PROGRAMMATION ET LOGICIELS ECONOMETRIQUES

Manfred GILLI

UNIVERSITE DE GENEVE
Département d'économétrie
1993 – 2006
(Ed. 10–11–2006)

TABLE DES MATIÈRES

1

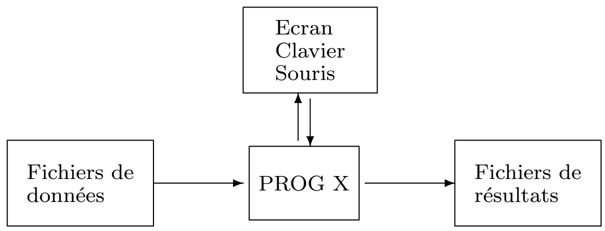
ENVIRONNEMENT DE TRAVAIL

L'environnement de travail des utilisateurs d'un ordinateur est en constante évolution (le *voice computing* n'est déjà plus de la science fiction). Pour fixer les idées il est néanmoins utile de faire la distinction entre:

- Système d'exploitation (DOS, Windows95, NT, Linux, etc.)
- Applications (programmes spécifiques: Excel, Word, Notepad, Powerpoint, Netscape, TSP, Maple, Matlab, S-Plus, etc.)

Un ordinateur est fait d'une unité de calcul (le processeur) et de composantes péripheriques comme la mémoire (rapide et permanente), l'écran, le clavier, etc. Le système d'exploitation gère les périphériques et permet de communiquer avec le processeur.

Les interactions existant lors de l'exécution d'un programme donné peuvent être schématisées comme suit:



Pour gérer les fichiers données et les fichiers résultats, on a besoin de deux choses:

- Le système d'exploitation pour déplacer, rénommer, imprimer, défiler, etc. Sous Windows95 on utilisera l'utilitaire `Explorer` et sous DOS les commandes spécifiques.
- Un programme particulier, appelé *éditeur de texte*, pour manipuler le contenu du fichier. Sous Windows95 on utilisera `Notepad` et sous DOS `Edit`.

Dans beaucoup de programmes ces deux fonctionnalités sont intégrées, c'est-à-dire que l'on peut exécuter ces tâches à partir du programme de manière transparente. Dans d'autres situations on doit recourir explicitement au système d'exploitation et à un éditeur de texte.

Fichiers

Toute information que l'on veut retrouver d'une séance de travail à une autre doit se trouver sur un fichier. Les codages peuvent être:

- Binaire (codé spécifiquement pour une application)
- ASCII (traité par un éditeur de texte). On l'utilise souvent pour les fichiers données.
- etc.

Noms de fichiers

La syntaxe complète du nom d'un fichier est:

[*unité disque*:] [*nom répertoire*\\]nom_fichier[.*extension*]

Schéma arborescent

Répertoire par défaut

2

TSP: NOTIONS GÉNÉRALES

TSP est un programme écrit en FORTRAN qui évolue depuis quelque 20 années. Originellement conçu à l'Université de Stanford, le programme est maintenant développé et maintenu par TSP International à Palo Alto. Il a été porté sur la plateforme PC dans les années 80 sous le système d'exploitation DOS. Depuis la version 4.4 il existe une interface Windows TLG (Through the Looking Glass).

Initialement conçu pour l'analyse des séries temporelles (Time Series Processor), ce logiciel est maintenant capable de résoudre pratiquement tous les problèmes économétriques. La gestion des données (génération de variables, base de données), qui constitue une tâche importante pour les problèmes économétriques, est inclue dans le logiciel.

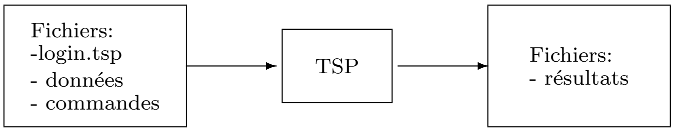
Du point de vue de sa performance (sophistication), comparé aux autres logiciels, TSP est bien placé. Son atout principal réside dans le fait qu'il est très largement répandu (un des premiers arrivés sur le marché).

2.1 MODES DE FONCTIONNEMENT

TSP peut fonctionner soit en mode "batch" (traitement par lot) soit en mode interactif. On choisira le mode de fonctionnement suivant la tâche à exécuter.

Mode batch

Le déroulement du programme est conditionné à l'avance. La séquence des instructions à exécuter se trouve dans un fichier. Une fois lancé, on ne peut plus intervenir dans le déroulement du programme.

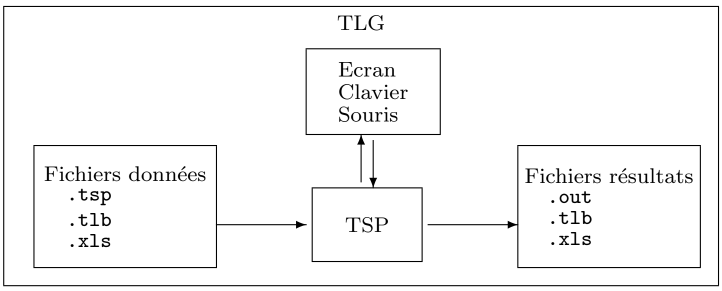


Ci-après la commande qui lance TSP dans un environnement DOS pour exécuter des commandes contenues dans un fichier `tp4.tsp`:

```
tsp tp4
```

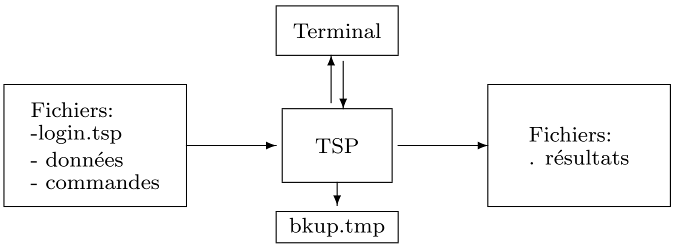
Notons que, pour des fichiers dont l'extension est `.tsp`, il n'est pas necessaire de la spécifier. Les résultats seront écrits dans un fichier de nom `tp4.out`.

Grâce à l'interface TLG (Through the Looking Glass), disponible avec la version 4.4 de TSP, l'édition des fichiers donnés et des fichiers résultats ainsi que le lancement de l'exécution de TSP peuvent se faire à partir d'une fenêtre Windows. L'éditeur disponible avec TLG n'est pas un éditeur ASCII pur. TLG filtre son contenu au moment de l'exécution.



Mode interactif

Le fonctionnment en mode interactif est seulement possible dans un environnement DOS. TSP exécute alors une commande après l'autre que l'utilisateur introduit au prompt du programme. Ainsi il est possible de contrôler le déroulement des opérations en fonction des résultats obtenus.



L'intérêt de ce type de fonctionnement réside dans le fait que l'utilisateur peut immédiatement corriger une erreur de syntaxe. Il peut également modifier l'orientation des calculs au vu des résultats intermédiaires. La suite des commandes exécutées par TSP est captée et conservée dans le fichier `bkup.tmp`.

Pour démarrer TSP en mode interactif (à partir d'un prompt DOS) il convient de préparer un fichier `tsp.bat` qui spécifie le nom (chemin) complet du fichier `tsp.exe` sur le système utilisé[1]. Les commandes pour démarrer et terminer une séance interactive de TSP à partir d'un répertoire quelconque sont alors:

```
tsp
```

logo TSP

```
?  ...
?  ...

?  quit
```

Le symbole `?` correspond au prompt de TSP et la commande `quit` permet de quitter la séance. Certaines commandes TSP sont spécifiques au fonctionnement en mode interactif. Ces commandes sont expliquées aux pages 33 et suivantes.

Pour la phase d'initiation à TSP on suggère d'utiliser le mode batch avec l'interface TLG.

[1] Sur une machine personelle ce nom correspond en général à `c:\progra~1\tsp44\tsp.exe`.

2.2 PRINCIPALES RÈGLES DE SYNTAXE

- TSP n'est pas "case sensitive". Les minuscules sont traduites en majuscules.
- Composition des noms:
  - Tout nom doit commencer par une lettre, _ % # ou @[2].
  - Les autres caractères peuvent être choisis parmi les lettres (A–Z) ou les digits (0–9).
  - La longueur maximale d'un nom est de 64 caractères.
- Des blancs ou une virgule constituent des séparateurs de noms.
- `;` Séparateur de commandes. En mode interactif il suffit de taper sur la touche `Return`.
- \ Continuation d'une commande sur plusieurs lignes en mode interactif.
- `?` Début d'un commentaire (la suite de la ligne est ignorée par TSP).

2.3 DOCUMENTATION ET AIDE EN LIGNE

La commande `help` permet d'obtenir une documentation des quelque 160 commandes disponibles de TSP. Ci-après la réponse de TSP à la commande `help`. On voit comment on peut explorer la documentation soit par sujet soit par des requêtes spécifiques.

```
   TSP  HELP  Menu
HELP          gives this menu
HELP COMMANDS  lists all the TSP commands, 10 per line
HELP command   gives details on a particular command
HELP FUNCTION  describes the functions and operators
HELP NONLIN    describes the nonlinear options
HELP ALL       describes all commands, alphabetically
HELP GROUP     describes all commands, by group
HELP n         describes all commands, in the nth group:
     1:    Linear Estimation
     2:    Nonlinear Estimation and Formula Manipulation
     3:    Qualitative Variable and General Maximum Likelihood
```

[2]Les commandes 2SLS et 3SLS constituent la seule exception.

```
 4:    Forecasting and Model Simulation
 5:    Data Transformations
 6:    Moving Data to/from Files
 7:    Control Flow
 8:    Editing Commands and/or Data
 9:    Display and Diagnostics
10:    Options
11:    Matrix Operations
12:    Hypothesis Testing
```

2.4 EXEMPLE DE SÉQUENCE DE COMMANDES

TSP interprète et exécute les commandes l'une après l'autre, dans l'ordre où elles sont données. Ainsi, la séquence des commandes doit respecter la hiérarchie de certaines opérations.

TSP analyse en premier lieu des séries temporelles. Ainsi il faut décrire le problème à traiter en donnant notamment des informations sur la périodicité et l'intervalle d'observations des données à analyser. Ci-après l'exemple d'une séquence de commandes.

`[options crt;]`

Définit des paramètres pour la mise en page des résultats en 80 colonnes. Il est conseillé de commencer un programme TSP avec l'instruction `options crt`. On peut, par la suite, redéfinir d'autres options. Ci-après des exemples d'options:

```
options baseyear = 2000;        (Valeur par défaut = 1900)
options nwidth=6, signif=2;
```

`name` *nom_tâche* `['`*Commentaire*`'];`

Permet l'identification du fichier résultat par exemple. Maximum 60 caractères pour le *Commentaire*.

```
name Paul_Samuelson 'Travail pratique No 4';
```

`[title '`*Commentaire*`';]`

Permet l'impression d'un titre ou d'un commentaire afin de structurer l'output dans le fichier résultats.

```
title 'Lecture donnees';
```

`freq` *périodicité*;

Spécification de la périodicité des observations. Les choix possibles sont:

`a`	annuelle
`q`	trimestrielle
`m`	mensuelle
`n`	absente (none)

Exemple de définition d'une périodicité annuelle des observations:

```
freq a;
```

`smpl` *n1 n2*;

Spécification de l'intervalle des observations. Exemples de syntaxe en fonction de la périodicité:

`a`	`1982 1985` ou `82 85`
`q`	`1972:1 1975:4` ou `72:1 75:4`
`m`	`67:8 72:12`
`n`	`1 29`

Le deuxième exemple sélectionne des observations à partir de plusieurs intervalles disjoints:

```
smpl 1975 1984;
smpl 65 68  70 75  83 87;
```

`load [(`*options*`)]` *liste_noms_variables*;

Lecture des observations. Cette commande possède de nombreuses options.

```
freq n; smpl 1 5;
load X;
   10 11 16 13 15;
load P M;
   41  13
   46  14
   50  16
   55  17
   60  18;
```

Pour le premier `load` le nombre d'éléments lus est défini par le `smpl`. TSP n'est pas sensible à l'indentation du deuxième `load` mais cette pratique accroît la lisibilité des instructions.

`print` *liste_variables*;

Impression des observations des variables spécifiées dans la liste pour un intervalle défini par la commande `smpl`.

Exemple de commandes avec résultat pour l'impression de trois observations dans un format donné.

```
options nwidth=6, signif=1;
smpl 2 4;
print x p m;
                X     P     M
2            11.0  46.0  14.0
3            16.0  50.0  16.0
4            13.0  55.0  17.0
```

2.5 TRANSFORMATION ET GÉNÉRATION DE VARIABLES

En spécifiant des expressions algébriques, on peut former de nouvelles variables. Les nouvelles variables peuvent à leur tour être utilisées dans des expressions. Les opérateurs arithmétiques sont:

`+`	addition
`-`	soustraction
`*`	produit
`/`	division
`**`	puissance

TSP applique les règles usuelles de précédence pour ces opérateurs. Ci-après une liste de quelques fonctions élémentaires. La liste complète peut être obtenue avec la commande `help function`.

`log`	logarithme en base e
`exp`	exponentielle
`abs`	valeur absolue
`log10`	logarithme en base 10
`sqrt`	racine carrée

Exemple de modification et création de variable.

```
[genr] z = p - x;
[genr] q = m + .7 * z - log(x);
```

2.6 STOCKAGE DES OBSERVATIONS

On peut représenter la façon dont TSP organise le stockage des observations au moyen de la matrice donnée dans la figure 2.1. Dans cette matrice les lignes correspondent aux indices des observations (années, trimestres, etc.) et les colonnes aux différentes variables.

La dimension de cette matrice n'est pas prédéfinie au démarrage du programme mais s'ajuste dynamiquement, c'est-à-dire au fur et a mesure des besoins lors de l'exécution du programme. Ce sont les commandes `smpl` et la génération de variables `genr` qui structurent l'organisation des observations dans cette matrice.

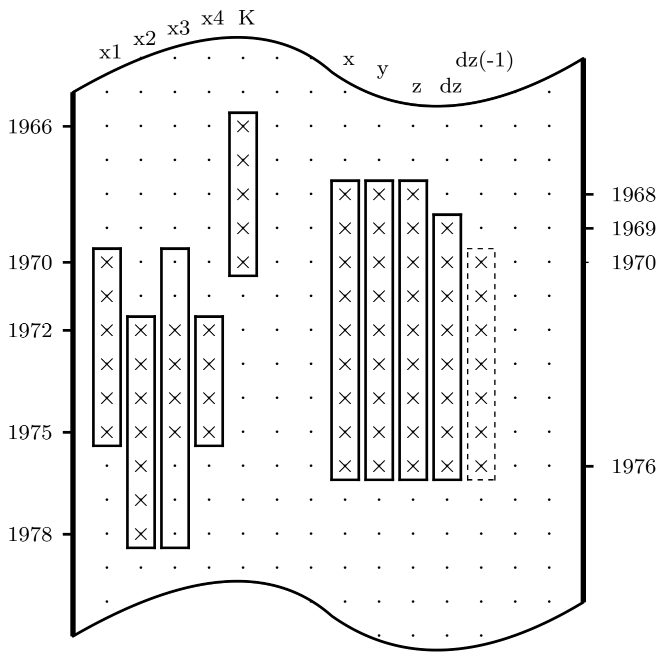


Figure 2.1 Matrice virtuelle des observations.

La commande `smpl` définit les indices des lignes auxquelles doivent correspondre des données chargées avec la commande `load` ou générées par une transformation.

Exemple de lecture de deux séries temporelles `x1` et `x2` définies sur des intervalles différents et génération d'une variable `x3` pour un intervalle couvrant `x1` et `x2`.

```
freq a;
smpl 1970 1975;
load x1;
   99.2 99 100 111.6 122.2 117.6;
smpl 1972 1978;
load x2;
   101 100.1 100 90.6 86.5 89.7 90.6;
smpl 1970 1978;
genr x3 = x1 / x2;
```

Cette opération engendrera le message suivant:

```
Current sample:    1970 to 1975
Current sample:    1972 to 1978
Current sample:    1970 to 1978
*** WARNING in line 7 Procedure GENR: Missing values for series ====>
    X1:  3,  X2:  2
*** WARNING in line 7 Procedure GENR: Some elements of a series set to
    missing values due to missing values.    Number ====>  5
```

On remarque que chaque commande `smpl` est suivie de l'impression "`Current sample`" qui définit le sous-ensemble des lignes (de la matrice virtuelle) qui sera affecté par les opérations subséquentes. D'autre part TSP imprime un avertissement lorsqu'on effectue des opérations sur des éléments non définis.

La création de la variable `x4` ci-après tient compte des périodes pour lesquelles `x1` et `x2` sont définies:

```
smpl 1972 1975;
genr x4 = x1 / x2;
```

2.7 VARIABLES DÉCALÉES DANS LE TEMPS

Dans la pratique économétrique on utilise fréquemment des variables décalées dans le temps. La notation `X(-1)` signifie que l'on considère les valeurs de la variable X décalées d'une période, c'est-à-dire, à l'année t, on utilisera la valeur de l'année $t-1$. On peut également décaler les variables dans le futur, par exemple `X(+3)`.

Il n'est pas nécessaire de définir les variables décalées au moyen d'une tranformation. TSP sélectionne les observations qui correspondent au décalage dans la colonne de la variable considérée. Ceci est illustré dans l'exemple qui suit:

```
freq a; smpl 1966 1970;
load K;
   2.4 3.9 3.2 2.8 3.5;
options  nwidth=5, signif=1;
print K(-1) K K(+1);
```

```
Current sample:    1966 to 1970
** WARNING in line 14 Procedure WRITE:
   Missing values for series ==> K(-1):  1, K(1):  1

         K(-1)     K  K(1)
1966        .    2.4   3.9
1967       2.4   3.9   3.2
1968       3.9   3.2   2.8
1969       3.2   2.8   3.5
1970       2.8   3.5    .
```

```
smpl 1967 1969;
print K(-1) K K(+1);
```

```
Current sample:    1967 to 1969
         K(-1)     K  K(1)
1967       2.4   3.9   3.2
1968       3.9   3.2   2.8
1969       3.2   2.8   3.5
```

Ainsi, lors de la définition de variables décalées, TSP adresse des éléments qui peuvent se trouver en dehors de l'intervalle de lignes définies par la commande `smpl`.

Considérons l'exemple suivant où l'on dispose des observations de 1968 à 1976 et avec lesquelles on désire estimer le modèle:

$$y = f(x, \Delta z_{-1}) \quad .$$

Du fait des variables décalées l'estimation ne peut se faire que sur l'intervalle 1970–1976.

```
freq a; smpl 1968 1976;
load y x z;
   ...
   ...
smpl 1969 1976;
dz = z - z(-1);
smpl 1970 1976;
```

```
olsq y c x dz(-1);
```

On rappelle qu'il n'est pas nécessaire de générer la variable décalée `dz(-1)` explicitement.

2.8 GRAPHIQUES

TSP permet la création de deux types de graphiques:

- Graphiques à faible résolution (character plot) d'aspect très médiocre, mais facilement imprimables;
- Graphiques à haute résolution qui ne peuvent être imprimés mais qui permettent l'analyse visuelle rapide de données et des résultats.

Les graphiques à haute résolution ne peuvent être produites qu'en mode interactif.

Graphiques à faible résolution

Le graphique s'imprime en faible résolution si les commandes sont exécutées en mode "batch".

`plot [(`*options*`)]` *var1 symbole1* `[`*var2 symbole2* ...`];`

Produit un graphique avec en abscisse la variable temps et en ordonnée les séries *var1* et *var2* avec les symboles *symbole1* respectivement *symbole2*. Tout caractère, excepté $; . ' " :, peut être utilisé comme symbole.

Ci-après les graphiques des séries temporelles `P` et `M` introduites plus haut:

```
plot p 1  m 2;
```

```
                              TIME SERIES PLOT
P PLOTTED WITH 1
M PLOTTED WITH 2
        MINIMUM                                              MAXIMUM
      13.00                                               60.00
        |-+-----------------------------------------------+-|
1961    | 2                             1                   |
1962    |  2                                 1              |
```

```
1963     |    2                                   1            |
1964     |     2                                        1      |
1965     |      2                                            1 |
         |-+---------------------------------------------+-|
       13.00                                        60.00
        MINIMUM                                          MAXIMUM
```

`graph` x y;

Produit un graphique avec en abscisse la variable x et en ordonnée la variable y.

Ci-après les graphiques des couples d'observations P et M introduites plus haut:

```
graph p m;
```

```
                              PLOT OF P        VERSUS M
P
             |---------------------------------------------------|
        60 -|                                                  *|
             |                                                   |
             |                                                   |
    55.250 -|                                        *          |
             |                                                   |
             |                                                   |
    50.500 -|                              *                    |
             |                                                   |
             |                                                   |
    45.750 -|          *                                        |
             |                                                   |
             |                                                   |
    41.000 -|*                                                  |
             |---------------------------------------------------| M
              |         |         |         |         |         |
              13             15.000              17.000
                    14.000              16.000              18.000
```

`hist [(`*options*`)]` *variable*;

Produit un histogramme des observations de *variable*.

Exemple de génération d'un vecteur u de 1000 observations d'une variable normale centrée réduite et construction de l'histogramme:

```
freq n; smpl 1 1000;
random u;
hist u;
```

```
HISTOGRAM OF U
               MINIMUM                                 MAXIMUM
             2.00000                                 244.00000
```

```
           |-+---------------------------------------+-|
-3.07663   | *                                         | 4
-2.42955   | ******                                    | 34
-1.78247   | ***************                           | 89
-1.13539   | ***********************************       | 207
-0.48831   | ***************************************** | 244
 0.15876   | **************************************    | 223
 0.80584   | *******************                       | 113
 1.45292   | *************                             | 72
 2.10000   | ***                                       | 12
 2.74708   | *                                         | 2
           |-+---------------------------------------+-|
         2.00000                                 244.00000
           MINIMUM                                 MAXIMUM
```

Graphiques à haute résolution

C'est le mode par défaut lorsque TSP est exécuté en mode interactif. Pour la commande `plot` les options sont réduites à ce qui est indiqué ci-après.

`plot [(max=`*valeur*`,min=`*valeur*`,origin)]` *liste_variables*;

`graph [(max=`*valeur*`,min=`*valeur*`,origin)]` *x y*;

3

ESTIMATION LINÉAIRE

Dans ce chapitre on traîte des modèles $y = f(x_1, \ldots, x_k)+u$ avec f une fonction linéaire, c'est-à-dire

$$y_t = \beta_0 + \beta_1 X_{t1} + \beta_2 X_{t2} + \ldots + \beta_k X_{tk} + u_t \quad t = 1, \ldots, T. \qquad (3.1)$$

Dans le cas d'un modèle à une seule variable explicative $y_t = \beta_0 + \beta_1 x_t + u_t$ les paramètres β_0 et β_1 à estimer définissent une droite. Géométriquement cette droite a la propriété de minimiser la somme des écarts au carré entre les points $(y_t,\, x_t)$ et la droite.

Figure 3.1 Modèle linéaire à une variable explicative.

Lorsqu'on considère deux variables explicatives le modèle devient un plan et au delà de deux variables explicatives, il s'agira d'un hyperplan.

3.1 LA COMMANDE `OLSQ`

`olsq [(`*options*`)]` *var_endogène* `[C]` *liste_vars_exogènes*`;`

Procédure d'estimation par les moindres carrés ordinaires. Le symbole `C` désigne la constante dans le modèle. Afin d'éviter toute confusion il ne faut pas utiliser le symbole `C` pour des variables. La dernière commande `smpl` qui précède la commande d'estimation, définit l'intervalle des observations pour lequel les paramètres du modèle sont estimés.

```
freq a;
smpl 1961 1965;
```

```
     load P M;
         41 13
         46 14
         50 16
         55 17
         60 18;
    options nwidth = 6, signif = 2;
    olsq  M   c  P;

Current sample:    1961 to 1965
                            Equation   1
                            ============
               Method of estimation = Ordinary Least Squares
Dependent variable:  M
Current sample:    1961 to 1965
Number of observations:    5
       Mean of dep. var. = 15.6      LM het. test = .032 [.857]
  Std. dev. of dep. var. = 2.07     Durbin-Watson = 2.52 [.458,.971]
Sum of squared residuals = .488  Jarque-Bera test = .353 [.838]
   Variance of residuals = .163   Ramsey's RESET2 = .565 [.531]
Std. error of regression = .403   F (zero slopes) = 103.  [.002]
               R-squared = .972    Schwarz B.I.C. = -1.68
      Adjusted R-squared = .962    Log likelihood = -1.28

           Estimated   Standard
Variable  Coefficient    Error    t-statistic  P-value
C         1.75         1.38       1.27         [.295]
P         .275         .027       10.1         [.002]
```

Description de l'output

TSP imprime les résultats de l'estimation sous forme de tableau. Il est également possible de manipuler les variables qui correspondent à ces résultats. Montrons comment l'utilisateur peut avoir connaissance du nom des variables et de leur contenu.

`show all;`

Cette commande donne la liste des variables définies au moment où la commande a été donnée ainsi que leur valeur. A la place de `all` il est aussi possible de spécifier `smpl`, `freq`, `scalar`, `series`, etc. pour n'obtenir que des informations plus restreintes.

Ci-après la liste de toutes les variables définies après avoir exécuté la commande `olsq` de l'exemple précédent.

```
    olsq M  c P;
    options nwidth = 8, signif = 4;
    show all;
Class    Name     Description
-----    ----     -----------
SCALAR   @NOB     constant 5.0000
MATRIX   @SMPL    vector, length  2
SCALAR   @FREQ    constant 1.0000
SERIES   P        5 obs. from  1961-1965, annual
         M        5 obs. from  1961-1965, annual
LIST     @LHV     1 members
SCALAR   @YMEAN   constant 15.6000
         @SDEV    constant 2.0736
         @SSR     constant 0.4882
         @S2      constant 0.1627
         @S       constant 0.4034
         @RSQ     constant 0.9716
         @ARSQ    constant 0.9622
         @LMHET   constant 0.03244
         %LMHET   constant 0.8571
         @DW      constant 2.5219
         %DWL     constant 0.4578
         %DWU     constant 0.9706
         @JB      constant 0.3532
         %JB      constant 0.8381
         @RESET2  constant 0.5647
         %RESET2  constant 0.5308
         @FST     constant 102.6844
         %FST     constant 0.002047
         @AIC     constant 1.3115
         @SBIC    constant -1.6826
         @LOGL    constant -1.2788
         @NCOEF   constant 2.0000
         @NCID    constant 2.0000
MATRIX   @COEF    2x1 general
         @SES     2x1 general
         @T       2x1 general
         %T       2x1 general
         @VCOV    2x2 symmetric
LIST     @RNMS    2 members
SERIES   @RES     5 obs. from  1961-1965, annual
         @FIT     5 obs. from  1961-1965, annual
```

On retrouve ici toutes les variables du tableau des résultats avec le nom que TSP leur assigne dans les procédures internes. L'utilisateur peut utiliser ces variables; nous donnerons des exemples plus loin.

Définition des variables

Afin de pouvoir donner la définition des variables qui font partie des résultats de la commande `olsq` considérons le modèle $y = X\beta + u$ défini en (3.1) où l'on note β le vecteur de $k+1$ paramètres, X la matrice des observations

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ 1 & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{T1} & \cdots & x_{Tk} \end{bmatrix}$$

et $y = [y_1\, y_2\, \ldots, y_T]'$ le vecteur des variables endogènes. Les définitions des variables calculées par la procédure `olsq` et les commandes TSP avec lesquelles l'utilisateur peut générer ces mêmes variables sont alors:

`@NOB` Nombre d'observations T

`@NCOEF` Nombre de paramètres $K+1$

`@FIT` Fitted values of the dependent variable:

$$\hat{y}_t = X_{t\,.}\,\hat{\beta}$$

`@RES` Residuals:

$$e_t = y_t - \hat{y}_t$$

```
RES = M - @FIT;
```

`@YMEAN` Mean of dependent variable:

$$\overline{y} = \frac{1}{T}\sum y_t$$

```
mat YMEAN = sum(M) / @NOB;
```

`@SDEV` Standard deviation of dependent variable:

$$\hat{\sigma}_y = \sqrt{\frac{SST}{T-1}} \qquad \text{avec} \qquad SST = \sum (y_t - \overline{y})^2$$

```
mat SST = sum((M - YMEAN)**2);
mat SDEV = sqrt(SST / (@NOB - 1));
```

`@SSR` Sum of squared residuals:

$$SSR = \sum e_t^2$$

```
mat SSR = sum(@RES**2);
```

`@S2` Variance of residuals:

$$\hat{\sigma}^2 = \frac{SSR}{T-(K+1)}$$

```
mat S2 = SSR / (@NOB - @NCOEF);
```

`@S` Standard error of the regression:

$$\hat{\sigma} = \sqrt{\frac{SSR}{T-(K+1)}}$$

```
mat S = sqrt(S2);
```

`@RSQ` R-squared:

$$R^2 = 1 - \frac{SSR}{SST} \qquad \textit{pour le modèle avec constante}$$

```
mat RSQ = 1 - SSR/SST;
```

`@ARSQ` Adjusted R-squared:

$$\overline{R}^2 = 1 - \frac{SSR/(T-(K+1))}{SST/(T-1)} \qquad \textit{pour le modèle avec constante}$$

```
mat ARSQ = 1 - (SSR  * (@NOB-1)) / (SST *(@NOB-@NCOEF));
```

`@DW` Durbin-Watson statistic:

$$\frac{\sum(e_t - e_{t-1})^2}{SSR}$$

```
smpl 1962 1965;
RES1 = RES - RES(-1);
mat DW = sum( RES1**2) / SSR;
smpl 1961 1965;
```

`@COEF` Estimated coefficients:

$$\hat{\beta} = (X'X)^{-1}X'y$$

```
mmake X  C P;
mat COEF = (X'X)"X'M;
```

`@S` Standard errors:

$$\hat{\sigma}_{\hat{\beta}} = \sqrt{\hat{\sigma}^2 \ \mathrm{diag}(X'X)^{-1}}$$

```
mat S = sqrt( S2 * diag((X'X)") );
```

`@T` t-statistics:

$$\hat{\beta}_i / \hat{\sigma}_{\hat{\beta}_i}$$

```
mat T =  (S") * COEF ;
```

Graphique valeurs calculées versus observations

`plots/noplot`

Si la commande `plots` précède la commande `olsq`, les résultats de l'estimation sont suivis d'un graphique des observations de la variable endogène y, des valeurs calculées par le modèle $\hat{y}$ et des résidus $\hat{e}$. Le défaut est `noplot`.

```
plots;
olsq  M   c  P;
```

```
ID   ACTUAL(*) FITTED(+)                     RESIDUAL(0)
                                                            0
1961  13.0000  13.0163  +                      -0.01627   +  0|  +
1962  14.0000  14.3906      *+                  -0.3906  0+   |  +
1963  16.0000  15.4901            +*             0.5099   +   |  +  0
1964  17.0000  16.8644                 +         0.1356   +   |0 +
1965  18.0000  18.2387                     *+   -0.2387   0   |  +
```

3.2 ESTIMATION DU MODÈLE AUTOREGRESSIF

Soit un modèle linéaire

$$y_t = x_t\beta + \epsilon_t \qquad t = 1, \ldots, T$$

dont le résidu aléatoire ϵ_t suit un procesus autoregressif du premier ordre definit comme

$$\epsilon_t = \rho\epsilon_{t-1} + u_t \tag{3.2}$$

avec $E(u) = 0$ et $E(uu') = \sigma^2 I$. Les paramètres β et ρ d'un tel modèle peuvent être estimés avec la commande `AR1`.

`ar1 [(`*options*`)]` *var_endogène* `[C]` *liste_vars_exogènes*`;`

Procédure d'estimation d'une équation avec autocorrelation du premier ordre des erreurs. Le symbole `C` désigne la constante dans le modèle. L'utilisateur a le choix parmi plusieures méthodes d'estimation. Par defaut le programme utilise la méthode du maximum de vraisemblance.

```
options nwidth = 6, signif = 2;
ar1 CP_HAB  c RD_HAB P_CP;
```

```
                                   Equation   2
                                   ============
                  FIRST-ORDER SERIAL CORRELATION OF THE ERROR
                  MAXIMUM LIKELIHOOD ITERATIVE TECHNIQUE
                  CONVERGENCE ACHIEVED AFTER   11 ITERATIONS
Dependent variable:  CP_HAB
Current sample:    1951 to 1976
Number of observations:    26
(Statistics based on transformed data) (Statistics based on original data)
         Mean of dep. var. = 526.         Mean of dep. var. = .302E+04
    Std. dev. of dep. var.  = 249. Std. dev. of dep. var. = .137E+04
  Sum of squared residuals = .174E+06  Sum of squared residuals = .195E+06
     Variance of residuals = .757E+04     Variance of residuals = .847E+04
  Std. error of regression = 87.0      Std. error of regression = 92.0
                 R-squared = .895                     R-squared = .996
        Adjusted R-squared = .886            Adjusted R-squared = .996
             Durbin-Watson = 1.62                 Durbin-Watson = 1.59
Rho(autocorrelation coef.)  = .873
     Standard error of rho = .088
       t-statistic for rho = 9.93
            Log likelihood = -152.
           Estimated   Standard
Variable  Coefficient    Error    t-statistic  P-value
C         .218E+04     370.0       5.89         [.000]
RD_HAB    649.0         69.5       9.33         [.000]
P_CP      -.441E+04    899.0      -4.91         [.000]
```

Les modèle estimé s'écrira comme

$$\begin{aligned}\hat{y}_t &= x_t\hat{\beta} + \hat{\rho}(y_{t-1} - x_{t-1}\hat{\beta}) \\ \hat{e}_t &= y_t - \hat{y}_t\end{aligned}$$

3.3 PRÉVISION DANS LE MODÈLE LINÉAIRE

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \vdots \\ \hat{y}_T \end{bmatrix} \quad \begin{bmatrix} 1 & x_{11} & \cdots & x_{1K} \\ 1 & x_{21} & \cdots & x_{2K} \\ 1 & x_{31} & \cdots & x_{3K} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ 1 & x_{T1} & \cdots & x_{TK} \end{bmatrix} \quad \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_K \end{bmatrix}$$

$$\begin{bmatrix} \hat{y}_{T+1} \\ \vdots \\ \hat{y}_{T+\tau} \end{bmatrix} \quad \begin{bmatrix} 1 & x_{T+1,1} & \cdots & x_{T+1,K} \\ \vdots & \vdots & & \vdots \\ 1 & x_{T+\tau,1} & \cdots & x_{T+\tau,K} \end{bmatrix}$$

4

BASES DE DONNÉES

TSP comporte des fonctionnalités pour la création et la gestion des données. La commande `help 6` liste les commandes qui sont spécifiques à la manipulation des fichiers de données.

```
help 6;
```

```
Moving Data to/from Files
CLOSE     Closes a data or output file
DBCOMP    Compresses a databank
DBCONV    Converts a databank to 4.0 format
DBCOPY    Converts TSP databank to portable format
DBDEL     Deletes variables from a databank
DBDUMP    Prints debug information on current databanks
DBLIST    Shows contents of TSP databanks
DBPRINT   Prints all series in a TSP databank
FETCH     Reads microTSP format databank files
IN        Search TSP databanks
INPUT     Reads TSP commands from an external file
KEEP      Marks variables for storage on a TSP databank
OUT       Causes automatic storage of new variables in databanks listed
OUTPUT    Directs output to a file instead of the screen
READ      Reads data from a file or from the command stream
RECOVER   Recovers lost program from INDX.TMP file
RESTORE   Restores TSP variables from a SAVE file
SAVE      Saves all current variables in a file
STORE     Writes microTSP-format databank files
TERMINAL  Redirects output to screen instead of to file
WRITE     Writes data to a file or to the screen/printout
```

4.1 BASES DE DONNÉES TLB

Lorsque l'utilisateur doit gérer des données d'un certain volume il convient de le faire avec des "bases de données TSP" qui sont des fichiers binaires. La lecture et l'écriture de ces fichiers se fait plus efficacement que ce serait le cas pour des fichiers données en format ASCII. Dans la suite on présente les commandes les plus importantes pour la création et la gestion des bases de données TSP.

`out` *nom_base_données*;

Crée un fichier de données binaire avec extension `.tlb` (Text Library). Toutes les variables chargés avec `load`, créées ou modifiées avec ou sans `genr` sont conservés dans le fichier avec l'information sur la périodicité et l'intervalle des observations.

Exemple de création d'une base de données `projet1.tlb` sur le répertoire `h:\ple`. Remarquons que le nom du fichier est spécifié entre quotes étant donné que le nom comporte plus de 8 caractères.

```
out 'h:\ple\projet1';
   freq a; smpl 1992 1995;
   load y x z;
        12.4  22.4  36.3
        12.7  22.0  37.0
        13.1  22.8  37.4
        13.8  24.1  40.2;
   genr w = x / z;
   smpl 1993 1995;
   r = log(w(-1)) + sqrt(y);
out;
```

La commande `out` sans argument à la fin termine la création de la base de données et ferme le fichier.

`DBlist` *nom_base_données*;

Liste le contenu de la base de données avec des informations sur le type de variable, fréquence et périodicité.

```
DBlist 'h:\ple\projet1';
```

```
Contents of Databank  PROJET1.TLB
Class    Name     Description
-----    ----     -----------
SERIES   Y        4 obs.  from  1992-1995, annual
         X        4 obs.  from  1992-1995, annual
         Z        4 obs.  from  1992-1995, annual
         W        4 obs.  from  1992-1995, annual
         R        3 obs.  from  1993-1995, annual
```

`DBprint` *nom_base_données*;

Imprime les observations des variables dans la base de données. L'intervalle des observations peut être défini avec la commnde `smpl`.

```
smpl 1994 1995;
options nwidth=7, signif=2;
DBprint 'h:\ple\projet1';
```

```
Current sample:    1994 to 1995
Values for all series in Databank  PROJET1.TLB
               Y       X       Z        W       R
1994       13.10   22.80   37.40    0.61    3.10
1995       13.80   24.10   40.20    0.60    3.22
```

`DBcopy` *nom_base_données*;

Crée un fichier ASCII avec extension `.tsp` qui correspond au contenu de la base de données. Ce fichier peut alors être exécuté sur une autre plateforme de calcul où il créera la même base de données.

```
smpl 1994 1995;
DBcopy 'h:\ple\projet1';
```

```
?  Re-create TSP databank
?  PROJET1.TLB
END; NOPRINT; OUT PROJET1;
FREQ A;
SMPL 1992,1995;
READ Y; 12.4 12.7 13.1 13.8;
READ X; 22.4 22 22.799999 24.1;
READ Z; 36.299999 37 37.400002 40.200001;
READ W; .61708 .594595 .609626 .599503;
SMPL 1993,1995;
READ R; 3.080949 3.099517 3.219925;
```

`DBdel` *nom_base_données* *liste_de_variables* ;

Efface les variables spécifiées dans la liste de la base de données.

`keep` *liste_de_noms*;

Sélectionne un ensemble de variables qui doivent être sauvées dans une base de données.

```
out 'h:\ple\projet2';
   keep w r;
out;
```

`in` *nom_base_données*;

Cette commande permet d'accéder aux variables d'une base de données créée préalablement avec la commande `out`.

```
in 'h:\ple\projet2';
   q = r * w;
in;
```

Les variables `w` et `r` sauvées préalablement dans le fichier `projet2` deviennent accessibles pour pouvoir calculer la variable `q`.

4.2 IMPORTATION ET EXPORTATION DE FICHIERS

Il est également possible d'échanger directement des données entre TSP et d'autres programmes. Ci-après nous donnons d'abord des informations sur les commandes qui permettent d'exporter ou d'importer des fichiers, dans le cas général, puis dans le cas particulier où l'on désire échanger des données entre TSP et EXCEL et TSP et MATLAB.

`read (file='`*nom_fichier*`',format='(`*spec_format*`)')` *liste_noms_variables*`;`

Importation de données vers TSP. Illustrons cette commande à l'aide de plusieurs exemples. Lecture à partir du fichier `ex1.dat` de trois observations des variables `x1`, `x2` et `x3` enregistrées en format libre (les valeurs sont séparées par un ou plusieurs blancs). Ci-après sont d'abord reproduites les trois lignes du fichier `ex1.dat`, puis les commandes TSP pour lire ce fichier:

```
12.1   1.2   3.7
13.3  2.2  4.2
 11.6  0.5   8.2
```

```
freq a;
smpl 1996 1998;
read (file='ex1.dat',format=FREE) x1 x2 x3;
```

Si les données dans le fichier `ex1.dat` correspondent à une matrice `X`, la lecture se fait comme:

```
read (file='ex1.dat',nrow=3,ncol=3)  X;
```

Lorsqu'on désire sélectionner des colonnes particulières il est nécessaire que le fichier de données soit écrit dans un format qui ne varie pas. Dans l'exemple suivant, on considère le fichier `ex2.dat` dans lequel on trouve les mêmes variables, mais où chaque observation est codée dans un champ de cinq caractères. Il est maintenant possible, en spécifiant les champs de lecture, de sélectionner des colonnes particulières:

```
12.10 1.20 3.70
13.30 2.20 4.20
11.60 0.50 8.20
```

```
freq a;
smpl 1996 1998;
read (file='ex2.dat',format='(F5.0,5X,F5.0)') x1 x3;
```

Dans la spécification du format, le symbole `F` indique un champ contenant des nombres réels (la longueur du champ étant ici de cinq) et le symbole `X` indique des espaces. Le nombre d'espaces à considérer précède le symbole `X` (ici ce nombre est cinq).

`write (file='`*nom_fichier*`',format='(`*spec_format*`)')` *liste_noms_variables*`;`

Exportation de données de TSP. Cette commande est en tous points semblable à la commande `read`.

Exemple de transformation à partir des variables `x1` et `x3` lues ci-dessus et exportation du résultat sur le fichier `ex3.dat`:

```
x4 = x1/x3;
write(file='ex3.dat',format='(3F10.3)') x1 x3 x4;
```

```
12.100     3.700     3.270
13.300     4.200     3.167
11.600     8.200     1.415
```

Les trois dernières lignes reproduisent le contenu du fichier `ex3.dat`. Le format correspond à trois champs de 10 caractères, avec trois champs après la virgule.

Echange de fichiers entre TSP et EXCEL

Considérons la feuille de calcul EXCEL qui contient trois observations de deux variables X1 et X2:

Date	X1	X2
1990	10.3	1.34
1991	20.6	3.7
1992	39.2	5.3

Afin de pouvoir être lus par TSP les noms des variables ne doivent pas excéder 8 caractères. Il faut ensuite sauvegarder le fichier en format `Microsoft EXCEl 4.0 Worksheet` et fermer la feuille de calcul EXCEL. Soit `ex_tab1.xls`, le nom sous lequel ce fichier a été sauvegardé dans le répertoitre `c:\ple\nc`. Les commandes TSP, qui permettent de lire le fichier `ex_tab1.xls`, transforment les données lues et créent une nouvelle feuille de travail de nom `ex_tab2.xls`, sont alors:

```
freq a;
smpl 1990 1992;
read (file='c:\ple\nc\ex_tab1.xls',format=EXCEL);
x3 = x1/x2;
write (file='c:\ple\nc\ex_tab2.xls',format=EXCEL)  X1 X3;
```

Après avoir fermé l'application TSP, la feuille de calcul EXCEL ainsi créée peut être ouverte avec EXCEL. Elle se présente ainsi:

Date	X1	X3
1990	10.3	7.686567
1991	20.6	5.567568
1992	39.2	7.396226

Echange de fichiers entre TSP et MATLAB

La première commande MATLAB de l'exemple suivant crée une matrice A. La deuxième sauvegarde les éléments de la matrice A dans un fichier `ex_tab3.txt` en code ASCII.

```
A = rand(3,4);
save ex_tab3.txt A -ASCII
```

Ci-après, le contenu du fichier `ex_tab3.txt`. Les champs sont définis en notation exponentielle, de longueur 16 avec 7 digits après la décimale. Un tel format se note: `E16.7`.

```
9.5012929e-001  4.8598247e-001  4.5646767e-001  4.4470336e-001
2.3113851e-001  8.9129897e-001  1.8503643e-002  6.1543235e-001
6.0684258e-001  7.6209683e-001  8.2140716e-001  7.9193704e-001
```

La lecture d'un tel fichier avec TSP s'effectue comme montré plus haut, c'est-à-dire:

```
freq n;  smpl 1 3;
```

```
read (file='ex_tab3.txt',format=FREE)  X1 X2 X3 X4;
close(file='ex_tab3.txt');
?
read (file='ex_tab3.txt',format='(E16.7,32X,E16.7)')   X5 X6;
```

Dans l'exemple, on lit deux fois le même fichier, une fois en format libre et la seconde fois dans un format défini. La commande `close`, entre les deux lectures, a pour effet de fermer le fichier de sorte que la seconde lecture se fasse à nouveau depuis la première ligne. (Les variables `X5` et `X6` de la seconde lecture correspondent aux variables `X1` et `X4` de la première lecture.)

Illustrons encore l'exportation des données de TSP vers MATLAB en créant deux fichiers, l'un formaté et l'autre en format libre.

```
x9  = log(x1);
x10 = exp(x2);
write (file='ex_tab5.txt',format='(E16.7,E16.7)')   X9 X10;
write (file='ex_tab6.txt',format=FREE)   X9 X10;
```

Le fichier `ex_tab5.txt` puis le fichier `ex_tab6.txt`:

```
-0.5115723E-01   0.1625772E+01
-0.1464738E+01   0.2438295E+01
-0.4994859E+00   0.2142765E+01

-0.5115723E-01   1.625772
-1.464738        2.438295
-0.4994859       2.142765
```

Dans MATLAB, on lira ces fichiers avec la commande:

```
load ex_tab5.txt -ASCII

whos

Name        Size      Bytes   Class
ex_tab5     3x2       48      double array
```

ce qui crée une matrice qui correspond au nom du fichier (sans l'extension).

5

MODE INTERACTIF

Une commande TSP peut en général être exécutée indifféremment, soit en mode batch, soit en mode interactif. Il existe cependant un ensemble de commandes destinées à faciliter le déroulement d'une séance interactive. En mode batch, ces commandes n'ont donc pas de raison d'être.

On rappelle que le fonctionnement interactif de TSP n'est possible que dans un environnement DOS. L'utilisateur définira d'abord son répertoire de travail en se servant notamment de la commande DOS `cd` (change directory). Par la suite toute référence à des fichiers se fera par défaut à des fichiers dans ce répertoire.

Pour démarrer une séance interactive on tape alors simplement la commande:

```
c:\>tsp
     Logo TSP
1 ?
```

Le symbole `?` est le "prompt" de TSP en mode interactif. La ligne de commande est pourvue d'un indice qui sera incrémenté de un pour chaque nouvelle commande. Cet indice permettra par la suite de se référer à une commande particulière donnée. Notons encore, qu'en mode interactif, il n'est pas nécessaire de terminer les commandes avec le symbole `;`.

Par la suite, toutes les commandes tapées par l'utilisateur seront conservées dans le fichier `bkup.tsp`, pour autant que la séance interactive se termine avec la commande `end`.

`help` (*options*)

Active le menu d'aide qui permet d'obtenir une description complète de toutes les commandes.

```
1 ?  help

   TSP  HELP  Menu
HELP           gives this menu
HELP COMMANDS  lists all the TSP commands, 10 per line
HELP command   gives details on a particular command
HELP FUNCTION  describes the functions and operators
HELP NONLIN    describes the nonlinear options
HELP ALL       describes all commands, alphabetically
HELP GROUP     describes all commands, by group
HELP n         describes all commands, in the nth group:
     1:    Linear Estimation
     2:    Nonlinear Estimation and Formula Manipulation
     3:    Qualitative Variable and General Maximum Likelihood
     4:    Forecasting and Model Simulation
     5:    Data Transformations
     6:    Moving Data to/from Files
     7:    Control Flow
     8:    Editing Commands and/or Data
     9:    Display and Diagnostics
    10:    Options
    11:    Matrix Operations
    12:    Hypothesis Testing
```

`input` *nom_fichier*

Lit une séquence de commandes TSP à partir d'un fichier et les exécute. Dans le fichier toutes les commandes doivent se terminer avec le symbole `;`. Lorsqu'on désire lire, par exemple, un fichier `ex01.tsp` qui se trouve dans le répertoire de travail, on utilise la forme suivante:

```
4 ?  input ex01
```

On remarque que le nom est spécifié sans extension (par defaut `.tsp`). Les noms des fichiers doivent obéir aux règles de syntaxe du DOS (maximum 8 caractères).

Si le fichier lu avec la commande `input` contient la commande `end`, TSP termine la séance interactive.

Lorsqu'on désire lire un fichier quelconque dans un répertoire quelconque on peut le faire comme indiqué ci-après:

```
5 ?  input  Return

Enter name of TSP input file:  c:\mne\ex\tp04.tsp
```

Le nombre maximum de caractères pour le nom du fichier est de 32. TSP permet aussi l'utilisation de commandes `input` imbriquées dans des fichiers successifs.

`dir`

Affiche la liste les fichiers d'un répertoire donné sans interrompre la séance interactive. Cette commande est utile si l'on cherche le nom d'un fichier que l'on veut lire avec la commande `input`).

```
1 ?  dir *                        (équivalent à dir *.tsp du DOS)
2 ?  dir  nom_fichier       (équivalent à dir nom_fichier.* du DOS)
3 ?  dir  [Return]
      files:   wildcards qualifiers                   (dir tp*.dat)
```

`review`

Affiche la liste des commandes entrées durant la séance interactive.

```
6 ?  review                          (affiche toutes les commandes)
7 ?  review  i                          (affiche la ième commande)
8 ?  review  n1 n2               (affiche les commandes de n1 à n2)
```

`exec`

Réexécute des commandes entrées durant la séance interactive.

```
10 ?  exec 5                               (exécute la commande 5)
11 ?  exec 3 7                       (exécute les commandes 3 à 7)
```

`find` *nom_commande*

Liste des commandes particulières.

```
9 ?  find olsq          (liste toutes les occurrences de  olsq)
```

`delete`

Efface des commandes entrées durant la séance interactive.

```
12 ?  delete i                           (efface la ième commande)
13 ?  delete n1 n2               (efface les commandes de n1 à n2)
```

`add/drop`

Ajoute ou enlève une variable contenue dans une liste de variables d'une commande et la réexécute.

```
14 ?  olsq C_cap c Revd Revp
15 ?  add pc tx_obl
16 ?  drop Revp
```

`output/terminal`

Cette commande dirige alternativement les résultats dans un fichier ou les affiche au terminal. Dans l'exemple suivant les résultats sont écrits dans un fichier `res01.out` sur le répertoire de travail courant. Plus rien n'apparaît au terminal après avoir donné la commande `output`:

```
18 ?  output res01
19 ?  terminal          (les résultats réapparaissent à l'écran)
20 ?  output res01          (la suite s'ajoute dans  res01.out)
21 ?  output 'a:res02.out'            (résultats écrits sur une
disquette)
```

Une succession de commandes `output` et `terminal` permet de diriger des résultats de manière sélective dans un fichier. Il n'est pas possible d'avoir les résultats à la fois au terminal et dans un fichier.

`show` (*options*)

Donne de l'information sur l'ensemble des variables définis dans la séance courante.

```
21 ?  show all      (affiche toutes les variables TSP définiés)

Class    Name     Description
-----    ----     -----------
SCALAR   @NOB     constant 5.00000
MATRIX   @SMPL    vector, length  2
SERIES   P        5 obs.  from  1961-1965, annual
         M        5 obs.  from  1961-1965, annual
LIST     @LHV     1 members
SCALAR   @YMEAN   constant 15.60000
         @SDEV    constant 2.07364
         @SSR     constant 0.48825
                ⋮
```

Il est aussi possible d'afficher uniquement les éléments d'une classe particulière avec la commande `show` *nom_classe*. Des noms de classes de variables sont `series`, `matrix`, `smpl`, `freq`, `matrix`, `proc`, etc.

```
22 ?  show series               (affiche uniquement les series)

Class    Name     Description
-----    ----     -----------
SERIES   P        5 obs.  from  1961-1965, annual
         M        5 obs.  from  1961-1965, annual
```

`system/continue`

Permet de revenir au DOS sans interrompre la séance interactive.

```
17 ?  system

    Enter System commands.  Type EXIT or CONTINUE to resume TSP

-> ...              (prompt DOS)
-> ...
-> ...
-> continue      (provoque le retour à TSP)

    Resuming interactive TSP session ...

18 ?
```

`collect/exit`

Permet l'exécution d'un bloc de commandes. (Voir programmation avec TSP). Ci-après le calcul d'une moyenne mobile en mode interactif:

```
23 ?  collect
   >> do i=2,5
     >>   set i1 = i-1
     >>   set x(i) = ( x(i1) + x(i) ) / 2
   >> enddo
   >> exit
24 ?
```

`end/quit`

Termine la séance interactive. En terminant avec la commande `quit` TSP ne sauvegarde pas l'historique des commandes de la session dans le fichier `bkup.tsp`.

Mentionnons encore qu'en mode interactif TSP conserve les commandes les plus récentes (environ 20) dans une une mémoire. Il est alors possible de les rappeler dans l'ordre à l'aide des touches "flèches" du pavé numérique.

6

GÉNÉRATION DE VARIABLES

Ce chapitre aborde quelques-uns des problèmes de création et de transformation que l'on rencontre couramment dans la pratique de la modélisation économétrique et montre comment les résoudre avec TSP.

Création d'une variable "trend"

`trend` *nom_variable* `[`*val_initiale*`]` `[`*incrément*`];`

Cette commande permet la création de variables artificielles définies par une progression arithmétique. Par défaut la valeur initiale et l'incrément valent 1.

```
smpl 60 63;
trend t1;
trend t2 13 5;
print t1 t2;

                t1          t2
1960       1.00000    13.00000
1961       2.00000    18.00000
1962       3.00000    23.00000
1963       4.00000    28.00000
```

Variables comportant une suite de termes constants

On peut toujours charger de telles variables avec `read` ou `load` en répétant les valeurs constantes. Ci-après deux exemples de comment effectuer ce travail plus efficacement:

```
freq n; smpl 1 5;
```

```
read x; 3*9 2*7;
smpl 1 3; y = 9;
smpl 4 5; y = 7;
smpl 1 5;
print x y;

                X            y
1         9.00000      9.00000
2         9.00000      9.00000
3         9.00000      9.00000
4         7.00000      7.00000
5         7.00000      7.00000
```

Création de variables (0/1) "dummy"

`dummy` *nom_variable* [*liste_noms_variables*] ;

Cette commande permet la création de variables artificielles logiques (*binaires*0/1) appélées "dummy's" ou variables muettes.

Exemple de création de variables muettes trimestrielles `Q1` à `Q4`:

```
freq q; smpl 1990:1 1991:2;
dummy;
options nwidth = 5, signif = 0; print Q1-Q4;

            Q1    Q2    Q3    Q4
 1960:1     1.        0.        0.        0.
 1960:2     0.        1.        0.        0.
 1960:3     0.        0.        1.        0.
 1960:4     0.        0.        0.        1.
 1961:1     1.        0.        0.        0.
 1961:2     0.        1.        0.        0.
```

TSP crée automatiquement les variables `Q1`, `Q2`, `Q3` et `Q4`. Notons avec la commande `print`, que TSP comprend la syntaxe qui désigne une liste de noms par le premier et le dernier nom de la liste reliés par un trait.

Exemple de construction d'une variable `DCE` qui caractérise le changement structurel intervenu au moment de la crise énergétique de 1973. On aura $\mathtt{DCE}_t = 0$ si $t < 1973$ et $\mathtt{DCE}_t = 1$ si $t \geq 1973$.

```
smpl 1960 1998;
trend t 1960;
DCE = (t >= 1973);

              DCE
1960      0.00000
```

```
  ⋮          ⋮
1972       0.00000
1973       1.00000
  ⋮          ⋮
1998       1.00000
```

Sélection d'un sous-ensemble d'observations

La commande `smpl` permet de sélectionner des observations en spécifiant des segments sous la forme d'une liste

```
smpl i j  k l  r s;
```

avec $i \leq j < k \leq l < r \leq s$. La sélection d'observations peut également se faire l'aide d'une variable logique (0/1) combinée avec les commandes `select` et `smplif`.

`select` *nom_variable*;

> La variable est soit une variable logique (0/1) ou interprétée comme telle (un élément est consideré comme "vrai" (1) si sa valeur est différente de zero et "faux" (0) sinon). La commande definit alors les segments d'intervalles d'observations pour lesquels la variable est "vraie".

Exemple de sélection d'un sous-ensemble de données avec une variable logique. On veut se restreindre aux observations pour lesquelles la variable `x` est strictement positive:

```
freq n; smpl 1 6;
load x; 3 -2 -1  8  4 -3;
D = (x > 0);
print x  D;
select D;      ?  select (x > 0) est équivalent
print  x;

Current sample:    1 to 6
                 X            D
1          3.00000      1.00000
2         -2.00000      0.00000
3         -1.00000      0.00000
4          8.00000      1.00000
5          4.00000      1.00000
6         -3.00000      0.00000
Current sample:  1 to 1, 4 to 5
                 X
1          3.00000
```

```
4          8.00000
5          4.00000
```

Pour redfinir l'intervalle d'observations original on peut exécuter la commande `select 1;` (création d'une variable logique avec tous les elements égaux à 1).

Si l'on desire imbriquer une procedure de selection (selection successive dans des sous-ensembles) on utilisera la commande `smplif`.

`smplif` *nom_variable*;

> Contrairement a la commande `select` cette commande slectionne les observations dfinis par le `smplif` précédent.

Dans l'exemple qui suit on sélectionne d'abord les éléments qui sont supérieurs à 3 et ensuite ceux qui sont inférieurs à 5 dans l'ensemble précédent.

```
freq n; smpl 1 6;
load x; 3 -2 -1  8  4 -3;
smplif (x > 3);
print x;
smplif (x < 5);
print x;

Current sample:    4 to 5
                    X
4          8.00000
5          4.00000
Current sample:    5 to 5
                    X
5          4.00000
```

L'exemple suivant montre comment il est possible de slectionner des observations en combinant les critères de selection sur trois variables `P`, `R` et `E`.

```
smplif (P > 0 & R > 0 & K < E);
```

Rappelons la définition des opérateurs logiques dans TSP:

&	et
|	ou
~	non

Equations dynamiques

Soit des variables définies par des équations dynamiques de l'une des deux formes suivantes:

$$\begin{aligned} y_t &= a_t y_{t-1} + b_t \qquad t = 2, \ldots, T \\ y_t &= a_t y_{t+1} + b_t \qquad t = T-1, \ldots, 1 \end{aligned}$$

Pour évaluer de telles variables, on peut utiliser la commande `genr`. Voici, à titre d'exemple, la création d'une variable définie comme $y_t = 2\,y_{t-1} + 5$, pour $t = 1996, \ldots, 1999$ avec $y_{1995} = 12.5$.

```
freq a; smpl 1995 1999;
set a = 2;
set b = 5;
set y(1995) = 12.5;
smpl 1996 1999;
genr y = a * y(-1) + b;
print y;

NOTE:  Dynamic GENR for Y
                    Y
1996          30.00000
1997          65.00000
1998         135.00000
1999         275.00000
```

La commande `genr` peut aussi être utilisée avec l'option `static` et `silent`.

Un autre exemple est le calcul de la valeur actualisée d'un cash flow. Soit $\texttt{CF}_t$, $t = 1999, \ldots, 2005$ le cash flow et $\texttt{rm}_t$ le taux du marché pour les périodes correspondantes. La relation qui définit les valeurs actualisées $\texttt{VA}_t$ s'écrit alors

$$\texttt{VA}_t = (\texttt{VA}_{t+1} + \texttt{CF}_{t+1})/(1 + \texttt{rm}_{t+1}) \qquad t = 2004, \ldots, 1999$$

et les commandes TSP correspondantes sont:

```
smpl 1999 2005;
load rm; 0.040 0.042 0.047 0.045 0.044 0.040 0.035;
load Cf;   800   800   900   950  1000  1500  2000;
Va = 0;
smpl 1999 2004;
genr  Va = ( Va(+1) + Cf(+1) ) / (1 + rm(+1));
print Va;

NOTE:  Reverse dynamic GENR for VA
                   VA
```

```
1999         6039.03027
2000         5492.66943
2001         4850.82520
2002         4119.11230
2003         3300.35303
2004         1932.36719
```

La classe particulière d'équations dynamiques

$$
\begin{aligned}
K_t &= (1-\delta)K_{t-1} + I_{t-1} \\
K_t &= (K_{t+1} - I_{t+1})/(1-\delta)
\end{aligned}
$$

avec K_t un stock de capital, I_t un flux d'investissements et δ un taux d'amortissement, peut être évaluée avec la commande `capitl`.

```
freq a; smpl 1995 1999;
load I; 620 725 533 835 642;
capitl (BENCHOBS=1997,BENCHVAL=15500) I  0.04  K;

                       K
1995        15485.02637
1996        15590.62500
1997        15500.00000
1998        15715.00000
1999        15728.40039
```

Illustrons encore comment la commande `genr` permet le calcul du cumul d'une série:

```
freq n; smpl 1 5;
trend x;
cum = x;
smpl 2 5;
genr (silent) cum = cum(-1) + x;
smpl 1 5; print x cum;

                   X           CUM
1            1.00000       1.00000
2            2.00000       3.00000
3            3.00000       6.00000
4            4.00000      10.00000
5            5.00000      15.00000
```

On vérifie que `CUM(5)` $= 5 \times 6/2 = 15$. Notons que l'option `silent` a supprimé les messages générés par `genr`. Cette option peut s'avérer utile dans des exercices de simulation de Monte Carlo où l'on peut répéter un grand nombre de fois la génération de telles variables.

Observations manquantes

Il est possible de travailler avec des données comportant des observations manquantes. Ces valeurs manquantes sont indiquées avec le symbole ".". Par la suite TSP signale la présence de ces valeurs manquantes, ou si nécessaire restreint l'intervalle d'observations avant d'effectuer une estimation.

```
smpl 61 65; read X; 13 .  16 17 20;
print X;

*** WARNING: Missing values for series ====>    X: 1
             X
1961      13.0
1962         .
1963      16.0
1964      17.0
1965      20.0

P = Z + X;

*** WARNING: Missing values for series ====>    X: 1

*** WARNING: Some elements of a series set to missing values due to
             missing values.    Number ====>   1

olsq X  c P0;

*** WARNING: Missing values for series ====>    X: 1

Dependent variable:  X
Current sample:    1961 to 1961, 1963 to 1965
Number of observations:    4
...
```

Normalisation d'une série

`normal` *nom_variable indice constante*;

Cette commande transforme une série x de la façon suivante:

$$x_i = \texttt{constante} \times \frac{x_i}{x_{\texttt{indice}}}$$

```
freq a; smpl 1987 1991;
read P; 127 128 132 138 145;
normal P 1990 100;
print P;

             P
1987       92.02
1988       92.75
1989       95.65
1990      100.00
1991      105.07
```

Autres commandes importantes

`divind`

Calcul de séries d'indices du type Laspeyres, Paasche, Fisher, etc.

`sama`

Désaisonnalisation des séries d'observation.

`sort`

Trie une série ou un vecteur.

`msd [(`*options*`)]` *liste_variables*`;`

Calcul des indices de position et de dispersion d'un ensemble de séries. La commande `msd` produit une table avec la moyenne, l'écart type, minimum, maximum, somme et variance d'une variable ou d'une liste de variables. Les options `corr cova moment` permettent d'obtenir également les matrices de corrélation, covariance et la matrice des moments non centrés. Soit

$$Z = [x\ y] \qquad \text{et} \qquad \overline{Z} = [x - \overline{x}\ y - \overline{y}]$$

alors

$$\texttt{Moment Matrix} = \frac{1}{n} Z'Z$$

$$\texttt{Covariance Matrix} = \frac{1}{n-1} \overline{Z}'\overline{Z}$$

et

$$\texttt{Correlation Matrix} = \frac{\mathrm{Cov}_{xy}}{\sigma_x \sigma_y}.$$

```
freq n; smpl 1 5;
load x; 41 46 50 55 60;
load y; 13 14 16 17 18;
msd (corr,cova,moment,print) x y;
```

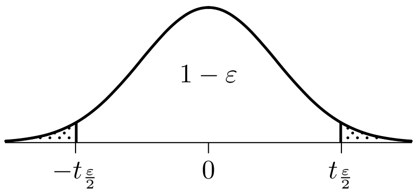
```
NUMBER OF OBSERVATIONS: 5
                              MOMENT MATRIX
             X         Y
X      2584.40
Y       798.40    246.80

          MEAN   STD DEV   MINIMUM   MAXIMUM       SUM  VARIANCE
X        50.40      7.44     41.00     60.00    252.00     55.30
Y        15.60      2.07     13.00     18.00     78.00      4.30
```

```
                      COVARIANCE MATRIX
             X        Y
X        55.30
Y        15.20     4.30
                      CORRELATION MATRIX
             X        Y
X          1.0
Y         0.99     1.00
```

Intégration des fonctions de densité

Lors de tests d'hypothèses notamment, il faut intégrer des fonctions de densité afin d'obtenir les probabilités liées à la réalisation de certains événements. Soit l'intégrale suivante d'une fonction de densité:



$$\int_{-t_{\frac{\varepsilon}{2}}}^{t_{\frac{\varepsilon}{2}}} f(t)\, dt = 1-\varepsilon \quad .$$

La commande `cdf` permet soit d'obtenir $t_{\frac{\varepsilon}{2}}$ pour ε donné, soit le contraire, c'est-à-dire $t_{\frac{\varepsilon}{2}}$ donné et ε calculé.

Soit une densité normale centré et réduite:

```
cdf (normal) 1.96;                    (on donne t_{ε/2} et on obtient ε)
  NORMAL Test Statistic:    1.960000   Two-tailed area:  .05000

cdf (normal,inv) .05;                 (on donne ε et on obtient t_{ε/2})
  NORMAL Critical Value:    1.959964   Two-tailed area:  .05000
```

Soit une densité de Student avec 6 degrés de liberté:

```
cdf (t,df=6) 2.1;                     (on donne t_{ε/2} et on obtient ε)
  T(6) Test Statistic:    2.100000   Two-tailed area:  .08048

cdf (t,inv,df=6) .05;                 (on donne ε et on obtient t_{ε/2})
```

```
T(6) Critical Value:    2.446912   Two-tailed area:  .05000
```

La commande `cdf` permet encore d'intégrer d'autres fonctions de densités comme celle du χ^2, de Fisher, de Dickey-Fuller, etc.

7

OPÉRATIONS AVEC MATRICES

Les objets manipulés en général avec TSP sont des séries temporelles définies par les commandes `freq` et `smpl`. TSP connaît cependant encore d'autres "classes" d'objets comme par exemple les matrices. Ce chapitre explique d'abord comment définir avec TSP des objets qui sont des matrices et ensuite on présente les commandes qui permettent d'effectuer des calculs avec les matrices.

7.1 CRÉATION DE MATRICES

Il existe plusieurs commandes qui permettent de créer ou de redéfinir une matrice. Les vecteurs sont à considérer comme des matrices, simplement que le nombre de lignes (ou colonnes) est égal à un.

`load (nrow=`*nl*`, ncol=`*nc*`, type=`*type_matrice*`)` *nom_matrice* `;`

Commande générale pour saisir un objet du type matrice (analogue au chargement d'une série temporelle). Les commandes `load` et `read` sont synonymes. Les paramètres *nl* et *nc* définissent la dimension de la matrice. Il est possible de spécifier une structure particulière avec le paramètre *type_matrice*. Les choix possibles sont:

`general`	Matrice sans structure particulière
`sym`	Matrice symétrique
`diag`	Matrice diagonale
`triang`	Matrice triangulaire

TSP lit les éléments de la matrice ligne après ligne. Ci-après différents exemples de lecture de matrices:

```
load (nrow=2,ncol=3,type=general)  A ;          (matrice générale)
1 2 3
4 5 6;
load (nrow=2,ncol=3,type=general)  A; 1 2 3 4 5 6;
load (nrow=2,type=sym) B;                       (matrice symétrique)
1
2 3;
load (nrow=2,type=sym) B;  1 2 3;
load (nrow=3,type=diag) D;                      (matrice diagonale)
1 2 3;
load (nrow=5,ncol=4) Z; 20*0;     (création d'une matrice avec zéros)
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

`mmake` *nom_matrice liste_series* ;

Transforme un ensemble de séries temporelles en une matrice. Les colonnes de la matrice correspondent aux séries temporelles. L'ensemble des observations qui correspondent aux lignes de la matrice sont définis avec la commande `smpl`.

```
freq a; smpl 1981 1985;
load  X1;  1  2  3  4  5;
load  X2;  6  7  8  9 10;
load  X3; 11 12 13 14 15;
smpl 1981 1984;
mmake M    X1-X3;
options nwidth=7, signif=2;
print M;

                  M
         1        2        3
   1    1.00     6.00    11.00
   2    2.00     7.00    12.00
   3    3.00     8.00    13.00
   4    4.00     9.00    14.00
```

`unmake` *nom_matrice liste_series* ;

Transforme les colonnes d'une matrice en séries temporelles (opération inverse de `mmake`).

Exemple de lecture d'une matrice `M` de dimension 4×5 et conversion des colonnes en cinq series trimestrielles `V1` à `V5` définis pour la période `79:4` à `80:3`.

```
freq n; smpl 1 4;
load (nrow=4, ncol=5,type=general) M;
10 14 18 22 26
11 15 19 23 27
12 16 20 24 28
13 17 21 25 29;
freq q; smpl 79:4  80:3;
unmake M   V1-V5;
options nwidth=6, signif=1;  print V1-V5;
```

```
 Current sample:    1979:4 to 1980:3
             V1      V2     V3     V4     V5
 1979:4      10.0   14.0   18.0   22.0   26.0
 1980:1      11.0   15.0   19.0   23.0   27.0
 1980:2      12.0   16.0   20.0   24.0   28.0
 1980:3      13.0   17.0   21.0   25.0   29.0
```

En utilisant alternativement les commandes `mmake` et `unmake` il est possible d'extraire des sousmatrices. A titre d'exemple montrons comment extraire une sousmatrice `S` de la matrice `M` définie ci-dessus. La sousmatrice `S` comporte les lignes 1, 2, 4 et les colonnes 1, 2, 4, 5 de la matrice `M`:

```
unmake M   V1-V5;
smpl  1 2  4 4;
mmake  S   V1 V2  V4 V5;
print  S;
```

```
                  S
   1    10.0   14.0   22.0   26.0
   2    11.0   15.0   23.0   27.0
   3    13.0   17.0   25.0   29.0
```

`copy` *nom_objet_original*   *nom_copie* ;

Effectue la copie d'un objet TSP. Dans l'exemple suivant on crée une matrice en copiant des matrices générées par la procédure `olsq`:

```
copy @coef  B;          (copie du vecteur des paramètres)
copy @VCOV  V;          (copie de la matrice des variances et covariances)
```

7.2 OPÉRATIONS MATRICIELLES

TSP permet d'effectuer une variété d'opérations matricielles telles que addition, soustraction, multiplication, inversion, calcul de valeurs propres, etc.

Toute opération matricielle **doit** être précédée par la commande `mat`.

`mat` *nom_matrice* = *expression_matricielle* ;

Commande générale pour les opérations matricielles. Cette commande vérifie, entre autres, la conformité des dimensions des matrices.
Attention: il ne faut jamais assigner le résultat d'une expression matricielle à une matrice de nom C. Il s'agit du nom réservé pour la variable constante dans la commande `olsq`.

Chargement de la matrice A et B et calcul du produit.

```
load (nrow=2,ncol=2,type=general) A; 1 2 3 4;
load (nrow=2,ncol=3) B; 5 6 7 8 9 0;
mat P = A*B;
```
(P contient le produit de A avec B)

Syntaxe pour les opérations matricielles

Dans le tableau qui suit, A et B sont des matrices dont la dimension est conforme aux opérations.

`A*B`	produit de A avec B
`A*s`	produit de A avec un scalaire s (`s*A`)
`A'`	transposée de A
`A'A`	produit de la transposée de A avec A
`A"`	inverse de A (A doit être carrée)
`A"B`	produit de l'inverse de A avec B
`A#B`	produit de Kronecker
`A%A`	produit élément par élément (produit de Hadamard)

```
mat D = 2*B;
mat Q = A'A;
mat H = A";
mat M = A"B;
print  D Q H M;
```

```
         D                Q              H                M

1  10.0 12.0 14.0   1  10.0          1  -2.0  1.0    1  -2.0 -3.0 -14.0
2  16.0 18.0  0.0   2  14.0 20.0     2   1.5 -0.5    2   3.5  4.5  10.5
```

Fonctions matricielles

`inv(M)`	matrice inverse de M
`diag(M)`	crée une matrice diagonale (diagonale de M)
`ident(n)`	crée une matrice d'identé d'ordre n
`eigval(M)`	calcule le vecteur des valeurs propres de M
`eigvec(M)`	calcule la matrice des vecteurs propres de M
`vec(M)`	crée un vecteur avec les éléments de M

Le résultat des fonctions suivantes est un scalaire (matrice 1×1). Il peut être utilisé dans une expression matricielle.

`det(M)`	déterminant de M	`min(M)`	plus petit élément de M
`tr(M)`	trace de M	`sum(M)`	somme des éléments de M
`nrow(M)`	nombre de lignes de M	`max(M)`	plus grand élément de M
`rank(M)`	rang de M	`ncol(M)`	nombre de colonnes de M

8

ESTIMATION NON-LINAIRE

Un modle linaire tel qu'il a t introduit en (3.1) est spcifi sans ambiguït par la donne de la variable endogne et la liste des variables exognes y intervenant (cf. commande `olsq`). Pour un modle non-linaire

$$y_t = h(x_t, \beta) + u_t \qquad t = 1, \ldots, T \tag{8.1}$$

il est indispensable de dcrire la fonction h en spcifiant la forme fonctionnelle particulire du modle et en indiquant les paramtres estimer.

On expliquera d'abord les commandes permettant de coder la (ou les) quation(s) du modle, puis on discutera la commande TSP pour estimer un modle non-linaire. Ces commandes sont illustres en estimant une fonction de consommation partir des observations tires d'une enqute sur des budgets de mnage. Les observations et les variables sont:

i	r_i	n_i	y_i
1	18'805	1.57	50
2	32'360	3.16	66
3	42'860	3.51	62
4	53'735	3.74	65
5	65'521	3.60	66
6	76'642	3.20	74
7	89'037	3.21	78
8	107'614	2.96	79

r_i Revenu moyen du mnage,
n_i Nombre moyen de personnes par mnage,
y_i Dpense par tte pour un bien donn.

8.1 CODAGE FORME FONCTIONNELLE

`frml` *nom_quation* *nom_variable* = *expression algbrique*;

Codage de la forme fonctionnelle d'une quation. Une quation ainsi code est par la suite referencee par son nom.

Soit alors la fonction de dpense

$$y_i = -\beta_0 \log(r_o) - \beta_1 \log(r_o) n_i + \beta_0 \log(r_i) + \beta_1 n_i \log(r_i)$$

et son codage sous le nom de "`dpense`":

```
frml depense  Y = -b0*log(r0) - b1*log(r0)*N
                    + b0*log(r) + b1*N*log(r);
```

Il est galement possible de donner l'quation sous forme implicite, c'est--dire en passant tous les termes d'un côt de l'quation. Soit l'quation prcdente sous forme implicite

$$y_i + \beta_0 \log(r_o) + \beta_1 \log(r_o) n_i - \beta_0 \log(r_i) - \beta_1 n_i \log(r_i) = 0$$

et son codage sous le nom de "`fi`":

```
frml  fi  Y + b0*log(r0) + b1*log(r0)*N - b0*log(r)
            - b1*N*log(r);
```

On remarque que l'on omet d'crire l'galit = 0.

`eqsub [(`*options*`)]` *nom_equation_principale liste quations substituer* ;

Effectue une ou plusieurs substitutions dans l'quation principale ce qui facilite le codage d'quations complexes. Soit les deux quations

$$\begin{aligned} y_t &= \beta_0 + \beta_1\ x_t + v_t \\ v_t &= \rho(y_{t-1} - (\beta_0 + \beta_1\ x_{t-1})) \end{aligned}$$

alors si l'on veut substituer la deuxime quation dans la premire on procde comme:

```
frml eq_y  y = b0 + b1*x + v;
frml eq_v  v = rho*(y(-1)-(b0+b1*x(-1)));
eqsub eq_y eq_v;
```

Une alternative consiste crire:

```
eqsub(NAME=eq_ar1,PRINT) eq_y eq_v;
```

ce qui construit une nouvelle quation `eq_ar1` et imprime le rsultat.

```
FRML EQ_AR1 Y = B0 + B1*X + RHO*(Y(-1) - (B0 + B1*X(-1)))
```

`param` *nom_paramtre* `[`*valeur initiale*`];`

Dfinition des paramtres dans une quation code avec la commande `frml`. Cette commande identifie les noms dans l'quation qui correspondent aux paramtres estimer. Il est possible et indiqu de donner ces paramtres des valeurs initiales pour l'estimation. En absence d'indication de valeur initiale, le paramtre est initialis zro.

Pour le cas de notre fonction de dpense les paramtres sont β_0, β_1 et r_0:

`param b0 b1 r0;` (sans indication de valeur initiale)

Les paramtres peuvent aussi tre dfinis dans des commandes successives:

```
param b0;
param b1;
param r0 1000;
```
(valeur initiale pour r_o, sinon $\log(r_o) \rightarrow -\infty$)

`const` *nom_constante valeur* `;`

Dfinition de variables scalaires dans une quation codifie avec la commande `frml`. Cette commande identifie les noms dans l'quation qui correspondent des variables scalaires constantes et leur assigne une valeur. Les constantes ne seront pas changes par l'estimation.

`const b0 20;` (b_o sera considr comme une constante valant 20)

8.2 ESTIMATION AVEC `LSQ`

`lsq [(`*options*`)]` *liste des quations* `;`

Produit des estimateurs des moindres carrs ou distance minimale des paramtres d'une ou plusieurs quations non-linaires. La liste des quations peut tre constitue d'une ou plusieurs quations. Si l'quation est linaire, `lsq` imprime un message avant de procder l'estimation.

La commande `lsq` comporte de nombreuses options. Par la suite on ne discutera que le cas simple d'une estimation portant sur les paramtres d'une seule quation. Procdons l'estimation de la fonction de dpense dfinie plus haut:

```
param b0 b1;
param r0 1000;
lsq depense;
```

```
                                 STARTING VALUES
                      B0             R0             B1
VALUE              0.00000    1000.00000        0.00000
F= 5.2604   FNEW= 2.1776   ISQZ= 0 STEP= 1.0000   CRIT=  4.9895
F= 2.1776   FNEW= 2.1738   ISQZ= 0 STEP= 1.0000   CRIT= 0.40467E-01
F= 2.1738   FNEW= 2.1734   ISQZ= 0 STEP= 1.0000   CRIT= 0.47789E-02
F= 2.1734   FNEW= 2.1734   ISQZ= 0 STEP= 1.0000   CRIT= 0.68715E-07
CONVERGENCE ACHIEVED AFTER   4 ITERATIONS
    8 FUNCTION EVALUATIONS.

                Log of Likelihood Function =    -20.4206
                    Number of Observations =       8
                         Standard
Parameter  Estimate        Error       t-statistic   P-value
B0         17.6584       4.95207       3.56587       [.000]
R0         857.626       650.166       1.31909       [.187]
B1         -.420246      .810022       -.518808      [.604]
Standard Errors computed from quadratic form of analytic first
derivatives (Gauss)
                              Equation    DEPENSE
                              ===================
Dependent variable:  Y
      Mean of dep. var. = 67.5000       R-squared = .877819
  Std. dev. of dep. var. = 9.50188  Adj. R-squared = .828946
Sum of squared residuals = 77.2245    LM het. test = 1.38862 [.239]
   Variance of residuals = 15.4449   Durbin-Watson = 2.41291 [<.868]
Std. error of regression = 3.93000
```

Description de l'output

Les rsultats imprims par `lsq` peuvent tre regroups en quatre points selon le type d'information qu'ils contiennent. Ces points sont les suivants:

- Valeurs initiales pour les paramtres (zro par dfaut) et valeurs des constantes lorsqu'elles ont t dfinies.
- Pour chaque itration on a des informations sur la valeur:
  - de la fonction objectif au dbut (`F`) et la fin de l'itration (FNEW),
  - des paramtres `ISQZ` et `STEP` qui dfinissent les pas vers la solution,
  - de la fonction critre d'arrt `CRIT` (devrait rapidement voluer vers zro).
- Message indiquant si la convergence a eu lieu ou non.
- Valeur des paramtres estims et statistiques associes.

Considrations pratiques

La solution de problmes de minimisation de fonctions non-linaires peut tre sensible aux valeurs initiales des paramtres estimer. Ceci peut tre dû, d'une part, l'existence de plusieurs minimums locaux, et d'autre part, des problmes numriques, lis au fait que la prcision d'un ordinateur est fini.

Ci-aprs quelques conseils qui, le cas chant, peuvent contribuer obtenir des rsultats dans des situations où l'estimation s'avre difficile. Il est conseill de:

- exprimenter des valeurs initiales diffrentes pour les divers paramtres;
- choisir des valeurs initiales proches de la solution, en procdant ventuellement des estimations d'un modle plus simple d'abord;
- procder l'estimation d'un sous-ensemble de paramtres en gardant les autres paramtres constants.

Pour illustrer cette dmarche conseille en cas de problmes, considrons la situation où la fonction de dpense a t estime en choisissant 1 comme valeur initiale pour le paramtre r_0. On obtient alors le rsultat suivant:

```
    param b0 0;
    param b1 0;
    param r0 1;
    lsq depense;
                                 STARTING VALUES
                    B0            R0            B1
VALUE            0.00000       1.00000       0.00000
F= 5.2604   FNEW= 2.7323   ISQZ= 0 STEP=  1.0000     CRIT=  4.9682
F= 2.7323   FNEW= 2.7206   ISQZ= 7 STEP= 0.27954E-01 CRIT=  3.2266
F= 2.7206   FNEW= 2.7152   ISQZ= 7 STEP= 0.31905E-01 CRIT=  3.1906
F= 2.7152   FNEW= 2.7098   ISQZ= 8 STEP= 0.17290E-01 CRIT=  3.1779
F= 2.7098   FNEW= 2.7275   ISQZ=11 STEP= 0.28775E-01 CRIT=  3.1622

                    B0            R0            B1
ESTIMATE         5.82028       2.78759       0.36575
CHANGES          9.10384      36.95672      0.041079

FAILURE TO IMPROVE OBJECTIVE FUNCTION AFTER 5 ITERATIONS (MAXSQZ)
   42 FUNCTION EVALUATIONS.
                Log of Likelihood Function =    -24.7125
                    Number of Observations =      8
                         Standard
Parameter  Estimate        Error       t-statistic   P-value
```

```
 B0          5.82028        5.49562        1.05908        [.290]
 R0          2.78759        21.4452        .129986        [.897]
 B1          .365752        .413592        .884330        [.377]
```

On constate que l'estimation ne converge pas. On fixe alors la valeur du paramtre β_o 20. Cette valeur pourraît tre inspire par des rsultats obtenus lors de l'estimation de modles semblables. On procde alors une nouvelle estimation:

```
const b0 20;
param r0  1;
lsq depense;
                                    CONSTANTS:
                     B0
 VALUE          20.00000
                                  STARTING VALUES
                     R0             B1
 VALUE           1.00000        0.36575
 F= 6.1326   FNEW= 5.7094   ISQZ= 0 STEP= 1.0000   CRIT=  5.9971
 F= 5.7094   FNEW= 5.2051   ISQZ= 0 STEP= 1.0000   CRIT=  5.9941
 F= 5.2051   FNEW= 4.5670   ISQZ= 0 STEP= 1.0000   CRIT=  5.9848
 F= 4.5670   FNEW= 3.6963   ISQZ= 0 STEP= 1.0000   CRIT=  5.9473
 F= 3.6963   FNEW= 2.5779   ISQZ= 0 STEP= 1.0000   CRIT=  5.7026
 F= 2.5779   FNEW= 2.1963   ISQZ= 0 STEP= 1.0000   CRIT=  3.2221
 F= 2.1963   FNEW= 2.1928   ISQZ= 0 STEP= 1.0000   CRIT= 0.42414E-01
 F= 2.1928   FNEW= 2.1928   ISQZ= 0 STEP= 1.0000   CRIT= 0.18118E-05
 CONVERGENCE ACHIEVED AFTER   8 ITERATIONS
    16 FUNCTION EVALUATIONS.
                 Log of Likelihood Function =    -20.5758
                     Number of Observations =      8
                          Standard
 Parameter  Estimate        Error       t-statistic   P-value
 R0         1144.38        303.766       3.76731       [.000]
 B1         -.772773       .370810       -2.08401      [.037]
```

L'estimation de ce modle contraint converge. Notons que TSP utilise comme valeur initiale pour le paramtre β_1 le rsultat de l'estimation prcdente.

On restime le modle complet avec comme valeurs initiales les estimations obtenues pour $\beta_0 = 20$. (On rappelle qu'au cours d'une mme sance de travail les paramtres restent dfinis d'une estimation l'autre).

```
param b0;
lsq depense;
                                  STARTING VALUES
                     B0             R0             B1
```

```
VALUE         20.00000    1144.38093      -0.77277
F= 2.1928  FNEW= 2.1790   ISQZ= 0 STEP= 1.0000   CRIT= 0.17900
F= 2.1790  FNEW= 2.1734   ISQZ= 0 STEP= 1.0000   CRIT= 0.56195E-01
F= 2.1734  FNEW= 2.1734   ISQZ= 0 STEP= 1.0000   CRIT= 0.47763E-07
CONVERGENCE ACHIEVED AFTER   3 ITERATIONS
    6 FUNCTION EVALUATIONS.
                Log of Likelihood Function =    -20.4206
                    Number of Observations =      8
                         Standard
Parameter  Estimate        Error       t-statistic   P-value
B0         17.6583       4.95205       3.56586       [.000]
R0         857.612       650.161       1.31908       [.187]
B1         -.420225      .810016       -.518785      [.604]
```

9

PROGRAMMATION DE PROBLÈMES NON-STANDARD

Certains problèmes particuliers ne peuvent pas être résolus directement à l'aide de procédures standard. Il s'avère alors nécessaire de pouvoir programmer la façon dont les différentes commandes doivent s'enchaîner. TSP contient un certain nombre d'instructions qui lui confèrent un caractère de langage de programmation. Il s'agit d'un langage, dit de haut niveau, c'est-à-dire qu'il est proche du langage de l'utilisateur. Cependant son exécution peut s'avérer lente pour des problèmes nécessitant des calculs intenses.

9.1 NOTIONS DE PROGRAMMATION

Un ordinateur est capable d'effectuer un certain nombre d'opérations élémentaires telles que l'addition, la soustraction, l'affectation, la comparaison, etc. Un *programme* orchestre le déroulement de ces opérations dans le temps. Le programme lui-même est un objet statique, mais lors de son exécution il évolue de façon dynamique en fonction de l'état des variables.

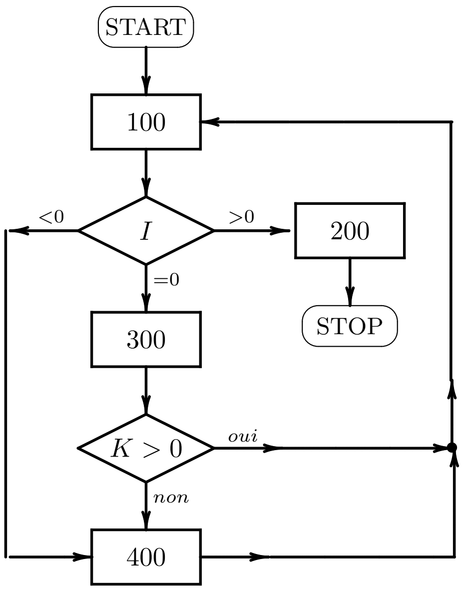
Il se pose alors le problème de savoir comment concevoir un programme afin d'avoir une idée aussi claire que possible des situations dynamiques qu'il engendre lors de son exécution. L'objectif de la programmation est donc une lisibilité aussi claire que possible du programme, afin de pouvoir le communiquer entre "programmeurs" et surtout pour pouvoir fournir une démonstration plus ou moins rigoureuse de sa validité (faire la preuve qu'il fournira les résultats désirés).

On verra comment atteindre au mieux ces objectifs grâce à des techniques adéquates de programmation.

Note historique

A l'origine de la programmation les enchaînements étaient contrôlés avec des instructions de branchement (`GOTO`, `IF` avec branchement, etc.). Ci-après le schéma d'un programme qui a recours à ce type d'instructions de branchement:

```
    START
100 continue
    .
    .
    .
          Calcul valeur de I
    .
    .
    .
    if (I) 400, 300, 200
300 continue
    .
    .
    .
          Calcul valeur de K
    .
    .
    .
    if (K > 0) GOTO 100
400 continue
    .
    .
    .
    GOTO 100
200 continue
    .
    .
    .
    STOP
```



On constate aisément qu'il est difficile d'imaginer les différents états dynamiques d'un tel programme. Les schémas fléchés, dits organigrammes, qui accompagnaient généralement ce type de programmes, ne permettaient d'ailleurs pas de contourner cette difficulté.

La programmation structurée

Les difficultés que l'on vient de mentionner ont motivé un courant de pensée en faveur d'un autre “style” de programmation. On trouve à l'origine de ces idées notamment Dijkstra (1968) avec son article “GOTO Statement Considered Harmful", et Wilkes (1968) avec "The Outer and the Inner Syntax of a Programming Language". Ces deux articles lancèrent une véritable polémique qui mettait en cause l'utilisation des instructions de branchement dans les langages de programmation.

Ceci a conduit à l'élaboration d'un nouveau style de programmation, qui est la *programmation structurée*, dans laquelle les branchements ont été abandonnés. Les seules structures de contrôle qui existent dans la programmation structurée sont:

- enchaînement,
- répétition,
- choix.

Enchaînement

L'enchaînement consiste en une simple succession d'un nombre quelconque d'instructions d'affectation, de modification, de lecture, d'écriture:

```
lire y
x = sqrt(y)
  ⋮
écrire z
```

Répétition

La répétition d'un ensemble (block) d'instructions, appelée aussi boucle, peut prendre deux formes:

`pour` $i \in I$ `repeter`	`tant que` *condition vraie* `repeter`
⋮ *(instructions)*	⋮ *(instructions)*
`fin`	`fin`

Pour la première forme, le nombre de répétitions pour la première forme est défini par le nombre d'éléments de l'ensemble I, alors que pour la seconde forme, la répétition se fait à l'infini si la condition reste vraie.

Choix

Les instructions qui permettent d'opérer un choix parmi différents blocks d'instructions possibles sont les suivantes:

```
si  condition 1 vraie  faire
    ⋮ (instructions)
sinon si  condition 2 vraie  faire
    ⋮ (instructions)
sinon si ...
 ⋮
sinon si  condition n vraie  faire
    ⋮ (instructions)
sinon
    ⋮ (instructions)
fin
```

On exécute les instructions qui suivent la première condition qui est vraie et on avance jusqu'à la fin. Si aucune condition n'est vraie, on exécute les instructions qui suivent `sinon`.

Tout programme est alors composé d'une succession de ces trois structures qui sont exécutées l'une après l'autre. Par opposition à un programme contenant des instructions de branchement, un programme conçu avec les trois éléments de la programmation structurée, permettra une lecture hiérarchique de haut en bas ce qui facilite sa maîtrise intellectuelle. Le recours aux organigrammes a d'ailleurs été abandonné.

9.2 PROGRAMMATION AVEC TSP

Classes de variables

Rappelons que TSP comporte trois classes de variables qui sont `SERIES`, `MATRIX` et `SCALAR`. D'autre part chaque exécution de commande TSP définit des variables dont le nom est caractérisé par le fait que le premier caractère est constitué par "@" ou "%". La commande `show all` renseigne sur les noms des variables définies. Toutes ces variables peuvent être utilisées dans les instructions d'un programme TSP.

Affectation

La syntaxe à utiliser pour affecter une valeur à une variable dépend de la classe à laquelle appartient la variable. La syntaxe pour les variables de la classe `SERIES` et `MATRIX` a déjà été présentée dans les chapitres précédents.

nom_série `=` *expression algébrique avec séries* `;`

`mat` *nom_matrice* `=` *expression algébrique avec matrices* `;`

L'affectation de valeur à une variable de la classe `SCALAR` se programme:

`set` *nom_scalaire* `=` *expression algébrique avec scalaires* `;`

L'expression algébrique qui définit la valeur ne doit faire intervenir que des variables scalaires ou des éléments d'une série (e.g. `x(i)`).

Exemple de calcul des deux bornes qui définissent un intervalle de confiance

$$P\{\hat{\beta}_2 - t_{\frac{\varepsilon}{2}}\hat{\sigma}_{\beta_2} < \beta_2 < \hat{\beta}_2 + t_{\frac{\varepsilon}{2}}\hat{\sigma}_{\beta_2}\} = 1 - \varepsilon$$

pour le deuxième paramètre d'une estimation linéaire:

```
set dl = @nob - @ncoef;                  (définir le degré de liberté)
cdf(t,inv,df=dl) .05 teps2;              (calcul de t_{ε/2})
set binf = @coef(2) - teps2*@ses(2);     (borne inf. de l'intervalle)
set bsup = @coef(2) + teps2*@ses(2);     (borne sup. de l'intervalle)
```

Répétition

L'exécution répétée d'un ensemble d'instructions se programme comme:

`do` *indice* `=` *debut, fin* `[,` *incrément*`];`

$\vdots$ *instructions*

`enddo;`

En omettant l'indice l'ensemble des instructions ne s'exécute qu'une seule fois. On utilise cette construction pour regrouper un ensemble d'instructions en un block.

```
do;
    ⋮       instructions
enddo;
```

Exemple d'étude de l'évolution du $\overline{R}^2$ d'un modèle $y_t = \beta_0 + \beta_1 x_t + u_t$, $t = 1980, \ldots, 1990$, en fonction de l'intervalle d'estimation que l'on désire réduire graduellement de 1980–1990 à 1985–1990:

```
supres coef smpl;
do i = 1980 1985;
   smpl   i 1990;
   olsq (silent) y  c x;
   set r2a(i) = @arsq;                (on récupère la valeur de R̄²)
enddo;
smpl 1980 1985;
print r2a;                            (impression de l'évolution du R̄²)
```

La commande `supres` avec les arguments `coef` et `smpl` supprime l'impression des coefficients et de l'intervalle des observations pour chaque estimation. L'option `silent` pour la commande `olsq` supprime l'impression de toutes les statistiques. Ainsi aucun résultat ne sera imprimé pour les estimations successives.

Dans cet exemple on a indenté les instructions d'un même block. TSP n'est pas sensible à cette indentation. Nous recommandons cependant vivement ce style d'écriture qui améliore significativement la lecture (et la compréhension) d'un programme.

On voit dans l'exemple précédent que l'instruction

```
do i = 1980 1985;
```

génère l'ensemble des entiers {1980 1981 1982 1983 1984 1985} et que par la suite TSP affecte successivement à la variable `i` la valeur des éléments de cet ensemble.

L'ensemble qui définit les répétitions peut également être constitué par des éléments qui représentent des noms. Dans ce cas la syntaxe se présente comme:

```
dot  liste;
   ⋮      instructions •
enddot;
```

On remarque que `do` a été modifié en `dot` et que l'élément générique à remplacer dans les instructions est maintenant indiqué par un point.

A titre d'exemple considérons les séries temporelles `XA`, `XB`, `XC` et `XD` que l'on désire successivement régresser sur la série `Y`:

```
dot  A B C D;
   olsq Y  c X. ;
enddot;
```

Soit encore les variables `X1`, `X2`, ..., `X100`, alors on peut programmer:

```
dot  1-100;
   olsq Y  c X. ;
enddot;
```

Dans cet exemple TSP exécutera successivement les instructions `olsq Y c X1`, `olsq Y c X2`, ...

Ou encore deux variables `Y` et `X` pour chacun des pays `UK`, `IT`, `DE` et `FR`:

```
list CEE  UK IT DE FR;
dot  CEE;
   olsq Y.  c X. ;
enddot;
```

Pour ce dernier exemple l'ensemble des noms a été défini avec la commande `list` ce qui est équivalent à la façon dont on a procédé auparavant.

Mentionnons encore qu'il est possible d'imbriquer des instructions de répétition.

Choix

Pour opérer un choix parmi deux blocks d'instructions possibles on utilise les commandes `if`, `then` et `else`.

```
if  expression_scalaire ; then;
   do;
         ⋮      instructions 1
   enddo;
else;
   do;
         ⋮      instructions 2
   enddo;
```

Si l'expression scalaire est "vraie" (> 0) alors l'ensemble d'instructions 1 est exécuté et l'ensemble d'instructions 2 n'est pas exécuté. Si l'expression scalaire est "fausse" (≤ 0) on obtient l'effet contraire. Si l'expression est une proposition logique (p. ex. `test < 2.0`), TSP affecte la valeur 1 à celle-ci si elle est vraie et 0 si elle est fausse.

Exemple de changement de méthode d'estimation si la statistique DW de Durbin-Watson vérifie $DW < 1.5$ où $DW > 2.5$:

```
olsq y c x1 x2;
if abs(@dw-2) < .5; then;
   do;
      ar1 y c x1 x2;
   enddo;
```

Exemple de changement de spécification si le paramètre associé la variable `x2` est inférieur ou égal a 1:

```
olsq y c x1 x2;
if @coef(3) > 1; then;
   do;
      copy @coef b;
   enddo;
else;
   do;
      olsq y c x1 x3;
   enddo;
```

Sous-programmes

Dans un programme on est souvent amené à exécuter une même tâche soit sur les mêmes variables ou sur des variables différentes. Il se peut aussi qu'une telle tâche soit commune à plusieurs programmes. Dans une telle situation il est possible d'éviter la répétition du même (sous-)programme en invoquant simplement (avec une seule instruction) un sous-programme qui exécutera cette tâche pour des arguments qui peuvent prendre des valeurs différentes.

Dans TSP les sous-programmes sont appelées procédures qui seront définies avec les instructions `proc` et `endproc`. La structure d'une procédure est la suivante:

```
proc  nom_procédure [ arguments] ;
        ⋮        instructions
endproc;
```

Exemple d'une procédure `calcvp` qui calcule la variation en pourcent d'une série:

```
proc calcvp  x xvp;      ( x: argument d'entrée,  xvp: argument de sortie)
   copy @smpl smpl_original;
   set debut = @smpl(1) + 1;
   smpl debut @smpl(2);
   xvp = 100*(x - x(-1)) / x(-1);
   smpl smpl_original;
endproc;
```

Le calcul de la variation en pourcent d'une série quelconque `y` se fera alors en appelant la procédure `calcvp` comme:

```
calcvp y yvp;
```

La procédure doit avoir été lue précédemment dans le programme TSP.

10

MODÈLES À ÉQUATIONS MULTIPLES

La construction, résolution et utilisation de modèles à équations multiples est un processus qui comporte notamment les étapes suivantes:

- Définition du modèle:
  - Codage des équations (`frml`, `ident`, `form`, `eqsub`);
  - Estimation des équations (`lsq`, `olsq`, `ar1`, `ml`, etc.);
  - Définition des listes d'équations et variables endogènes (`list`);
- Analyse de la structure causale (logique) du modèle (`model`);
- Choix de l'algorithme de résolution (`siml`, `solve`);
- Validation du modèle (analyse des propriétés quantitatives);
- Préparation des scénarios pour les variables exogènes;
- Simulations.

Avant de présenter le déroulement de ces différentes étapes avec TSP, on donne une introduction générale sur le processus de construction des modèles à équations multiples dans la section qui suit.

10.1 PROCESSUS DE CONSTRUCTION

Qu'est-ce qu'un modèle ? On peut formaliser un modèle comme un couple de deux ensembles, soit:

$$\text{Modèle:} \qquad M = (\mathcal{X}, \mathcal{H})$$

avec $\mathcal{X}$ un ensemble de variables et $\mathcal{H}$ un ensemble d'hypothèses qui définissent les relations existant entre les variables.

La définition de l'ensemble des hypothèses se fait en général par étapes successives. On commence par spécifier quelles variables interviennent dans quelles équations et on décide quelles sont les variables endogènes. Cette information définit la structure logique du modèle que l'on notera $\mathcal{H}_0$.

Dans une deuxième étape, on peut définir $\mathcal{H}_1$ en augmentant $\mathcal{H}_0$ par la définition du signe des relations entre les variables. Cette démarche, qui consiste à renforcer successivement les hypothèses étape par étape, conduira finalement à un modèle quantifié.

L'intérêt de cette démarche consiste dans le fait que pour chaque structure $\mathcal{H}_i$ on peut mettre en évidence des propriétés $\mathcal{P}_i$ qui restent invariantes pour les étapes successives. On peut formaliser cette démarche comme

$$\begin{array}{l} \mathcal{H}_0 \subset \mathcal{H}_1 \subset \cdots \subset \mathcal{H}_p \\ \mathcal{P}_0 \subset \mathcal{P}_1 \subset \cdots \subset \mathcal{P}_p \end{array}$$

Donnons $\mathcal{H}_0$ qui définit la structure logique d'un modèle économique:

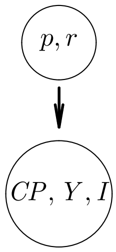
$$M = (\mathcal{X}, \mathcal{H}_0) \quad \Leftrightarrow \quad \left\{ \begin{array}{lll} CP = f_1(Y_{-1}, p) & & \\ \;I \;\;= f_2(Y, r) & & \\ \;Y \;= CP + I & & (\mathrm{id}_1\text{:} \quad CP + I - Y = \mathit{0}) \end{array} \right.$$

L'ensemble des variables est formé par $\mathcal{X} = \{CP, Y, p, I, r\}$ avec CP la consommation privée, Y la production, p un indice des prix, I les investissments et r le taux d'intérêt. $\mathcal{H}_0$ se limite à décrire quelles variables interviennent dans les relations.

Les fonctions f_1 et f_2 représentent des équations de comportement et id_1 est une identité comptable.

Donnons la partition $\mathcal{X} = \mathcal{Y} \cup \mathcal{Z}$ de l'ensemble des variables $\mathcal{X}$ en variables endogènes $\mathcal{Y}$, celles que le modèle doit expliquer, et variables exogènes $\mathcal{Z}$, celles qui sont données. Pour cet exemple on a $\mathcal{Y} = \{CP, Y, I\}$ et $\mathcal{Z} = \{p, r\}$.

Ceci définit une propriété très générale de la structure du modèle qui est illustré avec la figure suivante:



Variables exogènes $\mathcal{Z}$

Variables endogènes $\mathcal{Y}$

Etant donné une quantification des équations f_1 et f_2 et des valeurs pour les variables exogènes p et r, le modèle va donner des réponses numériques pour les variables endogènes CP, Y et I.

On verra cependant lors de l'analyse de la structure logique du modèle que certaines réponses que le modèle fournira ne dépendent pas de la quantification du modèle mais de sa structure.

10.2 DÉFINITION DU MODÈLE

Avec TSP, procédons à la quantification du modèle introduit plus haut. Pour cet exercice on utilisera les données suivantes:

Année	CP	I	p	r
1989	40	4.7	65	0.10
1990	45	3.2	70	0.15
1991	49	2.6	80	0.16
1992	50	5.0	105	0.04
1993	53	3.5	100	0.10
1994	55	4.7	95	0.05
1995	56	5.1	115	0.02
1996	57	4.7	90	0.05
1997	59	4.8	80	0.04
1998	55	5.1	120	0.01

Codage des équations (`form`, `frml`, `ident`, `eqsub`)

`form` *nom_équation*;

Le résultat d'une estimation d'une équation linéaire effectuée soit avec `olsq`, `inst`, `liml` ou `ar1` peut être codé automatiquement en faisant succéder la commande `form` à la commande d'estimation.

```
smpl 1989 1998;
DY = Y - Y(-1);
smpl 1990 1998;
olsq CP  c Y(-1) p;   form eq_CP;
olsq I   c DY r;      form eq_I;
print  eq_CP  eq_I;
```

```
EQUATION: EQ_CP
     FRML EQ_CP CP = (18.4662+ 0.7293* Y(-1))+ (-0.06260)* P
EQUATION: EQ_I
     FRML EQ_I   I = (5.4517+ 0.05803* DY)+ (-18.1597)* R
```

`frml` *nom_équation* *nom_variable* = *expression algébrique*;

Cette commande, déjà introduite pour l'estimation d'une équation non-linéaire, permet de coder la forme fonctionnelle d'une équation. A titre d'exemple, essayons une spécification non-linéaire pour la fonction de consommation:

```
frml  eqnl_CP  CP = b0 * Y(-1)**b1 + b2*p;
param b0  0;
param b1  1;
param b2  0.1;
lsq  eqnl_CP;
```

Codons les valeurs obtenues pour les estimations des paramètres et imprimons l'équation:

```
const  b0  3.89;
const  b1   .68;
const  b2  -.065;
frml  eqnl_CP  CP = b0 * Y(-1)**b1 + b2*p;
print eqnl_CP;
```

```
EQUATION: EQNL_CP
     FRML EQNL_CP CP = B0* Y(-1)** B1 + B2* P
```

Remarquons qu'il eût été également possible de coder les équations `eq_CP` et `eq_I` avec une commande `frml`, soit en définissant les paramètres comme constantes, soit en les spécifiant:

```
frml eq_CP CP = 18.4662 + 0.7293*Y(-1) - 0.06260*P;
```

ou bien

```
const a0 18.4662;
const a1  0.7293;
const a2 -0.0626;
frml eq_CP   CP = a0 + a1*Y(-1) - a2*P;
```

`ident` *nom_identité* ;

Codage d'identités et d'équations de définition. Il s'agit d'équations particulières pour lesquelles on n'estime pas de paramètres.

```
ident id_1   Y = CP + I;
ident aux_1  DY = Y - Y(-1);
```

Pour la résolution de modèles, il est indifférent qu'une identité soit codée avec la commande `ident` ou `frml`.

`eqsub` *nom_équation nom_équation_à_substituer*;

Cette commande permet de substituer des variables dans une équation. Les variables à substituer doivent être définies par une équation. Il est permis de substituer plusieurs variables avec une même commande.

Pour notre modèle, nous allons substituer la variable auxiliaire `DY`, étant donné qu'elle n'apporte pas d'information supplémentaire.

```
eqsub  eq_I  aux_1;
print  eq_I;
```

```
EQUATION: EQ_I
     FRML EQ_I I = (5.4517+ 0.05803* (Y- Y(-1)))+ (-18.1597)* R
```

`list` *nom_liste liste de noms* ;

Cette commande permet la définition d'un ensemble d'éléments constitué par des noms qui peuvent correspondre à toute variable TSP. On utilisera cette commande pour définir les ensembles d'équations et variables qui forment le modèle:

```
list  eq_mod1   eq_I  eq_CP    id_1;
list  eq_mod2   eq_I  eqnl_CP  id_1;
?
list  var_mod   CP Y I;
```

Les listes `eq_mod1` et `eq_mod2` définissent deux variantes pour le modèle.

10.3 STRUCTURE LOGIQUE DU MODÈLE

L'analyse de la structure logique du modèle n'est nécessaire que lorsqu'on choisit la procédure `solve` (voir ci-après) pour la résolution du modèle. Il est néanmoins vivement recommandé de toujours procéder à cette analyse, étant donné qu'elle renseigne sur la logique du modèle et peut, de ce fait, mettre en évidence des incohérences, qui se seraient introduites lors du processus de construction.

Rappelons qu'analyser la structure logique consiste à étudier les cheminements qui existent dans un modèle, dans le but:

- d'identifier les ensembles des variables qui sont interdépendantes;
- de mettre en évidence comment ces ensembles de variables interdépendantes sont reliés entre eux.

Pour pouvoir procéder à l'analyse de la structure causale avec TSP, il faut que les équations du modèle soient normalisées, c'est-à-dire qu'elles soient écrites comme

$$y_i = f_i(y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_n, z) \qquad i = 1, \dots, n$$

ce qui signifie que chaque équation explique une variable endogène différente.

Considérons alors la linéarisation du modèle

$$y = B_0 y + B_1 y_{-1} + \Gamma_0 x \tag{10.1}$$

avec B_0 la matrice jacobienne, B_1 la matrice des dérivées par rapport aux variables endogènes retardées et Γ_0 la matrice des dérivées par rapport aux variables exogènes. Pour notre exemple les matrices d'incidence des matrices B_0, B_1 et Γ_0 s'écrivent:

$$B_0 = \begin{array}{r} \\ \texttt{eq\_I} \\ \texttt{eq\_Cp} \\ \texttt{id\_1} \end{array} \begin{array}{c} \begin{array}{ccc} I & CP & Y \end{array} \\ \begin{bmatrix} 1 & . & 1 \\ . & 1 & . \\ 1 & 1 & 1 \end{bmatrix} \end{array} \qquad B_1 = \begin{array}{r} \\ \texttt{eq\_I} \\ \texttt{eq\_Cp} \\ \texttt{id\_1} \end{array} \begin{array}{c} Y_{-1} \\ \begin{bmatrix} 1 \\ 1 \\ . \end{bmatrix} \end{array} \qquad \Gamma_0 = \begin{array}{r} \\ \texttt{eq\_I} \\ \texttt{eq\_Cp} \\ \texttt{id\_1} \end{array} \begin{array}{c} \begin{array}{ccc} c & p & r \end{array} \\ \begin{bmatrix} 1 & . & 1 \\ 1 & 1 & . \\ . & . & . \end{bmatrix} \end{array}$$

L'analyse de la structure logique du modèle avec TSP consiste à réordonner la matrice jacobienne B_0 de sorte qu'elle devienne bloc-triangulaire. Les blocs de la diagonale forment des matrices indécomposables qui correspondent aux ensembles de variables interdépendantes.

`model` *liste_équations liste_variables nom_modèle*;

Cette commande vérifie si les équations sont normalisées, ordonne les équations, imprime la matrice d'incidence de la matrice jacobienne ordonnée et conserve l'information sur l'ordre des équations.

```
    model eq_mod1 var_mod  model1;

Block structure of MODEL1
Blk Eq# Equation Dep.Var. 123
 1R   1 EQ_CP    CP       X
 2S   2 ID_1     Y        XXX
 2S   3 EQ_I     I         XX
```

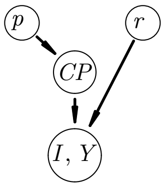
$$
\begin{array}{cc}
 & \begin{array}{ccc} CP & Y & I \end{array} \\
B_0 = \begin{array}{r} \texttt{eq\_Cp} \\ \texttt{id\_1} \\ \texttt{eq\_I} \end{array} & \left[\begin{array}{ccc} 1 & . & . \\ 1 & 1 & 1 \\ . & 1 & 1 \end{array}\right]
\end{array}
$$

La colonne `Blk` renseigne sur la partition de la matrice jacobienne qui définit la structure bloc-triangulaire. On y lit que le modèle est composé de deux blocs. Le bloc 1 est composé d'une équation récursive (`R`) qui correspond à l'équation `EQ_CP`. Le bloc 2 est constitué par deux équations simultanées (`S`) qui sont `ID_1` et `EQ_I`.

Un symbole `X` à la ligne i et à la colonne j dans la matrice d'incidence de la matrice jacobienne signifie qu'il existe un lien de causalité directe qui va de la variable j à la variable i. (Par exemple le symbole `X` à la ligne 2 et à la colonne 1 indique que la variable CP influence la variable Y à travers l'équation `ID_1`.)

Un bloc interdépendant est caractérisé par le fait que tout couple de variables (i, j) y appartenant vérifie un chemin (causalité directe et/ou indirecte) qui va de i à j et de j à i. Cette causalité circulaire ne permet pas de mettre en évidence de relation hiérarchique entre les variables du bloc.

On peut cependant ordonner les ensembles de variables interdépendantes de manière hiérarchique. Cette hiérarchie est donnée par le résultat de la commande `model` et se résume pour notre modèle dans la figure qui suit:



Ceci montre clairement la logique du modèle. On peut aussi directement en déduire la structure de la matrice des multiplicateurs d'impact, définie comme

$$\Pi_0 = (I - B_0)^{-1}\Gamma_0 \ .$$

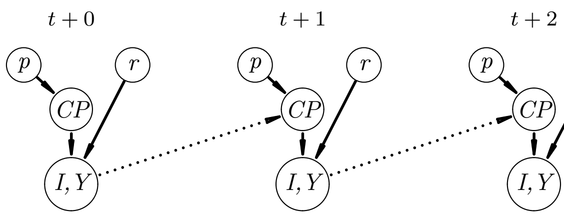
En effet la matrice d'incidence de la matrice Π_0 est:

$$\Pi_0 = \begin{array}{c} \\ CP \\ I \\ Y \end{array} \begin{array}{c} \begin{array}{cc} p & r \end{array} \\ \left[\begin{array}{cc} 1 & . \\ 1 & 1 \\ 1 & 1 \end{array}\right] \end{array}$$

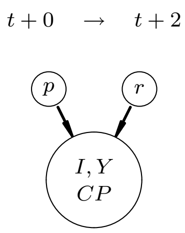
Un élément non nul π_{ij} dans cette matrice d'incidence indique qu'il existe dans le modèle un chemin qui va de la variable j à la variable i.

Modèles dynamiques (causalité intertemporelle)

Les variables endogènes retardées sont à l'origine de relations de causalité lorsque le modèle est résolu pour plus d'une période. En effet pour notre exemple on observe la relation $Y_{t-1} \longrightarrow CP_t$:



Ceci signifie que dans le long terme toutes les variables endogènes deviennent interdépendantes et qu'il n'y a plus de multiplicateur nul.



Pour mettre en évidence la structure logique du long terme avec TSP, il est nécessaire de redéfinir un modèle dans lequel tous les retards des variables endogènes ont été supprimés.

```
frml  eq_lt_CP    CP = 18.5 + 0.73*Y - 0.0626*p;
list  eq_mod_lt   eq_lt_CP  eq_I  id_1;
model eq_mod_lt var_mod   model_lt;
```

```
Block structure of MODEL_LT
Blk Eq# Equation Dep.Var. 123
 1S   1 EQ_LT_CP CP       XX
 1S   2 ID_1     Y        XXX
 1S   3 EQ_I     I         XX
```

On vérifie que les trois équations du modèle forment maintenant un bloc interdépendant `S` unique.

10.4 ALGORITHMES DE RÉSOLUTION

Deux types d'algorithmes sont disponibles dans TSP pour la résolution de systèmes d'équations non-linéaires. Ces algorithmes sont `solve` et `siml`.

`solve` [*options*] *nom_modèle*;

Il s'agit d'un algorithme itératif du type Gauss-Seidel. Pour pouvoir appliquer cet algorithme les équations doivent être normalisées et ordonnées au préalable avec la commande `model`. L'algorithme s'énonce:

pour $k = 0, 1, \ldots,$ *jusqu'à convergence*, faire
  pour $i = 1, 2, \ldots, n,$ faire
    $y_i^{(k+1)} = g_i(y_1^{(k+1)}, \ldots, y_{i-1}^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_n^{(k)}, z)$
  finfaire
finfaire

La solution du modèle donné en exemple pour la période de 1997 à 1998 donne les résultats suivants:

```
smpl 1997 1998;
options nwidth=7, signif=1;
solve (tag=m1) model1;
```

```
Current sample:    1997 to 1998
                          SIMULATION OF THE MODEL MODEL1
```

```
                        =============================
       METHOD = Gauss-Seidel
NUMBER OF EQUATIONS IN THE MODEL =  3
NUMBER OF BLOCKS IN THE MODEL = 2
          BLOCK #   NUMBER OF EQUATIONS
                1         1
                2         2
PERIOD: 1997      BLOCK      2 CONVERGED.
PERIOD: 1998      BLOCK      2 CONVERGED.

THE SOLVED VARIABLES  ARE STORED WITH A TAG:    M1
                              SIMULATION RESULTS
              CP       Y       I
1997        58.5    63.4    4.8
1998        57.2    62.5    5.2
```

L'option `tag=m1` conserve les solutions des variables du modèle dans des séries de même nom augmenté du symbole `m1`. Ces solutions sont alors disponibles pour des calculs ultérieurs.

```
    erY = 100*(Ym1 - Y) / Y;
    options nwidth=7, signif=1;
    print erY;
               ERY
1997          -0.7
1998           3.9
```

`siml (endog=`*liste_variables_endogènes*`[,`*options*`])` *liste_équations*`;`

Cette procédure peut résoudre un modèle écrit sous la forme $F(y) = 0$, c'est-à-dire sans que la normalisation des équations soit explicite (néanmoins l'existence d'une normalisation est une condition nécessaire à l'existence d'une solution). Elle utilise l'algorithme de Newton qui peut être formalisé comme:

`pour` $k = 0, 1, \ldots,$ *jusqu'à convergence*, `faire`
 `Calculer` $\nabla F(y^{(k)})$
 `Résoudre pour` $s^{(k)}$ $\quad \nabla F(y^{(k)})\, s^{(k)} = -F(y^{(k)})$
 $y^{(k+1)} = y^{(k)} + s^{(k)}$
`finfaire`

Considérons une spécification non-linéaire de la fonction de consommation et écrivons l'identité du modèle sous forme implicite. Cette spécification non-linéaire de la fonction de consommation diffère de celle introduite précédemment en ce que la variable endogène Y n'est pas rétardée. Bien que les résultats de l'estimation non-linéaire ne soient pas satisfaisants résolvons le modèle, à titre d'exemple, avec la commande `siml`:

```
    ident id_2  CP + I - Y;
    frml  eqnl_CP  CP = b0*Y**b1 + b2*p;
    param b0  1;
    param b1  1;
    param b2  1;
    lsq  eqnl_CP;
    list  eq_mod2  eqnl_CP eq_I id_2;
    ?
    smpl 1998 1998;
    siml (endog=var_mod,tag=m2) eq_mod2;
```

```
Current sample:    1998 to 1998
                              MODEL SIMULATION
                              ================
                 DYNAMIC SIMULATION
Working space used:  293
                               STARTING VALUES
               CP        Y         I
VALUE         55.0     60.1       5.1

F= 0.1406E-02 FNEW= 0.4268E-06 ISQZ= 0 STEP= 1.00 CRIT= 0.1406E-02
F= 0.4268E-06 FNEW= 0.1232E-13 ISQZ= 0 STEP= 1.00 CRIT= 0.4268E-06

CONVERGENCE ACHIEVED AFTER   2 ITERATIONS
    4 FUNCTION EVALUATIONS.

THE SOLVED VARIABLES  ARE STORED WITH A TAG:    M2
                             SIMULATION RESULTS

               CP        Y        I
1998          53.8     58.8      5.0
```

10.5 VALIDATION DU MODÈLE

Avant de passer à l'utilisation du modèle il convient d'analyser ses propriétés quantitatives. Une première vérification peut porter sur l'aptitude du modèle à reproduire les observations.

```
smpl 1990 1998;
solve (tag=m1) model1;
dot  var_mod;
    er.    = 100 * ( .m1  -  .  )    / .    ;
enddot;
print erY erCP erI;
```

```
             ERY    ERCP     ERI
1990         3.3     3.9    -5.6
1991         1.8     1.7     4.1
1992         0.3     0.5    -2.5
1993        -0.5    -0.9     5.6
```

1994	-2.4	-2.6	-0.8
1995	-3.5	-3.9	0.6
1996	-1.9	-1.9	-1.4
1997	-2.0	-2.2	0.8
1998	2.9	3.0	2.6

Multiplicateurs marginaux

L'analyse des multiplicateurs du modèle renseigne également sur la pertinence du modèle.

Rappelons que la forme réduite d'un modèle, ou de sa linéarisation (10.1), s'écrit

$$y = \underbrace{(I - B_0)^{-1} B_1}_{\Pi_1} y_{-1} + \underbrace{(I - B_0)^{-1} \Gamma_0}_{\Pi_0} x$$

où Π_0 est la matrice des multiplicateurs marginaux. Un élément π_{ij} de cette matrice mesure le taux de variation de la variable endogène y_i, suite à une variation marginale de la variable exogène x_j.

On peut inclure dans la liste des variables exogènes x les constantes des équations de comportement du modèle. Les multiplicateurs associés à cette constante mesurent alors l'impact d'une variation marginale d'une variable endogène sur les autres variables endogènes. On appelle ce type de multiplicateur parfois "shift multiplier".

TSP ne permet pas d'obtenir directement la matrice des multiplicateurs marginaux Π_0. Cependant, on peut calculer des multiplicateurs reflétant des variations finies d'une variable exogène x_j, en procédant de la façon suivante:

- Calculer y, la solution du modèle associé à x.
- Calculer $y^{(j)}$, la solution du modèle associé à x, avec x_j incrémenté de 1.

On a alors

$$\pi_{.\,j} = y^{(j)} - y$$

ce qui traduit la variation du vecteur y en fonction de la variation d'une unité de x_j.

Calculons la colonne $\pi_{.\,p}$ de la matrice des multiplicateurs marginaux qui correspond à la variable p de notre modèle.

```
smpl 1998 1998;
solve (tag=_b) model1;                    (Solution de base y)
p_sav = p;
p = p + 1;                                (Incrémenter p)
solve (tag=_m) model1;                    (Solution pour y^(p))
p = p_sav;
mform(type=general,nrow=3,ncol=1) Pi;
set  Pi(1) = CP_m - CP_b;
set  Pi(2) = Y_m  - Y_b;
set  Pi(3) = I_m  - I_b;
options nwidth=7, signif=3;
print Pi;

            1 (p)
   1 (CP)  -0.0626
   2 (Y)   -0.0665
   3 (I)   -0.0039
```

Ci-après un exemple de procédure TSP qui calcule une matrice des multiplicateurs `Pmat`. Les lignes et les colonnes de `Pmat` sont définies avec les listes `end_vars` et `exo_vars`. Le modèle est défini par `nom_mod` et doit avoir été analysé avec la commande `model` au préalable.

```
proc multiplicateurs   end_vars, exo_vars, nom_mod  Pmat;
length end_vars n_end;
length exo_vars n_exo;
mform(type=general,nrow=n_end,ncol=n_exo) Pmat;
solve (noprnsim,tag=b) nom_mod;
set jj = 0;
dot exo_vars;
    set jj = jj + 1;
    .sav = .;
    .  = 1.01*.;
    solve (noprnsim,tag=m) nom_mod;
    .  = .sav;
    set ii = 0;
    dot end_vars;
        set  ii = ii + 1;
        set Pmat(ii,jj) =  100*( .m  -  .b ) / .b;
    enddot;
enddot;
endproc;
```

On peut facilement modifier cette procédure afin d'otenir des variantes qui utilisent l'algorithme de résolution `siml` ou bien avec une définition alternative de la variation des solutions suite à un choc de variable exogène.

Exemple d'utilisation de la procédure `multiplicateurs`:

```
list exo_vars   p r;
multiplicateurs  var_mod, exo_vars, model1 PP;
print  PP;

        1       2
   1  -0.131   0.000
   2  -0.127 -0.0031
   3 -0.0888 -0.0370
```

Remarquons que le multiplicateur π_{12} est nul conformément au résultat obtenu lors de l'analyse de la structure logique du modèle.

10.6 PRÉPARATION DE SCÉNARIOS

```
smpl 1999 2002;
load accr; 2 3 1 4;
load taux; 0.1 0.15 0.4 0.3;
genr p = p(-1) + accr;
genr r = r(-1) * (1 + taux);
solve(tag=_s1) model1;
```

	CP	Y	I	P	R
1989	40.000	44.700	4.700	65.000	0.100
1990	45.000	48.200	3.200	70.000	0.150
1991	49.000	51.600	2.600	80.000	0.160
1992	50.000	55.000	5.000	105.000	0.0400
1993	53.000	56.500	3.500	100.000	0.100
1994	55.000	59.700	4.700	95.000	0.0500
1995	56.000	61.100	5.100	115.000	0.0200
1996	57.000	61.700	4.700	90.000	0.0500
1997	59.000	63.800	4.800	80.000	0.0400
1998	55.000	60.100	5.100	120.000	0.01000
	CP_S1	Y_S1	I_S1	P	R
1999	54.736	59.981	5.245	122.000	0.0110
2000	54.461	59.665	5.204	125.000	0.0126
2001	54.168	59.275	5.108	126.000	0.0177
2002	53.633	58.629	4.996	130.000	0.0230

10.7 SIMULATION (STOCHASTIQUE)

11

RÉSUMÉS

Classes (types) de variables TSP

La commande `show all` renseigne sur le type de variable défini dans l'espace travail TSP.

Classe	Information	Commandes associées
`SERIES`	nom, # obs, debut, fin, périodicité	`genr`, `load`, `read`, `unmake`, `freq`
`SCALAR`	nom, type (*params*, *const*), valeur	`set`, `const`, `param`, `unmake`
`MATRIX`	nom, dimension, type (*general*, *symetric*, *triangular*, *diagonal*)	`mat`, `mmake`, `mform`, `read`
`LIST`	nom, # membres	`list`
`EQUATION`	nom, type	`frml`, `ident`
`MODEL`	nom	`model`
`PROC`	nom, arguments	

Affectation

La syntaxe à utiliser pour affecter une valeur à une variable dépend de la classe à laquelle appartient la variable.

- Variables de la classe `SERIES`:

  nom_série `=` *expression algébrique avec séries* `;`

Les opérations avec les series nécessitent pas de commande particulière (eventuellement `genr`).

```
PROD = CONS + INV;
```

- Variables de la classe `SCALAR`:

`set` *nom_scalaire* `=` *expression algébrique avec scalaires* `;`

L'expression algébrique qui définit la valeur ne doit faire intervenir que des variables scalaires ou des éléments d'une série (e.g. `x(i)`).

```
set PROD(1995) = 328.75;
set DEGLIB = @NOB - @NCOEF;
```

- Variables de la classe `MATRIX`:

`mat` *nom_matrice* `=` *expression algébrique avec matrices* `;`

```
mat P = A*B;
```

Qu'est-ce-que est important dans l'output (`OLSQ`)

- valeur (signe) des paramètres
- t-statistique, p-value
- R^2, plots, `DW`, etc.

Test d'adéquation

Tester si un paramètre est significativement différent de zero.

$$y = \beta_0 + \beta_1 x + u \qquad u \sim \mathtt{N}(0, \sigma^2)$$

L'éstimation fournit $\widehat{\beta_0}$ and $\widehat{\beta_1}$ $\qquad (\widehat{\beta} = (X'X)^{-1}X'y)$

$$\widehat{\beta_i} \sim \mathtt{N}(\beta_{\mathtt{i}}, \sigma^2_{\beta_{\mathtt{i}}}) \qquad (\beta_{\mathtt{i}}\text{et}\sigma^2_{\beta_{\mathtt{i}}}\text{inconnus!!})$$

$$\frac{\widehat{\beta_i} - \beta_i}{\sigma_{\beta_i}} \sim \mathtt{N}(0, 1)$$

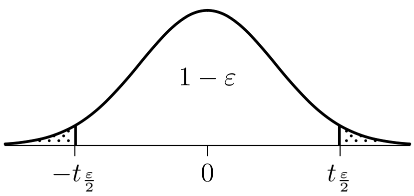
On remplace σ_{β_i} par $\widehat{\sigma_{\beta_i}}$

$$\frac{\widehat{\beta_i} - \beta_i}{\widehat{\sigma_{\beta_i}}} \sim \mathtt{t}(\mathtt{n} - \mathtt{k})$$

<u>Hypothèse:</u> $\beta_i = 0$

$$\frac{\widehat{\beta_i} - 0}{\widehat{\sigma_{\beta_i}}} \sim \mathtt{t}(\mathtt{n} - \mathtt{k})$$

Densité de student:



$$\int_{-t_{\frac{\varepsilon}{2}}}^{t_{\frac{\varepsilon}{2}}} f(t)\, dt = 1 - \varepsilon$$

Pour $n - k > 30$ et $\varepsilon = 0.05$ on a $|t_{\frac{\varepsilon}{2}}| \approx 2$

`P-value`$\equiv \varepsilon$