Formation GAUSS

AGF — Paris

(Compléments de programmation)

Thierry Roncalli
Groupe de Recherche Opérationnelle du Crédit Lyonnais
Bercy-Expo — Immeuble Bercy SUD — 4^{e} étage
90, Quai de Bercy — 75613 Paris Cedex 12

Juin 2003

Table des matières

Première partie
NIVEAU I — PRISE EN MAIN DE GAUSS

1 Introduction

1.1 Configuration de GAUSS

Le fichier *gauss.cfg* permet de configurer GAUSS. Il est composé de plusieurs parties :
- une partie permettant de définir les répertoires (exécutable, bibliothèque, DLLs, bases de données, etc.) ;
- une ligne de commande pour spécifier l'éditeur (vi par défaut) ;
- une partie pour gérer l'espace de travail ;
- et enfin, la déclaration par défaut de certaines variables (qui peuvent être globales).

Remarque 1 *Nous pouvons généralement changer les valeurs de ces variables localement dans un programme avec la commande* `sysstate`*.*

1.2 Un premier exemple

Considérons les lignes de commande suivantes :

```
new;
x = rndn(100,1);
print x;
```

Le mode commande (fenêtre GAUSS) permet bien sûr d'exécuter des lignes de commandes, mais il permet aussi de récupérer les résultats.
- Le fichier *gauss.err* contient l'ensemble des messages d'erreur d'une session.
- Le fichier *command.log* contient toutes les commandes exécutées. Celui-ci n'est pas initialisé à chaque session.

Exemple 1 *Exécutez la commande* `y = inv(x)` *qui produit une erreur puisque* `x` *est un vecteur.*

1.3 La philosophie de GAUSS

- Le concept de langage matriciel (GAUSS, MATLAB, etc.)
- Les opérateurs $E \times E$
- Un langage mathématique

$\Longrightarrow$ GAUSS n'est pas un logiciel statistique, un logiciel d'économétrie, un logiciel de finance, un logiciel de traitement du signal, etc. Pourtant, GAUSS est intensivement utilisé dans chacun de ces domaines. GAUSS n'a pourtant pas la vocation à devenir par exemple un logiciel de statistiques. Cela reste avant tout un langage de programmation mathématique.

Remarque 2 *La commande* `stepwise` *de SAS.*

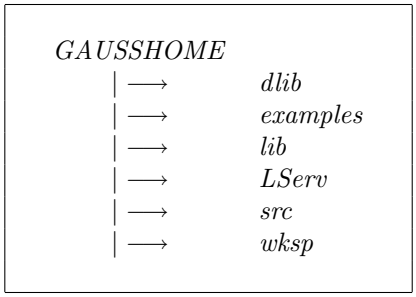
1.4 L'aide en ligne

voir le supplément Unix.

1.5 le debugger

voir le supplément Unix.

1.6 L'architecture



2 Découverte des commandes et des opérateurs de bases

2.1 Déclaration d'une matrice

Il existe différentes façons de déclarer une matrice avec la commande `let` :

```
let x = {1 2,3 4,5 6};        /* Matrice de dimension 3*2 */
let x[3,2] = 1 2 3 4 5 6;     /* Matrice de dimension 3*2 */
let x = 1 2 3 4 5 6;          /* Vecteur de dimension 6 */
let x[1,6] = 1 2 3 4 5 6;     /* Vecteur ligne de dimension 6 */
let x = 4;                    /* scalaire */
x = 4;                        /* scalaire (affectation) */
```

Pour les nombres complexes, on utilise la notation compacte `a+bi`. Par exemple :

```
let x = {1+5i, 2, 3};
```

Remarque 3 *GAUSS possède un nouveau* ***scan*** *pour la déclaration des matrices.(voir le fichier readme).*

2.2 Les opérateurs matriciels

```
Operators, Mathematical and String

                Operators
        ====================================
         =      %     ==    eq    $==    and
         +      [    .==   .eq   .$==   .and
        $+      ]     >=    ge    $>=    eqv
         -      :    .>=   .ge   .$>=   .eqv
         *      ,     >     gt    $>     not
        .*      .    .>    .gt   .$>    .not
         /      |     <=    le    $<=    or
        ./     $|    .<=   .le   .$<=   .or
         *~     ~     <     lt    $<     xor
        .*.    $~    .<    .lt   .$<    .xor
         ^     '     /=     ne    $/=    {
         !     .'    ./=   .ne   .$/=    }

The mathematical, matrix, string array, and string operators are:

   MATHEMATICAL OPERATORS             MATRIX AND STRING ARRAY OPERATORS
    +   addition                       |     matrix vertical concatenation
```

```
    -   subtraction or unary minus     ~     matrix horizontal concatenation
    *   multiplication                 $|    string array vert concatenation
   .*   ExE multiplication            $~    string array horiz concatenation
    ^   ExE exponentiation             '     transpose
    !   factorial                      .'    bookkeeping transpose

   ./   ExE division
    /   division or linear equation solution of Ax = b, for example: x = b/A;
    %   modulo division
   .*.  Kronecker product             STRING OPERATORS
    *~  horizontal direct product      $+      string concatenation

Symbols used for indexing matrices are:  ''['', '']'', ''.'' and '':''.  For example,
     x[1 2 5]       returns the 1st, 2nd and 5th elements of x.
     x[2:10]        returns the 2nd through 10th elements of x.
     x[.,2 4 6]     returns all rows of the 2nd, 4th, and 6th columns of x.
```

```
x = ones(5,1);                /* cre'ation d'un vecteur unitaire   */
y = zeros(5,2);               /* cre'ation d'une matrice nulle     */
z = eye(5);                   /* cre'ation d'une matrice identite' */

xz = x * z;                   /* multiplication */
z = 2 + z;                    /* addition       */
x = x + x;                    /* addition       */

x = x~x;                      /* concate'nation horizontale */
z = z|z:                      /* concate'nation verticale   */

/*
** Cre'ation de nombres ale'atoires CHI2 10 degre's de liberte'
*/

x = rndn(10,4);
x = x^2;                      /* introduction d'un ope'rateur ExE */
x = sumc(x);

@ ou @

x = rndn(10,4);
x = x .* x;
x = sumc(x);
```

Cette notion d'opérateurs $E \times E$ est primordiale pour bien maîtriser GAUSS. Considérons le calcul de

$$f(x) = \frac{\sin(x)\cos(x)}{x + \sqrt{x} + 1}$$

pour $x = 1, 2, \ldots, 100$. Nous avons :

```
x = seqa(1,1,100);                            /* sequence additive */
y = (sin(x) .* cos(x)) ./ (x +sqrt(x) + 1);   /* f(x)              */
```

Calculons maitenant la matrice $(f(i) \times f(j))_{i,j}$. Nous avons :

```
y = y .* y';
```

De nouveaux exemples de manipulations $E \times E$:

```
let x[1,2] = 1 2;
let y = {3 4,5 6};
z = x + y;
z = x' + y;
z = x .* y;
z = x' .* y;
z = x ./ y;
z = y ./ x;
z = y ./ x';
```

2.3 Les sous-matrices

- `x[r,c]` permet d'accéder à l'élément de la r-ième ligne et c-ième colonne.
- La syntaxe précédente se généralise lorsque `r` et `c` sont des vecteurs.
- La notation `.` permet de sélectionner l'ensemble des lignes/colonnes.
- La notation `d :f` est équivalente à `seqa(d,1,d-f+1)`.

```
x = rndu(150,65);
print x[1,1];
print x[150,65];
print x[1,1]~x[1,2];
print x[1,1 2];

y = x[1:10 15 25,1 2 5:12 20 21:26];
print y;

/*
** Exemples d'affectation
*/

y[1,1] = 0;
y[1,2] = 1;
print y;

y[1,1 2] = 0~1;
```

2.4 Les commandes mathématiques

- `chol` : décomposition de Cholesky
- `det` : déterminant d'une matrice
- `inv` : inversion d'une matrice carrée
- `invpd` : inversion d'une matrice pds
- `eig`, `eigv`, `eighv` : décomposition vecteurs/valeurs propres
- `svd` : décomposition valeurs sigulières
- etc.

Exemple 2 (Simulation d'une loi normale multidimensionnelle) *Soit Q une matrice telle que $\Sigma = QQ^\top$, alors*

$$\mathcal{N}(\mu, \Sigma) = \mu + Q\mathcal{N}(\mathbf{0}, I)$$

Il existe alors plusieurs décompositions possibles de Σ : La décomposition de Cholesky implique que $Q = \mathbf{P}$, avec la décomposition en valeurs propres $\Sigma = V\Lambda V^\top$, nous avons $Q = V\Lambda^{\frac{1}{2}}$, enfin $Q = US^{\frac{1}{2}}$ lorsque nous utilisons la décomposition en valeurs singulières $\Sigma = USV^\top$.

2.5 Les commandes statistiques

- `cdf*` : fonctions de répartition (par exemple, `cdfn`)
- `cdf*c` : fonctions de répartition complémentaire (par exemple, `cdfnc`)
- `cdf*i` : fonctions de répartition inverse ou quantile (par exemple, `cdfni`)
- `rnd*` : nombres aléatoires (par exemple, `rndn`)
- `meanc`, `stdc`, `vcx`, `corrx`, `quantile`, etc.

2.6 Les commandes de manipulation de matrices

- `cols` : nombre de colonnes
- `rows` : nombre de lignes
- `diag` : diagonale de la matrice
- `maxc`, `minc` : maximum & minimum
- `maxindc`, `minindc` : position des maximum & minimum
- `miss`, `missex`, `missrv`, `packr` : valeurs manquantes
- `vec`, `vech`, `xpnd`, `reshape`, `rev` : réorganisation des matrices
- `trimr` : élimination des premières et dernières lignes d'une matrice
- `eye`, `ones`, `zeros` : matrices identité, unitaire et nulle
- `imag`, `real`, `complex` : matrices complexes

2.7 Les opérateurs conditionnels et relationnels

- Les opérateurs conditionnels : `not`, `and`, `or`, `xor` et `eqv`
- Les opérateurs conditionnels $E \times E$: `.not`, `.and`, `.or`, `.xor` et `.eqv`
- Les opérateurs relationnels : `eq`, `ne`, `gt`, `lt`, `ge`, `le`
- Les opérateurs relationnels $E \times E$: `.eq`, `.ne`, `.gt`, `.lt`, `.ge`, `.le`
- Les opérateurs relationnels (forme symbolique) : `==`, `/=`, $>$, $<$, $>=$, $<=$
- Les opérateurs relationnels $E \times E$ (forme symbolique) : `.==`, `./=`, `.>`, `.<`, `.>=`, `.<=`
- Les opérateurs relationnels flous (*fuzzy*) : `feq`, `fne`, `fgt`, `flt`, `fge`, `fle`
- Les opérateurs relationnels flous $E \times E$ (*fuzzy*) : `dotfeq`, `dotfne`, `dotfgt`, `dotflt`, `dotfge`, `dotfle`

2.8 Les structures de contrôle

- `do while ... endo`
- `do until ... endo`
- `for i(start,end,incr) ... endfor`
- `if ... elseif ... else ... endif`
- `break`
- `continue`
- `goto`
- `gosub`
- `pop`

3 La différence entre une procédure et une commande

```
(gauss) inv(0|0)
```

```
(0) : error G0039 : Matrix must be square to invert
(gauss) pinv(0|0)
Singular values all zero
Currently active call: PINV [259]
```

Dans l'exemple précédent, nous calculons l'inverse d'un vecteur. Nous obtenons une erreur aussi bien pour la **commande** `inv` que pour la **procédure** `pinv` (inverse de Moore-Penrose). Cependant, les messages d'erreur ne présentent pas le même format :

- Le premier message d'erreur commence par **(0)** et affiche un code d'erreur GAUSS **G0039** commençant par la lettre **G**.
- Le second message d'erreur fait référence à la procédure `PINV` et affiche un nombre `[259]`.

```
(gauss) x = eye(5)
(gauss) y = eye(2)
(gauss) x*y

(0) : error G0036 : Matrices are not conformable
(gauss) crossprd(x,y)

C:\GAUSS\SRC\CROSSPRD.SRC(41) : error G0058 : Index out of range
Currently active call: CROSSPRD [41]
```

Une nouvelle fois, nous obtenons deux formats d'erreur différente. **L'explication est la suivante : `inv` et l'opérateur $*$ sont des commandes intrinsèques de GAUSS écrites en langage C ou Fortran, alors que `pinv` et `crossprd` sont des procédures GAUSS écrites en langage GAUSS.** Elles font toutes les deux parties d'une bibliothèque. D'ailleurs, il est tout à fait possible de visualiser le code, puisque le message d'erreur nous indique dans quel fichier se trouve la procédure correspondante.

```
/*
** crossprd.src
** (C) Copyright 1988-1998 by Aptech Systems, Inc.
** All Rights Reserved.
**
** This Software Product is PROPRIETARY SOURCE CODE OF APTECH
** SYSTEMS, INC.    This File Header must accompany all files using
** any portion, in whole or in part, of this Source Code.   In
** addition, the right to create such files is strictly limited by
** Section 2.A. of the GAUSS Applications License Agreement
** accompanying this Software Product.
**
** If you wish to distribute any portion of the proprietary Source
** Code, in whole or in part, you must first obtain written
** permission from Aptech Systems.
**
**> crossprd
**
**  Purpose:    Computes the cross products (vector products) of
**              sets of 3x1 vectors.
**
**  Format:     z = crossprd(x,y);
**
**  Input:      x    3xK matrix, each column is treated as a 3x1 vector.
**
**              y    3xK matrix, each column is treated as a 3x1 vector.
**
```

```
**  Output:     z    3xK matrix, each column is the cross product
**                   (sometimes called vector product) of the
**                   corresponding columns of x and y.
**
**  Remarks:    The cross product vector (z) is orthogonal to both x and y.
**              sumc(x.*z) and sumc(y.*z) will be Kx1 vectors all of whose
**              elements are 0 (except for rounding error).
**
**  Globals:    None
*/

proc crossprd(x,y);
    local r1, r2, r3;
    r1 = x[2,.].*y[3,.]-x[3,.].*y[2,.];
    r2 = x[3,.].*y[1,.]-x[1,.].*y[3,.];
    r3 = x[1,.].*y[2,.]-x[2,.].*y[1,.];
    retp( r1|r2|r3 );
endp;
```

La dernière version de GAUSS comprend plus de 800 opérateurs, commandes et procédures. Sur ce nombre, plus de la moitié ne font pas partie du langage GAUSS, mais correspondent à la bibliothèque GAUSS.

Remarque 4 *GAUSS est un langage relativement puissant, qui permet finalement de* ***tout*** *développer dans ce langage. Il existe actuellement une quarantaine de bibliothèques GAUSS, et toutes sont écrites en pur langage GAUSS sans recourir à d'autres langages tels que le C pour réduire les temps de calcul (politique d'APTECH).*

- OPTMUM : Bibliothèque d'optimisation (à comparer avec la bibliothèque d'optimisation de S+)
- CML & MaxLik : Bibliothèques de maximum de vraisemblance (à comparer avec les procédures équivalentes de SAS).
- ARIMA : Procédure d'estimation de modèles ARMA (à comparer avec la procédure de SAS).
- TSM : Bibliothèque de séries temporelles (à comparer avec la bibliothèque de signal de MatLab, les procédures d'estimation de modèles espace-état de SAS et S+).

Les points à développer :

1. La notion de **procédure**
2. La notion de **bibliothèque**

Considérons la loi de Gumbel. La fonction de distribution correspond à

$$\mathbf{F}(x) = \exp(-\exp(-x))$$

```
(gauss) x = seqa(-5,0.1,101)
(gauss) cdf = exp(-exp(-x))
```

Nous pourrions très bien envisager de créer une fonction (ou une procédure) qui permettrait de calculer $\mathbf{F}(x)$ en employant la ligne de commande

```
cdf = cdfGumbel(x) ;
```

La solution est

```
fn cdfGumbel(x) = exp(-exp(-x))
```

1. Manipulation de la fonction `cdfGumbel`
2. Construction de la bibliothèque ritme
   (a) Créer un répertoire *src/ritme*
   (b) Créer un fichier *gumbel.src* contenant la fonction `cdfGumbel`

(c) Créer la bibliothèque ritme avec la ligne de commande :

```
lib ritme gumbel.src -a
```

3. Utilisation de la bibliothèque avec la commande :

```
library ritme ;
```

4 Quelques explications sur les procédures et les bibliothèques

La syntaxe d'une procédure est

```
proc (n) = nom_de_la_procedure(entree_1,entree_2,...,entree_m);
  local variableLocale_1,variableLocale_2,...,variableLocale_p;
  local retour_1,retour_2,...,retour_n;

  /* Code source de la procedure
  :: Les variables locales servent a definir des calculs intermediaires
  :: Les variables retour_* sont les variables ''retour'' et doivent donc
  :: etre toujours definies
  */

  retour_1 = ... ;
  retour_2 = ... ;

  retour_n = ... ;

  retp(retour_1,retour_2,...,retour_n);
endp;
```

Dans ce cas, l'appel de la procédure se fait de la façon suivante :

```
{retour_1,retour_2  ,  ...,retour_n} =
                       nom_de_la_procedure(entree_1,entree_2,...,entree_m) ;
```

Quelques cas particuliers. Si nous désirons excéuter la procédure, mais ne pas définir les retours, nous employons la commande `call` :

```
call nom_de_la_procedure(entree_1,entree_2,...,entree_m) ;
```

Si le nombre de retour est égal à 1, il n'est pas nécessaire de définir `(n)`. Nous pouvons avoir

```
proc nom_de_la_procedure(entree_1,entree_2,...,entree_m) ;
```

Dans ce cas, nous pouvons employer indifférement

```
{retour_1} = nom_de_la_procedure(...) ;
```

ou plus simplement

```
retour_1 = nom_de_la_procedure(...) ;
```

Si le nombre de retour est égal à zéro, nous avons

```
proc (0) = nom_de_la_procedure(entree_1,entree_2,...,entree_m) ;
```

Nous appelons alors la procédure de la façon suivante :

```
nom_de_la_procedure(entree_1,entree_2,...,entree_m) ;
```

Enfin, la syntaxe d'un procédure sans argument d'entrée est

```
proc (n) = nom_de_la_procedure ;
```

Voyons différents exemples :

▶ cumsumc

```
proc cumsumc(x);
    if rows(x) == 1;
        retp(x);
    else;
        retp( recserar(x,x[1,.],ones(1,cols(x)) ) );
    endif;
endp;
```

▶ olsqr2

```
proc (3) = olsqr2(y,x);
    local flag,n,p,qraux,work,pvt,job,b,k,
        rsd,xb,info,qy,qty,rd;

    /* check for complex input */
    if iscplx(x);
        if hasimag(x);
            errorlog ''ERROR: Not implemented for complex matrices.'';
            end;
        else;
            x = real(x);
        endif;
    endif;

    if iscplx(y);
        if hasimag(y);
            errorlog ''ERROR: Not implemented for complex matrices.'';
            end;
        else;
            y = real(y);
        endif;
    endif;

    n = rows(x);
    p = cols(x);
    qraux = zeros(p,1);
    work = qraux;
    pvt = qraux;
    flag = 1;       /* Use pivoting */
        /* compute matrix dimensions and other inputs to qrsl subroutine  */
    if rows(y) ne n;
        errorlog ''ERROR: OLSQR2 - X and Y must have same length'';
```

```
        end;
    elseif n < p;
        errorlog ''ERROR: OLSQR2 - Problem is underdetermined (N < P)'';
        end;
    endif;

    b = zeros(p,1);         /* Vector to hold ols coeffs */
    rsd = zeros(n,1);       /* Vector to hold residuals */
    xb = rsd;       /* Vector to hold predicted values */
    info = 0;
    job = 111;      /* compute b, rsd, xb */
    qy = rsd;
    qty = rsd;

    k = minc(n|p);

    x = x';

#ifDLLCALL
#else

    if rows(_qrdc) /= 647 or _qrdc[1] $== 0;
        _qrdc = zeros(647,1);
        loadexe _qrdc = qrdc.rex;
    endif;
    callexe _qrdc(x,n,n,p,qraux,pvt,work,flag);

#endif

#ifDLLCALL

    dllcall qrdc(x,n,n,p,qraux,pvt,work,flag);

#endif

    rd = abs(diag(trimr(x',0,n-p)));        /* abs of diagonal of R  */
    k = sumc( rd .> _olsqtol*rd[1,1] );     /* number of diagonal elements of
                                            :: R that are greater than
                                            :: tolerance
                                            */

#ifDLLCALL
#else

    if rows(_qrsl) /= 455 or _qrsl[1] $== 0;
        _qrsl = zeros(455,1);
        loadexe _qrsl = qrsl.rex;
    endif;
    callexe _qrsl(x,n,n,k,qraux,y,qy,qty,b,rsd,xb,job,info);

#endif
```

```
#ifDLLCALL

    dllcall qrsl(x,n,n,k,qraux,y,qy,qty,b,rsd,xb,job,info);

#endif

    /* sort b to put it in correct order */
    b = submat( sortc(b~pvt,2),0,1);

    retp(b,rsd,xb);
endp;
```

Les variables locales ne sont pas accesibles en dehors de la procédure. Les variables d'entrée sont aussi des variables locales. Cela veut dire qu'il y a copie des entrées.

▶ A titre d'illustration, que fait le programme suivant ?

```
proc f(x);
  x = x^2;
  retp(x);
endp;

x = 2;
y = f(x);
print y~x;
z = f(y);
print z~y;
```

Exemple 3 *Ecrire une procédure de simulation de nombres χ_2 à n degrés de liberté dont la syntaxe est :*

```
u = rndchi(r,c,n);
```

Exemple 4 *Nous considérons la fonction de distribution "Generalized Extreme Value distribution" (GEV) $\mathcal{G}(\mu,\sigma,\xi)$*

$$\mathbf{G}(x) = \exp\left\{-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}}\right\}$$

défini sur le support $\Delta = \left\{x : 1+\xi\left(\frac{x-\mu}{\sigma}\right) > 0\right\}$. La fonction de densité correspondante est

$$g(x) = \frac{1}{\sigma}\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-\left(\frac{1+\xi}{\xi}\right)} \exp\left\{-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}}\right\}$$

Le quantile $\mathbf{G}^{-1}(\alpha)$ d'ordre α pour la distribution GEV est donné par la formule suivante :

$$\mathbf{G}^{-1}(\alpha) = \mu - \frac{\sigma}{\xi}\left[1-(-\ln\alpha)^{-\xi}\right]$$

Ecrire quatre procédures pour la fonction de distribution, la fonction de densité, le quantile et la génération de nombres aléatoires.

▶ Solution proposée :

```
/*
**> pdfGEV
**
*/
```

```
proc pdfGEV(x,mu,sigma,xi);
  local y,e,pdf;

  x = (x - mu) ./ sigma;
  y = (1 + xi .* x);
  e = y .> 0;
  y = y .* e + __macheps * (1-e);
  pdf = (y^(-(1+xi)./xi)) .* exp( -(y^(-1./xi)) ) ./ sigma;
  pdf = pdf .* e;

  retp(pdf);
endp;


/*
**> cdfGEV
**
*/


proc cdfGEV(x,mu,sigma,xi);
  local y,e,cdf;

  x = (x - mu) ./ sigma;
  y = (1 + xi .* x);
  e = y .> 0;
  y = y .* e + __macheps * (1-e);
  cdf = exp(-( y^(-1./xi) ));

  cdf = cdf .* e + (1-e) .* (xi .< 0);

  retp(cdf);
endp;


/*
**> cdfGEVi
**
*/


proc cdfGEVi(alpha,mu,sigma,xi);
  local cdfi;

  alpha = miss(miss(alpha,0),1);
  cdfi = mu - sigma.*(1 - (-ln(alpha))^(-xi))./xi;

  retp(cdfi);
endp;


/*
```

```
**> rndGEV
**
*/


proc rndGEV(r,c,mu,sigma,xi);
  local u;

  u = cdfGEVi(rndu(r,c),mu,sigma,xi);

  retp(u);
endp;
```

▶ Manipulation de la procédure `OLS` (I)

```
/*
** ols1.prg
**
*/

new;

rndseed 123;

Nobs = 100;
K = 5;
sigma = 0.5;


x = ones(Nobs,1)~rndu(Nobs,K);
beta = seqa(1,1,K+1);
y = x*beta + rndn(Nobs,1)*sigma;

output file = ols1.out reset;

call ols(0,y,x);

output off;
```

```
Valid cases:                 100     Dependent variable:                Y
Missing cases:                 0     Deletion method:                None
Total SS:                659.196     Degrees of freedom:               94
R-squared:                 0.965     Rbar-squared:                  0.963
Residual SS:              22.988     Std error of est:              0.495
F(5,94):                 520.297     Probability of F:              0.000

                       Standard                Prob   Standardized  Cor with
Variable     Estimate     Error      t-value    >|t|     Estimate    Dep Var
-------------------------------------------------------------------------------
CONSTANT     0.743400   0.228847    3.248453    0.002       ---         ---
X1           2.004655   0.191707   10.456887    0.000    0.205635    0.330995
X2           3.087867   0.185820   16.617507    0.000    0.322132    0.331370
X3           4.307761   0.186731   23.069299    0.000    0.455438    0.314800
```

```
X4          5.413424    0.182912   29.595702     0.000    0.589843    0.457336
X5          5.665314    0.184540   30.699685     0.000    0.602168    0.626387
```

▶ Manipulation de la procédure `OLS` (II)

```
/*
** ols2.prg
**
*/

new;

rndseed 123;

Nobs = 100;
K = 5;
sigma = 0.5;


x = ones(Nobs,1)~rndu(Nobs,K);
beta = seqa(1,1,K+1);
y = x*beta + rndn(Nobs,1)*sigma;

output file = ols2.out reset;

{vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat} = ols(0,y,x);
print b;
print $vnam;

{vnam,vnam,vnam,stb,vc,stderr,sigma,cx,sigma,resid,dwstat} = ols(0,y,x);
print $vnam;
print sigma;

output off;
```

```
Valid cases:                100     Dependent variable:                Y
Missing cases:                0     Deletion method:                None
Total SS:                   659.196     Degrees of freedom:               94
R-squared:                    0.965     Rbar-squared:                  0.963
Residual SS:                 22.988     Std error of est:              0.495
F(5,94):                    520.297     Probability of F:              0.000

                         Standard                Prob   Standardized  Cor with
Variable     Estimate      Error      t-value     >|t|     Estimate    Dep Var
-------------------------------------------------------------------------------
CONSTANT     0.743400    0.228847    3.248453     0.002       ---         ---
X1           2.004655    0.191707   10.456887     0.000    0.205635    0.330995
X2           3.087867    0.185820   16.617507     0.000    0.322132    0.331370
X3           4.307761    0.186731   23.069299     0.000    0.455438    0.314800
X4           5.413424    0.182912   29.595702     0.000    0.589843    0.457336
X5           5.665314    0.184540   30.699685     0.000    0.602168    0.626387

      0.74339992
```

```
       2.0046546
       3.0878667
       4.3077609
       5.4134239
       5.6653139

             X1
             X2
             X3
             X4
             X5
              Y
Valid cases:                  100     Dependent variable:                  Y
Missing cases:                  0     Deletion method:                  None
Total SS:                 659.196     Degrees of freedom:                 94
R-squared:                  0.965     Rbar-squared:                    0.963
Residual SS:                22.988     Std error of est:                0.495
F(5,94):                   520.297     Probability of F:                0.000

                       Standard                Prob   Standardized  Cor with
Variable     Estimate     Error      t-value    >|t|     Estimate    Dep Var
-------------------------------------------------------------------------------
CONSTANT     0.743400   0.228847    3.248453    0.002       ---         ---
X1           2.004655   0.191707   10.456887    0.000    0.205635    0.330995
X2           3.087867   0.185820   16.617507    0.000    0.322132    0.331370
X3           4.307761   0.186731   23.069299    0.000    0.455438    0.314800
X4           5.413424   0.182912   29.595702    0.000    0.589843    0.457336
X5           5.665314   0.184540   30.699685    0.000    0.602168    0.626387

             X1
             X2
             X3
             X4
             X5
              Y
     0.49452562
```

▶ Manipulation de la procédure `OLS` (III)

```
/*
** ols3.prg
**
*/

new;

rndseed 123;

Nobs = 100;
K = 5;
sigma = 0.5;


x = ones(Nobs,1)~rndu(Nobs,K);
```

```
beta = seqa(1,1,K+1);
y = x*beta + rndn(Nobs,1)*sigma;

output file = ols3.out reset;

__output = 0;
{vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat} = ols(0,y,x);
print b;

__output = 1;
let __altnam = C AGE SEXE TAILLE POIDS PAYS NOTE;
{vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat} = ols(0,y,x);

output off;


      0.74339992
       2.0046546
       3.0878667
       4.3077609
       5.4134239
       5.6653139
Valid cases:                 100      Dependent variable:                NOTE
Missing cases:                 0      Deletion method:                   None
Total SS:                659.196      Degrees of freedom:                  94
R-squared:                 0.965      Rbar-squared:                     0.963
Residual SS:              22.988      Std error of est:                 0.495
F(5,94):                 520.297      Probability of F:                 0.000

                        Standard                Prob   Standardized  Cor with
Variable     Estimate      Error      t-value     >|t|     Estimate    Dep Var
-------------------------------------------------------------------------------
C            0.743400    0.228847    3.248453    0.002        ---         ---
AGE          2.004655    0.191707   10.456887    0.000    0.205635    0.330995
SEXE         3.087867    0.185820   16.617507    0.000    0.322132    0.331370
TAILLE       4.307761    0.186731   23.069299    0.000    0.455438    0.314800
POIDS        5.413424    0.182912   29.595702    0.000    0.589843    0.457336
PAYS         5.665314    0.184540   30.699685    0.000    0.602168    0.626387
```

Remarque 5 *La procédure* `OLS` *est une vieille procédure écrite dans les années 1985. A l'époque,* `GAUSS` *ne permettait pas les retours multiples. La syntaxe de la procédure était alors*

```
q = ols(dataset,depvar,indvars);
```

avec `q` *un buffer matriciel construit avec la procédure* `vput`*. Aujourd'hui, une implémentation de la procédure* `OLS` *serait différente. Pourquoi ?*

5 La bibliothèque graphique PGRAPH

5.1 Premier exemple

Nous considérons la fonction

$$f(x) = \sin(x)\cos(x)\,\mathcal{Y}_3^{(2)}(x)$$

Dans l'exemple suivant, nous graphons cette fonction. La version UNIX nécessite quelques manipulations nécessaires afin de déclarer la fenêtre graphique (manipulations qui n'existent pas pour NT).

```
new;
library pgraph;

proc f(x);
  local y;

  y = sin(x) .* cos(x);
  y = y .* bessely(3,x);

  retp(y);
endp;

y = seqa(pi,2*pi/100,301);
x = f(y);

#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);

#ENDIF


xy(y,x);


#IFUNIX

call WinSetActive(1);

#ENDIF
```

5.2 Améliorer la mise en forme du graphique

Il existe de nombreuses variables globales pour améliorer la mise en page. Nous pouvons ainsi ajouter un titre, des labels, ajouter une légende, des messages, dessiner des symboles, etc.

```
new;
library pgraph;

proc f(x);
  local y;

  y = sin(x) .* cos(x);
  y = y .* bessely(3,x);

  retp(y);
endp;

y = seqa(pi,2*pi/100,301);
```

```
x = f(y);

#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);

#ENDIF


graphset;
  _pdate = ''ritme '';
  _pnum = 2;
  title(''Un exemple de trace' de fonction'');
  xlabel(''x'');
  ylabel(''f(x)'');
  _pcross = 1;
  _pframe = 0;
  xy(y,x);


#IFUNIX

call WinSetActive(1);

#ENDIF
```

5.3 Les fontes graphiques

Nous considérons un premier changement de fontes.

```
new;
library pgraph;

proc f(x);
  local y;

  y = sin(x) .* cos(x);
  y = y .* bessely(3,x);

  retp(y);
endp;

y = seqa(pi,2*pi/100,301);
x = f(y);

#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);
```

```
#ENDIF


graphset;
  _pdate = ''ritme '';
  _pnum = 2;
  fonts(''complex'');
  title(''Un exemple de trace' de fonction'');
  xlabel(''x'');
  ylabel(''f(x)'');
  _pcross = 1;
  _pframe = 0;
  ytics(-0.15,0.15,0.05,5);
  xtics(pi,7*pi,pi,0);
  let labx = ''PI'' ''2*PI'' ''3*PI'' ''4*PI'' ''5*PI'' ''6*PI'' ''7*PI'';
  asclabel(labx,0);
  xy(y,x);


#IFUNIX

call WinSetActive(1);

#ENDIF
```

Voyons maintenant comment mélanger et appeler plusieurs fontes en même temps.

```
new;
library pgraph;

proc f(x);
  local y;

  y = sin(x) .* cos(x);
  y = y .* bessely(3,x);

  retp(y);
endp;

y = seqa(pi,2*pi/100,301);
x = f(y);

#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);

#ENDIF


graphset;
  _pdate = ''ritme '';
  _pnum = 2;
```

```
  fonts(''complex simgrma'');
  title(''Un exemple de trace' de fonction\Lf(x) = sin(x) cos(x) Y]3[[(2)](x)'');
  xlabel(''x'');
  ylabel(''\201f(x)'');
  _pcross = 1;
  _pframe = 0;
  ytics(-0.15,0.15,0.05,5);
  xtics(pi,7*pi,pi,0);
  let labx = ''\202p\201'' ''\2012\202p\201'' ''\2013\202p\201'' ''\2014\202p\201''
             ''\2015\202p\201'' ''\2016\202p\201'' ''\2017\202p\201'';
  asclabel(labx,0);
  graphprt(''-c=1 -cf=pgraph4.ps'');
  xy(y,x);


#IFUNIX

call WinSetActive(1);

#ENDIF
```

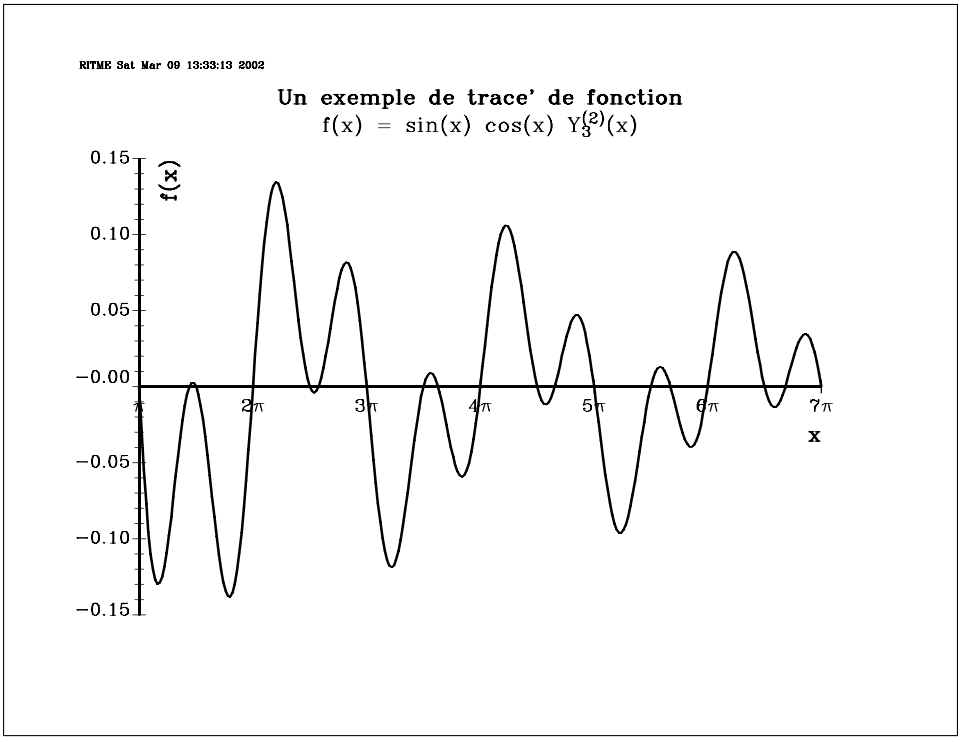


FIG. 1 – Un exemple de la bibliothèque **PGRAPH**

5.4 Les légendes

```
new;
library pgraph;
```

```
proc f(x);
  local y;

  y = sin(x) .* cos(x);
  y = y .* bessely(1~2~3~4,x);

  retp(y);
endp;

y = seqa(pi,2*pi/100,301);
x = f(y);

#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);

#ENDIF


graphset;
  _pdate = '''';
  _pnum = 2;
  fonts(''complex simgrma'');
  title(''Un exemple de trace' de fonction\Lf(x) = sin(x) cos(x) Y]\202a\201[[(2)](x)'');
  _pltype = 6|1|3|4; _pstype = 8|7|3|5; _plctrl = 0|0|10|10;
  xlabel(''x'');
  ylabel(''\201f(x)'');
  _pcross = 1;
  _pframe = 0;
  ytics(-0.20,0.20,0.05,5);
  xtics(pi,7*pi,pi,0);
  let labx = ''\202p\201'' ''\2012\202p\201'' ''\2013\202p\201'' ''\2014\202p\201''
             ''\2015\202p\201'' ''\2016\202p\201'' ''\2017\202p\201'';
  asclabel(labx,0);
  _plegstr = ''\202a\201 = 1\000\202a\201 = 2\000\202a\201 = 3\000\202a\201 = 4'';
  _plegctl = {2 5 7 0.5};
  graphprt(''-c=1 -cf=pgraph5.ps'');
  xy(y,x);


#IFUNIX

call WinSetActive(1);

#ENDIF
```

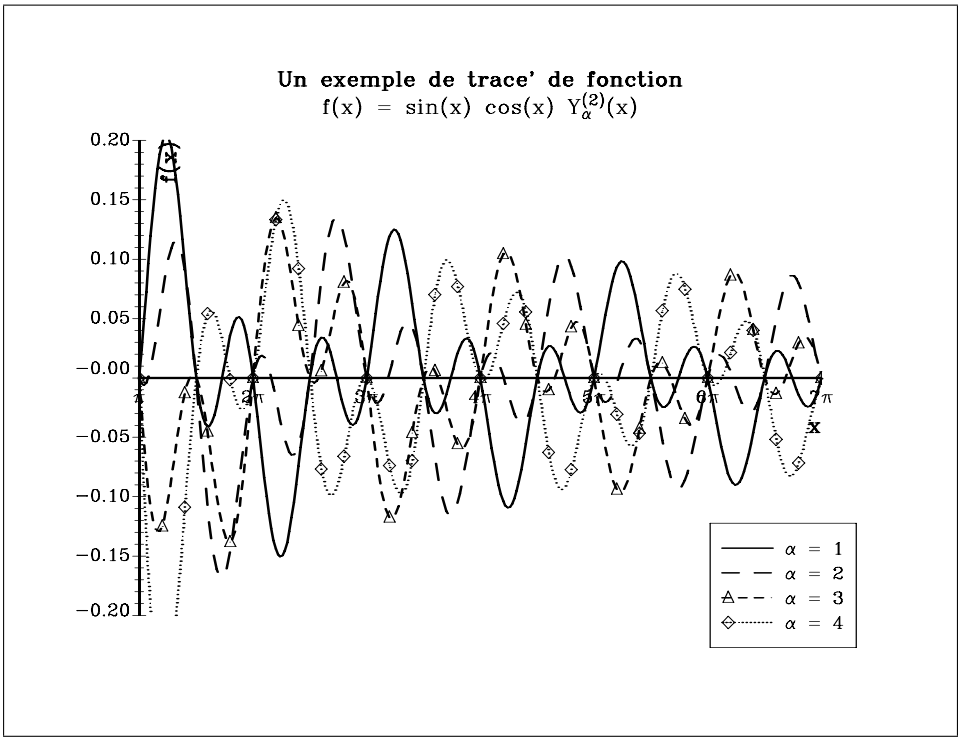


FIG. 2 – Un second exemple de la bibliothèque **PGRAPH**

5.5 Le mode fenêtre

Il est possible d'afficher plusieurs graphes simultanément. Pour cela, nous utilisons le mode fenêtre. Les commandes de base sont

- `begwind` : initialisation du mode fenêtre
- `window` : construction de fenêtres proportionnelles
- `makewind` : construction de fenêtres non proportionnelles
- `setwind` : sélection d'une fenêtre
- `endwind` : termine le mode fenêtre

5.5.1 La procédure `window`

Nous considérons le problème de programmation linéaire suivant :

$$\max f(x) = 14x_1 + \alpha x_2 + 13x_3$$

avec

$$\begin{cases} x_1 + x_2 + x_3 & \leq & 40 \\ 6x_1 + 9x_2 + 5x_3 & \leq & 280 \\ x_i & \geq & 0 \end{cases}$$

```
/*
** Qprog1.prg
**
*/

new;
```

```
library pgraph;

alpha = seqa(5,0.1,251);
N = rows(alpha);
x = zeros(N,3);
fmax = zeros(N,1);

let bounds = {0 1e200};
let C[2,3] = 1 1 1 6 9 5;
let D = 40 280;

sv = ones(3,1)/3;

i = 1;
do until i > rows(alpha);
  r = 14|alpha[i]|13;
  {xi,u1,u2,u3,u4,retcode} = QProg(sv,zeros(3,3),r,0,0,-C,-D,bounds);
  x[i,.] = xi';
  fmax[i] = r'xi;
  i = i + 1;
endo;


#IFUNIX

let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
wxy = WinOpenPQG(v,''XY Plot'',''XY'');
call WinSetActive(wxy);

#ENDIF


graphset;
  begwind;
  window(2,2,0);
    _pnum = 2; _paxht = 0.25; _pnumht = 0.25;
    fonts(''simplex simgrma'');

  setwind(1);
    _pltype = 6|1|3; _pcolor = 3|6|2;
    xtics(5,30,5,5);
    ytics(-10,50,10,2);
    xlabel(''\202a\201'');
    xy(alpha,x);

  setwind(2);
    ylabel(''x]1[(\202a\201)'');
    _plctrl = -10; _pstype = 5;
    xy(alpha,x[.,1]);

  setwind(3);
    ylabel(''x]2[(\202a\201)'');
```

```
    _plctrl = 50; _pstype = 9;
    xy(alpha,x[.,2]);

  setwind(4);
    ylabel(''x]3[(\202a\201)'');
    _plctrl = -10; _psymsiz = 6;
    _pmsgstr = ''Ici, les symboles sont\000plus grands'';
    _pmsgctl = -10~50~0.45~-40~1~13~0 |
               10~-5~0.25~0~1~12~0 ;
    xy(alpha,x[.,3]);

  graphprt(''-c=1 -cf=qprog1.ps'');

  endwind;


#IFUNIX

call WinSetActive(1);

#ENDIF
```

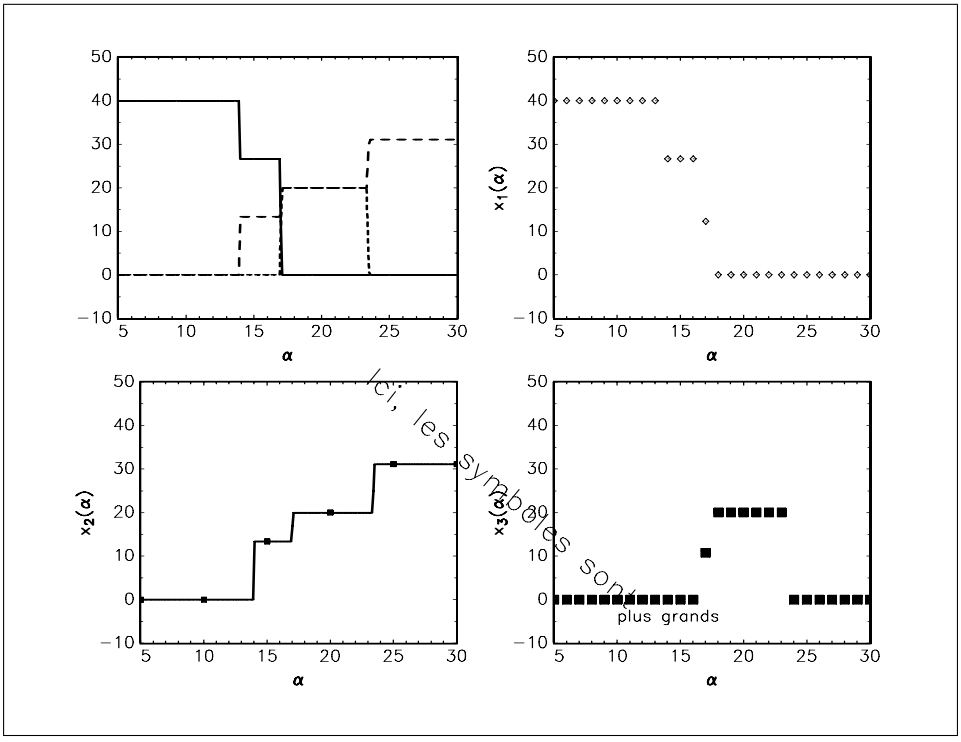


FIG. 3 – Le mode multi-fenêtres

5.5.2 La procédure `makewind`

voir l'exemple de la formation CCF (page 42).

6 Un mot sur les chaînes de caractères

▶ Manipulation I : initialiser une chaîne de caractères

```
/*
** string1.prg
*/

new;

output file = string1.out reset;

str = ''Il fait beau aujourd'hui'';
print str;

l = strlen(str);
print l;

str1 = strsect(str,1,7);
print str1;

str = str $+ ''\L'' $+ str $+ ''\t'' $+ str;
print str;

output off;
```

```
Il fait beau aujourd'hui
       24.000000
Il fait
Il fait beau aujourd'hui
Il fait beau aujourd'hui Il fait beau aujourd'hui
```

▶ Manipulation II : la commande `ftos`

```
/*
** string2.prg
*/

new;

output file = string1.out on;

str = ''Il fait beau aujourd'hui'';
l = strlen(str);
str1 = ''La longueur de la chaine de caracteres \34'' $+ str $+ ''\34 est egale a'';
str1 = str1 $+ ftos(l,'' %lf caracteres.'',4,3);
print str1;

output off;
```

```
La longueur de la chaine de caracteres ''Il fait beau aujourd'hui'' est egale a 24
.000 caracteres.
```

▶ Manipulation III : la différence avec un vecteur de caractères

```
/*
** string3.prg
*/

new;

outwidth 256;
output file = string3.out reset;

let x = ''Janvier'' ''Fevrier'' ''Mars'' ''Avril'';

print x;
print $x;

let x = Janvier Fevrier Mars Avril;
print $x;

y = x $+ x;
print $y;

y = '''' $+ x $+ x;
print y;

y = reshape(y,3,5);
print y;

output off;



  1.6373109e-306
  1.6373108e-306
  9.5694382e-315
  2.3004960e-312

         Janvier
         Fevrier
            Mars
           Avril

         JANVIER
         FeVRIER
            MARS
           AVRIL

        JANVIERJ
        FeVRIERF
        MARSMARS
        AVRILAVR

  JANVIERJANVIER
```

```
  FeVRIERFeVRIER
        MARSMARS
       AVRILAVRIL

  JANVIERJANVIER   FeVRIERFeVRIER         MARSMARS       AVRILAVRIL   JANVIERJANVIER
  FeVRIERFeVRIER         MARSMARS       AVRILAVRIL   JANVIERJANVIER   FeVRIERFeVRIER
        MARSMARS       AVRILAVRIL   JANVIERJANVIER   FeVRIERFeVRIER         MARSMARS
```

▶ Manipulation IV : les tableaux de chaînes de caractères

```
/*
** string4.prg
*/

new;

output file = string4.out reset;

sa = '''' $+ zeros(500,4);

sa[250,3] = ''Formation ritme'';

let string sa[2,3] = ''pentium'' ''intel'' ''unix'' ''NT'' ''PC'' ''Gauss'';
print sa;

sa = sa $~(sa[.,3] $+ ''-'' $+ sa[.,3]);
print sa;

print (sa .$== ''unix'');

output off;
```

```
         pentium            intel             unix
              NT               PC            Gauss

         pentium            intel             unix        unix-unix
              NT               PC            Gauss      Gauss-Gauss

      0.00000000       0.00000000        1.0000000       0.00000000
      0.00000000       0.00000000       0.00000000       0.00000000
```

7 La gestion des dates

GAUSS possède différentes commandes pour gérer les dates :

```
Time and Date Functions

-------------------------------------------------------------------------------
       date              Returns current date as { yy,mm,dd,ss }.
       datestr           Returns current date as mm/dd/yy string.
       dayinyr           Returns day number of current date.
```

```
       etdays            Computes difference between two times (days).
       ethsec            Computes difference between two times (hsecs).
       etstr             Converts elapsed time to string.

       hsec              Elapsed time since midnight, in hsecs.
       time              Returns current system time.
       timestr           Formats time as hh:mm:ss string.
-------------------------------------------------------------------------------
Use hsec to time segments of code. For example,
                et = hsec;
                x = y*y;
                et = hsec - et;
will time the multiplication operator.



Year 2000 Considerations
------------------------

GAUSS for Unix 3.2.38 has been certified as Year 2000-ready.

However, since GAUSS is a programming language, it is entirely possible for
the user to introduce data into a program that is not Year 2000-compliant,
particularly when specifying date information.

GAUSS provides various functions to manipulate dates as set forth
below. TO ENSURE CORRECT RESULTS, 4 DIGIT YEAR DATA MUST BE USED.

  Date Function                 Purpose
 ============================================================================
  d = date;              returns current date in a 4-element column vector
  d = datestr(d);        formats date as a string: mo/dy/yr
  d = datestrymd(d);     formats date as a string: yyyymmdd
  d = datestring(d);     formats date as a string: mo/dy/yyyy
  etdy = etdays(ds,de);  elapsed time between dates in days
  eths = ethsec(ds,de);  elapsed time between dates in 100th's of seconds
  daynum = dayinyr(dt);  day number in year of a date

   a. Function datestr(d): datestr(d) formats a date using only the last
two digits of the year. Since a date formatted using only two digits for
the year results in a loss of information, it is up to the user to
guarantee that dates so formatted are not misused leading to incorrect
program results. This applies also to date information that is exported to
or imported from other programs.

datestr is provided *only* for backwards compatibility with legacy programs.

   b. Sysstate Base Year Toggle: In early versions of GAUSS for Unix and
Windows, the date function returned (in the year element) the number of
years since 1900 instead of the actual 4 digit year. This has been fixed
and GAUSS now returns the actual 4 digit year. However, a Sysstate Option
has been added to provide users backwards compatibility with code that may
```

```
have been written with these earlier versions. (See Sysstate, Case 30, Base
Year Toggle). If you have programs that assume the year element contains
the number of years since 1900, we recommend that you modify your code to
use the actual 4 digit year. In the interim, you may use the Sysstate Base
Year Toggle so that your programs will run as expected.
```

Voici quelques exemples de manipulation de dates :

```
new;

output file = date1.out reset;

d0 = {2000,01,01,0};
d = date;
et = etdays(d,d0);

print et;
print dayinyr(d);

print datestr(d);
print datestrymd(d);
print datestring(d);

et = ethsec(d,d0);
print etstr(et);

output off;


101.00000
265.00000
9/22/99
19990922
9/22/1999
100 days  12 hours  33 minutes  4.92 seconds
```

Dans l'exemple suivant, nous illustrons l'utilisation de la commande `sysstate`.

```
new;

output file = date2.out reset;

oldsysdate = sysstate(30,1);

d = date;
print datestring(d);

oldsysdate = sysstate(30,0);
d = date;
print datestring(d);

call sysstate(30,oldsysdate);
d = date;
print datestring(d);
```

```
output off;


9/22/1999
9/22/0099
9/22/1999
```

Pour mesurer les temps de calcul, nous utilisons les commandes `hsec` ou `ethsec`. Voyons un premier exemple.

```
new;
cls;

t0 = hsec;

x = rndn(300,300);
x = invpd(x'x);

t1 = hsec;

output file = date3.out reset;

print ftos(t1-t0,''Temps de calcul : %lf centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %*.*lf centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %*.*lE centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %#*.*lG centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %*.*lG centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %- *.*lf centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %- *.*lE centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %- #*.*lG centie‘mes de secondes'',4,2);
print ftos(t1-t0,''Temps de calcul : %- *.*lG centie‘mes de secondes'',4,2);

print (t1-t0)/100;

format /rd 18,6;
print (t1-t0)/100;
t = t1-t0;
print etstr(t);
print etstr(5600);
print etstr(256250);
print etstr(125256250);
print etstr(10125256250);

output off;

format /mat /on /mb1 /ros 16,8;
format /sa /on /mb1 /ros 16,-1;
format /str /off /mb1 /ros 16,-1;


Temps de calcul : 263.00 centie‘mes de secondes
Temps de calcul : 263.00 centie‘mes de secondes
Temps de calcul : 2.63E+002 centie‘mes de secondes
Temps de calcul : 2.6E+002 centie‘mes de secondes
```

```
Temps de calcul : 2.6E+002 centie‘mes de secondes
Temps de calcul :  263.00 centie‘mes de secondes
Temps de calcul :  2.63E+002 centie‘mes de secondes
Temps de calcul :  2.6E+002 centie‘mes de secondes
Temps de calcul :  2.6E+002 centie‘mes de secondes
       2.6300000
          2.630000
2.63 seconds
56.00 seconds
42 minutes  42.50 seconds
14 days  11 hours  56 minutes  2.50 seconds
1171 days  21 hours  42 minutes  42.50 seconds
```

8 Les entrées et sorties

8.1 La procédure save

Syntaxe :

```
Purpose:    Saves symbols to a disk file.

Format:     save [path = dpath] x, [lpath = ] y;

Input:      dpath       literal or ^string, the default path to use for this
                        and all subsequent save commands, until changed by
                        another path = subcommand.
            x           the symbol to be saved.  The file will have the same
                        name as the symbol, with the proper extension for that
                        type of symbol added.
            lpath       literal or ^string, a local path and filename to be
                        used for a particular symbol.  The path overrides any
                        default path and the filename overrides the name of
                        the object.  The extension cannot be overridden.

save can be used to save matrices, strings or string arrays, procedures,
functions and keywords. Procedures, functions and keywords must be compiled and
resident in memory before they can be save'd. The file extensions given for
different symbol types are:
            procedure   .fcg          matrix         .fmt
            function    .fcg          string         .fst
            keyword     .fcg          string array   .fst
```

La commande `save` permet donc de sauvegarder différents types de données (matrices ou tableaux de chaînes de caractères).

```
new;

x = rndn(10,10);

/* Cre'ation d'un fichier x.fmt */
save x;

/* Cre'ation d'un fichier avec un nom plus complexe */
save un-fichier-contenant-des-nombres-aleatoires = x;
```

```
/* Cre'ation d'un fichier avec un nom plus complexe */
save un-fichier-contenant-des-nombres-aleatoires = x;

x = ''une chaine de caracteres'';
save un-fichier-contenant-une-chaine-de-caracteres = x;

let string x[1,2] = ''une chaine de caracteres'' ''une deuxieme chaine de caracteres'';
save un-fichier-contenant-un-tableau-de-chaines-de-caracteres = x;

output file = save.out reset;

fnames = filesa(''*.f*'');
print fnames;

output off;


UN-FICHIER-CONTENANT-DES-NOMBRES-ALEATOIRES.fmt
UN-FICHIER-CONTENANT-UN-TABLEAU-DE-CHAINES-DE-CARACTERES.FST
UN-FICHIER-CONTENANT-UNE-CHAINE-DE-CARACTERES.FST
           X.fmt
```

8.2 Les procédures `load`, `loadf`, `loadm` et `loads`

▶ Chargement d'une matrice (*.fmt*)

```
new;

output file = load1.out reset;

x = rndn(5,1);
save x;
save y = x;
load y;
load nom_de_variable = x;
print x~y~nom_de_variable;

output off;


     -0.95877623      -0.95877623      -0.95877623
       1.0390460        1.0390460        1.0390460
     -0.36588347      -0.36588347      -0.36588347
     -0.83261601      -0.83261601      -0.83261601
       1.5656592        1.5656592        1.5656592
```

▶ Chargement d'une chaîne de caractères (*.fst*)

```
(gauss) xx = ''une chaine de caracteres'';
(gauss) save xx;    /* x.fst */
(gauss) load xx;    /* xx.fmt */

C:\GAUSS\ritme\LOAD2.PRG(7) : error G0014 : 'XX.FMT' : File not
```

```
found (gauss) load xx.fst;

load XX.FST
       ^
C:\GAUSS\ritme\LOAD3.PRG(5) : error G0008 : '.FST' : Syntax error

1 error(s)
(gauss) load xx = xx.fst
(gauss) loads xx
```

8.3 Les procédures `fopen, f*`

- `fopen` : ouverture d'un fichier (mode ascii ou binaire)
- `close` : fermeture d'un fichier
- `eof` : teste la fin d'un fichier
- `fgets`, `fgetsa`, `fgetsat`, `fgetst` : lecture des données
- `fputs`, `fputst` : écriture d'un fichier
- `fseek` : positionne le pointeur
- `ftell` : position du pointeur
- `fcheckerr`, `fclearerr`, `fflush`, `fstrerror`

▶ Premier exemple :

```
/*
** fopen
*/

new;

output file = fopen.out reset;

let string x[2,1] = ''Il fait beau'' ''Il ne fait pas beau'';

fw = fopen(''un_simple_essai'',''w'');  /* ouverture en mode e'criture */

i = 1;
do until i > 250;
  call fputs(fw,x);
  i = i + 1;
endo;

str = getf(''un_simple_essai'',0);
print strsect(str,1,50);

let string x[2,1] = ''Il fait beau\L'' ''Il ne fait pas beau\L'';

fw = fopen(''un_simple_essai'',''w'');  /* ouverture en mode e'criture */

i = 1;
do until i > 250;
  call fputs(fw,x);
  i = i + 1;
endo;
```

```
str = getf(''un_simple_essai'',0);
print strsect(str,1,50);

output off;


Il fait beauIl ne fait pas beauIl fait beauIl ne f
Il fait beau
Il ne fait pas beau
Il fait beau
I
```

▶ Deuxième exemple :

```
/*
** fseek
*/

new;

output file = fseek.out reset;

fr = fopen(''un_simple_essai'',''r'');  /* ouverture en mode lecture */

pos = ftell(fr);
print pos;
str1 = fgets(fr,50);
print str1;
print strlen(str1);
pos = ftell(fr);
print pos;

str2 = fgets(fr,50);
print str2;
print strlen(str2);
pos = ftell(fr);
print pos;

str3 = fgets(fr,50);
print str3;
print strlen(str3);
pos = ftell(fr);
print pos;

print strlen(str1)+strlen(str2)+strlen(str3);

closeall;

output off;


      0.00000000
Il fait beau
```

```
       13.000000
       14.000000
Il ne fait pas beau

       20.000000
       35.000000
Il fait beau

       13.000000
       49.000000
       46.000000
```

Deuxième partie

NIVEAU II — PROGRAMMATION AVANCÉE GAUSS

9 La récursivité

Cette notion est très importante. Pourtant, elle n'est documentée nulle part et beaucoup d'utilisateurs GAUSS ne sont pas informés de l'existence de cette possibilité. Pour bien comprendre la récursivité, nous considérons l'exemple de la factorielle. Nous avons

$$n! = n \times (n-1)!$$

avec $1! = 1$. En notant $f(n) = n!$, nous avons alors une fonction de récurrence. Celle-ci peut être facilement implémentée dans GAUSS et nous vérifions bien que la procédure `factorial` donne le même résultat que l'opérateur `!`.

```
new;

proc factorial(n);

  if n > 1;
    retp(n*factorial(n-1));
  else;
    retp(1);
  endif;

endp;

output file = recurs1.out reset;

factorial(5);
print 5!;

output off;
```

```
  120.0000
  120.0000
```

Remarque 6 *Attention, la mise en place de la récursivité en informatique n'est pas toujours très naturelle et ne correspond pas forcément à la logique humaine. En effet, la structure de la procédure pourrait nous faire penser que n prend les valeurs successives 5, 4, 3, 2 et enfin 1. Or, cela fonctionne à l'envers. Pourquoi ? Tout simplement parce que nous pouvons calculer $f(n)$ à partir de $f(n-1)$ mais non le contraire.*

```
new;

proc factorial(n);
  local y;

  if n > 1;
    y = n*factorial(n-1);
  else;
    y = 1;
```

```
  endif;

  print n~y;

  retp(y);

endp;

output file = recurs2.out reset;

call factorial(5);

output off;
```

```
    1.0000     1.0000
    2.0000     2.0000
    3.0000     6.0000
    4.0000    24.0000
    5.0000   120.0000
```

La propriété de récursivité est particulièrement utile pour la construction des arbres (notament en finance). Considérons par exemple la fonction suivante

$$\begin{aligned} f_1(x) &= p\frac{1}{1+x} + (1-p)\frac{1}{1-x} \\ f_2(x) &= p\frac{1}{1+p\frac{1}{1+x}+(1-p)\frac{1}{1-x}} + (1-p)\frac{1}{1-p\frac{1}{1+x}-(1-p)\frac{1}{1-x}} \\ f_3(x) &= \ldots \end{aligned}$$

Le programme suivant illustre comment nous utilisons la récursivité pour calculer $f_n(x)$.

```
new;

proc tree(x,n,p);
  local y;

  if n > 1;
    y = p ./ (1+tree(x,n-1,p)) + (1-p) ./ (1-tree(x,n-1,p));
  else;
    y = p ./ (1+x) + (1-p) ./ (1-x);
  endif;

  retp(y);

endp;

output file = recurs2.out reset;

tree(5,1,0.25);
n = 1;
x = 5;
p = 0.25;
y = p ./ (1+x) + (1-p) ./ (1-x);
```

```
print y;

tree(5,2,0.25);
n = 2;
x = 5;
p = 0.25;
y = p ./ ( 1 + p ./ (1+x) + (1-p) ./ (1-x) )
    + (1-p) ./ (1 - p ./ (1+x) - (1-p) ./ (1-x) );
print y;

output off;


   -0.1458
   -0.1458
    0.9472
    0.9472
```

Il faut savoir que la récursivité est généralement gourmante en temps de calcul. Elle peut aussi être une solution inadaptée, surtout lorsque nous sommes aussi intéressés par les valeurs intermédiaires prises par la fonction. Néanmoins, elle permet dans certains cas d'écrire un code élégant et d'exploiter de plus les fonctionnalités des opérateurs $E \times E$ de GAUSS.

```
new;
library pgraph;

proc tree(x,n,p);
  local y;

  if n > 1;
    y = p ./ (1+tree(x,n-1,p)) + (1-p) ./ (1-tree(x,n-1,p));
  else;
    y = p ./ (1+x) + (1-p) ./ (1-x);
  endif;

  retp(y);

endp;

x = seqa(-5,.1,101);
p = seqa(0,0.1,11)';

y = tree(x,10,p);
```

Remarque 7 *La procédure* `sortmc` *de GAUSS exploite la propriété de récursivité pour effectuer un tri multi-colonnes.*

10 La commande sysstate

Cette commande permet d'obtenir les valeurs des paramètres système et de les modifier. La syntaxe générale est :

```
{rets...} = sysstate(case,x) ;
```

L'argument `case` peut prendre différentes valeurs :

1	Version Information	
2	GAUSS System Paths	**EXE** file location
3		**loadexe** path
4		**save** path
5		**load**, **loadm** path
6		**loadf**, **loadp** path
7		**loads** path
8	Complex Number Toggle	
9	Complex Trailing Character	
10	Printer Width	
11	Auxiliary Output Width	
12	Precision	
13	Lu Tolerance	
14	Cholesky Tolerance	
15	Screen State	
16	Automatic **print** Mode	
17	Automatic **lprint** Mode	
18	Auxiliary Output	
19	**print** Format	
21	Imaginary Tolerance	
22	Source Path	
24	Dynamic Library Directory	
27	Missings Comparison Toggle	
30	Base Year Toggle	

Cette commande est peu utilisée dans la pratique, pourtant elle se révèle très importante. Voyons quelques exemples pratiques d'application. Nous cherchons à créer un fichier temporaire dans lequel nous voulons stocker certaines informations. Une première solution peut être la suivante :

```
screen off;
outwidth 256;
TempFile = tempname(0,0,0);

output file = ^TempFile reset;

/*
**> Affichage de rsultats intermdiaires dans un fichier temporaire
**
*/

print ''Affichage de rsultats intermdiaires dans un fichier temporaire'';

output off;
```

Dans un premier temps, nous supprimons l'affichage à l'écran. Ensuite, nous modifions la largeur de la sortie auxiliaire. Enfin, nous créons un fichier temporaire et stockons les informations dans ce fichier. Le problème du code précédent est de modifier de façon **définitive** (et non **temporaire**) les paramètres système. C'est pourquoi le code suivant est plus approprié :

```
oldwidth = sysstate(11,0);
```

```
call sysstate(11,256);

{oldstate,oldfile} = sysstate(18,0);
if oldstate;
  output off;
endif;

oldscreen = sysstate(15,0);
screen off;

TempFile = tempname(0,0,0);

output file = ^TempFile reset;

/*
**> Affichage de rsultats intermdiaires dans un fichier temporaire
**
*/

print ''Affichage de rsultats intermdiaires dans un fichier temporaire'';

output off;

output file = ^oldfile;

if oldstate;
  output on;
else;
  output off;
endif;

call sysstate(11,oldwidth);

if oldscreen;
  screen on;
endif;
```

Dans ce cas, la sortie auxiliare n'est pas interompue et le système est remis dans son état d'origine (aspect de l'écran, largeur de la sortie auxiliaire, etc.).

Nous pouvons connaître l'emplacement du programme **gauss.exe** avec l'instruction

```
sysstate(2,0) ;
```

Voici un exemple d'utilisation de `sysstate(1,0)` qui peut parfois se révéler très utile :

```
proc sysstate1(args);
  local vi,version;

  vi = sysstate(1,0);

  version = 1000*vi[1] + 100*vi[2] + vi[3];   /* 3.2.35 ===> version = 3235 */

  if version > 3200;
```

```
    /* algorithme utilisant les fonctions de GAUSS ver. > 3.2.20 */

  else;

    /* algorithme n'utilisant pas ces fonctions */

  endif;

  retp(rets);
endp;
```

11 Compilation et exécution

■ Pour exécuter un programme, nous utilisons la commande `run` :

```
run nom_du_programme
```

Cette commande permet en fait de compiler dans un premier temps le programme et dans un deuxième temps d'interpréter le pseudo-code. Elle est donc équivalente à

```
compile programme.prg ;
run programme.gcg ;
```

■ La compilation au préalable d'un programme est vraiment intéressante, si celui-ci est important et si le programme est souvent utilisé. Dans ce cas, nous pouvons *charger* le code compilé dans un autre programme avec la commande `use`.

■ Notons aussi que la commande `saveall` permet de sauvergarder la table de symbole de GAUSS, qui peut être rechargée avec la commande use.

■ Pour exécuter des programmes en mode batch, nous devons fermer la session GAUSS en cours à partir du programme lui-même. Il faut alors utiliser la commande `system`.

12 Les directives de compilation

L'utilisation des directives de compilation est particulière. Elles permettent surtout de développer des bibliothèques compatibles pour différentes plateformes et de prendre en compte les caractéristiques de chaque système d'exploitation. Voyons un premier exemple avec la directive de compilation **#define** :

```
new;

#define e 2.7182818

print e;

x = e + e;
print x;

show;
```

```
       2.7182818
       5.4365636
```

```
X
       8 bytes at [01c40020]       1,1       MATRIX

65536 bytes program space, 0% used
62849008 bytes workspace, 62849000 bytes free
1 global symbols, 1500 maximum, 1 shown
```

Cet exemple montre que le symbole `e` est une autre écriture de la valeur numérique 2.7182818. Mais ce n'est pas une variable (elle ne fait pas partie de la table des symboles).

Les directives de compilation s'avèrent finalement indispensables pour l'élaboration d'une bibliothèque multi-plateforme. Les bibliothèques d'**Aptech**, comme la bibliothèque intrinsèque **GAUSS**, les utilise fréquement, ce qui permet d'utiliser le même code source pour la version **Windows** et pour la version **Unix**.

```
#ifUNIX
  print ''systme Unix'';
#else
  print ''autre systme'';
#endif


#ifDOS
  print ''systme Dos'';
#else
  print ''autre systme'';
#endif


#ifOS2WIN
  print ''systme Os2-Windows'';
#else
  print ''autre systme'';
#endif


#ifDLLCALL
  print ''systme supportant les DLLs'';
#else
  print ''systme ne supportant pas les DLLs'';
#endif


#ifLIGHT
  print ''version Light de GAUSS'';
#else
  print ''version professionnelle de GAUSS'';
#endif


#ifREAL
  print ''version GAUSS-386'';
#else
  print ''version GAUSS-386i'';
```

```
#endif


#ifCPLX
  print ''version GAUSS-386i'';
#else
  print ''version GAUSS-386'';
#endif
```

Remark 1 *Il existe aussi une directive de compilation* ***#include*** *qui permet d'inclure dans un programme un autre fichier source. Ceci est notamment très utile lorsque plusieurs programmes possèdent une partie commune.*

13 Les procédures

13.1 Génération de nombre aléatoires gaussiens multidimensionnels

Nous considérons la construction d'une procédure de génération de nombre aléatoires gaussiens multidimensionnels basée sur l'algorithme de Cholesky. La procédure `rndmn` est classique, mais présente un double intérêt :

1. La procédure teste si la matrice de covariance est PD. Pour cela, nous autorisons la décomposition de Cholesky même si la matrice n'est pas PD avec la commande `trap 1,1 ;` puis nous testons si une erreur est survenue avec la commande `scalerr`.
2. Si la matrice n'est pas PD, nous affichons un message d'erreur avec la commande `errorlog` et nous renvoyons un message d'erreur qui correspond ici à une valeur manquante.

Remarque 8 *Pourquoi ne pas utiliser la commande* ***print*** *à la place de la commande* ***errorlog*** *?*

Nous allons utiliser l'exemple numérique plusieurs fois. Pour cela, nous créons un fichier contenant les données du problème qui sera appelée avec la directive de compilation **#include**.

La procédure

```
proc (1) = rndmn(mu,SIGMA,N);
  local K,u,Pchol,y;
  local oldtrap;

  K = rows(mu);
  u = rndn(K,N);

  oldtrap = trapchk(1);
  trap 1,1;
  Pchol = chol(SIGMA)';
  trap oldtrap,1;
  if scalerr(Pchol);
    ERRORLOG ''erreur: SIGMA n'est pas une matrice PD.'';
    retp(error(0));
  endif;

  y = mu + Pchol*u;

  retp(y');
endp;
```

Le fichier rndmn.inc

```
let sigma[3,3] = 3 0.25 0.5
                 0.25 5 -0.9
                 0.5 -0.9 2;
mu = 10|15|17;
```

L'exemple

```
new; library ritme;

#include rndmn.inc;

y = rndmn(mu,SIGMA,1000);

output file = rndmn1.out reset;

print ''mu = '' mu;
print ''estimation = '' meanc(y);
print;
print ''SIGMA = '' SIGMA;
print ''estimation = '' vcx(y);

output off;


mu =
       10.000000
       15.000000
       17.000000
estimation =
       10.046476
       15.062753
       17.082670

SIGMA =
       3.0000000       0.25000000       0.50000000
      0.25000000        5.0000000      -0.90000000
      0.50000000      -0.90000000        2.0000000
estimation =
       3.0379946       0.22053032       0.61400550
      0.22053032        5.0675993      -0.91023170
      0.61400550      -0.91023170        2.1592710
```

13.2 Une première approche de l'analyse numérique par blocs

GAUSS est très apprécié pour ses qualités matricielles. On lui repproche parfois de ne pas savoir traiter les bases de données. Ceci est inexact, puisque tous les modules GAUSS d'APTECH sont construits pour les bases de données. Cela suppose quelques connaissance sur l'analyse numérique par blocs.

La procédure

```
/*
**> dat_cov
**
*/
```

```
proc (4) = dat_cov(dataset,vars);
  local m,sigma,cov,cor;
  local fh,str,names,indx,k,c,nobs,xx,sum,nr,x,xc;

  /* Nous verifions que la base de donnees existe */

  open fh = ^dataset for read;
  if fh == -1;
    str = ''error: la procedure dat_cov ne peut pas ouvrir la base de donnees '' $+ dataset $+ ''.'';
    errorlog str;
  endif;

  /* Construction des indices */

  {names,indx} = indices(dataset,vars);

  k = rows(indx);    /* Nombre de variables     */
  c = colsf(fh);     /* Nombre de colonnes de la base de donnees */
  nobs = rowsf(fh);  /* Nombre de lignes de la base de donnees   */
  xx = 0;            /* initialisation de xx    */
  sum = 0;           /* initialisation de sum   */

  /*
  :: Calcul du nombre de lignes lues pour l'acces sequentiel
  ::
  :: getnr utilise deux variables globales __row et __rowfac
  :: ainsi que la procedure maxvec
  */

  nr = getnr(6,c);

  /*
  :: Boucle principale pour calculer la moyenne
  */

  do until eof(fh);
    x = readr(fh,nr);
    x = submat(x,0,indx);
    sum = sum + sumc(x);
  endo;

  m = sum/nobs;

  /*
  :: Boucle principale pour calculer la matrice de covariance
  */

  call seekr(fh,1);   /* repositionne le pointeur a la premiere ligne */

  do until eof(fh);
    x = readr(fh,nr);
    x = submat(x,0,indx);
```

```
    xc = x - m';
    xx = xx + moment(xc,0);
  endo;

  cov = xx / (nobs-1);
  sigma = sqrt(diag(cov));
  cor = cov ./ sigma ./ sigma';

  retp(m,sigma,cov,cor);
endp;
```

L'exemple

```
new; library ritme;

#include rndmn.inc;

output file = dat_cov.out reset;

y = rndmn(mu,SIGMA,20000);

nom_de_la_base = ''rndmn'';

call saved(y,nom_de_la_base,0 $+ ''Y'');
m1 = meanc(y);
cov1 = vcx(y);

let names = Y1 Y2 Y3;
{m2,sigma2,cov2,cor2} = dat_cov(nom_de_la_base,names);

print m1~m2;
print cov1;
print cov2;
print cov1-cov2;

nom_des_variables = getname(nom_de_la_base);

create pointeur_de_fichier = ^nom_de_la_base with ^nom_des_variables,3,8;

/*
call close(pointeur_de_fichier);
open pointeur_de_fichier = ^nom_de_la_base for append;
*/

i = 1;
do until i > 100;
  y = rndmn(mu,SIGMA,20000);
  call writer(pointeur_de_fichier,y);
  i = i + 1;
endo;

/*
**> N'oubliez surtout pas de fermer la base !!!
```

```
*/
closeall;

{m3,sigma3,cov3,cor3} = dat_cov(nom_de_la_base,1|2|3);

print ''Nombre de lignes dans la base de donne'es : '' 100*20000;
{fnames,finfo} = fileinfo(nom_de_la_base $+ ''.dat'');
print ''Taille (nombre de Mo)'' finfo[8]/1000/1000;

print m3;
print cov3;
print cor3;

output off;


       9.9654977        9.9654977
       14.989501        14.989501
       17.001857        17.001857

       3.0341654       0.29596508       0.51390103
      0.29596508        5.0052495      -0.90585223
      0.51390103      -0.90585223        2.0231067

       3.0341654       0.29596508       0.51390103
      0.29596508        5.0052495      -0.90585223
      0.51390103      -0.90585223        2.0231067

  4.4408921e-016       0.00000000       0.00000000
      0.00000000       0.00000000       0.00000000
      0.00000000       0.00000000       0.00000000


Nombre de lignes dans la base de donne'es :        2000000.0
Taille (nombre de Mo)       48.000232


       9.9989120
       14.999213
       17.000604

       3.0005896       0.25552792       0.49882462
      0.25552792        4.9886559      -0.89638329
      0.49882462      -0.89638329        1.9991219

       1.0000000      0.066045508       0.20366900
     0.066045508        1.0000000      -0.28384571
      0.20366900      -0.28384571        1.0000000
```

Remarque 9 *Introduire la commande* `rndseed` *et parler des algorithmes de génération de nombre aléatoires.*

14 Les pointeurs

voir la formation CCF (page 23).

```
(gauss) f

(0) : error G0159 : Wrong number of parameters
Currently active call: F
```

Supposons que `f` soit une fonction en ligne ou une procédure. Exécutez la ligne de commande " `f` " conduit systématiquement à une erreur (voir ci-dessus). Cela est tout à fait normal puisque nous avons une erreur de syntaxe. Comment pouvons-nous alors échanger des procédures ? En définissant un pointeur avec l'opérande `&`. Quelques remarques :

1. `&f` n'est pas une procédure.
2. `&f` est un scalaire.
3. `&f` est en fait une pseudo-adresse correspondant à l'emplacement de la procédure `f`.

```
new;

output file = point1.out reset;

pointeur1 = &f;
pointeur2 = &g;
pointeur3 = &g|&f;
pointeur4 = &pointeur3;
pointeur5 = &pointeur4;


print pointeur1;
print;
print pointeur2;
print;
print pointeur3;
print;
print pointeur4;
print;
print pointeur5;

x = 0;
print &x;

show;

output off;

fn f(x) = sin(x);

proc g(x);
  retp(sin(x));
endp;
```

```
       44.000000

       88.000000


       88.000000
       44.000000

       110.00000

       132.00000
       176.00000

F
     104 bytes at [01c40020]       1=1       FUNCTION   local refs
G
     120 bytes at [01c40088]       1=1       PROCEDURE  local refs
POINTEUR1
       8 bytes at [01c40100]       1,1       MATRIX
POINTEUR2
       8 bytes at [01c40108]       1,1       MATRIX
POINTEUR3
      16 bytes at [01c40110]       2,1       MATRIX
POINTEUR4
       8 bytes at [01c40120]       1,1       MATRIX
POINTEUR5
       8 bytes at [01c40128]       1,1       MATRIX
X
       8 bytes at [01c40130]       1,1       MATRIX

65536 bytes program space, 0% used
62849008 bytes workspace, 62848728 bytes free
8 global symbols, 1500 maximum, 8 shown
```

Pour déclarons une fonction comme argument dans un procédure, nous utilisons son pointeur et nous déclarons ce pointeur comme une procédure avec la syntaxe `local ?? : proc`.

```
new;

proc EvaluateFunction(f,x0);
  local f:proc;
  local y;

  y = f(x0);

  retp(y);
endp;

fn f1(x) = sin(x);
fn f2(x) = cos(x);

output file = point2.out reset;
```

```
EvaluateFunction(&f1,0);
EvaluateFunction(&f1,pi);
EvaluateFunction(&f2,0);
EvaluateFunction(&f2,pi);

output off;
```

```
      0.00000000
  1.2246064e-016
       1.0000000
      -1.0000000
```

15 Les variables globales externes

15.1 La communication des informations entre les procédures d'une même bibliothèque

Cette communication se fait à l'aide de variables globales externes. Pour cela, nous les définissons à l'aide d'un fichier `.dec` que nous ajoutons à la bibliothèque. Dans l'exemple qui suit, nous cherchons le point fixe d'une fonction. Pour cela, nous employons la procédure `Qnewton`.

Le fichier *ritme.dec*

```
declare matrix _FixedPoint_f;
```

Le fichier *ritme.ext*

```
external matrix _FixedPoint_f;
```

La procédure

```
proc FixedPoint(f,x0);
  local xp,fp,gp,retcode;

  _FixedPoint_f = f;
  {xp,fp,gp,retcode} = Qnewton(&_FixedPoint,x0);

  if retcode /= 0;
    retp(error(0));
  endif;

  if fne(fp,0);
    retp(error(0));
  endif;

  retp(xp);
endp;

proc _FixedPoint(x);
  local f,y;
  f = _FixedPoint_f;
  local f:proc;

  y = f(x)-x;
```

```
  retp(sumc(y^2));
endp;
```

L'exemple

```
new; library ritme;

output file = fp1.out reset;

proc f1(x);
  retp(cos(x));
endp;

FixedPoint(&f1,0);

proc f2(x);
  retp(cos(x) + sin(x)^2 - sqrt(x) );
endp;

FixedPoint(&f2,0);

output off;


================================================================================
 QNewton Version 3.2.39                                 9/24/1999   2:47 pm
================================================================================

return code =    0
normal convergence

Value of objective function        0.000000

Parameters    Estimates    Gradient
----------------------------------------
P01              0.7391        0.0000
Number of iterations        6
Minutes to convergence     0.00000
      0.73908513

================================================================================
 QNewton Version 3.2.39                                 9/24/1999   2:47 pm
================================================================================

return code =    0
normal convergence

Value of objective function    50904.114938

Parameters    Estimates    Gradient
----------------------------------------
P01            212.0403        0.0003
```

```
Number of iterations       37
Minutes to convergence     0.00050
                .
```

15.2 La distinction entre les informations indispensables et les informations complémentaires

GAUSS autorise l'utilisation de plus de 1000 entrées et sorties pour définir une procédure. Considérons la procédure `optmum`. Celle-ci est définie de la façon suivante. Elle appele en fait une autre procédure qui comprend 27 entrées et 6 sorties. Un appel direct à la procédure `_optmum` est donc un véritable casse tête pour mémoriser la syntaxe et surtout l'ordre des variables. Or `optmum` ne demande que 2 entrées et fournit 4 retours :

```
{x,f,g,retcode} = optmum(&fcnt,x0);
```

Cette syntaxe est très facile à mémoriser. L'idée est donc d'utiliser des variables globales (**d'entrée**) qui prennent des valeurs par défaut et que nous pouvons modifier à souhait. De même, toutes les sorties ne sont pas d'égale importance, par exemple la matrice hessienne finale. Dans ce cas, nous définissons des variables globales (**de sortie**) que nous pouvons récupérer.

```
proc (4) = optmum(fnct,x0);
    local x,f0,g,retcode;
    local Lopfhess,Lopitdta,LLoutput;

#ifUNIX
    LLoutput = __output /= 0;
#else
    LLoutput = __output;
#endif

    { x,f0,g,retcode,Lopfhess,Lopitdta } = _optmum(fnct,x0,
      _opalgr,
      _opdelta,
      _opdfct,
      _opditer,
      _opgdmd,
      _opgdprc,
      _opgrdh,
      _opgtol,
      _ophsprc,
      _opkey,
      _opmbkst,
      _opmdmth,
      _opmiter,
      _opmtime,
      _opmxtry,
      _opparnm,
      _oprteps,
      _opshess,
      _opstep,
      _opstmth,
      _opusrch,
      _opusrgd,
      _opusrhs,
```

```
      LLoutput,
      __title
      );

    _opfhess = Lopfhess;
    _opitdta = Lopitdta;

    retp(x,f0,g,retcode);
endp;
```

▶ Voyons un exemple avec les moindres carrés ordinaires.

Le fichier *ritme.dec*

```
declare matrix _FixedPoint_f;
declare matrix _mco_DW;
declare matrix _mco_ddl;
declare matrix _print = 1;
```

La procédure

```
proc (4) = mco(y,x);
  local Nobs,k,ddl,xxinv,xy,beta,u,scr,sigma;
  local Mcov,stderr,student,pvalue,du,DW,parnm;

  Nobs = rows(x);
  k = cols(x);
  ddl = Nobs - k;

  xxinv = invpd(x'x);
  xy = x'y;
  beta = xxinv*xy;

  u = y-x*beta;
  scr = sumc(u^2);

  sigma = sqrt(scr/ddl);

  Mcov = (sigma^2) * xxinv;
  stderr = sqrt(diag(Mcov));
  student = beta ./ stderr;
  pvalue = 2*cdftc(abs(student),ddl);

  du = trimr(u - lag1(u),1,0);
  DW = sumc(du^2)/sumc(u^2);

  _mco_DW = DW;
  _mco_ddl = ddl;

  if _print;
    call Header(''mco'','''',0);
    print ftos(Nobs,''Nombre d'observations : %lf'',5,0);
    print ftos(k   ,''Nombre de variables :   %lf'',5,0);
    print;
    print ftos(ddl ,''Degres de libertes :    %lf'',5,0);
    print;
```

```
    print;
    print ''-------------------------------------------------------------------------------'';
    print ''variable      coefficient       ecart-type        t-student           pvalue'';
    print ''-------------------------------------------------------------------------------'';

    if ( __ALTNAM /= 0 ) and ( rows(__ALTNAM) == K);
      parnm = __ALTNAM;
    else;
      parnm = 0 $+ ''X'' $+ ftocv(seqa(1,1,K),1,0);
    endif;

    call printfmt(parnm~beta~stderr~student~pvalue,0~1~1~1~1);

  endif;

  retp(beta,stderr,Mcov,u);
endp;
```

L'exemple

```
new; library ritme;

rndseed 123;

Nobs = 5000;
K = 5;
sigma = 1;

beta = seqa(1,1,K);
x = rndu(Nobs,K);
u = rndn(Nobs,1)*sigma;
y = x*beta + u;

output file = mco1.out reset;

call mco(y,x);

_print = 0;
{beta,stderr,Mcov,u} = mco(y,x);

ddl = _mco_ddl;
print ddl;

_print = 1;
let __altnam = ''var01'' ''constant'' ''age'' ''TAILLE'' ''VaR00005'';
{beta,stderr,Mcov,u} = mco(y,x);

output off;


================================================================================
 mco                                                    9/24/1999   3:45 pm
================================================================================
```

```
Nombre d'observations :  5000
Nombre de variables :       5

Degres de libertes :     4995


-------------------------------------------------------------------------------
variable      coefficient       ecart-type        t-student           pvalue
-------------------------------------------------------------------------------
      X1         1.069213      0.043706907        24.463249   6.2999322e-125
      X2        1.9385819      0.044306461        43.753933                0
      X3        3.0070166      0.044612526        67.402967                0
      X4        3.9783595      0.044034141        90.347158                0
      X5        5.0346595      0.044076162        114.22636                0
       4995.0000
===============================================================================
 mco                                                    9/24/1999   3:45 pm
===============================================================================
Nombre d'observations :  5000
Nombre de variables :       5

Degres de libertes :     4995


-------------------------------------------------------------------------------
variable      coefficient       ecart-type        t-student           pvalue
-------------------------------------------------------------------------------
   var01         1.069213      0.043706907        24.463249   6.2999322e-125
constant        1.9385819      0.044306461        43.753933                0
     age        3.0070166      0.044612526        67.402967                0
  TAILLE        3.9783595      0.044034141        90.347158                0
VaR00005        5.0346595      0.044076162        114.22636                0
```

16 Les bibliothèques

Une bibliothèque GAUSS est un outil de gestion de procédures (elle est assez différente d'une bibliothèque FORTAN ou C). Elle est constituée de plusieurs composantes :

- un fichier *.lcg* qui gére le contenu de la bibliothèque et la localisation des procédures ;
- des fichiers *.dec* et *.ext* de déclaration de variables globales externes ;
- des fichiers *.src* qui contiennent les procédures.

Ce système a un avantage indéniable : **Le code source est disponible**. De plus, il faut bien comprendre qu'une bibliothèque GAUSS est bien plus qu'un système de localisation de procédures et de variables globales. C'est aussi un système de gestion automatisée de compilation :

Seuls les symboles référencés dans un programme sont compilés

```
(gauss) new; (gauss) library ritme (gauss) show


65536 bytes program space, 0% used
62849008 bytes workspace, 62849008 bytes free
0 global symbols, 1500 maximum, 0 shown
```

```
(gauss) x = 0;
(gauss) _print = 1
(gauss) show

X
       8 bytes at [01c40020]       1,1       MATRIX
_PRINT
       8 bytes at [01c40028]       1,1       MATRIX

65536 bytes program space, 0% used
62849008 bytes workspace, 62848992 bytes free
2 global symbols, 1500 maximum, 2 shown
(gauss)
```

- Les procédures sources et les fichiers de déclaration sont généralement localisés dans le répertoire *src*. Cela pose cependant un problème, et il est préférable de localiser l'ensemble des procédures dans un répertoire qu'APTECH n'utilise pas, par exemple *src/ritme*.
- La recherche des symboles est effectuée par l'Autoloader (voir section 4 du manuel de GAUSS) qui s'appuie sur la déclaration de la variable **src_path** du fichier *gauss.cfg* (ou des chemins définis par la commande `sysstate`).
- Afin d'éviter les confusions et les pertes de temps liées à la recherche des symboles, il est préférable de définir le chemin lors de la construction de la bibliothèque :

```
lib ritme *.src /a
```

- La commande `lib` supporte plusieurs options :

lib

```
Purpose:    Builds and updates library files.

Format:     lib library [file] [flag1 flag2 ...];

Input:      library  literal, name of library.
            file     optional literal, name of source file to be updated/added.
            flags    optional literal, controls operation of library update.

Path type flags:                        Library update type flags:
 /addpath (default) add paths where      /update (default) update symbol
          none, expand relative paths.           info for specified file.
 /repath  reset all paths.               /build  update symbol information
 /leavepath leave paths untouched.               for entire library.
 /nopath  remove all paths.
                                        Temporary files location flags:

Symbol info type flags:                  /tmp  (default) use directory listed by
 /strong (default) use strongly                TMP_PATH var (usu. on ramdisk).
         typed symbol entries.           /disk use directory listed by
 /weak   save no type information.             LIB_PATH config. var.

If the filenames in a library include a full path, the compilation process is
faster because no directory searches are needed during autoloading.
```

▶ Un exemple de priorité entre les procédures

Nous considérons plusieurs fichiers *libuser.src*, *lib1.src* et *lib2.src* contenant une procédure portant le nom

cumsumc. Le code de cumsumc est :

```
proc cumsumc(x);

    print ''lib1'';  /* ou ''lib2'' ou ''user'' */

    if rows(x) == 1;
        retp(x);
    else;
        retp( recserar(x,x[1,.],ones(1,cols(x)) ) );
    endif;

endp;
```

Nous ajoutons cette procédure au bibliothèques user, lib1 et lib2. Voyons comment se comporte GAUSS pour le choix de la procédure cumsumc (présente dans quatre bibliothèques gauss, user, lib1 et lib2).

```
(gauss) new
(gauss) call cumsumc(0)
user
(gauss) library lib1
(gauss) call cumsumc(0)
user
(gauss) /* bibliothe‘que USER de’sactive’ */
(gauss) call cumsumc(0)
user
(gauss) library gauss
(gauss) call cumsumc(0)
user
(gauss) new
(gauss) library gauss
(gauss) call cumsumc(0)
(gauss) library lib1
(gauss) call cumsumc(0)
(gauss) new
(gauss) library lib1
(gauss) call cumsumc(0)
lib1
(gauss) library lib2,lib1
(gauss) call cumsumc(0)
lib1
(gauss) new
(gauss) library lib2,lib1
(gauss) call cumsumc(0)
lib2
(gauss) new
(gauss) library gauss,lib2,lib1
(gauss) call cumsumc(0)
(gauss) new
(gauss) library lib1,lib2
(gauss) call cumsumc(0)
lib1
(gauss)
```

Les priorités sont en fait les suivantes (manuel GAUSS page 65) :

- **Autodelete ON**
  1. user library
  2. user-specified libraries
  3. gauss library
  4. Current directory, then **src_path** for files with a **.g** extension
- **Autodelete OFF**
  1. user library
  2. user-specified libraries
  3. gauss library

17 La construction de l'aide en ligne

La construction de l'aide en ligne est assez aisée. En fait, le browser de GAUSS cherche dans les bibliothèques actives la procédure demandée. Lorsque celle-ci est trouvée, le browser affiche le fichier source contenant la procédure. Il suffit donc de commenter le code pour créer l'aide en ligne. De plus, le browser scanne le fichier et repère l'ensemble des lignes commençant par **>. Si le nom de la procédure est placée à la suite de cette marque, le browser pointera directement sur cette ligne. Considérons par exemple la bibliothèque **MVT** disponible à l'adresse Internet :

`http ://gro.creditlyonnais.fr`

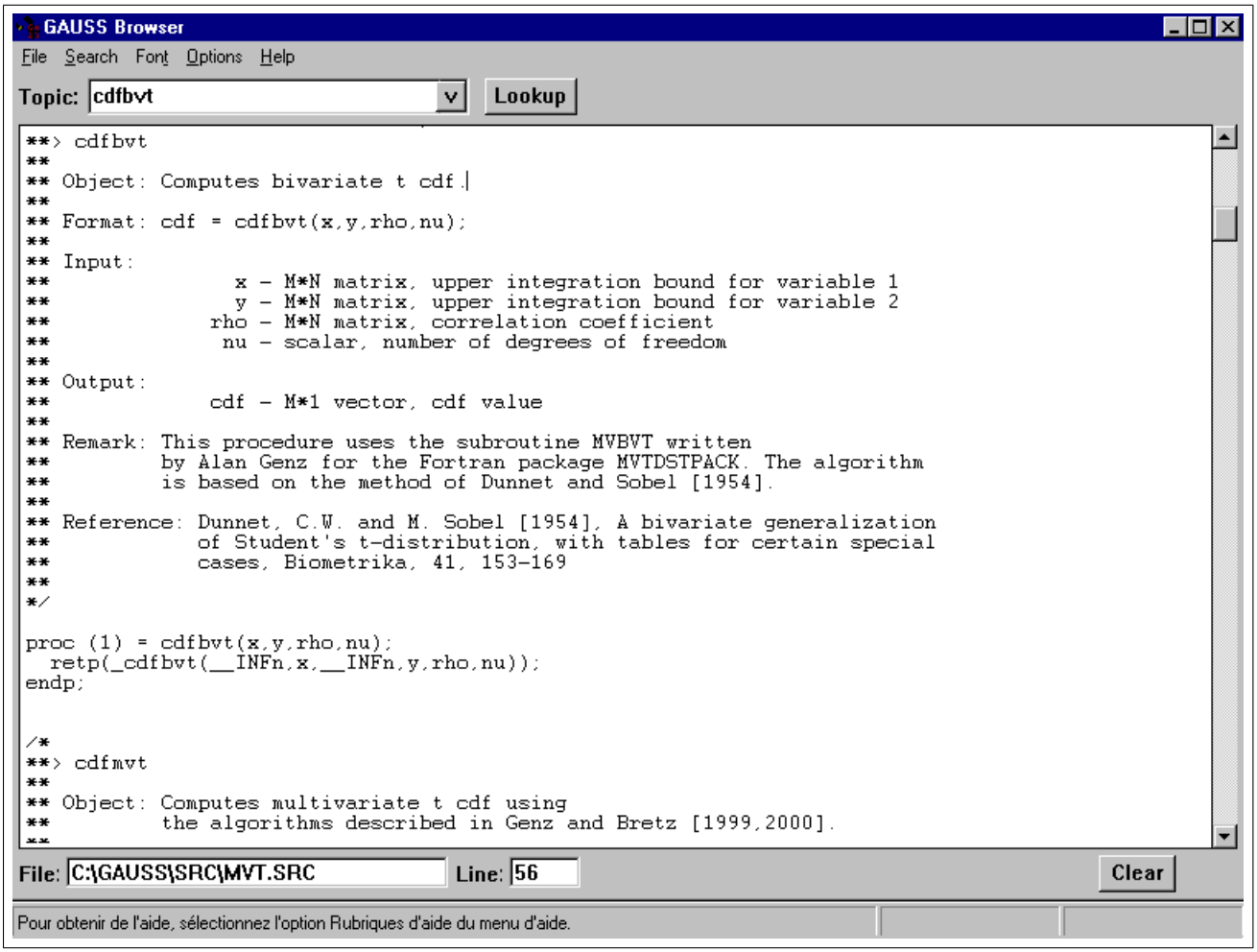
Le fichier source *mvt.src* contient plusieurs procédures, par exemple une procédure `cdfbvt` pour calculer la cdf de la distribution **t** à deux dimensions. Cette procédure est précédée par le commentaire suivant :

```
/*
**> cdfbvt
**
** Object: Computes bivariate t cdf.
**
** Format: cdf = cdfbvt(x,y,rho,nu);
**
** Input:
**               x - M*N matrix, upper integration bound for variable 1
**               y - M*N matrix, upper integration bound for variable 2
**             rho - M*N matrix, correlation coefficient
**              nu - scalar, number of degrees of freedom
**
** Output:
**             cdf - M*1 vector, cdf value
**
** Remark: This procedure uses the subroutine MVBVT written
**         by Alan Genz for the Fortran package MVTDSTPACK. The algorithm
**         is based on the method of Dunnet and Sobel [1954].
**
** Reference: Dunnet, C.W. and M. Sobel [1954], A bivariate generalization
**            of Student's t-distribution, with tables for certain special
**            cases, Biometrika, 41, 153-169
**
*/

proc (1) = cdfbvt(x,y,rho,nu);
```

```
  retp(_cdfbvt(__INFn,x,__INFn,y,rho,nu));
endp;
```

Lorsque nous cherchons l'aide en ligne de `cdfbvt` avec le browser, nous obtenons alors l'écran suivant.



Notons qu'il peut être utile de créer un chapeau afin de répertorier l'ensemble de procédures du fichier ainsi que les numéros de ligne des aides en lignes. Dans ce cas, il suffit de double cliquer sur le numéro de ligne pour aller directement à l'aide en ligne correspondante.

```
/*
** mvt.src - MVT library.
**
**  Procedure                    Object                                    Line
** ============================================================================
**  mvtset                       Initializes the MVT library                 20
**
**  cdfbvt                       Computes bivariate t cdf                    56
**
**  cdfmvt                       Computes multivariate t cdf                 87
**
**  cdfmvtnc                     Computes non-central multivariate t cdf   153
```

```
**
*/
```

18 Inclure des graphiques GAUSS dans TeX ou MSWord

La première solution est de sauvegarder les graphiques au format PostScript avec l'instruction `graphprt`. Par exemple,

```
graphprt(''-c=1 -cf=graph1.ps'');
```

Cette ligne de commande est utilisée avant l'appel de la procédure graphique (`xy`, `bar`, `histp`, etc.) en mode normal. En mode fenêtres, elle est placée avec la commande `endwind`.

Dans **MSWord**, il suffit d'inclure le graphique avec les commandes **Insertion/Image/A partir du fichier...**

Dans **Scientific Word**, on procède de la même façon (**File/Import Picture**).

Plus généralement, dans **TeX**, on utilise le package *psfig.tex* (`\usepackage{psfig}` et `\usepackage[dvips]{graphicx}`) et la commande `\includegraphics` :

```
\begin{figure}[tbph]
\includegraphics[width = 4.8793in, height = 3.7196in]{corr1}
\caption{Relationship between the parameter $\rho^{\mathbf{S}}$ and
the survival time correlation}
\label{fig-corr1}
\end{figure}
```

Remarque 10 *Les paramètres de conversion sont définis dans le fichier pqgrun.cfg (sous le répertoire \gauss). Les trois variables les plus importantes sont* `cvt_orient`, `cvt_margins` *et* `cvt_color`.

```
################################################################################
#
# cvt_orient values:
#
#      L - Landscape
#      P - Portrait

cvt_orient = P

################################################################################
#
# cvt_margins values:
#
#      <left>,<right>,<top>,<bottom>  - comma-delimited list of decimal
#                                       numbers. Units are in inches.

cvt_margins = 0,0,0,0

################################################################################
#
# cvt_color values:
#
#      BW   - black and white
#      GRAY - gray scale color
#      RGB  - color

cvt_color = BW
```

```
##############################################################################
```

Remarque 11 *Nous pouvons sauvegarder les graphiques dans d'autres formats avec le logiciel* ***PlayW*** *fourni avec* ***GAUSS****.*

19 Manipulation des variables buffer

Pour construire une variable buffer, nous utilisons la procédure `vput`. Pour lister le contenu d'une variable buffer, nous employons la procédure `vlist`. Pour lire une variable, nous utilisons `vread`. Enfin, l'extraction se fait avec la procédure `vget`.

```
new;

x = 0;
x = vput(x,rndn(3,3),''u'');
x = vput(x,''il est 10:00'',''time'');
x = vput(x,rndn(9,1),''v'');

vlist(x); print;

u = vread(x,''u'');

u = vread(x,''U'');

vlist(x); print;

{u,x} = vget(x,''U'');

vlist(x);
```

20 Utilisation des commandes varput/varputl et varget/vargetl

Les commandes `varput` et `varget` permettent de définir et d'accéder à des variables globales en utilisant leurs noms.

– Considérons un premier exemple. Après avoir défini les variables globales `y1` et `y2`, nous faisons une copie de ces deux matrices avec la commande `varget`. Pour savoir si un symbole existe, nous utilisons la commande `typecv`. Dans l'exemple, `x3` n'est pas défini.

```
new;

y1 = rndu(10,2);
y2 = rndu(15,3);

x1 = varget(''y1'');

if typecv(''y2'') /= error(0);
  x2 = varget(''y2'');
endif;

if typecv(''y3'') /= error(0);
  x3 = varget(''y3'');
endif;
```

```
print x1;
print x2;
print x3;
```

– Nous pouvons aussi définir des variables globales avec la commande `varput`. Néanmoins, il n'est pas possible d'utiliser directement ces variables dans le programme car cela produit une erreur à la compilation. En revanche, ces variables sont directement accessibles en mode commande.

```
new;

call varput(rndu(10,2),''y1'');
call varput(rndu(10,2),''y2'');

x1 = varget(''y1'');
x2 = varget(''y2'');

print x1;
print x2;

/*
print y1;
print y2;
*/
```

– Pour utiliser directement les variables globales générées par `varput`, il faut les déclarer au préalable.

```
new;

declare matrix y1,y2;

call varput(rndu(10,2),''y1'');
call varput(rndu(10,2),''y2'');

x1 = varget(''y1'');
x2 = varget(''y2'');

print x1;
print x2;

print y1;
print y2;
```

– Voici un exemple plus élaboré. On génère 50 matrices carrées aléatoires dont la dimension est elle aussi aléatoire. Une fois cette simulation faite, nous calculons le déterminant de chacune des matrices.

```
new;

i = 1;
do until i > 50;
  r = floor(100*rndu(1,1));
  x = rndn(r,r);
  name = ftos(i,''x*.*lf'',1,0);
  call varput(x,name);
  i = i + 1;
endo;
```

```
name = ''x'' $+ ftocv(seqa(1,1,50),1,0);

i = 1;
do until i > rows(name);
  x = varget(name[i]);
  print det(x);
  i = i + 1;
endo;
```

Remarque 12 *Ces procédures sont très utiles pour certaines applications Monte Carlo ou économétriques. Par exemple, lorsque j'ai programmé les procédures relatives aux modèles VARMA pour la bibliothèque* ***TSM****, certaines procédures renvoyaient une multitude de matrices. C'est par exemple le cas des fonctions d'impulsion. Il m'a semblé judicieux dans ce cas de les stocker avec la procédure* ***varput****, ce qui permet à l'utilisateur de les récupérer facilement avec la procédure* ***varget****.*

```
/*
**> arma_orthogonal
**
** Purpose: Compute the responses to orthogonal impulses (and the interim
**          multipliers) of a Vector ARMA process
**
** Format:  call arma_orthogonal(beta,p,q,SIGMA,N);
**
** Input:   beta - np*1 vector, the coefficients of a Vector ARMA process
**             p - scalar, the autoregressive order
**             q - scalar, the moving average order
**         SIGMA - K*K matrix, the SIGMA matrix
**             N - scalar, the maximum order of the impulse function
**                         to be evaluated
**
** Output:
**          The responses to orthogonal impulses and the accumulated impulses
**          function (or the interim multipliers) can be read with the varget
**          command. For the responses to orthogonal impulses of order i, the
**          generic string name is ''IMPULSE'' followed by the index i. For exam-
**          ple, if we want the responses to orthogonal impulses of order 20,
**          we write x = varget(''IMPULSE20''). For the interim multipliers, the
**          generic string name is ''_IMPULSE''.
**
*/
```

Remarque 13 *Pour définir et accéder à des variables locales, les commandes correspondantes sont* ***varputl*** *et* ***vargetl****. Elles s'emploient de la même façon que* ***varget*** *et* ***varput****.*

21 La gestion des bases de données

Il existe deux formats de bases de données **GAUSS**. L'ancien format v89 est facilement reconnaissable, puisqu'il nécessite deux fichiers pour définir une base de données : un fichier header *.dht* qui contient les informations sur les variables et notament les labels, et un fichier *.dat* qui contient les données. Avec le nouveau format v96, le fichier *.dht* disparaît. Nous déclarons le format par défaut dans le fichier *gauss.cfg*.

21.1 Création d'une base de données

Il existe plusieurs façons pour créer une base de données. La méthode la plus simple est d'utiliser la procédure `saved`.

21.1.1 La procédure `saved`

Les données sont stockées dans une matrice et sont transformées en une base de données avec la commande `saved`. La seule chose à faire est de spécifier les noms des variables.

```
new;

let data[10,5] = Peter   M 10  London   25
                 Mary    F 5   Paris    20
                 Eva     F 4   Cergy    17
                 Theo    M 5   Cergy    18
                 Pauline F 6   Langon   15
                 Arthur  M 3   Langon   15
                 Antoine M 1.5 Langon    9
                 Elise   F 13  Bordeaux 30
                 Pierre  M 0.5 Rennes    6
                 Eva     F 2   Rouen     9;

call printfmt(data,0~0~1~0~1);

x = data[.,5] ./ data[.,3];

let vnames = Nom Sexe Age Ville Poids P/Age;

call saved(data~x,''uneBase'',vnames);
```

– Si les données sont stockées au format ASCII dans un fichier, nous pouvons employer la commande `load`.

Les données ASCII

```
Peter   M 10  London   25
Mary    F 5   Paris    20
Eva     F 4   Cergy    17
Theo    M 5   Cergy    18
Pauline F 6   Langon   15
Arthur  M 3   Langon   15
Antoine M 1.5 Langon    9
Elise   F 13  Bordeaux 30
Pierre  M 0.5 Rennes    6
Eva     F 2   Rouen     9
```

Le programme GAUSS

```
new;
load data[10,5] = uneBase.asc;
x = data[.,5] ./ data[.,3];
let vnames = Nom Sexe Age Ville Poids P/Age;
call saved(data~x,''uneBase'',vnames);
```

21.1.2 Les commandes `create` et `writer`

Lorsque les données sont trop importantes, c'est-à-dire lorsque elles ne peuvent être contenues dans la mémoire vive, nous pouvons utiliser **le mode séquentiel**. Dans l'exemple suivant, nous créons une base de

données avec la commande `create`. Remarquez l'utilisation de l'opérateur `^` qui permet de spécifier le contenu d'une variable. Ensuite, nous remplissons cette base de données avec 1000 copies des données. Enfin, nous fermons la base de données avec la commande `close`. Plus rigoureusement, nous fermons le canal correspondant (la variable `f` est un scalaire).

```
new;
load data[10,5] = uneBase.asc;
x = data[.,5] ./ data[.,3];
data = data~x;
let vnames = Nom Sexe Age Ville Poids P/Age;

create f = uneBase2 with ^vnames,6,8;

print f;

i = 1;
do until i > 1000;
  call writer(f,data);
  i = i + 1;
endo;

f = close(f);

print f;
```

- Considérons un Monte Carlo **ENORME**. Nous cherchons à analyser la convergence de l'estimateur des moindres carrés ordinaires.

Remarque 14 ***Ne lancez pas ce programme. Il nécessite un espace disque de plus de 74505 Giga Octets !***

```
new;

proc rndt(r,c,nu);
  retp(cdftci(rndu(r,c),nu));
endp;


Nobs = 100;

X = 1000*rndn(Nobs,5);
let beta = 1 2 3 4 5;
sigma = 25;
nu = 3;

create f1 = MC1 with beta,5,8;
create f2 = MC2 with beta,5,8;

i = 1;
do until i > 1e12;
  y = x*beta + rndn(Nobs,1)*sigma;
  bols = y/x;
  call writer(f1,bols');
  y = x*beta + rndt(Nobs,1,nu)*sigma;
  bols = y/x;
```

```
  call writer(f2,bols');

  i = i + 1;
endo;

closeall;
```

21.2 Visualisation de la base de données

Pour visualiser une base de données, nous employons la procédure `datalist`. Remarquons que la base de données *UneBase2* contient des données **numériques** et des données **caractères**. Dans les anciennes version de **GAUSS**, il n'était pas possible de typer directement les variables. Néanmoins, il était possible de procéder à un typage de façon indirecte en adoptant la convention suivante :

1. **les noms des variables dites numériques sont des chaînes de caractères majuscules ;**
2. **les noms des variables dites caractères sont des chaînes de caractères minuscules ;**

Remarque 15 *Il est très important d'utiliser les guillemets pour définir les noms des variables :*

```
let vnames = ''nom'' ''sexe'' ''AGE'' ''ville'' ''POIDS'' ''P/AGE'';
```

Sinon, les noms des variables sont en majuscules. Par exemple, la ligne de commande

```
let vnames = nom sexe AGE ville POIDS P/AGE;
```

est équivalente à celle-ci

```
let vnames = NOM SEXE AGE VILLE POIDS P/AGE;
```

```
new;
load data[10,5] = uneBase.asc;
x = data[.,5] ./ data[.,3];
data = data~x;
let vnames = ''nom'' ''sexe'' ''AGE'' ''ville'' ''POIDS'' ''P/AGE'';

create f = uneBase2 with ^vnames,6,8;

print f;

i = 1;
do until i > 1000;
  call writer(f,data);
  i = i + 1;
endo;

f = close(f);

dataset = ''UneBase2'';

datalist ^dataset;
```

Résultat de la procédure `datalist`

```
---------------------- c:\gauss\gro\ritme/unebase2.dat ----------------------

                nom      sexe       AGE     ville     POIDS     P/AGE
```

```
 #1     Peter         M      10.0    London      25.0      2.50
 #2      Mary         F      5.00     Paris      20.0      4.00
 #3       Eva         F      4.00     Cergy      17.0      4.25
 #4      Theo         M      5.00     Cergy      18.0      3.60
 #5   Pauline         F      6.00    Langon      15.0      2.50
 #6    Arthur         M      3.00    Langon      15.0      5.00
 #7   Antoine         M      1.50    Langon      9.00      6.00
 #8     Elise         F      13.0   Bordeaux     30.0      2.31
 #9    Pierre         M     0.500    Rennes      6.00      12.0
#10       Eva         F      2.00     Rouen      9.00      4.50
#11     Peter         M      10.0    London      25.0      2.50
#12      Mary         F      5.00     Paris      20.0      4.00
#13       Eva         F      4.00     Cergy      17.0      4.25
#14      Theo         M      5.00     Cergy      18.0      3.60
#15   Pauline         F      6.00    Langon      15.0      2.50
#16    Arthur         M      3.00    Langon      15.0      5.00
#17   Antoine         M      1.50    Langon      9.00      6.00
#18     Elise         F      13.0   Bordeaux     30.0      2.31
#19    Pierre         M     0.500    Rennes      6.00      12.0
#20       Eva         F      2.00     Rouen      9.00      4.50
--------------------------------------------------------------------------------

PgDn  PgUp  Home  End              Ctrl-   Ctrl-   ?  Esc  [Esc]
```

Remarque 16 *On peut ne visualiser qu'une partie des variables, par exemple*

```
datalist uneBase2 nom sexe poids ;
```

21.3 Manipulation des variables

Il existe de nombreuses commandes et procédures pour manipuler les variables d'une base de données. En voici quelques unes.

- `getname` permet de connaître les noms des variables.
- `indices` permet d'obtenir le nom et l'indice d'une variable donnée.

```
new;

name = getname(''uneBase2'');
print $name;
print;

{name,indx} = indices(''uneBase2'',''age'');
call printfmt(name~indx,0~1);
print; print;

{name,indx} = indices(''uneBase2'',''age''|''P/AGE'');
call printfmt(name~indx,0~1);
print; print;

{name,indx} = indices(''uneBase2'',3|4);
call printfmt(name~indx,0~1);
print;
```

```
            nom
           sexe
            AGE
          ville
          POIDS
          P/AGE

    AGE               3

    AGE               3
  P/AGE               6


    AGE               3
  ville               4
```

– Pour créer des variables globales portant le même nom que les variables de la base de données, nous utilisons la procédure `makevars`. Pour créer des matrices à partir des noms de symboles, nous employons la procédure `mergevar`. Remarquez que la procédure `vartype` permet de connaître le typage implicite des variables d'une base de données.

```
new;

dataset = ''uneBase2'';
call makevars(loadd(dataset),getname(dataset),getname(dataset));

x = mergevar(''age sexe'');
y = mergevar(''age''|''sexe''|''poids''|''nom'');

names = getname(dataset);
call printfmt(names~vartype(names),0~1);


    nom               0
   sexe               0
    AGE               1
  ville               0
  POIDS               1
  P/AGE               1
```

21.4 Lecture des données

21.4.1 La procédure `loadd`

Pour lire une base de données, nous utilisons la procédure `loadd`.

```
new;

dataset = ''uneBase2'';
data = loadd(dataset);

call printfmt(data,vartype(getname(dataset))');
```

– Pour sélectionner certaines colonnes, nous combinons les commandes `loadd` et `indices` (ou `indcv`).

```
new;

dataset = ''uneBase2'';

{name,indx} = indices(dataset,''age''|''sexe'');
data = loadd(dataset);
data = data[.,indx];

data = submat(loadd(dataset),0,indcv(''age''|''sexe'',getname(dataset)));
```

21.4.2 La commande `readr`

Nous pouvons aussi charger des données avec la commande **`readr`**. Pour cela, il est nécessaire d'ouvrir un canal avec l'instruction **`open ... for read`**. Pour connaître le nombre de lignes et colonnes de la base de données, nous employons les commandes **`rowsf`** et **`colsf`**.

```
new;

dataset = ''uneBase2'';

{name,indx} = indices(dataset,''age''|''sexe'');

open f = ^dataset for read;

r = rowsf(f);
data = readr(f,r);
data = data[.,indx];

f = close(f);
```

- Bien sûr, l'intéret principal de la commande **`readr`** est la lecture séquentielle, particulièrement adaptée aux bases de données volumineuses. Dans l'exemple qui suit, nous créons une base de données de nombres aléatoires χ^2_ν de dimension $ns \times nr$ (nombre de simulations $\times$ nombre de réplications). La génération de ces nombres aléatoires utilisent la relation qui existe entre une variables chi-deux et des variables gaussiennes indépendantes. Si ν est élevé, il se peut que nous disposions pas de mémoire suffisante pour simuler l'ensemble des nombres aléatoires en une seule fois. En effet, cela revient à simuler $ns \times nr \times \nu$ nombres aléatoires gaussiens. Par exemple, si $ns = 1000$, $nr = 100$ et $\nu = 100$, la matrice **`rndn(ns*nr,nu)`** prend plus de 76 Mo. Une solution est alors d'utiliser le mode séquentiel.

```
new;

dataset = ''huge'';

ns = 1000;
nr = 10;
nu = 100;

create f = ^dataset with chi,nr,8;

i = 1;
do until i > ns;
  u = sumc(rndn(nu,nr)^2);
  call writer(f,u');
  i = i + 1;
```

```
endo;

f = close(f);

x = 0;
x2 = 0;

open f = ^dataset for read;
do until eof(f);
  u = readr(f,10);
  x = x + sumc(u);
  x2 = x2 + sumc(u^2);
endo;

f = close(f);

mean = x/ns;
std = sqrt(x2/ns - mean^2);

print ftos(nu,''Moyenne theorique : %lf'',7,4);
print ftos(sqrt(2*nu),''Ecart-type theorique : %lf'',7,4);

print mean~std;
```

21.5 Manipulation et transformation des données

Pour cela, nous pouvons utiliser le mode **dataloop**. Les principales commandes sont `make`, `select`, `recode`, `keep`, `code`, `delete`, `drop`, `extern`, `lag`. Voyons quelques exemples pour comprendre l'utilisation de ce mode de programmation.

- A partir de la base de données sci, nous créons une autre base de données *dd1*. Cette dernière comprend trois variables : `sex`, `job` et `cnst`. Les deux premières sont issues de la base *sci*. La dernière est créée avec les commandes `extern` et `vector`. Notons aussi que la base contient uniquement les observations telles que la variable `enrol` soit différente de la valeur manquante (commande `select`).

```
m = {.};
i = 1;
dataloop sci dd1;
    extern m, i;
    vector cnst = i;
    select enrol $/= m;
    keep sex job cnst;
endata;
```

- La base de données *freqdata* contient 4 variables : `AGE`, `PAY`, `sex` et `WT`. On crée deux autres variables `Y1` et `COUNT`. On retient les observations numérotées 13 à 45. Dans la base finale, nous éliminons les variables `sex` et `WT`.

```
disable;      /* because the data set contains missing data */

base = 1;     /* base for counter variable */

dataloop freqdata dd1;

    /* create counter variable -- the data for each iteration of the  */
```

```
    /*                          read loop is in x_x                */
    extern base;
    make count = seqa(base,1,rows(x_x));
    # base = base + rows(x_x);

    make y1 = age^2 ;
    select 13 < count and count < 45;
    drop wt sex;
endata;

datalist dd1;
```

– Un exemple classique de codage (commandes `code` et `recode`).

```
mv = {.};

dataloop freqdata dd1;
    extern mv;
    code cage with
        1  for  age >= 6,
        0  for age < 6,
        mv for age $== mv;

    recode  sex with
       1  for sex $== ''M'',
       0  for sex $== ''F'',
       mv for sex $== mv;
    keep cage sex pay age;
endata;
```

– Un autre exemple de codage.

```
/*  This example censors the variable job if job < 2.7  */

dataloop sci dd1;
    vector k = 1;
    recode job with
       0 for job < 2.7;
    select female /= k;
    keep job sex female k;
endata;
```

Remarque 17 ***Les opérations du mode dataloop sont aussi disponibles en mode programmation (voir le paragraphe sur le mode séquentiel).***

Recoding Data

```
-------------------------------------------------------------------------
       code              Code the data in a vector by applying a logical set
                         of rules to assign each data value to a category.
       dummy             Creates a dummy matrix, expanding values in a vector
                         to rows with ones in columns corresponding to true
                         categories and zeros elsewhere.
       dummybr           Similar to dummy.
       dummydn           Similar to dummy.
```

```
      recode            Similar to code, but leaves the original data in
                        place if no condition is met.
      substute          Similar to recode, but operates on matrices.
      subscat           Simpler version of recode, but uses ascending bins
                        instead of logical conditions.
-------------------------------------------------------------------------------

code, recode, and subscat allow the user to code data variables and operate
on vectors in memory.  substute operates on matrices, and dummy and
dummybr and dummydn create matrices.
```

21.6 Les formats externes

21.6.1 Les fichiers ASCII

Nous avons déjà vu qu'il était facile de charger un fichier ASCII en mémoire avec la commande `load` et ennsuite de le convertir en une base de données **GAUSS** avec la procédure `saved`. Néanmoins, cette méthode n'est plus valable lorsque le fichier ASCII est volumineux.

21.6.1.1 Le programme ATOG **GAUSS** est livré avec un programme qui permet de convertir les fichiers ASCII en bases de données au format GAUSS. Ce programme est `ATOG` et se trouve sous le répertoire *\gauss*. Voici un exemple d'utilisation :

```
INPUT dataset.asc;
INVAR $ name # age $ sex # pay;
OUTPUT atog;
OUTTYP 8;
OUTVAR $ name # age $ sex # pay;
```

La base de données ASCII est donc *dataset.asc*. **ATOG** crée une base de données GAUSS *atog.dat*. Les variables de *dataset.asc* sont `name` (variable caractère), `age` (variable numérique), `sex` (variable caractère) et `pay` (variable numérique). La précision pour la sauvergarde est 8 bytes (`OUTTYP 8`). Enfin, la commande `OUTVAR` spécifie les variables exportées dans la base de données GAUSS.

Remarque 18 *Vous pouvez consulter le manuel pour plus de précisions concernant l'utilisation d'ATOG.*

21.6.1.2 Les commandes `fopen`, `f*` Les commandes d'entrée et sortie sont certainement la méthode la plus puissante pour gérer les bases de données ASCII. Elles permettent de prendre en compte n'importe quel format, les irrégularités. Par exemple, dans la bibliothèque **GAUSS** du GRO, nous avons une procédure `BigTxT_SaveData` qui permet de convertir des bases de données de plusieurs GigaOctets en quelques minutes.

– Voici un exemple qui utilise les commandes `fopen`, `fgetsa` et `stof` et la procédure `token`.

```
new;

fin = fopen(''uneBase.asc'',''r'');

let vnames = ''nom'' ''sexe'' ''AGE'' ''ville'' ''POIDS'' ''P/AGE'';
create fout = uneBase3 with ^vnames,6,8;

let mask = 0 0 1 0 1;

do until eof(fin);
  str = fgetsa(fin,1);    /* Lecture d'une ligne de fichier */
  sa = ConvertString(str);
  x = ConvertStringArray(sa',mask);
```

```
  x = x~x[5]/x[3];
  call writer(fout,x);
endo;

fin = close(fin);
fout = close(fout);

datalist uneBase3;

proc (1) = ConvertString(str);
  local sa,sa_;

  {sa,str} = token(str);

  do until str $== '''';
    {sa_,str} = token(str);
    sa = sa $| sa_;
  endo;

  retp(sa);
endp;

proc (1) = ConvertStringArray(sa,mask);
  local r,c,x,i,j;

  r = rows(sa);
  c = cols(sa);
  x = 0 $+ sa;

  i = 1;
  do until i > c;
    if mask[i] == 1;
      j = 1;
      do until j > r;
        x[j,i] = stof(sa[j,i]);
        j = j + 1;
      endo;
    endif;
    i = i + 1;
  endo;

  retp(x);
endp;
```

21.6.2 Les procédures export/f et import/f

GAUSS possède des procédures pour exporter/importer des bases de données externes.

export/f

```
*
**> export/f
**
**  Purpose:  Exports a matrix or GAUSS data set to a spreadsheet, database,
```

```
**            or ASCII formatted file.
**
**  Format:   y = export(x, fname, namelist);
**  Format:   y = exportf(dataset, fname, namelist);
**
**  Input:    x          NxK matrix of data to be exported.
**            dataset    string, name of GAUSS data set to be exported.
**            fname      string, path and filename of target file
**            namelist   Kx1 character vector of names,
**                       or scalar 0 (for defaults).
**
**  Output:   y          scalar, 1 if successful,
**                              0 if not.
**
**  Globals: _dxftype    string, file type. The file type is normally
**                       taken from the filename extension; this can be
**                       overridden by specifying one of the file extensions
**                       listed below. Set to '''' (empty string) to use
**                       file extensions.
**
**           _dxtype     scalar or Kx1 vector, column data type flags:
**                       1's for numeric data, 0's for character data.
**                       Use scalar if all columns are the same type.
**                       Default is scalar 1--all columns numeric.
**
**           _dxwidth    scalar or Kx1 vector of spreadsheet column widths
**                       in characters. Use scalar if all columns have the
**                       same width. Default is scalar 12.
**
**           _dxprcn     scalar or Kx1 vector of spreadsheet column precision
**                       (number of digits after the decimal). Use scalar if
**                       all columns have the same precision. Default is 4.
**
**           _dxtxdlim   scalar, ASCII value for character that delimits
**                       fields in ASCII files. (Tab = 9, comma = 44,
**                       space = 32 (default))
**
**           _dxaschdr   scalar, ASCII file column headers flag: 1 - write
**                       column names as headers, 0 - don't write. Default
**                       is 0.
**
**           _dxwkshdr   scalar, spreadsheet file column headers flag: 1 -
**                       write column names as headers, 0 - don't write.
**                       Default is 1.
**
**           _dxmiss     scalar, missing value representation. Default is
**                       standard GAUSS missing value (indefinite NaN).
**
**           _dxprint    scalar, 1 - print progress messages,
**                               0 - quiet. Default 1.
**
**  Remarks:
```

```
**       The following file types are supported:
**
**         ''WKS''                  Lotus v1.0
**         ''XLS''                  Excel v2.1
**         ''WQ1''                  Quattro v1.0
**         ''WRK''                  Symphony v1.0
**         ''DB2''                  dBase II
**         ''DBF''                  dBase III
**         ''DB''                   Paradox v3.0
**         ''CSV'' ''TXT'' ''ASC''      ASCII character delimited
**         ''PRN''                  ASCII formatted
**         ''DAT''                  GAUSS data set
**
**       The number of elements in namelist should conform to the number
**    of columns in X for a matrix transfer.
**
**       For a data set, if namelist = 0, all the vectors will be exported;
**     otherwise the subset of vectors named in namelist will be exported.
**
**       The elements of namelist will be inserted in the first row of each
**    column, unless _dxaschdr (ASCII files) or _dxwkshdr (spreadsheet files)
**    is set to 0.
**
**       Missing values will be written as blank cells to spreadsheets,
**    and as _dxmiss to ASCII files.
**
**   Examples:
**             fname = ''c:\\temp\\tstdta.xls'';
**             let names = gnp invest consump exports;
**             call export(x,fname,names);
**
**             fname = ''c:\\temp\\tstdta.dbf'';
**             dname = ''c:\\gauss\\dat\\mydata.dat'';
**             call exportf(dname,fname,0);
**
**       The first example exports a four column matrix (x) to an
**       Excel file. The second example takes a GAUSS data set and
**       creates a Dbase file.
**
*/
```

import/f

```
/*
**> import/f
**
**  Purpose:  Copy data from a spreadsheet, database or ASCII file to
**            a GAUSS matrix or data set.
**
**  Format:   {x,namelist} = import(fname, range, sheet);
**                       y  = importf(fname,dataset,range, sheet);
**
**  Input:   fname      string, path and filename of source file.
```

```
**           dataset    string, path and filename of GAUSS data set.
**           range      string, range of cells for spreadsheets,
**                      descriptor for packed ASCII files, or scalar 0.
**                      Default is 0.
**           sheet      scalar, page or sheet number. Default is 1.
**
**  Output:  x          NxK matrix of data from spreadsheet.
**           namelist   Kx1 character vector of column names.
**           y          scalar, 1 if successful, 0 if not.
**
**  Globals: _dxftype   string, file type. The file type is normally
**                      taken from the filename extension; this can be
**                      overridden by specifying one of the file extensions
**                      listed below. Set to '''' (empty string) to use
**                      file extensions.
**
**           _dxbuffer  Import buffer size in Mbytes. 1 Mbyte is approx
**                      131000 elements. If the data size exceeds
**                      this, you should increase _dxbuffer.
**
**           _dxtxdlim  For delimited ASCII files, scalar ASCII value
**                      of the character that delimits fields. (Tab = 9,
**                      comma = 44, space = 32 (default)). Not used
**                      in packed ASCII files.
**
**           _dxaschdr  scalar, ASCII file column headers flag: 1 - read
**                      column names as headers, 0 - don't read. Default
**                      is 0.
**
**           _dxwkshdr  scalar, row number of spreadsheet file column
**                      headers: 0 - no headers. Default is 1.
**
**           _dxmiss    scalar, missing value representation. Default is
**                      standard GAUSS missing value (indefinite NaN).
**
**           _dxprint   scalar, 1 - print progress messages (default),
**                              0 - quiet.
**
**  Remarks:
**       The following file types are supported:
**
**         ''WKS'' ''WK1'' ''WK2''      Lotus v1-v2
**         ''WK3'' ''WK4'' ''WK5''      Lotus v3-v5
**         ''XLS''                   Excel v2.1-v7.0
**         ''WQ1'' ''WQ2'' ''WB1''      Quattro v1-v6
**         ''WRK''                   Symphony v1.0-1.1
**         ''DB2''                   dBase II
**         ''DBF''                   dBase III/IV, Foxpro, Clipper
**         ''DB''                    Paradox
**         ''CSV'' ''TXT'' ''ASC''      ASCII character delimited
**         ''PRN''                   ASCII packed
**         ''DAT''                   GAUSS data set
```

```
**
**       For spreadsheets, _dxwkshdr indicates the row to use for column
**       names; for no column names, set _dxwkshdr to 0.
**
**       For ASCII files, column names are assumed to exist if _dxaschdr is
**       set to 1.
**
**       Column names will be returned in namelist.
**
**       For spreadsheets, range indicates the columns to be imported, and
**       can be specified in the form ''A1..X27'' or ''A1:X27''. A range of 0
**       imports all data.
**
**       For packed ASCII files, range is a descriptor that defines field
**       name, type, width, and optionally precision. It is a single string
**       of the form:
**
**               name [type] fldspec [name [type] fldspec ...]
**
**       where:  name is the column name for the field. You can specify a
**                  set (e.g., x01-x09); the subsequent type and fldspec
**                  are applied to all columns in the set.
**               type is $ for a character field, and blank for a numeric
**                  field.
**               fldspec is either a column range (e.g., 5-8), a start
**                  column and field width (e.g., 5,4 or 5,4.2), or just a
**                  field width (e.g., 4 or 4.2). If only a field width is
**                  given, the start column is imputed from the previous
**                  field. If the field width is specified with a decimal
**                  (e.g., 4.2) then a decimal point will be inserted that
**                  many places in from the right edge of the field.
**
**       sheet is the spreadsheet page number, and is only supported
**       for spreadsheet formats. If sheet = 0, the first (possibly only)
**       page will be imported.
**
**       Spreadsheet cells that are #ERR or #N/A will be imported as GAUSS
**       missing values. Elements (from any format) that have the value
**       _dxmiss will be imported as GAUSS missing values.
**
**   Examples:
**            fname = ''c:\\temp\\tstdta.xls'';
**            range = ''a2..g51'';
**            {x,names} = import(fname,range,1);
**
**            fname = ''c:\\temp\\tstdta.asc'';
**            dname = ''c:\\gauss\\dat\\mydata.dat'';
**            call importf(fname,dname,0,0);
**
**            fname = ''c:\\temp\\tstdta.asc'';
**            schema = ''var1 $ 6,2 var2 8-15 var3 18,4.1 gnp 7 cons 6.2
**                           xs1-xs20 2'';
```

```
**            {x,names} = import(fname,schema,0);
**
**   The first example shows the creation of a GAUSS matrix x from an
**   Excel file, using the specified range. In the second example, a
**   GAUSS data set is created from a space delineated ASCII data set.
**   In the third example, a GAUSS matrix (x) is created from a packed
**   ASCII data set, using the descriptor shown.
**
*/
```

– Voici un exemple.

```
/*****************************************************************************
  DATAXCHN.E
  Demo program for dataxchn package
  Econotron Software, 1997
*****************************************************************************/
         @ temp code until GAUSS.LCG is updated @
#include dataxchn.dec
#include dataxchn.src
/*****************************************************************************/

    cls;
    ''        Demo program for dataxchn package

    This program takes an existing dataset (sci.dat) on the \\EXAMPLES
    directory, writes some of the data out to an Excel file, and then
    reads the data back in, and displays it.

    '';
    ''''; ''Press any key to continue''; call keyw;

    cls;
    gpath = sysstate(2,1);
    gname = gpath $+ ''examples\\sci.dat'';
    fname = gpath $+ ''examples\\sci.xls'';
    names = getname(gname);
    call exportf(gname,fname,names[1:6]);
    ''''; ''Press any key to continue''; call keyw;
    {x,nm} = import(fname,0,0);
    ''''; ''Press any key to continue''; call keyw;
    cls;
    ''Imported vectors '' $nm;
    '' '';
    format 8,4;
    ''Data (first 10 obs)'';
    x[1:10,.];
```

21.7 Utilisation avancée de l'accès séquentiel

La vraie richesse de **GAUSS** concernant les bases de données est l'accès séquentiel, ce qui permet de gérer des bases de données énormes. Voici quelques exemples.

– Chargement partiel de la base de données (sélection de variables).

```
proc (1) = LoadData(dataset,vars);
  local name,indx,c,f,r,x,nr,r1,r2,data;

  {name,indx} = indices(dataset,vars);
  c = rows(indx);

  open f = ^dataset for read;

  r = rows(f);
  x = zeros(r,c);
  nr = getnr(6,colsf(f));

  r1 = 1;
  r2 = 0;

  do until eof(f);
    data = readr(f,nr);
    r2 = r1 + rows(data) - 1;
    x[r1:r2,.] = submat(data,0,indx);
    r1 = r2 + 1;
  endo;

  f = cols(f);

  retp(x);
endp;
```

– Copie partielle de la base de données (sélection de variables).

```
proc (0) = CopyData(dataset,vars,newdataset);
  local name,indx,c,fin,fout,nr;

  {name,indx} = indices(dataset,vars);
  c = rows(indx);

  open fin = ^dataset for read;
  create fout = ^newdataset with ^name,c,8;

  nr = getnr(6,colsf(fin));

  do until eof(fin);
    call writer(fout,submat(readr(fin,nr),0,indx));
  endo;

  closeall fin, fout;

  retp;
endp;
```

– Chargement partiel de la base de données (sélection de variables) avec conditions.

```
proc (1) = LoadDataIf(dataset,vars,procCnd,varsCnd);
  local procCnd:proc;
  local name,indx,indxCnd,c,f,nr,data,x;
```

```
  {name,indx} = indices(dataset,vars);
  c = rows(indx);
  {name,indxCnd} = indices(dataset,varsCnd);

  open f = ^dataset for read;

  x = {};
  nr = getnr(6,colsf(f));

  do until eof(f);
    data = readr(f,nr);
    data = selif(data[.,indx],procCnd(data[.,indxCnd]));
    if data /= error(0);
      x = x | data;
    endif;
  endo;

  f = cols(f);

  retp(x);
endp;
```

– Chargement de certaines observations de la base de données (sélection de variables).

```
proc (1) = LoadRows(dataset,vars,row);
  local name,indx,c,f,r,x,nr,i;

  {name,indx} = indices(dataset,vars);
  c = rows(indx);

  open f = ^dataset for read;

  r = rows(row);
  x = miss(zeros(r,c),0);
  nr = getnr(6,colsf(f));
  row = missex(row, row .> rowsf(f) );

  i = 1;
  do until i > r;
    if ismiss(row[i]);
      i = i + 1;
      continue;
    endif;
    call seekr(f,row[i]);
    x[i,.] = submat(readr(f,1),0,indx);
    i = i + 1;
  endo;

  f = cols(f);

  retp(x);
endp;
```

22 L'utilisation des DLLs dans GAUSS

C'est un aspect relativement peu exploité par les programmeurs. Pourtant, il permet de réaliser un certain nombre de taches qui

1. prennent trop de temps en GAUSS (par exemple, les procédures qui font appel à de nombreuses boucles) ;
2. ou qui ne sont pas accessibles à partir des commandes de GAUSS (par exemple, l'accès à *user.dll*, *kernel.dll* ou *mm.dll*).

Le lecteur peut consulter le séminaire de Londres pour un traitement plus complet de ce sujet :

http : //www.city.ac.uk/cubs/ferc/thierry/conf3pdf.zip
http : //www.city.ac.uk/cubs/ferc/thierry/gdll1.html

Les exemples qui suivent utilisent le compilateur gratuit LCC-Win32 disponible sur le net

http ://www.cs.virginia.edu/~lcc-win32/

Les instructions sont les suivantes

1. Write the C file *dll1.c*.
2. Add the file *\lcc\lib\wizard\dll.tpl* to the C file. You have to do that because the file .dll need an entry point to load/unload DLLs.
3. Compile the file with the following command line **lcc dll1.c**.
4. Build the DLLs with the following command line **lcclnk -dll dll1.obj**.
5. The file dll1.exp contains the names of exported procedures.
6. You could now use the DLLs with GAUSS.

Fichier *dll1.c*

```
#include <windows.h>
#include <malloc.h>

/*-------------------------------------------------------------------------
 Procedure:     LibMain ID:1
 Purpose:       Dll entry point.Called when a dll is loaded or
                unloaded by a process, and when new threads are
                created or destroyed.
 Input:         hDllInst: Instance handle of the dll
                fdwReason: event: attach/detach
                lpvReserved: not used
 Output:        The return value is used only when the fdwReason is
                DLL_PROCESS_ATTACH. True means that the dll has
                sucesfully loaded, False means that the dll is unable
                to initialize and should be unloaded immediately.
 Errors:
-------------------------------------------------------------------------*/
BOOL WINAPI __declspec(dllexport) LibMain(HINSTANCE hDLLInst, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH:
            // The DLL is being loaded for the first time by a given process.
            // Perform per-process initialization here.  If the initialization
            // is successful, return TRUE; if unsuccessful, return FALSE.
```

```
            break;
        case DLL_PROCESS_DETACH:
            // The DLL is being unloaded by a given process.  Do any
            // per-process clean up here, such as undoing what was done in
            // DLL_PROCESS_ATTACH.  The return value is ignored.


            break;
        case DLL_THREAD_ATTACH:
            // A thread is being created in a process that has already loaded
            // this DLL.  Perform any per-thread initialization here.  The
            // return value is ignored.

            break;
        case DLL_THREAD_DETACH:
            // A thread is exiting cleanly in a process that has already
            // loaded this DLL.  Perform any per-thread clean up here.  The
            // return value is ignored.

            break;
    }
    return TRUE;
}


__declspec(dllexport) int dllTDGSolve(double *a,double *b,double *c,double *d,
                                      double *n,double *x,double *bprime, double *dprime)
{
 int i;
 int N;
 double w;

 N = (int) *n;

 bprime[N-1] = b[N-1];
 dprime[N-1] = d[N-1];

 for (i = N-2; i >= 0; --i)
  {
   w = c[i] / bprime[i+1];
   bprime[i] = b[i] - w * a[i+1];
   dprime[i] = d[i] - w * dprime[i+1];
  }

    x[0] = dprime[0] / bprime[0];

 for (i = 1; i <= N-1; ++i)
  {
   x[i] = (dprime[i] - a[i] * x[i-1]) / bprime[i];
  }

 return 0;
}
```

Troisième partie
APPLICATIONS

23 Optimisation d'une fonction

Il existe de nombreuses procédures **GAUSS** pour optimiser une fonction. Ces procédures de base sont :
- `eqSolve` — `{x, retcode} = eqSolve(&f,x0)` — Résolution d'un système non-linéaire ;
- `sqpSolve` — `{x,f,lagr,retcode} = sqpSolve(&fct,start)` — Optimisation d'une fonction sous contraintes linéaires et non-linéaires ;
- `QNewton` — `{x,f,g,retcode} = QNewton(&fct,x0)` — Minimisation d'une fonction ;
- `Qprog` — `{x,u1,u2,u3,u4,ret} = QProg(start,q,r,a,b,c,d,bnds)` — Résolution d'un problème de programmation quadratique.

Des bibliothèques spécialisées complètent ces procédures : **optmum**, **nlsys**, **curvefit**, **co**, **maxlik**, **cml**.

23.1 Les procédures `optmum` et `QNewton`

- Considérons un premier exemple. La fonction à minimiser est

$$f(x) = x^2 \tag{1}$$

```
new;

fn f(x) = x^2;

{x0,f0,g0,retcode} = Qnewton(&f,-3);

{x0,f0,g0,retcode} = Qnewton(&f,5);

_qn_PrintIters = 1;
{x0,f0,g0,retcode} = Qnewton(&f,5);
```

- Considérons un second problème dans $\mathbb{R}^2$:

$$\min x_1^2 + 5x_2^2 + 3(x_1 - 2x_2 + 7)^2 \tag{2}$$

```
new;

fn f(x) = x[1]^2 + 5*x[2]^2 + 3*(x[1]-2*x[2]+7)^2;

{x0,f0,g0,retcode} = Qnewton(&f,-3|5);

{x0,f0,g0,retcode} = Qnewton(&f,5|1);

_qn_PrintIters = 1;
{x0,f0,g0,retcode} = Qnewton(&f,5|-1);
```

- Reprenons le second exemple en implémentant un gradient analytique.

```
new;

fn f(x) = x[1]^2 + 5*x[2]^2 + 3*(x[1]-2*x[2]+7)^2;
fn g(x) = 2*x[1]+6*(x[1]-2*x[2]+7) | 10*x[2]-12*(x[1]-2*x[2]+7);
```

```
_qn_GradProc = &g;

{x0,f0,g0,retcode} = Qnewton(&f,-3|5);

{x0,f0,g0,retcode} = Qnewton(&f,5|1);

_qn_PrintIters = 1;
{x0,f0,g0,retcode} = Qnewton(&f,5|-1);
```

– Le quatrième exemple est un problème classique de programmation quadratique :

$$\min \frac{1}{2} x^\top Q x - x^\top R \tag{3}$$

Nous avons

$$\frac{\partial}{\partial x^\top} \left(0.5 x^\top Q x - x^\top R\right) = Qx - R \tag{4}$$

Nous en déduisons que

$$\arg\min \frac{1}{2} x^\top Q x - x^\top R = Q^{-1} R \tag{5}$$

La valeur de la fonction objectif au point minimum est alors $0.5 R^\top Q^{-1} R$.

```
new;

Q = { 0.78 -0.02 -0.12 -0.14,
     -0.02  0.86 -0.04  0.06,
     -0.12 -0.04  0.72 -0.08,
     -0.14  0.06 -0.08  0.74 };

R = { 0.76, 0.08, 1.12, 0.68 };

x0 = { 1, 1, 1, 1 };

proc fct(x);
    retp(.5*x'*Q*x-x'R);
endp;

_qn_RelGradTol = 1e-15;
{x,fmin,g,retcode} = Qnewton(&fct,x0);

print x;
print; print ''Valeur de la fonction = '' fmin;

print invpd(Q)*R;
print; print ''Valeur de la fonction = '' -0.5*R'invpd(Q)*R;
```

– Le dernier exemple concerne un problème de moindres carrés non linéaires. Le modèle statistique est le suivant :

$$y_i = \frac{b_1 x_i}{b_2 + x_i} + u_i \tag{6}$$

```
/*
**   The data are taken from Douglas M. Bates and Donald G. Watts,
**   Nonlinear Regression Analysis and Its Applications, page 269.
**
*/
```

```
new;
library optmum;

proc mm(b,x);
    retp(b[1]*x./(b[2] + x));
endp;

/*  treated  */
y1 = { 76,   47,   97,  107,  123,  139,  159,  152,  191,  201,  207,  200 };
x1 = { .02,  .02,  .06,  .06,  .11,  .11,  .22,  .22,  .56,  .56,  1.1,  1.1 };

/*  untreated  */
y2 = { 67,   51,   84,   86,   98,  115,  131,  124,  144,  158,  160 };
x2 = { .02,  .02,  .06,  .06,  .11,  .11,  .22,  .22,  .56,  .56,  1.1 };

proc scr1(beta);
  local u;
  u = y1 - beta[1]*x1 ./ (beta[2] + x1);
  retp(u'u);
endp;

proc scr2(beta);
  local u;
  u = y2 - beta[1]*x2 ./ (beta[2] + x2);
  retp(u'u);
endp;

let sv = 100 5;

__output = 0;
{beta1,fmin,g,retcode} = optmum(&scr1,sv);
{beta2,fmin,g,retcode} = optmum(&scr2,sv);
```

23.2 Les procédures `co` et `sqpSolve`

Ces deux procédures permettent de résoudre un problème de programmation non linéaire :

$$\min f(x) \tag{7}$$

avec

$$\begin{cases} Ax = B \\ g(x) = \mathbf{0} \\ Cx \geq D \\ h(x) \geq \mathbf{0} \\ x^- \leq x \leq x^+ \end{cases} \tag{8}$$

Outre la solution du problème, les procédures retournent les valeurs prises par les lagrangiens pour l'ensemble des contraintes d'égalité et d'inégalité.

– Considérons un premier exemple. Nous avons

$$f(x) = (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_2)^2 \tag{9}$$

Nous voulons imposer la contrainte d'inégalité suivante :

$$-x_1^3 + 6x_2 + 4x_3 \geq 3 \tag{10}$$

De même, nous voulons que

$$\sum x_i = 1 \tag{11}$$

De plus, la solution doit être dans $\mathbb{R}^3_+$. La première contrainte est spécifiée à l'aide de la variable externe _sqp_IneqProc. Pour les deux autres restrictions, nous utilisons les variables externes _sqp_EqProc et _sqp_Bounds.

```
new;

proc fct(x);
   retp( (x[1] + 3*x[2] + x[3])^2 + 4*(x[1] - x[2])^2 );
endp;

proc ineqp(x);
    retp(6*x[2] + 4*x[3] - x[1]^3 - 3);
endp;

proc eqp(x);
    retp(1-sumc(x));
endp;

_sqp_Bounds = { 0 1e256 };

start = { .1, .7, .2 };

_sqp_IneqProc = &ineqp;
_sqp_EqProc = &eqp;

{ x,f,lagr,ret } = sqpSolve( &fct,start );

print;
print ''published solution'';
print '' 0      0       1'';
```

– Remarquez qu'imposer des bornes revient à imposer des contraintes d'inégalité supplémentaires. Bien sûr, cela ne change pas la solution.

```
new;

proc fct(x);
   retp( (x[1] + 3*x[2] + x[3])^2 + 4*(x[1] - x[2])^2 );
endp;

proc ineqp(x);
    retp(
         -x[1]^3 + 6*x[2] + 4*x[3]  - 3 |
          x[1]                          |
                    x[2]                |
                             x[3]
        );
endp;

proc eqp(x);
    retp(1-sumc(x));
```

```
endp;

_sqp_Bounds = { -1e256 1e256 };

start = { .1, .7, .2 };

_sqp_IneqProc = &ineqp;
_sqp_EqProc = &eqp;

{ x,f,lagr,ret } = sqpSolve( &fct,start );

print;
print ''published solution'';
print '' 0      0       1'';
```

– La procédure `co` s'utilise de la même manière que `sqpSolve`. La différence majeure est que `co` ne retourne pas directement les valeurs des lagrangiens. Ceux-ci sont stockés dans une variable externe `_co_Lagrange`. Pour lire les valeurs, nous utilisons la procédure `vread`, puisque `_co_Lagrange` est une variable **buffer** (construite à partir de la procédure `vput`). Pour connaître la liste des variables, nous employons la commande `vlist`.

```
new;
library co;
coset;

proc fct(x);
   retp( (x[1] + 3*x[2] + x[3])^2 + 4*(x[1] - x[2])^2 );
endp;

proc ineqp(x);
    retp(6*x[2] + 4*x[3] - x[1]^3 - 3);
endp;

proc eqp(x);
    retp(1-sumc(x));
endp;

_co_Bounds = { 0 1e256 };

start = { .1, .7, .2 };

_co_IneqProc = &ineqp;
_co_EqProc = &eqp;

{ x,f,g,ret } = coprt(co( &fct,start ));

print;
print ''published solution'';
print '' 0      0       1'';

print;
print ''nonlinear equality Lagrangeans'';
print vread(_co_Lagrange,''nlineq'');
```

```
print;
print ''nonlinear inequality Lagrangeans'';
print vread(_co_Lagrange,''nlinineq'');
print;
print ''boundary Lagrangeans'';
print ''Lower / Upper'';
print vread(_co_Lagrange,''bounds'');
```

Remarque 19 *Il existe un bug dans la procédure* `co`*. Il faut remplacer la ligne 596 (*`if rows(_co_Bounds)==2;`*) dans le fichier coutils.src par celle-ci :*

```
if cols(_co_Bounds)==2;
```

Pourquoi ?

– Nous considérons un dernier exemple. Nous cherchons à estimer les paramètres d'un modèle tobit. Nous observons la variable y_i définie par

$$y_i = \max(z_i, 0) \tag{12}$$

avec

$$z_i = x_i\beta + u_i \tag{13}$$

et $u_i \sim \mathcal{N}(0, \sigma^2)$. Si $y_i > 0$, la log-vraisemblance individuelle s'écrit :

$$\ell(y_i; \beta, \sigma) = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}(y_i - x_i\beta)^2 \tag{14}$$

Si $y_i = 0$, celle-ci devient

$$\ell(y_i; \beta, \sigma) = \ln\Phi(x_i\beta/\sigma) \tag{15}$$

Dans le programme suivant, nous estimons dans un premier temps les paramètres non contraints. Ensuite, nous imposons la restriction

$$\beta_1 + \beta_2 \geq 0 \tag{16}$$

Enfin, nous considérons la contrainte

$$\beta_1 + \beta_2 = \beta_3 \tag{17}$$

```
/*
**  The Tobit Model with Inequality/Equality Constraints
**
*/

new;
library co,optmum;
coset;
optset;

data = loadd(''tobit'');
y = data[.,1];
x = data[.,2 3 4];

proc loglProc(theta);
  local beta,sigma2,xb,e,u,logl1,logl2,logl;

  beta = theta[1:3];
  sigma2 = theta[4]^2;
```

```
  xb = x*beta;
  e = y ./= 0;
  u = y - xb;

  logl1 = -0.5*ln(2*pi) -0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;
  logl2 = lncdfnc(xb/sqrt(sigma2));
  logl = e .* logl1 + (1-e) .* logl2;

  retp(logl);
endp;

proc mlProc(theta);
  retp( - sumc(loglProc(theta)) );
endp;

let sv = 1 1 1 1;

{theta1,logl1,g1,retcode} = optmum(&mlProc,sv);

_co_C = { 1 1 0 0 };
_co_D = { 0 };
_co_Bounds = { -10 10,
               -10 10,
               -10 10,
               .01 10 };

{theta2,logl2,g2,retcode} = co(&mlProc,sv);

_co_A = { 1 1 -1 0 };
_co_B = { 0 };
let _co_C = . ;
let _co_D = . ;

sv = theta2;

{theta3,logl3,g3,retcode} = co(&mlProc,sv);
```

23.3 La procédure `Qprog`

La procédure `Qprog` permet de résoudre un problème de programmation linéaire :

$$\min \frac{1}{2} x^\top Q x - x^\top R \tag{18}$$

avec

$$\begin{cases} Ax = B \\ Cx \geq D \\ x^- \leq x \leq x^+ \end{cases} \tag{19}$$

C'est une procédure extrêmement rapide qui permet de résoudre facilement des systèmes de très grandes dimensions.

Considérons le problème de moindres carrés :

$$\min \left(Y - X\beta\right)^\top \left(Y - X\beta\right) \tag{20}$$

avec

$$\left\{ \begin{array}{l} A\beta = B \\ C\beta \geq D \\ \beta^- \leq \beta \leq \beta^+ \end{array} \right. \tag{21}$$

Nous pouvons écrire la fonction objectif de la façon suivante :

$$(Y - X\beta)^\top (Y - X\beta) = Y^\top Y - 2Y^\top X\beta + \beta^\top X^\top X\beta \tag{22}$$

Nous en déduisons que

$$\arg\min (Y - X\beta)^\top (Y - X\beta) = \arg\min \frac{1}{2}\beta^\top X^\top X\beta - \beta^\top X^\top Y \tag{23}$$

Nous obtenons donc un problème quadratique linéaire avec

$$Q = X^\top X \tag{24}$$

et

$$R = X^\top Y \tag{25}$$

La procédure `qpOLS` permet d'estimer le vecteur de paramètre β du problème de moindres carrés. Néanmoins, l'estimation de la matrice de covariance est plus difficile à construire.

```
proc (2) = qpOLS(y,x,A,B,C,D,bnds);
  local xx,xy;
  local bols,beta,u1,u2,u3,u4,retcode;

  xx = x'x;
  xy = x'y;

  bols = y/x;
  {beta,u1,u2,u3,u4,retcode} = Qprog(bols,xx,xy,A,B,C,D,bnds);

  if retcode /= 0;
    retp(error(0),bols);
  endif;

  beta = beta .* dotfne(beta,0);

  retp(beta,bols);
endp;
```

– Voyons un exemple.

```
new;
library ritme;

rndseed 123;

Nobs = 1000;
k = 25;
sigma = 1;

coeff = rndu(k,1);
x = 50*rndu(Nobs,k);
y = x*coeff + rndn(Nobs,1)*sigma;
```

```
/* Estimation non contrainte */

{beta1,bols} = qpOLS(y,x,0,0,0,0,0);

/* beta[1] = 2 */

A = 1~zeros(1,k-1); B = 2;
{beta2,bols} = qpOLS(y,x,A,B,0,0,0);

/*
   beta[1] + ... + beta[k] = 0
   beta[1] + 2 * beta[2] > 5
   beta[3] - beta[4] > 2
   -0.5 < beta[i] < 4
*/

A = ones(1,k); B = 0;
C = 1~2~zeros(1,k-2)      |
    0~0~1~-1~zeros(1,k-4) ;
D = 5|2;
bnds = {-0.5 4};
{beta3,bols} = qpOLS(y,x,A,B,C,D,bnds);

output file = qpOLS1.out reset;

print coeff~beta1~beta2~beta3;

output off;
```

– Dans l'exemple suivant, nous comparons les inférences statistiques "classiques" et de Wald.

```
new;
library ritme,cml;
cmlset;

rndseed 123;

Nobs = 1000;
k = 5;
sigma = 1;

coeff = rndu(k,1);
x = 50*rndu(Nobs,k);
y = x*coeff + rndn(Nobs,1)*sigma;

/*
   beta[1] + ... + beta[k] = 0
   beta[1] + 2 * beta[2] > 5
*/

A = ones(1,k); B = 0;
C = 1~2~zeros(1,k-2);
```

```
D = 5;
bnds = { . } ;
{beta,bols} = qpOLS(y,x,A,B,C,D,bnds);

u = y - x*beta;
rss = u'u;
df = rows(u) - cols(x);
sigma = sqrt(rss/df);
cov = (sigma^2)*invpd(x'x);
stderr = sqrt(diag(cov));
tstudent = beta./stderr;
pvalue = 2*cdftc(abs(tstudent),df);

proc mlProc(theta,data);
  local k,y,x,b,sigma2,L,u;

  k = cols(data) - 1;
  y = data[.,1];
  x = data[.,2:1+k];
  b = theta[1:k];
  sigma2 = theta[1+k]^2;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;

  retp(L);
endp;

sv = beta | sigma;

_cml_A = ones(1,k)~0;
_cml_B = 0;
_cml_C = 1~2~zeros(1,k-2)~0;
_cml_D = 5;

__output = 1;
{theta,Logl,g,cov2,retcode} = cmlprt(cml(y~x,0,&mlProc,sv));

print;
call printfm(beta~stderr~tstudent~pvalue,1,''%lf''~8~3);
```

Exemple 5 *La procédure* ***Qprog*** *est utilisée dans la bibliothèque* ***DEA*** *(Data Envelopment Analysis). Voir le site web :*

http ://www.city.ac.uk/cubs/ferc/thierry/gauss.html

24 Maximum de vraisemblance

Notons $\ell(y_i;\theta)$ la log-vraisemblance individuelle. L'estimateur du maximum de vraisemblance est alors défini de la façon suivante :

$$\hat{\theta}_{\mathrm{ML}} = \arg\max \sum_{i=1}^{n} \ell(y_i;\theta) \tag{26}$$

– Prenons par exemple le cas du modèle statistique linéaire. Nous avons

$$y_i = x_i\beta + u_i \tag{27}$$

avec $u_i \sim \mathcal{N}\left(0, \sigma^2\right)$. Nous en déduisons que

$$\ell\left(y_i; \beta, \sigma\right) = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\left(y_i - x_i\beta\right)^2 \tag{28}$$

```
new;
library optmum;

n = 100;
let beta = 1 2 3;
sigma = 2;

x = 100*rndu(n,3);
y = x*beta + sigma * rndn(n,1);

b_ols = y/x;

proc logl(theta);
  local b,sigma2,L,u;

  b = theta[1:3];
  sigma2 = theta[4]^2;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;

  retp(L);
endp;

proc fmin(theta);
  retp( -sumc(logl(theta)) );
endp;

let sv = 1 1 1 1 ;

{theta_ml,f,g,retcode} = optmum(&fmin,sv);
b_ml = trimr(theta_ml,0,1);
```

Une fois obtenue l'estimateur du maximum de vraisemblance, nous pouvons faire de l'inférence statistique. Notons $\ell\left(y;\theta\right)$ la fonction de log-vraisemblance, c'est-à-dire

$$\ell\left(y;\theta\right) = \sum_{i=1}^{n} \ell\left(y_i;\theta\right) \tag{29}$$

L'estimateur du maximum de vraisemblance vérifie la propriété de normalité asymptotique :

$$n^{1/2}\left(\hat{\theta}_n - \theta_0\right) \rightarrow \mathcal{N}\left(0, \mathbf{I}\left(\theta_0\right)^{-1}\right) \tag{30}$$

$\mathbf{I}\left(\theta_0\right)$ est la matrice d'information de Fisher. Nous avons

$$\mathbf{I}\left(\theta_0\right) = -\mathbb{E}\left[\frac{\partial^2}{\partial\theta\,\partial\theta^\top}\ell\left(y;\theta_0\right)\right] \tag{31}$$

ou

$$\mathbf{I}(\theta_0) = E\left[\frac{\partial}{\partial\theta}\ell(y;\theta_0)\,\frac{\partial}{\partial\theta^\top}\ell(y;\theta_0)\right] \tag{32}$$

Ces résultats suggèrent deux estimateurs de la matrice de covariance de $\hat{\theta}_{\mathrm{ML}}$:

$$\mathrm{cov}\left[\hat{\theta}_{\mathrm{ML}}\right] = \left(-\frac{\partial^2}{\partial\theta\,\partial\theta^\top}\ell\left(y;\hat{\theta}_{\mathrm{ML}}\right)\right)^{-1} \tag{33}$$

et

$$\mathrm{cov}\left[\hat{\theta}_{\mathrm{ML}}\right] = \left(J\left(\hat{\theta}_{\mathrm{ML}}\right)^\top J\left(\hat{\theta}_{\mathrm{ML}}\right)\right)^{-1} \tag{34}$$

avec $J\left(\hat{\theta}_{\mathrm{ML}}\right)$ le jacobien de $(\ell(y_1;\theta),\ldots,\ell(y_n;\theta))$.

– Reprenons l'exemple précédent et calculons les écart-types à partir de la matrice de covariance construite à partir de l'inverse de l'opposé de la matrice hessienne.

```
new;
library optmum;

n = 100;
let beta = 1 2 3;
sigma = 2;

x = 100*rndu(n,3);
y = x*beta + sigma * rndn(n,1);

b_ols = y/x;
u = y - x*b_ols;
rss = sumc(u^2);
df = rows(x) - cols(x);
df = rows(x);                /* POUR RETROUVER LES RESULTATS ML */
sigma_ols = sqrt(rss/df);

theta_ols = b_ols /* | sigma_ols */ ;

cov_ols = (sigma_ols^2) * invpd(x'x);
stderr_ols = sqrt(diag(cov_ols));
tstudent_ols = theta_ols ./ stderr_ols;
pvalue_ols = 2*cdftc(abs(tstudent_ols),df);

proc logl(theta);
  local b,sigma2,L,u;

  b = theta[1:3];
  sigma2 = theta[4]^2;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;

  retp(L);
endp;

proc fmin(theta);
```

```
  retp( -sumc(logl(theta)) );
endp;

let sv = 1 1 1 1 ;
_opgtol = 1e-7;
{theta_ml,f,g,retcode} = optmum(&fmin,sv);

b_ml = theta_ml[1:3];
sigma_ml = theta_ml[4];

H = -hessp(&fmin,theta_ml);
cov_ml = invpd(-H);
stderr_ml = sqrt(diag(cov_ml));
tstudent_ml = theta_ml ./ stderr_ml;
pvalue_ml = 2*cdftc(abs(tstudent_ml),df);

cls;

call printfmt(theta_ols~stderr_ols~tstudent_ols~pvalue_ols,1);
print;
call printfmt(theta_ml~stderr_ml~tstudent_ml~pvalue_ml,1);
```

– Dans le programme qui suit, nous comparons les écart-types obtenus à partir des trois estimateurs de la matrice de covariance (le troisième étant l'estimateur hétéroscédastique de White).

```
new;
library optmum;

n = 100;
let beta = 1 2 3;
sigma = 2;

x = 100*rndu(n,3);
y = x*beta + sigma * rndn(n,1);

proc logl(theta);
  local b,sigma2,L,u;

  b = theta[1:3];
  sigma2 = theta[4]^2;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;

  retp(L);
endp;

proc fmin(theta);
  retp( -sumc(logl(theta)) );
endp;

let sv = 1 1 1 1 ;
```

```
{theta_ml,f,g,retcode} = optmum(&fmin,sv);

b_ml = theta_ml[1:3];
sigma_ml = theta_ml[4];

H = -hessp(&fmin,theta_ml);
cov1_ml = invpd(-H);

J = gradp(&logl,theta_ml);
cov2_ml = invpd(J'J);

cov3_ml = invpd(-H)*(J'J)*invpd(-H);

stderr1_ml = sqrt(diag(cov1_ml));
stderr2_ml = sqrt(diag(cov2_ml));
stderr3_ml = sqrt(diag(cov3_ml));

print;
call printfmt(stderr1_ml~stderr2_ml~stderr3_ml,1);
```

– Considérons maintenant un modèle linéaire

$$y_i = x_i\beta + u_i \tag{35}$$

avec des résidus non "homoscédastiques"

$$u_i \sim \mathcal{N}\left(0, (\sigma_1 + \sigma_2 z_i)^2\right) \tag{36}$$

```
new;
library optmum;

n = 1000;
let beta = 1 2 3;
sigma = 2;

x = 100*rndu(n,3);
z = rndu(n,1);
u = rndn(n,1);
sigma = 2 + .5 * z;
y = x*beta + sigma .* u;

proc logl1(theta);
  local b,sigma2,L,u;

  b = theta[1:3];
  sigma2 = theta[4]^2;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*(u.*u)/sigma2;

  retp(L);
endp;

proc fmin1(theta);
```

```
  retp( -sumc(logl1(theta)) );
endp;

let sv = 1 1 1 1 ;

{theta1_ml,f,g,retcode} = optmum(&fmin1,sv);

proc logl2(theta);
  local b,sigma,L,u;

  b = theta[1:3];
  sigma = sqrt(theta[4]^2) + sqrt(theta[5]^2) * z;

  u = y - x*b;
  L = -0.5*ln(2*pi) - 0.5*ln(sigma^2) - 0.5*(u^2)./(sigma^2);

  retp(L);
endp;

proc fmin2(theta);
  retp( -sumc(logl2(theta)) );
endp;

let sv = 1 1 1 1 1;

{theta2_ml,f,g,retcode} = optmum(&fmin2,sv);
```

– Nous cherchons maintenant à construire un test de rapport de vraisemblance pour tester l'hypothèse $\rho = 0$ du modèle $y_t = x_t\beta + u_t$ avec $u_t = \rho u_{t-1} + \varepsilon_t$ et $\varepsilon_t \sim \mathcal{N}\left(0, \sigma^2\right)$. Nous utilisons la fonction de log-vraisemblance de Beach et MacKinnon [1978].

```
new;
library optmum,pgraph;

rndseed 1234;

Nobs = 5000;
u = recserar(rndn(Nobs,1),0,0.3);
x = 10+rndn(Nobs,1);
y = 2+3*x+u;

data = y ~ones(Nobs,1) ~x;

proc ml1(theta);
  local k,y,x,beta,rho,sigma,T,u,du,logl;

  k = cols(data) - 1;
  y =  data[.,1]; x = data[.,2:k+1];

  beta = theta[1:k]; rho = theta[k+1]; sigma = theta[k+2];

  T = rows(y);
  u = zeros(T,1);
```

```
  u = y-x*beta;
  du = u[2:T]-rho*u[1:T-1];

  logl  = zeros(T,1);
  logl[2:T] = -0.5*ln(2*pi)-0.5*ln(sigma^2)-0.5*(du.*du)./(sigma^2);

  /* Log-likelihood for the first observation */

  logl[1] = -0.5*ln(2*pi)-0.5*ln(sigma^2)-0.5*(1-rho^2)*(u[1]^2)/(sigma^2);

  retp(-sumc(logl));
endp;

proc ml2(theta);
  local k,y,x,beta,sigma,u,logl;

  k = cols(data) - 1;
  y =  data[.,1]; x = data[.,2:k+1];
  beta = theta[1:k]; sigma = theta[k+1];

  u = y-x*beta;
  logl = -0.5*ln(2*pi)-0.5*ln(sigma^2)-0.5*(u.*u)./(sigma^2);

  retp(-sumc(logl));
endp;

sv2 = data[.,1]/data[.,2 3];
sv1 = sv2|0|stdc(data[.,1]-data[.,2 3]*sv2);
sv2 = sv2 | sv1[4];

{theta1,Logl1,G1,retcode} = optmum(&ml1,sv1);
Logl1 = -Logl1;
G1 = -G1;
H1 = -hessp(&ml1,theta1);
{theta2,Logl2,g2,retcode} = optmum(&ml2,sv2);
Logl2 = -Logl2;
G2 = -G2;
H2 = -hessp(&ml2,theta2);

output file = ml5.out reset;

/*
**  Likelihood ratio statistic
**
**  DAVIDSON and MACKINNON [1993], Estimation and Inference in Econometrics
**  Oxford University Press, page 437, formula (13.06)
*/


LR = 2*(Logl1-Logl2);      /* Likelihood ratio                              */
pvalue = cdfchic(LR,1);    /* Approximating the noncentral chi-squared CDF */
```

```
print ftos(LR,''Likelihood ratio statistic: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);

output off;
```

Remarque 20 *A partir du programme précédent, il est très facile de construire les tests **LM** et **Wald**.*

– Le dernier exemple concerne le modèle **LOGIT**. C'est un modèle binomial $y_i = 0, 1$ avec

$$\Pr\{y_i = 1\} = \frac{1}{1 + \exp(-x_i\beta)} \tag{37}$$

```
/*
**> pdfLOGIT
*/

proc pdfLOGIT(x);
  retp( exp(-x) ./ ((1 + exp(-x))^2) );
endp;

/*
**> cdfLOGIT
*/

proc cdfLOGIT(x);
  retp( 1 ./ (1 + exp(-x)) );
endp;

/*
**> Logot_LogL
*/

proc Logit_LogL(data,beta);
  local y,x,xb,logl;

  y = data[.,1];
  x = data[.,2:cols(data)];
  xb = x*beta;
  logl = - ln( 1 + exp(-xb) ) - (1 - y) .* xb;

  retp(logl);
endp;

/*
**> Logit_Logl_gradp
*/

proc Logit_LogL_gradp(data,beta);
  local y,x,xb,grd;

  y = data[.,1];
  x = data[.,2:cols(data)];
  xb = x*beta;
```

```
  grd = (y - cdfLOGIT(xb)) .* x;

  retp(grd);
endp;

/*
**> Logit_Logl_gradp
*/

proc Logit_LogL_hessp(data,beta);
  local y,x,xb,hss;

  y = data[.,1];
  x = data[.,2:cols(data)];
  xb = x*beta;
  hss = -(x .* pdfLOGIT(xb))'x;

  retp(hss);
endp;
```

– Pour estimer un modèle **LOGIT**, on utilise généralement l'algorithme de Newton-Raphson car la convergence est très rapide.

```
new;
library optmum;

#include logit.src;

@

DATA FROM table 21.1  Greene 1993 - page 639 -
================================================
OBS  GPA    TUCE  PSI  GRADE  (32x5)

@

let data[32,5] =

  1   2.66     20    0      0
  2   2.89     22    0      0
  3   3.28     24    0      0
  4   2.92     12    0      0
  5   4.00     21    0      1
  6   2.86     17    0      0
  7   2.76     17    0      0
  8   2.87     21    0      0
  9   3.03     25    0      0
 10   3.92     29    0      1
 11   2.63     20    0      0
 12   3.32     23    0      0
 13   3.57     23    0      0
 14   3.26     25    0      1
 15   3.53     26    0      0
```

```
 16   2.74     19    0      0
 17   2.75     25    0      0
 18   2.83     19    0      0
 19   3.12     23    1      0
 20   3.16     25    1      1
 21   2.06     22    1      0
 22   3.62     28    1      1
 23   2.89     14    1      0
 24   3.51     26    1      0
 25   3.54     24    1      1
 26   2.83     27    1      1
 27   3.39     17    1      1
 28   2.67     24    1      0
 29   3.65     21    1      1
 30   4.00     23    1      1
 31   3.10     21    1      0
 32   2.39     19    1      1;

x = ones(32,1)~data[.,2 3 4];
y = data[.,5];

sv = y/x;

proc mlProc(theta);
  retp( -sumc(Logit_Logl(y~x,theta)) );
endp;

proc grdProc(theta);
  retp( -sumc(Logit_Logl_gradp(y~x,theta))' );
endp;

proc hssProc(theta);
  retp( -Logit_Logl_hessp(y~x,theta) );
endp;

_opgdprc = &grdProc;
_ophsprc = &hssProc;
_opalgr = 5;
{beta,f,g,retcode} = optmum(&mlProc,sv);
```

Remarque 21 *Nous pouvons aussi estimer le modèle* ***LOGIT*** *avec l'algorithme du score ou l'algorithme* ***BHHH****.*

25 Méthode des moments généralisés

25.1 Le principe de la méthode des moments

Nous considérons r moments empiriques recentrés $m(y_i;\theta)$. Soit

$$g(\theta) = \frac{1}{n}\sum_{i=1}^{n} m(y_i;\theta) \tag{38}$$

L'estimateur des moments $\hat{\theta}_{\text{MM}}$ est défini par

$$\hat{\theta}_{\text{MM}} = \arg\min g(\theta)^{\top} W g(\theta) \tag{39}$$

où W est une matrice de poids (qui est souvent une matrice identité).

– Considérons un exemple avec la distribution gaussienne.

```
new;
library optmum;

rndseed 123456;

ns = 1000;
m = 1;
sigma = 2;
z = m + rndn(ns,1)*sigma;

save data_gmm = z;

proc Moments(theta);
  local beta,m,sigma,h1,h2;

  m = theta[1];
  sigma = sqrt(theta[2]^2);

  /* first moment E[z] = m */
  h1 = z - m;

  /* second moment var[z] = sigma^2 */
  h2 = h1.*h1 - sigma^2;

  retp(h1~h2);
endp;

proc gmm_(theta);
  local g;

  g = meanc(Moments(theta));

  retp( g'g );
endp;

let sv = .1 .2;

{theta,f,g,retcode} = optmum(&gmm_,sv);
```

25.2 L'utilisation des moments simulés

Dans certains cas, il est difficile d'avoir des expressions analytiques pour les moments. Dans ce cas, nous pouvons utiliser des moments simulés.

– Reprenons l'exemple précédent.

```
new;
library optmum;
```

```
ns = 5000;
load z = data_gmm;

proc Moments(theta);
  local m,sigma,h1,h2,x;

  m = theta[1];
  sigma = sqrt(theta[2]^2);

  rndseed 123;
  x = m + sigma*rndn(ns,1);

  /* first moment */
  h1 = z - meanc(x);

  /* second moment */
  h2 = h1.*h1 - stdc(x)^2;

  retp(h1~h2);
endp;

proc gmm_(theta);
  local g;

  g = meanc(Moments(theta));

  retp( g'g );
endp;

let sv = .1 .2;

{theta,f,g,retcode} = optmum(&gmm_,sv);
```

– Nous supposons maintenant que Z est la somme de deux variables aléatoires

$$Z = Z_1 + Z_2 \tag{40}$$

avec

$$Z_1 \sim \mathcal{N}\left(m, \sigma^2\right) \tag{41}$$

et

$$Z_2 \sim \mathbf{t}_5 \tag{42}$$

Dans ce cas, nous pouvons calculer les moments de Z à partir des moments de Z_1 et Z_2. Il est plus facile d'utiliser des moments simulés.

```
new;
library optmum;

ns = 5000;
load z = data_gmm;

proc Moments(theta);
  local m,sigma,h1,h2,x;
```

```
  m = theta[1];
  sigma = sqrt(theta[2]^2);

  rndseed 123;
  x = m + sigma*rndn(ns,1);
  x = x + cdftci(rndu(ns,1),5);

  /* first moment */
  h1 = z - meanc(x);

  /* second moment */
  h2 = h1.*h1 - stdc(x)^2;

  retp(h1~h2);
endp;

proc gmm_(theta);
  local g;

  g = meanc(Moments(theta));

  retp( g'g );
endp;

let sv = .1 .2;

{theta,f,g,retcode} = optmum(&gmm_,sv);
```

25.3 La méthode des moments généralisés

Dans la méthode des moments généralisés, nous considérons un vecteur θ de paramètres de dimension k et r conditions d'orthogonalité (pas forcément des moments) :

$$\mathbb{E}\left[h\left(y_i;\theta\right)\right] = \mathbf{0} \tag{43}$$

Soit

$$g\left(\theta\right) = \frac{1}{n}\sum_{i=1}^{n} h\left(y_i;\theta\right) \tag{44}$$

L'estimateur des moments généralisés $\hat{\theta}_{\text{GMM}}$ est défini par

$$\hat{\theta}_{\text{GMM}} = \arg\min Q\left(\theta\right) \tag{45}$$

avec $Q\left(\theta\right) = \left(\theta\right)^{\top} W g\left(\theta\right)$ et W une matrice de poids. En général, la matrice des poids W correspond à l'inverse de la matrice de covariance Φ des conditions d'orthogonalité (dans ce cas, nous obtenons la matrice des poids "optimale"). Dans ce cas, on montre que

$$\text{cov}\left[\hat{\theta}_{\text{GMM}}\right] = \frac{1}{n}\left(\hat{D}^{\top}\hat{\Phi}^{-1}\hat{D}\right)^{-1} \tag{46}$$

avec

$$\hat{D} = \frac{\partial}{\partial\theta^{\top}} g\left(\hat{\theta}_{\text{GMM}}\right) \tag{47}$$

Dans le cas où $r > k$, Hansen [1982] suggère le test suivant de suridentification

$$nQ\left(\hat{\theta}_{\text{GMM}}\right) \sim \chi^2_{r-k} \tag{48}$$

– La procédure `GMM` nécessite de définir la procédure `h` qui retourne la matrice $n \times r$ des conditions d'orthogonalité, ainsi que le vecteur `sv` des valeurs de départ de dimension $k \times 1$. La convergence est controlée par deux variables globales `_gmm_MaxIters` et `_gmm_tol`. Si `_gmm_W` est initialisé à 0, alors la procédure `GMM` utilise la matrice Φ pour calculer la matrice des poids W.

```
declare matrix __gmm_h;
declare matrix __gmm_W;

declare matrix _gmm_Iters;
declare matrix _gmm_J;
declare matrix _gmm_retcode;

declare matrix _gmm_MaxIters = 5;
declare matrix _gmm_tol = 1e-5;
declare matrix _gmm_output = 1;
declare matrix _gmm_mtd = 1;
declare matrix _gmm_W = 0;

proc (4) = GMM(h,sv);
  local h0,r,n,k;
  local theta,stderr,tstudent,pvalue;
  local cov,Q,J,grd,D,df,Phi,oldtrap;
  local st,parnm,omat,mask,fmt;

  __gmm_h = h;
  h0 = _gmm_h(sv);
  r = cols(h0); n = rows(h0); k = rows(sv);

  if _gmm_W == 0;
    __gmm_W = eye(r);
  else;
    __gmm_W = _gmm_W;
  endif;

  _gmm_Iters = 1;
  do until _gmm_Iters > _gmm_MaxIters;

    {theta,Q,grd,_gmm_retcode} = optmum(&_gmm_Q,sv);

    if _gmm_retcode == 1;
      break;
    endif;

    if _gmm_retcode /= 0;
      retp(error(0),error(0),error(0),error(0));
    endif;

    if maxc(abs(theta-sv)) < _gmm_tol;
      break;
```

```
  endif;

  if _gmm_W == 0;
    Phi = _gmm_Phi(theta);

    oldtrap = trapchk(1);
    trap 1,1;
    if _gmm_mtd /= 1;
      __gmm_W = invpd(diagrv(eye(r),diag(Phi)));
    else;
      __gmm_W = invpd(Phi);
    endif;
    trap oldtrap,1;
    if scalerr(__gmm_W);
      __gmm_W = pinv(Phi);
    endif;
  else;
    __gmm_W = __gmm_W;
  endif;

  sv = theta;

  _gmm_Iters = _gmm_Iters + 1;
endo;

df = n - k;
D = gradp(&_gmm_g,theta);
oldtrap = trapchk(1);
trap 1,1;
cov = invpd(D'__gmm_W*D)/n;
trap oldtrap,1;
if scalerr(cov);
  cov = miss(zeros(k,k),0);
endif;
if hasimag(cov) == 0;
  cov = real(cov);
else;
  cov = miss(zeros(k,k),0);
endif;

stderr = sqrt(diag(cov));
if iscplx(stderr);
  cov = cov = miss(zeros(k,k),0);
  stderr = diag(cov);
endif;

if r > k;
  J = n*Q;
  pvalue = cdfchic(J,r-k);
else;
  J = error(0);
  pvalue = error(0);
```

```
  endif;

  _gmm_J = 0;
  _gmm_J = vput(_gmm_J,J,''J'');
  _gmm_J = vput(_gmm_J,pvalue,''pvalue'');

  if _gmm_output;

    call header(''GMM - Generalized Method of Moments'','''',0);
    print;

    st = ''Usable observations:                              %*.*lf  '';
    print ftos(n,st,15,0);
    st = ''Number of parameters:                             %*.*lf  '';
    print ftos(k,st,15,0);
    st = ''Number of moments:                                %*.*lf  '';
    print ftos(r,st,15,0);
    st = ''Degrees of freedom:                               %*.*lf  '';
    print ftos(df,st,15,0);
    print;

    st = ''Value of the criterion function:                  %*.*lf  '';
    print ftos(Q,st,15,5);
    if not ismiss(J);
      st = ''Hansen's test of overidentifying restrictions:   %*.*lf  '';
      print ftos(vread(_gmm_J,''J''),st,15,5);
      st = ''p-value:                                          %*.*lf  '';
      print ftos(vread(_gmm_J,''pvalue''),st,15,5);
    endif;

    print;  print;
    print ''Parameters      estimates       std.err.    ''\
          ''  t-statistic         p-value   '';
    print ''----------------------------------------''\
          ''---------------------------------'';
    mask=0~1~1~1~1;
    let fmt[5,3]=  ''-*.*s''  8 8  ''*.*lf'' 16 6 ''*.*lf'' 16 6
                   ''*.*lf'' 17 6  ''*.*lf'' 16 6;
    tstudent = theta ./ miss(stderr,0);
    pvalue = 2*cdftc(abs(tstudent),df);
    parnm = 0 $+ ''P'' $+ ftocv(seqa(1,1,k),2,0);
    omat = parnm~theta~stderr~tstudent~pvalue;
    call printfm(omat,mask,fmt);

  endif;

  retp(theta,stderr,cov,Q);
endp;

proc (1) = _gmm_h(theta);
  local h;
  h = __gmm_h;
```

```
  local h:proc;
  retp( packr(h(theta)) );
endp;

proc (1) = _gmm_g(theta);
  retp( meanc(_gmm_h(theta)) );
endp;

proc (1) = _gmm_Q(theta);
  local g;
  g = _gmm_g(theta);
  retp( g'__gmm_W*g );
endp;

proc (1) = _gmm_Phi(theta);
  local h;
  h = _gmm_h(theta);
  retp( (h'h)/rows(h) );
endp;
```

25.4 Un exemple ARCH

– Un exemple sans suridentification.

```
new;
library ritme,optmum;

cls;

rndseed 123;

n = 250;
alpha0 = 0.5; alpha1 = 0.6;

u = zeros(n,1);
h = zeros(n,1);
i = 2;
do until i > n;
  h[i] = sqrt(alpha0^2 + alpha1^2*u[i-1]^2);
  u[i] = rndn(1,1)*h[i];
  i = i + 1;
endo;

x =  3*rndu(n,1);
y = 2 + 3*x + u;

proc ARCH(theta);
  local M,u,h2,u2;

  M = zeros(n,4);
  u = y - theta[1] - theta[2] * x;

  /* first moment */
```

```
  M[.,1] = u;

  /* computing h(t)^2 */
  u2 = u .* u;
  h2 = theta[3]^2 + theta[4]^2 * lag1(u2);
  h2[1] = theta[3]^2;

  /* second moment */
  M[.,2] = u2 - h2;

  /* u(t) uncorrelated with x(t) */
  M[.,3] = x .* u;

  /* u(t)^2-h(t)^2 uncorrelated with u(t-i)^2, i = 1 */
  M[.,4] = (u2-h2).*lagn(u2,1);

  retp(M);
endp;

sv = 2|3|0.5|0.6;
_gmm_tol = 1e-10;
{theta,stderr,cov,Q} = gmm(&arch,sv);
```

– Un exemple avec suridentification (2 conditions d'orthogonalité supplémentaires).

```
new;
library ritme,optmum;

cls;

rndseed 123;

n = 1000;
alpha0 = 0.5; alpha1 = 0.5;

u = zeros(n,1);
h = zeros(n,1);
i = 2;
do until i > n;
  h[i] = sqrt(alpha0^2 + alpha1^2*u[i-1]^2);
  u[i] = rndn(1,1)*h[i];
  i = i + 1;
endo;

x =  3*rndu(n,1);
y = 2 + 3*x + u;

proc ARCH(theta);
  local M,u,h2,u2,i;

  M = zeros(n,6);
  u = y - theta[1] - theta[2] * x;
```

```
  /* first moment */
  M[.,1] = u;

  /* computing h(t)^2 */
  u2 = u .* u;
  h2 = theta[3]^2 + theta[4]^2 * lag1(u2);
  h2[1] = theta[3]^2;

  /* second moment */
  M[.,2] = u2 - h2;

  /* u(t) uncorrelated with x(t) */
  M[.,3] = x .* u;

  /* u(t)^2-h(t)^2 uncorrelated with u(t-i)^2, i = 1,2,3 */
  i = 1;
  do until i > 3;
    M[.,3+i] = (u2-h2).*lagn(u2,i);
    i = i + 1;
  endo;

  retp(M);
endp;

sv = 2|3|0.5|0.6;
_gmm_tol = 1e-10;
_gmm_MaxIters = 10;
{theta,stderr,cov,Q} = gmm(&arch,sv);
```

25.5 Les variables instrumentales

La régression linéaire est un cas particulier d'estimation **GMM** en considérant les conditions d'orthogonalité suivantes :

$$\begin{cases} \mathbb{E}\left[u_i\right] = 0 \\ \mathbb{E}\left[u_i^2 - \sigma^2\right] = 0 \\ x_i \perp u_i \end{cases} \tag{49}$$

Lorsque nous utilisons des variables instrumentales, nous remplaçons la dernière condition par

$$z_i \perp u_i \tag{50}$$

avec z_i les variables instrumentales.

```
new;
library ritme,optmum;

cls;

rndseed 123;

n = 1000;
x = 10*rndu(n,4);
z = x^2;
```

```
beta = rndn(4,1);
sigma = 2;
y = x*beta + rndn(n,1)*sigma;

sv = beta|sigma; /* starting values */

proc H1(theta);
  local beta,sigma,u,M,i;

  M = zeros(n,6);

  beta = theta[1:4];
  sigma = theta[5];

  /* first moment */
  u = y - x*beta;
  M[.,1] = u;
  /* second moment */
  M[.,2] = u.*u - sigma^2;

  i = 1;
  do until i > 4;
    M[.,2+i] = u.*x[.,i];
    i = i + 1;
  endo;

  retp(M);
endp;

proc H2(theta);
  local beta,sigma,u,M,i;

  M = zeros(n,6);

  beta = theta[1:4];
  sigma = theta[5];

  /* first moment */
  u = y - x*beta;
  M[.,1] = u;
  /* second moment */
  M[.,2] = u.*u - sigma^2;

  i = 1;
  do until i > 4;
    M[.,2+i] = u.*z[.,i];
    i = i + 1;
  endo;

  retp(M);
endp;
```

```
_gmm_tol = 1e-7;

{theta1,stderr,cov,Q} = gmm(&h1,sv);

{theta2,stderr,cov,Q} = gmm(&h2,sv);

print theta1~theta2;

print ''OLS'' invpd(x'x)*x'y;
print ''IV'' invpd(x'*z*invpd(z'*z)*z'*x)*x'*z*invpd(z'*z)*z'*y;
```

26 Maximum de vraisemblance simulé

26.1 Calcul de probabilités par simulation

26.1.1 Un premier exemple

Soit un vecteur aléatoire gaussien standard (X, Y) de corrélation. Nous cherchons à évaluer la probabilité p définie par

$$p = \Pr\{(X, Y) \in [a, b] \times [c, d]\} \tag{51}$$

Notons $\phi(x, y; \rho)$ la fonction de densité de la distribution gaussienne bivariée de corrélation ρ. Nous avons

$$p = \int_a^b \int_c^d \phi(x, y; \rho)\, \mathrm{d}x\, \mathrm{d}y \tag{52}$$

Nous pouvons évaluer cette double intégrale en utilisant la procédure `intquad2`. Notons aussi qu'il existe une commande spécifique `cdfbvn2` qui permet de calculer cette probabilité. La troisième méthode est basée sur la convergence

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(X_i^\star, Y_i^\star) = \int_\Omega f(x, y)\, \mathrm{d}x\, \mathrm{d}y \tag{53}$$

où $(X_i^\star, Y_i^\star)$ sont des vecteurs aléatoires *i.i.d.* dont la distribution est la loi uniforme sur Ω. La quatrième méthode utilise le fait que p est une probabilité. Dans les premières méthodes, p est un calcul d'intégrale.

```
new;

declare matrix rho;

fn rndu2(r,c,a,b) = a + (b-a) .* rndu(r,c);

proc pdfn2(x1,x2,mu1,mu2,sigma1,sigma2,rho);
  local w,x12,pdf;

  w = 1 - rho^2;

  x1 = (x1 - mu1)/sigma1;
  x2 = (x2 - mu2)/sigma2;
  pdf = x1^2 - 2 * rho * x1 .* x2 + x2^2;
  pdf = exp(-0.5*pdf/w)/(2*pi*sigma1*sigma2*sqrt(w));

  retp(pdf);
endp;
```

```
fn pdfn2_cr(x,y,rho) = pdfn2(x,y,0,0,1,1,rho);

fn FnInt(x,y) = pdfn2_cr(x,y,rho);

I = zeros(4,1);


/*
** Carre [0,1] x [0,1]
*/

rho = 0.5;

_intord = 40;
I[1] = intquad2(&FnInt,1|0,1|0);    @ Methode I : intquad1  @
I[2] = cdfbvn2(0,1,0,1,rho);        @ Methode II : cdfbvn2   @

ns = 5000;
x = rndu(ns,1); y = rndu(ns,1);
aire = 1;
I[3] = aire * meanc(pdfn2(x,y,0,0,1,1,rho));  @ Method III : calcul d'une aire
                                                              par simulation          @
x = rndn(ns,1);
y = rho * x + sqrt(1-rho^2) * rndn(ns,1);
I[4] = meanc( x .>= 0 .and x .<= 1
           .and y .>= 0 .and y .<= 1);        @ Method IV : calcul d'une probabilite
                                                            par simulation            @

print I;
print;

/*
** Carre [-1,1] x [-1,1]
*/

rho = 0.5;
I[1] = intquad2(&FnInt,1|-1,1|-1);
I[2] = cdfbvn2(-1,2,-1,2,rho);

ns = 5000;
x = rndu2(ns,1,-1,1); y = rndu2(ns,1,-1,1);
aire = 4;
I[3] = aire * meanc(pdfn2(x,y,0,0,1,1,rho));

x = rndn(ns,1);
y = rho * x + sqrt(1-rho^2) * rndn(ns,1);
I[4] = meanc( x .>= -1 .and x .<= 1 .and y .>= -1 .and y .<= 1);

print I;
```

26.1.2 Un second exemple

Considérons un problème un peu plus compliqué. Dans le programme précédent, il est facile de calculer la probabilité p car Ω est un carré

$$\Omega = [a, b] \times [c, d] \tag{54}$$

Si Ω n'est pas un carré, il peut être difficile de procéder à une intégration numérique directe par la méthodes des quadratures. De même, la simulation du vecteur uniforme de distribution $\mathcal{U}_\Omega$ n'est pas forcément aisée. Dans ce cas, le plus simple est d'utiliser l'algorithme d'acceptation-rejet pour évaluer la probabilité

$$p = \Pr\{(X, Y) \in \Omega\}$$

En effet, nous avons

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\left\{(X_i^\bullet, Y_i^\bullet) \in \Omega\right\} \cdot f\left(X_i^\bullet, Y_i^\bullet\right) = \int_\Omega f(x, y)\, \mathrm{d}x\, \mathrm{d}y \tag{55}$$

où $(X_i^\bullet, Y_i^\bullet)$ sont des vecteurs aléatoires *i.i.d.* dont la distribution est la loi uniforme sur Δ ($\Omega \subset \Delta$). Dans le programme qui suit, nous implémentons cette algorithme lorsque Ω est un disque.

```
new;

fn rndu2(r,c,a,b) = a + (b-a) .* rndu(r,c);

proc pdfn2(x1,x2,mu1,mu2,sigma1,sigma2,rho);
  local w,x12,pdf;

  w = 1 - rho^2;

  x1 = (x1 - mu1)/sigma1;
  x2 = (x2 - mu2)/sigma2;
  pdf = x1^2 - 2 * rho * x1 .* x2 + x2^2;
  pdf = exp(-0.5*pdf/w)/(2*pi*sigma1*sigma2*sqrt(w));

  retp(pdf);
endp;

fn pdfn2_cr(x,y,rho) = pdfn2(x,y,0,0,1,1,rho);

r = 1;

rho = 0.5;

ns = 50000;
x = rndu2(ns,1,-r,r);
y = rndu2(ns,1,-r,r);
e = x^2 + y^2 .<= r^2;
aire = 4*r^2;
I = aire * sumc(pdfn2_cr(x,y,rho) .* e)/ns;

print r~I;
```

26.1.3 Un troisième exemple

Nous considérons le problème classique de calcul de probabilités sur une sphère de centre $(0, 0, 0)$ et de rayon r :

$$\mathcal{S} = \left\{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 \leq r^2\right\} \tag{56}$$

Nous pouvons reformuler $\mathcal{S}$ de la façon suivante :

$$\mathcal{S} = \left\{(x,y) \in \mathcal{S}', -\sqrt{r^2 - x^2 - y^2} \leq z \leq \sqrt{r^2 - x^2 - y^2}\right\} \tag{57}$$

avec

$$\mathcal{S}' = \left\{x \in [-r, r], -\sqrt{r^2 - x^2} \leq y \leq \sqrt{r^2 - x^2}\right\} \tag{58}$$

Dans le programme suivant, nous considérons un vecteur aléatoire gaussien (X, Y, Z) de moyenne μ et de matrice de covariance Σ et nous cherchons à évaluer la probabilité

$$\Pr\{(X, Y, Z) \in \mathcal{S}\} \tag{59}$$

Pour cela, nous utilisons la procédure `intgrat3`.

```
new;

declare matrix r, Mu, Sigma;

fn h1(x,y) = sqrt(r^2 - x^2 - y^2);
fn h2(x,y) = -h1(x,y);

fn g1(x) = h1(x,0);
fn g2(x) = -g1(x);

proc pdfn3(x,y,z);
  local n,pdf,t,invSigma,i;

  n = rows(z);
  pdf = zeros(n,1);

  t = x|y|0;
  invSigma = invpd(Sigma);

  i = 1;
  do until i > n;
    t[3] = z[i];
    pdf[i] = (t-Mu)'*invSigma*(t-Mu);
    i = i + 1;
  endo;

  retp( exp( -0.5*pdf) / (2*pi)^(3/2) / sqrt(detl) );
endp;

rayon = seqa(0.5,0.5,10);
Pr = zeros(10,2);

Mu = zeros(3,1);
Sigma = eye(3);

i = 1;
do until i > 10;
  _intrec = 0;
  _intord = 40;
  r = rayon[i];
```

```
  Pr[i,1] = intgrat3(&pdfn3,r|-r,&g1|&g2,&h1|&h2);
  i = i + 1;
endo;

let Mu = 1 -1 2;
let Sigma[3,3] = 1.0  0.5  0.25
                 0.5  1.0  0.15
                 0.25 0.15 1.0;

i = 1;
do until i > 10;
  _intrec = 0;
  _intord = 40;
  r = rayon[i];
  Pr[i,2] = intgrat3(&pdfn3,r|-r,&g1|&g2,&h1|&h2);
  i = i + 1;
endo;


output file = sphere1.out reset;

print ''============================================================='';
print ''           rayon               Probabilite      '';
print ''                             Sigma1           Sigma2     '';
print ''============================================================='';

call printfmt(rayon~Pr,1);

output off;
```

```
============================================================
           rayon               Probabilite
                             Sigma1           Sigma2
============================================================
             0.5      0.030859827      0.001106328
               1       0.19874932      0.012295476
             1.5       0.47783511      0.057167117
               2       0.73853814       0.16751465
             2.5        0.8999406       0.35198809
               3       0.97070974       0.57228075
             3.5       0.99342616       0.76627256
               4       0.99886606       0.89532937
             4.5       0.99984936       0.96157493
               5       0.99998456       0.98835233
```

– En fait, il est beaucoup plus facile d'estimer ces probabilités par la méthode de Monte Carlo. Pour cela, nous utilisons l'algorithme d'acceptation-rejet.

```
new;

cls;
```

```
declare matrix r, Mu, Sigma;

proc pdfn3(x,y,z);
  local n,pdf,t,invSigma,i;

  n = rows(z);
  pdf = zeros(n,1);

  invSigma = invpd(Sigma);

  i = 1;
  do until i > n;
    t = x[i]|y[i]|z[i];
    pdf[i] = (t-Mu)'*invSigma*(t-Mu);
    i = i + 1;
  endo;

  retp( exp( -0.5*pdf) / (2*pi)^(3/2) / sqrt(detl) );
endp;

fn rndu2(r,c,a,b) = a + (b-a) .* rndu(r,c);

proc MonteCarlo(ns);
  local I,iter,x,y,z,in,cube;

  ns = ns | 1;
  I = 0;
  iter = ns;
  do until iter > ns[2];
    x = rndu2(ns[1],1,-r,r);
    y = rndu2(ns[1],1,-r,r);
    z = rndu2(ns[1],1,-r,r);
    in = selif(x~y~z, (x^2 + y^2 + z^2) .<= (r^2) );
    if in == error(0);
      iter = iter + 1;
      continue;
    endif;
    I = I + sumc(pdfn3(in[.,1],in[.,2],in[.,3]));
    iter = iter + 1;
  endo;

  cube = 8 * r^3;
  I = cube * I /(ns[1]*ns[2]);

  retp(I);
endp;


rayon = seqa(0.5,0.5,10);
Pr = zeros(10,2);

let ns = 5000 100;
```

```
Mu = zeros(3,1);
Sigma = eye(3);

i = 1;
do until i > 10;
  r = rayon[i];
  Pr[i,1] = MonteCarlo(ns);
  print /flush i;
  i = i + 1;
endo;

let Mu = 1 -1 2;
let Sigma[3,3] = 1.0  0.5  0.25
                 0.5  1.0  0.15
                 0.25 0.15 1.0;

i = 1;
do until i > 10;
  r = rayon[i];
  Pr[i,2] = MonteCarlo(ns);
  print /flush i;
  i = i + 1;
endo;


output file = sphere2.out reset;

print ''============================================================='';
print ''           rayon              Probabilite      '';
print ''                            Sigma1           Sigma2     '';
print ''============================================================='';

call printfmt(rayon~Pr,1);

output off;
```

26.2 Calcul d'espérances conditionnelles par simulation

Nous pouvons bien sûr utiliser les méthodes précédentes pour calculer des espérances conditionnelles. Dans ce paragraphe, nous ne développons que la quatrième méthode. Considérons un vecteur aléatoire gaussien standard (X, Y) de corrélation ρ. Nous cherchons à évaluer l'espérance conditionnelle $\mathcal{E}$ avec

$$\mathcal{E} = \mathbb{E}\left[X \mid Y \in \Omega\right] \tag{60}$$

Dans le programme suivant, nous calculons

$$\mathcal{E} = \mathbb{E}\left[X \mid Y > 1\right]$$

```
new;

ns = 50000;
rho = 0.5;
```

```
x = rndn(ns,1);
y = rho * x + sqrt(1-rho^2) * rndn(ns,1);

cnd = y .> 1;
E = sumc( cnd .* x) / sumc(cnd);
print E;
```

– Voici un exemple un peu plus complexe. Nous calculons

$$\mathcal{E} = \mathbb{E}\left[X_1 X_2^2 + X_3^2 \mid (X_3, X_4) \notin \mathcal{D}\left((0,0),1\right)\right]$$

avec (X_1, X_2, X_3, X_4) un vecteur aléatoire gaussien standard de matrice de corrélation ρ et $\mathcal{D}\left((0,0),1\right)$ le disque centré en $(0,0)$ et de rayon 1.

```
new;

proc (1) = rndmn_cr(rho,ns);
  retp( rndn(ns,rows(rho))*chol(rho) );
endp;

ns = 50000;

let rho[4,4] = 1.0 0.5 0.5 0.5
               0.5 1.0 0.5 0.5
               0.5 0.5 1.0 0.5
               0.5 0.5 0.5 1.0;

x = rndmn_cr(rho,10000);

cnd = x[.,3]^2 + x[.,4]^2 .> 1^2;
E = sumc( cnd .* (x[.,1] .* x[.,2]^2 + x[.,3]^2) ) / sumc(cnd);
print E;
```

26.3 Un exemple simple de maximum de vraisemblance simulé

– Un exemple qui n'appelle pas d'explications.

```
new;
library optmum,pgraph;

x = rndp(100,1,6.5);

fn negML(lambda) = -sumc(-lambda + x .* ln(lambda) - ln(x!) );

{lambdaML,Logl,g,retcode} = optmum(&negML,1);

proc SML_Logl(lambda);
  local N,nx,Pr,i,Logl;

  rndseed 123;
  N = rndp(1000,1,lambda);

  nx = rows(x);
  Pr = zeros(nx,1);
```

```
  i = 1;
  do until i > nx;
    Pr[i] = meanc( N .== x[i]);
    i = i + 1;
  endo;

  Logl = ln(Pr+__macheps);

  retp( Logl );
endp;

fn SML(lambda) = sumc(SML_Logl(lambda));

Lambda = seqa(0.1,0.1,200);
Logl = zeros(200,1);

i = 1;
do until i > 200;
  Logl[i] = SML(Lambda[i]);
  i = i + 1;
endo;

LambdaSML = Lambda[maxindc(Logl)];

print lambdaML~lambdaSML;


graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0; _plwidth = 5;
  fonts(''simplex simgrma'');
  xlabel(''\214\2021\201'');
  ylabel(''\214Simulated log-likelihood function'');
  xtics(0,20,2.5,5);
  ytics(-3500,0,500,2);
  graphprt(''-c=1 -cf=sml1.ps'');
  xy(lambda,Logl);
```

26.4 Le simulateur GHK

La procédure

```
proc (1) = rndGHK(Mu,Sigma,a,b,u);
  local k,c,j,i,Ta,Tb,dT,T,w;

  k = rows(Mu);
  c = chol(Sigma)' + __macheps;
  u = u';

  a = a - Mu;
  b = b - Mu;
```

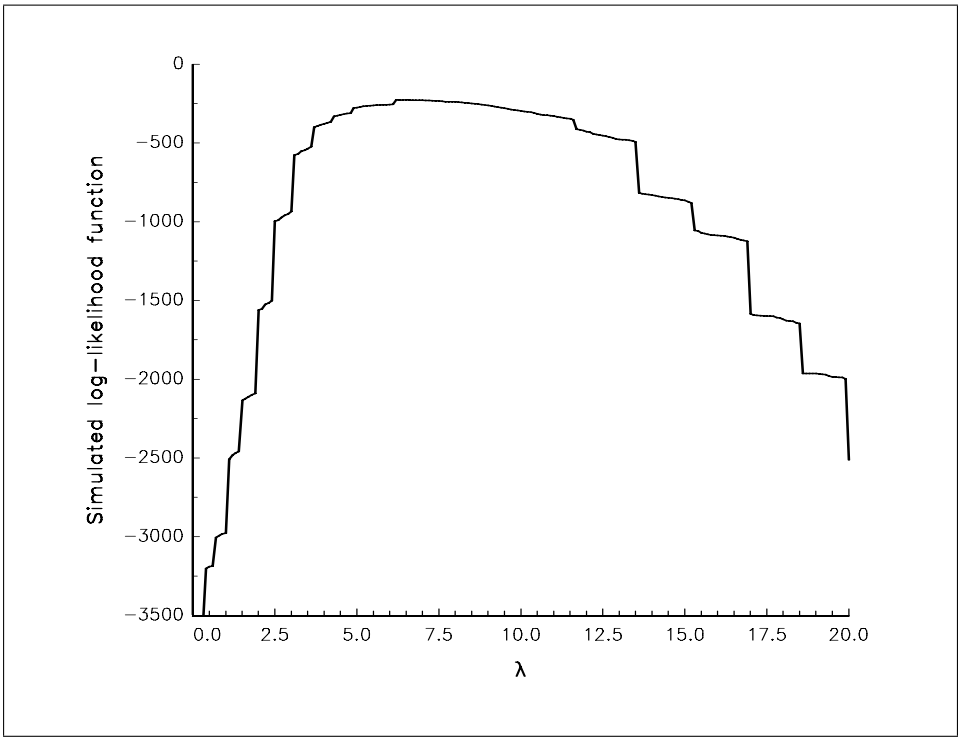


FIG. 4 – Log-vraisemblance simulée d'un modèle de Poisson

```
  Ta = cdfn( a[1] / c[1,1] );
  Tb = cdfn( b[1] / c[1,1] );
  dT = Tb - Ta;
  T = cdfni( Tb - u[1,.] .* dT );
  w = dT;

  j = 2;
  do until j > k;
    i = seqa(1,1,j-1);
    Ta = cdfn( (a[j] - c[j,i]*T) / c[j,j] );
    Tb = cdfn( (b[j] - c[j,i]*T) / c[j,j] );
    dT = Tb - Ta;
    T = T | cdfni( Tb - u[j,.] .* dT );
    w = w .* dT;
    j = j + 1;
  endo;

  retp( meanc(w') );
endp;
```

L'exemple

```
new;

#include ghk.src;

Let Mu = 1 2 3;
```

```
let Sigma[3,3] = 1.0 0.5 0.2
                 0.5 1.0 0.2
                 0.2 0.2 1.0;

let a = 1 1 1;
let b = 3 3 3;

u = rndu(1000,3);
print rndGHK(Mu,Sigma,a,b,u);
```

27 Données de panel

27.1 Une petite introduction

Considérons les données de JUDGE, HILL, GRIFFITHS, LÜTKEPOHL et LEE [1988]. Celles-ci sont stockées dans la base de données GAUSS *jdata.dat*. Nous avons 40 observations et 3 variables. La première variable X1 indique le groupe. Les seconde et troisième variables sont respectivement la variable explicative et la variable expliquée. Voici la visualisation d'une partie de la base de données avec la commande `datalist`.

```
------------------------ c:\gauss\gro\ritme/jdata.dat -------------------------

                 X1        X2        X3
        #1     1.00      43.7      38.5
        #2     1.00      45.9      35.3
        #3     1.00      4.74      3.78
        #4     1.00      40.6      35.3
        #5     1.00      25.9      20.8
        #6     1.00      36.0      36.7
        #7     1.00      50.9      41.7
        #8     1.00      42.5      30.7
        #9     1.00      25.6      23.7
       #10     1.00      49.8      39.5
       #11     2.00      51.0      32.5
       #12     2.00      27.8      18.7
       #13     2.00      35.7      27.0
       #14     2.00      35.8      18.7
       #15     2.00      43.3      25.6
       #16     2.00      48.5      39.2
       #17     2.00      64.2      47.7
       #18     2.00      38.3      27.0
       #19     2.00      45.4      33.6
       #20     2.00      43.7      27.3
------------------------------------------------------------------------------

PgDn  PgUp  Home  End             Ctrl-   Ctrl-   ?  Esc  [Esc]
```

– Nous cherchons à déterminer le nombre de groupes ainsi que le nombre d'observations de chacun des groupes. Pour cela, nous utilisons les commandes `unique` et `counts`.

```
new;
library pgraph;

data = loadd(''jdata'');
```

```
grp = data[.,1]; x = data[.,2]; y = data[.,3];

mod = unique(grp,1);
frq = counts(grp,mod);

call printfmt(mod~frq,1);
```

– Nous considérons dans un premier temps un modèle de la forme

$$y_{i,t} = \beta x_{i,t} + v_{i,t} \tag{61}$$

avec $v_{i,t} \sim \mathcal{N}\left(0, \sigma^2\right)$. Le résultat de la régression linéaire est reportée sur le graphique 5.

```
new;
library pgraph;

data = loadd(''jdata'');

grp = data[.,1]; x = data[.,2]; y = data[.,3];

nobs = rows(data);
beta = y / x;

z = seqa(0,70/(nobs-1),nobs);

graphset;
  _pdate = ''''; _pframe = 0; _pcross = 1; _pnum = 2;
  _plctrl = -1|0; _pstype = 9; _psymsiz = 3.5;
  _pltype = 6; _plwidth = 5;
  xlabel(''\214x'');
  ylabel(''\214y'');
  graphprt(''-c=1 -cf=panel2.ps'');
  xy(x~z,y~beta*z);
```

– Nous considérons maintenant un modèle de la forme

$$y_{i,t} = \beta_i x_{i,t} + v_{i,t} \tag{62}$$

avec $v_{i,t} \sim \mathcal{N}\left(0, \sigma_i^2\right)$. Comme nous avons 4 groupes (ou 4 individus), ce modèle est équivalent à 4 modèles linéaires **totalement indépendants** (voir le graphique 6).

```
new;
library pgraph;

data = loadd(''jdata'');

beta = zeros(4,1);
x = zeros(10,4);
y = zeros(10,4);

i = 1;
do while i <= 4;
  d = selif(data[.,2 3],data[.,1] .== i);
  x[.,i] = d[.,1];
  y[.,i] = d[.,2];
```

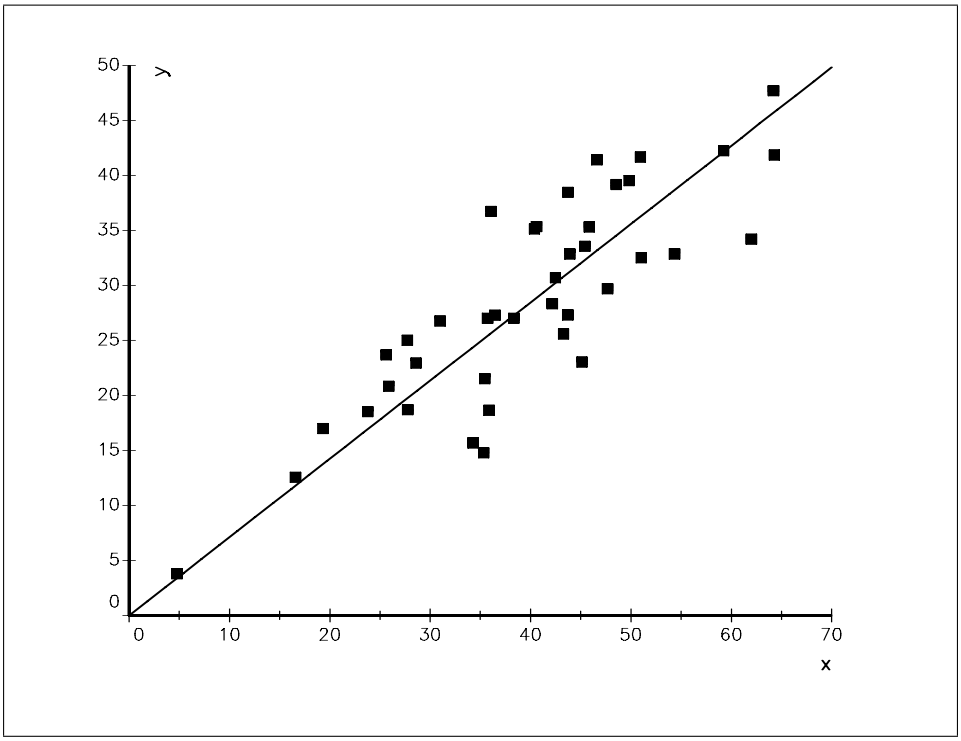


FIG. 5 – Régression linéaire sur l'ensemble des observations

```
  beta[i] = y[.,i] / x[.,i];
  i = i + 1;
endo;

print beta;

graphset;

  begwind;
  window(2,2,0);

    _pdate = ''''; _pframe = 0; _pcross = 1; _pnum = 2;
    _plctrl = -1; _pstype = 9; _psymsiz = 3.5;
    xlabel(''\214x'');
    ylabel(''\214y'');
    xtics(0,70,10,2);
    ytics(0,50,10,10);
    _paxht = 0.25; _pnumht = 0.25; _ptitlht = 0.30;

  i = 1;
  do until i > 4;
    setwind(i);
    title(ftos(i,''\214Groupe = %lf'',1,0));
    _pline = 1~3~0~0~70~beta[i]*70~1~9~5;
    xy(x[.,i],y[.,i]);
```

```
  i = i + 1;
endo;

graphprt(''-c=1 -cf=panel3.ps'');

endwind;
```

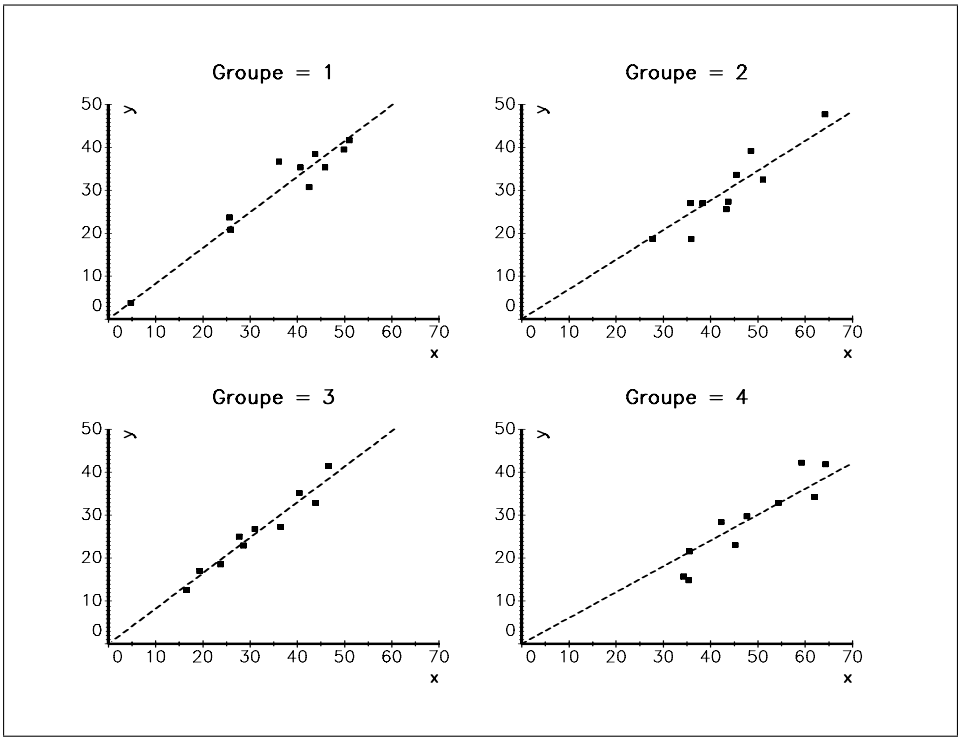


FIG. 6 – Régression linéaire de chaque groupe

– Les modèles de panel sont en fait des modèles intermédiaires entre les deux cas extrêmes précédents (totale dépendance / totale indépendance). Considérons par exemple un modèle de la forme

$$y_{i,t} = \beta x_{i,t} + \gamma_i + v_{i,t} \tag{63}$$

avec $v_{i,t} \sim \mathcal{N}\left(0, \sigma^2\right)$. Nous avons un paramètre commun β ("effet commun") et des paramètres spécifiques γ_i $(i = 1, \ldots, n)$. Remarquez néanmoins que la variance de l'erreur est constante et ne dépend pas du groupe (ou de l'individu).

```
new;
library pgraph;

data = loadd(''jdata'');

y = data[.,3];
x = ones(rows(data),1)~data[.,2];

bols = invpd(x'x)*x'y;

print ''bols = '' bols;
```

```
d = dummy(data[.,1],1|2|3);
x = data[.,2] ~d;

theta = invpd(x'x)*x'y;

beta = theta[1];
gamma_ = trimr(theta,1,0);

print ''beta = '' beta;
print ''gamma = '' gamma_;

u = y - x*theta;
u = d .* u;

m = sumc(u) ./ sumc(d);
sigma = sqrt( sumc(u^2)./sumc(d) - m^2);

print m~sigma;
```

– Nous remarquons que les écart-types empiriques des individus sont très différents. Cela suggère d'utiliser un modèle plus général :

$$y_{i,t} = \beta x_{i,t} + \gamma_i + v_{i,t} \tag{64}$$

avec $v_{i,t} \sim \mathcal{N}(0, \Omega)$. Pour estimer les paramètres β et γ_i $(i = 1, \ldots, n)$ de façon plus efficace, nous pouvons utiliser un estimateur à deux étapes (`GLS`). Nous pouvons aussi estimer ce modèle par maximum de vraisemblance ou par d'autres méthodes (2SLS ou 3SLS).

27.2 La bibliothèque TSCS

La bibliothèque **TSCS** permet d'estimer les modèles de panel décrit dans le livre de Judge, Hill, Griffiths, Lütkepohl et Lee [1988].

```
new;
library tscs;

cls;

_tsmeth = 1;
{ b_dv,vc_dv,m_dv,b_ec,vc_ec,m_ec } = tscs(''jdata'',''x3'',''x2'',''x1'');
```

```
================================================================================
 TSCS Version 3.1.2                                        3/11/2002  10:46 pm
================================================================================
                             Data Set:  jdata
--------------------------------------------------------------------------------


---------------------- OLS DUMMY VARIABLE RESULTS  ----------------------

    -------------------------
    Dependent variable:  X3
    -------------------------

Observations          :  40
Number of Groups      :  4
Degrees of freedom    :  35
Residual SS           :  352.652
Std error of est      :  3.174
Total SS (corrected)  :  3474.836
F                     =  309.870      with 1,35 degrees of freedom
P-value               =  0.000

Var          Coef.       Std. Coef.    Std. Error    t-Stat        P-Value
--------------------------------------------------------------------------------
X2            0.802940     0.947899     0.045613    17.603131       0.000
```

```
Group Number        Dummy Variable    Standard Error

      1             1.247312             1.946579
      2            -5.100511             2.218553
      3             0.718028             1.749809
      4           -10.099849             2.407791


   F-statistic for equality of dummy variables :
      F(3, 35) = 22.2912       P-value:  0.0000


------------------- OLS ESTIMATE OF CONSTRAINED MODEL ----------------------

    -------------------------
     Dependent variable:  X3
    -------------------------

Observations          :  40
Number of Groups      :  4
Degrees of freedom    :  38
R-squared             :  0.715
Rbar-squared          :  0.707
Residual SS           :  1026.456
Std error of est      :  5.197
Total SS (corrected)  :  3598.176
F                     =  95.207       with 2,38 degrees of freedom
P-value               =  0.000

Var          Coef.       Std. Coef.    Std. Error    t-Stat        P-Value
------------------------------------------------------------------------------
CONSTANT      3.516057          ---     2.706407     1.299160       0.202
X2            0.631615     0.845416     0.064732     9.757384       0.000


------------------------------------------------------------------------------
 FULL, RESTRICTED, AND PARTIAL R-SQUARED TERMS--DUMMY VARIABLES ARE CONSTRAINED
-------------------------------------
 TABLE OF R-SQUARED TERMS
-------------------------------------
 R-squared--full model:        0.902
 R-squared--constrained model: 0.715
 Partial R-squared:            0.656
-------------------------------------


------------------------------------------------------------------------------
 FULL, RESTRICTED, AND PARTIAL R-SQUARED TERMS--X VARIABLES ARE CONSTRAINED
-------------------------------------
 TABLE OF R-SQUARED TERMS
-------------------------------------
 R-squared--full model:        0.902
 R-squared--constrained model: 0.034
 Partial R-squared:            0.899
-------------------------------------

---------------------- GLS ERROR COMPONENTS RESULTS  ----------------------

    -------------------------
     Dependent variable:  X3
    -------------------------

Observations          :  40
Number of Groups      :  4
Degrees of freedom    :  38
Residual SS           :  513.320
Std error of est      :  3.675
Total SS (corrected)  :  3499.107
F                     =  350.102      with 2,38 degrees of freedom
P-value               =  0.000
Std. errors of error terms:
 Individual constant terms: 2.028
 White noise error        : 3.174

Var          Coef.       Std. Coef.    Std. Error    t-Stat        P-Value
------------------------------------------------------------------------------
CONSTANT     -1.630423          ---     2.423184    -0.672843       0.505
X2            0.760808     0.923742     0.051174    14.867137       0.000


Group Number        Random Components

      1             3.548822
      2            -1.319397
      3             2.949679
      4            -5.179104


          Lagrange Multiplier Test for Error Components Model


    Null hypothesis:  Individual error components do not exist.

    Chi-squared statistic (1): 38.8192
    P-value: 0.0000


          Hausman (1978) Chi-Squared Specification Test
```

```
Null hypothesis:  Error components model is the correct specification.

Chi-squared statistic (1) = -3.2983
P-value = ---
```

27.3 Le programme DPD98 de Manuel Arellano et Stephen Bond

La modèle général de **DPD98** est le suivant :

$$y_{i,t} = \sum_{k=1}^{p} \alpha_k y_{i,t-k} + \beta (L)^{\top} x_{i,t} + \lambda_t + \eta_i + v_{i,t}$$

avec $t = 1, \ldots, T_i$ et $i = 1, \ldots, n$. Nous remarquons donc que **DPD98** considère tout aussi bien des modèles *balanced* ($T_i = T$) et *unbalanced* ($T_i \neq T_j$). Les effets spécifiques sont captés par le biais des coefficients η_i alors que les effets temporels sont modélisés avec les paramètres λ_t.

La première utilisation de **DPD98** peut paraître difficile. En fait, la principale difficulté est de comprendre le stockage des données. Pour cela, nous considérons l'exemple de Arellano et Bond. **DPD98** nécessite deux bases de données. La première base contient les données.

```
y = loadd(''xdata'');
print $getname(''xdata'')';
call printfmt(y[1:50,.],1);
```

IND	YEAR	EMP	WAGE	CAP	INDOUTPT
7	1977	5.0409999	13.1516	0.58939999	95.707199
7	1978	5.5999999	12.3018	0.6318	97.356903
7	1979	5.0149999	12.8395	0.6771	99.608299
7	1980	4.7150002	13.8039	0.6171	100.5501
7	1981	4.0929999	14.2897	0.50760001	99.558098
7	1982	3.1659999	14.8681	0.42289999	98.615097
7	1983	2.9360001	13.7784	0.39199999	100.0301
7	1977	71.319	14.7909	16.9363	95.707199
7	1978	70.642998	14.1036	17.242201	97.356903
7	1979	70.917999	14.9534	17.5413	99.608299
7	1980	72.030998	15.491	17.6574	100.5501
7	1981	73.689003	16.196899	16.713301	99.558098
7	1982	72.418999	16.131399	16.246901	98.615097
7	1983	68.517998	16.305099	17.3696	100.0301
7	1977	19.156	22.691999	7.0974998	95.707199
7	1978	19.440001	20.6938	6.9468999	97.356903
7	1979	19.9	21.2048	6.8565001	99.608299
7	1980	20.24	22.197001	6.6546998	100.5501
7	1981	19.57	24.871401	6.2136002	99.558098
7	1982	18.125	24.8447	5.7146001	98.615097
7	1983	16.85	28.9077	7.3431001	100.0301
8	1977	26.16	14.8283	8.4902	118.2223
8	1978	26.74	14.8379	8.7419996	120.1551
8	1979	27.280001	14.8756	9.1869001	118.8319
8	1980	27.83	15.2332	9.4035997	111.9164
8	1981	27.169001	17.252899	8.8938999	97.554001
8	1982	24.504	19.314199	8.3905001	92.198196
8	1983	22.562	20.005501	7.4351001	92.404099
7	1976	86.677002	20.632299	22.3804	94.899101
7	1977	87.099998	18.2782	22.2703	96.503799
7	1978	87	18.1369	25.167	98.816299
7	1979	90.400002	18.1896	25.3515	100.4835
7	1980	89.199997	19.289301	28.239401	100.1223
7	1981	82.699997	19.8242	25.7959	98.527
7	1982	73.699997	19.3873	20.368099	99.508301
3	1976	0.74800003	23.1889	0.16940001	102.7724
3	1977	0.76599997	20.539301	0.1618	107.027
3	1978	0.76200002	20.9387	0.1499	108.6788
3	1979	0.72899997	21.7626	0.14579999	111.119
3	1980	0.73100001	23.470699	0.1468	101.9265
3	1981	0.77899998	24.1252	0.14120001	99.497101
3	1982	0.78200001	25.206499	0.1261	99.310303
3	1976	1.6	30.851299	0.99339998	104.7664
3	1977	1.65	28.555	0.93300003	107.4791
3	1978	1.6799999	29.860201	0.96499997	108.9188
3	1979	1.6799999	30.589399	0.96289998	111.5591
3	1980	1.66	27.4886	0.9224	100
3	1981	1.5599999	31.486799	0.99919999	99.3965
3	1982	1.54	32.0728	0.88020003	99.293098
3	1976	9.1450005	30.908899	4.5486002	104.7664
3	1977	10.855	24.467899	4.6194	107.4791
3	1978	10.749	28.055799	5.0373001	108.9188
3	1979	10.959	28.744499	4.8748002	111.5591
3	1980	10.698	28.485399	3.2221	100
3	1981	8.1739998	31.7356	2.8343	99.3965
3	1982	4.7010002	29.896299	1.7506	99.293098
3	1976	2.006	26.0825	0.38240001	102.7724
3	1977	2.6559999	21.9545	0.4551	107.027
3	1978	2.7119999	22.809999	0.45289999	108.6788
3	1979	2.744	23.372	0.4524	111.119
3	1980	2.757	23.0737	0.46759999	101.9265

```
3          1981          2.849       24.304399       0.48100001       99.497101
3          1982       2.7520001       26.898399       0.47530001       99.310303
7          1976          3.823         23.3533           1.1293         95.3657
7          1977       3.9349999         21.3251           1.0143       96.731499
7          1978       3.5280001         20.9319           1.1169       99.233299
7          1979           3.52         20.4536           1.1626       100.7335
7          1980          3.467         23.4946           1.0827            100
7          1981       3.0899999       25.955299        1.0635999       98.232399
7          1982       3.2620001         26.0361           1.0958       99.763496
3          1976          1.359       23.061399       0.40990001       104.7664
3          1977          1.346       20.227501       0.57410002       107.4791
3          1978       1.3380001       22.143499       0.55629998       108.9188
3          1979          1.353       22.549601       0.51450002       111.5591
3          1980       1.2640001         22.5875       0.58950001            100
3          1981          1.158         25.9585       0.48469999        99.3965
3          1982          1.175         24.0345           0.4359       99.293098
3          1976           1.78       34.299099       0.25690001       104.7664
3          1977          1.643         30.3979       0.20720001       107.4791
3          1978           1.39       31.143101           0.1999       108.9188
3          1979           1.35       33.310501           0.2007       111.5591
3          1980          1.082       30.214701           0.1473            100
3          1981     0.77999997         29.7414           0.1158        99.3965
3          1982     0.68000001       31.123501           0.1036       99.293098
1          1976      2.3529999       22.383301       0.57849997       125.8064
1          1977      2.3859999       23.060699       0.56129998       127.9649
1          1978          2.375       23.493099           0.5273       125.4294
1          1979          2.415         26.0042       0.51800001       125.0259
1          1980          2.266       24.167101       0.46079999       106.3044
1          1981          1.777       26.057301            0.375       98.072899
1          1982      1.5369999         28.7064       0.31470001       96.867401
1          1978          3.165       20.911301       0.39019999       127.3866
1          1979          3.096         21.8887       0.43110001       124.6424
1          1980      2.9909999          23.701           0.4228       118.9132
1          1981      2.6919999         21.3925       0.38100001       99.357597
1          1982          2.175       25.523701       0.30509999       97.242798
1          1983          1.801         25.7211           0.2041       97.194397
1          1984          1.428       25.645201       0.16850001       99.986198
9          1977      7.2199998         22.5898           1.8498       105.3355
9          1978      7.3210001       22.054899           1.9215       108.1833
```

La seconde base contient deux colonnes.

```
____________________ c:\gauss\gro\ritme\dpd98/auxdata.dat _____________________

                T         MUL
 #1          7.00        103.
 #2          8.00        23.0
 #3          9.00        14.0
```

```
--------------------------------------------------------------------------------

PgDn  PgUp  Home  End                Ctrl-   Ctrl-   ?  Esc  [Esc]
```

D'après Arellano et Bond, "*this auxiliary data set must contain two columns : elements of the first column contain the number of time-series observations per individual unit in the appropriate section of the main data file ; and elements of the second column contain the number of individual units which have this number of time-series observations*". Si nous analysons la variable `YEAR`, nous remarquons qu'une séquence temporelle de

7 dates consécutives est répétée 103 fois, puis une séquence temporelle de 8 dates consécutives est répétée 23 fois, enfin une séquence temporelle de 9 dates consécutives est répétée 14 fois. Donc la base de données est

7	103
8	23
9	14

Nous vérifions que $7 \times 103 + 8 \times 23 + 9 \times 14 = 1031$, qui est le nombre d'observations de la base de données `xdata`. Nous pouvons aussi déterminer le nombre d'individus qui est égal à $103 + 23 + 14 = 140$.

- La construction de la base de données *auxdata* peut s'avérer très difficile. J'ai programmé la procédure `DPD98_AuxData` pour construire cette base automatiquement.
- Remarquez que 103 observations d'une séquence temporelle de 7 dates consécutives ne veut pas forcément dire que ces séquences sont les mêmes. Là aussi, j'ai programmé la procédure `DPD98_TimeSeq` qui permet de repérer l'ensemble des séquences identiques. Par exemple, les données de Arellano et Bond sont structurées de la façon suivante :
  1. Les 4 premières observations correspondent à la séquence temporelle 1977 :1983.
  2. Ensuite, il y a 9 observations pour lesquelles la séquence temporelle est 1976 :1982.
  3. Nous trouvons ensuite une observation dont la séquence temporelle est 1977 :1983.
  4. etc.

```
new;

cls;

y = loadd(''xdata'');

table = DPD98_AuxData(y[.,1],y[.,2]);
{nSeq,TimeSeq} = DPD98_TimeSeq(y[.,1],y[.,2]);

print table;

fmt[1,3] = { ''*.*lg '' 6 4 };
fmt = ''-*.*lf ''~4~0 |
      ''-*.*lf ''~10~0 |
      fmt .* ones(cols(TimeSeq),1);
call printfm(nSeq~cumsumc(nSeq)~TimeSeq,1,fmt);


proc DPD98_AuxData(ind,year);
  local r,nobs,comp,i,table;

  r = rows(year);
  nobs = {};
  comp = 1;
  i = 2;
  do until i > r;

    if (year[i] == year[i-1] + 1) and (ind[i] == ind[i-1]);
      comp = comp + 1;
      if i == r;
        nobs = nobs | comp;
      endif;
```

```
    else;
      nobs = nobs | comp;
      if i == r;
        nobs = nobs | 1;
      else;
        comp = 1;
      endif;
    endif;

    i = i + 1;
  endo;

  r = rows(nobs);

  table = {};
  comp = 1;
  i = 2;
  do until i > r;

    if nobs[i] == nobs[i-1];
      comp = comp + 1;
      if i == r;
        table = table | (nobs[i]~comp);
      endif;
    else;
      table = table | (nobs[i-1]~comp);
      comp = 1;
    endif;

    i = i + 1;
  endo;

  retp(table);
endp;

proc (2) = DPD98_TimeSeq(ind,year);
  local table,TimeSeq,nSeq,n,i,year_,comp,j;

  table = DPD98_AuxData(ind,y[.,2]);

  TimeSeq = unique(year,1)';
  nSeq = {};

  n = 0|cumsumc(prodc(table'));

  i = 1;
  do until i > rows(table);

    year_ = reshape(year[1+n[i]:n[i+1]],table[i,2],table[i,1]);
    comp = 1;
    j = 2;
    do until j > table[i,2];
```

```
      if year_[j,.] == year_[j-1,.];
        comp = comp + 1;
        if j == table[i,2];
          nSeq = nSeq | comp;
          TimeSeq = _dpd98_add(TimeSeq,year_[j-1,.]);
        endif;
      else;
        nSeq = nSeq | comp;
        TimeSeq = _dpd98_add(TimeSeq,year_[j-1,.]);
        comp = 1;
        if j == table[i,2];
          nSeq = nSeq | comp;
          TimeSeq = _dpd98_add(TimeSeq,year_[j,.]);
        endif;
      endif;

      j = j + 1;
    endo;

    i = i + 1;
  endo;

  TimeSeq = trimr(TimeSeq,1,0);

  retp(nSeq,TimeSeq);
endp;

proc (1) = _dpd98_add(x,y);
  local indx;

  x = x | miss(zeros(1,cols(x)),0);

  if cols(y) == cols(x);
    x[rows(x),.] = y;
  else;
    indx = indnv(y[1],x[1,.]');
    x[rows(x),indx:indx+cols(y)-1] = y;
  endif;

  retp(x);
endp;
```

```
       7.0000000        103.00000
       8.0000000        23.000000
       9.0000000        14.000000


4    4            ---   1977   1978   1979   1980   1981   1982   1983    ---
9    13          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    14           ---    ---   1978   1979   1980   1981   1982   1983   1984
1    15           ---   1977   1978   1979   1980   1981   1982   1983    ---
```

```
1    16          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    18           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    19          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    21           ---   1977   1978   1979   1980   1981   1982   1983    ---
5    26          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    27           ---    ---   1978   1979   1980   1981   1982   1983   1984
2    29           ---   1977   1978   1979   1980   1981   1982   1983    ---
2    31          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    32           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    33          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    35           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    36          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    37           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    38          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    39           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    40          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    41           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    42          1976   1977   1978   1979   1980   1981   1982    ---    ---
3    45           ---   1977   1978   1979   1980   1981   1982   1983    ---
7    52          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    54           ---   1977   1978   1979   1980   1981   1982   1983    ---
2    56          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    57           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    58          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    59           ---   1977   1978   1979   1980   1981   1982   1983    ---
1    60          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    62           ---   1977   1978   1979   1980   1981   1982   1983    ---
3    65          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    66           ---   1977   1978   1979   1980   1981   1982   1983    ---
2    68          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    70           ---   1977   1978   1979   1980   1981   1982   1983    ---
2    72          1976   1977   1978   1979   1980   1981   1982    ---    ---
2    74           ---   1977   1978   1979   1980   1981   1982   1983    ---
2    76          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    77           ---   1977   1978   1979   1980   1981   1982   1983    ---
5    82          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    83           ---   1977   1978   1979   1980   1981   1982   1983    ---
9    92          1976   1977   1978   1979   1980   1981   1982    ---    ---
1    93           ---   1977   1978   1979   1980   1981   1982   1983    ---
4    97          1976   1977   1978   1979   1980   1981   1982    ---    ---
5    102          ---   1977   1978   1979   1980   1981   1982   1983    ---
1    103         1976   1977   1978   1979   1980   1981   1982    ---    ---
9    112          ---   1977   1978   1979   1980   1981   1982   1983   1984
1    113         1976   1977   1978   1979   1980   1981   1982   1983    ---
4    117          ---   1977   1978   1979   1980   1981   1982   1983   1984
1    118         1976   1977   1978   1979   1980   1981   1982   1983    ---
3    121          ---   1977   1978   1979   1980   1981   1982   1983   1984
2    123         1976   1977   1978   1979   1980   1981   1982   1983    ---
3    126          ---   1977   1978   1979   1980   1981   1982   1983   1984
14   140         1976   1977   1978   1979   1980   1981   1982   1983   1984
```

– Lorsque les données ont été bien construites, il suffit ensuite de se laisser guider par le programme *dpd98.run*.

```
new ,80000;
#include dpd98.fns;

@-------------------------- SPECIFY GAUSS OPTIONS --------------------------@

@ Set sys=1 to exit to DOS @                                  sys = 0;
@ Set bat=1 to use in batch mode @                            bat = 1;

@ Set pseud=1 to use pseudo-inverses @                        pseud = 0;
@ in computing two-step weighting    @
@ matrix and Wald tests              @

@------------------------ SET UP FOR BATCH OPERATION ------------------------@

@ The following set up will only be used when bat is set to 1 @

@ Form of model @                                             imod = 1;
@ Choice of constants @                                       icon = 1;
@ Robust covariance & two-step option @                       irob = 1;
@ Descriptive statistics @                                    ides = 0;
@ Print covariance matrices @                                 icov = 0;
@ Save results to GAUSS matrices @                            isav = 0;

@------------------------------ SPECIFY DATA SET ------------------------------@

@ Specify main GAUSS data set @                               open f1 = xdata;
@ Specify auxiliary data set @                                open f2 = auxdata;

@ Start reading at line (f2) @                                startf2 = 1;
@ Stop reading at line (f2) @                                 stopf2 = rowsf(f2);

@------------------------------ DATA INFORMATION ------------------------------@

@ Number of companies to @
@ process in each read @                                      ncomp = 20;

@ Data column for year @                                      yearcol = 2;
@ First year of data @                                        year1 = 1976;
@ Number of years in data set @                               nyears = 9;

@ Data column for industry @                                  indcol = 1;
@ Number of industry classes @                                indmax = 9;

@----------------------------- MODEL INFORMATION -----------------------------@

@ Longest lag to be constructed @                             lag = 2;

@------------ DATA TRANSFORMATIONS AND MODEL SELECTION SUBROUTINE ------------@
goto below;
model:

@ The main data set is read in to a matrix called data @
```

```
@ Operations using columns of data may be performed here @

@ Function back(c,l) returns the l'th lag of the series in column c of data @

@ Function timdum(y1~y2~...~yn) returns the 0/1 dummy with value 1 for @
@ years y1 or y2 or ... or yn and 0 otherwise. Format for years is 19xx. @

@ Function inddum(i1~i2~...~in) returns the 0/1 dummy with value 1 for @
@ industries i1 or i2 or ... or in, and 0 otherwise. @

@ Variable names temp* are reserved for storing intermediate transformations @

data=ln(data);
@temp=data[.,5]-back(5,1);
data=data~temp;@

@ SELECT DEPENDENT VARIABLE @

y=dif(3,0);
namey=''Dn'';

@ SELECT REGRESSORS @

x=dif(3,1)~dif(3,2)~dif(4,0)~dif(4,1)~dif(5,0)~dif(6,0)~dif(6,1);
namex=''Dn(-1)''~''Dn(-2)''~''Dw''~''Dw(-1)''~''Dk''~''Dys''~''Dys(-1)'';

@ SELECT INSTRUMENTS @

z=gmm(3,2,99)~x[.,3:7];
namez=''n(2,all)''~namex[.,3:7];

return;
below:
@---------------- USER-DEFINED WALD TEST OF JOINT SIGNIFICANCE ----------------@

@ Set waldtest=1 for this option @      waldtest=1;
@ Select columns of x to be tested @    testcols=3~4;

@-------------------------- SPECIFY FILE FOR OUTPUT --------------------------@

output file = dpd98.out on;

@ Note that ''on'' will append and ''reset'' will overwrite @

@*****************************************************************************@
@                                                                             @
@              AFTER SELECTING THE MODEL THIS PROGRAM CAN BE                  @
@              RUN DIRECTLY FROM THE GAUSS EDITOR BY HITTING F2               @
@                                                                             @
@*****************************************************************************@

#include dpd98.prg; end;
```

```
@------------------------------ END OF PROGRAM ------------------------------@
```

```
  DDD     PPPP    DDD       RRRR   EEEEE   SSSS  U   U  L      TTTTT   SSSS
  D  D    P   P   D  D      R   R  E      S      U   U  L        T    S
  D   D   PPPP    D   D     RRRR   EEEE    SSS   U   U  L        T     SSS
  D  D    P       D  D      R  R   E          S  U   U  L        T        S
  DDD  .  P    .  DDD  .    R   R  EEEEE  SSSS    UUU   LLLLL    T    SSSS
-------------------------------------------------------------------------------
```

IV, FIRST DIFFERENCES

```
Number of firms:    140      Sample period is 1979 to 1984
Observations:       611      Degrees of freedom:       598

Dependent variable is:      Dn

Instruments used are:

   CONST n(2,all)       Dw   Dw(-1)       Dk      Dys  Dys(-1) TIM DUMS
```

```
-------------------------------------------------------------------------------
```

ONE-STEP ESTIMATES

```
RSS =      8.219380           TSS =     12.599978
Estimated sigma-squared (levels)  =      0.006872

Wald test of joint significance:     352.585004   df =   7     p = 0.000
Wald test - jt sig of time dums:      11.254987   df =   6     p = 0.081

     Wald test selected by user:      91.830846   df =   2     p = 0.000
     Testing:      Dw   Dw(-1)

                   Sargan test:      73.858107   df =  25     p = 0.000

Variable      Coefficient      Std. Error     T-Statistic        P-Value
-------------------------------------------------------------------------
   CONST        0.005427        0.012814        0.423552        0.671892
  Dn(-1)        0.534614        0.127418        4.195740        0.000027
  Dn(-2)       -0.075069        0.043441       -1.728079        0.083974
      Dw       -0.591573        0.061907       -9.555793        0.000000
  Dw(-1)        0.291510        0.095558        3.050603        0.002284
      Dk        0.358502        0.034868       10.281733        0.000000
```

```
     Dys        0.597199        0.127326        4.690304        0.000003
 Dys(-1)       -0.611705        0.167947       -3.642250        0.000270
     D80        0.005608        0.020075        0.279335        0.779988
     D81       -0.038305        0.017635       -2.172098        0.029848
     D82       -0.027785        0.018522       -1.500107        0.133587
     D83       -0.006850        0.019021       -0.360148        0.718737
     D84        0.006314        0.023754        0.265799        0.790394

Test for  first-order serial correlation:  -3.409  [  140 ]    p = 0.001
Test for second-order serial correlation:  -0.369  [  140 ]    p = 0.712

NOTE: Standard errors and test statistics not robust to heteroskedasticity

--------------------------------------------------------------------------------


                ONE-STEP ESTIMATES WITH ROBUST TEST STATISTICS


Wald test of joint significance:     219.623310   df =   7     p = 0.000
Wald test - jt sig of time dums:      11.450408   df =   6     p = 0.075

     Wald test selected by user:      12.486527   df =   2     p = 0.002
     Testing:      Dw   Dw(-1)

Variable      Coefficient      Std. Error     T-Statistic        P-Value
--------------------------------------------------------------------------
   CONST        0.005427        0.009714        0.558696        0.576369
  Dn(-1)        0.534614        0.166449        3.211871        0.001319
  Dn(-2)       -0.075069        0.067979       -1.104302        0.269462
      Dw       -0.591573        0.167884       -3.523705        0.000426
  Dw(-1)        0.291510        0.141058        2.066597        0.038772
      Dk        0.358502        0.053828        6.660098        0.000000
     Dys        0.597199        0.171933        3.473441        0.000514
 Dys(-1)       -0.611705        0.211796       -2.888179        0.003875
     D80        0.005608        0.015378        0.364660        0.715365
     D81       -0.038305        0.017445       -2.195731        0.028111
     D82       -0.027785        0.017908       -1.551541        0.120772
     D83       -0.006850        0.022055       -0.310593        0.756110
     D84        0.006314        0.019713        0.320284        0.748753

Test for  first-order serial correlation:  -2.493  [  140 ]    p = 0.013
Test for second-order serial correlation:  -0.359  [  140 ]    p = 0.719


          Estimated serial correlation matrix - differenced residuals

 1.000
-0.478  1.000
 0.047 -0.141  1.000
 0.136 -0.170 -0.439  1.000
```

```
 0.259 -0.124  0.015 -0.454  1.000
 0.158 -0.190  0.050  0.412 -0.559  1.000

          Number of observations available to sample covariances

    80
    80    138
    80    138    140
    80    138    140    140
    18     76     78     78     78
    14     33     35     35     35     35
```

TWO-STEP ESTIMATES

```
Wald test of joint significance:     371.987810   df =   7     p = 0.000
Wald test - jt sig of time dums:      26.904500   df =   6     p = 0.000

     Wald test selected by user:     111.105724   df =   2     p = 0.000
     Testing:      Dw   Dw(-1)

                   Sargan test:      30.112471   df =  25     p = 0.220
```

Variable	Coefficient	Std. Error	T-Statistic	P-Value
CONST	0.010509	0.007251	1.449224	0.147275
Dn(-1)	0.474151	0.085303	5.558424	0.000000
Dn(-2)	-0.052968	0.027284	-1.941316	0.052220
Dw	-0.513205	0.049345	-10.400258	0.000000
Dw(-1)	0.224640	0.080063	2.805799	0.005019
Dk	0.292723	0.039463	7.417736	0.000000
Dys	0.609775	0.108524	5.618817	0.000000
Dys(-1)	-0.446373	0.124815	-3.576284	0.000349
D80	0.003633	0.012734	0.285327	0.775394
D81	-0.050962	0.013710	-3.717118	0.000202
D82	-0.032149	0.013986	-2.298604	0.021527
D83	-0.012356	0.012842	-0.962161	0.335969
D84	-0.020730	0.013679	-1.515434	0.129662

```
Test for  first-order serial correlation:  -2.826  [  140 ]    p = 0.005
Test for second-order serial correlation:  -0.327  [  140 ]    p = 0.744


  Execution time is      4.530  seconds
```

27.4 Un programme de Curt Wells

La procédure

```
/*
**> twoway
**
**  Procedure to remove the individual and time mean, multiplied by theta, from a balanced panel.
**  Use for two-way panels.
**
**  Format: {TWOWAY,meanW,meanT,meanA} = twoway(x,N,T,theta);
**
**  Input:     x    N*T by k matrix of data to be operated on
**                  note that the data must be arranged as
**                       x1(11)  x2(11) ... xk(11)
**                       x1(12)  x2(12) ... xk(12)
**                         .       .     .    .
**                         .       .     .    .
**                         .       .     .    .
**                       x1(1T)  x2(1T) ... xk(1T)
**                       x1(21)  x2(21) ... xk(11)
**                       x1(22)  x2(22) ... xk(22)
**                         .       .     .    .
**                         .       .     .    .
**                         .       .     .    .
**                       x1(2T)  x2(2T) ... xk(2T)
**                         .       .     .    .
**                         .       .     .    .
**                         .       .     .    .
**                       x1(N1)  x2(N1) ... xk(N1)
**                       x1(N2)  x2(N2) ... xk(N2)
**                         .       .     .    .
**                         .       .     .    .
**                         .       .     .    .
**                       x1(NT)  x2(NT) ... xk(NT)
**                  Note that the number of variables (k) will
**                  be read from the matrix.
**
**             N    Scalar, The numer of individuals
**             T    Scalar, The number of time periods
**         Theta    3 by 1 vector. The means will be multiplied by Theta.
**                    For OLS, set Theta=ones(3,1)
**                    For GLS, set
**                         Theta[1]=1-sqrt(S(v)^2/lambda_2)
**                         Theta[2]=1-sqrt(S(v)^2/lambda_3)
**                         Theta[3]=Theta[1]+Theta[2]+sqrt(S(v)^2/lambda_4)-1
**
**   Output:   TWOWAY A matrix where typical element is
**                   x(it) - theta[1]*x(i.) - theta[2]*x(.t) + theta[3]*x(..)
**
**             meanW n by 1 vector of means of individuals over time, x(i.)
**
**             meanT T by 1 vector of means of time over individuals, x(.t)
**
**             meanA cols(x) by 1 vector of the overall mean of the variable, x(..)
*/
```

```
proc (4) = twoway(x,n,t,theta);
  local D,TWOWAY,meanW,meanT,meanA,k,i,mw,mt,ma;

  k = cols(x);                              @ the number of variables @
  TWOWAY = zeros(n*t,k);                    @ initialize; devs from means @
  meanW = zeros(n,k);                       @ initialize: time means @
  meanT = zeros(t,k);                       @ initialize: indiv means @
  meanA = zeros(k,1);                       @ initialize: overall mean @
  i = 1;
  do until i > k;                           @ loop thru the data matrix @
    D = reshape(x[.,i],n,t);                @ a N by T matrix for each var @
    mw = meanc(D');                         @ means indiv over time @
    mt = meanc(D);                          @ means time over indiv @
    ma = meanc(mt);                         @ overall mean of the variable @
    meanW[.,i] = mw;                        @ save means indiv over time @
    meanT[.,i] = mt;                        @ save means time over indiv @
    meanA[i,1] = ma;                        @ save the overall mean @
    D = (D' - Theta[2]*mt)'                 @ remove mean time over indiv @
         - Theta[1]*mw                      @ remove mean indiv over time @
         + Theta[3]*ma;                     @ put back total mean @
    TWOWAY[.,i] = reshape(D,n*t,1);         @ save deviations @
    i = i + 1;                              @ next column in data matrix @
  endo;
  retp(TWOWAY,meanW,meanT,meanA);
endp;
```

Le programme

```
/*
** This will do the between regression in table 2.1
** AND, as an added benefit,  get the book's results
**
** It uses twoway (could have used between) to get group means
** and then uses formulas i Greene on page 619
**
** The program then uses the P transformation and finally the
** transformation suggested by Baltagi on p 16
*/

new;
#include twoway.src;

msk = zeros(1,1)~ones(1,4);
let fmt[5,3] = ''*.*s '' 8 8 ''*.*lg'' 12 4 ''*.*lg'' 12 4 ''*.*lg'' 12 4 ''*.*le'' 12 2;
msk1 = zeros(1,1)~ones(1,1);
let fmt1[2,3] = ''*.*s '' 8 8 ''*.*lg'' 12 4;

N = 10;    @ number of individuals @
T = 20;    @ length of time period @
NT = N*T;
K = 2;     @ number of independents @
```

```
load Data[NT,K+1] = grunfeld.txt;


/*
**   NOTE THAT THE DEPENDENT VARIABLE IS THE FIRST COLUMN IN Data
**
**   This is not required by the subroutine WITHIN but it makes book keeping
**   easier. Thus the independents are columns 2 to K+1
**
**   If the data is not organized in this way, you could always rearrange it
**   after it is entered. Thus, if column J is the dependent and cols 1:J-1, and
**   J+1:K+1 are the independents, simple code
**
**   y = Data[.,J];
**   x = Data[.,1:J-1 J+1:K+1];
**
**   and call WITHIN and the rest will work.
*/

y0 = Data[.,1];
x0 = Data[.,2:k+1];

names = ''   Value''|''   Stock'';
namesc = ''Constant''|names;

{z,m1,m2,m3} = twoway(y0~x0,N,T,(1|0|0));  @ we want group means in m1 @

x = m1[.,2:3]-m3[2:3]';                    @ our x matrix @
y = m1[.,1]-m3[1];                         @ our y matrix @


/*
**   The P-transformation in Baltagi actually removes the overall mean
**   from each of the group means and stacks the resulting matrix t times
**
**   The way I do it is easier, but the result is the same
*/

sxx = t*x'x;          @ Sum of squares: Green, middle of page 619 @
sxy = t*x'y;          @ Sum of squares: Green, middle of page 619 @

b = solpd(sxy,sxx);             @ 14-13 in Greene @
A = m3[1]-m3[2:3]'*b;           @ the constant @
u = m1[.,1]-A-m1[.,2:3]*b;      @ residuals @
s2 = T*u'u/(N-k-1);             @ residual variance @
vcv = s2*invpd(Sxx);            @ vcv matrix @
se = sqrt(diag(vcv));           @ standard errors @

tF = b./se;                           /* t-stats */
sgF = 2*cdftc(abs(tF),(N-3));         /* Significance levels */

va = s2/NT + m3[2:3]'*vcv*m3[2:3];    /* var(ALFA) */
```

```
sea = sqrt(va);                         /* standard error of ALFA */
t_a = A/sea;                            /* t-stat for ALFA */
sga = 2*cdftc(abs(t_a),(N-3));          /* signif level for ALFA */

r2 = 1-T*u'u/(T*y'y);                    /* R2 */


/* print the final estimates */

format 16,6; print; print;
f0 = ''The Between regression (using transposes):'';
$f0;
''************************************************************'';
''Degrees of freedom           : '';; (N-3);
''Residual sum of squares      : '';; T*u'u;
''Residual variance            : '';; s2;
''R-squared                    : '';; r2;
''************************************************************'';
''Variable         coef     std.err      t-stat      Signif'';
f0 = printfm(namesc~(A|b)~(sea|se)~(t_a|tF)~(sga|sgF),msk,fmt);
''************************************************************'';

/* Now do it by brute force and compare */

P = eye(N) .*. (ones(T,T)/T);       /* The P transformation */
x = ones(rows(P),1)~P*x0;
y = P*y0;

b = y/x;
e = y - x*b;
s2 = e'e/(N-K-1);
ss = s2;
se = sqrt(diag(s2*invpd(x'x)));
tF = b./se;                         /* t-stats */
sgF = 2*cdftc(abs(tF),(N-3));       /* Significance levels */

r2 = 1-e'e/(y'y-NT*meanc(y)^2);     /* R2 */

/* print the final estimates */

format 16,6;
print;print;
f0 = ''The between regression (using brute force):'';
$f0;
''************************************************************'';
''Degrees of freedom           : '';; (N-3);
''Residual sum of squares      : '';; e'e;
''Residual variance            : '';; s2;
''R-squared                    : '';; r2;
''************************************************************'';
''Variable         coef     std.err      t-stat      Signif'';
f0 = printfm(namesc~(b)~(se)~(tF)~(sgF),msk,fmt);
```

```
''************************************************************'';


/* Finally we can do it as suggested by Baltagi in (2.26) */

y = m1[.,1]*sqrt(T);
x = (ones(N,1)~m1[.,2:3])*sqrt(T);

b = y/x;
e = y - x*b;
s2 = e'e/(N-K-1);
se = sqrt(diag(s2*invpd(x'x)));
tF = b./se;                              /* t-stats */
sgF = 2*cdftc(abs(tF),(N-K-1));          /* Significance levels */

r2 = 1-e'e/(y'y - N*meanc(y)^2);         /* R2 */

format 16,6;
print;print;
f0 = ''The between regression (using Baltagi's 2.26):'';
$f0;
''************************************************************'';
''Degrees of freedom           : '';; (N-3);
''Residual sum of squares      : '';; e'e;
''Residual variance            : '';; s2;
''R-squared                    : '';; r2;
''************************************************************'';
''Variable         coef     std.err      t-stat      Signif'';
f0 = printfm(namesc~(b)~(se)~(tF)~(sgF),msk,fmt);
''************************************************************'';
```

28 Manipulation des bibliothèques disponibles sur Internet

A ce stade de la formation, vous avez tous les éléments pour utiliser des bibliothèques disponibles sur Internet. La grande différence entre celles-ci et les bibliothèques commerciales est qu'elles sont souvent très mal faites. Elles comportent souvent des chemins de répertoire, ils manquent parfois des déclarations de variables, et dans quelques cas il y a des bugs. Il est donc nécessaire de mettre les mains dans le code avant de pouvoir les utiliser.

– Commençons par des bibliothèques qui ne devraient pas poser de difficultés. Ce sont les bibliothèques `MVT` et `QREG` disponibles sur le site web du GRO :

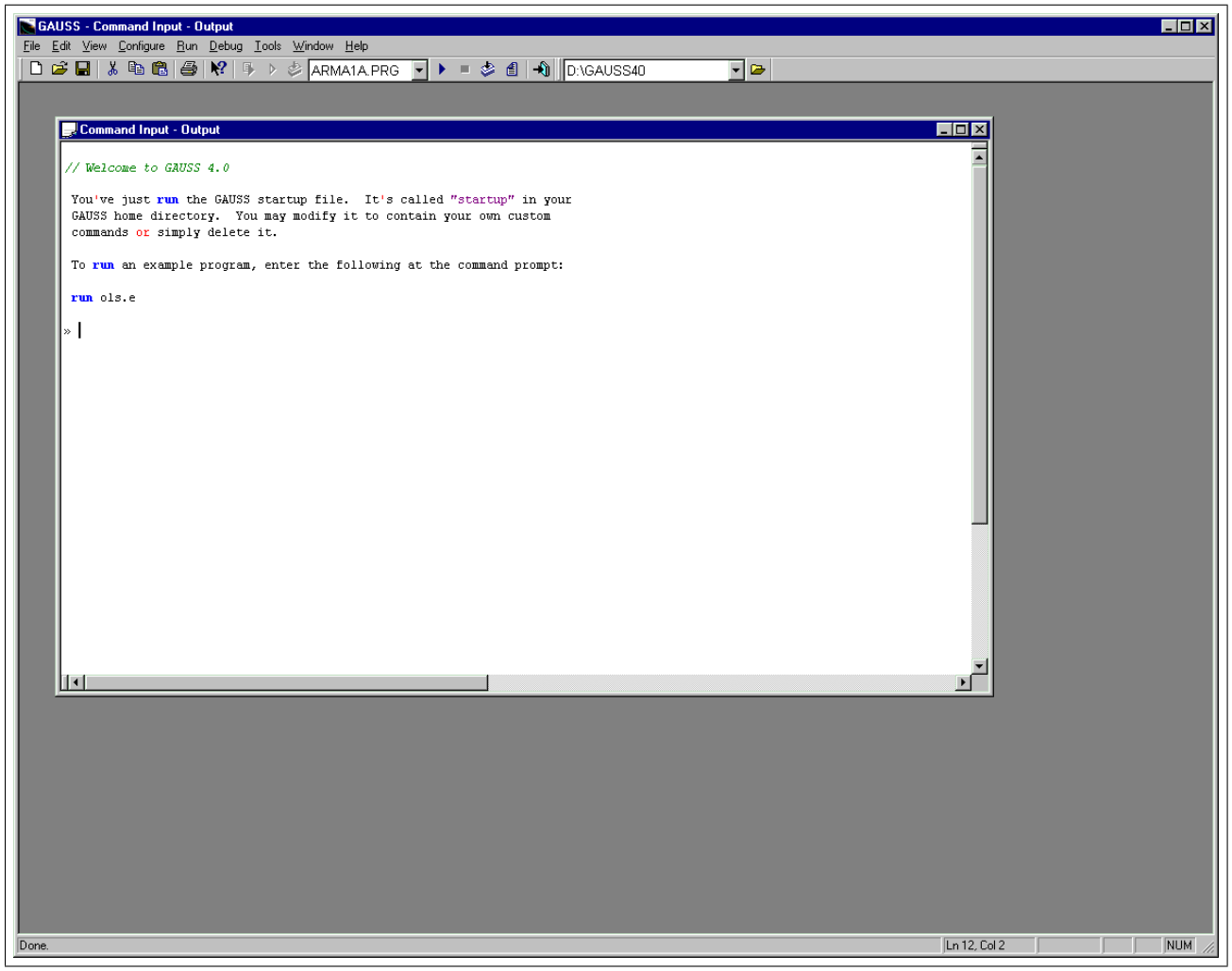
`http ://gro.creditlyonnais.fr`

– Maintenant, nous allons utiliser des bibliothèques qui vont nous poser beaucoup plus de difficultés[1]...

[1]La décence m'interdit de les nommer. Néanmoins, je trouve parfaitement scandaleux de mettre du code "approximatif" sur Internet.

Quatrième partie
GAUSS 4.0

La version 4.0 de **GAUSS** bénéficie d'un nouvel environnement de programmation. Par exemple, l'éditeur a été modifié et supporte la syntaxe surlignée (**C**onfigure / **E**ditor properties).



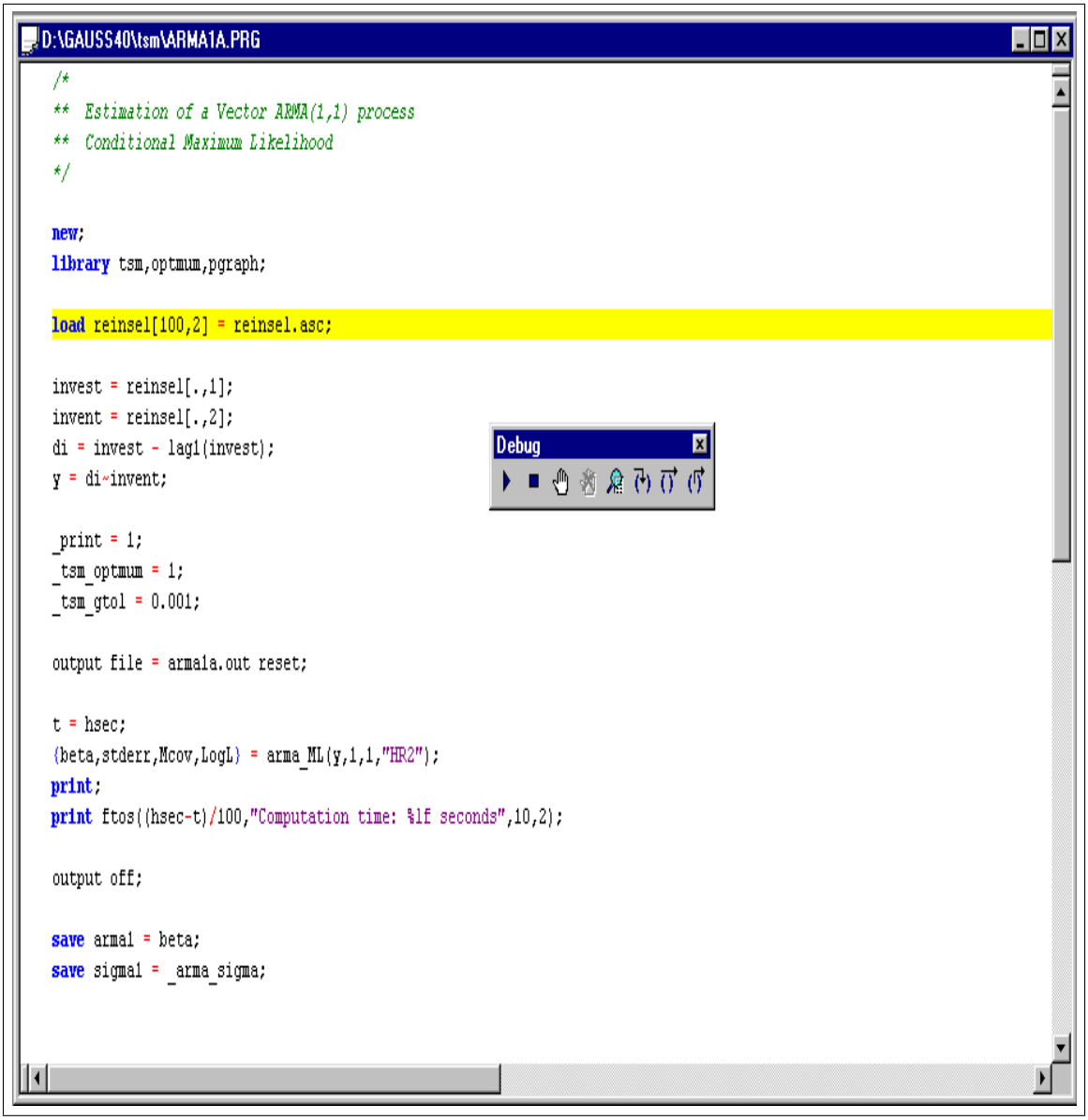
29 Le debugger

Le nouveau debugger est très proche de celui fourni avec les versions 3.5 et 3.6 de **GAUSS**. Deux types de points d'arrêt sont disponibles :

- Procedure breakpoints
- Line number breakpoints

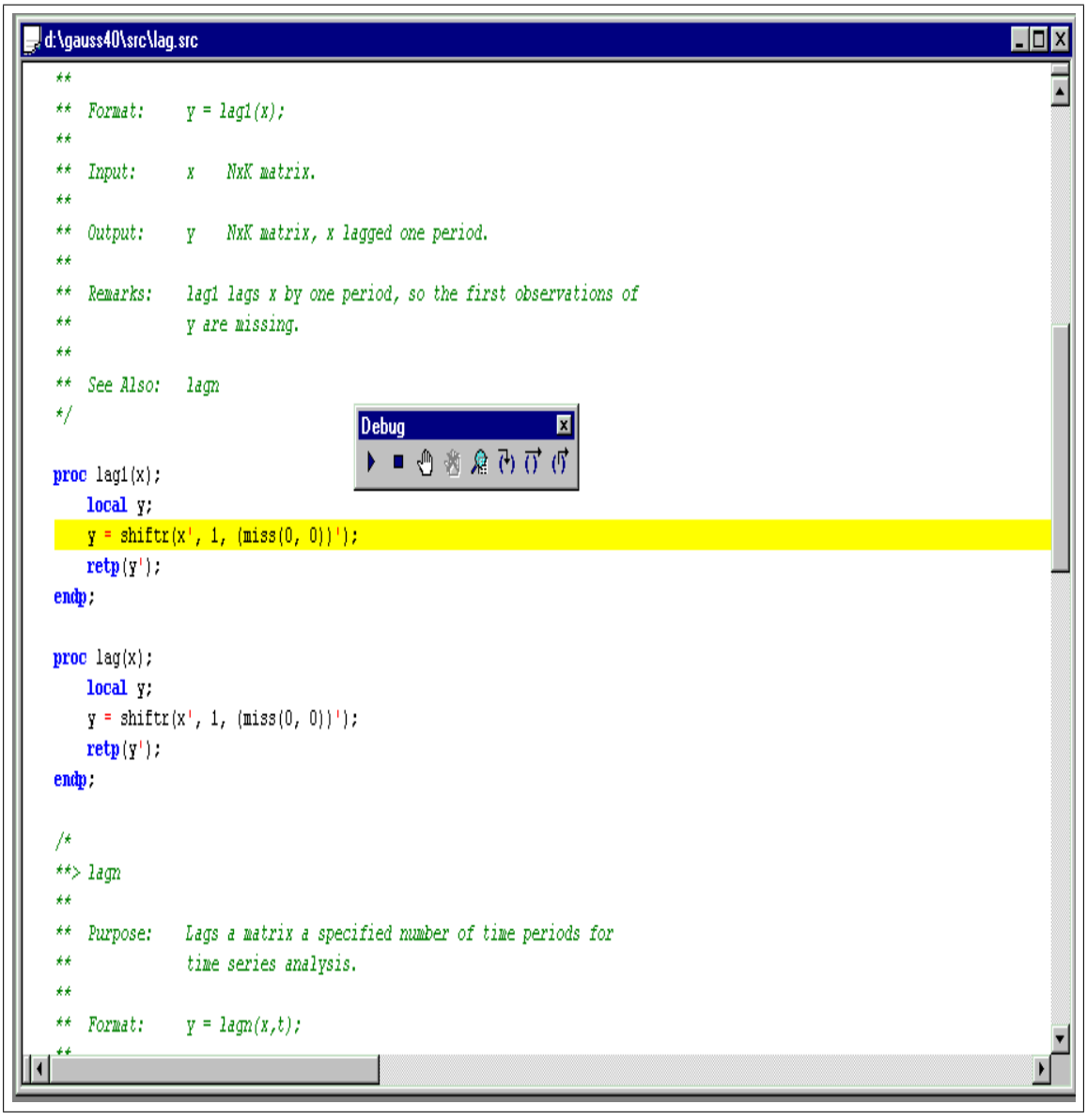
Dans le cas d'un point d'arrêt de type procédure, le debugger s'arrête lorsque la procédure est exécutée. De même, trois types d'exécution sont possibles : Step Into, Step Over, Step Out. On peut par exemple choisir

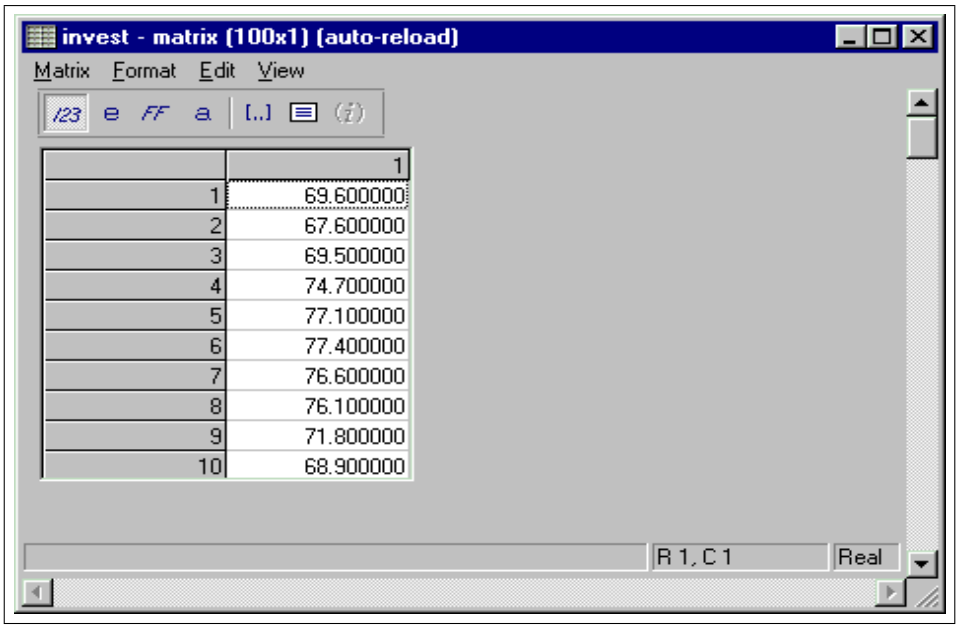
de rentrer ou non dans une procédure. Toutes les variables (globales et locales) sont éditables avec l'éditeur matriciel.

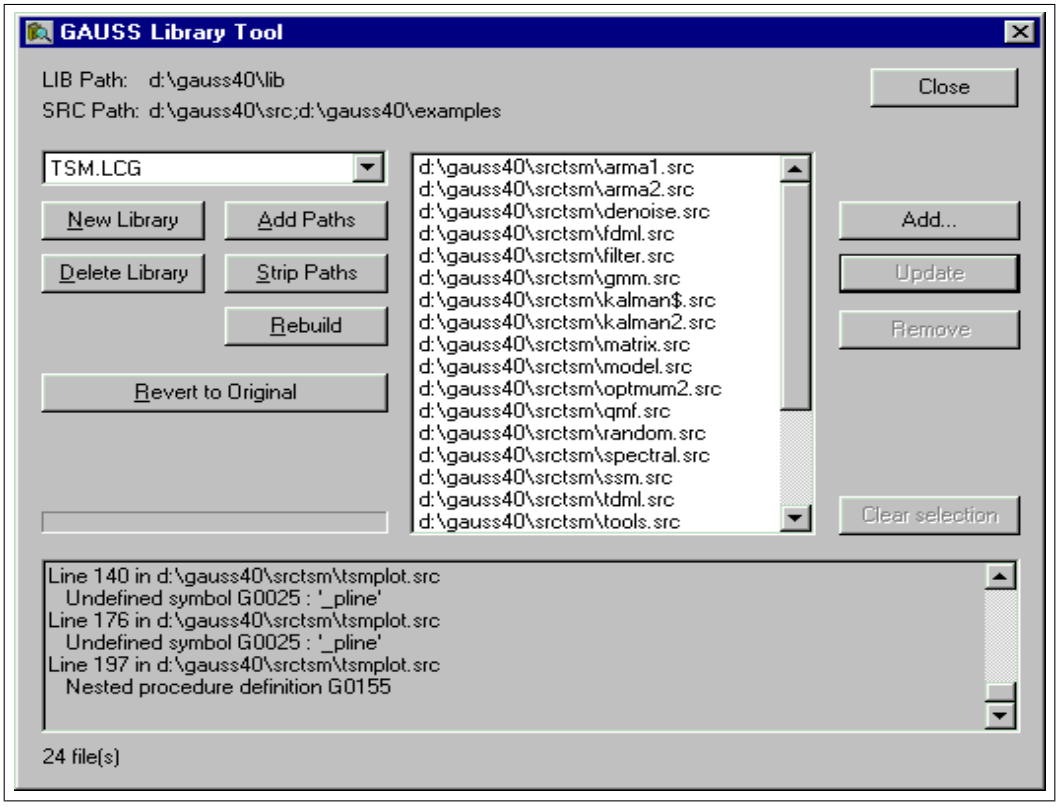


30 La gestion des bibliothèques

Dans les versions antérieures, la gestion des bibliothèques se faisait exclusivement avec la commande `lib`. Désormais, **GAUSS** est livré avec un utilitaire LibTool qui permet de gérer les bibliothèques de façon plus conviviale. Les options de la commande lib (`-update`, `-build`, `-delete`, `-list`, `-addpath`, `-gausspatch`, `-leavepath`, `-nopath`) sont disponibles dans LibTool.


d:\gauss40\src\lag.src
Debug
**
** Format: y = lag1(x);
**
** Input: x NxK matrix.
**
** Output: y NxK matrix, x lagged one period.
**
** Remarks: lag1 lags x by one period, so the first observations of
** y are missing.
**
** See Also: lagn
*/
proc lag1(x);
local y;
y = shiftr(x', 1, (miss(0, 0))');
retp(y');
endp;
proc lag(x);
local y;
y = shiftr(x', 1, (miss(0, 0))');
retp(y');
endp;
/*
**> lagn
**
** Purpose: Lags a matrix a specified number of time periods for
** time series analysis.
**
** Format: y = lagn(x,t);


invest - matrix (100x1) (auto-reload)
Matrix Format Edit View
1
1 69.600000
2 67.600000
3 69.500000
4 74.700000
5 77.100000
6 77.400000
7 76.600000
8 76.100000
9 71.800000
10 68.900000
R 1, C 1
Real


GAUSS Library Tool
LIB Path: d:\gauss40\lib
SRC Path: d:\gauss40\src;d:\gauss40\examples
Close
TSM.LCG
New Library
Add Paths
Delete Library
Strip Paths
Rebuild
Revert to Original
d:\gauss40\srctsm\arma1.src
d:\gauss40\srctsm\arma2.src
d:\gauss40\srctsm\denoise.src
d:\gauss40\srctsm\fdml.src
d:\gauss40\srctsm\filter.src
d:\gauss40\srctsm\gmm.src
d:\gauss40\srctsm\kalman$.src
d:\gauss40\srctsm\kalman2.src
d:\gauss40\srctsm\matrix.src
d:\gauss40\srctsm\model.src
d:\gauss40\srctsm\optmum2.src
d:\gauss40\srctsm\qmf.src
d:\gauss40\srctsm\random.src
d:\gauss40\srctsm\spectral.src
d:\gauss40\srctsm\ssm.src
d:\gauss40\srctsm\tdml.src
d:\gauss40\srctsm\tools.src
Add...
Update
Remove
Clear selection
Line 140 in d:\gauss40\srctsm\tsmplot.src
Undefined symbol G0025 : '_pline'
Line 176 in d:\gauss40\srctsm\tsmplot.src
Undefined symbol G0025 : '_pline'
Line 197 in d:\gauss40\srctsm\tsmplot.src
Nested procedure definition G0155
24 file(s)

31 Les générateurs de nombres aléatoires

La génération des nombres aléatoires a été légèrement modifiée. Désormais, deux nouveaux types de générateurs sont disponibles : **LC** et **KM**. Ils s'utilisent de façon récursive. Par exemple,

```
{x,state} = rndLCn(r,c,state);
```

Lors du premier appel au générateur, il convient de l'initialiser en spécifiant la variable `state`. Ensuite, il faut utiliser la nouvelle valeur de *state* pour générer d'autres nombres aléatoires.

32 Les structures

Pour déclarer une structure, nous utilisons la commande (ou le mot-clé) `struct`. 4 Types de variables peuvent être définis (comme pour la commande `declare`) :

1. **scalar**
2. **matrix**
3. **string**
4. **string array**

Voici un exemple de déclaration d'une structure à quatre membres :

```
struct EconometricModel {
matrix y;                   // Endogeneous variable
matrix x;                   // Exogeneous variables
string model;               // Model type = ols, nls
matrix f;                   // Pointer (function y = f(x;beta) + u)
};

fn fun(x,beta) = x[.,1]*beta[1] + x[.,2]^beta[2];

struct EconometricModel m1;
m1.f = &fun;
m1.model = ''nls'';
```

Une structure peut être un membre d'une autre structure :

```
struct person
{
string name;
string phone;
};

struct company
{
scalar id;
string name;
struct person boss;
};

struct company Vivendi;

Vivendi.name = ''Vivendi Universal'';
Vivendi.boss.name = ''J2M'';
```

Pour passer une structure dans une procédure, il faut utiliser le mot-clé `struct` dans les arguments. Une structure passée comme argument d'entrée dans une procédure est de type local. Si celle-ci est modifiée dans la procédure, alors **GAUSS** procède à une copie de la structure (ce qui est inefficient d'un point de vue programmation). Par exemple, il y a copie de la structure `cercle` dans la procédure perimeter, mais pas dans la procédure `area`.

```
new;

struct cercle
{
scalar x;
scalar y;
scalar r;
scalar area;
scalar perimeter;
};


proc (1) = area(struct cercle c);
  local r;
  r = c.r;
  retp(pi*r^2);
endp;

proc (1) = perimeter(struct cercle c);
  c.perimeter = 2*pi*c.r;
  retp(c.perimeter);
endp;

struct cercle rond;

rond.r = 1;
a = area(rond);

print a;
print rond.area;

p = perimeter(rond);
print p;
print rond.perimeter;

       3.1415927
      0.00000000
       6.2831853
      0.00000000
```

Une procédure peut retourner des structures. Il faut néanmoins que celles-ci soient déclarées.

```
new;

struct cercle
{
scalar x;
scalar y;
```

```
scalar r;
scalar area;
scalar perimeter;
};

proc (1) = area_perimeter(struct cercle c1);
  local r;
  struct cercle c2;

  r = c1.r;

  c2.x = c1.x;
  c2.y = c1.y;
  c2.r = r;

  c2.area = pi*r^2;
  c2.perimeter = 2*pi*r;

  retp(c2);
endp;

struct cercle rond;
rond.r = 1;

struct cercle c;

c = area_perimeter(rond);

print c.area;
print c.perimeter;


       3.1415927
       6.2831853
```

33 Les nouvelles commandes

- `getnamef`
- `vartypef`
- `tkf2ps`
- `tkf2eps`
- Les commandes **LaPack**

Cinquième partie
APPLICATIONS ECONOMETRIQUES

34 L'analyse en composantes principales

Soit $X = \{x_{i,j}\}$ une matrice de scores de dimension $n \times k$ — les indices i et j désignent respectivement les individus et les variables. Nous notons $M = \{m_{i,j}\}$ la matrice des scores réduits, c'est à dire

$$m_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\sigma(\bar{x}_j)} \tag{65}$$

Dans l'analyse en composantes principales, on définit le j-ième axe par la relation linéaire :

$$\mathcal{S}_j(x) = v_{1,j}x_1 + \ldots + v_{j,k}x_k \tag{66}$$

Si on désigne par V la matrice des coefficients des k axes, V est la matrice des vecteurs propres associés à $M^\top M/n$. Par construction, nous avons

$$\sum_{i=1}^{n} \mathcal{S}_j(x_i) = 0 \tag{67}$$

et

$$\sum_{i=1}^{n} \mathcal{S}_j^2(x_i) = \lambda_j \tag{68}$$

avec λ_j la valeur propre du j-ième vecteur propre. Notons aussi que

$$\text{var}[M] = \sum_{j=1}^{k} \lambda_j = k \tag{69}$$

La qualité de représentation d'un axe factoriel est définie par $\text{QLT}(j) = \lambda_j/k$.

En adoptant une métrique euclidienne, la distance entre deux individus est

$$d^2(i_1, i_2) = \sum_{j=1}^{k} (m_{i_1,j} - m_{i_2,j})^2 \tag{70}$$

Il vient que la distance entre l'individu i et l'individu moyen est

$$d^2(i) = \sum_{j=1}^{k} d_j^2(i) = \sum_{j=1}^{k} m_{i,j}^2 \tag{71}$$

$d_j^2(i)$ est la distance entre l'individu i et l'individu moyen sur le j-ième axe factoriel. La qualité de représentation de l'individu i sur l'axe j correspond à

$$\text{QLT}(i; j) = d_j^2(i)/d^2(i) \tag{72}$$

La tableau de saturation $\mathbf{S}$ de dimension $k \times k$ est défini par

$$s_{j_1,j_2} = v_{j_1,j_2}\sqrt{\lambda_{j_2}} \tag{73}$$

On peut alors calculer la qualité de représentation de la variable j_1 sur l'axe j_2 :

$$\text{QLT}(j_1; j_2) = s_{j_1,j_2}^2 \tag{74}$$

ainsi que la contribution :

$$\text{CTR}(j_1; j_2) = s_{j_1,j_2}^2/\lambda_{j_2} \tag{75}$$

```
proc (4) = PCA(x,nf);
  local m,results,cor,n,k,va,ve;
  local QLT,QLTc,SAT,QLTvar,CTRvar,QLTind;

  n = rows(x); k = cols(x);
  m = sqrt(n/(n-1)) * ( x - meanc(x)' ) ./ stdc(x)';
  cor = m'm / n;
  {va,ve} = eighv(cor);         /* Matrice hermitienne */

  ve = submat(real(ve),0,seqa(k,-1,k));
  va = real(va); va = rev(va .* dotfne(va,0));

  if nf < 1 or nf > k;
    nf = k;
  endif;

  va = va[1:nf]; ve = ve[.,1:nf];

  QLT = 100*va/sumc(diag(cor));
  QLTc = cumsumc(QLT);

/* Etude par variables */

  SAT = ve .* sqrt(va');    /* Tableau des saturations */
  QLTvar = 100 * SAT^2;     /* Qualite de representation des variables */
  CTRvar = QLTvar ./ va';   /* Contribution des variables              */

  results = 0;
  results = vput(results,QLT,''QLT'');
  results = vput(results,QLTc,''QLTc'');
  results = vput(results,SAT,''SAT'');
  results = vput(results,QLTvar,''QLTvar'');
  results = vput(results,CTRvar,''CTRvar'');

/* Etude par individus */

  QLTind = 100 * (m * ve)^2 ./ miss( sumc(m' .* m'), 0);

  results = vput(results,QLTind,''QLTind'');

  retp(m,va,ve,results);
endp;
```

Voyons un exemple très simple. Nous considérons 6 élèves et trois notes :

	Maths	Français	Sport
A	10	10	10
B	18	8	12
C	4	17	12
D	6	10	17
E	10	12	14
F	10	12	8

Dans le programme qui suit, nous faisons une ACP et nous représentons les individus sur le plan factoriel.

```
new;
library ritme,pgraph;

data = {10 10 10,
        18  8 12,
         4 17 12,
         6 10 17,
        10 12 14,
        10 12  8};

{x,va,ve,results} = PCA(data,0);

QLT = vread(results,''QLT'');
QLTc = vread(results,''QLTc'');

QLTvar = vread(results,''QLTvar'');
QLTind = vread(results,''QLTind'');

CTRvar = vread(results,''CTRvar'');
score = x * ve;

graphset;
  _pdate = ''''; _pnum = 2; _pcross = 1; _pframe = 0;
  _pmsgstr = ''A\000B\000C\000D\000E\000F'';
  _pmsgctl = (0.05+score[.,1])~score[.,2]~ones(6,1)*0.2~
             ones(6,1)*315~ones(6,1)~ones(6,1)*7~ones(6,1)*5;
  _psym = score[.,1 2]~ones(6,1)*8~ones(6,1)*3~ones(6,1)*2~ones(6,2);
  xtics(-3,3,1,0); ytics(-3,3,1,0);
  xlabel(''\214First axis''); ylabel(''\214Second Axis'');
  graphprt(''-c=1 -cf=pca1.ps'');
  draw;
```

Exercice 1 *Représenter les individus sur l'hyper plan factoriel de dimension 3. Pour cela, utiliser la variable* `_psym3d`.

L'exemple suivant est tiré de "La Structure Par Terme des Taux Zéro" de Roncalli [1998] — les programmes **GAUSS** sont disponibles sur le site web http ://www.city.ac.uk/cubs/ferc/thierry/gauss.html. La base de données *NS_FRF* contient l'ensemble des coefficients estimés de modèle de Nelson et Siegel pour la France pour la période allant du 10/02/1994 au 30/08/1996. A partir de cette base, nous pouvons construire les taux zéro pour n'importe quelle maturité avec la procédure suivante :

```
proc NelsonSiegel_TauxZero(coefficients,tau);
  local Nobs,mu1,mu2,mu3,tau1;
  local w,ww,www;
  local composante1,composante2,composante3,TauxZero;

  Nobs = rows(tau);

  mu1 = coefficients[1,.];
  mu2 = coefficients[2,.];
  mu3 = coefficients[3,.];
  tau1 = coefficients[4,.];
```

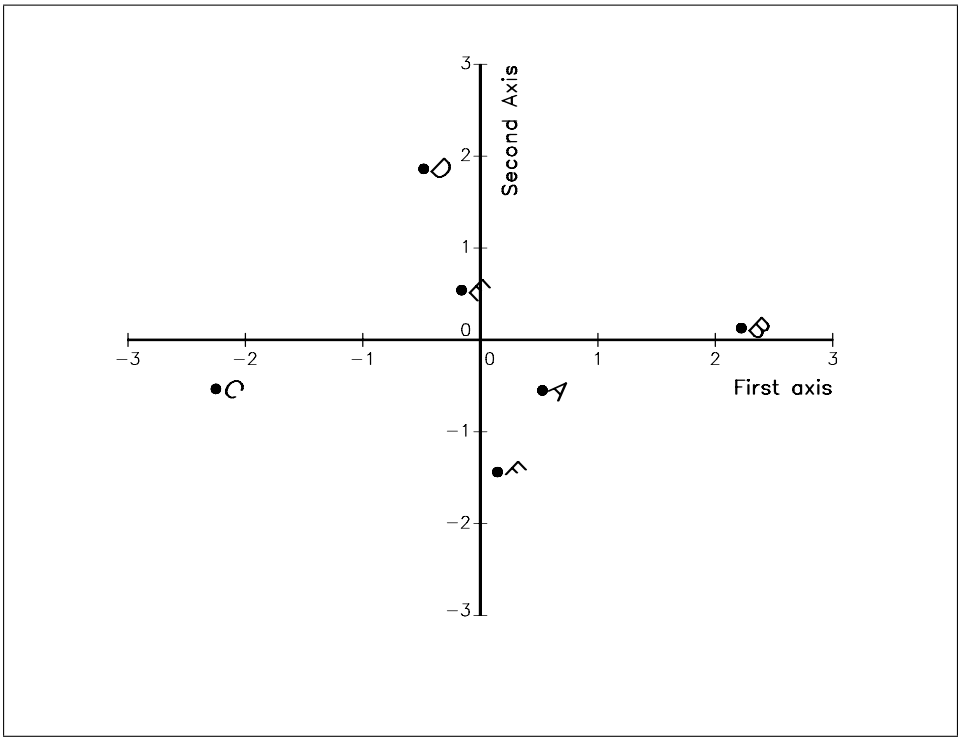


FIG. 7 – Plan factoriel de l'ACP

```
  w = tau./tau1;
  ww = exp(-w);
  www = (1-ww)./w;

  composante1 = mu1.*ones(Nobs,1);
  composante2 = mu2.*www;
  composante3 = mu3.*(www-ww);

  TauxZero = composante1 + composante2 + composante3;

  retp(TauxZero);
endp;
```

Le programme suivant fait une analyse en composantes principales de la structure par terme des taux zéro et fait le graphe du cercle des corrélations.

```
new;
library ritme,pgraph;

#include nelson.src;

load NS_frf;
Dates_ = NS_frf[.,1]; coefficients = NS_frf[.,2 3 4 5];

R0 = coefficients[.,1] + coefficients[.,2];
Rinf = coefficients[.,1];
tau = 1/12|2/12|3/12|6/12|9/12|1|2|3|4|5|6|7|8|9|10|15|20|25|30;
```

```
TauxZero = NelsonSiegel_TauxZero(coefficients',tau)';
TauxZero = R0~TauxZero~Rinf;

outwidth 256;

output file = pca2.out reset;

{x,va,ve,results} = pca(TauxZero,5);

print ''                  ----------------------------------'';
print ''Vecteurs propres'';
print ve;

output off;

sat = vread(results,''sat'');

c1 = sat[.,1];
c2 = sat[.,2];
t = seqa(0,2*pi/100,101);

k = cols(TauxZero);

graphset;
  _pnum = 2; _pdate = ''''; _pcross = 1; _pframe = 0; _pnum = 2;
  fonts(''simplex simgrma'');
  title('' '');
  _pmsgstr = ''0\0001/12\0002/12\0003/12\0006/12\0009/12\0001''\
             ''\0002\0003\0004\0005\0006\0007''\
             ''\0008\0009\00010\00015\00020\00025\00030\000\202\56\201'';
  _pmsgctl = 0.00+c1~c2~ones(k,5).*(0.17~0~1~7~0.1);
  _psym = c1~c2~ones(k,5).*(8~3~2~1~1);
  xtics(0.75,1,0.05,0);
  ytics(-0.7,0.3,0.2,0);
  xlabel(''\2141[er] axe factoriel        '');
  ylabel(''\2142[eme] axe factoriel                    '');
  graphprt(''-c=1 -cf=pca2.ps'');
  xy(cos(t),sin(t));
```

35 Les méthodes de régression

La procédure suivante permet d'afficher de façon "propre" les paramètres d'une régression, ainsi que les écart-types, les t de student et les probabilités marginales.

```
proc (0) = PrintCoeffs(beta,stderr,tstudent,pvalue,cov);
  local k,parnm,cor,omat,mask,fmt;

  k = rows(beta);

  if ( __ALTNAM /= 0 ) and ( rows(__ALTNAM) == K);
    parnm = __ALTNAM;
```

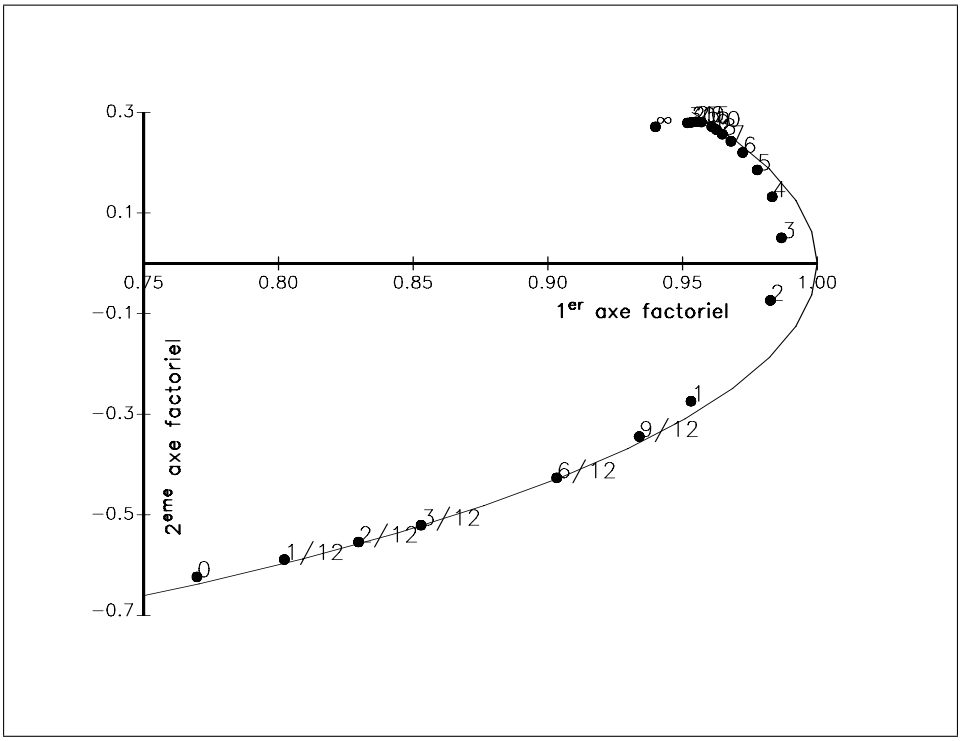


FIG. 8 – Cercle des corrélations de l'ACP des taux zéro

```
else;
  parnm = 0 $+ ''P'' $+ ftocv(seqa(1,1,K),1,0);
endif;

print;  print;
print ''-------------------------------------------------------------------------------'';
print ''Parameters     Estimates        Std-err.    ''\
      ''  T-statistic         P-value   '';
print ''-------------------------------------------------------------------------------'';

let fmt[5,3]=  ''-*.*s''  8 8  ''*.*lf'' 16 6 ''*.*lf'' 16 6
               ''*.*lf'' 17 6  ''*.*lf'' 16 6;
omat = parnm~beta~stderr~tstudent~pvalue;
call printfm(omat,0~1~1~1~1,fmt);
print;
print;

if cov /= 0;
  stderr = miss(stderr,0);
  cor = cov ./ stderr ./ stderr';
  omat = miss(0,0) ~parnm' | parnm ~miss(upmat1(cor),0);
  mask = zeros(1,k+1) | zeros(k,1) ~ones(k,k);
  print ''---------------------------'';
  print ''Correlation matrix '';
  print ''---------------------------'';
```

```
    call printfm(omat,mask,''*.*lf''~8~3);
    print;
  endif;

  retp;
endp;
```

35.1 Les moindres carrés ordinaires

Nous considérons le modèle linéaire standard :

$$y_i = \mathbf{x}_i\beta + u_i \tag{76}$$

avec $u_i \sim N\left(0, \sigma^2\right)$. L'estimateur des moindres carrés est

$$\hat{\beta}_{\mathsf{OLS}} = \left(X^\top X\right)^{-1} X^\top Y \tag{77}$$

Sa matrice de covariance estimée est

$$\text{var}\left[\hat{\beta}_{\mathsf{OLS}}\right] = \hat{\sigma}^2 \left(X^\top X\right)^{-1} \tag{78}$$

Voici une procédure qui permet d'estimer $\hat{\beta}_{\mathsf{OLS}}$ et var $\left[\hat{\beta}_{\mathsf{OLS}}\right]$:

```
new;
library ritme;

rndseed 123;

Nobs = 100;
K = 5;
sigma = 0.5;

x = ones(Nobs,1)~rndu(Nobs,K);
beta = seqa(1,1,K+1);
y = x*beta + rndn(Nobs,1)*sigma;

output file = ols4.out reset;

call ols(y,x);

output off;

__output = 0;

{beta,stderr,cov,u} = ols(y,x);


================================================================================
 OLS                                                    4/06/2002   7:52 pm
================================================================================
Number of observations:   100
Number of exogeneous:       6

Degrees of freedom:        94
```

```
------------------------------------------------------------------------------
Parameters     Estimates        Std-err.      T-statistic         P-value
------------------------------------------------------------------------------
P1              0.743400        0.228847         3.248453        0.001610
P2              2.004655        0.191707        10.456887        0.000000
P3              3.087867        0.185820        16.617507        0.000000
P4              4.307761        0.186731        23.069299        0.000000
P5              5.413424        0.182912        29.595702        0.000000
P6              5.665314        0.184540        30.699685        0.000000


---------------------------
Correlation matrix
---------------------------
       .      P1      P2      P3      P4      P5      P6
      P1   1.000  -0.372  -0.422  -0.565  -0.522  -0.330
      P2       .   1.000   0.006   0.050  -0.117  -0.156
      P3       .       .   1.000   0.029   0.064  -0.087
      P4       .       .       .   1.000   0.206  -0.004
      P5       .       .       .       .   1.000   0.076
      P6       .       .       .       .       .   1.000
```

Pour faire de la prévision, nous utilisons la méthode de Salkever (voir Greene [2000], page 308) :

$$\begin{bmatrix} y \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} X & \mathbf{0} \\ X^0 & -I \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} + \begin{bmatrix} u \\ u^0 \end{bmatrix} \tag{79}$$

Cette méthode consiste à faire une régression augmentée et nous avons

$$\begin{aligned} \hat{y}^0 &= X^0\hat{\beta} \\ &= \hat{\gamma} \end{aligned} \tag{80}$$

L'intérêt de cette méthode est de fournir directement la matrice de covariance de $\hat{y}^0$.

```
proc (2) = ols_predict(y,x,x0);
  local k,n,n0,n_star,k_star,df_star;
  local x_star,y_star,xx_star,xy_star,xx_star_inv;
  local beta_star,u_star,sigma2_star,cov_star;
  local y0,s0;

  k = cols(x);
  n = rows(x);

  n0 = rows(x0);
  n_star = n + n0;
  k_star = k + n0;
  df_star = n_star - k_star;

  x_star = x  ~zeros(n,n0) |
           x0 ~-eye(n0)    ;
  y_star = y | zeros(n0,1);
```

```
  xx_star = x_star'x_star;
  xy_star = x_star'y_star;
  xx_star_inv = invpd(xx_star);

  beta_star = xx_star_inv * xy_star;
  u_star = y_star - x_star*beta_star;
  sigma2_star = sumc(u_star^2)/df_star;

  cov_star = sigma2_star * xx_star_inv;

  y0 = trimr(beta_star,k,0);
  s0 = trimr(sqrt(diag(cov_star)),k,0);

  retp(y0,s0);
endp;
```

Voyons l'exemple de Greene. Les données sont les suivantes :

```
0.161 1.058 5.16 4.40
0.172 1.088 5.87 5.15
0.158 1.086 5.95 5.37
0.173 1.122 4.88 4.99
0.195 1.186 4.50 4.16
0.217 1.254 6.44 5.75
0.199 1.246 7.83 8.82
0.163 1.232 6.25 9.31
0.195 1.298 5.50 5.21
0.231 1.370 5.46 5.83
0.257 1.439 7.46 7.40
0.259 1.479 10.28 8.64
0.225 1.474 11.77 9.31
0.241 1.503 13.42 9.44
0.204 1.475 11.02 5.99
```

Le programme est le suivant :

```
new;
library ritme;

load data[15,4] = greene1.asc;

constant = ones(15,1);
trend = seqa(1,1,15);

y = data[.,1];
x = constant~trend~data[.,2 3 4];

let __altnam = ''Constant'' ''Trend'' ''GNP'' ''Int.rate'' ''Inf.rate'';

output file = ols5.out reset;

{beta,stderr,cov,u} = ols(y,x);

output off;
```

```
let x0[1,5] = 1 16 1.5 10 4;
{y0,s0} = ols_predict(y,x,x0);
t = cdftci(0.05/2,10);

output file = ols5.out on;

print ''         Lower           Predict          Upper'';
print y0+(-1~0~1).*t*s0;

output off;


================================================================================
 OLS                                                    4/06/2002   8:43 pm
================================================================================
Number of observations:    15
Number of exogeneous:       5

Degrees of freedom:        10


--------------------------------------------------------------------------------
Parameters     Estimates        Std-err.      T-statistic         P-value
--------------------------------------------------------------------------------
Constant       -0.509071        0.055128        -9.234394        0.000003
Trend          -0.016580        0.001972        -8.408927        0.000008
GNP             0.670383        0.054997        12.189407        0.000000
Int.rate       -0.002326        0.001219        -1.908270        0.085449
Inf.rate       -0.000094        0.001347        -0.069768        0.945754


---------------------------
Correlation matrix
---------------------------
       .     Con     Tre     GNP     Int     Inf
     Con   1.000   0.942  -0.993   0.083  -0.043
     Tre       .   1.000  -0.938  -0.120  -0.016
     GNP       .       .   1.000  -0.109  -0.031
     Int       .       .       .   1.000  -0.457
     Inf       .       .       .       .   1.000

         Lower           Predict          Upper
      0.18478390       0.20758273       0.23038155
```

35.2 Les moindres carrés non-linéaires

Un modèle non-linéaire standard est de la forme

$$y_i = g\left(x_i; \beta\right) + u_i \tag{81}$$

avec $u_i \sim \mathcal{N}\left(0, \sigma^2\right)$. L'estimateur des moindres carrés non linéaires est alors défini par

$$\hat{\beta}_{\mathsf{NLS}} = \arg\min \sum_{i=1}^{n} \left(y_i - g\left(x_i; \beta\right)\right)^2 \tag{82}$$

Afin de pouvoir estimer des modèles non-linéaires plus généraux, nous préférons considérer les modèles de la forme :

$$u_i = h\left(y_i, x_i, u_{i-1}, \ldots, u_{i-q}; \beta\right) \tag{83}$$

avec

$$u_i \mid y_i, x_i, u_{i-1}, \ldots, u_{i-q} \sim \mathrm{N}\left(0, \sigma^2\right) \tag{84}$$

Définissons l'estimateur $\hat{\beta}_{\mathsf{CLS}}$ comme la solution du programme d'optimisation

$$\sum_{i=1}^{n} h^2\left(y_i, x_i, u_{i-1}, \ldots, u_{i-q}; \beta\right) \tag{85}$$

$\hat{\beta}_{\mathsf{CLS}}$ est appelé l'estimateurs des moindres carrés conditionnels ("conditional least squares"). Notons $\hat{\mathbf{u}}\left(\beta\right)$ le vecteur de dimension $n \times 1$ dont les éléments s'écrivent

$$\hat{u}_i\left(\beta\right) = h\left(y_i, x_i, \hat{u}_{i-1}, \ldots, \hat{u}_{i-q}; \beta\right) \tag{86}$$

Nous pouvons montrer que

$$\mathrm{var}\left[\hat{\beta}_{\mathsf{CLS}}\right] = \hat{\sigma}^2\left(H^\top H\right)^{-1} \tag{87}$$

avec

$$H = \frac{\partial\, \hat{\mathbf{u}}\left(\hat{\beta}_{\mathsf{CLS}}\right)}{\partial\, \beta^\top} \tag{88}$$

```
declare matrix _nlls_h;
declare matrix _nlls_cov = 1;

proc (4) = NLLS(h,sv);
  local beta,rss,grd,retcode;
  local u,n,k,df,sigma2,J;
  local cov,stderr,tstudent,pvalue;

  _nlls_h = h;
  output off;
  {beta,rss,grd,retcode} = optmum(&_NLLS_rss,sv);
  output on;

  u = _NLLS_u(beta);

  n = rows(packr(u));
  k = rows(beta);
  df = n - k;
  sigma2 = rss/df;

  if _nlls_cov;
    J = packr(gradp(&_NLLS_u,beta));
    cov = sigma2*invpd(J'J);
  else;
    cov = miss(zeros(k,k),0);
```

```
  endif;
  stderr = sqrt(diag(cov));

  if __output;
    tstudent = beta./stderr;
    pvalue = 2*cdftc(abs(tstudent),df);
    call Header(''NLLS - Non-Linear Least Squares'','''',0);
    print ftos(n   ,''Number of observations: %lf'',5,0);
    print ftos(k   ,''Number of exogeneous:   %lf'',5,0);
    print;
    print ftos(df  ,''Degrees of freedom:     %lf'',5,0);
    PrintCoeffs(beta,stderr,tstudent,pvalue,cov);
  endif;

  retp(beta,stderr,cov,u);
endp;

proc _NLLS_u(beta);
  local h,u;
  h = _nlls_h;
  local h:proc;
  u = h(beta);
  retp( u );
endp;

proc _NLLS_rss(beta);
  local u,rss;
  u = _NLLS_u(beta);
  rss = sumc( packr(u .* u) );
  retp(rss);
endp;
```

Voyons un premier exemple. Le modèle considéré est le suivant

$$y_i = \frac{\beta_1 x_i}{\beta_2 + x_i} + u_i \tag{89}$$

```
new;
library ritme,optmum;

/*  treated  */
y1 = { 76,   47,   97,  107,  123,  139,  159,  152,  191,  201,  207,  200 };
x1 = { .02,  .02,  .06,  .06,  .11,  .11,  .22,  .22,  .56,  .56,  1.1,  1.1 };

/*  untreated  */
y2 = { 67,   51,   84,   86,   98,  115,  131,  124,  144,  158,  160 };
x2 = { .02,  .02,  .06,  .06,  .11,  .11,  .22,  .22,  .56,  .56,  1.1 };

y = y1 | y2;
x = x1 | x2;

proc model(b);
    retp(y - b[1]*x./(b[2] + x));
endp;
```

```
let sv = 100 5;

__output = 1;

output file = nlls1.out reset;

{beta,stderr,cov,u} = nlls(&model,sv);

output off;


================================================================================
 NLLS - Non-Linear Least Squares                          4/06/2002  11:36 pm
================================================================================
Number of observations:    23
Number of exogeneous:       2

Degrees of freedom:        21


--------------------------------------------------------------------------------
Parameters     Estimates        Std-err.      T-statistic         P-value
--------------------------------------------------------------------------------
P1            190.806421        8.764599        21.770124        0.000000
P2              0.060389        0.010769         5.607894        0.000014


---------------------------
Correlation matrix
---------------------------
       .      P1      P2
      P1   1.000   0.776
      P2       .   1.000
```

Considérons maintenant le modèle SETAR(2,7,2) proposé par Tong [1990] pour modéliser la série lynx :

$$x_t = \begin{cases} \beta_1 + \beta_2 x_{t-1} + \beta_3 x_{t-2} + \beta_4 x_{t-3} + \beta_5 x_{t-4} + \\ \beta_6 x_{t-5} + \beta_7 x_{t-6} + \beta_8 x_{t-7} + u_t & \text{si } x_{t-2} \leq \dot{x} \\ \beta_9 + \beta_{10} x_{t-1} + \beta_{11} x_{t-2} + u_t & \text{sinon} \end{cases}$$

Outre les paramètres β_1 à β_{11}, il faut aussi déterminer la valeur optimale du seuil $\dot{x}$. Tong [1990] trouve $\dot{x} = 3.116$. Pour estimer le seuil $\dot{x}$, il convient de discrétiser le problème puisque la somme des carrés des résidus n'est pas différentiable pour $\dot{x}$.

```
new;
library ritme,optmum,pgraph;

declare matrix threshold;

load x[] = lynx.asc;
x = log(x);

proc setar(c);
```

```
  local n,beta,u,i;

  n = rows(x);
  beta = c[1:11];
  u = miss(zeros(n,1),0);

  i = 3;
  do until i > n;
    if x[i-2] > threshold;
      u[i] = x[i] - beta[9] - beta[10]*x[i-1] - beta[11]*x[i-2];
    else;
      if i < 8;
        i = i + 1;
        continue;
      endif;
      u[i] = x[i] - beta[1] - beta[2:8]'x[i-1:i-7];
    endif;
    i = i + 1;
  endo;

  retp(u);
endp;

xs = trimr(sortc(x,1),10,10);
n = rows(xs);
rss = zeros(n,1);
beta = zeros(11,n);
sv = zeros(11,1);

__output = 0;
_nlls_cov = 0;

i = 1;
do until i > n;

  cls;
  locate 5,5;
  print /flush ftos(i,''Iteration no. %lf'',5,0);

  threshold = xs[i];
  {beta[.,i],stderr,cov,u} = nlls(&setar,sv);
  u = packr(u);
  rss[i] = u'u;
  sv = beta[.,i];

  i = i + 1;
endo;

output file = nlls2.out reset;

opt = xs[minindc(rss)];
print ftos(opt,''Optimal value of the threshold: %lf'',4,3);
```

```
_nlls_cov = 1; __output = 1;
threshold = 3.116;
{beta,stderr,cov,u} = nlls(&setar,sv);

output off;

graphset;
  _pdate = ’’’’; _pnum = 2; _plwidth = 5; _pframe = 0;
  xlabel(’’\214threshold’’);
  ylabel(’’\214rss’’);
  _pline = 1~3~opt~0~opt~10~1~15~5 |
           1~4~3.116~0~3.116~10~1~12~5 ;
  graphprt(’’-c=1 -cf=nlls2.ps’’);
  xy(xs,rss);


Optimal value of the threshold: 3.310

================================================================================
 NLLS - Non-Linear Least Squares                          4/07/2002  12:01 am
================================================================================
Number of observations:   108
Number of exogeneous:      11

Degrees of freedom:        97


--------------------------------------------------------------------------------
Parameters     Estimates        Std-err.      T-statistic         P-value
--------------------------------------------------------------------------------
P1              0.545812        0.344560         1.584082        0.116430
P2              1.032041        0.118518         8.707883        0.000000
P3             -0.172987        0.195770        -0.883622        0.379085
P4              0.170638        0.186915         0.912918        0.363549
P5             -0.431039        0.191562        -2.250126        0.026700
P6              0.332410        0.212747         1.562466        0.121435
P7             -0.284128        0.209532        -1.356016        0.178243
P8              0.209504        0.127250         1.646401        0.102918
P9              2.346535        0.549563         4.269823        0.000046
P10             1.532825        0.088633        17.294132        0.000000
P11            -1.276113        0.165136        -7.727642        0.000000
```

35.3 Les estimateurs 2SLS et 3SLS

Considérons le système suivant de M équations avec $t = 1, \ldots, T$

$$\begin{cases} y_{t,1}\gamma_{1,1} + \cdots + y_{t,M}\gamma_{M,1} + x_{t,1}\beta_{1,1} + \cdots + x_{t,K}\beta_{K,1} + e_{t,1} = 0 \\ \vdots \\ y_{t,1}\gamma_{1,M} + \cdots + y_{t,M}\gamma_{M,M} + x_{t,1}\beta_{1,M} + \cdots + x_{t,K}\beta_{K,M} + e_{t,M} = 0 \end{cases} \tag{90}$$

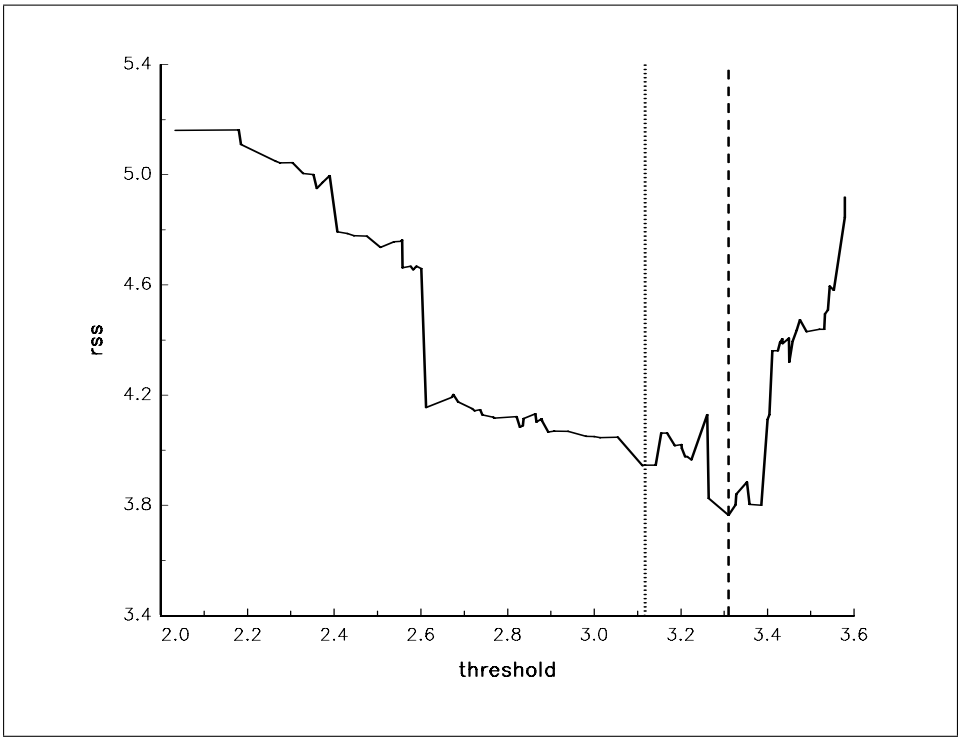


FIG. 9 – Détermination du seuil optimal du modèle SETAR(2,7,2)

Soient

$$y_j = \{y_{t,j}\} = \begin{bmatrix} y_{1,j} \\ y_{2,j} \\ \vdots \\ y_{T,j} \end{bmatrix}, \; x_j = \{x_{t,j}\} = \begin{bmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{T,j} \end{bmatrix}, \; \Gamma_j = \{\gamma_{m,j}\} = \begin{bmatrix} \gamma_{1,j} \\ \gamma_{2,j} \\ \vdots \\ \gamma_{M,j} \end{bmatrix}, \; B_j = \{\beta_{k,j}\} = \begin{bmatrix} \beta_{1,j} \\ \beta_{2,j} \\ \vdots \\ \beta_{K,j} \end{bmatrix} \tag{91}$$

et

$$e_j = \{e_{t,j}\} = \begin{bmatrix} e_{1,j} \\ e_{2,j} \\ \vdots \\ e_{T;j} \end{bmatrix} \tag{92}$$

Le modèle (90) peut se mettre sous la forme

$$Y\Gamma + XB + E = 0 \tag{93}$$

avec $Y = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_M \end{bmatrix}$, $X = \begin{bmatrix} X_1 & X_2 & \cdots & X_M \end{bmatrix}$, etc. Le modèle (90) est généralement structurel. Ainsi, un certain nombre de coefficients $\gamma_{i,j}$ et $\beta_{i,j}$ sont nuls. Considérons alors X^i et Y^i les sous-matrices de X et Y qui correspondent à l'équation i du modèle structurel. Pour l'équation i, nous avons

$$Y^i\Gamma^i + X^i\beta^i + e^i = 0 \tag{94}$$

Nous décidons de normaliser le coefficient d'une variable endogène à -1. Soient y^i, cette variable endogène et $\bar{Y}^i$ la matrice Y^i sans la variable y^i telles que

$$\begin{array}{rcl} y^i & = & \bar{Y}^i\gamma^i + X^i\beta^i + e^i \\ & = & Z_i\delta_i + e_i \end{array} \tag{95}$$

avec

$$\delta_i = \begin{bmatrix} \gamma^i \\ \beta^i \end{bmatrix} \qquad \text{et} \qquad Z_i = \begin{bmatrix} \bar{Y}^i & X^i \end{bmatrix} \tag{96}$$

L'expression de l'estimateur 2SLS de δ_i est

$$\hat{\delta}_i^{\text{2SLS}} = \left[Z_i^\top X \left(X^\top X \right)^{-1} X^\top Z_i \right]^{-1} Z_i^\top X \left(X^\top X \right)^{-1} X^\top y^i \tag{97}$$

avec

$$\text{var}\left[\hat{\delta}_i^{\text{2SLS}} \right] = \left[Z_i^\top X \left(X^\top X \right)^{-1} X^\top Z_i \right]^{-1} \tag{98}$$

Nous en déduisons que

$$\hat{\Sigma} = \frac{\hat{u}^\top \hat{u}}{T} \tag{99}$$

où

$$\hat{u} = \begin{bmatrix} \hat{u}_1 & \hat{u}_2 & \cdots & \hat{u}_M \end{bmatrix} \tag{100}$$

et

$$\hat{u}_i = y^i - Z_i \hat{\delta}_i^{\text{2SLS}} \tag{101}$$

Soient

$$Z = \begin{bmatrix} Z_1 & & & \\ & Z_2 & & \\ & & \ddots & \\ & & & Z_M \end{bmatrix}, \qquad \mathbf{y} = \text{vec}\left(Y\right) \qquad \text{et} \qquad \delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_M \end{bmatrix} \tag{102}$$

alors l'estimateur 3SLS de δ est

$$\hat{\delta}^{\text{3SLS}} = \left[Z^\top \left[\hat{\Sigma}^{-1} \otimes X \left(X^\top X \right)^{-1} X^\top \right] Z \right]^{-1} Z^\top \left[\hat{\Sigma}^{-1} \otimes X \left(X^\top X \right)^{-1} X^\top \right] \mathbf{y} \tag{103}$$

avec

$$\text{var}\left[\hat{\delta}^{\text{3SLS}} \right] = \left[Z^\top \left[\hat{\Sigma}^{-1} \otimes X \left(X^\top X \right)^{-1} X^\top \right] Z \right]^{-1} \tag{104}$$

```
proc (4) = gls(y,x,gammaMatrix,bMatrix,normVector);
  local M,K,L,nobs,s,data,T,u,nreg;
  local ns,nsc,i,xx,z;
  local varnames,eqnames,depnames,ind;
  local y_,x_,z_,beta_,cov_,stderr_;
  local beta,stderr,cov,tstudent,pvalue,sigma,residuals,sigmaU;
  local rss,tss,df,Gamma2,SigmaXX;

  M = cols(y); K = cols(x); L = cols(bMatrix);

  nobs = rows(y); s = seqa(1,1,nobs); data = packr(s~y~x);
  s = data[.,1]; y = data[.,2:M+1]; x = data[.,M+2:M+K+1];

  T = rows(y);
  u = zeros(T,L);
  nreg = sumc(gammaMatrix|bMatrix)-1;
  ns = sumc(nreg); nsc = cumsumc(nreg);
  sigma = zeros(L,1);
```

```
gamma2 = GammaMatrix;
beta = {}; stderr = {}; tss = {}; df = {};
i = 1;
do until i > L;
  gamma2[normVector[i],i] = 0;
  i = i + 1;
endo;

xx = x*invpd(x'x)*x';
z = zeros(L*T,ns);

varnames = {}; eqnames = {}; depnames = {};
i = 1;
do until i > L;

  ind = indexcat(gamma2[.,i],1);
  if not scalmiss(ind);
    y_ = y[.,ind];
    eqnames = eqnames | ( 0 $+ ''Eq.'' $+ ftocv(i*ones(rows(ind),1),2,0) );
    varnames = varnames | ( 0 $+ ''Y'' $+ ftocv(ind,2,0) );
  else;
    y_ = {};
  endif;

  ind = indexcat(bMatrix[.,i],1);
  if not scalmiss(ind);
    x_ = x[.,ind];
    eqnames = eqnames | ( 0 $+ ''Eq.'' $+ ftocv(i*ones(rows(ind),1),2,0) );
    varnames = varnames | ( 0 $+ ''X'' $+ ftocv(ind,2,0) );
  else;
    x_ = {};
  endif;

  z_ = y_~x_;

  y_ = y[.,normVector[i]];
  depnames = depnames | ( 0 $+''Y'' $+ ftocv(normVector[i],2,0) );


  cov_ = inv(z_'xx*z_);
  beta_ = cov_ * z_'xx*y_;
  beta = beta | beta_;
  u[.,i] = y_ - z_*beta_;

  sigma[i] = sqrt(sumc(u[.,i]^2)/(T-nreg[i]));
  cov_ = sigma[i]^2 * cov_;

  stderr_ = sqrt(diag(cov_));
  stderr = stderr | stderr_ ;
  df = df | (T-nreg[i])*ones(nreg[i],1);
  tss = tss | sumc(y_^2);
```

```
    if i == 1;
      z[(1+(i-1)*T):i*T,1:nsc[1]] = z_;
    else;
      z[(1+(i-1)*t):i*t,(1+nsc[i-1]):nsc[i]] = z_;
    endif;

    i = i + 1;
  endo;

  tstudent = beta ./ stderr;
  pvalue = 2*cdftc(abs(tstudent),df);
  rss = diag(u'u);
  sigmaU = u'u / T;

  residuals = miss(zeros(nobs,L),0);
  residuals[s,.] = u;

  if __output;
    call glsPrint(2,eqnames,depnames,varnames,nobs,T,nreg,tss,rss,sigma,
                  beta,stderr,tstudent,pvalue,SigmaU);
  endif;

  SigmaXX = inv(sigmaU) .*. xx;
  cov = inv(z'SigmaXX*z);
  beta = cov * z'SigmaXX*vec(y[.,normVector]);
  stderr = sqrt(diag(cov));
  tstudent = beta ./ stderr;
  pvalue = 2*cdftc(abs(tstudent),df);

  u = reshape(vec(y[.,normVector]) - z*beta,L,T)';
  residuals[s,.] = u;
  rss = diag(u'u);
  sigmaU = u'u / T;

  df = T - ones(L,1)*K;
  sigma = sqrt(diag(sigmaU));

  if __output;
    call glsPrint(3,eqnames,depnames,varnames,nobs,T,nreg,tss,rss,sigma,
                  beta,stderr,tstudent,pvalue,SigmaU);
  endif;

  retp(beta,stderr,cov,residuals);
endp;

proc (0) = glsPrint(cn,eqnames,depnames,varnames,nobs,T,nreg,tss,rss,sigma,
                  beta,stderr,tstudent,pvalue,SigmaU);
  local omat,mask,fmt;

  if cn == 2;
    call header(''GLS-2SLS'','''',0);
  else;
```

```
    call header(''GLS-3SLS'','''',0);
  endif;

  print;
  print ftos(nobs,''Number of observations: %*.*lf'',5,0);
  print ftos(T,   ''Valid cases:            %*.*lf'',5,0);
  print;
  print;
  omat = unique(eqnames,0) ~depnames ~(T-nreg) ~tss ~rss ~sigma;
  mask = 0~0~1~1~1~1;
  let fmt[6,3] = ''-*.*s'' 11 5
                 ''-*.*s'' 10 5
                 ''*.*lf'' 6 0
                 ''*.*lf'' 20 5
                 ''*.*lf'' 16 5
                 ''*.*lf'' 14 5;
  print ''   Eq     Norm           DF                TSS''\
        ''            RSS         Sigma'';
  print ''-------------------------------------------------------------''\
        ''----------------'';
  call printfm(omat,mask,fmt);
  print;
  print;
  omat = eqnames ~varnames ~beta ~stderr ~tstudent ~pvalue;
  let fmt[6,3] = ''-*.*s'' 11 5 ''-*.*s'' 10 5 ''*.*lf'' 14 6
                 ''*.*lf'' 14 6 ''*.*lf'' 14 6 ''*.*lf'' 14 6;
  print ''   Eq      Var             Estimate       Std-Err''\
        ''   T-statistic       P-value'';
  print ''-------------------------------------------------------------''\
        ''----------------'';
  call printfm(omat,mask,fmt);
  print;
  print;
  print ''-----------------------------------'';
  print ''Correlation matrix of residuals'';
  print ''-----------------------------------'';
  call printfm(corrvc(sigmaU),1,''*.*lf''~8~3);
  print;

  retp;
endp;
```

Avec la procédure précédente, nous pouvons obtenir les estimateurs 2SLS ou 3SLS. Les variables `gammaMatrix` et `bMatrix` sont des matrices composées de 1 et 0 : elles permettent de savoir quels sont les coefficients contraints (c'est-à-dire ceux égaux à zéro) du modèle. Ces deux matrices sont définies respectivement par rapport à Γ et B. Si un coefficient n'est pas contraint, alors la valeur correspondante pour les matrices `gammaMatrix` et `bMatrix` est 1 (0 si le coefficient est contraint à 0). La variable `normVector` est un vecteur qui définit la variable endogène normalisée pour chaque équation.

L'exemple suivant est issu du livre de Judge, Hill, Griffiths, Lütkepohl et Lee [1988] (pages 656 à 663). Considérons le modèle

$$\left\{\begin{array}{l} y_{t,1}\gamma_{1,1} + y_{2,t}\gamma_{2,1} + y_{3,t}\gamma_{3,1} + x_{t,1}\beta_{1,1} + e_{t,1} = 0 \\ y_{t,1}\gamma_{1,2} + y_{2,t}\gamma_{2,2} + x_{t,1}\beta_{1,2} + x_{t,2}\beta_{2,2} + x_{t,3}\beta_{3,2} + x_{t,4}\beta_{4,2} + e_{t,2} = 0 \\ y_{t,2}\gamma_{2,3} + y_{3,t}\gamma_{3,3} + x_{t,1}\beta_{1,3} + x_{t,2}\beta_{2,3} + x_{t,5}\beta_{5,3} + e_{t,3} = 0 \end{array}\right.$$

Nous avons donc

$$\Gamma = \begin{bmatrix} \gamma_{1,1} & \gamma_{1,2} & 0 \\ \gamma_{2,1} & \gamma_{2,2} & \gamma_{2,3} \\ \gamma_{3,1} & 0 & \gamma_{3,3} \end{bmatrix}$$

et

$$B = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} \\ 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & \beta_{3,2} & 0 \\ 0 & \beta_{4,2} & 0 \\ 0 & 0 & \beta_{5,3} \end{bmatrix}$$

Nous pouvons donc définir les matrices correspondantes `gammaMatrix` :

$$\texttt{gammaMatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

et `bMatrix` :

$$\texttt{bMatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Il nous faut ensuite définir les variables de normalisation. Pour la première équation, nous avons choisi $y_{t,1}$, pour la seconde $y_{2,t}$, et pour la troisième $y_{3,t}$. Nous avons donc

$$\texttt{normVector} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
new;
library ritme;

cls;

let data[20,8] =
1       3.06    1.34    8.48    28      359.27  102.96  578.49
1       3.19    1.44    9.16    35      415.76  114.38  650.86
1       3.3     1.54    9.9     37      435.11  118.23  684.87
1       3.4     1.71    11.02   36      440.17  120.45  680.47
1       3.48    1.89    11.64   29      410.66  116.25  642.19
1       3.6     1.99    12.73   47      530.33  140.27  787.41
1       3.68    2.22    13.88   50      557.15  143.84  818.06
1       3.72    2.43    14.5    35      472.8   128.2   712.16
1       3.92    2.43    15.47   33      471.76  126.65  722.23
1       4.15    2.31    16.61   40      538.3   141.05  811.44
1       4.35    2.39    17.4    38      547.76  143.71  816.36
```

```
1        4.37    2.63    18.83   37      539     142.37  807.78
1        4.59    2.69    20.62   56      677.6   173.13  983.53
1        5.23    3.35    23.76   88      943.85  223.21  1292.99
1        6.04    5.81    26.52   62      893.42  198.64  1179.64
1        6.36    6.38    27.45   51      871     191.89  1134.78
1        7.04    6.14    30.28   29      793.93  181.27  1053.16
1        7.81    6.14    25.4    22      850.36  180.56  1085.91
1        8.09    6.19    28.84   38      967.42  208.24  1246.99
1        9.24    6.69    34.36   41      1102.61 235.43  1401.94;

x = data[.,1:5];
y = data[.,6:8];

let g[3,3] = 1 1 0
             1 1 1
             1 0 1;

let b[5,3] = 1 1 1
             0 1 1
             0 1 0
             0 1 0
             0 0 1;

let n[3] = 1 2 3;

output file = 3sls.out reset;

{beta,stderr,cov,u} = gls(y,x,g,b,n);

output off;
```

```
================================================================================
 GLS-2SLS                                                  4/08/2002   1:00 am
================================================================================

Number of observations:    20
Valid cases:               20
```

Eq	Norm	DF	TSS	RSS	Sigma
Eq.01	Y01	17	9183657.68560	5151.13017	17.40712
Eq.02	Y02	15	519670.34010	17.48548	1.07968
Eq.03	Y03	16	17525372.43180	334.42941	4.57185

Eq	Var	Estimate	Std-Err	T-statistic	P-value
Eq.01	Y02	-9.408595	1.959459	-4.801630	0.000166
Eq.01	Y03	2.409556	0.312774	7.703828	0.000001
Eq.01	X01	-65.894052	28.522020	-2.310287	0.033688

Eq.02	Y01	0.195939	0.003629	53.986868	0.000000
Eq.02	X01	39.554214	1.054224	37.519734	0.000000
Eq.02	X02	-3.860192	0.517703	-7.456387	0.000002
Eq.02	X03	-6.067779	0.478169	-12.689603	0.000000
Eq.02	X04	1.644120	0.137833	11.928342	0.000000
Eq.03	Y02	1.939869	0.526245	3.686248	0.002000
Eq.03	X01	-8.792440	8.127770	-1.081778	0.295387
Eq.03	X02	80.874048	9.530396	8.485907	0.000000
Eq.03	X05	5.069837	0.687619	7.373035	0.000002

Correlation matrix of residuals

1.000	0.740	-0.915
0.740	1.000	-0.561
-0.915	-0.561	1.000

==========

GLS-3SLS 4/08/2002 1:00 am

==========

Number of observations: 20
Valid cases: 20

Eq	Norm	DF	TSS	RSS	Sigma
Eq.01	Y01	17	9183657.68560	5333.37118	16.32999
Eq.02	Y02	15	519670.34010	18.50882	0.96200
Eq.03	Y03	16	17525372.43180	381.37757	4.36679

Eq	Var	Estimate	Std-Err	T-statistic	P-value
Eq.01	Y02	-9.640965	1.751683	-5.503829	0.000039
Eq.01	Y03	2.446697	0.279584	8.751192	0.000000
Eq.01	X01	-63.116145	25.760205	-2.450141	0.025406
Eq.02	Y01	0.198939	0.002224	89.435327	0.000000
Eq.02	X01	39.208588	0.772632	50.746814	0.000000
Eq.02	X02	-3.747980	0.322986	-11.604167	0.000000
Eq.02	X03	-6.139364	0.304588	-20.156313	0.000000
Eq.02	X04	1.543914	0.100502	15.362024	0.000000
Eq.03	Y02	2.223697	0.340035	6.539608	0.000007
Eq.03	X01	-11.869018	6.277135	-1.890834	0.076896
Eq.03	X02	75.888057	6.292314	12.060438	0.000000
Eq.03	X05	4.666784	0.413069	11.297836	0.000000

Correlation matrix of residuals

```
 1.000   0.811  -0.955
 0.811   1.000  -0.668
-0.955  -0.668   1.000
```

Nous obtenons

$$\hat{\Gamma}^{3\text{SLS}} = \begin{bmatrix} -1 & 0.19 & 0 \\ -9.64 & -1 & 2.22 \\ 2.44 & 0 & -1 \end{bmatrix}$$

et

$$\hat{B} = \begin{bmatrix} -93.11 & 39.20 & -11.86 \\ 0 & -3.74 & 75.88 \\ 0 & -6.13 & 0 \\ 0 & 1.54 & 0 \\ 0 & 0 & 4.66 \end{bmatrix}$$

Au seuil de 5%, seul le coefficient $\beta_{1,3}$ n'est pas significatif.

Remarque 22 *La plupart des logiciels économétriques ne permettent pas de choisir les variables de normalisation. Pourtant, le choix d'une normalisation n'est pas neutre. Calculez les matrices de la forme réduite du système, c'est-à-dire* $\hat{\Pi} = -\hat{B}\hat{\Gamma}^{-1}$*, pour deux normalisations distinctes et vérifiez que ces matrices sont différentes.*

36 Estimation des paramètres d'une équation différentielle stochastique

36.1 Simulation d'une solution d'équation différentielle stochastique

Considérons un processus de diffusion dont la représentation SDE est

$$\begin{cases} \mathrm{d}x(t) = \mu(t, x(t))\,\mathrm{d}t + \sigma(t, x(t))\,\mathrm{d}W(t) \\ x(t_0) = x_0 \end{cases} \tag{105}$$

avec $W(t)$ un mouvement brownien. Pour simuler une solution de ce processus, nous pouvons employer l'algorithme d'Euler-Maruyama :

$$X(t+k) = X(t) + k\mu(t, X(t)) + \sqrt{k}\sigma(t, X(t))\,\epsilon(t) \tag{106}$$

où $\epsilon(t)$ est un bruit blanc gaussien standard. Pour le mouvement brownien géométrique

$$\mathrm{d}x(t) = \mu x(t)\,\mathrm{d}t + \sigma x(t)\,\mathrm{d}W(t)$$

ou pour le processus d'Ornstein-Uhlenbeck

$$\mathrm{d}x(t) = a(b - x(t))\,\mathrm{d}t + \sigma x(t)\,\mathrm{d}W(t)$$

nous pouvons employer des algorithmes exacts.

```
proc (2) = rndGBM(x0,mu,sigma,t0,TT,N,Ns);
  local k,t,x,k1,k2,u,i;

  k = (TT-t0)/(N-1);
  t = seqa(t0,k,N);
  x = zeros(N,Ns);

  x[1,.] = x0 .* ones(1,Ns);
  k1 = (mu-0.5*sigma^2)*k;
```

```
  k2 = sigma * sqrt(k);
  u = rndn(N,Ns);

  i = 1;
  do until i > N-1;
    x[i+1,.] = x[i,.] .* exp( k1 + k2 .* u[i+1,.]);
    i = i + 1;
  endo;

  retp(t,x);
endp;

proc (2) = rndOU(x0,a,b,sigma,t0,TT,N,Ns);
  local k,t,x,k1,k2,k3,u,i;

  k = (TT-t0)/(N-1);
  t = seqa(t0,k,N);
  x = zeros(N,Ns);

  x[1,.] = x0 .* ones(1,Ns);
  k1 = exp(-a*k);
  k2 = b .* (1-k1);
  k3 = sigma .* sqrt( (1-exp(-2*a*k))./(2*a) );

  u = rndn(N,Ns);

  i = 1;
  do until i > N-1;
    x[i+1,.] = k1 .* x[i,.] + k2 + k3 .* u[i+1,.];
    i = i + 1;
  endo;

  retp(t,x);
endp;

proc (2) = rndSDE(x0,mu,sigma,t0,TT,N,Ns);
  local mu:proc,sigma:proc;
  local k,t,x,k1,u,i,ti,xi;

  k = (TT-t0)/(N-1);
  t = seqa(t0,k,N);
  x = zeros(N,Ns);

  x[1,.] = x0 .* ones(1,Ns);
  k1 = sqrt(k);

  u = rndn(N,Ns);

  i = 1;
  do until i > N-1;
    ti = t[i];
    xi = x[i,.];
```

```
    x[i+1,.] = xi + k * mu(ti,xi) + k1 * sigma(ti,xi) .* u[i+1,.];
    i = i + 1;
  endo;

  retp(t,x);
endp;
```

36.2 Estimation par maximum de vraisemblance

Soit une série temporelle $\{x_t\}$ avec $t = 1, \ldots, T$. En adoptant un schéma de Taylor, la log-vraisemblance pour l'observation t associée à processus de diffusion homogène s'écrit :

$$\ell_t(x_t;\theta) = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln k - \frac{1}{2}\ln\sigma^2(x_{t-1};\theta) - \frac{1}{2k}\left(\frac{x_t - x_{t-1} - k\mu(x_{t-1};\theta)}{\sigma(x_{t-1};\theta)}\right)^2 \tag{107}$$

avec k l'unité de temps entre deux observations.

La procédure `_sde_logl` calcule le vecteur des log-vraisemblances individuelles.

```
proc _sde_e(mu,theta,x,k);
  local mu:proc;
  local x_;
  x_ = lag1(x);
  retp( x - x_ - k*mu(x_,theta) );
endp;

proc _sde_v(sigma,theta,x,k);
  local sigma:proc;
  retp( k*sigma(lag1(x),theta)^2 );
endp;

proc _sde_logl(mu,sigma,theta,x,k);
  local e,v,logl;

  e = _sde_e(mu,theta,x,k);
  v = _sde_v(sigma,theta,x,k);
  logl = -0.5*ln(2*pi) - 0.5*ln(v) - 0.5*e.*e./v;

  retp(logl);
endp;
```

Voyons un exemple avec un mouvement brownien géométrique.

```
new;
library ritme,pgraph,optmum;

cls;

rndseed 123456;
T = 2;
{t,gbm} = rndGBM(1,0.50,0.20,0,T,365*T+1,1);

graphset;
  _pdate = ''''; _pframe = 0; _pnum = 2;
  xlabel(''t'');
```

```
  ylabel(''GBM process'');
  xtics(0,2,0.5,5);
  xy(t,gbm);

proc mu(x,theta);
  retp( theta[1] * x );
endp;

fn sigma(x,theta) = theta[2]*x;

proc ml(theta);
  retp( -sumc(packr(_sde_logl(&mu,&sigma,theta,gbm,1/365))) );
endp;

let sv = 0.4 0.3;

output file = sde1.out reset;

{theta,logL,grd,retcode} = optmum(&ml,sv);

output off;


================================================================================
   iteration:  1
   algorithm: BFGS         step method: STEPBT
   function: -1986.86171    step length:  0.00000      backsteps:  0
--------------------------------------------------------------------------------
   param.      param. value     relative grad.
       1             0.4000            0.0015
       2             0.3000            0.7155

================================================================================
   iteration:  11
   algorithm: BFGS         step method: STEPBT
   function: -2094.21361    step length:  1.00000      backsteps:  0
--------------------------------------------------------------------------------
   param.      param. value     relative grad.
       1             0.3437            0.0000
       2             0.1935            0.0003
```

36.3 Estimation par la méthode des moments généralisés

En reprenant le schéma de Taylor, nous avons

$$E\left[x_t - x_{t-1} - k\mu\left(x_{t-1};\theta\right)\right] = 0 \tag{108}$$

et

$$E\left[\left(x_t - x_{t-1} - k\mu\left(x_{t-1};\theta\right)\right)^2 - k\sigma^2\left(x_{t-1};\theta\right)\right] = 0 \tag{109}$$

Si le modèle comporte plus de deux paramètres, nous pouvons utiliser les conditions d'orthogonalité suivantes :

$$x_t - x_{t-1} - k\mu\left(x_{t-1};\theta\right) \perp x_{t-1}, x_{t-2}, \ldots \tag{110}$$

et

$$\left(x_t - x_{t-1} - k\mu\left(x_{t-1};\theta\right)\right)^2 - k\sigma^2\left(x_{t-1};\theta\right) \perp x_{t-1}, x_{t-2}, \ldots \tag{111}$$

Reprenons l'exemple précédent. Nous avons :

```
new;
library ritme,pgraph,optmum;

cls;

rndseed 123456;
T = 2;
{t,gbm} = rndGBM(1,0.50,0.20,0,T,365*T+1,1);

graphset;
  _pdate = ''''; _pframe = 0; _pnum = 2;
  xlabel(''t'');
  ylabel(''GBM process'');
  xtics(0,2,0.5,5);
  xy(t,gbm);

proc mu(x,theta);
  retp( theta[1] * x );
endp;

fn sigma(x,theta) = theta[2]*x;

proc H(theta);
  local m1,m2;

  m1 = _sde_e(&mu,theta,gbm,1/365);
  m2 = m1 .* m1 - _sde_v(&sigma,theta,gbm,1/365);

  retp(m1~m2);
endp;

let sv = 1 1;

_gmm_tol = 1e-5;
_gmm_MaxIters = 5;

output file = sde2.out reset;

{theta,stderr,cov,Q} = gmm(&h,sv);

output off;


================================================================================
 GMM - Generalized Method of Moments                    4/07/2002   7:48 pm
================================================================================

Usable observations:                                        730
```

```
Number of parameters:                                   2
Number of moments:                                      2
Degrees of freedom:                                   728

Value of the criterion function:                  0.00000


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------
P01              0.332126       0.139939         2.373356        0.017886
P02             -0.194123       0.005675       -34.207456        0.000000
```

Remarque 23 *Vous trouverez des prolongements (maximum de vraisemblance exact et gmm exact) dans "La Structure Par Terme des Taux Zéro" de* Roncalli *[1998].*

37 Les processus ARCH et GARCH

37.1 Simulation

Dans l'exemple suivant, nous simulons le processus ARCH(1) :

$$\begin{cases} u_t \mid \mathcal{I}_{t-1} \sim \mathcal{N}\left(0, h_t^2\right) \\ h_t^2 = \alpha_0 + \alpha_1 u_{t-1}^2 \end{cases} \tag{112}$$

```
new;
library pgraph;

rndseed 123;

alpha0 = 0.5;
alpha1 = 0.25;
N = 1000;
Ns = 1;

t = seqa(1,1,N);

u = zeros(N,Ns);
h2 = zeros(N,Ns);

i = 1;
do until i > N;
  if i == 1;
    h2[i,.] = alpha0 / (1 - alpha1) .* ones(1,Ns);
  else;
    h2[i,.] = alpha0 + alpha1 * u[i-1,.]^2;
  endif;

  u[i,.] = sqrt(h2[i,.]) .* rndn(1,Ns);

  i = i + 1;
endo;

graphset;
```

```
  _pdate = ''''; _pnum = 2; _pframe = 0;
  xtics(0,1000,100,10);
  xlabel(''\214t'');
  ylabel(''\214h[2]]t['');
  title(''Conditional variance'');
  xy(t,h2);

  ylabel(''u]t['');
  title(''Arch Process'');
  xy(t,u);
```

Dans l'exemple suivant, nous simulons le processus GARCH(1,1) :

$$\begin{cases} u_t \mid \mathcal{I}_{t-1} \sim \mathcal{N}\left(0, h_t^2\right) \\ h_t^2 = \alpha_0 + \alpha_1 u_{t-1}^2 + \beta_1 h_{t-1}^2 \end{cases} \tag{113}$$

```
new;
library pgraph;

rndseed 123;

alpha0 = 0.5;
alpha1 = 0.25;
beta1 = 0.65;

N = 1000;
Ns = 1;

t = seqa(1,1,N);

u = zeros(N,Ns);
h2 = zeros(N,Ns);

i = 1;
do until i > N;
  if i == 1;
    h2[i,.] = alpha0 / (1 - alpha1 - beta1) .* ones(1,Ns);
  else;
    h2[i,.] = alpha0 + alpha1 * u[i-1,.]^2 + beta1 * h2[i-1,.];
  endif;

  u[i,.] = sqrt(h2[i,.]) .* rndn(1,Ns);

  i = i + 1;
endo;

graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0;
  xtics(0,1000,100,10);
  xlabel(''\214t'');
  ylabel(''\214h[2]]t['');
  title(''Conditional variance'');
  xy(t,h2);
```

```
  ylabel(''u]t['');
  title(''Garch Process'');
  xy(t,u);
```

37.2 Estimation par la méthode du maximum de vraisemblance

Les processus GARCH sont des processus gaussiens conditionnels. La log-vraisemblance individuelle s'écrit donc

$$\ell_t = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln h_t^2 - \frac{1}{2}\frac{u_t^2}{h_t^2} \tag{114}$$

```
new;
library ritme,pgraph,optmum;

rndseed 123456;

alpha0 = 0.5;
alpha1 = 0.5;
beta1 = 0.2;

u = zeros(100,1);
h = zeros(100,1);

i = 2;
do until i > 100;
  h[i] = sqrt(alpha0 + alpha1*u[i-1]^2 + beta1*h[i-1]^2);
  u[i] = rndn(1,1)*h[i];
  i=i+1;
endo;

x = seqa(0,0.5,100);
y = 2 + 3*x + u;
data = y~x;

proc negML(theta);
  local nobs,u,h2,i,logl;

  theta[3 4 5] = sqrt(theta[3 4 5]^2);    /* Positivite de la variance */

  nobs = rows(data);
  u = data[.,1] - theta[1] - theta[2]*data[.,2];
  h2 = zeros(nobs,1);
  h2[1] = theta[3];

  i = 2;
  do until i > nobs;
    h2[i] = theta[3] + theta[4]*u[i-1]^2 + theta[5]*h2[i-1];
    i = i + 1;
  endo;

  logl = -0.5*ln(2*pi) - 0.5*ln(h2) - 0.5*u.*u./h2;

  retp( -sumc(logl) );
```

```
endp;

beta = y / (ones(100,1)~x);
sv = beta|1|1|1;

{theta,Logl,grd,retcode} = optmum(&negML,sv);
theta[3 4 5] = sqrt(theta[3 4 5]^2);

nobs = rows(data);
u = data[.,1] - theta[1] - theta[2]*data[.,2];
h2 = zeros(nobs,1);
h2[1] = theta[3];

i = 2;
do until i > nobs;
  h2[i] = theta[3] + theta[4]*u[i-1]^2 + theta[5]*h2[i-1];
  i = i + 1;
endo;

graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0; _plwitdh = 5;
  xtics(0,100,10,10);
  xlabel(''\214t'');
  ylabel(''\214h[2]]t['');
  title(''\214Conditional variance'');
  graphprt(''-c=1 -cf=garch3.ps'');
  xy(seqa(1,1,100),h2);
```

37.3 Estimation par la méthode des moments généralisés

– voir l'exemple dans la section "La méthode des moments généralisés".

37.4 Prolongements

– Arch-in-Mean, Garch-in-Mean
– Site web de Ron Schoenberg
– FanPac

38 Les tests

38.1 Tests des résidus

38.1.1 Test de significativité de la régression (linéaire et non-linéaire)

Sous l'hypothèse que les coefficients de la régression sont nuls, alors

$$\frac{R^2/(K-1)}{(1-R^2)/(N-K)} \sim F_{K-1,N-K} \tag{115}$$

```
proc (2) = FisherTest(N,K,R2);
  local F,pvalue;

  F = ( R2/(K-1) ) / ( (1-R2)/(N-K) );
```

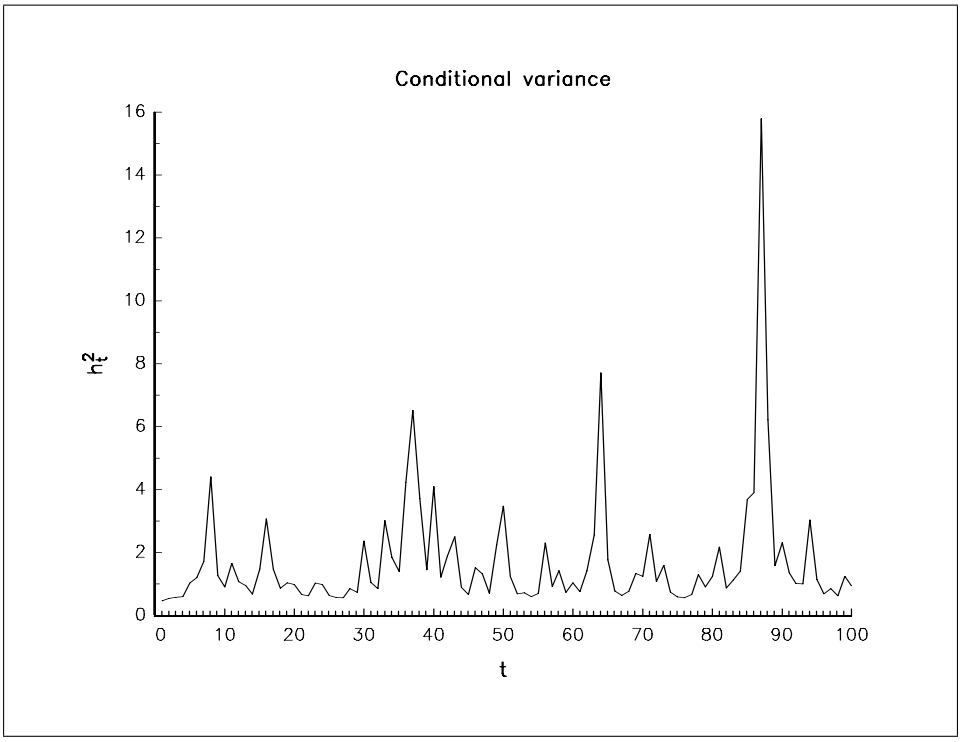


FIG. 10 – Estimation de la variance conditionnelle

```
  pvalue = cdffc(F,K-1,N-K);

  if __output;
    print;
    call header(''FisherTest'','''',0);
    print;
    print ftos(F,     ''F-Test =  %lf'',5,3);
    print ftos(pvalue,''P-value = %lf'',5,3);
  endif;

  retp(F,pvalue);
endp;
```

Reprenons l'exemple de Greene et testons l'hypothèse que la régression n'est pas significative (en dehors de la constante) :

```
new;
library ritme;

load data[15,4] = greene1.asc;

constant = ones(15,1);
trend = seqa(1,1,15);

y = data[.,1];
x = constant~trend~data[.,2 3 4];
```

```
let __altnam = ''Constant'' ''Trend'' ''GNP'' ''Int.rate'' ''Inf.rate'';

{beta,stderr,cov,u} = ols(y,x);

RSS = u'u;
y = y - meanc(y);
TSS = y'y;
R2 = 1 - RSS/TSS;

output file = res1.out reset;

{F,pvalue} = FisherTest(rows(y),cols(x)-1,R2);

output off;
```

```
================================================================================
 FisherTest                                                 4/08/2002  11:15 pm
================================================================================

F-Test =  129.343
P-value = 0.000
```

38.1.2 Test d'autocorrélation des résidus

Il suffit de tester la significativité de la régression portant sur les résidus estimés $\hat{u}_t$:

$$\hat{u}_t = \gamma_1 \hat{u}_{t-1} + \ldots + \gamma_p \hat{u}_{t-p} + \varepsilon_t \tag{116}$$

```
proc (2) = AutoCorrTest(u,p);
  local mv,uLag,beta,e,rss,tss;

  let mv = {.};
  uLag = u .* ones(1,p);
  uLag = shiftr(uLag',seqa(1,1,p),mv)';

  u = trimr(u,p,0);
  uLag = trimr(uLag,p,0);
  beta = u / uLag;
  e = u - uLag*beta;
  rss = e'e;
  tss = u'u;

  retp( FisherTest(rows(u),P,1-rss/tss) );
endp;
```

Voyons un exemple.

```
new;
library ritme;

x = rndu(100,2);
let b = 1 -1;
u = recserar(rndn(100,1)*0.25,0|0,0.5|-0.7);
```

```
y = x*b + u;

{beta,stderr,cov,u} = ols(y,x);

output file = res2.out reset;

data = packr(u~lag1(u)~lagn(u,2));
{beta,stderr,cov,e} = ols(data[.,1],data[.,2 3]);

{F,pvalue} = AutoCorrTest(u,3);

{F,pvalue} = AutoCorrTest(e,3);

output off;
```

```
================================================================================
 OLS                                                      4/09/2002  12:01 am
================================================================================
Number of observations:    98
Number of exogeneous:       2

Degrees of freedom:        96


--------------------------------------------------------------------------------
Parameters     Estimates        Std-err.      T-statistic         P-value
--------------------------------------------------------------------------------
P1              0.567138        0.076411         7.422180        0.000000
P2             -0.662909        0.076405        -8.676205        0.000000


---------------------------
Correlation matrix
---------------------------
       .      P1      P2
      P1   1.000  -0.341
      P2       .   1.000


================================================================================
 FisherTest                                               4/09/2002  12:01 am
================================================================================

F-Test =  48.602
P-value = 0.000

================================================================================
 FisherTest                                               4/09/2002  12:01 am
================================================================================

F-Test =  1.515
```

```
P-value = 0.225
```

38.1.3 Test d'observations abérrantes

Celui-ci est basé sur la *hat matrix* :

$$H = X\left(X^\top X\right)^{-1} X^\top \tag{117}$$

Les éléments diagonaux de la matrice H peuvent être considérés comme des "contributions" à la construction de l'estimateur OLS. Considérons les résidus standardisés

$$\hat{e}_i = \frac{\hat{u}_i}{\hat{\sigma}\sqrt{1 - H_{i,i}}} \tag{118}$$

Pour détecter les points abérrants, nous pouvons comparer ces résidus à un bruit blanc gaussien standard.

```
new;
library ritme,pgraph;

x = rndu(100,2);
let b = 1 -1;
u = rndn(100,1)*0.25;
y = x*b + u;

x[50,.] = 3~2;
y[50] = x[50,.]*(4|-3);

{e,h} = HatMatrixTest(y,x);

t = seqa(1,1,100);

graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0;
  _plctrl = -1; _pstype = 8;
  _pline = 1~3~0~1.96~100~1.96~1~12~10 |
           1~3~0~-1.96~100~-1.96~1~12~10 ;
  ytics(-3,12,3,3);
  xlabel(''\214Observations'');
  ylabel(''\214Standardized residuals'');
  graphprt(''-c=1 -cf=res3.ps'');
  xy(t,e);

graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0;
  _plctrl = -1; _pstype = 11;
  xlabel(''\214Observations'');
  ylabel(''\214h statistic'');
  graphprt(''-c=1 -cf=res4.ps'');
  xy(t,h);
```

38.2 Tests de normalité

Les propriétés asymptotiques des estimateurs sont généralement obtenues sous l'hypothèse de normalité. Nous pouvons donc procéder à l'inférence statistique seulement si cette hypothèse est vérifiée. C'est pour cette raison qu'il est important de tester la normalité des variables (exogènes et endogène) ou des résidus.

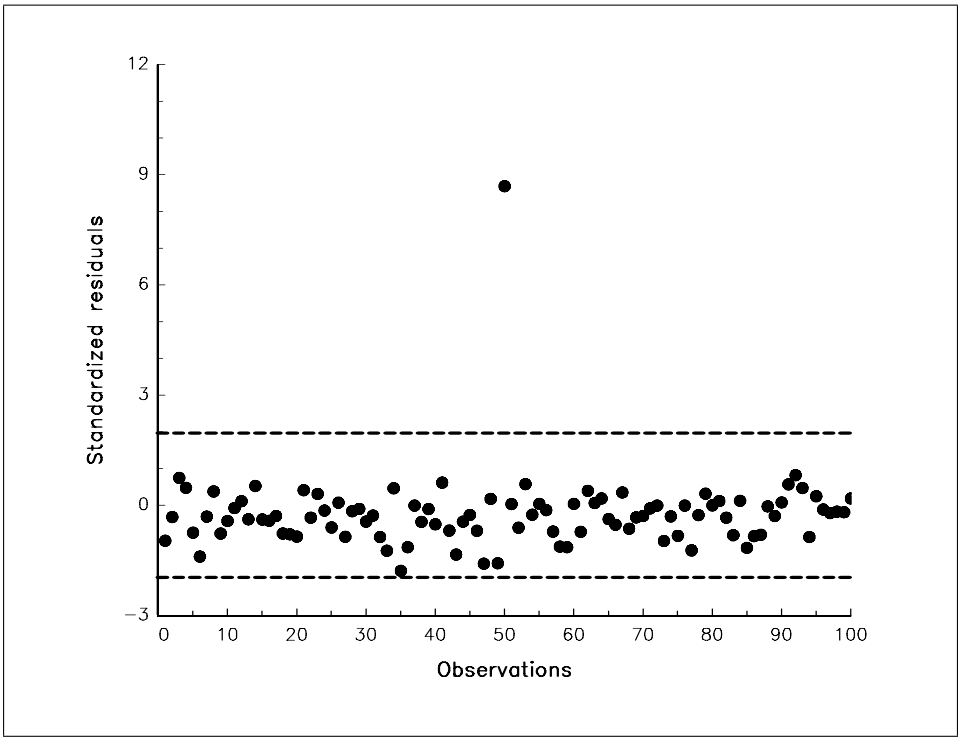


FIG. 11 – Test d'observations abérrantes

Soit une variable aléatoire X. Considérons alors l'hypothèse nulle $H_0 : X \sim \mathcal{N}\left(m, \sigma^2\right)$. Soient $\{x_1, \ldots, x_T\}$ les valeurs prises par la variable aléatoire X. Nous définissons le skewness et la kurtosis empiriques de X par

$$SK = \frac{m_3}{s^3} \qquad \text{et} \qquad KT = \frac{m_4}{s^4} \tag{119}$$

avec

$$s^2 = \frac{1}{T-1} \sum_{t=1}^{T} \left(x_t - \bar{x}\right)^2 \tag{120}$$

et

$$m_i = \frac{1}{T} \sum_{t=1}^{T} \left(x_t - \bar{x}\right)^i \tag{121}$$

Soient $u_t = \left(x_t - \bar{x}\right)/s$ les variables centrées et réduites. Tester l'hypothèse H_0 revient à tester que le skewness de u est nul et que la kurtosis de u vaut 3. Sous H_0, nous avons

$$\mathbf{t}_{SK} = \frac{1}{\sqrt{6T}} \sum_{t=1}^{T} u_t^3 \underset{as}{\sim} \mathcal{N}(0,1) \tag{122}$$

et

$$\mathbf{t}_{KT} = \frac{1}{\sqrt{24T}} \sum_{t=1}^{T} \left(u_t^4 - 3\right) \underset{as}{\sim} \mathcal{N}(0,1) \tag{123}$$

Jarque et Bera montrent que sous l'hypothèse nulle H_0, la statistique **JB** (ci-dessous) converge en loi vers la loi chi-deux à 2 degrés de liberté. Nous avons

$$\mathbf{JB} = \frac{T}{6} SK^2 + \frac{T}{24} \left(KT - 3\right)^2 \underset{as}{\sim} \chi_2^2 \tag{124}$$

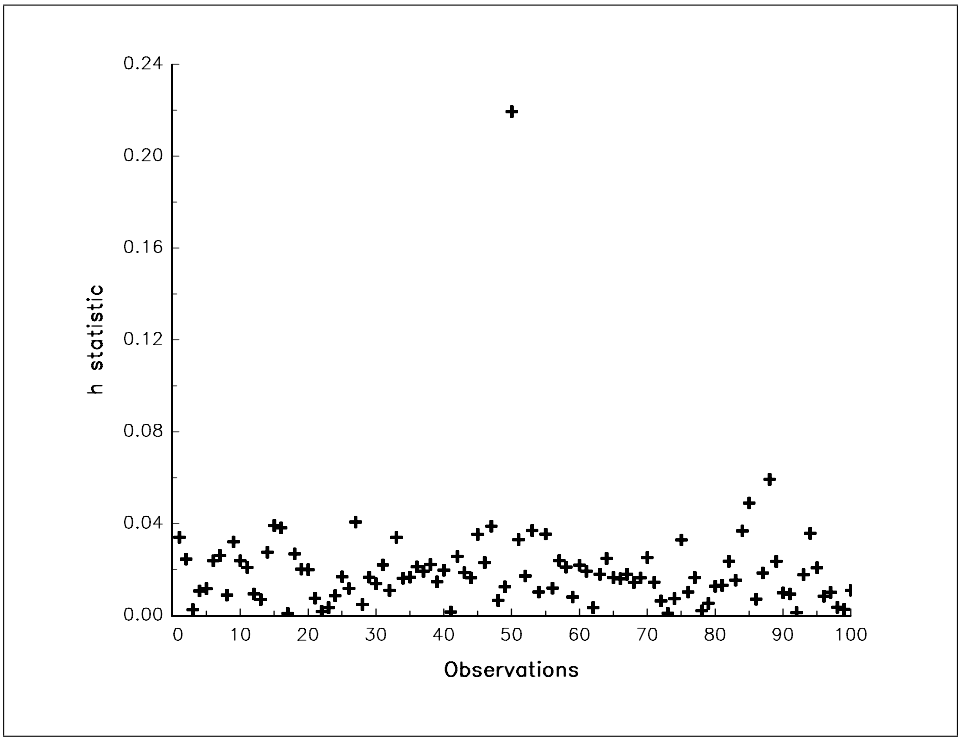


FIG. 12 – Statistiques $H_{i,i}$

```
proc (4) = NormalityTest(x);
  local v,pvalue,n,m,s,y,m3,m4,SK,KT;

  v = zeros(3,1);
  pvalue = zeros(3,1);

  x = packr(x);
  n = rows(x);
  m = meanc(x);
  s = stdc(x);

  y = (x-m)/s;

  /* Skewness test */

  v[1] = sumc(y^3) / sqrt(6*n);
  pvalue[1] = 2 * cdfnc(abs(v[1]));

  /* Kurtosis test */

  v[2] = sumc(y^4-3) / sqrt(24*n);
  pvalue[2] = 2 * cdfnc(abs(v[2]));

  /* Jarque-Bera test */
```

```
  m3 = meanc((x-m)^3);
  m4 = meanc((x-m)^4);
  SK = m3 / (s^3);
  KT = m4 / (s^4);

  v[3] = (n/6)*(SK^2) + (n/24)*((KT-3)^2);
  pvalue[3] = cdfchic(v[3],2);

  if __output == 1;
    print;
    call header(''Normality test'','''',0);
    print ftos(SK,  ''Skewness:         %lf'',6,3);
    print ftos(v[1],''Skewness test:    %lf'',6,3);
    print ftos(pvalue[1],''P-value:          %lf'',6,3);
    print;
    print ftos(KT,  ''Kurtosis:         %lf'',6,3);
    print ftos(v[2],''Kurtosis test:    %lf'',6,3);
    print ftos(pvalue[2],''P-value:          %lf'',6,3);
    print;
    print ftos(v[3],''Jarque-Bera test: %lf'',6,3);
    print ftos(pvalue[3],''P-value:          %lf'',6,3);
    print;
  endif;

  retp(SK,KT,v,pvalue);
endp;
```

– Un exemple avec des résidus gaussiens...

```
new;
library ritme,pgraph;

Nobs = 5000;
x = rndu(Nobs,2);
let b = 1 -1;
u = rndn(Nobs,1)*0.25;
y = x*b + u;

beta = y / x;
residuals = y - x*beta;

output file = norm1.out reset;

{SK,KT,test,pvalue} = NormalityTest(residuals);

output off;
```

```
================================================================================
 Normality test                                              4/09/2002   1:03 am
================================================================================
Skewness:          0.004
Skewness test:     0.128
```

```
P-value:           0.898

Kurtosis:          2.976
Kurtosis test:    -0.341
P-value:           0.733

Jarque-Bera test:  0.133
P-value:           0.936
```

– Un exemple avec des résidus de type Student...

```
new;
library ritme,pgraph;

Nobs = 5000;
x = rndu(Nobs,2);
let b = 1 -1;
u = 0.25 * rndn(Nobs,1)/sqrt(rndn(Nobs,1)^2);

y = x*b + u;

beta = y / x;
residuals = y - x*beta;

output file = norm2.out reset;

{SK,KT,test,pvalue} = NormalityTest(residuals);

output off;
```

```
================================================================================
 Normality test                                                 4/09/2002   1:05 am
================================================================================
Skewness:         -0.007
Skewness test:    -0.190
P-value:           0.850

Kurtosis:          2.405
Kurtosis test:    -8.590
P-value:           0.000

Jarque-Bera test: 73.819
P-value:           0.000
```

38.3 Tests d'hypothèses sur les coefficients

Considérons l'hypothèse nulle $\mathrm{H}_0 : R(\theta) = 0$. Soient $I(\theta)$ la matrice d'information, T le nombre d'observations et $\hat{\theta}_{\mathrm{MV}}$ l'estimateur du maximum de vraisemblance. Sous H_0, nous pouvons montrer que

$$\mathbf{W} = TR\left(\hat{\theta}_{\mathrm{MV}}\right)^{\top}\left[\frac{\partial R}{\partial \theta^{\top}}\left(\hat{\theta}_{\mathrm{MV}}\right) I\left(\hat{\theta}_{\mathrm{MV}}\right)^{-1}\frac{\partial R^{\top}}{\partial \theta}\left(\hat{\theta}_{\mathrm{MV}}\right)\right]^{-1} R\left(\hat{\theta}_{\mathrm{MV}}\right) \underset{as}{\sim} \chi^2_g \tag{125}$$

avec g le nombre de restrictions. Ce test de Wald peut être étendu à d'autres estimateurs. On parle alors de tests basés sur le principe de Wald. Soit $\hat{\theta}$ un estimateur de θ, nous avons sous l'hypothèse nulle

$$\mathbf{W} = R\left(\hat{\theta}\right)^{\top}\left[\frac{\partial R}{\partial \theta^{\top}}\left(\hat{\theta}\right)\operatorname{var}\left[\hat{\theta}\right]\frac{\partial R^{\top}}{\partial \theta}\left(\hat{\theta}\right)\right]^{-1} R\left(\hat{\theta}\right) \underset{as}{\sim} \chi_g^2$$

où $\operatorname{var}\left[\hat{\theta}\right]$ est la matrice de covariance estimée. Sous cette forme, nous pouvons appliquer le test avec les estimateurs du maximum de vraisemblance, mais aussi avec ceux des moments généralisés (GMM), des moindres carrés non linéaires, etc.

Dans le cas d'hypothèses linéaires, nous avons $R\theta = r$. C'est donc un cas particulier de l'analyse précédente, puisque nous avons

$$R(\theta) = R\theta - r = 0 \tag{126}$$

Nous en déduisons que

$$\mathbf{W} = (R\theta - r)^{\top}\left[R\operatorname{var}\left[\hat{\theta}\right]R^{\top}\right]^{-1}(R\theta - r) \tag{127}$$

```
proc (2) = WaldTest(R,theta,covtheta);
  local R:proc;
  local R0,DR,g,W,pvalue;

  R0 = R(theta);
  g = rows(r0);

  DR = gradp(&R,theta);
  w = R0'invpd(DR*covtheta*DR')*R0;
  pvalue = cdfchic(W,g);

  if __output;
    call header(''Wald Test'','''',0);
    print ftos(W,     ''Wald test: %lf'',6,4);
    print ftos(pvalue,''P-value:   %lf'',6,4);
  endif;

  retp(W,pvalue);
endp;
```

Exemple 6 *Simulons le modèle*

$$\begin{cases} y_t = 2 + 3t + u_t & t = 1, \ldots, 1000 \\ u_t \sim \mathcal{N}(0,1) \end{cases} \tag{128}$$

Soit le modèle économétrique correspondant

$$y_t = \beta_1 + \beta_2 t + u_t \tag{129}$$

nous allons tester les hypothèses

$$\begin{cases} \beta_1 = 2 \\ \beta_2 = 3 \end{cases} \tag{130}$$

et

$$\beta_1 \ln(\beta_1) = 3\ln(2) \tag{131}$$

```
new;
library gauss,ritme;

rndseed 124;

x = seqa(1,1,1000);
y = 2 + 3 * x + rndn(1000,1);

proc Wald1(theta);
  local R;
  R = theta[1] - 2 |
      theta[2] - 3 ;
  retp(R);
endp;

proc Wald2(theta);
  local R;
  R = theta[2]*ln(theta[1]) - 3*ln(2);
  retp(R);
endp;

__output = 0;
{vnam,m,b,stb,vc,sdterr,sigma,cx,rsq,resid,dwstat} = ols(0,y,x);

output file = wald.out reset;

__output = 1;
call WaldTest(&Wald1,b,vc);
{w2,pvalue} = WaldTest(&Wald2,b,vc);

output off;


================================================================================
 Wald Test                                                  4/09/2002   1:38 am
================================================================================
Wald test: 1.1696
P-value:   0.5572

================================================================================
 Wald Test                                                  4/09/2002   1:38 am
================================================================================
Wald test: 0.3306
P-value:   0.5653
```

39 Intégration et cointégration

39.1 Tests de racine unité

39.1.1 Test KPSS

```
/*
```

```
** kpss.src - Time Series Modelling.
** (C) Copyright 1996 by Thierry Roncalli.
** All Rights Reserved.
**
**  Format                        Purpose                                  Line
** ============================================================================
**  {nu,tau} = kpss(x,order)      KPSS test                                 13
*/


/*
** kpss
**
** Purpose: Compute the KPSS nu and tau statistics
**
**          Kwiatkowski, Phillips, Schmidt and Shin [1992],
**          Testing the null hypothesis of stationarity against
**          the alternative of a unit root: how sure are we that
**          economic series have a unit root,
**          Journal of Econometrics, 54, 159-178
**
** Format:  {nu,tau} = kpss(x,order);
**
** Input:     x - Nobs*1 vector, data.
**        order - scalar, maximum lag tested.
**
** Output:   nu - (order+1)*1 vector, the NU statistic values.
**          tau - (order+1)*1 vector, the TAU statistic values.
**
*/

proc (2) = KPSS(x,order);
  local T,const,trend,x1,x2,beta1,beta2,u1,u2;
  local s1,s2,nw1,nw2,nu_test,tau_test,i,fmt;

  x = packr(x); T = rows(x);
  const = ones(T,1); trend = seqa(1,1,T);
  x1 = const;          x2 = const~trend;
  beta1 = x/x1;        beta2 = x/x2;

  u1 = x-x1*beta1;     u2 = x-x2*beta2;
  s1 = cumsumc(u1);    s2 = cumsumc(u2);
  s1 = sumc(s1^2);     s2 = sumc(s2^2);

  nw1 = zeros(order+1,1);
  nw2 = zeros(order+1,1);

  i = 0;
  do until i>order;
    nw1[i+1] = _nw_sigma2(u1,i);
    nw2[i+1] = _nw_sigma2(u2,i);
    i = i+1;
```

```
  endo;

  nu_test = s1./nw1; nu_test = nu_test/(T^2);
  tau_test = s2./nw2; tau_test = tau_test/(T^2);
  i = seqa(0,1,order+1);

  if __output;
    call header(''Kwiatkowsky, Phillips, Schmidt and Shin Test'','''',0);
    print;
    print ''      LAGS      NU-TEST       TAU-TEST'';
    print ''-----------------------------------------------'';
    let fmt[3,3] =
              ''*.*lf'' 8 0
              ''*.*lf'' 15 6
              ''*.*lf'' 15 6;
    call printfm(i~nu_test~tau_test,1~1~1,fmt);
    print;
    print ''-----------------------------------------------'';
    print ''             Critical values          '';
    print ''-----------------------------------------------'';
    print ''           10%     5%      1%         '';
    print ''-----------------------------------------------'';
    print ''NU_test   0.347   0.463   0.739       '';
    print ''TAU_test  0.119   0.146   0.216       '';
    print ''-----------------------------------------------'';
  endif;

  retp(nu_test,tau_test);
endp;

/*
** _nw_sigma2
**
** Purpose: Compute the Newey-West estimator of the variance of a process.
**
** Format:  sigma2 = _nw_sigma2(x,l);
**
** Input:         x - Nobs*1 vector, data.
**                l - scalar, the lag.
**
** Output:   sigma2 - scalar, the NW estimate.
**
*/

proc _nw_sigma2(u,l);
  local T,s,w,e0,e,sig2,ulag;

  u = packr(u); u = u-meanc(u);
  T = rows(u);
  if l == 0;

    e0 = u^2;
```

```
    sig2 = sumc(e0)/T;

  else;

    s = seqa(1,1,l);
    w = 1-s/(l+1);
    e0 = u^2;
    ulag = shiftr(u'.*ones(l,T),seqa(1,1,l),0)';
    e = sumc(u.*ulag);
    sig2 = sumc(e0)+2*sumc(w.*e); sig2 = sig2/T;

  endif;

  retp(sig2);
endp;
```

– Voici un exemple d'utilisation de la procédure KPSS.

```
new;
library ritme;

y = recserar(rndn(300,1),5,1);   /* Unit root process */

output file = kpss.out reset;

{test1,test2} = KPSS(y,8);

y = y - lag1(y);
{test1,test2} = KPSS(y,8);

output off;
```

```
================================================================================
 Kwiatkowsky, Phillips, Schmidt and Shin Test              4/09/2002  10:46 pm
================================================================================

      LAGS      NU-TEST       TAU-TEST
------------------------------------------
       0      21.075571       3.993897
       1      10.682032       2.059345
       2       7.197598       1.405871
       3       5.452219       1.077915
       4       4.401683       0.879728
       5       3.698791       0.746525
       6       3.194955       0.650649
       7       2.815854       0.578154
       8       2.520051       0.521277

------------------------------------------
             Critical values
------------------------------------------
           10%     5%      1%
```

```
-----------------------------------------
NU_test   0.347   0.463   0.739
TAU_test  0.119   0.146   0.216
-----------------------------------------


=============================================================================
 Kwiatkowsky, Phillips, Schmidt and Shin Test              4/09/2002  10:46 pm
=============================================================================

      LAGS      NU-TEST       TAU-TEST
-----------------------------------------
       0       0.223667       0.018800
       1       0.231896       0.019650
       2       0.240347       0.020551
       3       0.270237       0.023396
       4       0.301397       0.026474
       5       0.330058       0.029457
       6       0.357743       0.032485
       7       0.386307       0.035752
       8       0.415215       0.039231

-----------------------------------------
             Critical values
-----------------------------------------
           10%     5%      1%
-----------------------------------------
NU_test   0.347   0.463   0.739
TAU_test  0.119   0.146   0.216
-----------------------------------------
```

39.1.2 Test ADF

```
           /* Unit Root Tests (Isaac and Rapach,May96) */

/*************************************************************************
**************************************************************************
UNIT ROOT TESTING: a collection of procedures
Authors: Alan G. Isaac and David Rapach
==> For public, non-commercial use only
==> Please include proper attribution should this code be used in
    a research project or in other code
==> Authors make no performance guarantees
==> Program written for GAUSS Version 3.1.13
    (vendor: Aptech Systems, Maple Valley, WA)
**************************************************************************
*************************************************************************/

/* Procedure Definition Area */

/*************************************************************************
Proc            : ADF
```

```
Authors         : Alan G. Isaac and David Rapach
Last revised    : 27 May 1996
Input           : y -- (N x 1) vector, the time series of interest
                  p -- scalar, maximum lag (max autocorr order)
Output          : Prints augmented Dickey-Fuller t-stats
Globals         : None
Notes           : Requires procs 'varlags()' and 'adf_ls()'(provided)
References      : Russell Davidson and James G. MacKinnon,
                       _Estimation and Inference in Econometrics_
                       (New York: Oxford, 1993)
                       See references therein for original sources
                       See p.708, table 20.1 for critical values
                  Harris, 1992, Economics Letters
*************************************************************************/
proc(0)=adf(y,p);
    local obs,i,b,ty,temp,oldcv,oldnv;
    obs=rows(y);
    format 8,4;
    ''One-sided test of H0: Unit root vs. H1: Stationary'';?;
    ''Approximate asymptotic critical values (t-ratio):'';
    ''-----------------------------------------------------------------'';
    ''  1%      5%      10%      Model'';
    ''-----------------------------------------------------------------'';
    ''-2.56   -1.94   -1.62'';;
    ''     Simple ADF (no constant or trend)'';
    ''-3.43   -2.86   -2.57'';;
    ''     ADF with constant (no trend)'';
    ''-3.96   -3.41   -3.13'';;
    ''     ADF with constant & trend'';
    ''-----------------------------------------------------------------'';
''You began with a series of ''$+ftocv(obs,1,0)$+'' observations, in which case'';
''Harris (1992) recommends using ''$+ftocv(int(12*(obs/100)^(1/4)),1,0)$+'' lags'';
''
-------------------------------------------------------------
                    ADF coef on lagged level
                            (t-ratio)
                For model including the following:
                ----------------------------------
         Lags   No Constant  Constant    Trend           Obs
                No Trend     No Trend    Constant
-------------------------------------------------------------'';
    i=0;
        oldcv=__fmtcv;oldnv=__fmtnv;
    do while i<=p;
        {b,ty,obs}=adf_ls(y,i,''off'');
        __fmtcv=''*.*s''~12~8;__fmtnv=''*.*lg''~12~3;
        temp=printfmt(i~b[1,.]~obs,1);print;
        temp=printfmt(''  ''~(0$+''(''$+ftocv(ty[1,.],1,2)$+'')'')~''  '',0);print;
        i=i+1;
    endo;
''-------------------------------------------------------------'';
        __fmtcv=oldcv;__fmtnv=oldnv;
```

```
retp;
endp;

/*************************************************************************
Proc            : ADF_LS
Authors         : Alan G. Isaac and David Rapach
Last revised    : 27 May 1996
Input           : y   -- (N x 1) vector, the time series of interest
                  p   -- scalar, maximum lag (max autocorr order)
                  prt -- string, prt=''on'' to print results
Output          : Prints augmented Dickey-Fuller t-stats
                  b   -- coefs from ADF regression
                  t   -- t-stats from ADF regression
                  obs -- scalar, rows(y)-(p+1)
Globals         : None
Notes           : Requires procs 'varlags()' (provided)
References      : Russell Davidson and James G. MacKinnon,
                      _Estimation and Inference in Econometrics_
                      (New York: Oxford, 1993)
                      See references therein for original sources
*************************************************************************/
proc(3)=adf_ls(y,p,prt);
    local labelx,addx,y_1,dy,dylags,temp,report,rhs,gap,oldcv,oldnv,
          T,b,e,SSE,sig2,covb,seb,ty,i;
    T=rows(y);                      @ number of observations @
    labelx=''x(-1)'';
    addx={};
    {y,y_1}=varlags(y,1);           @ generating lags @
    dy=y-y_1;                       @ differencing @
    if p /= 0;
      {dy,dylags}=varlags(dy,p);    @ generating lags @
      addx=addx~dylags;
      labelx=labelx|(0$+''dx(-''$+ftocv(seqa(1,1,p),1,0)$+'')'');
    endif;
    labelx=labelx|''constant''|''trend''|''RSS'';
    T=rows(dy);                     @ # of usable obs @
    y_1=y_1[p+1:p+T];               @ usable lagged y @
    addx=y_1~addx;
    gosub regress(addx,miss(zeros(2,1),0));
      pop temp;
      report=temp;
    addx=addx~ones(T,1);            @ add constant to regressors @
    gosub regress(addx,miss(0,0)); @ add constant to regressors @
      pop temp;
      report=report~temp;
    addx=addx~seqa(-int(T/2),1,T); @ add trend to regressors @
    temp={};
    gosub regress(addx,temp);
      pop temp;
      report=report~temp;
    if prt $== ''on'';
      print;
```

```
      ''ADF results for ''$+ftocv(p,1,0)$+'' lags'';
''-------------------------------------------------------------
                     ADF regression coefficients
                            (t-ratio)
                For model including the following:
                ---------------------------------
                No Constant  Constant    Trend
     RHS var    No Trend     No Trend    Constant
-------------------------------------------------''';
    i=1;
    oldcv=__fmtcv;oldnv=__fmtnv;
    __fmtcv=''*.*s''~12~8;__fmtnv=''*.*lg''~12~3;
    temp=printfmt(labelx[1]~report[1,1|3|5],0~1~1~1);print;
    do while i<=p+3;
      temp=printfmt(''  ''~(0$+''(''$+ftocv(report[i,2|4|6],1,2)$+'')'')~''  '',0);
      print;
      i=i+1;
      temp=printfmt(labelx[i]~report[i,1|3|5],0~1~1~1);print;
    endo;
''-------------------------------------------------------------''';
''Note: Final line is sum of squared residuals'';
/* Could use these for Dickey Fuller (1981) Phi tests */
       __fmtcv=oldcv;__fmtnv=oldnv;
    endif;
retp(report[.,1|3|5],report[.,2|4|6],T); @ coef, t-stat, usable obs @
regress:
    pop gap;
    pop rhs;
    b=dy/rhs;                    @ OLS estimates @
    e=dy-rhs*b;                  @ resids @
    SSE=e'e;                     @ sum of squared resids @
    sig2=SSE/(T-cols(rhs));      @ e2 estimate @
    covb=sig2*invpd(rhs'rhs);    @ var-cov(b) @
    seb=sqrt(diag(covb));        @ std errs @
    return((b|gap|SSE)~((b./seb)|gap|miss(0,0)));
endp;

/*********************  PROC VARLAGS  ****************************
**   author: Alan G. Isaac
**   last update: 5 Dec 95      previous: 15 June 94
**   FORMAT
**          { x,xlags } = varlags(var,lags)
**   INPUT
**        var  - T x K matrix
**        lags - scalar, number of lags of var (a positive integer)
**   OUTPUT
**          x -     (T - lags) x K matrix, the last T-lags rows of var
**          xlags - (T - lags) x lags*cols(var) matrix,
**                  being the 1st through lags-th
**                  values of var corresponding to the values in x
**                  i.e, the appropriate rows of x(-1)~x(-2)~etc.
**   GLOBAL VARIABLES: none
```

```
*************************************************************************/
proc(2)=varlags(var,lags);
    local xlags;
    xlags = shiftr((ones(1,lags) .*. var)',seqa(1-lags,1,lags)
                                          .*. ones(cols(var),1),miss(0,0))';
    retp(trimr(var,lags,0),trimr(xlags,0,lags));
endp;

/*************************************************************************
Proc            : KPSS
Author          : David Rapach
Last revised    : 27 May 1996
Input           : y -- (T x 1) vector of obs on variable of interest
                : p -- specified max order of autocorrelation
Output          : Prints KPSS statistics
Globals         : None
Reference       : Denis Kwiatowski, Peter C.B. Phillips, Peter Schmidt,
                      and Tongcheol Shin, ''Testing the null hypothesis of
                      stationarity against the alternative of a unit root:
                      How sure are we that economic series have a unit
                      root?'' _Journal of Econometrics_ 54 (Oct./Dec.
                      1992): 159-178
Notes           : KPSS(trend) based on y(t)=a+bt+e(t) -- trend stationarity
                  KPSS(level) based on y(t)=a+e(t)    -- level stationarity
*************************************************************************/
proc(0)=kpss(y,l);
    local T,trend,X,b,e,g0,p,g,i,w,s2,S,ei,etathat,etamhat,lag,
          eta,oldnv;

    /* Level-stationary test */

    T=rows(y);              @ # of obs @
    X=ones(T,1);            @ RHS var @
    b=y/X;                  @ OLS estimates @
    e=y-X*b;                @ OLS resids @
    g0=e'e/T;               @ gamma_0=(1/T)sum[e(t)^2] @
    etamhat=zeros(l+1,1);
    p=1;
    do until p>l;
        g=zeros(p,1);       @ autocovs @
        i=1;
        do until i>p;
            g[i,1]=(e[1+i:T,1]'e[1:T-i,1])/T;@ gamma_i=(1/T)sum[e(t)e(t-i)] @
            i=i+1;
        endo;
        w=zeros(p,1);
        i=1;
        do until i>p;
            w[i,1]=(p+1-i)/(p+1);   @ Bartlett window weight @
            i=i+1;
        endo;
        s2=g0+2*w'g;                @ consistent error variance estimate @
```

```
    S=zeros(T,1);               @ resid partial sum process @
    i=1;
    do until i>T;
        ei=e[1:i,.];
        S[i,.]=sumc(ei);        @ S(i)=sum[e(i)] @
        i=i+1;
    endo;
    etamhat[1,.]=(1/(g0*T^2))*S'S;      @ KPSS eqn (13), l=0 @
    etamhat[p+1,.]=(1/(s2*T^2))*S'S;    @ KPSS eqn (13) @
    p=p+1;
endo;

/* Trend-stationary test */

trend=seqa(1,1,T);      @ linear time trend @
X=ones(T,1)~trend;      @ regressor matrix @
b=y/X;                  @ OLS estimates @
e=y-X*b;                @ OLS resids @
g0=e'e/T;               @ gamma_0=(1/T)sum[e(t)^2] @
etathat=zeros(l+1,1);
p=1;
do until p>l;
    g=zeros(p,1);       @ autocovs @
    i=1;
    do until i>p;
        g[i,1]=(e[1+i:T,1]'e[1:T-i,1])/T;@ gamma_i=(1/T)sum[e(t)e(t-i)] @
        i=i+1;
    endo;
    w=zeros(p,1);
    i=1;
    do until i>p;
        w[i,1]=(p+1-i)/(p+1);   @ Bartlett window weight @
        i=i+1;
    endo;
    s2=g0+2*w'g;                @ consistent error variance estimate @
    S=zeros(T,1);               @ resid partial sum process @
    i=1;
    do until i>T;
        ei=e[1:i,.];
        S[i,.]=sumc(ei);        @ S(i)=sum[e(i)] @
        i=i+1;
    endo;
    etathat[1,.]=(1/(g0*T^2))*S'S;      @ KPSS eqn (17), l=0 @
    etathat[p+1,.]=(1/(s2*T^2))*S'S;    @ KPSS eqn (17) @
    p=p+1;
endo;
format 8,4;
''One-sided test of H0: Stationary vs. H1: Unit root'';?;
''Approximate asymptotic critical values'';
''-------------------------------------------------'';
''              10%     5%      1%'';
''-------------------------------------------------'';
```

```
    ''Level        0.347   0.463   0.739'';
    ''Trend        0.119   0.146   0.216'';
    ''-------------------------------------------------''';?;
    lag=seqa(0,1,l+1);
    eta=etamhat~etathat;
    oldnv=__fmtnv;
    __fmtnv=''*.*lg''~8~3;
    ''KPSS stats'';
    ''-------------------------------------------------'';
    ''      Lag     Level    Trend'';
    ''-------------------------------------------------'';
    i=1;
    do until i>l+1;
        call printfmt(lag[i,.],1);''    '' eta[i,.];
        i=i+1;
    endo;
    ''-------------------------------------------------'';
    ''Obs'';;call printfmt(T,1);
    __fmtnv=oldnv;
endp;


proc(1)=adf2(y,p);
    local obs,i,b,ty,temp,oldcv,oldnv;
    local tau;

    tau = zeros(p+1,3);

    obs=rows(y);
    format 8,4;
    ''One-sided test of H0: Unit root vs. H1: Stationary'';?;
    ''Approximate asymptotic critical values (t-ratio):'';
    ''-------------------------------------------------------------''';
    ''  1%      5%      10%      Model'';
    ''-------------------------------------------------------------'';
    ''-2.56   -1.94   -1.62'';;
    ''     Simple ADF (no constant or trend)'';
    ''-3.43   -2.86   -2.57'';;
    ''     ADF with constant (no trend)'';
    ''-3.96   -3.41   -3.13'';;
    ''     ADF with constant & trend'';
    ''-------------------------------------------------------------''';
''You began with a series of ''$+ftocv(obs,1,0)$+'' observations, in which case'';
''Harris (1992) recommends using ''$+ftocv(int(12*(obs/100)^(1/4)),1,0)$+'' lags'';
''
-------------------------------------------------------------
                    ADF coef on lagged level
                            (t-ratio)
                For model including the following:
                ----------------------------------
          Lags  No Constant  Constant    Trend          Obs
                No Trend     No Trend    Constant
```

```
------------------------------------------------------------’’;
    i=0;
        oldcv=__fmtcv;oldnv=__fmtnv;
    do while i<=p;
        {b,ty,obs}=adf_ls(y,i,’’off’’);
        __fmtcv=’’*.*s’’~12~8;__fmtnv=’’*.*lg’’~12~3;
        temp=printfmt(i~b[1,.]~obs,1);print;
        temp=printfmt(’’  ’’~(0$+’’(’’$+ftocv(ty[1,.],1,2)$+’’)’’)~’’  ’’,0);print;
        tau[i+1,.] = ty[1,.];
        i=i+1;
    endo;
’’------------------------------------------------------------’’;
        __fmtcv=oldcv;__fmtnv=oldnv;
retp(tau);
endp;
```

– Voici un exemple d’utilisation de la procédure ADF.

```
new;
library ritme;

cls;

y = recserar(rndn(300,1),5,1);   /* Unit root process */

output file = unit.out reset;

call adf(y,8);

y = packr(y - lag1(y));
call adf(y,8);

output off;


One-sided test of H0: Unit root vs. H1: Stationary

Approximate asymptotic critical values (t-ratio):
------------------------------------------------------------
  1%      5%      10%      Model
------------------------------------------------------------
-2.56   -1.94   -1.62     Simple ADF (no constant or trend)
-3.43   -2.86   -2.57     ADF with constant (no trend)
-3.96   -3.41   -3.13     ADF with constant & trend
------------------------------------------------------------


You began with a series of 300 observations, in which case
Harris (1992) recommends using 15 lags

------------------------------------------------------------
                    ADF coef on lagged level
                           (t-ratio)
```

```
                For model including the following:
                ----------------------------------
         Lags   No Constant  Constant    Trend           Obs
                No Trend     No Trend    Constant
-------------------------------------------------------------
           0    -0.00237    -0.00566     -0.0119         299
                 (-0.39)     (-0.86)     (-1.33)
           1    -0.00182    -0.00514     -0.0113         298
                 (-0.29)     (-0.77)     (-1.26)
           2    -0.00174     -0.0048     -0.0117         297
                 (-0.28)     (-0.72)     (-1.30)
           3    -0.00255    -0.00563     -0.0123         296
                 (-0.41)     (-0.84)     (-1.37)
           4    -0.00211    -0.00498     -0.0124         295
                 (-0.34)     (-0.74)     (-1.37)
           5    -0.00267    -0.00511     -0.0139         294
                 (-0.43)     (-0.76)     (-1.54)
           6    -0.00189    -0.00421     -0.0136         293
                 (-0.30)     (-0.63)     (-1.51)
           7      -0.002    -0.00435     -0.0136         292
                 (-0.32)     (-0.64)     (-1.50)
           8    -0.00209    -0.00436      -0.014         291
                 (-0.33)     (-0.64)     (-1.54)
-------------------------------------------------------------
One-sided test of H0: Unit root vs. H1: Stationary

Approximate asymptotic critical values (t-ratio):
-------------------------------------------------------------
  1%      5%      10%      Model
-------------------------------------------------------------
-2.56   -1.94   -1.62     Simple ADF (no constant or trend)
-3.43   -2.86   -2.57     ADF with constant (no trend)
-3.96   -3.41   -3.13     ADF with constant & trend
-------------------------------------------------------------
You began with a series of 299 observations, in which case
Harris (1992) recommends using 15 lags

-------------------------------------------------------------
                    ADF coef on lagged level
                             (t-ratio)
                For model including the following:
                ----------------------------------
         Lags   No Constant  Constant    Trend           Obs
                No Trend     No Trend    Constant
-------------------------------------------------------------
           0       -1.05       -1.06       -1.06         298
                (-18.20)    (-18.25)    (-18.22)
           1       -1.04       -1.05       -1.05         297
                (-12.33)    (-12.37)    (-12.36)
           2       -0.96      -0.972      -0.973         296
                 (-9.25)     (-9.31)     (-9.30)
           3      -0.984      -0.999          -1         295
```

```
              (-8.34)     (-8.39)     (-8.39)
          4    -0.914      -0.929      -0.933         294
              (-7.00)     (-7.03)     (-7.05)
          5    -0.987          -1       -1.01         293
              (-6.99)     (-7.02)     (-7.05)
          6    -0.976      -0.995          -1         292
              (-6.37)     (-6.41)     (-6.44)
          7     -0.97      -0.991          -1         291
              (-5.91)     (-5.95)     (-5.99)
          8    -0.945      -0.968      -0.981         290
              (-5.41)     (-5.45)     (-5.49)
------------------------------------------------------------
```

39.2 Calcul des valeurs critiques du test ADF par Monte Carlo

```
new;
library ritme;


N = 1000;
Ns = 1000;
sigma = 0.25;

y0 = zeros(1,Ns);
phi = ones(1,Ns);

y = recserar(sigma*rndn(N+1,Ns),y0,phi);

yy = trimr(lag1(y),1,0);
y = trimr(y,1,0);

rho = sumc(y .* yy) ./ sumc( yy .* yy);
e = y - yy .* rho';
sigma2 = stdc(e)^2 * (N-1) / N;
stderr = sqrt( sigma2 ./ sumc( yy .* yy) );
t = (rho - 1) ./ stderr;
cv = quantile(t,0.01|0.05|0.10);

print ''Approximate asymptotic critical values (t-ratio):'';
print ''-------------------------------------------------'';
print ''  1%      5%      10%'';
print ''-------------------------------------------------'';
print ''-2.56   -1.94   -1.62'';
print '''';
print ''Our critical values:'';
print cv;


Approximate asymptotic critical values (t-ratio):
-------------------------------------------------
  1%      5%      10%
-------------------------------------------------
```

```
-2.56   -1.94   -1.62

Our critical values:

  -2.449
  -1.932
  -1.612
```

39.3 Cointégration vectorielle (méthode de Johansen)

39.3.1 Les procédures

```
declare matrix _johansen_lag = 0;
declare matrix __altnam_x = 0;
declare matrix __altnam_y = 0;

proc (5) =  JohansenTest(X,p);
  local N,T,dX,data,indx,i,exog,M,r0,R1,R,S,S00,S01,S11;
  local X_1,A,lambda,lambda_trace,lambda_max,logl;
  local omat,mask,fmt;
  local table;
  local HL_G,HL_A,HL_lambda;

  N = cols(X);

  dX = X - lag1(X);
  data = dX;
  i = 1;
  do until i > p;
    data = data~lagn(dx,i);
    i = i + 1;
  endo;

  data = packr(data~lag1(x));
  T = rows(data);

  if p == 1;
    exog = ones(T,1);
  else;
    exog = ones(T,1)~data[.,N+1:(p+_johansen_lag)*N];
  endif;

  dX = data[.,1:N];
  X_1 = data[.,(p+1)*N+1:(p+2)*N];

  M = eye(T) - exog*invpd(exog'exog)*exog';
  R0 = M * dX;
  R1 = M * X_1;
  R = R0~R1;

  S = R'R/T;
  S00 = S[1:N,1:N];
```

```
  S01 = S[1:N,N+1:2*N];
  S11 = S[N+1:2*N,N+1:2*N];

  A = invpd(S11)*S01'*invpd(S00)*S01;

  lambda = sortc(eig(A),1);
  lambda_trace = - T * cumsumc( ln(1-lambda) );
  lambda_max = - T * ln(1 - lambda);

  logl = - 0.5*T*cumsumc(rev(ln(1-lambda)));

  HL_G = inv(chol(S11)');
  HL_A = HL_G*S01'invpd(S00)*S01*HL_G';
  HL_lambda = eig(HL_A);

  if __output;
    call header(''Johansen Cointegration Procedure'',0,0);
    omat = seqa(1,1,N)~lambda~logl~lambda_trace~lambda_max;
    print;
    print;
    print ftos(N,''Number of variables:    %lf'',5,0);
    print ftos(T,''Number of observations: %lf'',5,0);
    print ftos(p,''Number of lags:         %lf'',5,0);
    print;
    print;
    print ''        Lambda              Logl              Trace            L-max'';
    print ''------------------------------------------------------------------------------'';
    let fmt[5,3] = ''*.*lf'' 2 0
                   ''*.*lf'' 7 3
                   ''*.*lf'' 7 3
                   ''*.*lf'' 7 3
                   ''*.*lf'' 7 3;
    call printfm(omat,1,fmt);
    print;

  endif;

  table = 0;
  table = vput(table,S00,''S00'');
  table = vput(table,S01,''S01'');
  table = vput(table,S11,''S11'');
  table = vput(table,A,''A'');

  table = vput(table,HL_G,''HL_G'');
  table = vput(table,HL_A,''HL_A'');

  retp(lambda,logl,lambda_trace,lambda_max,table);
endp;

proc (3) = JohansenCoint(table,r);
  local S00,S01,S11,N;
  local A,lambda,beta,indx,alpha,Pi_;
```

```
  local HL_G,HL_mat;

  S00 = vread(table,''S00'');
  S01 = vread(table,''S01'');
  S11 = vread(table,''S11'');
  A = vread(table,''A'');
  HL_G = vread(table,''HL_G'');

  N = rows(S00);

  {lambda,beta} = eigv(A);
  indx = submat(rev(sortc(seqa(1,1,N)~lambda,2)),0,1);
  beta = beta[.,indx];
  beta = real(beta[.,1:r]);
  alpha = S01*beta*invpd(beta'*S11*beta);
  Pi_ = alpha*beta';

  retp(alpha,beta,Pi_);
endp;

proc (4) = VECM(Y,X,Pi_,lag_);
  local K,Np,Nobs,beta,u;
  local dy,dx,c,exog,i,stderr,Mcov,sigma;
  local XXX,YYY,beta_,u_,stderr_;
  local data;

  K = cols(y);
  if rows(X) == rows(Y);
    Np = K*lag_ + 1 + 2*cols(X);
  else;
    Np = K*lag_ + 1;
  endif;

  Nobs = rows(Y);
  beta = zeros(Np,K);
  u = zeros(Nobs,K);
  stderr = zeros(Np,K);

  dy = y - lag1(y);
  dx = x - lag1(x);
  c = ones(Nobs,1);

  exog = {};
  i = 1;
  do until i > lag_;
    exog = exog~lagn(dy,i);
    i = i + 1;
  endo;

  if __altnam_y /= 0 and __altnam_x /= 0 and rows(__altnam_y) == K and rows(__altnam_x) == cols(x);

    __altnam = {};
```

```
    i = 1;
    do until i > lag_;
       __altnam = __altnam | 0 $+ ''d_'' $+ __altnam_y $+ ''_'' $+ ftocv(i,1,0) ;
      i = i + 1;
    endo;

    __altnam = __altnam | ''constant'';
    __altnam = __altnam | 0 $+ __altnam_x;
    __altnam = __altnam | 0 $+ ''d_'' $+ __altnam_x ;

  elseif __altnam_y /= 0 and rows(__altnam_y) == K and rows(y) /= rows(x);

    __altnam = {};
    i = 1;
    do until i > lag_;
       __altnam = __altnam | 0 $+ ''d_'' $+ __altnam_y $+ ''_'' $+ ftocv(i,1,0) ;
      i = i + 1;
    endo;

    __altnam = __altnam | ''constant'';

  else;

    __altnam = 0;

  endif;

  i = 1;
  do until i > K;
    YYY = dy[.,i]-lag1(y)*Pi_[i,.]';

    if rows(X) == rows(Y);
      XXX = exog~c~x~dx;
    else;
      XXX = exog~c;
    endif;

    {beta_,stderr_,Mcov,u_} = OLS(YYY,XXX);

    beta[.,i] = beta_;
    stderr[.,i] = stderr_;
    u[.,i] = u_;

    i = i + 1;
  endo;

  sigma = vcx(packr(u));

  retp(beta,stderr,sigma,u);
endp;
```

39.3.2 Détermination du nombre de cointégrations

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
*/

new;
library ritme;

cls;

load y[92,3] = lutkepoh.asc;
y = ln(y[1:76,.]);

INVEST = y[.,1];
INCOME = y[.,2];
CONSUM = y[.,3];

let names = INVEST INCOME CONSUM;

outwidth 256;
output file = vecm1.out reset;

lag_ = 1;
do until lag_ > 10;
  {lambda,logl,lambda_trace,lambda_max,table} = JohansenTest(y,lag_);
  lag_ = lag_ + 1;
endo;

output off;

lag_ = 4;

{lambda,logl,lambda_trace,lambda_max,table} = JohansenTest(y,lag_);

output file = vecm1.out on;

rank_ = 1;
do until rank_ > 1;
  {alpha,beta,Pi_} = JohansenCoint(table,rank_);
  print ''-----------------------------------------------'';
  print ftos(rank_,''            rank = %lf'',3,0);
  print ''Pi = '';
  print Pi_;
  print ''beta = '';
  print beta./beta[2];
  rank_ = rank_ + 1;
endo;

output off;
```

```
================================================================================
 Johansen Cointegration Procedure                          4/09/2002  11:54 pm
================================================================================


Number of variables:        3
Number of observations:    65
Number of lags:            10


        Lambda              Logl              Trace            L-max
--------------------------------------------------------------------------
 1        0.003            12.740              0.173            0.173
 2        0.176            19.019             12.730           12.557
 3        0.324            19.106             38.211           25.481

------------------------------------------------
            rank =   1
Pi =

-0.004754  -0.6686   0.6930
0.0007231   0.1017  -0.1054
0.001821   0.2561  -0.2654
beta =

0.007110
   1.000
  -1.036
```

39.3.3 Estimation du modèle à correction d'erreur vectoriel

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
*/

new;
library ritme;

cls;

load y[92,3] = lutkepoh.asc;
y = ln(y[1:76,.]);

INVEST = y[.,1];
INCOME = y[.,2];
CONSUM = y[.,3];

let names = INVEST INCOME CONSUM;
```

```
lag_ = 3;
{lambda,logl,lambda_trace,lambda_max,table} = JohansenTest(y,lag_);
rank_ = 2;
{alpha,beta,Pi_} = JohansenCoint(table,rank_);

lag_ = lag_ - 1;

let __altnam_y = ''INV'' ''INC'' ''CON'';
__altnam_x = 0;
{coeffs,stderr,sigma,u} =  vecm(y,0,Pi_,lag_);

__fmtnv = { ''*.*lg '' 10 3 };
coeffs = alpha'|coeffs;
__altnam = (0 $+ ''L'' $+ ftocv(seqa(1,1,rank_),1,0)) | __altnam;

output file = vecm2.out reset;

call printfmt(__altnam~coeffs,0~ones(1,3));

output off;


      L1       0.92     -0.116     -0.206
      L2     0.0683     -0.011      0.011
 d_INV_1     -0.285      0.037     0.0174
 d_INC_1      0.538     -0.196     0.0674
 d_CON_1      0.929      0.287     -0.198
 d_INV_2     -0.139     0.0456      0.048
 d_INC_2      0.424    -0.0175       0.26
 d_CON_2       1.14    -0.0386    -0.0467
constant    -0.0308     0.0162     0.0312
```

40 Les séries temporelles

- TS
- GaussX
- TSM

Sixième partie
GAUSS 5.0

41 Les tableaux multidimensionnels

aconcat	Concatenates conformable matrices and arrays in a user-specified dimension.
amean	Computes the mean across one dimension of an N-dimensional array.
areshape	Reshapes a scalar, matrix, or array into an array of user-specified size.
arrayalloc	Creates an N-dimensional array with unspecified contents.
arrayinit	Creates an N-dimensional array with a specified fill value.
arraytomat	Changes an array to type matrix.
asum	Computes the sum across one dimension of an N-dimensional array.
atranspose	Transposes an N-dimensional array.
getarray	Gets a contiguous subarray from an N-dimensional array.
getdims	Gets the number of dimensions in an array.
getmatrix	Gets a contiguous matrix from an N-dimensional array.
getmatrix4D	Gets a contiguous matrix from a 4-dimensional array.
getorders	Gets the vector of orders corresponding to an array.
getscalar3D	Gets a scalar from a 3-dimensional array.
getscalar4D	Gets a scalar form a 4-dimensional array.
loopnextindex	Increments an index vector to the next logical index and jumps to the specified label if the index did not
mattoarray	Changes a matrix to a type array.
nextindex	Returns the index of the next element or subarray in an array.
previousindex	Returns the index of the previous element or subarray in an array.
putarray	Puts a contiguous subarray into an N-dimensional array and returns the resulting array.
setarray	Sets a contiguous subarray of an N-dimensional array.
walkindex	Walks the index of an array forward or backward through a specified dimension.

TAB. 1 – Liste des functions concernant les tableaux multidimensionnels

Voyons quelques exemples d'utilisation des tableaux multidimensionnels (qui sont en fait des tenseurs).

```
new;

cls;

let array x[2,2,3] = 1 2 3 4 5 6 7 8 9 10 11 12;
print "x = " x;

y = exp(x) .* x .* cos(x);
print "y = " y;

z = y[1,.,.] + y[2,.,.];
print "z = " z;

w = aconcat(y[1,.,.],z,3);
print "w = " w;


x =
```

```
Plane [1,.,.]

       1.0000000        2.0000000        3.0000000
       4.0000000        5.0000000        6.0000000

Plane [2,.,.]

       7.0000000        8.0000000        9.0000000
       10.000000        11.000000        12.000000
y =

Plane [1,.,.]

       1.4686939       -6.1498646       -59.653593
      -142.75093        210.49601        2324.1620

Plane [2,.,.]

       5787.2795       -3469.8359       -66446.685
      -184817.80        2914.8336        1648095.3
z =

Plane [1,.,.]

       5788.7482       -3475.9858       -66506.338
      -184960.55        3125.3296        1650419.5
w =

Plane [1,.,.]

       1.4686939       -6.1498646       -59.653593
      -142.75093        210.49601        2324.1620

Plane [2,.,.]

       5788.7482       -3475.9858       -66506.338
      -184960.55        3125.3296        1650419.5
```

Un exemple de simulation d'une variable aléatoire χ_2.

```
new;

cls;

/*
** Simulation d'une matrice 1000*3 de chi2(1)
*/

r = 1000;
c = 3;

x = rndn(r,c);
chi2 = x^2;
```

```
print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant une boucle
*/

nu = 5;
chi2 = zeros(r,c);

for i (1,c,1);
  chi2[.,i] = sumc(rndn(nu,r)^2);
endfor;

print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant la commande reshape
*/

rndseed 123;
chi2 = reshape(sumc(rndn(nu,r*c)^2),r,c);

print meanc(chi2)~stdc(chi2);

/*
** Simulation d'une matrice 1000*3 de chi2(5)
**
** en utilisant les tableaux multidimensionnels
*/

rndseed 123;
x = rndn(r*c*nu,1);
orders = nu|r|c;

y = areshape(x,orders);
y = y .* y;

chi2 = arraytomat(asum(y,3));
print meanc(chi2)~stdc(chi2);


       1.0904490        1.5662331
       1.1000481        1.5862624
       1.0854098        1.4747488

       5.0120703        3.1569619
       4.8923906        3.2383112
```

```
      5.0366850       3.1582330

      4.9158512       2.9633891
      5.0314151       3.1005409
      5.0650645       3.2240497

      4.9158512       2.9633891
      5.0314151       3.1005409
      5.0650645       3.2240497
```

Dans le programme suivant, nous considrons la simulation d'un processus GBM multidimensionnel. **Notons que la simulation de plusieurs trajectoires nécessitent l'utilisation d'une boucle.**

```
new;
library pgraph;

/*
** Modele GBM multidimensionnel
**
** dX_i(t) = \mu_i * X_i(t) * dt + \sigma_i * X_i(t) * dW_i(t), i = 1,...,n
**
** E[W_i(t)W_j(t)] = rho_{i,j}
**
**
*/

proc (1) = simulate_mGBM(x0,mu,sigma,rho,t);
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = zeros(Nt,Nx);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(1,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
    x[i,.] = x0;
```

```
    i = i + 1;
  endo;

  retp(x);
endp;

let x0 = 100 50 75;
let mu = 0.05 0.07 0.08;
let sigma = 0.15 0.25 0.05;

let rho = {1,
           0.5, 1,
           0.25, 0.15, 1};

rho = xpnd(rho);  // transforme le vecteur 'vech(rho)' en matrice de correlation

t = seqa(0,1/365,5*365+1);

x = simulate_mGBM(x0,mu,sigma,rho,t);

graphset;
  _pdate = ""; _pframe = 0; _pnum = 2;
  _pltype = 6|1|3; _plwidth = 5;
  title("\214GBM Simulation");
  xlabel("\214time");
  xtics(0,5,1,12);
  xy(t,x);


/*
** Si nous voulons obtenir non pas une seule trajectoire simulee du processus 3D
** mais plusieurs trajectoires, nous pouvons utiliser une boucle
**
*/

Ns = 1000;
Nt = rows(t);

x1 = zeros(Nt,Ns); x2 = x1; x3 = x1;

t0 = hsec;

for i (1,Ns,1);
  x = simulate_mGBM(x0,mu,sigma,rho,t);
  x1[.,i] = x[.,1];
  x2[.,i] = x[.,2];
  x3[.,i] = x[.,3];
endfor;

ct = (hsec-t0)/100;
```

```
print ftos(ct,"Computational time : %f seconds.",5,2);

graphset;
  _pltype = 3;
  xy(t,x1[.,1:100]);   // Graphe des 100 premieres trajectoires de X_1(t)


Computational time : 59.45 seconds.
```

Avec les tenseurs, il n'est plus nécessaire d'employer une boucle (et les temps de calcul sont fortement réduits).

```
new;
library pgraph;

cls;

/*
** Modele GBM multidimensionnel
**
** dX_i(t) = \mu_i * X_i(t) * dt + \sigma_i * X_i(t) * dW_i(t), i = 1,...,n
**
** E[W_i(t)W_j(t)] = rho_{i,j}
**
**
*/

proc (1) = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = zeros(Nt,Nx);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(1,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
```

```
    x[i,.] = x0;

    i = i + 1;
  endo;

  retp(x);
endp;

proc (1) = array_simulate_mGBM(x0,mu,sigma,rho,t,Ns);
  local Nt,Nx,x,Pchol,i,dt,k1,k2,u;

  Nt = rows(t);
  Nx = rows(x0);

  x = arrayinit(Nt|Ns|Nx,0);
  x0 = x0';
  mu = mu';
  sigma = sigma';
  Pchol = chol(rho);                // Decomposition Cholesky de la matrice de correlation

  i = 1;
  do until i > Nt;

    if i == 1;
      dt = t[1];
    else;
      dt = t[i] - t[i-1];
    endif;

    k1 = (mu-0.5*sigma^2) * dt;
    k2 = sigma * sqrt(dt);
    u = rndn(ns,nx)*Pchol;

    x0 = x0 .* exp( k1 + k2 .* u);
    x[i,.,.] = x0;

    i = i + 1;
  endo;

  retp(x);
endp;

let x0 = 100 50 75;
let mu = 0.05 0.07 0.08;
let sigma = 0.15 0.25 0.05;

let rho = {1,
           0.5, 1,
           0.25, 0.15, 1};

rho = xpnd(rho);  // transforme le vecteur 'vech(rho)' en matrice de correlation
```

```
dt = 365;
t = seqa(0,1/dt,5*dt+1);

/*
** Verifions tout d'abord que les procedures array_simulate_GBM
** et matrix_simulate_GBM donnent les memes resultats lorsque
** le nombre de trajectoires est fixe a 1.
*/

rndseed 123;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1);
x = getmatrix(atranspose(x,2|1|3),1);

rndseed 123;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);

err = maxc(maxc(abs(x-y)));
print ftos(err,"Maximal absolute error = %lE",10,5);


/*
** Comparons maintenant les temps de calcul
*/

print;
print "Computational time for matrix_simulate_mGBM";

t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for one simulation = %lf seconds",5,2);

t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for two simulations = %lf seconds",5,2);

t0 = hsec;
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
print ftos((hsec-t0)/100,"for three simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,10,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
print ftos((hsec-t0)/100,"for 10 simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,100,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
```

```
print ftos((hsec-t0)/100,"for 100 simulations = %lf seconds",5,2);

t0 = hsec;
for i (1,1000,1);
  y = matrix_simulate_mGBM(x0,mu,sigma,rho,t);
endfor;
print ftos((hsec-t0)/100,"for 1000 simulations = %lf seconds",5,2);

print;
print "Computational time for array_simulate_mGBM";

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1);
print ftos((hsec-t0)/100,"for one simulation = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,2);
print ftos((hsec-t0)/100,"for two simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,3);
print ftos((hsec-t0)/100,"for three simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,10);
print ftos((hsec-t0)/100,"for 10 simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,100);
print ftos((hsec-t0)/100,"for 100 simulations = %lf seconds",5,2);

t0 = hsec;
x = array_simulate_mGBM(x0,mu,sigma,rho,t,1000);
print ftos((hsec-t0)/100,"for 1000 simulations = %lf seconds",5,2);


Ns = 100;

x = array_simulate_mGBM(x0,mu,sigma,rho,t,Ns);

x = atranspose(x,3|1|2);
x1 = getmatrix(x,1);              /* Matrice Nt * 100 : la colonne i correspond a la i-ieme trajectoire
x2 = getmatrix(x,2);
x3 = getmatrix(x,3);


Computational time for matrix_simulate_mGBM
for one simulation =  0.06 seconds
for two simulations =  0.11 seconds
for three simulations =  0.17 seconds
for 10 simulations =  0.59 seconds
for 100 simulations =  5.78 seconds
```

```
for 1000 simulations = 57.86 seconds

Computational time for array_simulate_mGBM
for one simulation =  0.08 seconds
for two simulations =  0.06 seconds
for three simulations =  0.08 seconds
for 10 simulations =  0.11 seconds
for 100 simulations =  0.44 seconds
for 1000 simulations =  3.80 seconds
```

42 L'importation et l'exportation de données EXCEL

Dans l'ancienne version de GAUSS, nous utilisons les commandes `export/exportf` et `import/importf`.

```
new;
library gauss;

fname = "tmp.xls";
{data,varnames} = import(fname,0,2);

call export(data,"tmp2.xls",varnames);
```

La nouvelle version de GAUSS dispose de 3 nouvelles commandes plus puissantes et plus souples :

- `SpreadSheetReadM` : Lecture d'un fichier Excel et importation des données dans une matrice.
- `SpreadSheetReadSA` : Lecture d'un fichier Excel et importation des données dans un tableau de chaines de caractères.
- `SpreadSheetWrite` : Ecriture dans un fichier Excel.

```
new;
library gauss;

data = rndn(100,101);

SpreadsheetWrite(data,"d:\\gauss50\\ritme\\tmp3.xls","a1:c10",1);    // Ecriture de data[1:10,1:3]

SpreadsheetWrite(data,"d:\\gauss50\\ritme\\tmp3.xls",range(data),2); // Ecriture de data

x = SpreadsheetReadM("d:\\gauss50\\ritme\\tmp3.xls","a1:c100",2);    // Lecture de data[1:100,1:3]

sa = SpreadsheetReadSA("d:\\gauss50\\ritme\\tmp3.xls","a1:c100",2);  // Lecture sous forme d'un tableau
                                                                     // de chaines de caracteres

proc range(x);
  local r,c,xxx,rg;

  r = rows(x);
  c = cols(x);

  if c <= 26;
    rg = "A1:" $+ chrs(64+c) $+ ftos(r,"%lf",1,0);
  elseif c <= 26^2;
    xxx = trunc(c/26);
    xxx = xxx | (c-26*xxx);
```

```
    rg = "A1:" $+ chrs(64+xxx[1]) $+ chrs(64+xxx[2]) $+ ftos(r,"%lf",1,0);
  elseif c <= 26^3;
    xxx = trunc(c/26^2);
    xxx = xxx | (c-26^2*xxx);
    xxx[2] = trunc(xxx[2]/26);
    xxx = xxx | (c-26^2*xxx[1]-26*xxx[2]);
    rg = "A1:" $+ chrs(64+xxx[1]) $+ chrs(64+xxx[2]) $+ chrs(64+xxx[3]) $+ ftos(r,"%lf",1,0);
  endif;

  retp(rg);
endp;
```

Remarque 24 *Il est nécessaire de donner le chemin complet. Dans le mode écriture, GAUSS ne remplace que les cellules spécifiées et ne réinitialise pas la feuille EXCEL.*

43 Un exemple : la gestion de portefeuille

43.1 Le problème mathématique

Nous considèrons M actifs. Nous notons $\tilde{R}_m$ le rendement aléatoire de l'actif m. Soit $\tilde{R}$ le vecteur aléatoire des rendements défini par

$$\tilde{R} = \begin{bmatrix} \tilde{R}_1 \\ \vdots \\ \tilde{R}_m \\ \vdots \\ \tilde{R}_M \end{bmatrix}$$

Soit α la stratégie représentée par le vecteur des proportions des actifs du portefeuille P. Dans le cadre méthodologique de Markowitz, l'objectif de l'agent est alors de maximiser le rendement espéré du portefeuille sous la contrainte d'un niveau fixé de risque $\sigma^\star$. D'un point de vue mathématique, le programme s'écrit :

$$\begin{aligned} &\max_{\alpha} E\left[\tilde{R}_P\right] \\ &\text{s.c.} \quad \begin{cases} \sigma\left[\tilde{R}_P\right] = \sigma^\star \\ \alpha_m \geq 0 \end{cases} \end{aligned} \tag{132}$$

avec $E\left[\tilde{R}_P\right]$ le rendement moyen du portefeuille et $\sigma\left[\tilde{R}_P\right]$ le risque du portefeuille.

Le problème (132) est relativement difficile à résoudre. La contribution majeure de Markowitz à la théorie du portefeuille est d'avoir montré que ce problème est équivalent au problème suivant :

$$\begin{aligned} &\max_{\alpha} \psi E\left[\tilde{R}_P\right] - \sigma^2\left[\tilde{R}_P\right] \\ &\text{s.c.} \quad \begin{cases} \sum_{m=1}^{M} \alpha_m = 1 \\ \alpha_m \geq 0 \end{cases} \end{aligned} \tag{133}$$

Nous pouvons interpréter le coefficient ψ comme un coefficient de préférence pour le rendement. Plus ce coefficient est élevé, plus la stratégie optimale sera rentable, mais risquée. Si $E\left[\tilde{R}\right] = \mu$ et $\sigma^2\left[\tilde{R}\right] = \Sigma$, nous avons

$$E\left[\tilde{R}_P\right] = \alpha^\top \mu$$

et

$$\sigma^2\left[\tilde{R}_P\right] = \alpha^\top \Sigma \alpha$$

Le portefeuille optimal est donc la solution du programme quadratique :

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^\top [2\Sigma]\,\alpha - \alpha^\top [\psi\mu] \\ &\text{s.c.} \quad \left\{ \begin{array}{l} \mathbf{1}^\top \alpha = 1 \\ \mathbf{0} \leq \alpha \leq \mathbf{1} \end{array} \right. \end{aligned} \tag{134}$$

La frontière efficiente correspond alors à l'ensemble des portefeuilles optimaux ($\psi \in [-\infty, +\infty]$).

Nous pouvons obtenir la solution du problème précédent en utilisant la procédure `QProg`.

```
external matrix _PO_A;
external matrix _PO_B;
external matrix _PO_C;
external matrix _PO_D;
external matrix _PO_bnds;

proc (3) = PortfolioOptimization(mu,cov,psi);
  local Number_Assets,N,i;
  local sv,Q,R,A,B,C,D,bnds;
  local alpha,u1,u2,u3,u4,retcode,ER,SD;

  Number_Assets = rows(mu);
  sv = ones(Number_Assets,1)/Number_Assets;

  if _PO_A == 0;
    A = ones(1,Number_Assets);
    B = 1;
  else;
    A = _PO_A;
    B = _PO_B;
  endif;

  if _PO_C == 0;
    C = 0;
    D = 0;
  else;
    C = _PO_C;
    D = _PO_D;
  endif;

  if _PO_bnds == 0;
    bnds = {0 1};
  else;
    bnds = _PO_bnds;
  endif;

  Q = 2*cov;
  N = rows(psi);
  alpha = zeros(Number_Assets,N);
  i = 1;
```

```
  do until i > N;
    R = psi[i]*mu;
    {alpha[.,i],u1,u2,u3,u4,retcode} = Qprog(sv,Q,R,A,B,C,D,bnds);
    if retcode /= 0;
      alpha[.,i] = miss(zeros(Number_Assets,1),0);
    endif;
    i = i + 1;
  endo;

  if N /= 1;
    alpha = alpha';
    ER = alpha * mu;
    SD = sqrt(diag(alpha*cov*alpha'));
  else;
    ER = alpha' mu;
    SD = sqrt(alpha'cov*alpha);
  endif;

  retp(alpha,ER,SD);
endp;
```

43.2 Un premier exemple

L'exemple suivant est issu de "Markowitz portfolio analysis for the individual investor" par SCHOENBERG [1995]. Il considère 6 actifs *Merill Lynch, Paine Webber, Digital Equipment, Microsoft, Silicon Graphics* et *Hewlett-Packard*.

```
new;
library ritme;

let mu[6,1] = 15.852
              14.262
              31.336
              25.775
              50.228
              14.842;

let sigma[6,1] = 37.215
                 41.773
                 33.165
                 62.009
                 60.720
                 23.757;

let cor[6,6] =   1.000  0.944  0.146  0.231  0.379  0.258
                 0.944  1.000  0.109  0.239  0.413  0.223
                 0.146  0.109  1.000 -0.169 -0.229  0.691
                 0.231  0.239 -0.169  1.000  0.882 -0.256
                 0.379  0.413 -0.229  0.882  1.000 -0.284
                 0.258  0.223  0.691 -0.256 -0.284  1.000;

cov = cor .* sigma .* sigma';
```

```
psi = seqa(20,10,6);
{alpha,ER,SD} = PortfolioOptimization(mu,cov,psi);
cls;

output file = po1.out reset;

format /rd 10,3;
print "=========================================================================";
print "             PREFERENCE DE L'INVESTISSEUR POUR LE RETURN           ";
print "-------------------------------------------------------------------------";
print seqa(20,10,6)';
print "=========================================================================";
print "                     RENDEMENT MOYEN DU PORTEFEUILLE                 ";
print "-------------------------------------------------------------------------";
print ER;
print "=========================================================================";
print "                          RISQUE DU PORTEFEUILLE                     ";
print "-------------------------------------------------------------------------";
print SD;
print "=========================================================================";
print "                        COMPOSITION DU PORTEFEUILLE                  ";
print "-------------------------------------------------------------------------";
print alpha';
print "=========================================================================";

output off;
```

```
=========================================================================
             PREFERENCE DE L'INVESTISSEUR POUR LE RETURN
-------------------------------------------------------------------------
    20.000     30.000     40.000     50.000     60.000     70.000
=========================================================================
                     RENDEMENT MOYEN DU PORTEFEUILLE
-------------------------------------------------------------------------

    29.721
    33.384
    37.047
    38.065
    38.377
    38.690
=========================================================================
                          RISQUE DU PORTEFEUILLE
-------------------------------------------------------------------------

    21.198
    23.258
    25.868
    26.686
    27.006
    27.380
```

```
================================================================
                    COMPOSITION DU PORTEFEUILLE
----------------------------------------------------------------

    0.000     0.000     0.000     0.000     0.000     0.000
    0.000     0.000     0.000     0.000     0.000     0.000
    0.336     0.481     0.626     0.644     0.627     0.611
    0.000     0.000     0.000     0.000     0.000     0.000
    0.264     0.300     0.336     0.356     0.373     0.389
    0.400     0.219     0.038     0.000     0.000     0.000
================================================================
```

Pour obtenir la frontière efficiente, il suffit de faire varier ψ dans un intervalle $[\psi_-, \psi_+]$ et de représenter la fonction paramétrée $\left(\sigma\left[\tilde{R}_P\right], E\left[\tilde{R}_P\right]\right)$.

```
new;
library ritme,pgraph;

#include po.inf;

psi = seqa(-25,1,126);
{alpha,ER,SD} = PortfolioOptimization(mu,cov,psi);

graphset;
  _pdate = ""; _pnum = 2; _paxht = 0.18; _pnumht = 0.18; _ptitlht = 0.18; _pframe = 0;
  _pmcolor = 0|0|0|0|0|0|0|0|15;
  title("\214Efficient Frontier");
  xlabel("\214Risk");
  ylabel("\214Return");
  graphprt("-c=1 -cf=po2.ps");
  xy(SD,ER);
```

43.3 La prise en compte de préférences allocatives et de contraintes techniques

La procédure `PortfolioOptimization` permet de prendre en compte les préférences allocatives de l'agent. Supposons par exemple que l'agent désire détenir 1/3 de son portefeuille en actions du secteur finance. Dans ce cas, la contrainte $A\alpha = B$ du programme d'optimisation quadratique devient

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \alpha = \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix}$$

Si l'agent désire détenir au moins 25% de son portefeuille en actions Paine Webber, la contrainte $C\alpha \geq D$ devient

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \alpha \geq 0.25$$

Le dernier exemple considére une réallocation de portefeuille. Le portefeuille original de l'agent est

$$\alpha_0 = \begin{bmatrix} 0.15 \\ 0.20 \\ 0.17 \\ 0.14 \\ 0.04 \\ 0.30 \end{bmatrix}$$

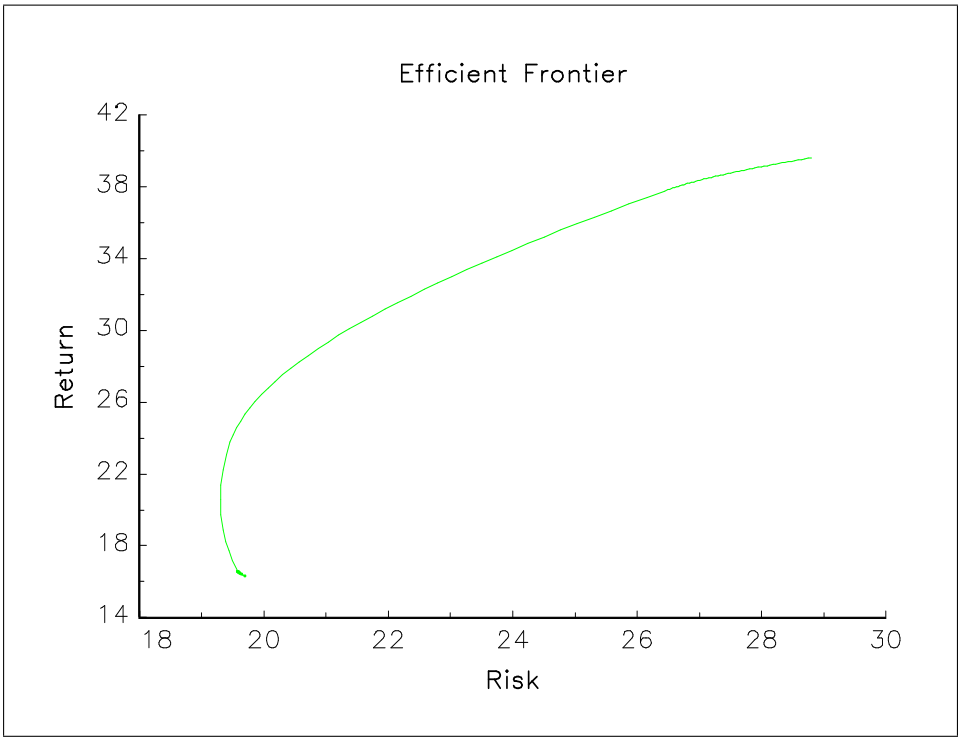


FIG. 13 – Frontière efficiente

Cet agent désire modifier cette composition de la façon suivante :

$$-0.10 \leq \alpha - \alpha_0 \leq 0.10$$

```
new;
library ritme;

#include po.inf;

psi = 20;    /* Preference de l'investisseur pour le return */

output file = po3.out reset;

print "L'agent desire detenir 1/3 de son portefeuille en actions du secteur finance";
print "et 2/3 en actions du secteur informatique";

_PO_A = {1 1 0 0 0 0,
         0 0 1 1 1 1};
_PO_B = {0.334,0.666};
_PO_C = 0; _PO_D = 0; _PO_bnds = 0;

{alpha,ER,SD} = PortfolioOptimization(mu,cov,psi);
print alpha;

print "L'agent desire detenir au mois 25% de son portefeuille en actions Painer Webber";
```

```
_PO_A = 0; _PO_B = 0; // _PO_C = eye(6); _PO_D = 0|0.25|0|0|0|0; _PO_bnds = 0;
_PO_C = 0; _PO_D = 0;
_PO_bnds = {0 1, 0.25 1, 0 1, 0 1, 0 1, 0 1};

{alpha,ER,SD} = PortfolioOptimization(mu,cov,psi);
print alpha;

print "L'agent desire recomposer son portefeuille (+/- 10%)";

alpha0 = 0.15|0.20|0.17|0.14|0.04|0.30;

_PO_A = 0; _PO_B = 0; _PO_C = 0; _PO_D = 0;
_PO_bnds = alpha0 + (-0.10~0.10);

{alpha,ER,SD} = PortfolioOptimization(mu,cov,psi);

print "Portefeuille d'origine    Nouveau portefeuille";
print alpha0~alpha;

SD0 = sqrt(alpha0'*cov*alpha0);
ER0 = alpha0'mu;

print "Rendement moyen ";
print ER0~ER;
print "Risque";
print SD0~SD;

output off;

L'agent desire detenir 1/3 de son portefeuille en actions du secteur finance
et 2/3 en actions du secteur informatique

     0.334
     0.000
     0.346
     0.000
     0.158
     0.161
L'agent desire detenir au mois 25% de son portefeuille en actions Painer Webber

     0.000
     0.250
     0.353
     0.000
     0.175
     0.222
L'agent desire recomposer son portefeuille (+/- 10%)
Portefeuille d'origine    Nouveau portefeuille

     0.150      0.050
     0.200      0.100
     0.170      0.270
```

```
     0.140      0.040
     0.040      0.140
     0.300      0.400
Rendement moyen
    20.628     24.679
Risque
    23.063     20.669
```

Remarque 25 *La procédure* `PortfolioOptimization` *permet de prendre en compte l'existence d'un marché à terme. Ainsi, autoriser la vente de contrats à terme sur l'actif i revient à imposer la restriction $\alpha_i < 0$. Ceci peut être pris en compte lors de la définition des variables* `_PO_C` *ou* `_PO_bnds`*.*

44 L'utilisation de Mercury

Celle-ci a fait l'objet d'une présentation intensive dans un précédent séminaire :

http ://www.business.city.ac.uk/ferc/thierry/conf3pdf.zip

44.1 Un premier exemple

```
Sub Test1()
    mercury.Load
    Dim z As Double
    Dim gcode As String
    Sheets("sheet1").Select              ' select worksheet
    ge.GaussOpen                         ' open Gauss
    ge.Exec ("x = 2.5")                  ' executes Gauss code
    ge.ValPut "y", 2#                    ' copies a scalar to Gauss
    gcode = "print x;" & _
            "print y;" & _
            "z = sin(x) * (y/x); print z"
    ge.OutputReset
    ge.Exec gcode                        ' do Gauss calculation
    ge.ValGet "z", z                     ' retrieve result
    ge.OutputShow
    ge.RangeGet "z", "a1"
    ge.GaussClose
End Sub
```

44.2 Un deuxième exemple

```
Sub Test2()
    mercury.Load
    Dim gcode As String
    Sheets("sheet1").Select
    ge.GaussOpen
    ge.Exec ("x = rndn(100,10);")
    ge.RangeGet "x", "d1"
    ge.GaussClose
End Sub
```