

INSTITUT DE STATISTIQUE
de l'Université Pierre et Marie Curie

Cycle Supérieur 1^{ère} année
2011-12

Introduction au logiciel SAS
François-Xavier LEJEUNE

Plan du cours

Séance n°1	Débuts en SAS : Généralités - Présentation - Étape DATA.....	3
Séance n°2	Procédures usuelles - Graphiques	11
Séance n°3	Proc REG - SAS / Insight	17
Séance n°4	Programmation en SAS (1) : macro-langage - O.D.S.....	23
Séance n°5	Programmation (2) : calcul matriciel avec SAS / IML.....	29
Annexe	Projet SAS (sujet 2010-11)	

Dernière modification le 18 février 2012

Séance n°1 Début en SAS : Généralités - Présentation - Étape DATA

L'objectif de ce cours est de proposer une entrée en matière à l'utilisation du logiciel SAS tout en illustrant divers éléments de Statistique élémentaire. SAS étant un logiciel complet largement adopté dans l'industrie, il est à noter qu'une bonne maîtrise du logiciel est un pré-requis récurrent dans les offres de stages et d'emplois destinées aux étudiants de filière Statistique.

Généralités sur SAS

SAS – Statistical Analysis System – est un logiciel complet qui couvre une large gamme des méthodes d'analyse en Statistique. De conception américaine, il est développé par la société SAS Institute Inc. basée à Cary, en Caroline du Nord ; et a acquis, depuis sa mise sur le marché en 1976, une situation dominante dans de nombreuses branches d'activités économiques. Le système SAS se présente sous la forme d'un ensemble de modules logiciels adaptés pour la gestion et l'analyse statistique de gros volumes de données (tableaux de plusieurs giga-octets) et la création de rapports de synthèse. Un avantage important sur d'autres logiciels de Statistique présents dans l'industrie tels que SPSS, SPAD 7, Simca-P+ (en chimométrie), par exemple, vient des possibilités avec SAS pour programmer ses propres procédures. Nous verrons dans la seconde partie de ce cours comment automatiser l'enchaînement de plusieurs traitements. Les principales (relatives) faiblesses de SAS sont les suivantes : coût élevé, un langage de programmation peu avancé laissé à l'utilisateur – en fait, une combinaison de commandes empruntées à plusieurs langages : macro, IML, SQL, *etc.* –, des graphiques moyens, *etc.* Créée en 1983, la filiale française est basée à Grégy-sur-Yerres (Seine-et-Marne). Des versions équivalentes de SAS existent pour chacun des environnements Unix et Windows (depuis 1990). La dernière version V9 (2003) est distribuée en France depuis 2004 et sa toute dernière évolution V9.3 est sortie en juillet 2011.

1. Mise en route et présentation de l'interface

Pour activer la version Windows disponible à l'ISUP, aller dans

démarrer > Tous les programmes > SAS > SAS 9.3 (Français)

ou bien, à partir du *bureau*, cliquer directement sur le pictogramme du logiciel SAS. L'environnement SAS est constitué de 3 fenêtres principales :

SAS Program Editor (F5)	éditeur de texte de SAS dans lequel on écrit les commandes à exécuter ;
SAS Log (F6)	fenêtre de gestion de la session, compilation des commandes ligne par ligne ;
SAS Output (F7)	listing des sorties d'un programme SAS.

Remarques La fenêtre *Log* est très importante pour s'assurer de la bonne exécution d'un programme avec les **notes en bleu**, les **avertissements (warnings) ou erreurs non fatales en vert** ...et les **erreurs fatales en rouge** ! Les fenêtres *Log* et *Output* ne sont pas purgées automatiquement par SAS entre les exécutions ; il est donc conseillé d'en effacer régulièrement le contenu en utilisant la commande `Edit > Clear all` ou l'**icône « page blanche »**. Dans la version 9.3, les sorties 'texte', dont certaines agrémentées de graphiques, s'affichent par défaut dans une fenêtre **HTML Results Viewer** au lieu de la fenêtre listing Output.

D'autres fenêtres sont aussi utiles :

SAS Explorer	pour visualiser et accéder aux bibliothèques et aux tables SAS ;
SAS Toolbox	pour donner des ordres à l'interface SAS ;
SAS Results	arborescence pour accéder directement aux sorties textuelles et graphiques.

Dans la fenêtre *Program Editor* (ou *Enhanced Editor* sous Windows), un programme SAS est généralement composé :

1. d'une ou plusieurs **étapes DATA** : définition / lecture / modification d'un ou plusieurs tableaux de données ;
2. puis d'une ou plusieurs **étapes PROC** : toutes les procédures agissant sur les tableaux de données.

Le système SAS est constitué d'un ensemble de **logiciels modules** qui viennent compléter les fonctionnalités du logiciel. Parmi les principaux modules, citons : **SAS/BASE** – composante indispensable de SAS pour pouvoir lire, modifier et analyser des fichiers de données, pour éditer des rapports et pour utiliser les langages SAS de base, SQL et macro), **SAS/STAT** (ensemble de procédures de modélisation, de classification et de statistiques descriptives), **SAS/GRAPH** (édition de graphiques haute résolution), **SAS/INSIGHT** (analyse statistique interactive), **SAS/IML** (*Interactive Matrix Language*), **SAS/SQL** (*Structured Query Language*), **SAS/ETS** (*Econometrics and Time Series* : étude des séries chronologiques), **SAS/AF** (*Application Facility* : création d'interfaces), **SAS/OR** (recherche opérationnelle), **SAS/FSP** (*Full Screen Products* : manipulation de données en mode plein écran), **SAS/ASSIST** (interface cliquable pour créer des programmes SAS), *etc.* Il est à noter que certaines analyses sont redondantes dans SAS et peuvent être traitées par plusieurs modules à la fois.

✓ **Aide de SAS** : Help > SAS Help and Documentation ou touche **F1**.

2. Étape DATA

Un fichier de données ne peut être reconnu, lu ou traité par SAS que s'il est dans un format spécifique. Nous appellerons *TABLE SAS* un tel fichier écrit dans ce format où les colonnes représentent les *variables* et les lignes les *valeurs observées* des variables. Les tables SAS peuvent être temporaires ou permanentes. Une séquence de lecture des données se présente de la façon suivante :

```
DATA < nom de la table SAS >;
INPUT < liste des variables >;
CARDS;
< les données sont entrées "à la main" >
;
RUN;
```

L'instruction **CARDS** – équivalente à **DATALINES** et à **LINES** – indique le début de la saisie des observations (une seule observation par ligne). Les données sont alors séparées par des espaces, *une valeur manquante étant représentée par un point*. Notons que, dans la liste des variables, le séparateur est encore un blanc. Enfin, l'instruction **RUN** (facultative mais conseillée) termine l'étape DATA.

Exemple de programme

```
00001 /* exemple de programme; */           → ligne de commentaire
00002 DATA exemple;
00003 INPUT nom$ ddn DDMYY10. CSP$ auto$ nbsin1 nbsin2;
00004 CARDS;
00005 Pierre 08/04/1944 lib C5 0 2
00006 Paul 12/03/1976 arti 307 0 0
00007 Jacques 09/10/1953 cadre C3 1 2
00008 Carole 11/12/1964 lib modus 1 0
00009 Caroline 21/08/1970 cadre polo 0 0
00010 Nathalie 14/06/1962 cadsup 607 3 4
00011 ;
00012 RUN;
```

Remarque Ici, la numérotation des lignes de code est spécifique à la version Unix.

✓ **Variables SAS :** le nom des variables nominales est toujours suivi du symbole « \$ ». Les variables numériques peuvent être qualitatives (par ex. dichotomiques / binaires ou catégorielles...) ou quantitatives (discrètes ou continues). Le nom d'une variable comporte au plus 32 caractères, commence par une lettre « A-Z » ou « _ » (underscore) et ne doit pas contenir d'espace(s) ou de symboles spéciaux tels que « & », « % », « \$ », « # », *etc.* L'indication **DDMMYY10.** qui suit *ddn* s'appelle un **informat** pour que SAS comprenne que la donnée lue est une date écrite sous la forme *jj/mm/aaaa*. Dans une table SAS, la date est stockée comme le nombre de jours qui la sépare de la date référence du 1^{er} janvier 1960 (nombre négatif si date antérieure, nombre positif si date postérieure, la date référence étant la date zéro).

✓ **Compilation :** Menu > Run > Submit ou icône « bonhomme » ou touche **F3**.

SAS est un langage interprété. Lorsque l'on lance un programme, les lignes de code défilent une à une dans la fenêtre *Log*. Dans notre exemple, un message indique la création de la table *exemple* qui compte 6 lignes et 6 colonnes. Sur la version Unix, les lignes de code disparaissent de la fenêtre *Program Editor* lors de l'exécution. Celles-ci peuvent être « rappelées » ensuite avec la commande Menu > Run > Recall Last Submit.

✓ **Visualisation des données :** une table SAS peut être visualisée, corrigée et modifiée depuis une fenêtre **VIEWTABLE** accessible par SAS Explorer. Il importe de fermer cette fenêtre ensuite pour pouvoir accéder à cette table par des étapes DATA ou PROC. On peut également visualiser une table SAS avec la procédure d'impression **PROC PRINT**.

✓ **Bibliothèques (libraries) de SAS :** toute nouvelle table est par défaut conservée dans la bibliothèque *temporaire* **Work** et ne reste en mémoire que le temps de la session SAS en cours. Pour conserver une table au-delà de la session, il faut l'enregistrer dans une bibliothèque *permanente*. L'instruction **LIBNAME** ou l'icône « Add New Library » permet alors d'indiquer le chemin d'un répertoire où les tables seront systématiquement conservés au format SAS « .sas7bdat ».

Exemple : Création d'une bibliothèque permanente « mabib » avec LIBNAME

```
libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';
/* mabib est un nom qqc faisant référence à la bibliothèque permanente */
DATA mabib.exemplebis;      /* exemplebis est permanente */
  SET exemple;              /* exemple est temporaire */
RUN;
```

À propos des autres bibliothèques de SAS dans Explorer :

MAPS contient des données cartographiques pour réaliser des fonds de cartes ;

SASHELP contient des données fournies par SAS à des fins d'exercices (*class, air, shoes, etc.*) ;

SASUSER est une bibliothèque personnelle (chaque utilisateur a sa propre *SASUSER*) permanente (contrairement à *WORK*). Cette bibliothèque contient entre autres les préférences utilisateur pour personnaliser l'environnement de sa session SAS (couleurs, polices, taille et positionnement des fenêtres, *etc.*). Il est possible d'y conserver ses données, mais il est commode aussi de stocker des données en déclarant et en utilisant une nouvelle bibliothèque avec l'instruction **LIBNAME**.

Quelques commandes pour modifier une table SAS dans une étape DATA :

- `IF nom = 'Jacques' THEN naiss = -2280;` → *modification d'une valeur*
- `RENAME ddn = date_de_naissance;` → *changer le nom d'une variable*
- `KEEP nom nbsin1 nbsin2;` ↔ `DROP ddn CSP auto;` → *enlever des variables*
- `sintotal = nbsin1 + nbsin2;` → *ajouter une variable fonction des précédentes*

- `age = int((today()-ddn+1)/365.25);`
- `INPUT var_sup; CARDS; ...;` → ajouter une variable supplémentaire
- SAS reconnaît les fonctions mathématiques et statistiques usuelles : **ABS, ARCOS, ARSIN, ATAN, COS, COSH, SIN, SINH, EXP, LOG** (népérien), **MIN, MAX, RANGE, INT, ROUND, CEIL, FLOOR, SQRT, N** (nb de valeurs disponibles), **NMISS** (nb de valeurs manquantes), **SUM, TAN, TANH, MEAN, VAR, STD**, etc.
- `if nbsin2 = 0 then DELETE;` → enlever des observations

Fusionner des tables SAS :

Il est possible de combiner plusieurs tables SAS ayant des individus ou des variables en commun en une seule table en opérant des concaténations horizontales ou verticales.

1/ **Fusion horizontale** – Il s’agit d’aligner des tables SAS concernant les mêmes individus mais comportant des variables différentes. La fusion se fait dans une étape DATA avec les instructions **MERGE** ou **SET**.

- `MERGE TAB1 ... TABk;` ou `SET TAB1; ...; SET TABk;` → fusion horizontale

Si une variable de TAB1 porte malencontreusement le même nom qu’une variable de TAB2, le fichier fusionné gardera les valeurs de la variable de TAB2. Si une variable est commune à TAB1 et TAB2, il est aussi possible d’aligner les deux tables par rapport à cette variable qui joue le rôle de clé de fusion. Pour ce faire, TAB1 et TAB2 doivent d’abord être ordonnées selon les valeurs croissantes de la variable clé en utilisant la **PROC SORT** (procédure de tri) et l’instruction **BY**. On utilise ensuite **MERGE** avec **BY** pour opérer la fusion.

```
PROC SORT DATA=TAB1;
  BY toto; /* toto est une variable commune à TAB1 et TAB2 */
RUN;

PROC SORT DATA=TAB2;
  BY toto;
RUN;
```

puis

```
DATA NEWTAB;
  MERGE TAB1 TAB2; → fusion horizontale
  BY toto;
RUN;
```

2/ **Fusion verticale** – Il s’agit d’empiler les lignes de plusieurs tables SAS. La fusion se fait dans une étape DATA avec l’instruction **SET** ou en utilisant la **PROC APPEND**. Comme pour la fusion horizontale, si une variable est commune à toutes les tables, il est possible d’obtenir une table fusionnée où les observations sont rangées dans l’ordre croissant de la variable commune. Pour ce faire, on s’assurera d’abord que toutes les tables sont ordonnées selon les valeurs croissantes de la variable commune et on utilisera la **PROC SORT** si nécessaire.

```
DATA NEWTAB;
  SET TAB1 ... TABk; → fusion verticale
  BY toto;
RUN;
```

En utilisant la PROC APPEND :

```
PROC APPEND BASE=TAB1 DATA=TAB2 FORCE; → fusion verticale
RUN;
```

La table fusionnée prend le nom de la table de base TAB1 (qui est écrasée pendant de l'opération) avec les variables de TAB1 à laquelle se sont ajoutées les observations de TAB2. Il est donc préférable que TAB1 et TAB2 comportent exactement les mêmes variables. Autrement, si les variables de TAB2 ne sont pas toutes communes à TAB1, la fusion est possible avec l'option **FORCE**. Les valeurs des variables de TAB1 qui ne sont pas dans TAB2 seront alors considérées comme des valeurs manquantes pour TAB2 ; et les valeurs des variables de TAB2 qui ne sont pas dans TAB1 ne seront pas prises en compte.

IMPORTER les données d'un fichier dans une table SAS :

Il est possible d'importer sous SAS des fichiers texte « .txt », Excel « .xls » ou « .csv » (*Comma Separated Value* : délimiteur point-virgule). L'instruction **INFILE** indique le chemin du fichier contenant les données à importer.

- ❑ Syntaxe pour un fichier « .csv » :

```
DATA < nom de la Table SAS >;
  INFILE 'c: \ ... \ fichier.csv' delimiter = ';'; /* ou dlm = ';' */
  INPUT < liste des variables >;
RUN;
```

On peut aussi bien déclarer le chemin du fichier de données avant l'étape DATA – de préférence en tête du programme – avec l'instruction **FILENAME** :

```
filename file1 'c: \ ... \ fichier.csv';
/* file1 est un nom qqc faisant référence au chemin du fichier */
DATA < nom de la Table SAS >;
  INFILE file1 dlm = ';';
  INPUT < liste des variables >;
RUN;
```

Remarque 1 Pour un fichier texte, des valeurs usuelles de *dlm* sont ' ' (espace), ',' (virgule), '|' (barre verticale), ';' (point virgule) ou '09'x (tabulation). On peut aussi utiliser l'instruction *dlmstr* pour des valeurs séparées par une chaîne de caractères : ex. **DLMSTR** = '~\^' lorsque la séquence 'tilde-backslash-accent circonflexe' est utilisée comme séparateur.

Remarque 2 Dans un fichier « .sas », par souci de lisibilité, il est préférable de placer l'ensemble des *libname* et des *filename* en tête de programme avec les options générales sur l'affichage des sorties dans *Output*. L'emploi de l'instruction *filename* permet d'accéder simplement aux données d'un fichier déplacé (ou renommé) en modifiant simplement le chemin du fichier sans avoir à chercher dans tout le programme...

- ❑ Syntaxe pour un fichier Excel :

```
filename file2 DDE "Excel|m:\courssas\[import.xls]feuille1!L2C1:L19C4";
/* DDE pour Dynamic Data Exchange, Excel étant l'"Engine" */
DATA < nom de la Table SAS >;
  INFILE file2;
  INPUT < liste des variables >;
RUN;
```

Les données à importer sont dans l'onglet 'feuille' du fichier Excel *import.xls* de la ligne 2 à la ligne 19 et sur les 4 premières colonnes. Le fichier Excel doit impérativement être ouvert pour mener cette opération.

Plus simplement, nous pourrions encore importer des données depuis un fichier avec la commande `File > Import Data` en précisant ensuite le type et le chemin du fichier source. Enfin, une façon de récupérer des données sur des pages web « .html » ou « .htm » consiste à *copier* et *coller* les données dans un fichier via un éditeur de texte (*nedit* ou *emacs* sous Unix, le *bloc-note* sous Windows, par exemple), à formater les valeurs avec un séparateur pour ensuite les importer dans SAS. Indiquons enfin une troisième possibilité qui consiste à utiliser une **PROC IMPORT** avec la syntaxe *ad hoc*.

EXPORTER le contenu d'une table SAS dans un fichier :

Pour la démarche inverse, une étape **DATA** peut aussi servir à créer un fichier avec les données d'une table SAS existante. Cette étape **DATA** diffère de celles vues précédemment puisque l'objectif n'est pas ici de construire une nouvelle table mais d'utiliser les éléments d'une table SAS pour générer un fichier texte « .txt » ; on utilisera alors une instruction **DATA _NULL_** ; L'instruction **FILE** indique ensuite le chemin avec le nom du fichier qui doit recevoir les données exportées. L'instruction **PUT** s'utilise de manière analogue à l'instruction **INPUT** pour indiquer le nom des variables à récupérer. Elle permet, si on le souhaite, d'écrire les données en utilisant un **pointeur** d'écriture (position relative ou absolue pour l'écriture d'une valeur sur la ligne d'un individu) ainsi qu'un **format** d'écriture (la forme sous laquelle on va écrire la valeur). Ce format pourra aussi bien être un format SAS prédéfini qu'un format utilisateur préalablement défini avec une **PROC FORMAT**. La valeur exportée sera alors la valeur formatée au lieu de celle stockée dans le format SAS par défaut. Contrairement à **INPUT**, il n'est pas nécessaire de signaler les variables nominales par un « \$ » ; lors de l'export, le type des variables SAS est déjà connu.

Remarque 3 La **PROC FORMAT** et l'instruction **format** dans une étape **DATA** ou une procédure quelconque servent à spécifier les propriétés d'affichage des valeurs. Une autre utilisation de la **PROC FORMAT** qui est illustrée dans l'exercice 2 de la séance n°2 permet de discretiser des variables continues.

- Syntaxe pour un fichier « .txt » :

```
filename file3 'c: \ ... \ fichier.txt';

DATA _NULL_;
SET < nom de la table SAS >;
FILE 'c: \ ... \ fichier.txt' dlm = 'délimiteur';
/* ou FILE file3 avec l'instruction filename */
PUT < pos_point_1 > < var_1 > : < format_1 >
    ... < pos_point_p > < var_p > : < format_p >;
RUN;
```

Exemple d'export de données

```
DATA _NULL_;
SET exemple;
FILE 'm:\courssas\exemple.txt';
PUT nom 1-6 @9 ddn:DATE9. +1 sin1;
RUN;
```

Dans l'exemple, les données relatives à la variable *nom* sont affichées sur les colonnes 1 à 6 de la ligne d'un individu (tant pis si le nom a plus de 6 caractères !). Le curseur se place ensuite en position 9 (soit en 8^{ème} colonne) pour afficher la date de naissance *ddn* dans le **format** prédéfini **DATE.9**. Enfin, lorsqu'une valeur est inscrite, le curseur laisse (par défaut) un espace pour inscrire la valeur suivante ; avec la commande **+1**, le curseur est déplacé d'une colonne supplémentaire pour finalement afficher les données de *sin1*.

Pour exporter des données depuis un fichier, nous pourrions encore utiliser la commande `File > Export Data` (plus simple) ou bien une **PROC EXPORT** (plus compliqué...).

EXERCICES

Exercice n°1

La table *champ10* contient les résultats du County Championship de cricket disputée en Angleterre et au pays de Galles en 2010. Les variables indiquent respectivement pour chaque comté : le nom de l'équipe *Team*, le nombre victoires *W (Wins)*, de défaites *L (Losses)* et de matchs nuls *D (Draws)*, les points marqués à la batte *Bat (Batting pts)* et au lancer *Bowl (Bowling pts)*, les points déduits à la fin du championnat *Ded (Deducted pts)*.

1. Dans votre répertoire de travail, copier le répertoire « *datasas* » qui contient les tables SAS utiles pour ce cours. Récupérer ensuite les fichiers *champ10.txt*, *champ10.xls* et *champ10.csv* dans *datasas*.
2. Avec la commande `INFILE`, importer ces données sous SAS dans une table *Cricket* à partir des trois types de fichiers sources. Donner un titre à cette table (avec l'instruction `TITLE`) et l'afficher dans la fenêtre des sorties.
3. À partir de la table *Cricket*, créer une table *Cricket2* avec la variable supplémentaire : *pts*, le nombre total de points obtenus par une équipe sachant qu'une victoire rapporte 16 points et un match nul 3 points auxquels il faut ajouter les bonus acquis à la batte et au lancer et retrancher les points à déduire. Afficher la table *Cricket2*.
4. À partir de la table *Cricket2*, créer une table *Cricket3* qui reprend les données de la table *Cricket 2* triées par ordre décroissant de points *pts*. Afficher la table *Cricket3*. La première équipe a naturellement remporté le championnat alors que les deux dernières ont été reléguées en deuxième division.

Exercice n°2

Utiliser un moteur de recherche (par exemple, *Google*) pour trouver un tableau de données sur internet puis l'importer dans SAS. On pourra également chercher à récupérer l'un des nombreux tableaux de données de R.

Indication : On pourra copier/coller les données dans un fichier « *.txt* » en passant par un éditeur de texte (le *bloc-note*, par exemple).

Séance n°2 Procédures usuelles - Graphiques

Un programme SAS est composé d'une ou plusieurs étapes DATA (*cf.* séance n°1) suivies d'une ou plusieurs étapes PROC (ou procédures). Une **étape PROC** applique des traitements à des tables et génère des sorties, des graphiques ou encore de nouvelles tables. Les procédures PRINT, SORT et DELETE ont déjà été abordées au cours de la séance n°1. D'autres sont détaillées dans le polycopié de Josiane Confais.

1. Procédures usuelles

Syntaxe

Le début de l'étape est signalé par **PROC** suivi du nom de la procédure, puis du nom de la table à traiter. Il est important de spécifier la table sur laquelle on travaille avec l'instruction « DATA = < nom_tab > ». Par défaut, la procédure agit sur la dernière table créée. La fin de l'étape est reconnaissable à l'instruction **RUN**.

```
PROC < nom de la procédure > DATA = < nom de la Table SAS > < options >;  
...  
RUN;
```

Exemple : impression de la table *exemple*.

```
libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';  
  
proc print data=mabib.exemple; run;
```

Quelques instructions relatives aux procédures :

- L'instruction **PROC** (abréviation de procédure) est utilisée pour appeler une procédure SAS.

```
PROC < nom_proc > DATA = < nom_tab > < options >;
```

- L'instruction **VAR** (abréviation de variables) est suivie de la liste des variables à considérer.

```
VAR < liste des variables >;
```

- L'instruction **BY** permet le traitement par sous-groupes.

```
BY < sous-groupe de variables >;
```

Remarque La table SAS traitée par l'instruction BY doit avoir été triée suivant les variables mentionnées dans l'instruction.

- L'instruction **TITLE** définit le texte à imprimer en en-tête des pages de sortie. On peut ajouter jusqu'à 10 lignes de titres.

```
TITLEn 'titre';      où le n spécifie le nième titre.
```

- L'instruction **FOOTNOTE** définit le texte à imprimer en bas des pages de sortie. On peut ajouter jusqu'à 10 lignes de bas de pages.

```
FOOTNOTEn 'bas de page';      où le n spécifie le nième bas de page.
```

Titres SAS : *TITLE* et *FOOTNOTE* sont des instructions globales ; elles peuvent être utilisées n'importe où dans le programme (étape DATA, étape PROC ou en dehors). Quand on modifie un titre, tous les titres

hiérarchiquement inférieurs sont effacés. Pour supprimer les titres, on pourra utiliser l'instruction *TITLE* sans argument *TITLE* ;

Quelques exemples de procédures usuelles (liste non exhaustive...)

- Certaines de ces procédures agissent directement sur des tables SAS :

PROC CONTENTS : affiche les informations générales d'une table ;

PROC PRINT : procédure d'impression ;

PROC SORT : procédure de tri ;

PROC STANDARD pour centrer et réduire les données d'une table ;

```
PROC STANDARD DATA=nom_tab VARDEF=df MEAN=0 STD=1 OUT=reduit;  
RUN; /* VARDEF précise le diviseur dans le calcul de la variance */
```

PROC TRANSPOSE pour échanger les variables et les observations d'une table et réciproquement ;

- D'autres calculent des statistiques simples :

PROC MEANS édite des statistiques simples : moyenne, écart-type, maximum et minimum sur les variables numériques uniquement. Des options permettent d'obtenir des statistiques supplémentaires : *SUM*, *SKEWNESS* (coefficient d'asymétrie), *KURTOSIS* (coefficient d'aplatissement), *T* pour un test t de Student sur la moyenne... Il est possible de restreindre les variables d'étude avec l'instruction **VAR** ;

```
PROC MEANS DATA = nom_tab < options means >;  
RUN;
```

PROC CORR pour calculer les corrélations entre toutes les variables numériques ;

PROC UNIVARIATE pour obtenir un résumé statistique (complet) de chaque variable numérique ;

PROC FREQ pour effectuer des tris à plat et des tris croisés de variables à modalités discrètes et des tests d'indépendance du χ^2 .

- D'autres permettent de construire des modèles :

PROC REG pour effectuer un modèle de régression linéaire reliant une (ou plusieurs) variables(s) expliquée(s) (ou dépendante(s)) avec une (ou plusieurs) variable(s) descriptive(s) (ou prédicteur(s)) ;

```
PROC REG DATA=tabin < options reg >;  
MODEL < var_dép > = < var_exp > / < options >;  
PLOT < var_dép > * < var_exp > = 'symbol';  
OUTPUT OUT=tabout P=< nom1 > R=< nom2 >;  
/* sauvegarde des prédicteurs et des résidus dans une table tabout */  
RUN;
```

PROC GLM pour effectuer des modèles d'analyse de la variance non-orthogonaux, d'analyse de la covariance et des modèles linéaires généraux ;

```
PROC GLM DATA=tabin < options glm >;  
CLASS < variable(s) >;  
MODEL < var_dép > = < var_exp >;  
OUTPUT OUT=tabout keyword = < nom_de_variable >;  
RUN;
```

PROC GENMOD pour effectuer un modèle de régression linéaire généralisée. La régression logistique (disponible aussi avec la **PROC LOGISTIC**) est notamment une régression linéaire généralisée particulière lorsque la fonction de lien (« link function ») est la fonction *LOGIT*.

2. Graphiques à 2 dimensions

□ NUAGES DE POINTS

PROC PLOT/GPLOT produit des graphiques « en basse résolution » (avec des caractères) de nuages de points (scatter plots) à 2 dimensions ;

```
PROC GPLOT DATA = nom_tab < options plot >;
  PLOT < Y_var > * < X_var > = 'caractère' / < plot options >;
RUN;
```

Les options PLOT sont nombreuses, elles permettent entre autres de préciser les axes, les marques d'échelles et leurs libellés, de tracer des lignes supplémentaires, d'encadrer le graphe (BOX) ou de superposer plusieurs graphes (OVERLAY). Quelques options PLOT : BOX, OVERLAY, VREF = , HREF = ...

Remarque En exécutant la PROC PLOT sur un exemple, on pourra apprécier les faiblesses des graphiques affichés dans la fenêtre *Output* ! On préférera donc utiliser la **PROC GPLOT** qui permet d'obtenir des graphiques « en haute résolution » et dont la syntaxe est explicitée dans l'exercice n°4. Même remarque pour la PROC CHART décrite ci-dessous, on utilisera plutôt la **PROC GCHART**.

□ DIAGRAMMES

PROC CHART/GCHART trace « en basse résolution » des diagrammes en bâtons horizontaux (*hbar*), verticaux (*vbar*), à deux dimensions (*block*), des camemberts (*pie*) et des diagrammes en étoiles (*star*). Elle traite des variables quantitatives ou qualitatives. Les variables quantitatives sont codées explicitement ou automatiquement en classes.

```
PROC GCHART DATA = nom_tab < options chart >;
  VBAR < variable > / < chart options >; → édite des histogrammes verticaux
  HBAR < variable > / < chart options >; → édite des histogrammes horizontaux
  PIE < variables > / < chart options >; → édite des camemberts...
RUN;
```

Exemples d'options CHART :

- **SUMVAR** = variable numérique dont le cumul ou la moyenne est représenté ;
- **DISCRETE** spécifie que les valeurs sont de type discret (avec un nombre limité de modalités) ;
- **MIDPOINTS** = < valeurs > spécifie le point moyen de chaque bâton d'un histogramme, par exemple

```
MIDPOINTS = 10 TO 100 BY 10; → définit un histogramme à 10 bâtons centrés sur
les valeurs 10, 20, ..., 100 ;
MIDPOINTS = 'bleu' 'vert' 'rouge'; → définit un histogramme à 3 bâtons pour les
valeurs discrètes citées ;
```

- **GROUP** = < variable > représentation de plusieurs graphes côte à côte suivant les modalités d'une même variable ;
- **SUBGROUP** = < variable > découpage des barres ou colonnes selon la contribution de chacune des modalités d'une variable ;
- **LEVELS** = < nombre de classes > ;
- **TYPE** = spécifie ce que représente le graphique (par défaut une fréquence **FREQ**) ; autrement on peut choisir : **CFREQ** (fréquence cumulée), **CPT** (pourcentage cumulé), **PCT** (pourcentage)...

EXERCICES

Exercice n°1

La table *note* est extraite de « Multivariate Analysis », K.V. Mardia, J.T. Kent et J.M. Bibby (1979). Ces notes sont relatives à 88 étudiants d'une université anglaise ayant passé 5 épreuves notées par des notes entières de 0 à 100. Les 5 épreuves sont respectivement : mécanique, algèbre linéaire, algèbre des structures, analyse et statistique. En plus des 5 colonnes correspondant aux 5 épreuves, une première colonne indique le numéro de l'étudiant (de 01 à 88). Le but de cet exercice est d'observer les effets des lignes de code ci-dessous.

```
options pagesize=64 linesize=78 nodate; title; footnote 'TP2: Exercice 1';  
libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';
```

```
/* Sample Data Set: note */
```

```
proc print data=mabib.note; run;
```

```
proc means data=mabib.note /* vardef=n */; run;
```

```
proc univariate data=mabib.note /* plot vardef=n */; run;
```

```
proc corr data=mabib.note < options >; run;
```

Commentaires Par défaut, la PROC CORR calcule le coefficient de corrélation de Pearson. Autrement, on pourra spécifier l'option

- **SPEARMAN** : coefficient de corrélation des rangs de Spearman ; ou **KENDALL** : coefficient de corrélation de Kendall ; ou **HOEFFDING**.

D'autres options sont aussi disponibles parmi lesquelles :

- **NOSIMPLE** : pour supprimer l'impression des statistiques simples ;
- **COV** : pour imprimer les covariances (incompatible avec les options SPEARMAN et KENDALL !);
- **OUTP** = nom_tab : pour créer une table SAS contenant les coefficients de corrélation de Pearson.

```
proc plot data=mabib.note;  
  plot stat*meca anls*algb;  
run;
```

```
proc standard data=mabib.note vardef=df mean=0 std=1 out=reduit; run;
```

```
proc corr data=reduit cov vardef=df; run;
```

Exercice n°2

```
options pagesize=28 nodate; title; footnote 'TP2: Exercice 2';
```

```
libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';
```

```
/* Sample Data Set: htwt */
```

```
/* Décrire la table de données et noter l'effet de la commande "(obs=10)" */
```

```
proc print data=mabib.htwt (obs=10); run;
```

```
proc sort data=mabib.htwt out=TRI1;  
  by name;  
run;
```

```

proc print data=TRI1 (obs=10);
  id name;
  var sex age;
  title 'PROC SORT: Exemple 1';
run;

proc sort data=mabib.htwt out=TRI2;
  by sex age;
run;

proc print data=TRI2 (obs=10);
  id name;
  var sex age;
  title 'PROC SORT: Exemple 2';
run;

/* Tester la PROC MEANS avec et sans les options:
   n, mean, var, min, max, sum, nmiss et maxdec=1
   Notez l'effet de ces options. */

proc means data=mabib.htwt n mean min max sum nmiss maxdec=1;
  var age;
  class sex;
  title 'PROC MEANS: Exemple 2';
run;

proc corr data=mabib.htwt;
  var age height weight;
  title 'PROC CORR Exemple';
run;

proc univariate data=mabib.htwt;
  var age;
  title 'PROC UNIVARIATE Exemple';
run;

proc format;
  value $sex1
    'M' = 'Male'
    'F' = 'Female';
  value agef
    low-29 = '1'
    30-39 = '2'
    40-49 = '3'
    50-high = '4';
run;

proc print data=htwt;
  format sex $sex1. age agef.;
run;

```

Exercice n°3 : comment simuler des observations suivant une loi donnée ?

```

title 'Simulation de 1000 observations d''une loi gaussienne'; footnote 'TP2
Exercice3';

```

```

data simul;
  do i=1 to 1000;      → pour un échantillon de N observations
    y=rannor(10);     → pour une distribution Gaussienne centrée réduite
    output;
  end;
  keep y;
run;

```

On pourra simuler d'autres lois en utilisant les instructions : **ranbin** (binomiale), **rancau** (Cauchy), **ranexp** (exponentielle), **rangam** (gamma), **ranpoi** (Poisson), **ranuni** (uniforme [0,1]), *etc.* Les tirages sont des tirages pseudo-aléatoires. Entre deux tirages, il importe de changer la valeur initiale **s** (le *seed*) du générateur aléatoire de nombres. Autrement, les tirages sont tous identiques ! Par défaut, **s=0**.

```

proc gchart data=simul;
  title 'Histogramme des observations';
  vbar y; /* vbar3d y; */
run;
quit;

```

Exercice n°4 : exemple d'utilisation de la proc GPLOT

```

title; footnote 'TP2: Exercice4';
libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';

/*****
/* Sample Data Set: tabac
/* Sources: Direction des Services Fiscaux, ISEE
*****/

goptions reset=all rotate=landscape border ftext=hwdmx013 htext=0.25 cm;

title1 c=blue f=hwdmx020 h=1.4 j=center 'Evolution de la consommation de
cigarettes francaises et etrangeres entre 1989 et 2004';
title2 c=bl f=hwdmx011 h=1 'Source: ISEE';

symbol1 value=dot interpol=join line=1 color=blue width=2;
symbol2 v=circle i=join l=8 c=green w=2;
symbol3 v=square i=join l=15 c=red w=2;
symbol4 v=diamond i=splines l=22 c=yellow w=2;
symbol5 v=# i=needle l=29 c=black w=2;
symbol6 v=none i=step l=34 c=purple w=2;

axis1 length=17 cm
  label=(h=1 f=hwdmx021 'Annees')
  value=(f=hwdmx021 h=0.8)
  order=(1989 to 2004 by 1)
  minor=(h=0.2 width=1.0 n=4);
axis2 length=10.6 cm
  label=(h=1 f=hwdmx021 'Consommation en kg')
  value=(f=hwdmx021 h=0.8)
  order=(500 to 190500 by 10000)
  major=(h=0.5 width=1.5)
  minor=(h=0.2 width=1.0 n=4);

legend1 frame cframe=ligr position=(bottom center outside);

proc gplot data=mabib.tabac gout=graf1;
  plot cigfr*annee=1 ciget*annee=2 roule*annee=3
    / overlay haxis=axis1 vaxis=axis2 legend=legend1;
run;
quit;

```

De nombreux problèmes peuvent être décrits sous la forme d'un système à entrées et sorties : on peut plus ou moins agir sur les variables d'entrée X d'un système (variables descriptives) et on observe les variables de sortie Y (variables dépendantes). Le but de cette séance est de comprendre et de décrire simplement les relations entre Y et X lorsqu'un ajustement linéaire s'applique. Le modèle linéaire dans SAS est notamment traité par les procédures **ANOVA** (modèles d'analyse de la variance orthogonaux), **REG** (régression linéaire), **GLM** (modèles d'analyse de la variance non-orthogonaux, analyse de la covariance, modèles linéaires généraux), **GENMOD** (modèles linéaires généralisés), **PLS** (régression « Partial Least Squares ») et **MIXED** (modèles linéaires mixtes). En particulier, nous nous intéressons ici à la PROC REG. Des exemples sont donnés en exercice et nous abordons aussi la régression logistique avec la procédure **LOGISTIC** pour des variables Y qualitatives.

1. PROC REG

Modèle : pour n observations et p descripteurs, la PROC REG de SAS traite le modèle de régression :

$$Y_i = a + \sum_{k=1}^p b_k X_{i,k} + \varepsilon_i, \quad 1 \leq i \leq n ;$$

où **Y** désigne le vecteur dépendant ;

X = (**X**₁, **X**₂, ..., **X**_p) la matrice des p descripteurs ; $\varepsilon \sim (0, \sigma_\varepsilon^2 \mathbf{I}_n)$ le terme d'erreur ;

θ = (a, b₁, ..., b_p)' et σ_ε les paramètres du modèle que l'on souhaite estimer.

Le modèle s'écrit aussi sous la forme matricielle : $\mathbf{Y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\varepsilon}$ et par la méthode des moindres carrés ordinaires, nous obtenons un estimateur sans biais et de variance minimale pour **θ** : $\hat{\boldsymbol{\theta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$.

Référence bibliographique :

- J. Confais et M. Le Guen (2006) : Premiers pas en régression linéaire avec SAS, *Revue modulad*, n°35 < <http://www-rocq.inria.fr/axis/modulad/numero-35/Tutoriel-confais-35/confais-35.pdf> >.

La syntaxe de la PROC REG est donnée dans la séance n°2 ; pour prolonger notre étude, nous décrivons ici les sorties et nous donnons quelques options liées à la procédure :

□ SORTIES : 1. Table d'analyse de la variance

- **DF** : degrés de liberté associés au modèle et à l'erreur ;
- **Sum of Squares (SS)** ;
- **Mean Square (MS)** : $MS = SS / DF$;
- **F Value** : valeur de la statistique de Fisher du test du modèle où tous les coefficients sont nuls contre le modèle complet de régression ;
- **Prob>F** : probabilité pour que la valeur d'une statistique suivant la loi de Fisher dépasse la valeur « F Value » ;
- **Root MSE** : estimation de σ_ε, $\hat{\sigma} = \sqrt{MS \text{ Error}}$;
- **Dep Mean** : valeur de \bar{Y} ;
- **CV** : coefficient de variation, $CV = 100 / \bar{Y}$;
- **R-Square** : part de la variance expliquée par le modèle, $R^2 = SS \text{ Model} \div SS \text{ CorrectedTotal}$;
- **Adj R-Sq** : estimation sans biais de R²,

$$\text{Adj. } R^2 = 1 - [SS \text{ Error} \div (n - p - 1)] / [SS \text{ CorrectedTotal} \div (n - 1)]$$

$$= [(n - 1) / (n - p - 1)][1 - R^2].$$

2. Estimation des paramètres

- **INTERCEP** : paramètre Intercept = \hat{a} ;
- **Parameter Estimate** : estimateur du vecteur θ , $\hat{\theta} = (\hat{a}, \hat{b}_1, \dots, \hat{b}_p)$;
- **Standard Error** : estimation de l'écart type de l'erreur ;
- **T for H₀ : Parameter=0** : test de Student de nullité du coefficient ;
- **Prob>|T|** : probabilité que la valeur d'une statistique suivant la loi de Student dépasse T en valeur absolue ;

□ OPTIONS liées à l'instruction MODEL :

- **COVB** : matrice des variances et covariances des estimateurs de a, b_1, \dots, b_p ;
- **CP** : la valeur prise par le « Cp » de Mallows ;
- **P** : les valeurs prédites ;
- **R** : pour obtenir une analyse des résidus ;
- **DW** : pour calculer la statistique de test de Durbin-Watson ;
- **TOL, VIF** : colinéarité d'un descripteur vis à vis des autres ;
- **Influence** : analyse détaillée de l'influence de chaque observation sur les valeurs estimées et prédites.

Sélection des descripteurs. En particulier, la PROC REG offre des possibilités supplémentaires en ce qui concerne le choix automatique des descripteurs et la recherche de corrélations en termes d'erreurs.

- **SELECTION** = < méthode > ; pour sélectionner les descripteurs du modèle où *méthode* peut prendre les valeurs suivantes :
 - **stepwise** pour une sélection progressive (« pas à pas ») des descripteurs ;
 - **forward** pour une sélection ascendante ;
 - **backward** pour une élimination descendante...
 - **rsquare** calcule tous les $2^p - 1$ modèles possibles avec p descripteurs. On peut aussi se restreindre à la recherche du meilleur modèle à 1 puis 2 puis 3, ..., puis p descripteurs au sens du critère du Cp de Mallows à l'aide de l'option `selection=rsquare best=1 cp;`. À p fixé, on choisira le modèle qui donne le plus grand R^2 .

Étude des résidus. L'étude des résidus est une étape essentielle pour *valider* un modèle. En effet, les résidus sont des estimateurs des termes d'erreur qui doivent vérifier les hypothèses sous-jacentes au modèle : indépendance, moyenne nulle, variance constante (*critère d'homoscédasticité*) et éventuellement de distribution Gaussienne. Pour ce faire, la PROC UNIVARIATE calcule la moyenne ; et apporte des tests de normalité avec l'option *NORMAL* : Shapiro-Wilk, Kolmogorov-Smirnov, Cramér-von Mises, Anderson-Darling ou encore la « droite de Henri ». Une étude du graphe des résidus par rapport aux valeurs prédites de la variable dépendante ne doit laisser apparaître aucune tendance. L'option *DW* de la PROC REG fournit la valeur de la statistique de Durbin-Watson pour tester l'autocorrélation des résidus. Pour le critère d'homoscédasticité, l'option *SPEC* de la PROC REG propose un test du Chi². L'étude des résidus permet aussi de repérer d'éventuelles observations « aberrantes » ou des observations qui jouent un rôle important dans la détermination de la régression.

Sauvegarde des résultats. Il est possible de conserver dans une table SAS certains résultats de l'analyse tels que, par exemple, les résidus (**residual** ou **R**) et les valeurs prédites de la variable dépendante Y (**predicted** ou **P**). Pour cela, nous ajoutons après la définition du modèle la ligne :

```
OUTPUT OUT = < nom_tab > R = < nom_var1 > P = < nom_var2 >;
```

Exemple

```
DATA tension;
  INPUT age tension;
  CARDS;
  35 114
  45 124
  55 143
  65 158
  75 166
  ;
RUN;

PROC REG DATA=tension CORR SIMPLE OUTEST=estcoeff;
  MODEL tension = age / SPEC DW R CLI;
  PLOT tension*age / symbol = '.';
  OUTPUT OUT=sorties P=P R=R;
RUN;
QUIT;
```

La commande « OUTEST= » permet de créer une table SAS qui conserve les valeurs estimées des coefficients de la régression. L'option **CLI** donne un intervalle de confiance à 95% pour chacune des valeurs prédites de la variable dépendante. La seconde ligne de la PROC REG définit la modélisation souhaitée. Dans la troisième ligne, nous proposons aussi un graphique qui superpose le nuage de points et la droite d'ajustement.

Exercice Exécuter le programme ci-dessus et observer les sorties.

Remarques sur la stabilité des coefficients

cf. [Saporta G. (2006) : *Probabilités, analyse des données et statistique*, 2^e édition, éditions Technip, Paris]

L'écart-type des valeurs $\hat{a}, \hat{b}_1, \dots, \hat{b}_p$ donne déjà une bonne indication du caractère plus ou moins stable de l'estimation d'un coefficient. Si l'écart-type est du même ordre de grandeur que la valeur estimée, il est clair que celle-ci est mal déterminée !

La principale source d'instabilité pour estimer de $\hat{\theta}$ est la multicolinéarité des descripteurs (quand les descripteurs sont fortement corrélés entre eux). Dans ce cas, le déterminant de la matrice $\mathbf{X}'\mathbf{X}$ est proche de 0 et l'inverse de $\mathbf{X}'\mathbf{X}$ a des valeurs très élevées. Malgré une valeur élevée du R^2 , on pourra avoir des erreurs considérables dans les prédictions. Pour palier les problèmes liés à la multicolinéarité sans pour autant exclure des variables du modèle, il existe des alternatives au modèle de régression : *régression sur composantes principales*, *régression PLS* ou encore la « *ridge regression* » où la diagonale de $\mathbf{X}'\mathbf{X}$ est légèrement « perturbée » par une constante k positive.

2. SAS/Insight : exploration interactive de données

Le module SAS/Insight est un outil dynamique pour l'analyse exploratoire et graphique d'un jeu de données. Il se présente sous la forme d'une interface « presse-boutons » permettant notamment d'examiner les distributions univariées, de visualiser les données et de construire des modèles utilisant la régression, l'analyse de la variance et le modèle linéaire généralisé. Pour ce faire, l'utilisateur dispose d'une boîte de dialogue avec boutons et menus déroulants pour effectuer l'étude statistique sans avoir à écrire de code SAS dans la fenêtre *Program Editor*.

Appel de SAS/Insight

```
Explorer > solution > tools > analysis > Interactive Data Analysis
```

EXERCICES

Exercice n°1 : un exemple de régression linéaire simple

Soit un échantillon de $n=24$ appartements parisiens. Pour chaque appartement, on dispose de sa surface en mètres carrés et de son prix de vente en milliers d'euros. Ces données sont extraites de [Tenenhaus M. (2007) : *Statistique, Méthodes pour décrire, expliquer et prévoir*, Dunod, Paris] et rappelées ci-dessous :

S (m ²)	28	50	196	55	190	110	60	48	90	35	86	65
P (k€)	130	280	800	268	790	500	320	250	378	250	350	300
S (m ²)	32	52	40	70	28	30	105	52	80	60	20	100
P (k€)	155	245	200	325	85	78	375	200	270	295	85	495

- Visualiser la table SAS *apparts* stockée dans le répertoire *datasas* en utilisant la fenêtre Explorer.
- À l'aide de la PROC UNIVARIATE (avec l'option *plot*) ou du module SAS/Insight décrire les variables *Surface* et *Prix*. Pour chaque variable, donner la moyenne, la médiane, les quartiles, l'amplitude, l'écart-type et l'écart interquartile. Comment obtient-on le box-plot ? Identifier les valeurs extrêmes.
- À l'aide de la PROC CORR, justifier de la pertinence d'un ajustement linéaire.

On procède maintenant à la régression linéaire, en prenant comme variable descriptive X la surface et comme variable dépendante Y le prix.

- Lancer l'exécution du programme suivant :

```
proc reg data=lib1.apparts outest=coeffs;
  model prix=surface / SPEC DW R CLI;
  plot prix*surface / cframe=ligr;
  output out=sorties P=yhat R=residus;
run;
quit;
```

Questions : Pour chacune des quantités suivantes, rappeler sa signification et donner la valeur numérique à partir des sorties observées : α , β , σ^2 , SSM, SSE, SST, R^2 . Que valent les résidus ? On estime qu'une observation est atypique si son résidu studentisé n'est pas dans l'intervalle $[-2,2]$. Y a-t-il des observations atypiques ?

Exercice n°2 : simulation du modèle linéaire gaussien

On souhaite simuler un échantillon de taille $N=10$ du modèle linéaire gaussien :

$$Y_i = 2 + 1.5 X_i + \varepsilon_i, \quad 1 \leq i \leq N,$$

où les variables aléatoires ε_i sont i.i.d. de loi $N(0,1)$, et en prenant comme valeur $X_i = i$.

- Lancer l'exécution du programme suivant pour $N = 10$ valeurs simulées :

```
data simulation;
  do i=1 to 10;
    X=i; Y=2 + 1.5*X + rannor(-1); output;
  end;
run;
```

Quelles sont les vraies valeurs des paramètres de régression ? Grâce à la PROC REG de régression linéaire, donner les estimations des paramètres sur un jeu de données simulées. Recommencer avec cinq jeux de données distincts. Qu'observe-t-on ?

Reprendre l'exercice pour N = 50,100,250,500,750,1000 valeurs simulées. Qu'observe-t-on ?

Exercice n°3 : un exemple de régression logistique binaire

Dans l'exemple suivant, nous nous proposons d'étudier le lien entre la présence (ou absence) d'une maladie cardiaque coronarienne (CHD=Coronary Heart Disease) et l'âge au sein d'un échantillon de 100 individus. Les données sont extraites du livre de [Hosmer D. W. & Lemeshow S. (2000) : *Applied logistic regression*, 2nd édition, John Wiley & Sons, New York], et les variables d'étude sont les suivantes :

ID : identificateur de l'individu ;
AGRP : 'Age Group' ou classe d'âges ;
AGE : âge de l'individu en années ;
CHD : absence (0) ou présence (1) d'une maladie cardiaque coronarienne.

```
OPTIONS LS=64 PS=40;

libname mabib 'C:\Mes documents\Cours de SAS\TPSAS\datasas';

proc print data=mabib.heart noobs; run;

proc gplot data=mabib.heart;
  title 'GRAPHE DE CHD PAR RAPPORT A AGE';
  plot CHD*AGE;
  label CHD='CHD(y) '
        AGE='AGE(x) ' ;
  symbol1 value=dot;
run;
quit;

proc format;
  value fAGRP 1='20-29'
            2='30-34'
            3='35-39'
            4='40-44'
            5='45-49'
            6='50-54'
            7='55-59'
            8='60-69';
  value fCHD 0='Absent'
            1='Present';
run;

proc tabulate data=mabib.heart format=8.;
  class AGRP CHD;
  format AGRP fAGRP. CHD fCHD.;
  label AGRP='AgeGroup';
  table AGRP ALL,N*(ALL CHD) / box='Frequency Table of AgeGroup by CHD' RTS=12;
  keylabel ALL='Total'
           N='Frequency';
run;

proc logistic data=mabib.heart;
  model CHD=AGE;
  output out=sorties predprobs=individual;
run;
quit;

proc print data=sorties; run;
```

```
proc freq data=sorties;
  title 'table des bien et mal classés';
  table _from*_into_ / nopercnt norow nocol chisq;
run;
```

Exercice n°4 : sur les effets néfastes du phénomène de multicolinéarité en régression logistique ordinaire

Dans cet exemple, nous nous proposons d'observer l'influence de variables météorologiques sur la qualité annuelle des vins de Bordeaux (entre 1924 et 1957). Les variables d'étude de la table *vins* dans *datasas* sont les suivantes :

YEAR : année ;
TEMP : somme des températures moyennes journalières en °C ;
SUN : durée d'insolation en heures ;
HEAT : nombre de jours de grande chaleur ;
RAIN : hauteur des pluies en mm ;
QUALITY : qualité du vin 1 = bonne, 2 = moyenne, 3 = médiocre.

- Utiliser SAS/Insight pour une étude descriptive de chacune des variables météo *TEMP*, *SUN*, *HEAT*, *RAIN* (forme des distributions, statistiques simples, box-plots, *etc.*).
- À l'aide de la PROC CORR, justifier du choix des descripteurs retenus pour prédire la qualité. Noter les signes des coefficients de corrélation reliant la variable *QUALITY* à chacune des variables météo. Quels sont les risques probables d'un modèle de régression logistique à 5 descripteurs ?
- Observer les sorties de la PROC LOGISTIC pour le modèle à 5 descripteurs. Les estimations des coefficients pour les variables *Sun* et *Heat* vous paraissent-elles conformes aux résultats attendus ?

Séance n°4 Programmation en SAS (1) : macro-langage - O.D.S.

Le **macro-langage**, au même titre que le SQL ou l'IML (cf. séance n°5), améliore les possibilités du langage de base. Il permet de passer des paramètres entre les étapes DATA et PROC, et de systématiser l'enchaînement d'une séquence donnée d'instructions. L'**O.D.S. (Output Delivery System)** est notamment un outil très utile pour envoyer, formater et conserver des sorties SAS dans un fichier résultat (un fichier lue par Word, par exemple).

1. Les macros

Les macro-commandes de SAS permettent d'enchaîner l'exécution de plusieurs commandes SAS au moyen d'une seule instruction (qui sera appelée *macro*) utilisant, éventuellement, un ou plusieurs paramètres.

□ Les macro-variables

Les *macro-variables* sont les paramètres du macro-langage. La déclaration d'une macro-variable consiste à associer par la commande **%let** une chaîne de caractères à un identificateur.

Rôle	sert à stocker du texte (jusqu'à 65534 caractères en V9) ;
Nom	1 à 32 caractères (depuis V8) ;
Appel	par son nom précédé du symbole '&' (ex. &mvar) ;
Création	%LET < macro-variable = valeur > ;
Visualiser le contenu	%PUT &mvar ;

Exemple 1

```
%let valeur = 10;  
%put &valeur;
```

Le symbole « & » signifie « le paramètre qui renvoie à ». Dans cet exemple, « &valeur » est donc « le paramètre qui renvoie à *valeur* ».

Exemple 2

Considérons la table « tension » de la séance n°3 :

```
%let tabin = tension;  
%let varY = tension;  
%let varX = age;  
  
title "Scatter plot de &varY et &varX"; /* avec " au lieu de ' */  
proc gplot data=&tabin;  
  plot &varY*&varX;  
run;  
quit;
```

L'instruction **call symput** permet de créer une macro-variables en lui affectant les valeurs d'une variable d'une table SAS. Cette instruction s'utilise exclusivement dans une étape DATA.

Syntaxe

```
CALL SYMPUT('nom_macro-variable',valeur_macro_variable);
```

Exemple 3

```
data _NULL_;  
  set tension;  
  call symput('nobs',_N_);  
run;
```

Dans cet exemple, on affecte à la macro-variable *nobs*, les valeurs successives de la variable *_N_* qui correspond au rang de l'observation lue. Au terme de l'étape DATA, la macro-variable *nobs* contient le numéro de la dernière observation, c'est-à-dire le nombre d'observations de la table *tension*.

□ Les macro-fonctions

Les macro-fonctions sont des fonctions spécifiques au macro-langage pour manipuler des macro-variables.

Exemple 4 : %LENGTH et %SCAN

```
%let semaine = lundi - mardi - mercredi - jeudi - vendredi - samedi - dimanche;  
%let longueur = %length(&semaine);  
%let jour2 = %scan(&semaine,2,'-');
```

Dans cet exemple, la macro-variable *longueur* contient la longueur de la chaîne de caractères *semaine* soit 63. La macro-variable *jour2* correspond ensuite au 2^{ème} mot de la chaîne (soit *mardi*). Le troisième argument indique que le séparateur est un tiret.

Exemple 5 : %EVAL

Par définition, une macro-variable sert à stocker du texte. Par exemple, la valeur d'une macro-variable à laquelle on affecte « 1+2 » est « 1+2 » et non « 3 ». Toutefois, il est possible de forcer le compilateur macro à effectuer l'opération avec **%eval**. La macro-fonction `%eval(expression)` évalue des calculs (additions, soustractions, multiplications et divisions seulement) sur des entiers à partir d'expressions contenant des macro-variables. Si la valeur évaluée contient des décimales, la valeur est tronquée à la partie entière.

```
%let i = 22;  
%let j = &i/7; /* j contient 22/7 */  
%let k = %eval(&i/7); /* k contient 3 */  
  
%let pi = &k..%eval(100*&i/7-100*&k); /* pi contient 3.14 */
```

□ Les macro-programmes

Dans les séquences d'un programme SAS qu'on ne souhaite pas répéter plusieurs fois, nous distinguons celles dont la structure ne change pas mais dont seuls quelques éléments varient (ex. tables SAS, variables SAS, options de procédures...). Dans ce cas, le macro-langage nous permet de définir l'architecture d'une séquence qui sera paramétrée par des macro-variables et réutilisable à volonté. Cette architecture constitue alors le *code source* d'un *macro-programme*. Par exemple, on peut aussi souhaiter que certaines procédures ne soient exécutées que sous certaines conditions car autrement on en exécuterait d'autres... En ce sens, le macro-langage est encore utile pour définir des variables ou exécuter de manière conditionnelle des procédures. Ce langage offre donc de nouvelles possibilités très vastes pour enrichir le langage de base.

Une utilisation courante du macro-langage est la macro-expression qui s'écrit comme suit

```
%MACRO < nom >;  
  < texte >  
%MEND < nom >;
```

où *nom* est une chaîne de caractères qui ne dépasse pas trente-deux caractères.

Exemple 6

```
%macro triangle;  
  polygone à trois côtés  
%mend triangle;
```

Le caractère % est dans SAS le symbole des macros. Dans sa première ligne, une macro doit toujours comporter l'instruction **%macro** suivi du nom et se termine toujours par une ligne comportant l'instruction **%mend** éventuellement suivie du nom de la macro (« *triangle* » est donc facultatif ici mais conseillé...).

Création d'une macro-expression avec paramètres :

```
%MACRO < nom_prog > (mvar1,...,mvarp);  
  < contenu de la macro >           → code source du macro-programme  
%MEND < nom_prog >;
```

Appel de la macro :

```
%NOM(val1,...,valp);
```

Remarque À l'appel de la macro, les valeurs des *paramètres positionnels* (des macro-variables) sont annoncées après le nom du macro-programme entre parenthèses, séparées par des virgules et dans le même ordre que les paramètres.

Exemple 7

```
%macro printtab (table,varlist);  
  proc print data = &table;  
    var &varlist;  
  run;  
%mend printtab;
```

Considérons la table « *exemple* » de la séance n°1 :

```
%printtab(exemple,nom auto);
```

□ Bibliothèque de macro-commandes

Lorsque certaines macro-commandes sont régulièrement utilisées, il est pratique de pouvoir les conserver les unes à la suite des autres dans un fichier muni de l'extension « .sas ». En les ajoutant dans un fichier appelé « macros.sas », par exemple, il suffira ensuite dans n'importe quel programme SAS de faire l'appel :

```
%INCLUDE 'C:\Mes documents\Cours de SAS\TPSAS\macro.sas';
```

Remarques Les macro-variables et macro-fonctions peuvent être utilisées globalement tout au long d'un programme SAS ou encore localement à l'intérieur d'une macro.

```
%GLOBAL < nom_macro >;      ou      %LOCAL < nom_macro >;
```

À l'intérieur d'une macro, les instructions IF, THEN, ELSE, DO, TO, UNTIL, WHILE, END, *etc.* sont précédées du symbole « % ».

Enfin, notons que certaines macro-variables sont déjà définies dans SAS. Par exemple, les instructions suivantes

```
%put &sysdate;      %put &sysday;      %put &systemtime;
```

affichent respectivement la date, le jour et l'heure dans la fenêtre *Log*.

2. Contrôle des sorties SAS avec l'environnement O.D.S.

Par défaut, SAS présente les résultats d'un programme sous forme de fichier texte. Le système **O.D.S.** (*Output Delivery System*) est un outil permettant de sélectionner des sorties SAS et de les récupérer en choisissant des présentations appropriées à certains logiciels tels que des éditeurs de texte (Word, par exemple) ou des éditeurs de graphiques (Adobe Illustrator, par exemple), *etc.*

Le système O.D.S. offre entre autres les présentations suivantes :

LISTING	présentation par défaut avec des caractères de type mono-espace (Monaco, Courier) ;
PRINTER	fichiers de type postscript pour les imprimantes du même type ;
HTML	fichiers en langage html pour les pages Internet ;
RTF	fichiers utilisables dans un éditeur de texte (Word, par exemple) ;
PDF	fichiers en format pdf.

Voici un exemple de programme SAS dans lequel la même sortie est produite en trois formats de fichiers :

```
data note;
  input nom $ note @@;
  cards;
  Pierre 11 Paul 14
  Carole 13 Jacques 7
  ;
run;

ods rtf file = 'note_examen.rtf';
proc print data = note; run;
ods rtf close;

ods pdf file = 'note_examen.pdf';
proc print data = note; run;
ods pdf close;

ods printer file = 'note_examen.ps';
proc print data = note; run;
ods printer close;
```

Commentaires Dans ce programme, on commande d'abord la production d'un fichier « **.rtf** » (*Rich Text File*) en écrivant `ODS RTF;` suivi du nom du fichier à créer entre apostrophes. Ce fichier contient la sortie de la procédure PRINT sous la forme d'un tableau. On sort ensuite de l'environnement O.D.S. avec l'instruction `ODS RTF CLOSE;`.

Les sorties en formats « **.pdf** » (*Portable Document Format*) ou « **.ps** » (*postscript*) s'obtiennent de la même manière. Les fichiers pdf peuvent être insérés dans un éditeur de texte mais pas les fichiers postscript. Pour visualiser le fichier postscript nommé « `note_examen.ps` », il suffit de taper dans une fenêtre Terminal sous Unix l'instruction :

```
gv note_examen → « gv » pour ghostview.
```

On peut alors imprimer le fichier postscript avec une commande du menu *File*.

On peut aussi convertir un fichier postscript en un fichier pdf en tapant par exemple :

```
ps2pdf note_examen.ps note_examen.pdf
```

Enfin, notons aussi qu'il est possible d'interrompre l'affichage des sorties dans la fenêtre *Output* via l'instruction :

```
ODS LISTING CLOSE;
```

L'instruction contraire :

```
ODS LISTING;
```

permet ensuite de rétablir l'affichage. L'intérêt de cette instruction est évident lorsqu'une même procédure est exécutée un grand nombre de fois dans une boucle, notamment pour ne pas encombrer inutilement la fenêtre *Output*.

Référence bibliographique :

[Haworth L. E. (2001) : *Output Delivery System, the basics*, SAS Institute Inc., Cary, NC]

EXERCICES

Exercice n°1

Écrire une macro-commande *imprime* paramétrée par *tabin*, *varlist* et *n* qui affiche les *n* premières observations pour une liste de variables choisies d'une table SAS donnée. Supprimer l'affichage des rangs des observations et ajouter un titre qui renseigne le jour, la date et l'heure de l'impression.

Exercice n°2

- Écrire une macro-commande *gaussienne* à quatre paramètres qui génère une table en sortie *tabout* qui contient *n* valeurs simulées suivant une loi normale de moyenne *mu* et d'écart-type *sigma*.
- Exécuter cette macro en faisant varier *n*, *mu*, et *sigma* ; et observer la forme de la distribution associée en utilisant au choix la procédure UNIVARIATE ou SAS/Insight. Ajuster la fonction de densité théorique sur les histogrammes.

Remarque À chacune des valeurs entières du *seed*, choisie entre 1 et $2^{31} - 1$ (= 2,147,483,647), correspond un tirage pseudo-aléatoire déterminé et reproductible. Si $s \leq 0$, SAS utilise l'horloge pour initialiser le tirage et celui-ci n'est alors plus reproductible à l'identique.

Exercice n°3

- Écrire une macro-commande *correlation* qui produit la matrice des coefficients de corrélation de Pearson pour certaines variables d'une table SAS. Trois paramètres : *tabin* la table en entrée, *tabout* la table en sortie qui contient la matrice (utiliser l'option : OUP = < nom_tab >) et *varlist* pour les variables d'étude. Dire comment couper les sorties dans la fenêtre *Output*.
- Écrire une macro-commande *reglin* qui traite de manière automatique le modèle de régression linéaire défini dans le TP3. Cette macro reçoit quatre paramètres : *tabin* la table en entrée, *y_dep* la variable dépendante Y, *x_exp* la (ou les) variable(s) descriptive(s) X et *tabout* la table en sortie qui contient les valeurs estimées de x_1, \dots, x_p , Int et σ . Couper les sorties dans la fenêtre *Output*.
- Exécuter ces deux macros sur la table *htwt* où *height* est la variable dépendante et *weight* la variable descriptive.

Exercice n°4 : un exemple de programme en SAS : le Balanced Bootstrap*

La méthode du Balanced Bootstrap consiste à réutiliser exactement le même nombre de fois chacune des observations lors du rééchantillonnage. Simple à programmer, cette méthode possède de bonnes propriétés statistiques. Les B échantillons « balanced bootstrap » peuvent être obtenus de la manière suivante :

1. Dupliquer B fois le n-échantillon initial $x = (x_1, x_2, \dots, x_i, \dots, x_n)$ pour obtenir un nouvel nB-échantillon ;
2. Permutation aléatoire des nB lignes de cet échantillon ;
3. Découper le nB-échantillon obtenu en B parties égales.

On obtient finalement B échantillons bootstrap $x^* = (x_1^*, x_2^*, \dots, x_k^*, \dots, x_B^*)$ construits à partir d'un « rééchantillonnage équilibré » des observations (x_i) .

Proposer une macro *bootstrap* paramétrée par *tabin*, la table SAS à rééchantillonner et *nboot*, le nombre d'échantillons souhaité afin de générer *nboot* échantillons « balanced bootstrap » *boot_1, \dots, boot_nboot* à partir de *tabin*. On pensera à l'intérieur du programme à stocker le nombre d'observations de *tabin* dans une macro-variable *nobs*.

*[Davison A. C., Hinkley D. V., Schechtman E. (1986) : Efficient bootstrap simulation, *Biometrika*, **73**, 3, p.555-566]

Séance n°5 Programmation (2) : calcul matriciel avec SAS / IML

SAS/IML est un langage de programmation interactif qui utilise le calcul matriciel. L'élément fondamental d'IML est la matrice définie par un tableau à deux dimensions de valeurs numériques. Dans les expressions IML, les opérateurs s'appliquent directement aux matrices. Ainsi, l'expression $A+B$ additionne les éléments de deux matrices A et B ; et plusieurs opérateurs et fonctions sont déjà définis pour nous faciliter la tâche. Par exemple, une seule ligne est suffisante pour sommer tous les éléments positifs d'une matrice X avec IML quand d'autres langages de programmation nécessitent généralement l'utilisation de boucles. Les notations d'IML sont donc plus compactes. Autre exemple : pour estimer les paramètres d'un modèle de régression linéaire par la méthode moindres carrés ordinaires, il suffira d'écrire $B=INV(X' * X) * X' * Y$;. On peut de plus traiter des données à l'intérieur de l'environnement IML : lire une table, créer de nouvelles variables (chaque variable étant représentée par un vecteur), créer de nouvelles tables, etc. Il est à noter toutefois qu'IML ne reconnaît pas les valeurs manquantes.

1. Création d'une matrice

L'accès à IML se fait dans le programme avec la commande : `PROC IML;`. Après « submit », la fenêtre *Log* affiche alors le message *IML ready*. On peut ensuite créer des matrices en affectant des valeurs à des variables comme suit :

<code>PROC IML;</code>	
<code>RESET PRINT;</code>	→ affichage automatique de toutes les matrices créées, sinon : <code>RESET NOPRINT;</code>
<code>X=1;</code>	→ matrice 1 x 1 (un scalaire) contenant la valeur 1
<code>X={ 1 2 3 };</code>	→ matrice 1 x 3 (un vecteur ligne)
<code>X={ 1, 2, 3 };</code>	→ matrice 3 x 1 (un vecteur colonne)
<code>X={ 1 2 3, 4 5 6, 7 8 9 };</code>	→ matrice carrée 3 x 3
<code>SHOW NAMES;</code>	→ liste de toutes les matrices définies avec leurs attributs
<code>QUIT;</code>	→ pour quitter l'environnement IML

Commentaires Ici, la matrice X change quatre fois de dimensions et de valeurs. Les valeurs finalement contenues dans X sont alors les dernières affectées. Notons que toutes les matrices créées restent en mémoire tant qu'on ne quitte pas la procédure IML (avec la commande **QUIT**). La procédure IML doit être considérée comme n'importe quelle autre procédure SAS. Elle débute par l'instruction **PROC IML** et se termine par **QUIT** (au lieu de **RUN**). Néanmoins, les instructions utilisées à l'intérieur de cette procédure réfère à la programmation matricielle ce qui la singularise des autres procédures SAS.

□ Principaux opérateurs et commande PRINT

On peut créer une nouvelle matrice ou redéfinir une matrice déjà existante à l'aide des opérateurs matriciels de base qui sont « + », « - », « * », « / », etc.

<code>Y=X+1;</code>	→ crée une matrice Y à partir de X en ajoutant 1 à tous ses éléments
<code>X=X+1;</code>	→ réactualise X sans créer une nouvelle matrice

L'opérateur « || » permet de concaténer horizontalement les valeurs de deux matrices pour former une matrice plus grande.

<code>X={ 1 2, 3 4 };</code>	<code>Y=X { 0, 0 };</code>
------------------------------	-------------------------------

Pour concaténer verticalement deux matrices, on utilise le symbole « // ».

L'affichage des résultats matriciels demandés se fait par le biais de la commande **PRINT**. On peut afficher simultanément (côte à côte) plusieurs matrices en séparant les noms des matrices par des espaces. Pour afficher les matrices les unes en dessous des autres, on sépare les noms des matrices par des virgules sur la commande PRINT, la virgule indiquant un saut de ligne. Avec le séparateur « / », chaque matrice est affichée sur une page différente.

```

Exemple 1   NOM={Pierre, Paul, Carole, Nathalie};
              REVENU={35000, 24000, 27000, 31000};
              DEPENSES={32500, 23000, 26500, 28000};

              PRINT NOM REVENU DEPENSES;
              PRINT NOM, REVENU, DEPENSES;
              PRINT NOM / REVENU / DEPENSES;
  
```

Pour modifier un élément particulier d'une matrice, on utilise les crochets.

```

REVENU [ 2 ] = REVENU [ 2 ] + 3000;
  
```

Cet exemple ne modifie que le deuxième élément du vecteur *REVENU*. Si on sait qu'on a fait une erreur en entrant le revenu de Paul, mais qu'on ne connaît pas sa position dans le vecteur, on utilise la fonction **LOC** :

```

POS=LOC (NOM={ Paul });           →  retourne le rang de 'Paul' dans le vecteur NOM
REVENU [ POS ] = REVENU [ POS ] + 3000;
  
```

Supposons maintenant que le premier individu soit vendeur dans un magasin et qu'il soit responsable des ventes de cinq articles particuliers. On possède comme informations, pour chaque article, la quantité vendue au cours du mois et son prix de vente.

```

Exemple 2   ARTICLE={'art1', 'art2', 'art3', 'art4', 'art5'};
              QUANTITE={4, 0, 2, 3, 1};
              PRIX={25.12, 500.00, 110.50, 44.99, 85.00};
              PRINT ARTICLE QUANTITE PRIX;
  
```

L'opérateur « # » permet d'effectuer la multiplication de deux matrices élément par élément.

```

VENTES=QUANTITE#PRIX;
  
```

Pour multiplier une matrice par un scalaire, on utilise l'opérateur « * » ou « # ». Les opérateurs « ** » et « ## » désignent respectivement la fonction puissance matricielle et la fonction puissance élément par élément de la matrice :

```

PROC IML;
  X={1 2,3 4};
  Y=X**2;           affiche
  Z=X##2;          3   4   15   22   9   16
  PRINT X Y Z;
QUIT;
  
```

L'opérateur « @ » est utilisé pour le produit de Kronecker :

```

PROC IML;
  X={1 1,1 1};
  Y={1 2,3 4};     affiche
  Z=X@Y;           1   1   3   4
  PRINT X Y Z;     1   2   1   2
  QUIT;            3   4   3   4
  
```

Pour sommer les éléments de la matrice *VENTES*, on peut utiliser la fonction « **SUM** » ou autrement l'opérateur « + » :

```
TOTAL=SUM (VENTES) ;    ou    TOTAL=VENTES [+] ;
```

La commande **PRINT** s'emploie aussi avec les options **ROWNAME=** (ou **R=**) et/ou **COLNAME=** (ou **C=**). Les matrices spécifiées après le signe d'égalité sont de type caractère et permettent d'associer un nom à chaque ligne et/ou à chaque colonne de la matrice.

```
MATRICE=QUANTITE || PRIX || VENTES ;  
VARIABLE={QUANTITE, PRIX, VENTES} ;  
PRINT MATRICE [R=ARTICLE C=VARIABLE] ;
```

Ici, la commande **PRINT** affiche *MATRICE* avec les éléments de *ARTICLE* en observations et ceux de *VARIABLE* en variables.

L'opérateur « >:< » (resp. « <:> ») permet de déterminer la position de l'élément minimal (resp. maximal) d'une ligne ou d'une colonne.

```
POS=MATRICE [>:<, 1] ;    puis    PRINT ARTICLE [POS] ;
```

Cet exemple indique l'article qui s'est le moins vendu au cours du mois en termes de quantité. SAS/IML retourne pour la première colonne de *MATRICE* la position de la plus petite valeur.

L'opérateur pour la transposition de matrice est la fonction « **T** » ou « ` ».

```
TMATRICE=T (MATRICE) ;    ou    TMATRICE=MATRICE ` ;
```

La formulation suivante permet de créer une matrice qui contient les éléments de la troisième colonne de la matrice *MATRICE*.

```
VENTES=MATRICE [, 3] ;
```

❑ Créer une matrice à partir d'une table SAS

On peut utiliser la commande **READ** pour lire les *p* variables d'une table SAS et emmagasiner ces valeurs dans des vecteurs ou matrices IML.

```
PROC IML ;  
  USE < nom_tabSAS > ;  
  READ ALL ;  
  PRINT < nom_var1 > ... < nom_varp > ;  
QUIT ;
```

La commande **READ** utilisée avec l'option **ALL** permet de lire toutes les observations de la table *nom_tabSAS*. Les valeurs de chaque variable de la table sont alors reportées dans des vecteurs colonnes qui portent le même nom que les variables de la table.

```
Exemple 1    PROC IML ;  
              USE BILAN ;  
              READ POINT {1 3} VAR {QUANTITE VENTES} INTO M ;  
              PRINT M ;  
              QUIT ;
```

Dans cet exemple, la commande **READ** est utilisée avec **POINT** et **VAR** pour restreindre la lecture de la table *BILAN* aux observations n°1 et n°3 et aux variables *QUANTITE* et *VENTES*. Les valeurs sont alors reportées dans une matrice *M*.

```

Exemple 2   DATA CLASSES;
                INPUT NOM LANGUE COURS @@;
                CARDS;
                Marie E 1 Pierre A 2 Carole F 1
                Paul E 1 Jacques F 3 Julie A 2
                ;
                RUN;

                PROC IML;
                P=1:3;
                READ POINT P;
                PRINT NOM LANGUE COURS;
                QUIT;
    
```

Dans cet exemple, on affiche les valeurs de *NOM*, *LANGUE* et *COURS* relatives aux rangs des observations donnés par le vecteur *P* (*i.e.* les trois premières observations de la table).

□ Créer une table SAS à partir des valeurs de matrices IML

```

PROC IML;
USE CLASSES;
READ ALL;
CREATE CLASSES2 VAR {NOM COURS};
APPEND;
CLOSE CLASSES CLASSES2;
QUIT;
    
```

On crée donc une table *CLASSES2* qui contient les valeurs des matrices *NOM* et *COURS*, matrices formées à partir des observations lues dans la table *CLASSES* pour les variables du même nom. La présence de l'énoncé **APPEND** est indispensable, autrement le fichier *CLASSES2* serait vide. **CLOSE** ferme la table *CLASSES* ouverte pour lecture (avec **USE**) et ferme également la nouvelle table qu'on vient de créer.

□ Modifier un fichier SAS

Si on souhaite modifier une valeur de la table *CLASSES*, on doit l'ouvrir avec la commande **EDIT**, plutôt qu'avec **USE**, afin de pouvoir lire et écrire dans le fichier (et non être en mode « lecture seule »).

```

PROC IML;
EDIT CLASSES;
COURS=2;
REPLACE POINT 4 VAR{COURS};
SHOW DATASETS;
CLOSE CLASSES;
QUIT;
    
```

Ici, on remplace (commande **REPLACE**) dans l'observation n°4 la valeur de la variable *COURS* par 2. La commande **SHOW DATASETS** (optionnelle) indique que *CLASSES* est la table de lecture courante, mais également la table de sortie courante.

Dans l'exemple suivant, IML lit l'observation n° 2 (READ) du fichier *CLASSES*, modifie la valeur des variables *NOM* et *COURS* et ajoute l'observation modifiée (APPEND) à la fin de la table *CLASSES* :

2. Principales fonctions

Voici une liste non exhaustive des fonctions qui peuvent être utilisées dans IML :

ABS(matrice) retourne une matrice dont chaque valeur vaut la valeur absolue de l'élément correspondant dans la matrice argument. Même principe avec les fonctions **COS** (cosinus), **SIN** (sinus), **TAN** (tangente), **EXP** (exponentielle), **LOG** (logarithme), **INT** (partie entière), **SQRT** (racine carrée), *etc.* ;

ALL(matrice) retourne la valeur 1, si tous les éléments de la matrice sont non nuls, et 0 autrement ;

ANY(matrice) retourne la valeur 1, si au moins un des éléments de la matrice argument est non nul. Si tous les éléments sont nuls, la valeur retournée est 0. Les valeurs manquantes sont traitées comme des zéros ;

DET(matrice) calcule le déterminant d'une matrice carrée ;

DIAG(matrice_carrée) ou DIAG(vecteur) crée une matrice diagonale formée des éléments diagonaux de la matrice argument si celle-ci est carrée. Si l'argument est un vecteur, la matrice diagonale résultante est formée des éléments du vecteur ;

DO(début,fin,pas) produit une série arithmétique, *i.e.* un vecteur dont le premier élément est « début », le dernier élément est « fin » et les autres éléments les valeurs intermédiaires pour le pas donné ;

EIGVAL(matrice_symétrique) crée un vecteur colonne contenant les valeurs propres de la matrice symétrique argument (les valeurs propres sont en ordre décroissant) ;

EIGVEC(matrice_symétrique) crée une matrice qui contient les vecteurs propres orthonormés (chaque colonne représente un vecteur propre) de la matrice symétrique ;

HOMOGEN(matrice) résout un système d'équations linéaires homogène ;

$X = \text{HOMOGEN}(A) ; \quad \rightarrow \text{résout le système } A * X = 0 \text{ où } A \text{ est une matrice de dimensions } n \times p$

I(n) produit une matrice identité de dimensions $n \times n$;

INV(matrice) calcule l'inverse d'une matrice carrée non singulière. **GINV** pour l'inverse généralisée ;

J(n,p,valeur) crée une matrice de dimensions $n \times p$ de valeurs identiques ;

MAX(matrice) ou MAX(matrice1, matrice2, ..., matriceN) produit une seule valeur numérique (ou une seule chaîne de caractères) qui est l'élément le plus grand de tous les arguments (qui est la valeur de la chaîne de caractères la plus élevée). Idem avec **MIN** ;

MOD(valeur, diviseur) retourne le reste de la division des éléments du premier argument par les éléments du second ;

NCOL(matrice) retourne le nombre de colonnes de la matrice argument. **NROW** pour le nombre de lignes ;

ROOT(matrice) donne la décomposition de Cholesky d'une matrice symétrique et définie non négative ;

$X = \text{ROOT}(Y) ; \quad \rightarrow X \text{ est une matrice triangulaire supérieure telle que } Y = X * X$

SOLVE(matrice1, matrice2) résout un système d'équations linéaires ;

$X = \text{SOLVE}(A, B) ; \quad \rightarrow \text{résout le système d'équations : } AX = B, \text{ où } A \text{ est carrée et non singulière, commande équivalente à } X = \text{INV}(A) * B$
--

SSQ(matrice1, matrice2, ..., matriceN) retourne une valeur simple, soit la somme des carrés de tous les éléments de tous les arguments de la fonction. L'argument peut compter jusqu'à 15 matrices ;

TRACE(matrice) calcule la somme des éléments diagonaux de la matrice ;

VECDIAG(matrice) crée un vecteur colonne qui contient les éléments diagonaux d'une matrice carrée.

XSECT(matrice1, matrice2, ..., matriceN) retourne dans un vecteur ligne l'ensemble ordonné des éléments présents dans tous les arguments de la fonction.

EXERCICES

Exercice

Nous reprenons le modèle linéaire défini à la séance n°3 : $\mathbf{Y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\varepsilon}$. Sans utiliser les outils liés à la régression linéaire disponibles dans SAS, écrire avec SAS/IML une macro-commande *estmco* qui prend en paramètres *tabin* la table en entrée, *y_dep* la variable dépendante et *x_exp* la (ou les) variable(s) descriptive(s) et qui génère trois tables en sortie

- *est* contenant les estimateurs des moindres carrés ordinaires ;
- *y_pred* pour le vecteur des valeurs prédites de Y ;
- *res* pour le vecteur des résidus.

Vérifier ensuite les résultats obtenus avec ceux donnés par la PROC REG.

Quelques références en SAS

- J. Confais : Polycopié de SAS de l'ISUP ;
- J.-M. Azaïs, P. Besse, H. Cardot, V. Couallier, A. Croquette : *SAS sous UNIX, Logiciel hermétique pour système ouvert*, Laboratoire de Statistique et Probabilités, Université Paul Sabatier, Toulouse III. Ce cours de SAS est disponible en ligne à l'adresse suivante :

< <http://www.lsp.ups-tlse.fr/Besse/pub/saspdf.pdf> > ;

- O. Wolber : Cours de SAS (complet) et disponible en ligne sur le site du CNAM :

< <http://maths.cnam.fr/spip.php?article154> > ;

Également, *Introduction à SAS* : < <http://cedric.cnam.fr/~saporta/Cours%20SAS%20v2.pdf> > ;

- Manuel en ligne :

< http://www.stat.ucl.ac.be/cours/stat2430/documents/manuels_logiciels/Manuel_SAS_francais.pdf > ;

- O. Decourt : ressources illustrés par des exemples sur l'espace téléchargement du site :

< <http://www.od-datamining.com/> > ; (à voir pour les procédures GCHART et GPLOT entre autres)

- Inscription (gratuite) au *Club SAS Stat* des utilisateurs de SAS :

< http://www.sas.com/offices/europe/france/clubs_spe/clubstat.html > ;

- Haworth L. E. (2001) : *Output Delivery System, the basics*, SAS Institute Inc., Cary, NC ;
- Allison P. D. (1999) : *Logistic regression using the SAS system : Theory and application*, SAS Institute Inc., Cary, NC.

Projet SAS

PARTIE I : Cas d'étude de la procédure LOGISTIC

PARTIE II : Programmation d'une procédure de validation croisée

*Ce projet sera à rendre en binôme ou individuellement. Une attention particulière sera portée à la présentation et à la rédaction du projet. Le code SAS et les différentes sorties devront impérativement apparaître avec des commentaires. Toutes les sorties seront récupérées avec l'ODS. Le rapport sera à remettre en version papier (20 pages maximum).
Remise le vendredi 13 mai 2011 au plus tard.*

Dans la partie I de ce projet, nous nous intéressons à un cas d'usage de la procédure LOGISTIC de SAS sur un tableau de données fourni avec le projet. Dans la partie II, nous mettrons en œuvre une procédure de validation croisée qui sera utilisée pour évaluer la qualité d'un modèle à prédire de nouvelles données. Dans cette partie, nous utiliserons notamment le langage macro et éventuellement la procédure IML qui sont introduits dans les séances 4 et 5 du polycopié de SAS.

¹lejeufra@yahoo.fr

PARTIE I : Cas d'étude de la procédure LOGISTIC

Description de la procédure :

1. Expliquer en quelques mots le principe de la procédure ;
2. Donner la syntaxe générale ;
3. Donner deux options et dire ce qu'elles apportent en plus aux sorties ou à l'analyse.

Application :

Importer le tableau de données « bankruptcy ». Expliquer la procédure d'importation de ces données dans SAS.

1. Procéder à une analyse univariée des variables de la table. Une ACP appliquée aux variables "CF_TD", "NI_TA", "CA_CL", "CA_NS" permet-elle de distinguer les deux groupes d'entreprises ?
2. En utilisant la proc ANOVA, appliquer le test de comparaison des moyennes de DUNNET à chacune des variables "CF_TD", "NI_TA", "CA_CL", "CA_NS" pour observer une éventuelle différence entre les niveaux "1" et "2" de la variable "POP". Que peut-on conclure de cette étape ?
3. Donner un modèle LOGISTIC qui relie la variable la variable "POP" à toutes les variables "ratio". Les coefficients du modèle sont-ils tous significatifs ?
4. À l'aide de l'instruction SELECTION avec BACKWARD, FORWARD ou STEPWISE, peut-on proposer un modèle avec des coefficients significatifs pour une sélection bien choisie des 4 variables "ratio" ? Expliquer brièvement le critère servant à sélectionner les variables. Décrire les sorties de la proc LOGISTIC. Donner la table des "observés/prédits" (ou matrice de confusion) du modèle. Conclure.

PARTIE II : Programmation d'une procédure de validation croisée

Notation : n désigne le nombre d'observations.

La *validation croisée* permet d'estimer avec un biais réduit le taux de concordance entre les réponses observées et les réponses prédites par un modèle.

Le principe consiste à donner une prédiction et à évaluer l'erreur commise sur une partie de l'échantillon qui n'a pas participé à l'estimation des paramètres du modèle.

Ce calcul est répété avec plusieurs échantillons de validation afin d'améliorer la précision.

Dans cette partie, nous proposons de comparer trois manières de constituer des échantillons de validation pour les appliquer à l'estimation d'un taux de mauvais classement en régression logistique :

1. La validation croisée originale (*PRESS* de Allen) considérant n échantillons d'apprentissage de taille $n - 1$ obtenus en éliminant tour à tour chaque observation (*leave-one-out*) qui constitue l'échantillon test.
2. Le découpage aléatoire de l'échantillon en k parties (par défaut, $k = 7$) approximativement de même taille. L'échantillon d'apprentissage est alors constitué de $k - 1$ parties et la partie restante sert d'échantillon test. Après k itérations, chaque groupe a joué exactement une fois le rôle d'échantillon test.
3. La dernière manière reprend la deuxième avec une partition en k parties, mais ici nous faisons en sorte que chaque partie conserve les proportions des niveaux de la variable de groupe de la table complète.

Pour chaque méthode, nous estimons respectivement n et k fois le modèle sur la partie apprentissage de l'échantillon, nous prédisons la partie test et nous estimons le taux des mal classés à partir de la matrice de confusion. Le taux global est obtenu en moyennant les n ou k taux de mauvais classements.

En pratique, le choix de la méthode de validation croisée dépend principalement de la taille de l'échantillon : la première si n est petit et la deuxième si n est "assez" grand.

Questions :

- Écrire un macro programme *press_logistic* qui réalise la méthode PRESS décrite par 1).
- Écrire un macro programme *kfoldcv_logistic* qui réalise les méthodes 2) ou 3). Le choix de la méthode et le nombre de parties seront donnés en paramètres d'entrée du programme.
Choisir et expliquer les objets qui seront générés en sortie des deux macro programmes.
- Appliquer les 3 méthodes à la table *bankruptcy* de la partie I. Commenter les résultats.