Introduction à SAS/IML

Programmation & Logiciels Statistiques

Cours 6

Salim Lardjane - Université de Bretagne-Sud

Aperçu général

Aperçu général

- L'acronyme IML signifie *Interactive Matrix Language*
- Le langage SAS IML permet de lire des données dans des vecteurs et des matrices et de manipuler ces objets en faisant appel à des fonctions de haut niveau
- Il permet de formuler et de résoudre des problèmes mathématiques et statistiques de façon originale

Aperçu général

- Il permet d'analyser et de visualiser des données et d'implémenter des algorithmes spécifiques qui ne sont pas disponibles dans les procédures SAS
- Le langage SAS/IML contient plus de 300 fonctions et routines
- Il peut également appeler les centaines de fonctions de SAS/Base

Aperçu général

- Il est possible d'écrire des programmes SAS/IML de deux façons : en utilisant la PROC IML ou l'application SAS/IML Studio
- Les matrices constituent un élément fondamental du langage SAS/IML
- Une matrice est un tableau rectangulaires de nombres ou de chaînes de caractères
- Dans le cadre de la PROC IML, *toutes les variables sont des matrices*

Aperçu général

- Les matrices peuvent être utilisées pour stocker divers types d'information
- Par exemple, chaque ligne peut correspondre à un individu et chaque colonne à une variable statistique
- Dans le cas d'une matrice de variance-covariance, la composante *ij* représente la covariance entre la *i*-ème et la *j*-ème variable statistique

Exemple

- Les instructions de PROC IML suivantes lisent certaines données numériques d'un tableau de données dans une matrice *x*
- Le programme calcule ensuite des estimations robustes de position et d'échelle pour chaque variable
- Celles-ci sont ensuite utilisées pour centrer et réduire les données
- Le code est le suivant :

Exemple

```
proc iml;
      use Sashelp.Class;
      read all var _NUM_ into
            x[colname=VarNames];
      close Sashelp.Class;

      c = median(x);
      s = mad(x);
      stdX = (x-c)/s;

      print c[colname=VarNames];
      print s[colname=VarNames];
quit;
```

Exemple

- Cet exemple met en évidence les points suivants :
  - Il est possible de lire les données d'une table SAS dans une matrice
  - Il est possible de passer des matrices en arguments de fonctions
  - De nombreuses fonctions agissent par défaut sur les *colonnes* des matrices

Exemple

- Il est possible d'effectuer des opérations arithmétiques sur les matrices et vecteurs en utilisant une syntaxe naturelle
- Il est possible d'analyser des données et de calculer des statistiques sans écrire de boucles.

• De façon générale, le langage SAS/IML permet d'écrire des programmes compacts dans une syntaxe simple à utiliser.

Comparaison SAS/IML - étape Data

- Les possibilités de l'étape DATA sont suffisantes pour traiter les problèmes les plus fréquents
- Toutefois, on a parfois besoin d'implémenter un algorithme spécifique ou très récent
- On peut également avoir besoin d'utiliser des matrices pour combiner les résultats de diverses procédures
- Dans ce type de situations, on a recours à SAS/IML

Comparaison SAS/IML - étape Data

- La syntaxe de SAS/IML a beaucoup de points communs avec celle de l'étape DATA
- Ni l'un ni l'autre ne sont sensibles à la capitalisation
- Les noms de variables peuvent être composés de 32 caractères au plus
- Les instructions doivent se terminer par un point-virgule

Comparaison SAS/IML - étape Data

- La syntaxe des instructions IF-THEN/ELSE et du DO itératif sont les mêmes dans les deux langages.
- Les deux langages utilisent les mêmes symboles pour tester l'égalité (=), la différence (^=) et pour comparer des quantités (par exemple, <=).
- Le langage SAS/IML permet d'utiliser les même fonctions mathématiques que l'étape DATA : log, sqrt, abs, sin, cos, ceil, floor, etc.

Comparaison SAS/IML - étape Data

- Il y a deux différences principales entre l'étape DATA et un programme SAS/IML
- L'étape DATA boucle sur les observations alors que ce n'est pas le cas d'un programme SAS/IML
- L'unité fondamentale de l'étape DATA est l'observation alors que pour SAS/IML, c'est une matrice

Comparaison SAS/IML - étape Data

- De façon générale, SAS/IML est conçu pour les calculs statistiques alors que l'étape DATA est plutôt adaptée à la lecture, la fusion et la transformation des données
- Les calculs statistiques peuvent souvent être effectués de façon plus concise et plus simple avec SAS/IML

Le logiciel SAS/IML

- Le logiciel SAS/IML est formé de deux composantes : la PROC IML et l'application SAS/IML Studio
- La PROC IML fait partie du système SAS depuis la version 6
- SAS/IML Studio est plus récent
- C'est un environnement de programmation permettant de développer, d'exécuter et de débugger des programmes SAS/IML

Le logiciel SAS/IML

- SAS/IML Studio autorise à appeler des procédures SAS, des étapes DATA et des macro-fonctions à partir de programmes SAS/IML
- Il permet également d'obtenir des graphiques dynamiques

La PROC IML

- La PROC IML implémente le langage SAS/IML
- C'est une procédure interactive en ce sens que chaque instruction est exécutée au moment où elle est soumise
- On peut soumettre plusieurs instructions, examiner le résultat, soumettre de nouvelles instructions, etc.
- Lorsqu'on définit une matrice, elle existe jusqu'à ce qu'on quitte la procédure, qu'on vide la mémoire ou qu'on la redéfinisse

La PROC IML

- C’est différent de nombreuses autres procédures SAS, qui n’exécutent pas d’instructions jusqu’à ce qu’elles rencontrent une instruction RUN
- L’instruction RUN du langage IML est utilisée pour exécuter des modules ou des routines
- Afin de quitter la PROC IML, on utilise l’instruction QUIT

La PROC IML

- Il est nécessaire de disposer d'une licence SAS pour pouvoir utiliser la PROC IML
- Un programme IML commence par l'instruction PROC IML
- En voici un exemple :

La PROC IML

```
proc iml;
      Celsius = {-40, 0, 20, 37, 100};
      Fahrenheit = 9/5 * Celsius + 32;
      print Celsius Fahrenheit;
quit;
```

- L'instruction PRINT affiche la valeur des variables sur la destination de sortie courante
- Cette utilisation de la PROC IML revient à l'utiliser comme une calculatrice

SAS/IML Studio

- SAS/IML Studio est un environnement permettant de développer, exécuter et débugger des programmes SAS/IML.
- Il est disponible sans frais supplémentaires lorsqu'on dispose des licences SAS/IML et SAS/STAT.
- Le langage de programmation de SAS/IML Studio s'appelle IMLPlus

SAS/IML Studio

- IMLPlus est une extension du langage SAS/IML qui offre des fonctionnalités supplémentaires
- Il combine la souplesse d'utilisation de SAS/IML avec la possibilité d'appeler des procédures SAS et de créer et modifier des graphiques liés dynamiquement
- Ainsi, IMLPlus offre toutes les fonctionnalités de la PROC IML mais peut faire beaucoup plus

SAS/IML Studio

- SAS/IML Studio répond aux besoins des programmeurs SAS en fournissant un environnement permettant de développer des algorithmes, d'explorer les données, d'étudier les relations entre les variables, de comparer des modèles statistiques et de détecter et comprendre les valeurs aberrantes.

SAS/IML Studio

- L'environnement SAS/IML Studio permet de développer et d'exécuter plusieurs programmes **simultanément**
- Chaque environnement de programme (appelé **workspace**) a sa propre fenêtre de programme, fenêtre de sortie, fenêtre log et fenêtres graphiques et de données
- Lorsqu'on travaille sur un programme, les fenêtres associées à un autre programme sont cachées, ce qui permet d'éviter les confusions
- Chaque workspace a sa propre librairie Work pour stocker les données temporaires

SAS/IML Studio

- Le programme IML précédent est également valide dans SAS/IML Studio
- La seule différence est qu'on a pas à spécifier PROC IML en début de programmes
- SAS/IML Studio suppose que le programme est un programme IMLPlus
- Ainsi, le programme précédent s'écrit simplement :

SAS/IML Studio

```
Celsius = {-40, 0, 20, 37, 100};
Fahrenheit = 9/5 * Celsius + 32;
```

- Dans la suite du cours, on utilisera la syntaxe SAS/IML Studio et on ne mentionnera donc pas l'instruction PROC IML
- On va d'abord présenter les points communs à la PROC IML et à IMLPlus
- On abordera ensuite les graphiques et d'autres fonctionnalités qui ne sont pas disponibles via la PROC IML

Introduction au langage matriciel SAS/IML

Salim Lardjane - Université de Bretagne-Sud

Création de matrices

- Dans un programme SAS/IML, toutes les variables sont des matrices
- Les matrices sont dynamiquement redéfinies au besoin
- Par exemples, les instructions suivantes sont toutes valides :

Création de matrices

```
x = 1;
x = {1 2 3};
y = {1 2 3, 4 5 6};
y = {"homme" "femme"};
```

- La première instruction définit x comme une matrice scalaire numérique
- La deuxième instruction la redéfinit comme un vecteur ligne; les espaces séparent les données des différentes colonnes

Création de matrices

- La troisième instruction définit y comme une matrice numérique de dimensions 2 x 3
- La dernière instruction redéfinit y comme une matrice caractère de dimensions 1 x 2

Affichage de matrice

- On peut utiliser l'instruction PRINT pour afficher les valeurs d'une ou plusieurs matrices
- Par exemple,

```
print x, y;
```

affiche les valeurs des matrices définies précédemment

- La virgule signifie que y doit être affichée sur une nouvelle ligne
- Sinon, les matrices sont affichées côte à côte

Affichage de matrice

- L'instruction PRINT admet quatre options utiles qui affectent la façon dont une matrice est affichée
- COLNAME = *matrice* spécifie une matrice caractère à utiliser pour les noms de colonnes
- FORMAT = *format* spécifie un format SAS ou défini par l'utilisateur à utiliser lors de l'affichage des valeurs de la matrice

Affichage de matrice

- LABEL = *label* spécifie un label pour la matrice. Si cette option n'est pas spécifiée, le nom de la matrice est utilisé comme label
- ROWNAME = *matrice* spécifie une matrice caractère à utiliser pour les noms de lignes
- *Exemple :*

Affichage de matrice

```
ageGroup = {"<= 45", "> 45"};
status = {"single" "married" "divorced"};
counts = { 5 5 0,
           2 9 3 };
print counts[colname = status
             rowname = ageGroup
             label = "Marital status by age group"];
pct = counts / 24;
print pct[format = PERCENT7.1];
```

- On peut également utiliser ces options en y référant par leur première lettre c=, f=, l=, r=.

Dimensions d'une matrice

- On peut déterminer les dimensions d'une matrice en utilisant les fonctions NCOL et NROW
- Exemple :

```
n_x = nrow(x);
p_x = ncol(x);
print n_x p_x;
```

- Une matrice qui ne contient aucun élément est une matrice *vide*.

Dimensions d'une matrice

- Une matrice peut être vide pour plusieurs raisons :
  - Elle n'a pas été définie
  - La mémoire associée a été libérée à l'aide de l'instruction FREE
  - Elle est le résultat d'une requête qui renvoie l'ensemble vide
- Une matrice vide est de dimensions 0 x 0

Dimensions d’une matrice

- Exemple :

```
n_u = nrow(empty_matrix);
p_u = ncol(empty_matrix);
print n_u p_u;
```

Type d'une matrice

- Une matrice est soit de type numérique, de type caractère ou non définie
- On ne peut créer de matrice qui contienne à la fois des nombres et des chaînes de caractères
- On peut déterminer si une matrice est de type numérique ou caractère en utilisant la fonction TYPE
- Exemple :

Type d'une matrice

```
x = {1 2 3};
y = {"homme" "femme"};
type_x = type(x);
type_y = type(y);
print type_x type_y;
```

- Si la matrice est numérique, la fonction TYPE renvoie le caractère 'N'; si elle est de type caractère, elle renvoie 'C '. Si la matrice est vide, elle renvoie 'U ' pour *undefined* (non définie).

Type d'une matrice

- Exemple :

```
type_u = type(undefined_matrix);
if type_u = 'U' then
     msg = "La matrice n'est pas définie.";
else
     msg = "La matrice est définie.";
print msg;
```

Longueur d'une matrice caractère

- Les matrices caractères de SAS/IML présentent certaines similarités avec les variables caractère de l'étape DATA
- Dans l'étape DATA, la longueur d'une variable caractère est déterminée lorsque la variable est initialisée
- C'est fait soit explicitement à l'aide de l'instruction LENGTH ou implicitement par la longueur de la première valeur de la variable

Longueur d'une matrice caractère

- De façon analogue, chaque élément d'une matrice caractère SAS/IML a le même nombre de caractères : la longueur du plus long élément
- La longueur est déterminée lorsque la matrice est initialisée
- Les chaînes de caractères plus courtes sont complétées par des blancs jusqu'à atteindre la longueur du plus long élément

Longueur d'une matrice caractère

- Par exemple, les instruction suivantes définissent une matrice caractère de longueur 4 :

```
c = {"Low" "Med" "High"};
```

- La matrice c est initialisée à la longueur 4, la longueur de son plus long élément
- On peut déterminer la longueur d'une matrice caractère à l'aide de l'instruction NLENG
- Celle-ci renvoie un nombre

Longueur d'une matrice caractère

- On peut déterminer le nombre de caractère de chaque élément d'un matrice caractère à l'aide de l'instruction LENGTH
- Exemple :

```
c = {"Low" "Med" "High"};

nlen = nleng(c);
len = length(c);
print nlen len;
```

Longueur d'une matrice caractère

- La fonction LENGTH renvoie une matrice numérique de mêmes dimensions que son argument
- Dans l'exemple, l'élément *ij* de len est la longueur de l'élément *ij* de c
- Lorsqu'on affecte une valeur à un élément d'une matrice existante, celle-ci est *tronquée* si elle est trop longue

Longueur d'une matrice caractère

- Exemple :

```
c[2] = "Medium";
print c;
```

- La matrice c étant intialisée à la longueur 4, seuls les 4 premiers éléments de la nouvelle valeur sont retenus
- On ne peut changer dynamiquement la longueur d'une matrice caractère

Longueur d'une matrice caractère

- On peut toutefois copier son contenu dans un vecteur qui est plus long (ou moins long).
- Exemple faisant appel à l'instruction PUTC de SAS/Base :

```
c = {"Low" "Med" "High"};
d = putc(c, "$6.");
d[2] = "Medium";
nlen = nleng(d);
print nlen, d;
```

Longueur d'une matrice caractère

- La fonction PUTC applique le format $6. à chaque élément de c.
- Elle renvoie une matrice de mêmes dimensions que c.
- Celle-ci est stockée dans d.
- Notons que PUTC agit sur chaque élément de c
- C'est vrai *en général* lorsqu'on applique une fonction de SAS/Base à une matrice

Longueur d'une matrice caractère

- Les chaînes de caractères moins longues que le longueur maximale sont complétées par des blancs *à droite*.
- Cet ajout d'espaces peut poser problème lorsqu'on utilise la fonction CONCAT ou l'opérateur de concaténation + pour concaténer des chaînes de caractères.
- La solution consiste à utiliser la fonction TRIM de SAS/Base.

Longueur d'une matrice caractère

- Exemple :

```
msg1 = "I like the " + d[1] + " value.";
msg2 = "I like the " + trim(d[1]) + "
          value.";
print msg1, msg2;
```

- Parfois, des chaînes de caractères ont également des blancs au début.
- Dans ce cas, on peut utiliser la fonction SAS/Base STRIP pour éliminer les blancs en début et fin de chaîne.

Création de matrice à l'aide de fonctions

- Les matrices SAS/IML introduites précédemment ont toutes été crées en saisissant leurs éléments.
- De façon plus standard, les matrices SAS/IML sont générées à l'aide de fonctions SAS/IML ou obtenues en lisant des données dans une table SAS.

Création de matrice à l'aide de fonctions

- La matrice la plus simple est la matrice constante.
- En SAS/IML, la fonction J permet de créer une matrice constante.
- La syntaxe est *J(nrow,ncol,value)*.
- Si on omet le troisième argument, la valeur retenue est la valeur par défaut 1.
- Exemple :

Création de matrice à l'aide de fonctions

```
c = j(10,1,3.14);
r = j(1,5);
m = j(10,5,0);
miss = j(3,2,.);
```

- On peut également utiliser la fonction REPEAT pour créer une matrice constante ou, de façon plus générale, une matrice obtenue en répétant un ensemble de valeurs.
- Exemple :

Création de matrice à l'aide de fonctions

```
g = repeat({0 1}, 3, 2);
print g;
```

- Résultat :

g

0	1	0	1
0	1	0	1
0	1	0	1

- Il est également possible d'utiliser J et REPEAT pour créer des matrices caractères.

Vecteurs de valeurs séquentielles

- Les matrices les plus simple à part les matrices constantes sont celles dont les éléments suivent une progression arithmétique.
- La fonction DO (à ne pas confondre avec le DO itératif) permet d'obtenir un vecteur dont les éléments constituent une suite arithmétique.
- La syntaxe est *DO(start,stop,increment)*.
- On peut également utiliser l'opérateur « : ».

Vecteurs de valeurs séquentielles

- Exemple :

```
i = 1:5;
j = 5:1;
k = do(1,10,2);
print i, j, k;
```

- Résultat :

i

1	2	3	4	5

j

5	4	3	2	1

k

1	3	5	7	9

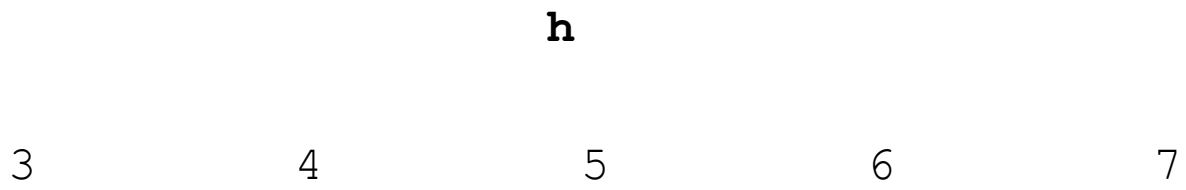
Vecteurs de valeurs séquentielles

- L'opérateur : est de priorité inférieure à celle des opérateurs arithmétiques.
- Exemple :

```
n1 = 1;
n2 = 10;
h = n1+2:n2-3;
print h;
```

- Résultat :

Vecteurs de valeurs séquentielles



- La fonction DO requiert que ses arguments soient numériques.
- On peut toutefois l'utiliser pour générer des noms de variables avec un préfixe commun.
- C'est fait de la façon suivante :

Vecteurs de valeurs séquentielles

```
varNames = "x1":"x10";
print varNames;
```

- Résultat :

```
                varNames

x1   x2   x3   x4   x5   x6   x7   x8   x9   x10
```

Matrices pseudo-aléatoires

- Il existe plusieurs fonctions SAS/IML permettant de générer des nombres pseudo-aléatoires.
- Les fonctions UNIFORM et NORMAL génèrent des valeurs distribuées uniformément sur [0,1] et selon une loi gaussienne standard, respectivement.
- Exemple :

Matrices pseudo-aléatoires

```
seed = j(10,1,1);
x = uniform(seed);
y = 3*x +2 + normal(seed);
```

- SEED fixe les dimensions des matrices à générer (ici 10 x 1) et la valeur du germe (ici 1).
- Les fonctions UNIFORM et NORMAL sont particulièrement utiles pour des simulations simples et rapides.

Matrices pseudo-aléatoires

- Un simulateur aléatoire plus sophistiqué est fourni par la routine RANDGEN.
- On utilisera de préférence celle-ci lorsqu'on a à générer des millions de nombres pseudo-aléatoires.
- Exemple :

Matrices pseudo-aléatoires

```
call randseed(12345);
x = j(10,1);
e = x;
call randgen(x, "Uniform");
call randgen(e, "Normal");
y = 3 * x + 2 + e;
```

Transposée d'une matrice

- Les vecteurs créés par la fonction DO et l'opérateur « : » sont des vecteurs lignes.
- On peut les *transposer* pour obtenir des vecteurs colonnes.
- La fonction à utiliser est la fonction T.
- La syntaxe en est *T(matrix)*.
- Exemple :

Transposée d'une matrice

```
s = {1 2 3, 4 5 6, 7 8 9, 10 11 12};
transpose = t(s);
print transpose;
```

- Résultat :

transpose

1	4	7	10
2	5	8	11
3	6	9	12

Transposée d'une matrice

- On peut également utiliser la notation ` pour transposer une matrice.
- Exemple :

```
sPrime = s`;
```

Redimensionnement

- Il est parfois utile de redimensionner une matrice.
- Supposons qu'on parte d'une matrice 1 x 12
- On peut vouloir transférer les données dans des matrices de dimensions 2 x 6, 3 x 4 ou 4 x 3 etc.
- Ceci peut être fait à l'aide de la fonction SHAPE.

Redimensionnement

- Exemple :

```
x = 1:12;
s = shape(x, 4, 3);
```

- La syntaxe est *SHAPE(ancienne matrice, nombre de lignes, nombres de colonnes).*
- On peut également spécifier uniquement le nombre de lignes ou le nombre de colonnes.
- La dimension non spécifiée est déterminée automatiquement.

Redimensionnement

- Pour spécifier uniquement le nombre de colonnes, on met 0 en nombre de lignes.
- Exemple :

```
s1 = shape(x,4);
s2 = shape(x,0,3);
```

- On peut également spécifier une valeur à utiliser lorsque les données initiales ne suffisent pas à remplir la nouvelle matrice.

Redimensionnement

- Exemple :

```
s = shape(1:10,4,3,.);
print s;
```

- Résultat :

s

1	2	3
4	5	6
7	8	9
10	.	.

Extraction d'éléments

- On peut identifier les éléments d'une matrice avec un indice ou deux. Par exemple, x[3] est le troisième élément de la matrice x lue ligne par ligne.
- Par contre, y[1,2] correspond à l'élément dans la première ligne et la deuxième colonne de y.
- On peut utiliser les indices des deux côtés du symbole d'affectation (=).

Extraction d'éléments

- Exemple :

```
x = {1 2 3, 4 5 6};
y = x[2,3];
x[2,3] = 7;
print x, y;
```

- Les indices négatifs ou nuls ne sont pas autorisés sous SAS/IML.

Extraction de lignes et de colonnes

- On peut identifier une sous-matrice d'une matrice en spécifiant des vecteurs d'indices.
- Par exemple, les instructions suivantes affectent à la matrice w les valeurs se trouvant dans les lignes impaires et les colonnes paires de la matrice z.
- Exemple :

Extraction de lignes et de colonnes

```
z = {1 2 3 4, 5 6 7 8, 9 10 11 12};
rows = {1 3};
cols = {2 4};
w = z[rows,cols];
print w;
```

- On peut sélectionner toutes les lignes d'une matrice en omettant les indices de lignes. De même pour les colonnes.
- Exemple :

Extraction de lignes et de colonnes

```
oddRows = z[rows,];
evenCols = z[,cols];
```

- Lorsqu'on extrait une sous-matrice en utilisant un simple vecteur d'indices, la matrice résultante est toujours un *vecteur colonne*.
- Exemple :

```
z = {1 3 5 7 9 11 13};
w = z[{2 4 6}];
t = z[,{2 4 6}];
print w t;
```

Extraction de lignes et de colonnes

- Résultat :

```
w                t

 3               3               7               11
 7
11
```

- w est un vecteur colonne bien que z soit un vecteur ligne.

Extraction de lignes et de colonnes

- On peut également extraire toutes les lignes *sauf* certaines. A cet effet, on utilise la fonction SETDIF.
- Exemple :

```
q = {1 2, . 3, 4 5, 6 7, 8 .};
v = {2 5};
idx = setdif(1:nrow(q),v);
r = q[idx,];
print idx, r;
```

Extraction de lignes et de colonnes

- Résultat :

```
idx

1        3        4
```

```
r

1        2
4        5
6        7
```

Diagonales

- Un sous-ensemble important d'une matrice est sa diagonale.
- Celle-ci peut être extraite à l'aide de la fonction VECDIAG.
- Exemple :

```
m = { 1 2 3,
      2 5 6,
      3 6 10};
d = vecdiag(m);
print d;
```

Diagonales

- Résultat :

```
d

     1
     5
    10
```

- La fonction VECDIAG renvoie un vecteur contenant la diagonale de la matrice.
- La fonction DIAG permet de créer une matrice à partir d'une diagonale.

Diagonales

- Exemple :

```
vals = {5, 2, -1};
s = diag(vals);
print s;
```

- Résultat :

s

```
5     0     0
0     2     0
0     0    -1
```

Diagonales

- La fonction I permet d'obtenir une matrice identité de dimension donnée.
- Exemple :

```
ident = I(3);
print ident;
```

- Résultat :

ident

1	0	0
0	1	0
0	0	1

Diagonales

- Si l'on souhaite extraire ou modifier les valeurs diagonales d'une matrices n x p quelconque, on peut directement spécifier les indices des éléments de la diagonale.
- Exemple :

```
n = nrow(m);
p = ncol(m);
diagIdx = do(1, n*p, p+1);
print diagIdx;
d2 = m[diagIdx];
print d2;
```

Diagonales

- Résultat :

diagIdx

```
1        5        9
```

d2

```
 1
 5
10
```

Affichage d'une sous-matrice ou expression

- On peut souhaiter afficher une sous-matrice d'une matrice donnée ou le résultat temporaire d'une expression matricielle.
- Cela peut être fait en utilisant des parenthèses avec PRINT.
- Exemple :

```
x = shape(1:1000,0,4);
print (x[1:3,]);
```

Affichage d'une sous-matrice ou expression

- Résultat :

```
1     2     3     4
5     6     7     8
9    10    11    12
```

- Le résultat ne contient pas de nom de matrice comme c'est le cas usuellement avec PRINT.
- On peut personnaliser l'affichage en utilisant les options LABEL et COLNAME de PRINT.
- Exemple :

Affichage d'une sous-matrice ou expression

```
varNames = 'Col1':'Col4';
print (x[1:3,])[label = "My Data" colname =
     varNames];
```

- Résultat :

My Data			
Col1	**Col2**	**Col3**	**Col4**
1	2	3	4
5	6	7	8
9	10	11	12

Affichage d'une sous-matrice ou expression

- On peut également afficher le résultat d'expressions matricielles en utilisant les parenthèses avec PRINT.
- Exemple :

```
x = 1:4;
print (3*x+1);
```

- Résultat :

```
 4          7         10         13
```

Opérateurs de comparaison

- Sous SAS/IML, l'opérateur = joue deux rôles. Il peut correspondre à une *affectation* ou à une *comparaison*.
- Les autres opérateurs de comparaison sont <, <=, >, >=, ^=.
- Usuellement, on compare une matrice à une autre matrice de mêmes dimensions ou à une valeur scalaire, bien qu'il soit théoriquement possible de comparer une matrice à un vecteur.

Opérateurs de comparaison

- Le résultat de la comparaison est une matrice de 0 et de 1. Les composantes pour lesquelles la condition de comparaison est vraie sont égales à 1; les autres sont égales à 0.
- Exemple :

```
x = {1 2 3, 2 1 1};
s1 = (x=2);
print s1;

z = {1 2 3, 3 2 1};
s2 = (x<z);
print s2;
```

Opérateurs de comparaison

- Résultat :

s1

```
0       1       0
1       0       0
```

s2

```
0       0       0
1       1       0
```

Opérateurs de comparaison

- Contrairement à l'étape DATA, les mots-clefs EQ, NE, GT, LT, GE et LE ne peuvent être utilisés pour remplacer =, ^=, >, <, >=, <=.
- Les opérateurs de comparaison traitent les valeurs manquantes comme des valeurs inférieures à toute valeur valide non-manquante.
- Exemple :

Opérateurs de comparaison

```
m = .;
n = 0;
r = (m<n);
print r;
```

- Résultat :

r

1

Instructions de contrôle

- Les instructions de contrôle SAS/IML incluent le IF-THEN/ELSE, le DO itératif, le DO/WHILE et le DO/UNTIL.
- Les opérateurs de comparaison sont le plus souvent utilisés dans la condition d'une instruction IF-THEN/ELSE.
- Exemple :

Instructions de contrôle

```
x = {1 2 3, 2 1 1};
z = {1 2 3, 3 2 1};
if x <= z then
     msg = "all(x <= z)";
else
     msg = "Some element of x greater than
     the corresponding element of z";
print msg;
```

Instructions de contrôle

- L'expression x <= z se résout en une matrice de 0 et de 1. Si cette matrice ne contient que des 1 alors l'instruction suivant le THEN est exécutée, sinon l'instruction suivant le ELSE est exécutée.
- Pour grouper plusieurs instructions à exécuter à la suite, on peut utiliser l'instruction DO/END.
- Exemple :

Instructions de contrôle

```
if x <= z then do;
     msg = "all(x <= z)";
     /* autres instructions */
end;
else do;
     msg = "Some element of x greater than
          the corresponding element of z";
     /* autres instructions */
end;
```

All/Any

- Afin de mettre l'accent sur la condition testée, on peut utiliser la fonction ALL.

```
if all(x <= z) then do;
      msg = "all(x <= z)";
      /* autres instructions */
end;
```

- La fonction ALL renvoie la valeur 1 si tous les éléments de son argument sont non nuls, 0 sinon.

All/Any

- Si on veut tester *s'il existe* un élément de la matrice qui est non nul, on peut utiliser la fonction ANY.
- Exemple :

```
if any(x < z) then
      msg = "Some element of x less than the
            corresponding element of z";
print msg;
```

All/Any

- La fonction ANY renvoie la valeur 1 si un élément au moins de son argument est non nul, 0 sinon.

DO Itératif

- Le DO *itératif* permet de répéter un groupe d'instructions plusieurs fois.
- Par exemple, on peut boucler sur les éléments d'une matrice ou sur les variables d'un tableau de données.
- La syntaxe est identique à ce qu'elle est dans l'étape DATA.
- Exemple :

DO Itératif

```
x = 1:5;
sum = 0;
do i = 1 to ncol(x);
     sum = sum + x[i];
end;
print sum;
```

- Résultat :

```
      sum

        15
```

DO Itératif

- En général, il est recommandé d'éviter de boucler sur tous les éléments d'un vecteur.
- Le langage SAS/IML dispose de nombreuses fonctions et instructions permettant d'éviter les boucles explicites.
- Par exemple, le programme précédent peut être réécrit de façon à éviter d'utiliser une boucle explicite :

DO Itératif

```
x = 1:5;
sum = sum(x);
```

- On utilise ici la fonction SUM qui prend pour argument une matrice et renvoie la somme de ses éléments.
- Le DO itératif supporte des clauses optionnelles WHILE et UNTIL.
- On peut les utiliser pour quitter une boucle lorsqu'une condition est vérifiée.

DO Itératif

- Exemple :

```
x = 1:5;
sum = 0;
do i = 1 to ncol(x) until (sum > 8);
     sum = sum + x[i];
end;
print sum;
```

- La clause UNTIL est évaluée en fin de boucle, alors que WHILE est évaluée en début de boucle.

DO Itératif

- Exemple :

```
x = {1 2 . 4 5};
sum = 0;
do i = 1 to ncol(x) while (x[i] ^= .);
     sum = sum + x[i];
end;
print sum;
```

Concaténation

- On a vu comment extraire un sous-ensemble d'éléments d'une matrice.
- On va voir à présent comment *concaténer* des matrices de façon à pouvoir obtenir des matrices plus grandes à partir de matrices plus petites.
- SAS/IML propose deux opérateurs de concaténation :

Concaténation

- L’opérateur de concaténation horizontale || permet d’ajouter de nouvelles colonnes à une matrice ou de combiner deux matrices ayant le même nombre de lignes.
- Une utilisation typique consiste à l’utiliser pour définir une matrice à partir de plusieurs vecteurs colonnes.
- Exemple :

Concaténation

```
a = {1,2,3,4,5};
b = {3,5,4,1,3};
c = {0,1,0,0,1};
x = a || b || c;
print x;
```

- Résultat :

x

1	3	0
2	5	1
3	4	0
4	1	0
5	3	1

Concaténation

- L’opérateur de concaténation verticale // permet d’ajouter de nouvelles lignes à une matrice ou de combiner des matrices qui ont le même nombre de colonnes.
- Un usage typique consiste à regrouper plusieurs sous-ensembles de données.
- Exemple :

Concaténation

```
hommes = {1 3 0, 2 5 1, 3 4 0};
femmes = {4 1 0, 5 3 1};
x = hommes // femmes;
```

- Les opérateurs de concaténation allouent une nouvelle matrice et y recopient les données.
- Il est donc recommandé de les éviter au sein des boucles.
- On allouera l'espace pour la matrice à définir *avant* la boucle, par exemple à l'aide de la fonction J avec . pour valeur.

Concaténation

- Les opérateurs de concaténation permettent de mettre des matrices « bout à bout ».
- Si on souhaite insérer des lignes ou des colonnes au sein même d'une matrice, on peut utiliser la fonction INSERT.
- Pour supprimer des lignes ou des colonnes, on utilisera la fonction REMOVE.
- Pour plus de détails, voir le *SAS/IML User's Guide*.

Opérateurs logiques

- Les instructions IF-THEN/ELSE, DO-UNTIL et DO-WHILE testent toutes si une condition est vraie.
- Lorsque cette condition fait intervenir une matrice, elle est considérée comme « vraie » lorsqu'elle est vraie pour chaque élément de la matrice.
- Il est également possible de tester si plusieurs conditions sont vérifiées simultanément en les combinant au sein d'une même expression logique.

Opérateurs logiques

- Les trois opérateurs logiques sont & (ET), | (OU) et ^ (NON).
- Ces opérateurs effectuent des opérations logiques sur des éléments de matrices.
- Exemple :

```
r = {0 0 1 1};
s = {0 1 0 1};
and = (r & s);
or = (r | s);
not = ^r;
print and, or, not;
```

Opérateurs logiques

- Résultat :

and

0	0	0	1

or

0	1	1	1

not

1	1	0	0

Opérateurs logiques

- Contrairement à l'étape DATA, SAS/IML ne reconnaît pas les mots-clefs AND, OR et NOT.
- De plus, SAS/IML ne procède *pas* par évaluation minimale : tous les termes d'une expression logiques sont évalués.

Opérations ensemblistes

- SAS/IML propose trois fonctions ensemblistes.
- La fonction UNION renvoie la réunion des éléments d'une ou plusieurs matrices.
- La fonction XSECT renvoie les valeurs se trouvant simultanément dans deux matrices ou plus (intersection).
- La fonction SETDIF renvoie toutes les valeurs se trouvant dans son premier argument et pas dans son second (différence ensembliste).

Opérations ensemblistes

- Exemple :

```
A = 1:4;
B = do(2.5,4.5,0.5);
inter = xsect(A,B);
u = union(A,B);
dif = setdif(A,B);
print u, inter, dif;
```

Opérations ensemblistes

- Résultat :

```
u

 1          2        2.5          3        3.5          4        4.5
inter

 3          4
dif

 1          2
```

Opérations ensemblistes

- Notons que XSECT et SETDIF peuvent renvoyer comme résultat une matrice vide.
- Pour éviter une erreur à l'exécution lors de l'appel à PRINT, on teste usuellement que les matrices ne sont pas vides en comparant leur nombre de colonnes à 0 ou leur type à 'U'.
- Exemple :

```
dif = setdif(A,B);
if ncol(dif) > 0 then
   print dif;
```

Opérations matricielles

- SAS/IML permet de manipuler des expressions combinant scalaires, vecteurs et matrices.
- SAS/IML détermine automatiquement s'il est possible de donner un sens à l'expression soumise.
- Exemple :

```
x = {7 7, 8 9, 7 9, 5 7, 8 8};
grandmean = 7.5;
y = x - grandmean;
mean = {7 8};
xc = x-mean;
print y xc;
```

Opérations matricielles

- Résultat :

y		xc	
-0.5	-0.5	0	-1
0.5	1.5	1	1
-0.5	1.5	0	1
-2.5	-0.5	-2	-1
0.5	0.5	1	0

Opérations matricielles

- Exemple :

```
r = {1.225 1};
std_x = xc/r;
c = x[,1];
y = x-c;
m = {6.5 7.5, 7.9 9.1, 7.5 8.5, 5.6 6.4, 7.5
     8.5};
deviations = x-m;
print std_x y deviations;
```

Opérations matricielles

- Résultat :

std_x		y		deviations	
0	-1	0	0	0.5	-0.5
0.8163265	1	0	1	0.1	-0.1
0	1	0	2	-0.5	0.5
-1.632653	-1	0	2	-0.6	0.6
0.8163265	0	0	0	0.5	-0.5

Opérations matricielles

- La multiplication matricielle est donnée par *.
- Exemple :

```
A = {7 7, 8 9, 7 9, 5 7, 8 8};
v = {1, -1};
y = A*v;
print y;
```

Opérations matricielles

- Résultat :

```
y

 0
-1
-2
-2
 0
```

Opérations matricielles

- Les opérateurs binaires agissant élément par élément sont : l'addition +, la soustraction -, la multiplication élément par élément #, la division / et l'élévation à une puissance ##.
- Le seul opérateur unaire est la négation - qui équivaut à la multiplication par -1.
- On peut utiliser * à la place de # pour multiplier une matrice par un scalaire.

Gestion du Workspace

- SAS/IML permet de visualiser et de gérer les matrices en mémoire.
- En tout point du programme, on peut visualiser les matrices définies, libérer de la mémoire et sauvegarder les matrices dans un catalogue SAS.
- On peut utiliser l'instruction SHOW pour afficher les noms, dimensions et types des matrices définies.

Gestion du Workspace

- On peut spécifier une liste de matrices ou utiliser l'instruction SHOW NAMES pour afficher l'information sur toutes les matrices.
- Exemple :

```
proc iml;
x = 1:3;
y = j(1e5, 100);
animals = {"Cat" "Dog",
           "Mouse" "Horse"};
show names;
show y animals;
```

Gestion du Workspace

- Résultat :

```
SYMBOL     ROWS    COLS TYPE    SIZE
 ------   ------ ------ ---- ------
 animals       2      2 char       5
 x             1      3 num        8
 y        100000    100 num        8
  Number of symbols = 3  (includes those without values)

                y 100000 rows    100 cols    (numeric)
                animals      2 rows      2 cols    (character, size 5)
```

Gestion du Workspace

- Lorsqu'on n'a plus besoin de certaines matrices, on peut libérer la mémoire associée à l'aide de l'instruction FREE.
- Exemple :

```
free y;
show names;
```

- Résultat :

```
SYMBOL     ROWS    COLS TYPE    SIZE
 ------   ------ ------ ---- ------
 animals       2      2 char       5
 x             1      3 num        8
  Number of symbols = 3  (includes those without
values)
```

Gestion du Workspace

- On peut libérer la mémoire occupée par *toutes* les matrices à l'aide de l'instruction FREE /.
- Exemple :

```
free /;
show names;
```

- Résultat :

```
 SYMBOL    ROWS   COLS TYPE   SIZE
 ------  ------ ------ ---- ------
  Number of symbols = 3  (includes those without
values)
```

Gestion du Workspace

- Si on souhaite sauvegarder certaines matrices, par exemple parce qu'elles sont le résultat de longs calculs et qu'on doit quitter SAS, on peut utiliser l'instruction STORE.
- Par défaut, STORE sauvegarde les matrices dans un catalogue SAS dans la librairie Work.
- Toutefois, cette dernière est temporaire.
- Il est donc préférable d'utiliser l'instruction RESET STORAGE pour sauvegarder les matrices de façon *permanente*.

Gestion du Workspace

- Exemple :

```
x = 1:3;
animals = {"Cat" "Dog", "Mouse" "Horse"};
libname MyLib "D:\data\sasdata";
reset storage = MyLib.MyStorage;
store _all_;
show storage;
```

Gestion du Workspace

- Résultat :

```
Contents of storage library = MYLIB.MYSTORAGE

Matrices:
ANIMALS                                 X

Modules:
```

Gestion du Workspace

- Afin de récupérer les matrices, on utilise l'instruction LOAD.
- Exemple :

```
proc iml;
reset storage = MyLib.MyStorage;
load _all_;
show names;
```

Gestion du Workspace

- Résultat :

```
SYMBOL     ROWS    COLS TYPE    SIZE
------   ------ ------ ---- ------
ANIMALS       2      2 char       5
X             1      3 num        8
 Number of symbols = 3  (includes those without values)
```

- Lorsqu'on n'a plus besoin des matrices stockées dans le catalogue SAS, on peut les supprimer à l'aide de l'instruction REMOVE.

Gestion du Workspace

- Exemple :

```
remove _all_;
show storage;
```

- Résultat :

```
Contents of storage library = MYLIB.MYSTORAGE

Matrices:

Modules:
```

Gestion du Workspace

- Au lieu d'utiliser le mot-clef _ALL_ on peut spécifier une liste de noms de matrices à sauvegarder (STORE), lire (LOAD) ou supprimer (REMOVE).

SAS/IML
Techniques de programmation

Salim Lardjane - Université de Bretagne-Sud

Lecture et écriture de données

- Il est possible de créer une matrice en lisant des données dans une table SAS.
- On peut également créer une table SAS à partir du contenu d'une matrice.
- Par exemple, la table SAS *Vehicles* (dans …/wicklin) contient les caractéristiques techniques et les performances énergétiques de 1187 véhicules vendus en 2007.

Lecture et écriture de données

- On peut lire les données d'une table SAS en utilisant les instruction USE et READ.
- On peut lire les variables dans des vecteurs différents en spécifiant une matrice caractère qui contient les noms des variables qu'on souhaite lire.
- L'instruction READ crée des vecteurs colonnes ayant ces mêmes noms.

Lecture et écriture de données

- Exemple :

```
libname MyLib "d:\data\sasdata\wicklin";
varNames = {"Make" "Model" "Mpg_City" "Mpg_Hwy"};
use MyLib.Vehicles;
read all var varNames;
close MyLib.Vehicles;

idx = 1:3;
print  (Make[idx])
       (Model[idx])
       (Mpg_City[idx])[label="Mpg_City"]
       (Mpg_Hwy[idx])[label="Mpg_Hwy"];
```

Lecture et écriture de données

- Résultat :

		Mpg_City	Mpg_Hwy
Aston Martin	V8 Vantage	14	20
Aston Martin	V8 Vantage ASM	15	21
BMW	Z4 3 0I	20	30

Lecture et écriture de données

- L'instruction USE ouvre MyLib.Vehicles en lecture. L'instruction READ lit les observations dans des vecteurs ou dans une matrice.
- Dans la suite, on utilisera le plus souvent l'option ALL afin de lire *toutes* les observations.
- Pour plus de détails sur READ, voir le *SAS/IML User's Guide*.

Lecture et écriture de données

- La table MyLib.Vehicles est composée de 1187 observations mais seules les trois premières sont *affichées*.
- On peut également lire un ensemble de variables dans *une matrice* (en supposant qu'elles sont toutes de même type, numérique ou caractère).
- A cet effet, on utilise la clause INTO de l'instruction READ.

Lecture et écriture de données

- Exemple :

```
varNames = {"Mpg_City" "Mpg_Hwy"
     "Engine_Cylinders"};
use MyLib.Vehicles;
read all var varNames into m;
close MyLib.Vehicles;
print (m[1:3,])[colname=varNames];
```

Lecture et écriture de données

- Résultat :

Mpg_City	Mpg_Hwy	Engine_Cylinders
14	20	8
15	21	8
20	30	6

Lecture et écriture de données

- L'option COLNAME = de l'instruction PRINT est utilisée pour afficher les noms des variables en sortie.
- L'instruction READ/INTO *n'associe pas* de noms de variables aux colonnes de la matrice créée.
- On peut toutefois nommer de façon permanente les colonnes d'une matrice à l'aide de l'instruction MATTRIB.

Lecture et écriture de données

- Il est possible de lire uniquement les variables *numériques* d'une table SAS en utilisant le mot-clef _NUM_ dans l'instruction READ.
- Exemple :

```
use MyLib.Vehicles;
read all var _NUM_ into y[colname=NumericNames];
close MyLib.Vehicles;
print NumericNames;
```

Lecture et écriture de données

- Résultat :

		NumericNames	
	COL1	**COL2**	**COL3**
ROW1	Engine_Liters	Engine_Cylinders	Mpg_City

		NumericNames	
	COL4	**COL5**	**COL6**
ROW1	Mpg_Hwy	Mpg_Hwy_Minus_City	Hybrid

Lecture et écriture de données

- La matrice *NumericNames* contient les *noms* des variables numériques qui sont lues; les colonnes de la matrice *y* contiennent les données.
- Afin de créer une table SAS à partir d'une matrice ou d'un vecteur, on peut utiliser les instructions CREATE et APPEND.

Lecture et écriture de données

- Exemple :

```
call randseed(12345);
x = j(10,1);
e = x;
call randgen(x,"Uniform");
call randgen(e,"Normal");
y = 3*x + 2 + e;

create MyLib.MyData var {x y};
append;
close MyLib.MyData;
```

Lecture et écriture de données

- L'instruction CREATE ouvre MyLib.MyData en écriture. Les variables *x* et *y* sont créées; le type des variables (numérique ou caractère) est déterminé par le type des vecteurs SAS/IML du même nom.
- L'instruction APPEND écrit les valeurs des vecteurs listés dans la clause VAR de l'instruction CREATE.
- L'instruction CLOSE ferme la table SAS.

Lecture et écriture de données

- Il est recommandé de toujours fermer les tables lorsqu'on a fini d'écrire ou de lire les données.
- Cela libère de la mémoire et des ressources système et permet d'éviter que les données soient corrompues si le programme s'arrête prématurément pour une raison ou pour une autre.

Lecture et écriture de données

- On peut écrire des vecteurs caractère de la même façon. Les vecteurs lignes sont écrits dans des tables SAS comme s'ils étaient des vecteurs colonnes.
- Si les vecteurs *x* et *y* ne contiennent pas le même nombre d'éléments, alors le vecteur le plus court est complété par des valeurs manquantes.

Lecture et écriture de données

- Si on souhaite créer une table SAS à partir d'une *matrice* de valeurs, on peut utiliser la clause FROM des instructions CREATE et APPEND.
- Si l'on ne spécifie pas explicitement les noms des variables, les noms par défaut sont COL1, COL2, etc.
- On peut spécifier les noms des variables en utilisant l'option COLNAME = de la clause FROM.

Lecture et écriture de données

- Exemple :

```
create MyLib.MyData2 from
      x[colname={"Random1" "Random2"
      "Random3"}];
append from x;
close MyLib.MyData2;
```

Références

- *Statistical Programming with SAS/IML Software,* R. Wicklin, SAS Institute
- *SAS/IML 9.22 User's Guide,* SAS Documentation.