a ete soumise,
 symbol2 i=join c=red v=none;                          * ce qui permet de mieux gerer l'attribution des
 symbol3 i=join l=2 c=blue v=none;                     * options Symbol...
 symbol4 i=join l=2 c=blue v=none;
 title 'Ajustement affine par moindres carres';
 plot benefices*ventes=1 pred*ventes=2 (lclm uclm)*ventes=3  / overlay name='ajust';
run;                                                  * 4 graphes en un seul, enregistre sous le nom "ajust";
 symbol;
 symbol1 i=needle c=blue v=dot h=1;
 symbol2 i=needle l=1 c=green v=dot h=1;
 symbol3 i=needle l=1 c=black v=dot h=1;
 title 'Residus en fonction des ventes';
 plot R*ventes=zone / cvref=red vref=0 name='resid';  * le graphe des residus, enregistre sous le nom "resid";
run; quit;                                            * la encore on quitte la procedure interactive gplot;

proc gchart data=chauss gout=goutchauss;
 title 'Ventes suivant le benefice et la densite de magasins';
 vbar benefices / subgroup=densite sumvar=ventes type=sum inside=sum name='vbar';
run; quit;                                            * trace d'un diagramme en bâton et sauvegarde ("vbar");

proc greplay nofs;
 igout goutchauss;
 tc tempcat ;
 tdef ensemble Des='4 graphes ensemble'
   1/LLX=0   LLY=50 ULX=0  ULY=100 URX=50  URY=100  LRX=50  LRY=50
   2/LLX=50  LLY=50 ULX=50 ULY=100 URX=100 URY=100  LRX=100 LRY=50
   3/LLX=0   LLY=0  ULX=0  ULY=50  URX=50  URY=50   LRX=50  LRY=0
   4/LLX=50  LLY=0  ULX=50 ULY=50  URX=100 URY=50   LRX=100 LRY=0;
 template ensemble;
 treplay 1:nuage 2:ajust 3:resid 4: vbar;
run; quit;
```

Expliquons un peu la structure de la procédure `greplay` ci-dessus[43] : d'abord on indique avec l'instruction `igout` dans quel catalogue graphique (ici `goutchauss`) il va falloir aller chercher les enregistrements de graphes. Ensuite on doit définir un catalogue à *templates* (ici `tc tempcat;`), puis le *template* qui va accueillir les graphes. Il est appelé ici `ensemble`, et est constitué de 4 morceaux rectangulaires dont la position dans la "fenêtre graphique" doit être définie : les coins inférieur gauche et supérieur droit de la fenêtre étant de coordonnées respectives $(0, 0)$ et $(100, 100)$ (en %), alors par exemple les coins inférieur gauche (lower-left=LL) et supérieur droit (upper-right=UR) du morceau 1 ont comme coordonnées respectives $(0, 50)$ et $(50, 100)$, autrement dit le morceau 1 est le quadran supérieur gauche de la fenêtre. Les deux dernières instructions indiquent quel *template* utiliser (`template ensemble`) et quel élément du catalogue assigner à quel morceau du *template* (instruction `treplay`). On dit alors qu'on a *rejoué* (*replay*) des graphiques et le résultat final est le suivant :

43. l'option `nofs` indique que la procédure `greplay` sera menée à bien de manière non-interactive, càd sans lancement d'une fenêtre dédiée dont on pourra remplir les champs ; voir de toute façon l'aide de SAS pour plus de détails sur ce sujet et cette procédure...



On peut noter que cette procédure `greplay` permet aussi de **superposer** des graphes : il suffit pour cela de définir de la même façon plusieurs morceaux [44] du `template`.

44. par exemple `LLX=0 LLY=0 ULX=0 ULY=100 URX=100 URY=100 LRX=100 LRY=0` pour la fenêtre graphique entière.

6. Quelques autres types de graphiques

Il n'y en a qu'un petit nombre, dont seule une poignée est susceptible d'être fréquemment utilisée. Les autres modules de SAS (comme QC-*Quality Control* par exemple) procurent leurs propres potentialités graphiques spécialisées, mais tous requièrent le module SAS/GRAPH pour fonctionner de manière optimale.

a. Procedure `G3D`

Elle permet de tracer des surfaces ou des nuages de points dans l'espace, avec ses deux seules instructions `plot` et `scatter`. Elle est fortement configurable, et on se contentera ici d'en indiquer la syntaxe de base.

Pour tracer une surface basée sur 3 variables numériques continues `x,y,z` d'une table SAS `don`, la syntaxe de base est

```
proc g3d data=don;  plot y*x=z;  run;
```

des options intéressantes étant `rotate=`, `tilt=` (pour pivoter la surface de différentes façons), et `side`.

Pour tracer un nuage de points de coordonnées `x,y,z`, l'instruction est `scatter`

```
proc g3d data=table_SAS;  scatter y*x=z;  run;
```

Par défaut, le symbole des points du nuage est une pyramide (les symboles pour les nuages 3D sont différents de ceux des nuages plans) de couleur verte, reliée au plan $x * y$ par une barre verticale. Pour tracer un nuage avec symbole et couleur différents suivant la valeur d'une variable catégorielle `cat`, il faut ajouter à la table SAS deux variables qui indiqueront le type et la couleur du symbole de chaque point-observation, et utiliser les options `shape=` et `color=` de l'instruction `scatter`. Voici un exemple dans lequel la variable `cat` prend les valeurs `a,b,c` :

```
data donscatter;  set don;
select cat; when 'a'  do; coul='black'; forme='pyramid'; end;
            when 'b'  do; coul='red'; forme='flag'; end;
            when 'c'  do; coul='blue'; forme='cube'; end;  end;
proc g3d data=don;  scatter y*x=z / shape=forme color=coul;  run;
```

Des formes 3D possibles pour l'option `shape=` sont par exemple `'pillar', 'cube', 'flag', 'star', 'heart', 'prism', 'balloon', 'square', 'cylinder'`. Là encore, de nombreuses options sont à explorer, et on peut annoter un nuage 3D de manière similaire à un nuage plan (voir section VII-4).

b. Tracer des histogrammes

Tracer des histogrammes se fait à l'aide de l'instruction graphique `HISTOGRAM` de la procédure `UNIVARIATE`. Les classes sont choisies par l'utilisateur à l'aide de l'une des deux options `midpoints=` ou `endpoints=`. Par exemple si `num` est la variable numérique d'intérêt,

```
proc univariate data=don;
histogram num / midpoints=1 to 11 by 2  outhist=donhist  midpercents  cfill=blue ;  run;
```

va considérer les classes $[0,2[$, $[2,4[$, ..., $[10,12[$, et colorier l'histogramme en bleu. Il est aussi possible de spécifier les centres des classes les uns après les autres (par exemple `midpoints=3 5 7 9`) ou alors de spécifier les extrémités des classes (avec `endpoints=`) ; dans les deux cas malheureusement, ils doivent nécessairement être régulièrement espacés (ce qui est dommage !).

L'option `midpercents` va faire afficher les fréquences pour chaque classe, et `outhist=` va créer une table de sortie qui contiendra ces fréquences (et les bornes des classes).

On remarquera que l'instruction `histogram num` dispense d'avoir une instruction `var num`, qui est donc sous-entendue. Comme les autres graphes, un histogramme peut être sauvegardé (option `gout=` de la procédure, et options `name=` et `description=` des instructions `histogram`).

instruction `histogram` *couplée avec une instruction* `class`

Lorsque en plus une instruction `class` est utilisée, alors on peut faire afficher les histogrammes internes à chaque classe les uns à côté des autres, ce qui est idéal pour les comparer. Par exemple, si `num` est la variable numérique d'intérêt, et `cat1`, `cat2`, `cat3` sont des variables catégorielles à 4, 2, et 2 modalités respectivement, le code

```
proc univariate data=don;  class cat1;  histogram num / nrows=4 cframe=ligr cfill=blue; run;
```

fait réaliser les histogrammes pour chaque sous-groupe défini par les modalités de `cat1`, placés (un peu serrés !) les uns au dessus des autres (option `nrows=4`) dans une même fenêtre graphique, et

```
proc univariate data=don;  class cat2 cat3;  histogram num / nrows=2 ncols=2 ... ;  run;
```

fait réaliser 4 histogrammes placés en carré (2 en haut, 2 en bas, via les options `nrows=2 ncols=2`) pour chaque croisement de modalités des 2 variables binaires `cat2` et `cat3`. En l'absence d'options `nrows=` et `ncols=`, les histogrammes ne sont cependant pas forcément affichés un seul par fenêtre en cas de instruction `class`.

Ajustements paramétriques et non-paramétriques

En plus de tracer des histogrammes, l'instruction `histogram` permet de **realiser des ajustements a des lois parametriques** (gaussiennes, beta, lognormales, exponentielles, gamma, weibull) ou bien de réaliser des estimations non-paramétriques (à noyau) de la densité sous-jacente. Dans le cas paramétrique, il est alors possible de faire calculer la valeur et la p-valeur du **test de Kolmogorv-Smirnov d'adequation** correspondant (voir instruction `inset` plus bas), ce qui ne semble pas possible avec une autre procédure. Par exemple

```
histogram num / normal(mu=2 sig=1 color=red w=2 L=2 fill) cfill=blue ;
```

fait tracer la densité de la gaussienne réduite de moyenne 2 au dessus de l'histogramme, avec un trait pointillé jaune d'épaisseur 2 et la zone sous-courbe colorée en bleu (`cfill=blue` ne colore alors donc plus l'histogramme en bleu, mais la zone sous la courbe). Quand on ne spécifie pas les champs `mu=` et `sig=` (qui sont des options de l'option `normal`), alors il y a ajustement à $\mathcal{N}((\bar{X}_n)_{obs}, (S_n^2)_{obs})$. Les options diffèrent suivant la famille de lois qui est ajustée, voir l'aide pour les détails de la syntaxe.

Maintenant, pour des ajustements non-paramétriques, il est possible de faire afficher simultanément jusqu'à 5 courbes d'estimateurs à noyau de la densité sous-jacente de l'échantillon : par exemple

```
histogram num / kernel(c=0.2 0.5 l=1 2 k=triangular color=green) cfill=blue ;
```

va tracer l'histogramme en bleu, avec 2 courbes d'estimateur à noyau (noyau triangulaire), l'une en trait vert plein et une fenêtre $h = 0.2$, et l'autre en trait pointillé vert et fenêtre $h = 0.5$.

instruction `inset`

Elle permet de faire apparaître des petits encadrés dans les graphiques produits par `univariate` (histogrammes, QQ-plots,...), qui contiennent les valeurs de certaines statistiques que l'on indique soi-même (n'importe quelle statistique que `proc univariate` peut produire). Des options utiles sont : `position=` (ou `pos` pour faire plus court) pour indiquer où placer ces inserts dans la fenêtre graphique (`ne,nw,`... pour nord-est, nord-ouest,...) ; `header=` pour donner un titre à ces inserts ; `height=` pour indiquer la taille du texte dans ces inserts ; `cfill=` et `cfillh=` pour spécifier les couleurs de fond.

C'est dans ces petits encadrés qu'il est possible de faire afficher les valeurs et p-valeurs des statistiques de Kolmogorov-Smirnov (ou apparentées) d'adéquation : par exemple

```
proc univariate data=don;
 histogram num / normal(mu=2 sig=1 color=green w=2) cfill=blue ;
 inset mean='Moyenne' std='Ecart-type' median='Mediane' normal(ksd='KS' ksdpval='KSpV')
       / pos=ne cfill=pink height=2 header='Ajout d''un "Inset"' cfillh=orange;  run;
```

va tracer l'histogramme, avec la densité de $\mathcal{N}(2, 1)$ superposée, et dans le coin supérieur droit (option `pos=ne` de `inset`) un encadré intitulé "Ajout d'un Inset" (sur fond orange) indiquant (sur fond rose) les moyenne, médiane, écart-type, statistique et p-valeur du test d'adéquation de Kolmogorov-Smirnov H_0 : "$F = \mathcal{N}(2, 1)$". Il ne semble malheureusement pas possible de sauvegarder ces valeurs dans une table...

c. Tracer des boites de dispersion

On peut tracer des boites de dispersion de deux manières différentes. La première est d'utiliser la valeur `boxt` (ou ses variantes `boxf`, `box05`,... cf paragraphe VII-7-a) de l'option d'interpolation `I=` de la commande `SYMBOL`. Par exemple, si la table SAS `don` contient une variable `sexe` et une variable `age`, et que l'on veut côte-à-côte les boites de dispersion des âges selon le sexe, il faudra utiliser de base la syntaxe

```
symbol i=boxt;  proc gplot data=don;  plot age*sexe; run;
```

(avec certainement des adaptations à appliquer au cas par cas pour obtenir un résultat satisfaisant). On notera que l'ordre *var.numérique∗var.catégorielle* est important ici, car SAS ne peut tracer des boites de dispersion verticales avec `gplot`.

L'autre manière est d'utiliser la procédure `boxplot` du module SAS/STAT ; son manuel d'utilisation dans l'aide de SAS/STAT est très bien fait, et nous vous y envoyons pour les détails, capacités, et syntaxe.

d. Tracer des QQ-Plots

Réaliser des graphes quantiles-quantiles, ou bien des *probability-plots*, passe par l'utilisation de la procédure `univariate` : on ne s'étendra pas sur le sujet, la syntaxe des instructions `qqplot` et `probplot` étant assez intuitive. On notera qu'il est possible de faire tracer des lignes de référence, et d'estimer ou non les paramètres de la distribution à laquelle on confronte les données.

7. Syntaxe des principales options graphiques

Ecrire un chapitre sérieux sur les différentes manières d'utiliser les options graphiques de SAS est une gageure : en réalité, la meilleure façon de procéder est de se former soi-même en parcourant le manuel de SAS/GRAPH et les multiples exemples illustratifs qui la parsèment. Ce paragraphe n'a comme but que de décrire relativement sérieusement l'importante commande `symbol`, puis de manière beaucoup moins détaillée les principales autres commandes d'options graphiques (`legend` n'est pas évoquée par exemple).

Remarque importante : les options graphiques sont dites *additives*, autrement dit elles continuent de prendre effet jusqu'à ce qu'elles soient explicitement changées, remises à leur valeur par défaut (par une commande `reset=`) ou bien que la session SAS courante se termine.

a. Commandes SYMBOL

Un exemple d'utilisation de la commande `symbol` a été fourni au paragraphe 2. Celle-ci comporte un certain nombre d'options :

- `I= / INTERPOLATE=` (stipule le *type de tracé*) : `join` (affine par morceaux ; valeur par défaut), `none` (supprime l'interpolation), `needle` (diagramme en bâtons), `stepl` ou `stepr` (courbe en escalier continue à droite ou à gauche, càd les points gérés par le champs `v=` seront situés à gauche de chaque marche ou bien à droite de chaque marche ; voir aussi les modes `stepj` et `stepc`), `rl` (trace le nuage de points ainsi que la droite de régression classique par moindres carrés correspondante ; `rq` et `rc` concernent les régressions quadratique et cubique, et les options supplémentaires `CLI` et `CLM` font tracer les bornes de confiance ponctuelles pour les moyennes ou les prédictions, de niveau 95%), `spline` (interplation par courbe lisse / spline), `boxt` (boite de dispersion de la variable numérique stipulée en ordonnée, voir paragraphe 6-c situé plus haut ; `boxf` colore la boite avec la couleur du champ `cv=`, `boxf00` / `boxf05` / `boxf10` prennent comme "moustache" respectivement le minimum et le maximum, les quantiles d'ordre 5% et 95%, ceux d'ordre 10% et 90%, tandis que `boxf25` supprime les "moustaches". Le champ `bwidth=` permet d'indiquer la largeur des boites, par exemple `bwidth=10pct` pour 10% de la largeur de la fenêtre), `hilo` et `std1/2/3` (graphes "high-low" ou "moyenne$\pm$1/2/3 écarts-types, sortes de boites de dispersion... sans boites)
- `L= / LINE=` (indique le *type de ligne*, `L=1` étant solide et la valeur par défaut, et toutes les valeurs entières ≥ 2 correspondant aux lignes en différents types de pointillés)
- `W= / WIDTH=` (*épaisseur du tracé*, la valeur par défaut étant 1)
- `V= / VALUE=` (indique le *type de symbole* utilisé pour les points, la valeur par défaut étant +, et `v=none` supprimant le tracé de points ; voici des exemples de valeurs avec les codes correspondants : `x` ($\times$), `dot` ($\bullet$), `circle` ($\circ$), `triangle` ($\triangle$), `diamond` ($\diamond$), `-` ($\odot$), `+` ($\oplus$), `=` ($\star$), `:` , `>` , `*` , `_` , `"` , `#` , `$` , `%` , `&`)
- `H= / HEIGHT=` (indique la *taille du symbole*, la valeur par défaut étant 1)
- Couleurs : `CI=` (*couleur du tracé* : on stipule une couleur à la suite du signe `=`, sans guillemets, par exemple `c=red` ; la page suivante de la toile liste les couleurs utilisables sous SAS

  http ://www.devenezia.com/docs/SAS/sas-colors.html

  `CV=` (*couleur des symboles*), `CO=` (*couleur des boites ou lignes des intervalles de confiance*)

L'utilisation des instructions `symbol` nécessite un peu d'habitude, en particulier quand plusieurs tracés avec des paramètres différents doivent être réalisés : cela nécessite d'utiliser des instructions `symbol1` puis `symbol2`, etc... qui vont s'appliquer de manière cyclique [45]. En tout état de cause, il est utile de savoir que soumettre l'instruction `symbol;` va faire tous les paramêtres graphiques gérés par toutes les commandes `symbol` ou `symbol`i. Voir l'aide de SAS à ce sujet (rubrique GPLOT, instruction PLOT, puis cliquer sur SYMBOL dans le cadre de départ).

b. Commandes AXIS

Une commande `AXIS`i va définir un axe en dehors de toute procédure, qui pourra alors être attribué à un ou plusieurs axes de instructions graphiques (comme `plot` dans `gplot`) lorsque l'on veut configurer les choses soi-même. L'attribution de la définition d'axe `AXIS1` à l'axe horizontal (resp. vertical) d'un graphe se fera alors par le biais de l'option `haxis=axis1` (resp. `vaxis=axis1`) de l'instruction concernée.

Les champs les plus utiles de la commande `axis` sont `label=` (gère le libélé de l'axe, par exemple `label=none` ou `label=(H=2 F=GREEK 'f' H=1 'n' H=2 '(x)')` qui affichera $\phi_n(x)$ comme libélé de l'axe), `length=`n (taille de l'axe), `value` et `order` (qui gèrent les graduations et leurs intitulés).

Toutefois la définition des axes est un sujet trop touffu et pas forcément passionant (nombre et aspect des petites et grandes graduations via les champs `order`, `major`, `minor` et `offset`, définition de l'origine avec

45. grosso modo, si par exemple 4 graphes doivent être tracés et que les commandes `symbol1` et `symbol2` ont été soumises, sans commande `symbol3`, alors les graphes 1 et 3 seront tracés suivant les paramètres donnés par `symbol1`, et les graphes 2 et 4 le seront suivant les paramètres donnés par `symbol2`. Cependant les choses peuvent se compliquer, particulièrement quand ce ne sont pas les mêmes champs qui sont renseignés dans les différentes commandes `symbol`i...

`origin`, etc...), on ne s'étendra donc pas plus dessus.

Par contre, pour finir, on présentera quand même un exemple, qu'il est difficile d'intuiter dans son coin : supposons que l'on veuille tracer les boites de dispersion relative à une variable numérique `num`, une par modalité d'une variable binaire `cat` valant soit `A` soit `B`. Le code

```
symbol i=boxt;   proc gplot;  var num*cat;  run;
```

va effectivement tracer les boites de dispersion, mais la première complètement à gauche, sur l'axe des ordonnées, et la seconde complètement à droite, ce qui est pour le moins inesthétique. Le code suivant corrige ce problème, tout en égayant le tout avec quelques couleurs

```
symbol i=boxf co=red cv=blue bwidth=25pct;   axis order= ' ' 'A' 'B' ' ';
proc gplot;  var num*cat;  run;
```

c. Commandes `TITLE`

L'utilisation de titres est fortement recommandée : pour les graphiques cela va de soi, mais aussi pour toutes les sorties de procédures. En effet, les titres qui seront choisis seront reportés dans la fenêtre Results de SAS, et ce simple fait simplifie considérablement la vie de l'utilisateur ; on le constate en TP, donc en milieu professionnel et en utilisation intensive encore plus.

Le principe est que la commande `title "Resultats du TP";` va faire afficher ce titre à toutes les fenêtres (même en sorties ODS) et ce, jusqu'à ce qu'une nouvelle instruction `title` soit soumise. l'instruction `title;` tout court, supprime l'affichage d'un titre. *Ce principe s'applique d'ailleurs à toutes les options mises en place par des commandes globales...*

Il est possible de contrôler la fonte et la taille des caractères utilisées dans les titres (de graphiques). Par exemple le code particulièrement illisible suivant

```
title 'Courbe de ' f=swissi 'f' h=1 'n' h=2 '(' f=greek 'a' f=swissi ')';
```

va donner le titre "Courbe de $f_n(\alpha)$" (remarque : la taille par défaut est `h=1`). On voit donc qu'une succession d'options `h=` et `f=` (respectivement `height=` et `font=`) permet d'obtenir des titres satisfaisants. Des fontes utiles sont `swiss/swissb/swissi` (défaut), `centb/centbi` (century), `zapf/zapfi/zapfb` (zapf), `german`, `script`, `complex`, `greek` (alphabet grec), `math` (symboles mathématiques), `special` et `marker` (sigles divers). L'onglet de l'aide de SAS détaillant toutes ces fontes est intitulé *Using SAS/GRAPH Software fonts.*

Si un titre est trop long, il faudra utiliser les commandes `title`n, qui constitueront les $n$ème lignes du titre. Toute définition de `title`k annule tous les autres titres $> k$.

Il est possible de colorer, souligner, et agrémenter de diverses façons les titres et autres notes, voir pour cela l'aide de SAS directement.

d. Commandes `NOTE` et `FOOTNOTE`

Ces instructions permettent de faire inscrire des notes au sein des graphiques, ou bien des notes de bas de page, le champ `J=` (`justify=`) indiquant si la note doit être placée à droite justifiée à droite (`R`), à gauche justifiée à gauche (`L`), ou au centre, centrée (`C`). Les notes de bas de page sont placées par défaut au centre, et les notes en haut à gauche dans la fenêtre graphique. Il est possible de créer plusieurs notes, par exemple `footnote1 J=R ...;` , `footnote2 J=L ...;`. Tout comme pour les titres, la fonte à utiliser est indiquée dans le champ `font=`, et la hauteur de texte dans le champ `h=`. Les notes peuvent subir des rotations, pouvant même devenir verticales (options `angle=` ou `rotate=`).

Il est aussi possible de customiser la manière dont les légendes de graphes sont affichées, avec la commande globale `legend` (quelques options sont `label, position, frame, down,`...).

Enfin, la commande globale `GOPTIONS` permet de mettr-276(de)dr(ettr 343 0 i c4 0 udes)-282(-376(de)s-376(dptions1(s)-3786

VIII. Le b-a-ba de l'écriture de macros SAS

Comme de nombreux logiciels, SAS a un langage permettant de créer des macro-commandes, ou plus communément *macros*. Une macro est en fait un morceau de code qui a besoin d'être réutilisé de manière intensive : elle est affublée d'un nom, et peut-être paramétrable (en fait, elle l'est le plus souvent). Il existe aussi des *macro-variables*, qui jouent plus ou moins le rôle de *variables globales* (par opposition aux variables d'une table SAS), et leur usage facilite énormément la vie de l'utilisateur de SAS (macro-commandes et macro-variables sont communément appelées des *alias* dans d'autres contextes).

Tous les chapitres de l'aide de SAS dont il est fait allusion ci-dessous sont compris dans l'onglet de l'aide

SAS Products → *Base SAS* → *SAS Macro Reference* → *Understanding and Using the Macro Facility.*

La liste des macro-commandes de base (et des exemples d'application) est située dans l'onglet

SAS Products → *Base SAS* → *SAS Macro Reference* → *Macro Language Dictionary* → *Macro Language Dictionary.*

1. Macro-variables et macro-commandes de base

Jusqu'à maintenant, l'utilisation de SAS s'est bornée à faire se succéder des étapes Data et Proc, toute valeur numérique d'intérêt se trouvant soit dans une table SAS, soit affichée à l'écran, et le Journal (Log) ne servant que d'interface permettant au logiciel de vous gronder à la moindre erreur de syntaxe. Cependant, en utilisant le langage macro de SAS, il va être désormais possible :

a) de définir des *macro-variables*, dans lesquelles pourront être stockées des valeurs (numériques ou non) *en dehors de tout attachement à une quelconque table SAS* ;

b) de se servir de la fenêtre Journal (Log) comme d'une fenêtre accueillant des résultats de calculs et de instructions Macro, et ainsi réaliser des calculs *sans faire appel à une quelconque procédure SAS* ;

c) de faire réaliser plus facilement des tâches répétitives, sans avoir besoin de taper ou soumettre un code long à chaque fois [46].

On voit donc poindre le fait que la seule utilisation des macro-variables ouvre tout de suite de nouveaux "horizons" dans la pratique de SAS...

Les macro-variables peuvent être définies de deux façons :

— soit explicitement avec la macro-commande `%LET` (auquel cas il s'agira de macro-variables **globales**) ;

— soit implicitement au sein de la définition d'une macro-commande de son cru (auquel cas il s'agira de macro-variables **locales**).

Dans ce paragraphe, comme nous n'avons pas encore vu ce qu'étaient des macro-commandes et comment les définir (ce qui est le sujet du paragraphe suivant), on se bornera donc à considérer les macro-variables créées à l'aide de la commande `%LET`. L'importante différence entre macro-variables locales et globales sera toutefois évoquée dans la paragraphe sur les macro-commandes.

Voyons maintenant des exemples d'utilisation de la commande `%LET` :

Exemple 1 : `%let x=3; %put &x;`

Cette instruction va créer une macro-variable globale `x`, "numérique" (voir remarque plus bas), de valeur actuelle 3, et à laquelle on se référera en tapant `&x`. Par exemple ici, `%put &x;` fait afficher dans la fenêtre Journal la valeur de `&x`, c'est-à-dire 3 (on remarque qu'il n'y a pas à mettre entre guillemets ce qui suit le mot-clef `%put`).

Si l'on avait omis le sigle `&`, alors la commande `%put x;` ferait simplement afficher dans le Journal la lettre `x`. En fait, la présence ou l'absence du sigle `&` est le seul moyen de discerner les macro-variables des variables standard d'une table SAS !

Dans la même veine, l'instruction `%put; %put La valeur de x est &x. ; %put;` fait afficher dans le Log la phrase *"La valeur de x est 3."* , suivie et précédée d'un saut de ligne (résultat des instructions `%put;`).

Exemple 2 : `%let x=3; %put %eval(&x*4+1);`

Cette instruction fait cette fois *afficher le résultat d'une opération* (4 fois `&x plus 1`), donc ici affiche 13 dans la fenêtre Journal. Cependant, c'est la présence de l'évaluateur `%eval` qui fait que le calcul est effectivement réalisé : si on l'omet, alors dans le Journal c'est plutôt l'énoncé de l'opération elle-même (`&x*4+1`), que son résultat, qui va s'afficher.

En cas de calculs sur des nombres non-entiers, c'est par contre l'évaluateur `%sysevalf` qu'il faudra utiliser : voir la documentation à ce sujet...

46. cela facilite également les modifications du code : on ne les modifie qu'à un seul endroit et cela se répercutera sur toutes les exécutions du code.

Exemple 3 : `%let x=3; %let y=bip;`

```
data don; input var1 @; var2=&x+1 ; input var&y; var&x="bip&y";
cards; (↩) 0 1 (↩) 1 4 ...
```

Cet exemple idiot est plus complexe que les précédents, mais a le mérite de bien présenter ce que les macro-variables permettent de faire. Ici on crée une table `don` qui aura 4 variables : la première, `var1`, dont les valeurs seront lues dans le champ `cards` ; la seconde, `var2`, qui vaudra constamment la valeur de la macro-variable `&x`, plus 1, autrement dit 4 ; la troisième, dont le nom sera `varbip`, comme résultat de la juxtaposition `var&y`, la variable `&y` valant `bip` ; la quatrième, `var3` (comme résultat de la juxtaposition `var&x`), vaudra la chaîne de caractères constante *bipbip* (comme résultat de la juxtaposition `bip&y`).

Plusieurs remarques s'imposent toutefois, ayant trait à *la nature même des macro-variables*, qui n'est pas évidente à cerner :

- la déclaration `%let y=bip` n'a pas vraiment comme résultat que la variable `&y` vaut la valeur alphanumérique constante `"bip"`, mais plutôt que `&y` vaut la succession de lettres `bip`, que l'utilisateur pourra utiliser de diverses manières dans un code à venir, en tant que "morceau de mot SAS" (si l'on peut s'exprimer ainsi) : par exemple dans `input var&y`, cela crée une variable de nom `varbip`, tandis que dans `var&x="bip&y"`, cela crée une constante alphanumérique "bipbip".
- dans la continuation de ce qui vient d'être dit, on comprend que si l'on avait écrit `%let y="bip"` au lieu de `%let y=bip`, le code `input var&y;` aurait occasionné une erreur, puisqu'une variable SAS ne peut pas contenir de guillemets dans son nom.
- dans la même veine, quand on écrit `%let x=3`, on pense définir une "macro-variable numérique" `&x` valant 3 puisque l'instruction `var2=&x+1` donne bien comme résultat la création d'une variable `var2` valant constamment 4 (sans pourtant utiliser l'évaluateur `%eval`) ; pourtant, le code `var&x="bip&y"` crée bien une variable `var3`, ce qui signifie que `&x` semble cette fois avoir été traîtée comme variable littérale ! La nature des macro-variables "numériques" n'est donc pas si claire, et il sera quelquefois nécessaire de rajouter $+0$ à une macro-variable que l'on pense numérique (voire d'utiliser `%eval`) pour la rendre vraiment utilisable comme nombre dans une commande...
- attention, dans la commande `var&x="bip&y"`, la macro-variable `&y` n'a été remplacée par sa valeur (littérale) que parce que la chaîne de caractères `"bip&y"` était entourée de guillemets doubles ; si la commande avait été `var&x='bip&y'`, alors la variable `var3` aurait valu la chaîne "bip&y", c'est-à-dire le sigle `&` aurait été traîté comme un caractère comme un autre et non comme le caractère spécial de SAS annonçant le nom d'une macro-variable...
- puisqu'il y a deux instructions `input`, le signe `@` est nécessaire à la fin de la première (voir page 7, mais cela n'a rien à voir avec les macros en particulier).

Exemple 4 : `%let ppr=%str(proc print; run;);  &ppr;`

crée une macro-variable `&ppr` contenant le code SAS `proc print; run;`, qu'on encadre grâce à la macro-commande `%str(...)` : il suffit alors de soumettre simplement le nom de la macro-variable pour que le code qu'elle contient soit exécuté. Alors certes, ici, le code n'est ni long ni impressionant, mais en tout cas il contient plusieurs points-virgules ; il pourrait être beaucoup plus long et même contenir des retours-chariot et donc s'étaler sur plusieurs lignes (toutefois, généralement on définit pour cela des macro-commandes, voir le paragraphe suivant).

Exemple 5 :

```
%let vars=var1 var2 var7 x z;
%put On realise ci-dessous l'ACP des variables &vars;
proc princomp data=don; var &vars; run;
```

la macro-variable `&vars` contient une liste de noms de variables, qui est ici utilisée au sein d'une procédure et en insertion dans un commentaire qui apparaitra dans la fenêtre Journal.

Remarque qui peut avoir son importance : il est parfois nécessaire d'indiquer au compilateur macro / à SAS, que l'on a fini de nommer une macro-variable. Ceci se fait par l'adjonction d'un *point* à la fin de l'appel à la macro variable. Par exemple,

```
%let x=Var;
data point;
input &x.2 @@ ;
cards;
3 4 1 2
;
```

crée une table sas contenant une seule variable, *Var2*, et 4 individus. Si on avait omis le point après `&x` , alors il y aurait eu une erreur puisque on aurait cherché la valeur de la macro variable `&x2`... L'exemple est idiot [47] mais montre bien que parfois, un simple point change la vie et peut résoudre bien des problèmes !

47. comme la plupart de ceux de ce chapitre, mais ce n'est pas grave...

Avant de parler des macro-commandes "faites maison", parlons encore de quelques sujets relatifs aux macro-variables :

– *Macro-variables Automatiques*

Le système SAS définit tout seul un certain nombre de macro-variables, qui s'avèrent bien pratiques voire indispensables : par exemple `&sysdate` contient la date du jour courant (en format `date7.`, *e.g.* 23JAN07), `&sysday` le nom du jour courant (lundi, mardi,...), `&systime` l'heure courante (en format `time5.`, *e.g.* 09h27),...

– *Fonction* `%scan`

Elle permet d'extraire des mots de l'expression d'une macro variable : par exemple, si `&vars` désigne la macro-variable de l'exemple 5 ci-dessus, alors `%scan(&vars,3)` désigne le troisième mot (`var7`), et `%scan(&vars,4)` désigne le quatrième mot (`x`). En l'occurence, ici c'est l'espace qui tient le rôle de délimiteur de mots, mais d'autres caractères peuvent également le faire (le f,...). Attention à bien écrire `%scan(&vars,3)` et non pas `%scan(vars,3)`.

– *Étapes DATA et utilisation des fonctions* `symput` *et* `symget`

Les fonctions `symput` et `symget` permettent, au sein d'étapes DATA particulières, de respectivement stocker dans une macro-variable une information contenue dans une table SAS, ou au contraire d'extraire le contenu d'une macro-variable. Les possibilités que ces fonctions SAS offrent sont particulièrement intéressantes, ce qui explique que nous prendrons un peu de temps pour les exposer un tant soit peu.

Par exemple, si `results` est une table contenant des résultats d'une régression linéaire simple, y compris la valeur du R^2 de Pearson (variable `_RSQ_`[48]), et que l'on souhaite stocker cette valeur dans une macro-variable `&R2` pour un usage ultérieur, on procède comme suit :

```
data _null_;  set results;  call symput('R2',_RSQ_); run;
%put La valeur du R2 de Pearson pour ces donnees est de &R2;
```

La première ligne est une étape `data _null_` qui ne crée aucune table SAS, mais permet de faire certaines opérations, comme ici l'utilisation de la fonction `symput` : elle nécessite d'être "appelée", par la fonction `call`, que nous ne décrirons pas plus ici... On remarque deux choses : d'abord, à la création de `&R2`, il n'y a pas de sigle `&` mais le nom de la macro-variable est entouré de guillemets simples dans `symput('R2',_RSQ_)`, ensuite un `run;` est requis après l'étape DATA car les macro-commandes (ici, `%put`) ne sont pas autorisées au sein d'une étape DATA.

Si maintenant la table `results` contient 6 lignes (observations) correspondant à l'analyse de régression réalisée pour chaque modalité d'une variable catégorielle `categ` prenant 6 valeurs (disons, a,b,c,d,e,f), on va créer 6 macro-variables contenant la valeur du R^2 pour chacune des 6 analyses, que nous appelerons `&R2a`, `&R2b`, ..., `&R2f`, à l'aide du code suivant :

```
data _null_;  set results;  call symput('R2'||categ,_RSQ_); run;
```

En effet, `results` contient 6 observations, donc pendant le premier passage de boucle DATA, on est sur la première observation et la variable `categ` vaut `a`, donc `'R2'||categ` vaut `'R2a'` (rappel : l'opérateur `||` est l'opérateur de concaténation des chaînes de caractères) ; au second passage de boucle, `categ` vaut `b`, donc `'R2'||categ` vaut `'R2b'`, etc...

Parlons maintenant rapidement de la fonction "inverse" `symget` : dans une étape DATA, `call symget('R2a')` équivaut à `&R2a`, donc à première vue cela ne sembla pas passionant. L'avantage est que le fait de travailler avec des chaînes de caractères permet plus de souplesse : par exemple si `don` désigne la table des données d'origine (avec variable numérique `x` et catégorielle `categ`), et que pour une raison saugrenue on voulait créer une variable `xdivR2` égale au quotient de `x` par la valeur du coefficient R^2 correspondant à sa catégorie (ce qui n'a aucun sens statistique), on écrirait

```
data don;  set don;  xdivR2=x/symput('R2'||categ);   proc print; run;
```

Sur cet exemple absurde, on est obligé de constater que l'utilisation de `symget` a permis de faire quelquechose qu'on ne savait pas faire autrement...

48. la présence de la variable `_RSQ_` dans la table issue de l'option `outest=` de `proc reg`, est obtenue en utilisant l'option `adjrsq` de la instruction `model`.

2. Macro-commandes de nies par l'utilisateur

Si les macro-variables jouent le rôle de variables globales lors d'une session SAS, alors les macro-commandes que l'utilisateur définit (ou a dénichées ailleurs) jouent le rôle des fonctions que l'on définit dans un langage fonctionnel comme Matlab ou Scilab par exemple. Elles ont elles aussi de multiples usages, et savoir les manipuler et en créer est un gage de bonne compréhension et maîtrise du logiciel SAS.

Définir une macro consiste à écrire le code entre les instructions `%MACRO` et `%MEND` (*macro end*), puis soumettre le tout. Dans ce cas, la **compilation de la macro** a lieu, sans qu'il y ait nécessairement de sortie sur la fenêtre Journal.

Par exemple, si l'on soumet la définition de macro simple suivante

```
%macro exemple(table,opt,alpha);
 proc univariate data=&table cibasic alpha=&alpha robustscale &opt; run;
%mend;
```

alors le code

```
%exemple(donnees,normal,0.01);
```

équivaut au code ci-dessous

```
proc univariate data=donnees cibasic alpha=0.01 robustscale normal; run;
```

Cet exemple simple est largement suffisant pour expliquer plusieurs aspects des macro-commandes et de la syntaxe associée :

- aucun `run;` n'est nécessaire après le lancement d'une macro.
- dans la définition de la macro, un certain nombre de macro-variables sont implicitement définies, comme arguments de la macro. Tout comme pour les macro-variables globales définies par `%let`, dans cette déclaration les macro-variables ne sont pas munies du sigle `&`, ce préfixe n'étant utilisé que lors des appels ultérieurs des macro-variables. Ici par exemple, le premier argument est `table` dans la déclaration de la macro, et on l'appelle `&table` dans le corps de la macro : en lançant `%exemple(donnees,normal,0.01)`, toutes les occurences de `&table` dans le corps de la macro seront remplacées par la chaîne de caractères `donnees`.
- on aurait très bien pu lancer `%exemple(donnees,normal round=2 mode,0.01)`, et la macro-variable `&opt` aurait pris la valeur `normal round=2 mode`.
- d'un autre côté, on peut aussi rendre vide un argument d'une macro : du moment que cela n'engendre pas d'erreur de syntaxe, c'est tout à fait autorisé. Par exemple, `%exemple(donnees,,0.01)` lance la macro avec la macro-variable `&opt` vide, mais `%exemple(,,0.01)` occasionnera une erreur car SAS lira le code `proc univariate data= cibasic alpha=....`
- il existe une autre syntaxe pour spécifier les arguments d'une macro : par exemple, `%exemple(donnees,alpha=0.01)` équivaut à `%exemple(donnees,,0.01)`. C'est pratique pour deux raisons : la première est qu'ainsi, on n'a pas à se souvenir de l'ordre des arguments (mais des noms, si), la seconde est que cela devient plus lisible et explicite au lancement de la macro.
- une macro n'a pas forcément besoin d'être affublée d'arguments comme ci-dessus (on peut très bien définir `%macro ppr; proc print; run; %mend;` et lancer `%ppr`).
- une macro peut même avoir des valeurs par défaut pour ses arguments : si la macro ci-dessus s'applique la plupart du temps avec `alpha=0.05` et `opt=mode`, on peut déclarer la macro comme suit

  ```
  %macro exemple(table,opt=mode,alpha=0.05);
  ```

  auquel cas il suffit d'appeler `%exemple(donnees)`, et si l'on veut changer la valeur de `alpha` seulement, on peut appeler `%exemple(donnees,alpha=0.1)`, et l'argument `opt` gardera sa valeur par défaut.
- entre les mots-clefs `%macro` et `%mend` se situe la définition de la macro : celle-ci peut très bien avoir des dizaines de lignes de longueur, et c'est d'ailleurs souvent le cas puisque l'intérêt même des macro-commandes "customisées" est justement de synthétiser un grand nombre de commandes...

Di erence entre macro-variables globales et macro-variables locales

On appelle macro-variable locale une macro-variable définie avec `%LET` *hors* d'une définition de macro. On appelle macro-variable locale une macro-variable définie au sein d'une macro, soit implicitement comme argument de la macro (dans la ligne de déclaration commençant par `%macro`) soit définie par une commande `%LET` mais *dans le corps* de la macro. Dans la macro ci-dessous, `&t`, `&a` et `&x` sont locales, `&y` est globale.

L'importante différence est qu'une macro-variable globale pourra être invoquée *n'importe où dans un code SAS*, alors qu'une macro-variable locale ne pourra PAS être invoquée en dehors du corps de la définition de cette macro : c'est ce que l'on appellerait en maths une variable *muette*[49].

49. Pour ceux qui connaissent Scilab ou Matlab, une variable locale est une variable définie comme paramêtre d'entrée d'une fonction ; l'évoquer en dehors du code définissant cette fonction va forcément occasionner une erreur ; une variable globale, elle, est une variable qui est définie au sein de l'interface de Scilab/Matlab, ou dans un fichier de commandes

La distinction dont il est question ici s'appelle *scope* en anglais ("portée"). Par exemple, si l'on soumet

```
%let y=passionant;
%macro essai(t,a);
 %let x=%evalf((&a+3)*3.5)
 data t2; set &t; a=&a; run;
%mend;
%essai(don,2);
%put x vaut &x; %put a vaut &a; %put Trop &y les TPs de SAS!; proc print data=t2; run;
```

alors SAS dira qu'il ne connait pas de macro-variable `&x` ou `&a`. Par contre il reconnait `&y`. Si l'on veut absolument que la macro-variable locale `&x` devienne globale, pour cela il faut rajouter au début de la macro (pour plus de clarté) l'instruction `%GLOBAL &x;` . Ceci est très important car on a souvent envie de "sortir" des informations d'une macro autrement que sous la forme d'une table SAS (voir un exemple plus bas avec la macro `%nbmots`).

Macro-commandes `%IF`, `%THEN`, `%ELSE`, `%DO`, `%END`, `%WHILE`, `%UNTIL`

L'avantage des macro-commandes définies par l'utilisateur, est que celui-ci peut décider d'enchaîner de nombreuses étapes DATA et PROC au sein d'une seule macro-commande (par exemple une procédure statistique comme `princomp` ou `reg` avec sauvegarde des résultats, suivi d'un traitement de la table en sortie au sein d'une étape DATA puis d'une présentation avec `proc tabulate` et sortie ODS, etc...). Cependant le programmeur peut vouloir que certains morceaux de ce gros code n'apparaissent que si on l'indique en argument de la macro. Pour ce faire il faut disposer d'une syntaxe permettant de lancer des morceaux de code *conditionnellement* à la valeur d'une macro-variable locale argument de la macro. C'est là que les macro-commandes du style `%if`, `%then`, et `%do ... %end` entrent en jeu.

Donnons un premier exemple de macro exploitant ces mots-clef :

```
%macro essai(table,nombre,var,renomme,mot1,mot2);
  %IF &renomme=oui %THEN %DO;
     data &table; set &table; if &var=0 then temp=&mot1; else temp=&mot2; drop &var; run;
     data &table; set &table; rename temp=&var; run;
  %END;   %* c'est la fin du %DO;
  %DO i=1 %to &nombre;
     data fichier&i; set &table; type=&i; run;
  %END;   %* c'est la fin du %DO i=...
%mend;
```

Dans cette macro, on prend une table `&table` en entrée, et dans un premier temps, si on a entré `oui` pour la variable `&renomme` (macro-instruction `%IF &renomme=oui %THEN`), on va renommer les modalités de la variable `&var` (qui est censée être une variable de `&table`, et deviendra alphanumérique si `&mot1` et `&mot2` le sont). Puisque ce qui suit le `%THEN` est plus long qu'une seule instruction, il est indispensable de l'encadrer avec un `%DO .. %END`, qui est la version macro de l'encadreur `do ... end` habituel. Dans un second temps, on utilise une macro instruction `%DO` itérative, qui crée une nouvelle macro-variable `&i` (locale), et la macro crée autant de fichiers `file1, file2,...` que la valeur de `&nombre` en entrée ; chacune de ces tables est la table `&table` affublée d'une nouvelle variable, `type`, valant la valeur constante `&i` correspondant au passage de la boucle.

Voyons maintenant un exemple plus complexe mais autrement plus utile, qui compte ET enregistre le nombre de mots dans une macro-variable littérale, et que je vous laisse le loisir d'étudier [50] et tester (elle utilise notamment l'instruction `%DO %WHILE ... %END` et les commandes `%global` et `%scan`) :

```
%macro nbmots(macvar);
  %global nbmots;
  %let j=1; %let mot=%scan(&macvar,1);
  %do %while("&mot" NE "");
     %let j=%eval(&j+1); %let mot=%scan(&macvar,&j);
  %end;
  %let nbmots=%eval(&j-1);
  %put Il y a &nbmots mots dans la macro-variable en entree;
%mend;
%let vars=je n aime pas les fichiers du type fic.doc;
%nbmots(&vars);
%put &nbmots;
```

Voici enfin deux autres exemples de macros sans plus de commentaires, à méditer et tester :

```
%macro combine;
  data grande; set %do i=1 %to 2; don&i %end; run;
%mend;
%combine;
```

50. Questions sur cette macro : *a*) lister les macro-variables en présence, et indiquer leur portée (locale ou globale) ; *b*) expliquer ce que réalise la boucle `%while` ; *c*) dire pourquoi cela occasionerait une erreur de remplacer la condition `"&mot" NE ""` par la condition `&mot NE ""` ; *d*) dire pourquoi `%put &nbmots` n'occasionne pas d'erreur, et pourquoi sa valeur est de 10 dans l'exemple.

```
%macro regress(table,vardep,varindeps,graphe,alpha=0.05);
  %nbmots(&varindeps);
  %do i=1 %to &nbmots;
   proc reg data=&table alpha=&alpha;
    model &vardep=&varindep / clm cli;
    %if &graphe=oui %then
     plot (&vardep P.)*&varindep / overlay ;
   run;
  %end;
%mend;
```

Pour terminer ce paragraphe, voici quelques remarques mineures relatives au macro-langage SAS.

- %put _USER_; va faire a cher dans le log toutes les macro-variables de nies par l'utilisateur, ce qui est extrêmement pratique ; il existe aussi les alias _GLOBAL_ et _LOCAL_ (qui referent respectivement aux macro-variables globales et locales de nies par l'utilisateur).
- Il est tres utile de pouvoir adjoindre des commentaires au code d'une macro : pour ce faire, il faut entourer son commentaire par %* et un point virgule (au lieu de * et un point-virgule, pour les commentaires SAS usuels).
- Il est possible de stocker des de nitions de macros dans un repertoire particulier, de sorte a ce que ces macros soient disponibles automatiquement a chaque lancement de session SAS (voir l'onglet *Storing and Reusing Macros*, qui est suivi d'un chapitre dedie a des conseils pour ecrire des macros e cacement).
- Il existe des bibliotheques de macros, certaines deja contenues dans SAS/Base ou ses modules, d'autres a telecharger sur certains sites de la toile dont celui de SAS : il n'est pas idiot d'aller y jeter un oeil car certaines d'entre elles peuvent être fort utiles...
- Une macro-fonction interessante est %SYSFUNC : ce ne serait pas une perte de temps d'aller voir dans l'aide ce qu'elle permet de realiser et comment elle fonctionne.
- En cas de problemes avec le macro-langage, l'onglet *Macro Quoting* de l'aide SAS peut s'averer fort utile (en particulier en ce qui concerne la gestion des caracteres speciaux et dedies et con its potentiels).

3. Mot de la n sur le macro-langage

Il est certain que ce chapitre n'a fait qu'effleurer les possibilités qu'offre le macro-langage SAS : les exemples présentés ici sont pour la plupart des exemples jouets qui n'ont presque rien à voir avec les macros qui sont manipulées sur le terrain (mais il en est presque toujours de même dès que l'on doit enseigner de nouveaux concepts et de nouvelles commandes !). Il parait difficile en une poignée de pages de montrer ce que le macro-langage peut apporter chaque jour à l'utilisateur SAS, et ce sera à ce dernier de se poser les bonnes questions, et de trouver tout seul le moyen de les résoudre ; dans ce processus d'auto-formation, il est certain que la lecture de macros toutes prêtes (sur la toile ou dans l'aide) est d'une aide précieuse (et, faut-il encore le rappeler, la lecture de l'aide [51] de SAS !). Et d'ailleurs, connaître le langage macro sert également à ça : savoir lire les macros écrites par les autres, afin de savoir les utiliser correctement ! Et la confrontation à de telles macros a souvent lieu plus tôt qu'on ne le pense...

51. le manuel du macro-langage dans l'aide de SAS est très bien conçu et rédigé, sa lecture active et structurée est fortement conseillée, et si besoin est, la bibliothèque de l'UVSQ contient un livre sur les macros, *SAS Macro Language*.

IX. Compléments : des commandes globales SAS, le système ODS, fusion de tables, et quelques liens

1. Des commandes globales utiles

La commande globale `OPTIONS` sert à configurer les sessions SAS, et plutôt que de le faire à la souris (via l'onglet *Tools* → *Options*), autant savoir l'écrire en code, qui pourra être relancé en début de session sans effort et directement intégré à chacun de vos programmes.

Il n'est pas question de faire le tour des aspects d'une session SAS que la commande `options` permet de gérer, mais seulement d'en indiquer quelques uns, dans le désordre, qui peuvent souvent être utiles (et plus généralement d'insister sur le fait que SAS est très configurable, et d'une complexité insoupçonnée pour l'utilisateur moyen) :

- `NOREPLACE` : interdit l'écrasement de tables SAS permanentes existantes
- `OBS=`n : définit globalement le numéro maximum des observations à traîter (très utile en phase de débuggage, pour tester sereinement des programmes sans passer toutes les données à la moulinette)
- `PAGESIZE=`n : nombre de lignes par page de sortie, très utile (par exemple) pour que les tables en sortie de `proc freq` ne soient pas découpées inutilement en morceaux.
- `NONUMBER` ou `NODATE` : supprime l'affichage en haut de l'écran des numéros de pages ou de la date
- `LINESIZE=`n : spécifie la longueur, en caractères, des sorties de procédures et du journal (Log)
- `MACRO` : autorise l'utilisation de macros SAS (valide par défaut)
- `USER=` : permet de spécifier l'adresse physique d'un répertoire, qui sera alors celui dans lequel toutes les tables non affublées d'un *libref* seront crées
- `WORK=` : permet d'indiquer l'adresse physique du répertoire qui sera le répertoire SAS de *libref* Work ; la différence avec l'option précédente est le répertoire Work est "nettoyé" à la fin et au début de toute session SAS...
- `_LAST_=` : permet de spécifier le nom SAS d'une table qui sera toujours utilisée pour les procédures qui seront dépourvues d'option `data=` ; par défaut, la table en question est la dernière table à avoir été créée.
- `YEARCUTOFF=` : utile pour que SAS comprenne des dates "années" avec seulement 2 chiffres : par exemple si `yearcutoff=1920`, alors la data 15 sera comprise comme 2015 et non comme 1915

Pour connaître la valeur des options en cours de validité, utiliser `proc options`. On peut aussi en spécifier certaines au sein d'un *fichier de configuration*. On notera qu'il y a un grand nombre d'options qui sont spécifiques à l'environnement (système d'exploitation), il est donc important de se référer au manuel adéquat pour des sujets ayant attrait à l'impression par exemple, ou à toute interactivité possible avec le système d'exploitation ; par exemple `DEVICE` (driver à utiliser pour imprimer ; ex. `psepsf` sous Unix), `DOCLOC` et `HELPLOC` (emplacement de la doc ou de l'aide SAS), `ENGINE` (moteur SAS à utiliser par défaut pour lire des fichiers SAS ; e.g. `V8` pour la version 8),...

D'autres commandes globales autres que `options` existent, on en a déjà vu plusieurs : `filename, libname, symbol, title, footnote`... Il est utile d'insister sur l'avantage qu'il y a à fournir un titre à toutes les procédures lancées et tous les graphiques produits : en effet, cela rend plus facile la lecture des résultats et graphiques dans les fenêtres correspondantes, mais surtout leur recherche à partir de la fenêtre Results, car touts les titres sont repris dans cette fenêtre. Les utilisateurs de SAS qui procèdent ainsi s'organisent beaucoup mieux, et gagnent du temps pendant leurs séances ou leur boulot.

2. Le systeme ODS (Output Delivery System)

Remarque : Ce paragraphe se contentera d'évoquer les commandes de base relatives à l'ODS : tous les chapitres de l'aide de SAS dont il est fait allusion ci-dessous sont compris dans l'onglet suivant de l'aide :
SAS Products → *Base SAS* → *Output Delivery System*

L'ODS n'est pas une fonctionnalité optionnelle : c'est, comme son nom l'indique, le système qui permet d'attribuer telle et telle *forme* (html, rtf, sortie à l'écran, xml, pdf, ps,...) à une sortie de procédure. Par défaut, un seul mode (le mode *listing*) est actif : celui qui consiste à afficher à l'écran (dans la fenêtre Output le plus souvent) le résultat des procédures. Pour le désactiver, il faut utiliser la commande globale

```
ods listing close;
```

et pour le réactiver, utiliser la commande globale `ods listing;` (tout court). Si l'on veut créer une version HTML de l'affichage des résultats à l'écran, il suffit d'inclure le code entre l'instruction `ods html;` et l'instruction `ods html close;` . Donnons un exemple dans lequel on contrôle la destination du fichier html en question (voir paragraphe I-4 au sujet des *fileref* et de la commande `filename`) :

```
filename nom="~/sas/essaiods.html";
ods html file=nom;
proc univariate data=don;  var x;  run;
ods html close;
```

Il est bien sûr possible de ne pas avoir recours à la commande `filename` et spécifier l'adresse du fichier directement dans le champ `file=` de l'instruction `ods html` (ici `ods html file="~/sas/essaiods.html";`). Une option intéressante de l'instruction `ods html` est `style=` qui permet de spécifier un certain style graphique (un peu comme une feuille de style) ; des styles disponibles sont `brick, d3d, statistical, journal, watercolor, analysis`,... mais il est aussi possible de customiser différents aspects (couleurs, bordures,...) à l'aide d'autres options [52].

On peut également créer des fichiers pdf, ps ou rtf, suivant la même syntaxe, avec les instructions `ods pdf`, `ods ps` ou `ods rtf`.

C'est simple d'utilisation de prime abord, mais au final les résultats peuvent être faussement esthétiques (jolies fontes, mais des infos superflues ou des noms de colonnes peu évocateurs dans les tableaux par exemple). A l'usage, il vaut mieux apprendre à bien formatter ses propres sorties, pour que l'ensemble ait bonne mine et que l'on obtienne exactement ce que l'on veut ; le mieux est d'aller voir les *Sample SAS Programs*, qui donnent de bonnes idées, et montrent que l'apprentissage de l'ODS est loin d'être si enfantin qu'il n'y parait (notamment la notion d'*ODS Document*...). Un bon point de départ pour affiner ses connaissances en ODS est de commencer par lire tout le contenu de l'onglet *Concepts* → *Output Delivery System : Basic Concepts*.

On peut produire différents formats de sortie sans relancer à chaque fois la procédure ou la série de procédure concernée : pour cela, il suffit de soumettre plusieurs instructions ODS à la suite, en préambule, par exemple

```
ods html file="result.htm"; ods pdf file="result.pdf"; proc ......; run; ods html close; ods pdf close;
```

Nous n'approfondirons pas la sujet, et nous contenterons d'indiquer une instruction ODS très intéressante en pratique, l'instruction `select`. Elle permet de restreindre la sortie à certaines tables seulement : par exemple, la procédure `univariate` produit de nombreux résultats, et de nombreuses *tables* à l'affichage, et l'on peut ne pas avoir envie de tout afficher (mode listing) ou tout sauvegarder (autre format). Il faut alors spécifier les noms des tables que l'on souhaite conserver. Par exemple, pour `univariate`, *BasicIntervals* désigne la table contenant les IdC pour la moyenne, l'écart-type, et la variance, et *Quantiles* celle contenant les déciles, quartiles, et autres quantiles. Pour `princomp`, *Eigenvalues* désigne la table contenant les valeurs propres correspondant à l'ACP qui est réalisée, etc... La syntaxe à utiliser est

```
ods select Quantiles Extremeobs;  proc univariate data=don; var x; run;
```

On constatera alors que seules les tables des quantiles et des valeurs extrêmes seront affichées à l'écran. Si une sauvegarde html était en cours avec une instruction `ods html`, alors seules ces deux tables seront reproduites dans le fichier html. La liste des "noms ODS" se trouve dans l'onglet *Details* → *ODS Table Names* de chaque procédure.

Remarque :
- On sent bien combien SAS "sent le vieux", puisque ODS est presente un peu "a part", et non tout a fait d'emblee comme fonctionnalite importante de SAS... Ceci se ressent au niveau de la qualite de l'aide concernant les fonctionalites ODS, qui est plutôt mediocre, surtout par manque d'illustrations. Cependant, les onglets `ods html`, `ods ps` et `ods pdf` de *ODS Language Statements* → *Dictionary of ODS Language Statements*, et l'onglet *Concepts* → *Output Delivery System and the DATA Step*, restent assez interessants. Une description des *options* peut être accedee en cliquant sur le lien *Options* precedant la liste des options d'une instruction ODS.
- Attention, pour supprimer uniquement la destination *listing* (sortie a l'ecran), il faut utiliser la commande `ODS listing close;`, et non pas la commande globale `options noprint;`, qui, elle, ferme *toutes* les destinations ODS...
- Depuis la version 9 de SAS, des instructions `ods graphics` experimentales sont disponibles, elles permettent de produire des graphiques esthetiques a partir de nombreuses procedures statistiques (comme `REG` ou `PRINCOMP` par exemple). Pour des details, voir l'aide d'ODS et l'aide de chaque procedure concernee.

52. *cf* procédure `template`, décrite justement dans le chapitre ODS, ainsi que le *Style Editor*, assez utile et fourni dans le package *Enterprise Guide* de SAS.

3. Concatenation, Fusion, Mise a jour de tables SAS

La mise en forme des données sous une forme satisfaisante est une activité importante du statisticien, et jusque là la question de la réunion ou fusion de plusieurs tables en une seule n'a pas été évoquée. L'objectif de ce paragraphe est d'en présenter quelques aspects, en se basant sur des exemples simples (qui permettront de comprendre les mécanismes et donc de les appliquer à des tables SAS plus proches de celles manipulées sur le terrain). Là encore l'aide de SAS est d'un précieux secours : voir notamment les *sample programs* ainsi que les onglets *Base SAS → SAS Language Concepts → DATA Step Concepts → Reading, Combining, and Modifying SAS Data sets* et *Base SAS → SAS Language Dictionary → Dictionary of Language Elements → Statements → SET/MERGE/UPDATE....*

Tout d'abord, la **concatenation** (*i.e.* l'empilement) de deux tables se fait à l'aide d'une instruction `SET` suivie des noms des deux tables que l'on souhaite concaténer. On peut aussi en concaténer plus que deux à la fois en fournissant les noms les uns à la suite des autres. Si certaines variables n'existent pas dans chacune des deux tables, alors des données manquantes sont indiquées aux endroits correspondants. Par exemple,

```
data table1;  input nom $ x1 x2;
 cards;
 Gautier  4 5
 Jones    6 4
 Worms    2 3
 ;
```

```
data table2; input nom $ x1 y;
 cards;
 Krank   2 12.8
 Silver  4 10.2
 ;
```

```
data result; set table1 table1sup;
```

va produire la table

nom	x1	x2	y
Gautier	4	5	·
Jones	6	4	·
Worms	2	3	·
Krank	2	·	12.8
Silver	4	·	10.2

Si maintenant les deux tables contiennent des observations qui partagent la même valeur d'une certaine variable commune, et que ces observations correspondent à un même individu statistique, alors une concaténation n'est certainement pas appropriée. Par exemple, la double instruction `SET` permet d'"accoler" des tables qui sont triées de la même manière suivant une variable commune (ici, `nom`)

```
data table1sup;  input nom $ y;
 cards;
 Gautier  11.2
 Jones    10.3
 Worms     9.7
 ;
```

```
data result;  set table1; set table1sup;
```

va produire la table

nom	x1	x2	y
Gautier	4	5	11.2
Jones	6	4	10.3
Worms	2	3	9.7

alors que `data result; set table1; set table2;` va produire la table dénuée de sens ci-dessous

nom	x1	x2	y
Krank	2	4	12.8
Silver	4	3	10.2

l'instruction `set table1; set table1sup;` a **fusionne** les deux tables, et l'a fait correctement car les deux tables étaient bien ordonnées de la même façon et surtout elles contenaient exactement les "mêmes" observations. Le même résultat aurait été obtenu par le biais de l'instruction `MERGE ....; BY` ci-dessous, qui est plus naturellement utilisée pour fusionner des tables :

```
data result; merge table1 table1sup; by nom;
```

l'instruction `merge` ne fonctionne que si les tables en question sont toutes deux triées de la même façon suivant la variable spécifiée après le mot-clef `by`. La différence avec le double `set` est que si maintenant on considère plutôt la table `table1sup` *sans l'individu Jones*, alors l'instruction `merge` fait toujours ce qu'on lui demande (avec une donnée manquante pour Jones pour la variable `y`) tandis que les deux `set` successifs ne font pas du tout ce qu'il faut (l'individu Jones a complètement disparu et la valeur de `x2` pour Worms est celle de Jones...) :

```
data table1sup;  input nom $ y;
 cards;
 Gautier  11.2
 Worms     9.7
 ;
```

```
data result;  merge table1 table1sup; by nom;
```

```
data result;  set table1; set table1sup;
```

```
proc print; run;
```

```
proc print; run;
```

vont respectivement produire les tables

nom	x1	x2	y
Gautier	4	5	11.2
Jones	6	4	·
Worms	2	3	9.7

et

nom	x1	x2	y
Gautier	4	5	11.2
Worms	6	4	9.7

Moralite : utiliser l'instruction `merge` dès qu'une fusion de tables est nécessaire, impliquant deux tables triées suivant une même variable et pouvant ensuite contenir des variables distinctes (et ne contenant pas forcément les mêmes individus statistiques, mais dans ce cas des données manquantes seront introduites).

Pour revenir à la concaténation, il est aussi possible de concaténer deux tables contenant les mêmes variables, tout en obtenant directement une table qui sera triée suivant une certaine variable (par rapport à laquelle les tables de départ doivent être elles-mêmes préalablement triées). Par exemple,

```
data result; set table1 table1sup; by nom;
```

va produire la table

nom	x1	x2	y
Gautier	4	5	·
Jones	6	4	·
Krank	2	·	12.8
Silver	4	·	10.2
Worms	2	3	·

Sur cet exemple, cela a peu d'intérêt, puisque chaque table contient des variables que l'autre ne contient pas. En outre, cela ne fait qu'économiser une procédure `sort` (ce qui peut avoir un intérêt si les tables en question sont volumineuses).

Il resterait encore beaucoup à dire sur le potentiel de l'instruction `set`, ainsi que sur le fonctionnement de l'instruction `merge ... by`[53], mais parlons maintenant de la **mise a jour** des tables SAS.

Par mise à jour on entend modification de certaines valeurs de variables pour certaines observations d'une table SAS appelée *table maîtresse* (*master data set*), les nouvelles valeurs étant contenue dans une table appelée *table de transaction*. A quelques détails près (relatifs au traitement des valeurs manquantes), les mises à jour peuvent être aussi bien effectuées par une instruction `update` *master transaction*`; by` *var* que par une instruction `merge`. Par exemple, si `table1sup` désigne la première des 2 versions vues dans ce paragraphe, et `table1MAJ` désigne la table ci-dessous contenant les modifications à apporter aux données de départ, alors

```
data table1MAJ;  input nom $ x1 y;
 cards;
 Gautier 0 10
 Grelon  1 5
 Worms   0 .
 ;
```

```
data mix;     merge table1 table1sup; by nom;

data result;  update mix table1MAJ;   by nom;
```

va produire la table `result`

nom	x1	x2	y
Gautier	0	5	10
Grelon	1	·	5
Jones	6	4	10.3
Worms	0	3	·

On remarquera que dans la table de transaction, toutes les variables n'ont pas besoin d'être présentes si elles n'ont à subir aucune modification (ici, `x2` n'est pas présente dans la table `table1MAJ`). Bien entendu, une telle instruction ne prend tout son intérêt que lorsque de nombreuses mises à jour sont à effectuer, car si seulement une ou deux observations doivent être modifiées, cela peut se faire à la main avec de simples instructions `IF`...

Il existe aussi une instruction `modify`, qui réalise des mises à jour dans le même esprit que le fait l'instruction `update`, à quelques nuances près qu'il est inutile d'évoquer dans ce document (voir l'aide).

53. quand par exemple plusieurs observations d'une des tables impliquée dans la instruction `merge` partage la même modalité de la variable argument du `by` : c'est ce que le manuel de SAS appelle *one-to-many merging* et *many-to-one merging*...

4. Quelques pistes et liens internet pour poursuivre [*liens un peu anciens*]

On ne le répétera jamais assez : l'utilisation intensive et systématique de l'aide de SAS pour résoudre un problème est fortement conseillée. En outre, le site internet de SAS fournit quelques FAQ qui peuvent aider à débloquer rapidement des situations, et la bibliothèque universitaire du site de Versailles (bâtiment Buffon) contient des livres empruntables sur plusieurs sujets SAS pointus (ou au contraire introductifs).

Toutefois, des spécialistes (liés à la société SAS Institute ou non) de par le monde fournissent des documents très pédagogiques et bien structurés, des FAQs et des dossiers thématiques, sur la programmation et les procédures SAS. C'est pourquoi sont indiqués quelques liens sur la toile (la plupart en anglais), pour clore ce document :

- SAS Online Doc : accessible depuis tout navigateur, en version pour le Web ou bien au format pdf

  `http://support.sas.com/documentation/onlinedoc/sas9doc.html`

  Noter que le site de SAS, section Support, fournit potentiellement de l'aide egalement. Par exemple, la page menant aux FAQs theme par theme ou procedure par procedure est :

  `http://support.sas.com/techsup/faq/products.html`

- Manuel en francais realise par l'equipe de l'universite de Toulouse (page du professeur Besse) :

  `http://www.lsp.ups-tlse.fr/Besse/enseignement.html`

- Pages de l'Universite de Californie (Los-Angeles, UCLA) : contient de nombreux tutoriels, FAQs, donnees, conseils, pour de nombreux langages ou logiciels statistiques (R, Stata, SPSS, LaTeX, html,...), incluant même des cours en ligne sous forme de videos-tutoriels pour certains sujets :

  `http://www.ats.ucla.edu/stat/` et `http://www.ats.ucla.edu/stat/sas/default.htm`

- Transparents tres pertinents traitant en details des nombreux aspects des etapes DATA notamment (par P.Spector, Berkeley)

  `http://www.stat.berkeley.edu/classes/s100/sas.pdf`

- Pages d'un service de l'Universite du Texas, contenant des tutoriels interessants pour debuter en SAS (ainsi que d'autres manuels introductifs pour SAS et d'autres logiciels, mathematiques ou statistiques) :

  `http://www.utexas.edu/its/rc/tutorials/` (tutoriels SAS I et SAS II)

- Une page ideale pour apprendre comment faire fonctionner SAS en arriere plan sous Unix (batchmode)

  `http://www.nyu.edu/its/socsci/Docs/sas_batch.html`

- Une page en francais contenant des exercices et TPs (batchmode)

  `http://www.jeannot.org/~js/sas/`

- Un autre site en francais, particulierement reussi au niveau de la syntaxe des etapes DATA

  `http://mrw.wallonie.be/dgrne/sibw/outils/methodo/SAS/home.html`

Contribution bienvenue : l'ecriture d'un tel document, avec comme seul relecteur moi-même, implique generalement un nombre d'erreurs de frappe et de syntaxe qui ne s'annule helas vraiment jamais, aussi je serai reconnaissant a tous les etudiants serieux qui y ont releve des bourdes ou des coquilles, de me les communiquer...

X. Index