GAUSS dans la banque

Paris, 20 avril 2000

Damien Jacomy & 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

Table des matières

1 Une courte introduction à GAUSS

Le lecteur peut télécharger les versions PDF des précédents séminaires pour avoir une vue plus complète du logiciel GAUSS à l'adresse suivante :

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

3 séminaires sont disponibles

http : //www.city.ac.uk/cubs/ferc/thierry/finpdf.zip
http : //www.city.ac.uk/cubs/ferc/thierry/conf2pdf.zip
http : //www.city.ac.uk/cubs/ferc/thierry/conf3pdf.zip

GAUSS est un langage de programmation **matriciel** doté d'une interface utilisateur. Ses points forts sont les suivants :

- Il s'agit d'un langage matriciel vectorisé ;
- Il possède de nombreuses fonctions mathématiques ;
- Les variables globales sont déclarées par défaut ;
- La gestion des fichiers sources est largement facilitée par l'utilisation de bibliothèques de procédures ;
- Il intègre une bibliothèque graphique très riche ;
- Il existe de nombreux modules applicatifs en Econométrie, Finance, Statistiques et Mathématiques Appliquées.

1.1 Quelques exemples

1.1.1 GAUSS est un langage matriciel vectorisé

Dans l'exemple qui suit, nous simulons des variables aléatoires de Student et illustrons le théorème de limite centrale. Noter qu'aucune boucle n'a été utilisée.

```
/*
** Simulation de variables aléatoires Student
** Illustration du théorème Central Limite
**
*/

new;

Ns = 1000; /* Nombre de simulations  */
df = 10;   /* Degrés de liberté      */

u = rndn(df,Ns);  /* Simulations de variables aléatoires gaussiennes                */
x = u^2;          /* Simulations de variables aléatoires Chi2 à 1 degré de liberté  */
y = sumc(x);      /* Simulations de variables aléatoires Chi2 à df degrés de liberté */

t = rndn(Ns,1) ./ y ; /* nombres aléatoires de student */

m = cumsumc(t) ./ seqa(1,1,Ns);

print m[1 10 25 50 100 1000];


      0.12623647
    -0.066408472
    -0.034980288
    -0.028814007
    -0.017800665
    0.0059631106
```

GAUSS

File Edit Search Font Options Window Help

Debug Run d:\gauss\gro\exemples\sde7.prg

Stop Edit d:\gauss\gro\exemples\sde7.prg

```
GAUSS for Windows NT/95  Version 3.2.38  ( Fe
(C) Copyright 1984-1999 Aptech Systems, Inc.
All Rights Reserved.
262078448 bytes workspace

(gauss) target exemples

(gauss) run d:\gauss\gro\exemples\sde7.prg
(gauss)
```

GAUSS-Edit

File Edit Search Font Options Window Help

Debug Run d:\gauss\gro\exemples\sde7.prg

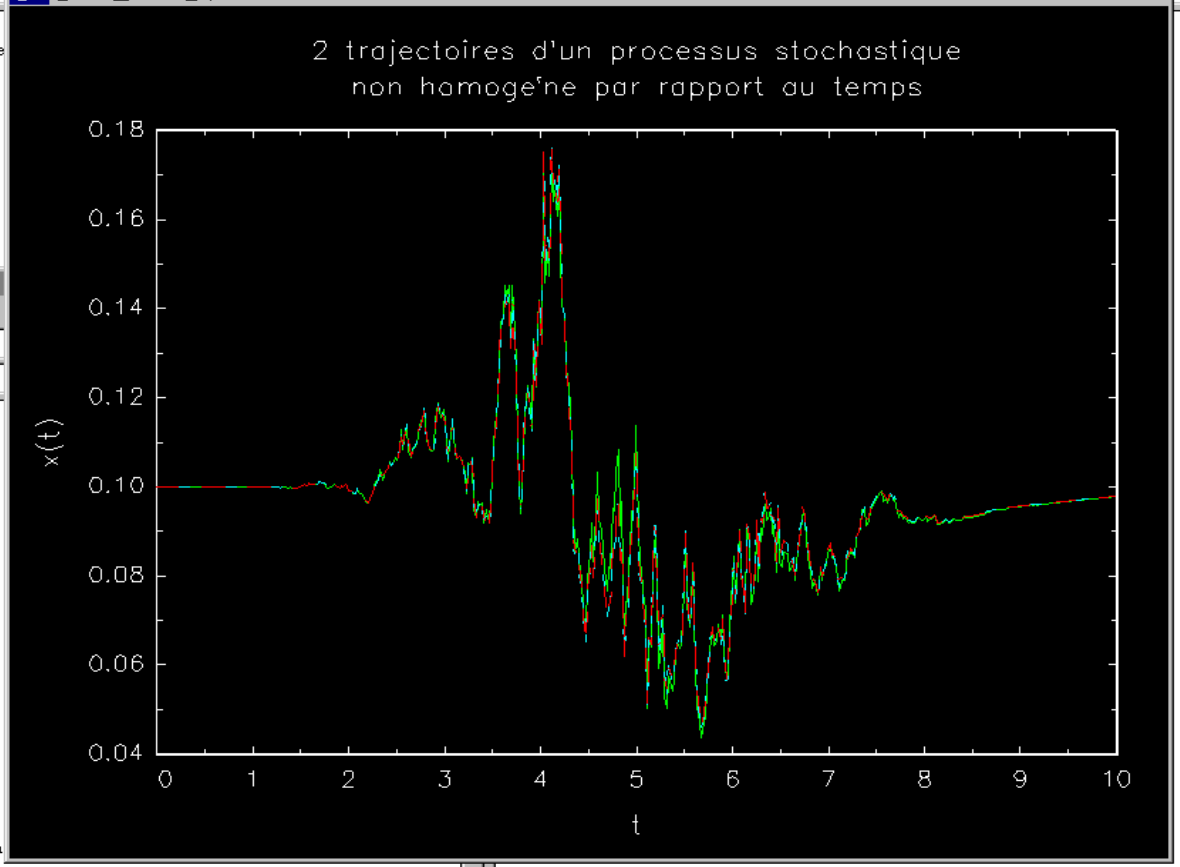
Save Edit d:\gauss\gro\exemples\sde7.prg

```
new;
library gro,pgraph;
libgro;

seed = 12546;

proc procMu(t,x);
  retp(0.8*(0.10-x));
endp;

proc procSigma(t,x);
  retp(exp(-(t-5)^2/3).*x);
endp;

proc procGradSigma(t,x);
  retp(exp(-(t-5)^2/3));
endp;

rndseed seed;
{t,x1} = simulate_SDE(0.1,&procMu,&procSigm
rndseed seed;
{t,x2} = simulate_Milstein(0.1,&procMu,&procSigma,0,0,10,500,1);
rndseed seed;
{t,x3} = simulate_Milstein(0.1,&procMu,&procSigma,&procGradSigma,0,10,500,1);

graphset;
  _pdate = ""; _pnum = 2; _pltype = 6|3|1;
  title("2 trajectoires d'un processus stochastique"\
```

Editing d:\gauss\gro\exemples\sde7.prg L:1 C:1

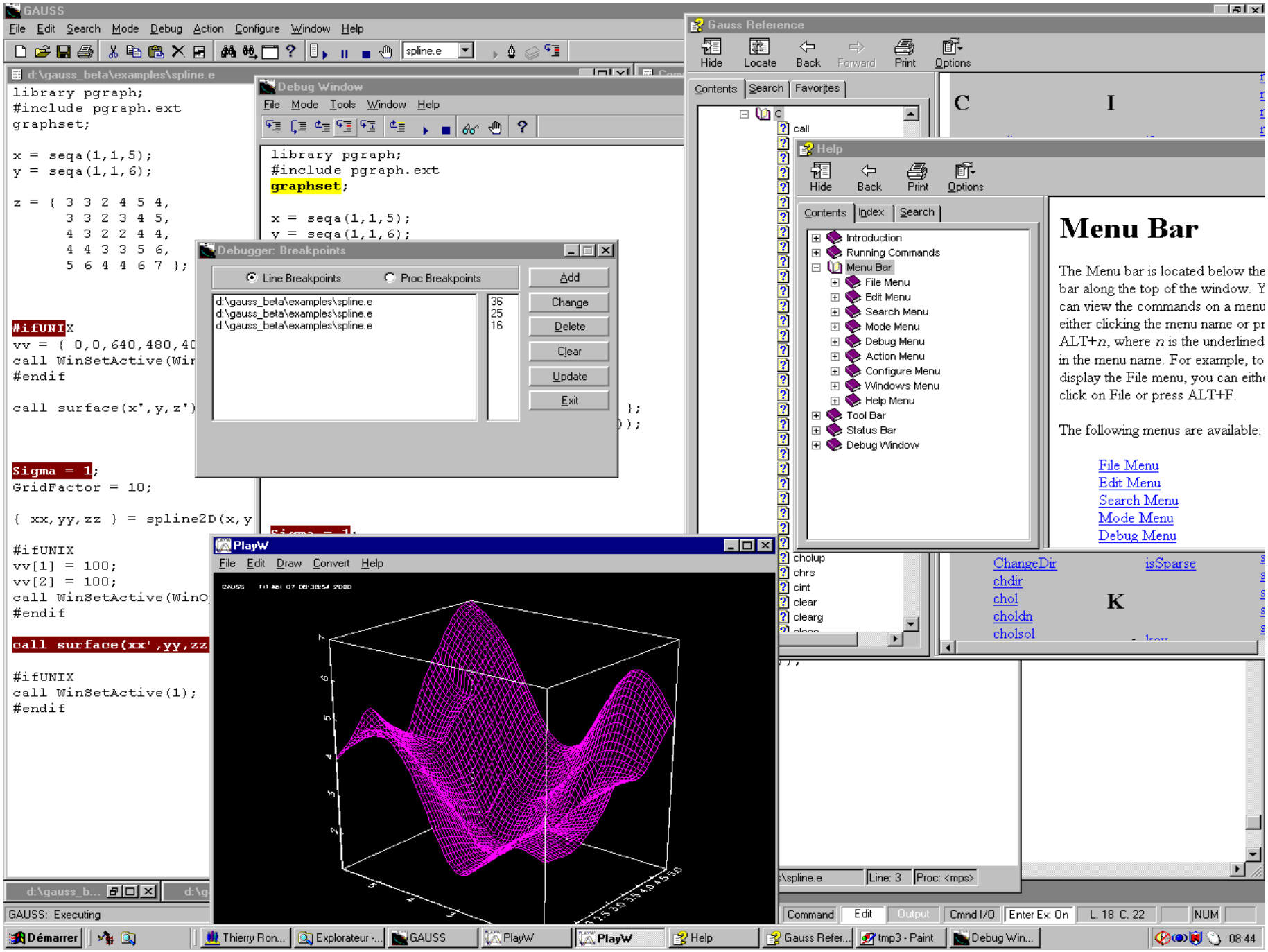
PQG-graphic.tkf

File Zoom Window Help



Enter Command

Démarrer | Thierry Roncalli - Co... | Explorateur - D:\User... | TextPad - [D:\Users\... | Scientific Word - [D:\... | GAUSS | GAUSS-Edit | PQG-graphic.tkf | 11:08


GAUSS
File Edit Search Mode Debug Action Configure Window Help
spline.e
d:\gauss_beta\examples\spline.e
library pgraph;
#include pgraph.ext
graphset;
x = seqa(1,1,5);
y = seqa(1,1,6);
z = { 3 3 2 4 5 4,
3 3 2 3 4 5,
4 3 2 2 4 4,
4 4 3 3 5 6,
5 6 4 4 6 7 };
#ifUNIX
vv = { 0,0,640,480,40
call WinSetActive(Wir
#endif
call surface(x',y,z')
Sigma = 1;
GridFactor = 10;
{ xx,yy,zz } = spline2D(x,y
#ifUNIX
vv[1] = 100;
vv[2] = 100;
call WinSetActive(WinO
#endif
call surface(xx',yy,zz
#ifUNIX
call WinSetActive(1);
#endif
Debug Window
File Mode Tools Window Help
library pgraph;
#include pgraph.ext
graphset;
x = seqa(1,1,5);
y = seqa(1,1,6);
Debugger: Breakpoints
Line Breakpoints
Proc Breakpoints
Add
Change
Delete
Clear
Update
Exit
d:\gauss_beta\examples\spline.e
d:\gauss_beta\examples\spline.e
d:\gauss_beta\examples\spline.e
36
25
16
PlayW
File Edit Draw Convert Help
Gauss Reference
Hide Locate Back Forward Print Options
Contents Search Favorites
call
cholup
chrs
cint
clear
clearg
Help
Hide Back Print Options
Contents Index Search
Introduction
Running Commands
Menu Bar
File Menu
Edit Menu
Search Menu
Mode Menu
Debug Menu
Action Menu
Configure Menu
Windows Menu
Help Menu
Tool Bar
Status Bar
Debug Window
Menu Bar
The Menu bar is located below the
bar along the top of the window. Y
can view the commands on a menu
either clicking the menu name or pr
ALT+n, where n is the underlined
in the menu name. For example, to
display the File menu, you can eith
click on File or press ALT+F.
The following menus are available:
File Menu
Edit Menu
Search Menu
Mode Menu
Debug Menu
ChangeDir
chdir
chol
choldn
cholsol
isSparse
K
\spline.e
Line: 3
Proc: <mps>
GAUSS: Executing
Command Edit Output Cmnd I/O Enter Ex: On L. 18 C. 22 NUM
Démarrer
Thierry Ron...
Explorateur -...
GAUSS
PlayW
PlayW
Help
Gauss Refer...
tmp3 - Paint
Debug Win...
08:44

1.1.2 GAUSS possède de nombreuses fonctions mathématiques

La version actuelle de GAUSS comprend plus de 300 fonctions mathématiques parmi les 800 commandes et opérateurs de base ! Les tableaux suivants présentent les commandes ordinaires (extraits de la formation CCF-DRI du mois d'octobre 1997) :

– Les opérateurs matriciels

`.'`	`B = A.';`	opérateur de transposition des éléments d'une matrice
`'`	`B = A';`	opérateur de transposition matricielle ($A^\top$ pour une matrice réelle et A^* pour une matrice complexe)
`!`	`B = A!;`	opérateur factoriel
`.^`	`C = B.^A;`	opérateur exposant
`*`	`C = A*B;`	produit matriciel
`*~`	`C = A*~B;`	produit direct horizontal
`.*`	`C = A.*B;`	produit d'Hadamard $C = A \odot B$
`.*.`	`C = A.*.B;`	produit de Kronecker $C = A \otimes B$
`./`	`C = A./B;`	division élément par élément
`/`	`x = b/A;`	solution d'un système équations linéaires $Ax = b$
`+`	`C = A+B;`	addition matricielle
`-`	`C = A-B;`	soustraction matricielle
`|`	`C = A|B;`	concaténation verticale
`~`	`C = A~B;`	concaténation horizontale

– Les fonctions mathématiques

ABS	`y = abs(x);`	retourne la valeur absolue ou le module complexe de x
COS	`y = cos(x);`	retourne le cosinus de x
EXP	`y = exp(x);`	retourne l'exponentielle de x
GAMMA	`y = gamma(x);`	retourne la valeur de $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt$
LN	`y = ln(x);`	retourne le logarithme de x
SIN	`y = sin(x);`	retourne le sinus de x
SQRT	`y = sqrt(x);`	retourne la racine carrée de x
TAN	`y = tan(x);`	retourne la tangente de x

– Les commandes d'algèbre linéaire

CHOL	`P = chol(A);`	Calcule la décomposition de Cholesky $A = P^\top P$ avec P une matrice triangulaire supérieure
DET	`d = det(A);`	Calcule le déterminant d'une matrice carrée
DFFT	`y = dfft(x);`	Calcule la transformée discrète de Fourier
DFFTI	`x = dffti(x);`	Calcule la transformée discrète inverse de Fourier
EIGHV	`{va,ve} = eighv(M);`	Calcule les valeurs propres et les vecteurs propres d'une matrice hermitienne
EIGV	`{va,ve} = eigv(M);`	Calcule les valeurs propres et les vecteurs propres d'une matrice carrée
FFT	`y = fft(x);`	Calcule la transformée rapide de Fourier
FFTI	`x = ffti(x);`	Calcule la transformée rapide inverse de Fourier
GRADP	`y = gradp(&f,x);`	Calcule le Jacobien de la fonction f en x
HESSP	`y = hessp(&f,x);`	Calcule la matrice Hessienne de la fonction f en x
INV	`B = inv(A);`	Calcule l'inverse d'une matrice carrée $B = A^{-1}$
INVPD	`B = invpd(A);`	Calcule l'inverse d'une matrice p.d.s. $B = A^{-1}$
PINV	`B = pinv(A);`	Calcule l'inverse généralisée de Moore-Penrose $B = A^+$
POLYCHAR	`c = polychar(A);`	Calcule le polynôme caractéristique de la matrice carrée A
POLYMULT	`c = polymult(c1,c2);`	multiplication polynômiale
POLYROOT	`r = polyroot(c);`	racines d'un polynôme

– Les commandes de manipulation de matrices

COLS	`n = cols(A);`	nombre de colonnes de la matrice A
CUMSUMC	`y = cumsumc(x);`	sommes cumulées des colonnes d'une matrice
COMPLEX	`z = complex(x,y);`	création d'une matrice complexe $z = x + iy$
DIAG	`d = diag(A);`	diagonale de la matrice A
EYE	`I = eye(n);`	matrice identité
IMAG	`y = imag(z);`	partie imaginaire d'une matrice complexe
MAXC	`m = maxc(x);`	vecteur des valeurs maximales de chaque colonne de la matrice x
MAXINDC	`pos = maxindc(x);`	vecteur des positions des valeurs maximales
MINC	`m = minc(x);`	vecteur des valeurs minimales de chaque colonne de la matrice x
MININDC	`pos = minindc(x);`	vecteur des positions des valeurs minimales
MISS	`y = miss(x,v);`	remplace dans la matrice x la valeur v par une donnée manquante
MISSRV	`y = missrv(x,v);`	remplace dans la matrice x les données manquantes par la valeur v
ONES	`A = ones(m,n);`	création de la matrice $\mathbf{1}_{m\times n}$
PACKR	`y = packr(x);`	élimination des lignes de la matrice x contenant des données manquantes
REAL	`x= real(z);`	partie réelle d'une matrice complexe
REV	`y = rev(x);`	renversement des lignes de la matrice
ROWS	`m = rows(A);`	nombre de lignes de la matrice A
SUMC	`s = sumc(x);`	sommes cumulées des colonnes d'une matrice
VEC	`v = vec(x);`	opérateur vec
VECH	`v = vech(x);`	opérateur $vech$
TRIMR	`y = trimr(x,a,b);`	élimination des a premières et b dernières lignes
ZEROS	`A = zeros(m,n);`	création de la matrice $\mathbf{0}_{m\times n}$

– GAUSS possède aussi un certain nombre de commandes relativement spécialisées, par exemple les transformées de Fourier multidimensionnelles, les opérateurs de matrices bandes ou creuses, les fonctions de Bessel

(premier ou second ordre, modifiées ou non, exponentielles, etc.), les formes Hessenberg, les décompositions de Schur, de Crout, etc.

```
/*
** Extraits de la bibliothèque GRO
** du Groupe de Recherche Opérationnelle
**
*/

new;

let Mcov[3,3] = 1.0 0.9 0.5
                0.9 2.0 1.0
                0.5 1.0 0.5;

sigma = sqrt(diag(Mcov)); /* :: Extrait la diagonale de la matrice de covariance
                             :: Construction du vecteur des écart-types          */


Mcor = Mcov ./ sigma ./ sigma';   /* Matrice de corrélation non PDS ! */

A = Mcov;

N = rows(A);
{T,Q} = schtoc(schur(A));      /* :: Décomposition réelle de Schur
                                  :: Forme complexe de Schur       */

B = Q * diagrv(zeros(N,N),sqrt(diag(T))) * Q';  /* :: Construction de la matrice racine carré
                                                   :: de la matrice de covariance à partir
                                                   :: de la forme Schur                        */


Mcov2 = real(B)*real(B);   /* Matrice de pseudo-covariances */

sigma2 = sqrt(diag(Mcov2));
Mcor2 = Mcov2 ./ sigma2 ./ sigma2';   /* Matrice de corrélation PDS ! */

print Mcor;

print Mcor2;


       1.0000000       0.63639610       0.70710678
      0.63639610        1.0000000        1.0000000
      0.70710678        1.0000000        1.0000000

       1.0000000       0.63637218       0.70477453
      0.63637218        1.0000000       0.99574161
      0.70477453       0.99574161        1.0000000
```

L'exemple suivant est extrait du manuel de la bibliothèque GRO. Le programme consiste à estimer un modèle circulaire par maximum de vraisemblance. Les données endogènes sont simulées afin que la transformation arctangente du modèle linéaire corresponde à la moyenne d'une variable aléatoire de Von Mises.

```
/*
**  N.I. Fisher [1993], Statistical Analysis of Circular Data
**  Cambridge University Press, New York
**
*/
```

```
new;
library gro,optmum;
Libgro;

cls;

rndseed 34534567;

nobs = 500;
numInd = 2;
mu = 0;
c = 0;
kappa = 1;

b = .5 * ones(numInd,1);
x = rndu(nobs,numInd);
y = rndvm(nobs,1,mu + G(c+x*b),kappa);

proc G(u);
   retp(2*atan(u));
endp;


proc ml(theta);
  local kappa,mu,c,b;
  local u,logl;

  kappa = theta[1];
  mu = theta[2];
  c = theta[3];
  b = theta[4:rows(theta)];

  u = y - mu - G(c + x*b);

  logl = -ln(mbesselei0(kappa)) + kappa * (cos(u) - 1);

  retp(logl);
endp;


/*
** Reparamétrisation
**
** kappa > 0
*/

proc ml2(theta);
  local kappa,mu,c,b;

  kappa = exp(theta[1]);
  mu = theta[2];
  c = theta[3];
  b = theta[4:rows(theta)];

  retp(ml(kappa|mu|c|b));
endp;
```

```
sv = 5*ones(NumInd+3,1);
_reg_PrintIters = 0;            /* Ne pas afficher les itérations de l'optimisation */
_reg_Mcov = 0;                  /* Ne pas calculer la matrice de covariance         */
__output = 0;                   /* Ne pas afficher les résultats                    */
{theta,stderr,Mcov,Logl} = regML(0,0,&ml2,sv,0,0);

kappa = exp(theta[1]); mu = theta[2]; c = theta[3]; b = theta[4:rows(theta)]; sv = kappa|mu|c|b;

output file = ml1.out reset;

_reg_Mcov = 1;                /* Matrice de covariance de type I               */
_reg_NoIter = 1;              /* Pas d'optimisation puisque l'optimum est connu */
__output = 1;
let _ParNames = "kappa" "mu" "c";
_ParNames = _ParNames | (0 $+ "b" $+ ftocv(seqa(1,1,NumInd),1,0));
__title = "A circular regression model";

{theta,stderr,Mcov,Logl} = regML(0,0,&ml,sv,0,0);

output off;
```

```
================================================================================
                          A circular regression model
================================================================================
 regML - Maximum Likelihood                                 4/05/2000   1:19 pm
================================================================================


Total observations:                                            500
Missing observations:                                            0
Usable observations:                                           500
Valid cases:                                                   500

Number of parameters:                                            5
Number of unrestricted parameters:                               5
Degrees of freedom:                                            495

Value of the maximized log-likelihood function:          109.73290
Mean log-likelihood:                                       0.21947


Parameters      estimates       std.err.      t-statistic         p-value
--------------------------------------------------------------------------
kappa           1.025257        0.075734        13.537582        0.000000
mu              5.580764        0.554264        10.068789        0.000000
c               0.177968        0.221188         0.804598        0.421438
b1              0.445208        0.275451         1.616286        0.106670
b2              1.742654        1.075136         1.620869        0.105683

Covariance matrix: inverse of the negative Hessian.
```

1.1.3 Les variables globales sont déclarées par défaut

Dans GAUSS, il n'est pas nécessaire de déclarer les variables ainsi que l'allocation mémoire.

```
new;

x = seqa(1,1,100);       /* Pas d'allocation mémoire pour la variable X */
x = x~x;
```

```
y = x[.,1] + x[.,2];

clear x;

y = y|y;                /* Ré-eallocation mémoire automatique */

y = y + seqa(0,1,5)';
```

1.1.4 La gestion des fichiers sources est largement facilitée par l'utilisation de bibliothèques de procédures

La procédure suivante est extraite de la bibliothèque GRO. Elle permet d'analyser les dépendences linéaires dans les problèmes de régression linéaire, d'analyse de variance ou encore de maximum de vraisemblance.

- **Une procédure est un corps de programme indépendant.**
- **Une bibliothèque est une collection de procédures (et aussi de variables externes).**
- **Pour utiliser les procédures d'une bibliothèque, il suffit de déclarer celle-ci en tête du programme.**

```
proc (4) = LinearDep(m,tol);
  local rr,ee,nirr,njrr,depcol,indepcol,Xi,tols;
  local col,row,omat,mask,fmt,N;

  if tol <= 0;
    tol = 1e-13;
  endif;

  {rr,ee} = qre(m);

  tols = abs(diag(rr)./sqrt(diag(m[ee,ee])));
  nirr = sumc(tols .> tol );
  njrr = rows(rr) - nirr;

  if njrr > 0 and nirr > 0;

    indepcol = ee[1:nirr];
    depcol = ee[rows(ee)-njrr+1:rows(ee)];
    Xi = utrisol(rr[1:nirr,nirr+1:rows(rr)],rr[1:nirr,1:nirr])';
    tols = tols[rows(ee)-njrr+1:rows(ee)];

    if __output;

      call header("LinearDep - Linear Dependencies Analysis","",0);
      print;

      if ( _ParNames /= 0 ) and ( rows(_ParNames) == rows(rr));
        col = _Parnames[depcol];
        row = _Parnames[indepcol];
      else;
        col = 0 $+ "#" $+ ftocv(depcol,1,0);
        row = 0 $+ "#" $+ ftocv(indepcol,1,0);
      endif;
      omat = dotfne(Xi',0) .* Xi';

      N = rows(col);
      mask = zeros(1,N+1);
      fmt = "-*.*s"~8~8|((zeros(N,1)$+("*.*s"))~(zeros(N,1)+(16~9)));
      call printfm(""~col',mask,fmt);
```

```
      print;

      mask = 0~ones(1,N);
      omat = row ~ omat;
      fmt = "-*.*s"~8~8|((zeros(N,1)$+("*.*lf"))~(zeros(N,1)+(16~4)));
      call printfm(omat,mask,fmt);
      print "_____________________________________________";
      omat = (0$+"tol:") ~ tols';
      fmt = "-*.*s"~8~8|((zeros(N,1)$+("*.*lE"))~(zeros(N,1)+(16~4)));
      call printfm(omat,mask,fmt);
      print;

    endif;

  else;

    depcol = error(0);
    indepcol = error(0);
    Xi = error(0);

  endif;

  retp(depcol,indepcol,Xi,tols);
endp;
```

L'exemple suivant teste la colinéarité de variables exogènes dans un modèle linéaire. Nous considérons 5 variables avec

$$\begin{aligned} x_4 &= x_1 - x_2 \\ x_5 &= x_1 + x_2 + x_3 \end{aligned}$$

```
new;
library gro;

rndseed 123;

x = rndu(1000,3);
x = x ~ (x[.,1]-x[.,2]) ~ (x[.,1]+x[.,2]+x[.,3]);

beta = 1|2|3|4|5;

y = x*beta + rndn(1000,1)*0.25;

output file = gauss5.out reset;

call ols(0,y,x);

H = x'x;  /* Matrice hessienne du critère des moindres carrés */

{depcol,indepcol,Xi,tols} = LinearDep(H,1e-5);

output off;

ERROR: covariance matrix of independent variables is singular.
================================================================================
 LinearDep - Linear Dependencies Analysis                  4/05/2000   2:01 pm
================================================================================
```

```
                    #1              #2
#5                0.5000          0.5000
#4                0.5000         -0.5000
#3               -0.5000         -0.5000

-----------------------------------------
tol:         6.3610E-015     2.5633E-015
```

La procédure `LinearDep` donne 2 dépendances :

$$\begin{aligned} x_1 &= \frac{1}{2}x_5 + \frac{1}{2}x_4 - \frac{1}{2}x_3 \\ x_2 &= \frac{1}{2}x_5 - \frac{1}{2}x_4 - \frac{1}{2}x_3 \end{aligned}$$

Nous retrouvons bien les deux dépendances puisque nous avons :

$$\begin{aligned} x_1 - x_2 &= x_4 \\ x_1 + x_2 &= x_5 - x_3 \end{aligned}$$

1.1.5 GAUSS intègre une bibliothèque graphique très riche

1.1.5.1 Les graphiques 2D $\Longrightarrow$ `XY, BAR, BOX, LOGX, LOGY, LOGLOG, POLAR, HIST, HISTF, HISTP`

1.1.5.2 Les graphiques 3D $\Longrightarrow$ `CONTOUR, XYZ, SURFACE`

1.1.5.3 Le mode fenêtre $\Longrightarrow$ `BEGWIND, WINDOW, MAKEWIND, SETWIND, LOADWIND, GETWIND`

Les symboles
$\Longrightarrow$ `_PSYM, _PSYM3D, _PARROW, _PARROW3D, _PLINE, _PGRID, _PMSGSTR, _PMSGCTL`

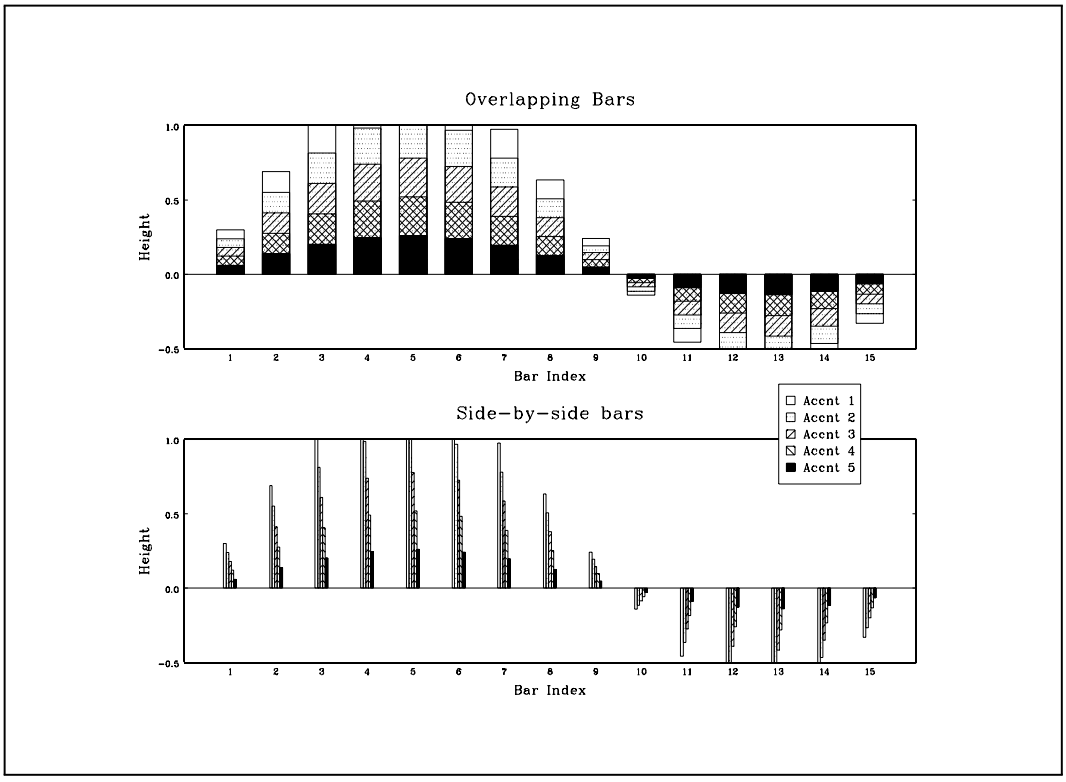


FIG. 1:

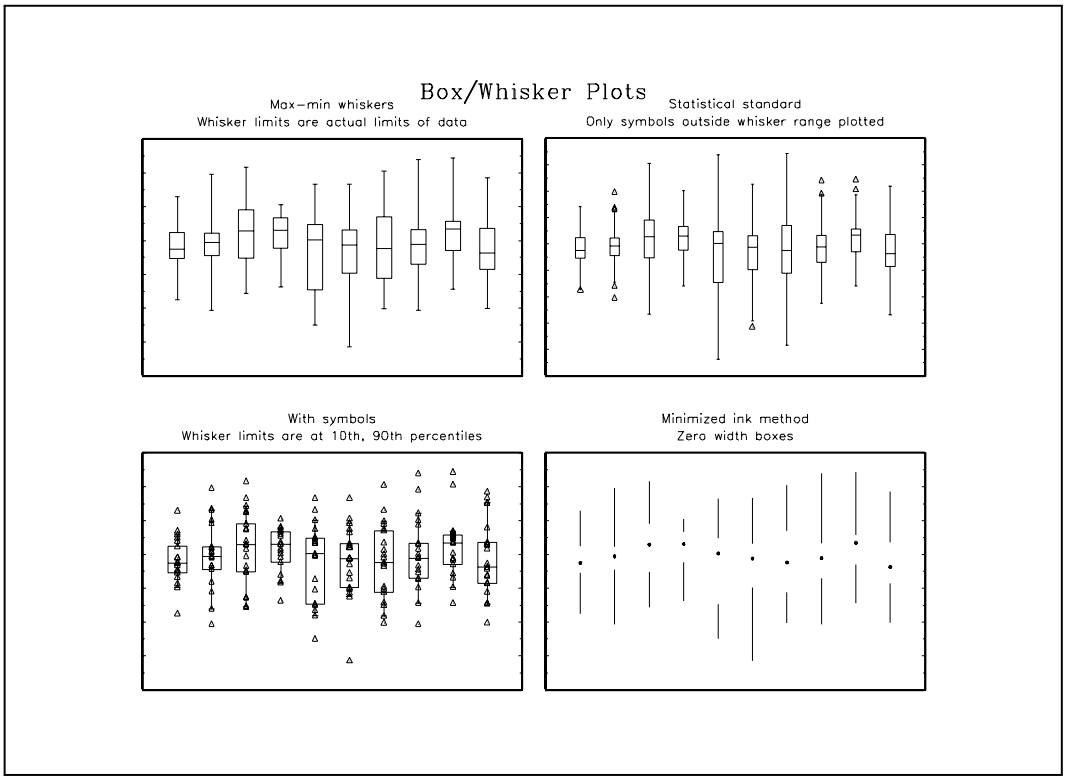

Box/Whisker Plots
Max−min whiskers
Whisker limits are actual limits of data
Statistical standard
Only symbols outside whisker range plotted
With symbols
Whisker limits are at 10th, 90th percentiles
Minimized ink method
Zero width boxes

FIG. 2:

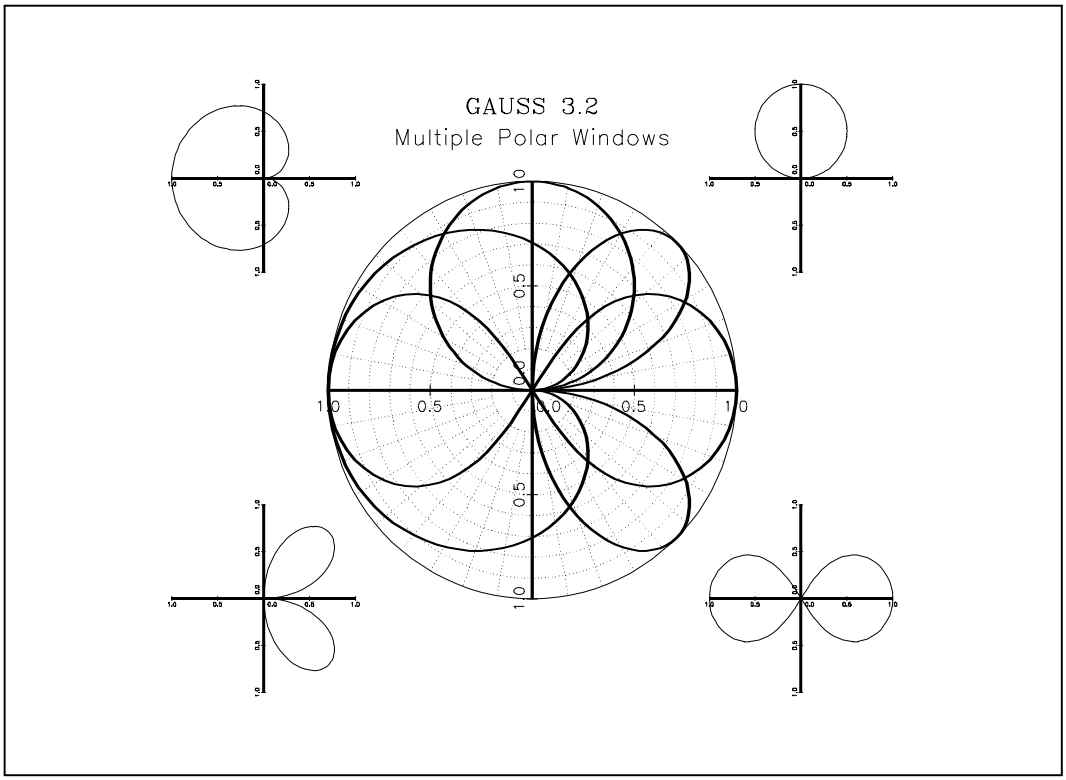

GAUSS 3.2
Multiple Polar Windows

FIG. 3:

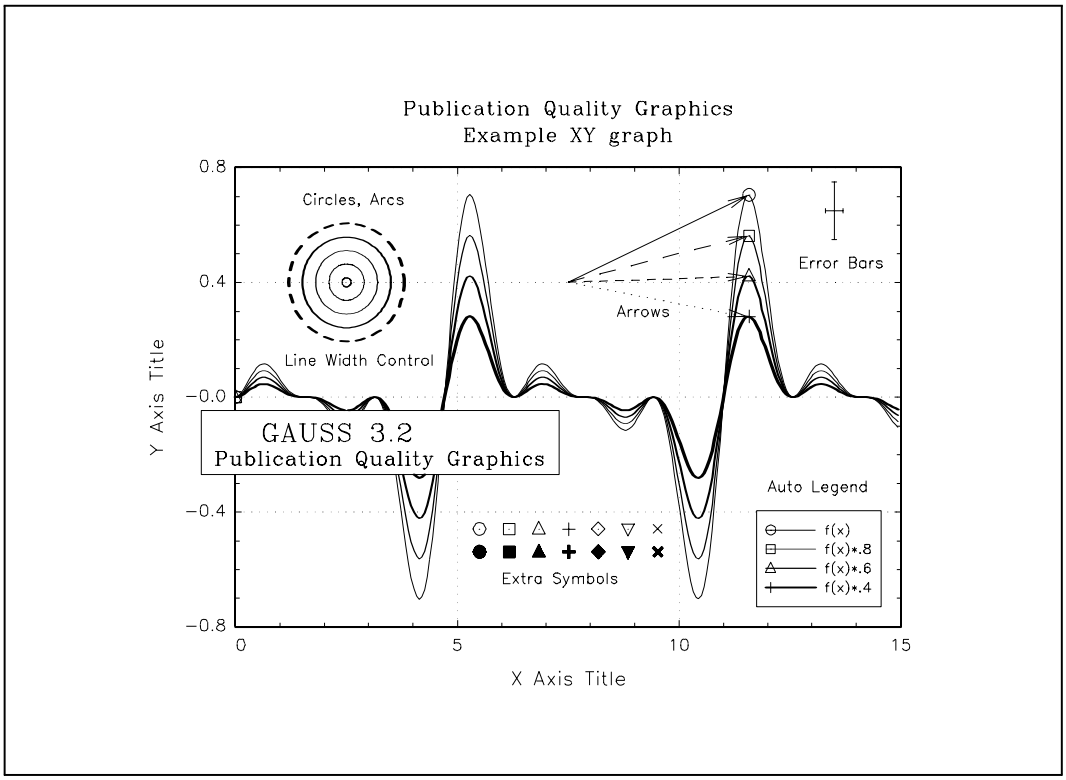

Publication Quality Graphics
Example XY graph
Circles, Arcs
Line Width Control
Arrows
Error Bars
GAUSS 3.2
Publication Quality Graphics
Auto Legend
f(x)
f(x)*.8
f(x)*.6
f(x)*.4
Extra Symbols
Y Axis Title
X Axis Title
0.8
0.4
−0.0
−0.4
−0.8
0
5
10
15

FIG. 4:

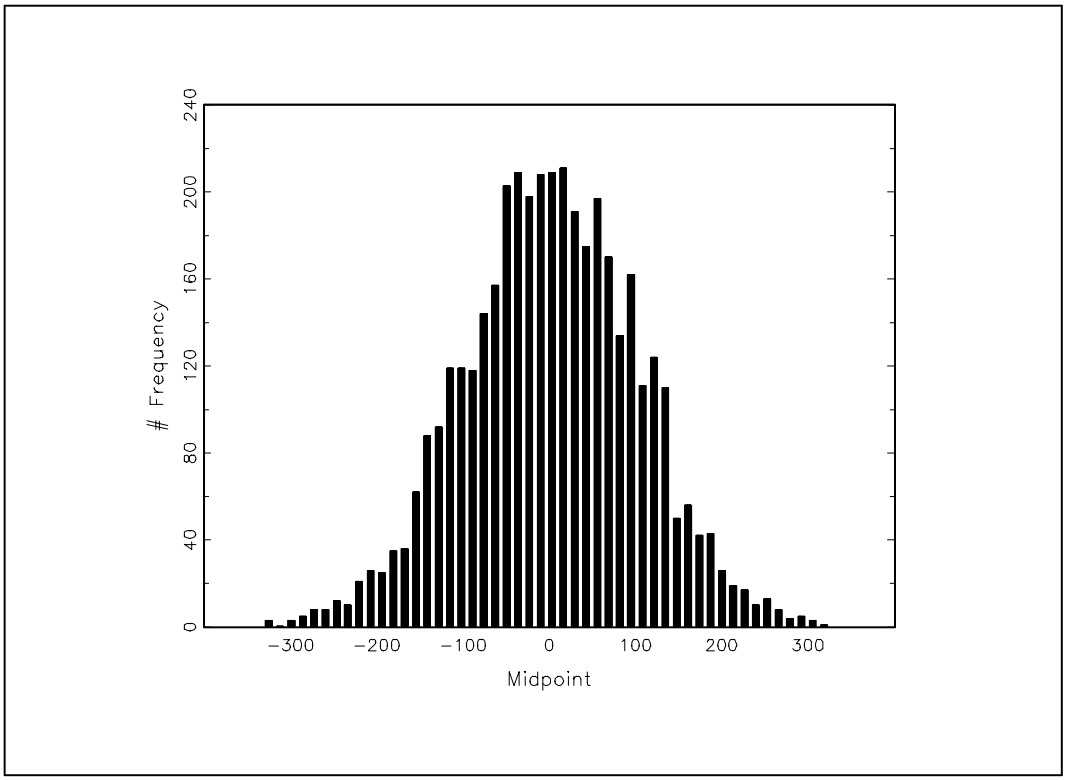

Frequency
Midpoint
0
40
80
120
160
200
240
−300
−200
−100
0
100
200
300

FIG. 5:

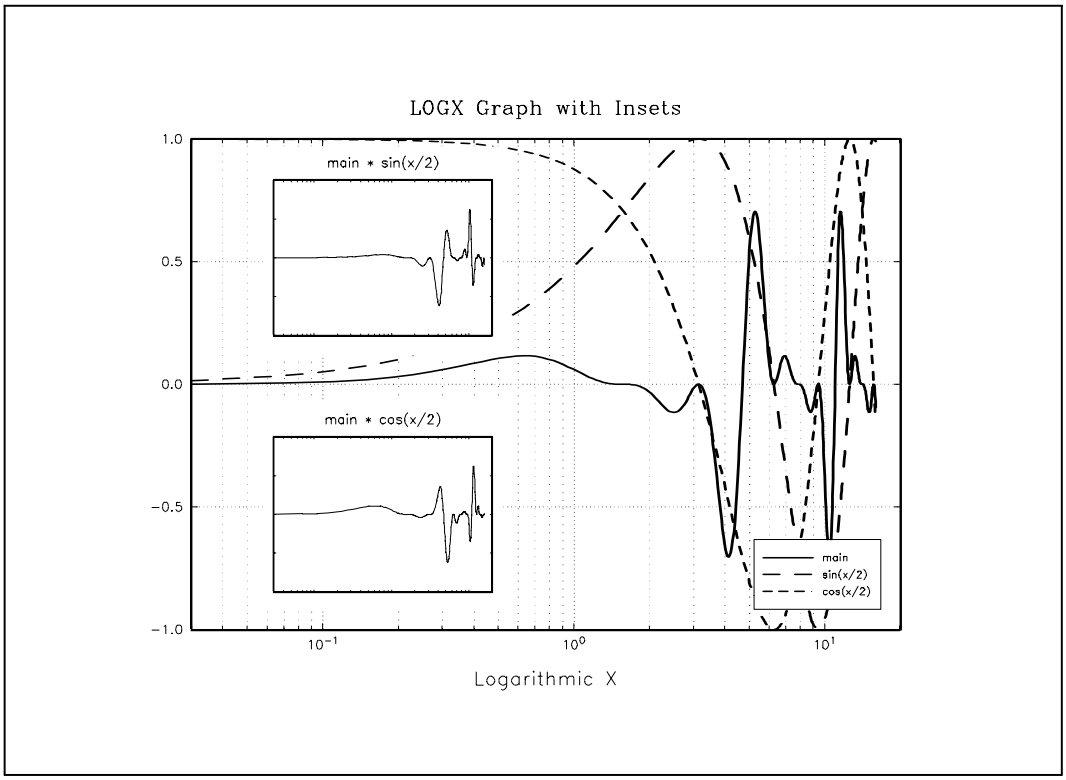

LOGX Graph with Insets
main * sin(x/2)
main * cos(x/2)
main
sin(x/2)
cos(x/2)
1.0
0.5
0.0
−0.5
−1.0
10^{-1}
10^{0}
10^{1}
Logarithmic X

FIG. 6:

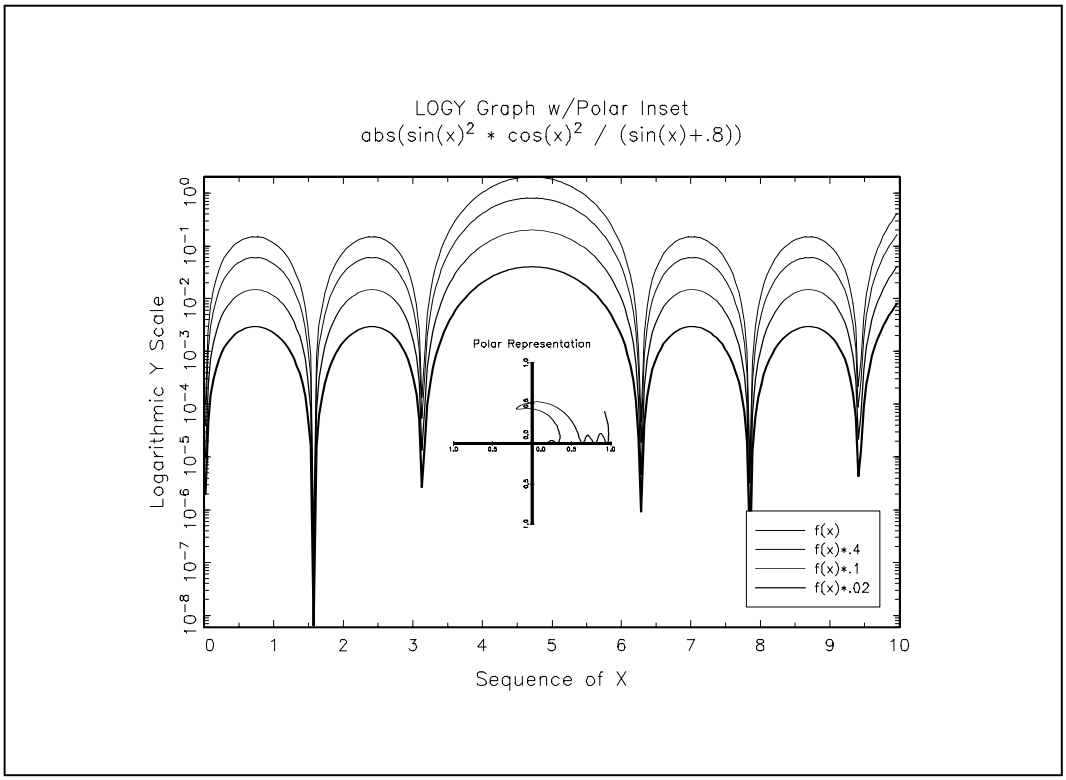

LOGY Graph w/Polar Inset
abs(sin(x)² * cos(x)² / (sin(x)+.8))
Polar Representation
f(x)
f(x)*.4
f(x)*.1
f(x)*.02
Logarithmic Y Scale
10^{-8} 10^{-7} 10^{-6} 10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^{0}
0 1 2 3 4 5 6 7 8 9 10
Sequence of X

FIG. 7:

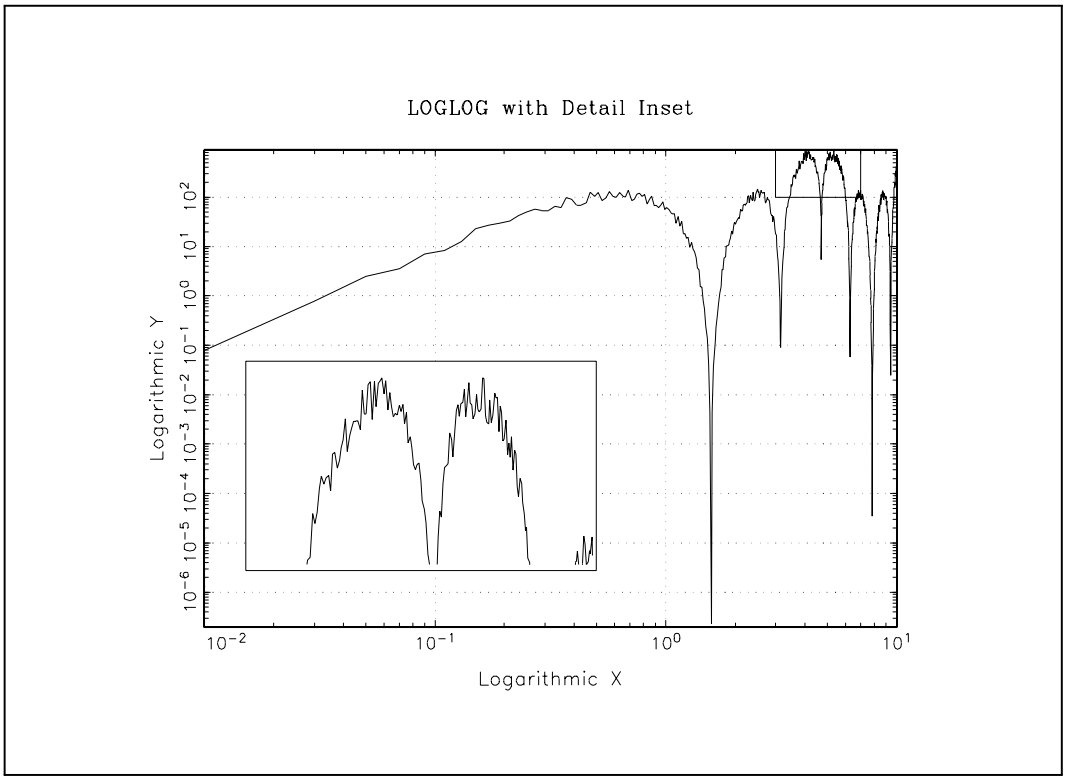

LOGLOG with Detail Inset
Logarithmic Y
Logarithmic X
10^{-6} 10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2}
10^{-2} 10^{-1} 10^{0} 10^{1}

FIG. 8:

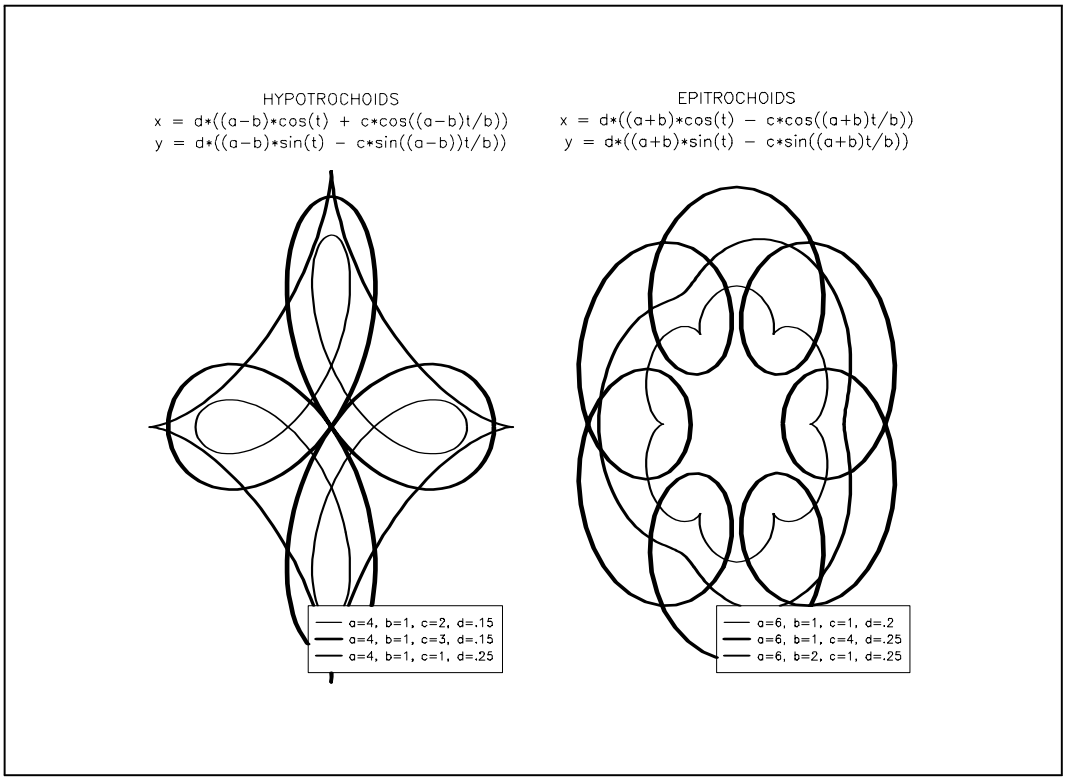

HYPOTROCHOIDS
x = d*((a−b)*cos(t) + c*cos((a−b)t/b))
y = d*((a−b)*sin(t) − c*sin((a−b)t/b))
a=4, b=1, c=2, d=.15
a=4, b=1, c=3, d=.15
a=4, b=1, c=1, d=.25
EPITROCHOIDS
x = d*((a+b)*cos(t) − c*cos((a+b)t/b))
y = d*((a+b)*sin(t) − c*sin((a+b)t/b))
a=6, b=1, c=1, d=.2
a=6, b=1, c=4, d=.25
a=6, b=2, c=1, d=.25

FIG. 9:

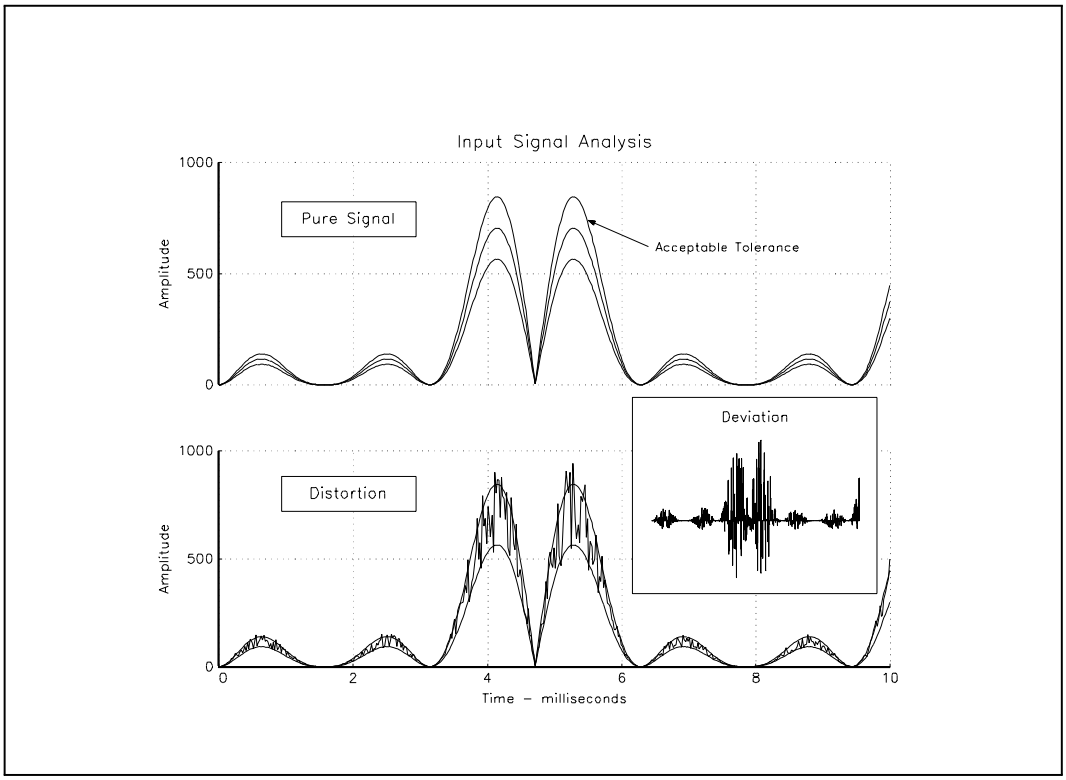

Input Signal Analysis
Pure Signal
Acceptable Tolerance
Deviation
Distortion
Amplitude
Amplitude
1000
500
0
1000
500
0
0
2
4
6
8
10
Time – milliseconds

FIG. 10:

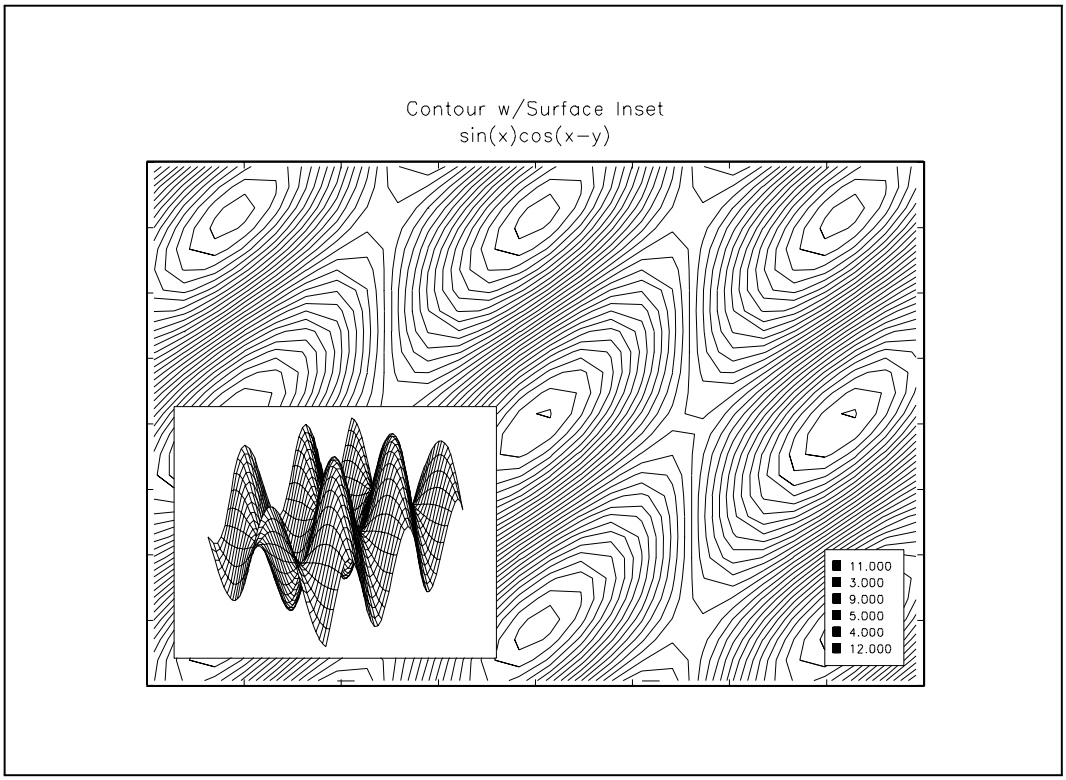

Contour w/Surface Inset
sin(x)cos(x−y)
11.000
3.000
9.000
5.000
4.000
12.000

FIG. 11:

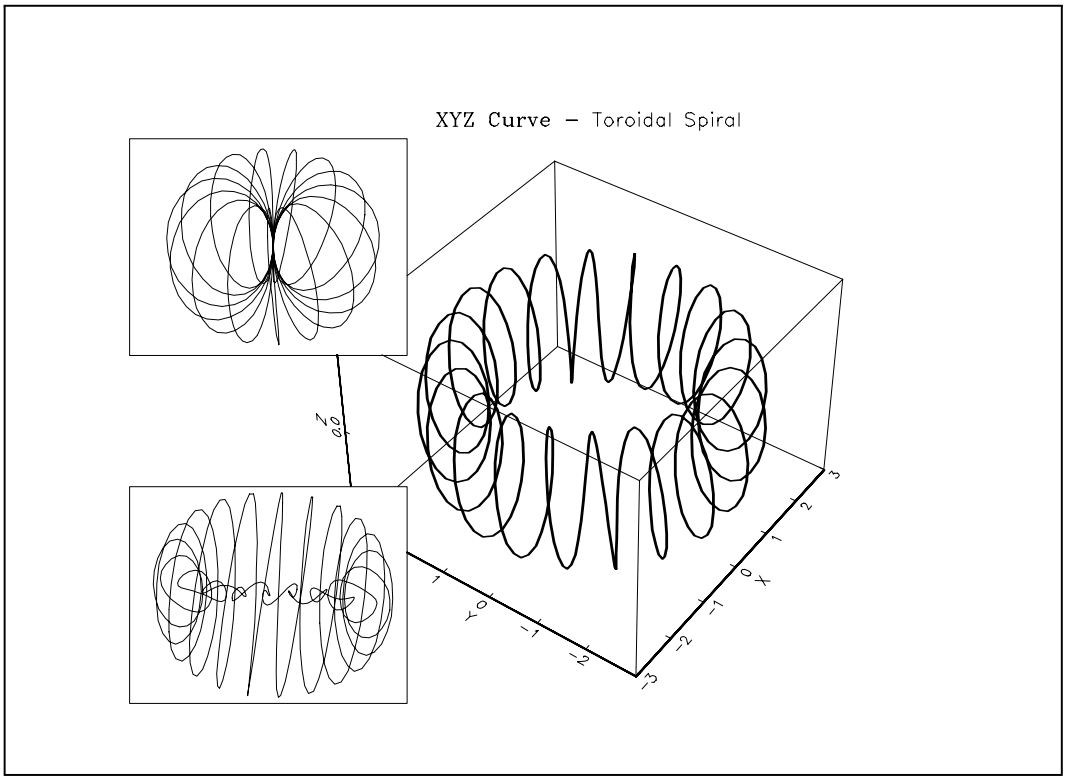


FIG. 12:

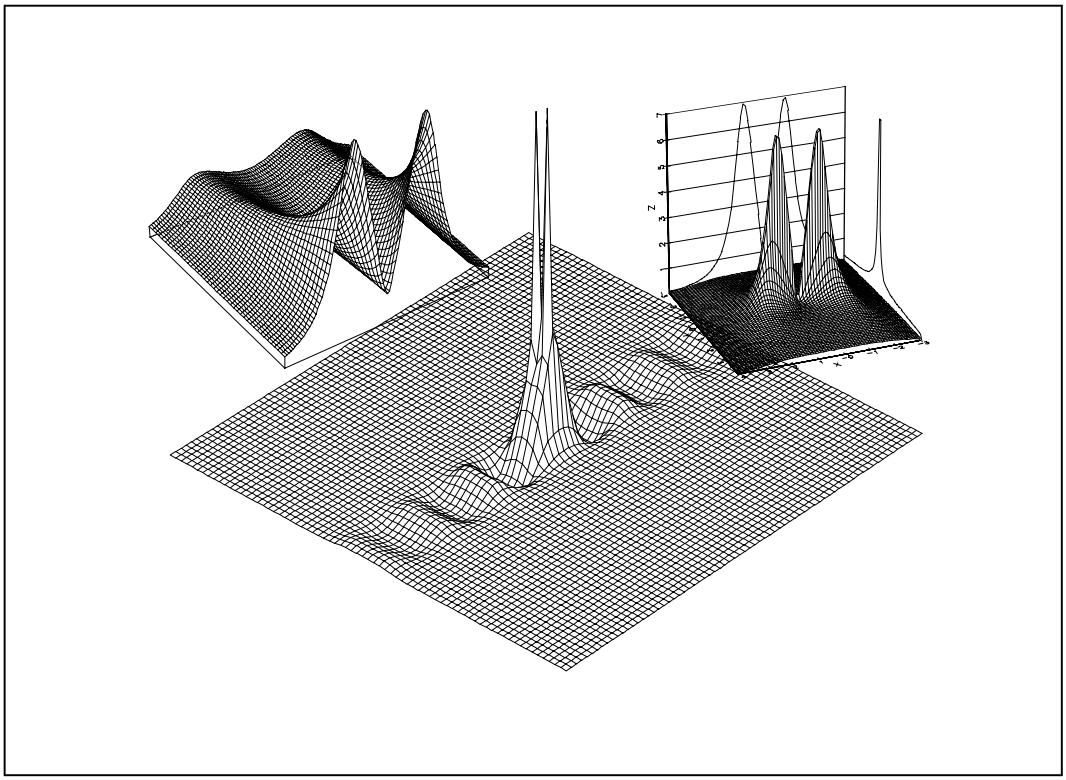


FIG. 13:

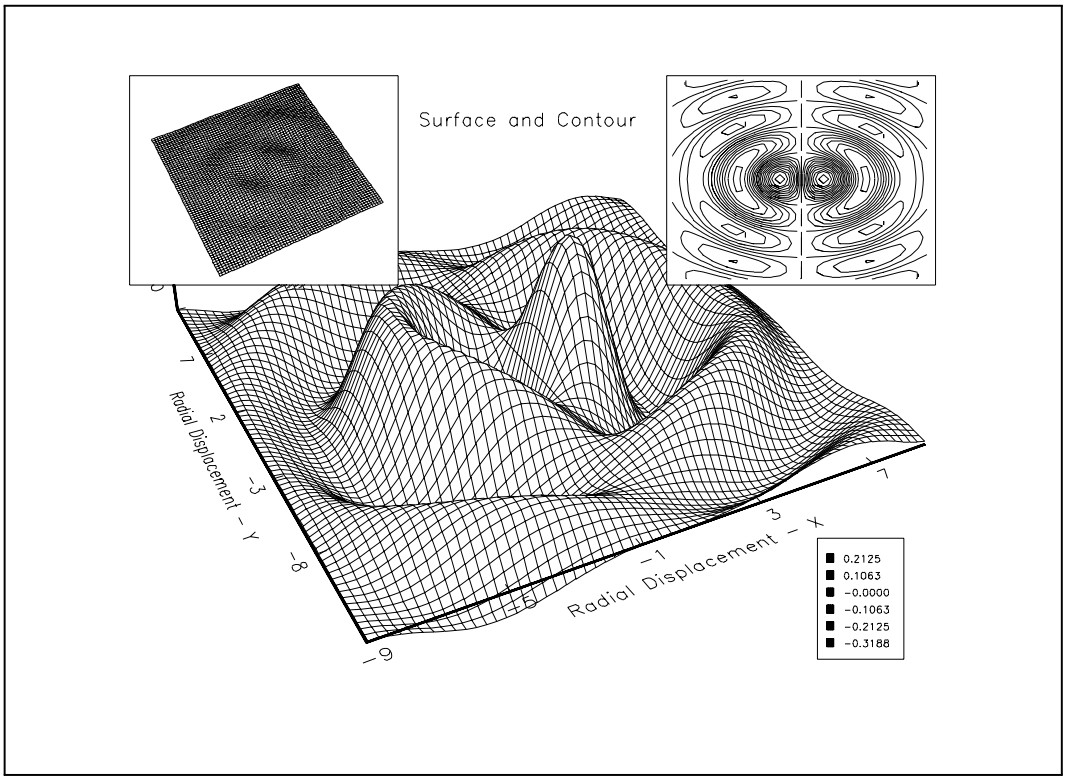

Surface and Contour
Radial Displacement - Y
Radial Displacement - X
0.2125
0.1063
-0.0000
-0.1063
-0.2125
-0.3188

FIG. 14:

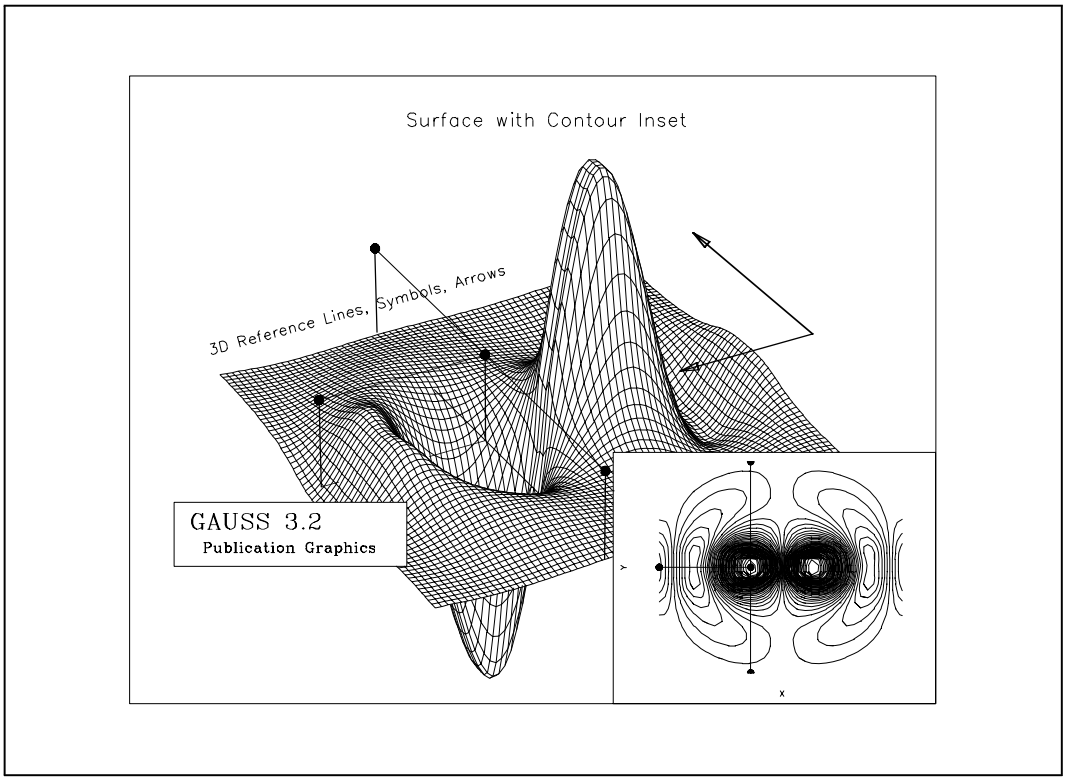

Surface with Contour Inset
3D Reference Lines, Symbols, Arrows
GAUSS 3.2
Publication Graphics

FIG. 15:

1.2 Présentation des modules

Le lecteur trouvera des informations plus précises sur les modules aux adresses suivantes :

http : //www.aptech.com
http : //www.aptech.com/apps.html
http : //www.aptech.com/3party.html
http : //www.scientificweb.com/software/aptech/gaussmoe.html

FANPAC a fait l'objet de présentation lors des précédents séminaires :

http : //www.city.ac.uk/cubs/ferc/thierry/conf2.pdf
http : //www.city.ac.uk/cubs/ferc/thierry/conf3.pdf

Une page WEB lui est aussi consacrée

http : //www.aptech.com/fp11.html
http : //www.aptech.com/fankey/fankey.html

Ron Schoenberg a écrit 3 articles sur les modules CO et CML :

http : //faculty.washington.edu/rons/articles/wlb.ps
http : //faculty.washington.edu/rons/articles/cmlinf.ps
http : //faculty.washington.edu/rons/articles/co.ps

Pour plus d'information sur IGX, vous pouvez consulter la page

http ://www.econotron.com/igx/igx.html

Bibliothèque DSTAT

```
new;
library dstat;
dstatset;

cls;

dataset = ''freq'';

__title = ''CT1.E:  Three way table'';
_dstat = 1;
_dspause = 0;
{ t,n } = crosstab(dataset,''AGE PAY sex'');

__title = ''FR1.E:  without weights'';
{ cats,ncats,freqs } = freq(dataset,1|2|3);


__title = ''TTEST2.E:  Test of difference of means by dichotomizing variable'';

dataset = ''scigau'';
vars = ''cit1 pub1'';
grpvar = { job };
_dsgrpnm = { Lo_Job, Hi_Job };
__miss = 1;
__row = 100;
_dscut = 2;
{ v,desc,mtest,vtest } = ttest(dataset,grpvar,vars);
```

Bibliothèque QUANTAL

```
new;
library quantal;
#include quantal.ext;
quantset;

cls;

/*
** LGTALD3.E:  Logit analysis of the Aldrich and Nelson data for a
**             trichotomous dependent variable.
*/

__title=''LGTALD3.E:  Logit Analysis for a Trichotomous Variable'';
dsn = ''aldnel'';
_qriter = 1;
_qrstat = 1;
dv = 1;
iv = { 2, 3, 4 };
_qrcatnm = { GR=A, GR=B, GR=C };
{ vnam,b,vc,n,pct,mn,sd,fit,df,tol } = logitprt(logit(dsn,dv,iv));


/*
**  QRTEST.E: Test of linear hypothesis of logit model
*/

_qrcatnm = { GRD=A, GRD=B, GRD=C };
dsn = ''aldnel'';
dv = { abc };
iv = { gpa, tuce, psi };
{ vnam,b,vc,n,pct,mn,sd,fit,df,tol } = logit(dsn,dv,iv);

test1 = ''gpa:2 + tuce:2 = 0'';
{ wald1 } = qtest(vnam,b, vc,test1);

test2 = ''gpa:1 - 2.5tuce:1 = 2,
        tuce:1 + psi:1 = 0,
        3gpa:2 + 2tuce:1 - psi:2 = 2'' ;
{ wald2 } = qtest(vnam,b, vc,test2);


/*
** OLALDNEL.E:  Ordinal Logit Analysis of the Aldrich and Nelson Data.
*/


__title=''OLALDNEL.E:  Ordinal Logit Analysis of Aldrich and Nelson Data'';
__range = { 0, 27 };
__weight = { aplus1 };
dataset = ''aldnel'';
depvar = 1;
indvars = { 2, 3, 4 };
_qrcatnm = { A, B, C };
_qrlogit = 1;
call ordered(dataset,depvar,indvars);
```

Bibliothèque LOGLIN

```
new;
```

```
library loglin;
#include loglin.ext;
llset;

cls;

/*
** lluser.e:  Agresti, page 87
*/

loadm agr87t,agr87n;
agr87d =
      { 1  1  0  1  0 -1  0,
        1  1  0  0  1  0  0,
        1  1  0 -1 -1  1  0,
        1  0  1  1  0  0 -1,
        1  0  1  0  1  0  0,
        1  0  1 -1 -1  0  1,
        1 -1 -1  1  0  1  1,
        1 -1 -1  0  1  0  0,
        1 -1 -1 -1 -1 -1 -1 };


lbl = { P1, P2, P3, P4, P5, Tau1, Tau2 };
__title=''lluser.e:  Row Effects Model from Agresti, page 87.'';
call lluser(agr87t,agr87n,0,agr87d,0,lbl);

/*
** ll3way.e:  From Fienberg, 2nd Edition, page 42
*/

__title=''ll3way.e:  From Fienberg, 2nd Edition, page 42.'';
loadm fien27t,fien27n;
call ll3way(fien27t,fien27n,0);

/*
** llhier1.e:  Fienberg, 2nd Edition, page 27.
*/

loadm fien27t,fien27n;
output file = llhier1.out reset;

letters(''S D H'');
nm = { SPECIES, DIAMETER, HEIGHT };

__title=''Solution 1:  ANOVA-like constraints.'';
cfg = { SH, SD };
call llhier(fien27t,fien27n,0,cfg,nm);

__title=''Solution 2:  Regression-like constraints.'';
cfg = { SH, SD };
_llsum0 = 0;
call llhier(fien27t,fien27n,0,cfg,nm);

output off;
```

Bibliothèque MAXLIK

```
/*
**   maxlik7.e  -  A bootstrapped Tobit model
```

```
**
*/

new;
library maxlik,pgraph;
#include maxlik.ext;
#include pgraph.ext;
graphset;
maxset;

proc lpr(x,z);
    local t,s,m,u;
    s = x[4];
    if s <= 1e-4;
      retp(error(0));
    endif;
    m = z[.,2:4]*x[1:3,.];
    u = z[.,1] ./= 0;
    t = z[.,1]-m;
    retp(u.*(-(t.*t)./(2*s)-.5*ln(2*s*pi)) + (1-u).*(ln(cdfnc(m/sqrt(s)))));
endp;


output file = max1.out reset;

_max_Width = 4;  /* these estimates have fat tails */
_max_BootFname = ''boot7'';
__title = ''tobit example'';

x0 = { 1, 1, 1, 1 };
call DosWinOpen(''ft'',error(0));
{x,f,g,cov,ret} = MAXboot(''tobit'',0,&lpr,x0);
call DosWinCloseAll;

__title = ''std errors from bootstrapped covariance matrix'';
call maxprt(x,f,g,cov,ret);

__title = ''confidence limits from  bootstrapped coefficients'';
cl = MAXBLimits(''boot7'');
call MAXCLPrt(x,f,g,cl,ret);

printdos ''\n'';
printdos ''Press any key for density plots...'';
call wait;

call MAXdensity(''boot7'',0);

printdos ''\n'';
printdos ''Press any key for histogram and surface plots...'';
call wait;

call MAXhist(''boot7'',0);

output off;
```

Bibliothèque MAXLIK

```
/*
**   maxlik9.e   Likelihood Profile and Profile t traces for a
**               nonlinear least squares problem
```

```
**
**   The data for this example have been created using nlls.sim.
*/

new;
library maxlik,pgraph;
#include maxlik.ext;
#include pgraph.ext;
maxset;

@ -- procedure to compute log-likelihood -- @

proc lnlk(b,z);
    local dev,s2;
    dev = z[.,1] - (b[1] * exp(-b[2]*z[.,2]));
    s2 = dev'dev/rows(dev);
    retp(-0.5*(dev.*dev/s2 + ln(2*pi*s2)));
endp;

proc grdlk(b,z);
    local d,s2,dev,r;
    d = exp(-b[2]*z[.,2]);
    dev = z[.,1] - b[1]*d;
    s2 = dev'dev/rows(dev);
    r = dev.*d/s2;
    retp(r~(-b[1]*z[.,2].*r));
endp;

proc hslk(b,z);
    local d,s2,dev,r, hss;
    d = exp(-b[2]*z[.,2]);
    dev = z[.,1] - b[1]*d;
    s2 = dev'dev/rows(dev);
    r = -z[.,2].*d.*(b[1].*d + dev);
    hss = -d.*d/s2~r~b[1].*z[.,2].*r;
    retp(xpnd(sumc(hss)));
endp;


startv = { 2, 1 };
output file = maxlik9.out reset;

_max_GradProc = &grdlk;
_max_HessProc = &hslk;

call maxprofile(''nlls'',0,&lnlk,startv);

output off;
```

Bibliothèque TSM

```
new;
library tsm,optmum,pgraph;
TSMset;

load reinsel[100,2] = reinsel.asc;

invest = reinsel[.,1];
invent = reinsel[.,2];
di = invest - lag1(invest);
```

```
y = di~invent;

{lambda,Iy} = PDGM2(y);

/*
** Procedure to compute the sgf of the SSM
*/

proc sgf(theta,lambda);
  local beta,Pchol,SIGMA,T,Q,H,Z,d,c,R;
  local Gy;

  beta = theta[1:8];
  Pchol = (theta[9]~0)|(theta[10]~theta[11]);
  SIGMA = Pchol*Pchol';

  {Z,d,H,T,c,R,Q} = arma_to_SSM(beta,1,1,SIGMA);
  call SSM_build(Z,d,H,T,c,R,Q,0);
  Gy = sgf_SSM(lambda);

  retp(Gy);
endp;


proc ml(theta);
  local Gy,Nstar,logl,j,Ij,Gj;

  Gy = sgf(theta,lambda);
  Nstar = rows(lambda);

  logl = zeros(Nstar,1);

  j = 1;
  do until j > Nstar;
   Ij = xpnd2(Iy,j);
   Gj = xpnd2(Gy,j);
   logl[j] = -0.5*ln(det(Gj)) -0.5*sumc(diag((inv(Gj)*Ij)));
   j = j + 1;
  endo;

  logl = real(logl);

  retp(logl);
endp;

load ARMA1;
load SIGMA1;

sv = arma1|vech_(chol(SIGMA1));    /* Starting values */

output file = fdml6.out reset;

_tsm_Mcov = 1;
{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;
```



```
/*
```

```
**  MSFT.E
**
**  GARCH analysis of two years of monthly returns on
**  Microsoft stock.  In the first analysis, two lagged
**  values are included in the equation, and in the
**  second the lagged values are excluded.
*/

new;
library fanpac,pgraph;

session msft 'daily Microsoft';
setVarNames date price lvol price1 lvol1 price2 lvol2;
setDataSet msft.asc;
setSeries price;
setIndVars price1 price2;  /* lagged values of price */

plotSeries;
computeLogReturns 251;
estimate run1 garch(2,1);

setIndEqs none;
estimate run2 garch(2,1);

plotcv;
showresults;
testsr;
plotqq;
```

Bibliothèque FANPAC

```
/*
**> Diagonal Vec model
**
*/

new;
library fanpac,pgraph;

session dvec 'Diagonal Vec model';
setDataSet stocks;
setSeries CPWR SUNW;
setIndVars CPWRvol SUNWvol;

/*
** scale volume
*/

_fan_IndVars = 1e-4*_fan_IndVars;

setIndEqs CPWR CPWRvol;
setIndEqs SUNW SUNWvol;


computeLogReturns 251;
constrainPDCovPar on;


_fan_Start = {3.76567039,3.03950712,7.97138687,0.57138704,
              0.30145121,0.05782660,0.17241156,0.12793222,0.22254379,
```

```
            -0.35588323,0.03815826,0.44672175,0.05152276};

estimate run1 dvgarch(1,1);

showresults;

plotcv;
plotcor;
```

Bibliothèque DCK

(extrait du manuel de la bibliothèque GRO)

1. Principe et commandes de base

La bibliothèque **DCK** est basée sur l'interface **ODBC** (Open DataBase Connectivity). Elle contient un ensemble de fonctions permettant entre autre de se connecter à une base de données contenant éventuellement plusieurs tables (`ODBCAddDataSource`), d'exécuter des commandes (`ODBCExecuteDirect`) sur l'une des tables de cette base et de lire toutes ou une sélection des données d'une table `ODBCReadComplete`. Les commandes passées en argument de `ODBCExecuteDirect` doivent être écrites en langage **SQL** (Structured Query Language). Elles permettent d'effectuer entre autres les opérations élémentaires suivantes :

- créer une nouvelle table : `Create table ...`
- ajouter des observations à une table existante : `Insert into ... values ...`
- supprimer des observations et/ou des variables : `Delete ... from ... where ...`
- supprimer une table : `drop ...`
- modifier une variable pour certaines observations : `update ... set ... where ...`
- sélectionner certaines variables et/ou observations : `select ... from ... where ...`

2 Exemple d'utilisation

Cet exemple utilise la base de données Access *Ecce.mdb*. Cette base de 45 Mo contient 4 tables. L'une d'elle, `devises`, fait correspondre à une monnaie donnée un code numérique. Une autre, `eccelot1`, est un échantillon d'entreprises. On dispose pour chaque entreprise de plusieurs renseignements dont la devise en laquelle est libellé son compte. Le programme suivant se déroule de la manière suivante :

1. l'utilisateur rentre une devise ;
2. GAUSS utilise l'interface ODBC pour interroger la table `devises` et trouver le code correspondant à cette devise ;
3. Il interroge ensuite la table `eccelot1` pour trouver les sirens des entreprises dont la variable `devise` correspond à cette entreprise ;
4. Il lit ces sirens ;
5. Il crée une nouvelle table dans *Ecce.mdb* et y inscrit les sirens ;

```
new;
library odbc;
dlibrary -a odbc;

_GUI = 0;
tutorialfil = sysstate(2,0) $+ "datagro\\dck\\ecce.mdb";
ret = ODBCAddDataSource("Microsoft Access Driver (*.mdb)","tutor",tutorialfile);
ODBCOpen(1,"tutor","","");
```

On commence par charger la bibliothèque ODBC : `library odbc`. Comme les fonctions de cette bibliothèque reposent sur des DLLs, il faut charger la bibliothèque dynamique correspondante : `dlibrary -a odbc`. On utilise ensuite la fonction `ODBCAddDataSource` pour établir une connection avec la base de données *ecce.mdb*, on appelle cette connection `tutor` puis on ouvre un canal de communication vers la base avec la fonction `ODBCOpen`. Ce canal est affecté du numéro 1.

```
selcurr = "";
print "Which currency do you select (Type fin to leave the program) :\n";
selcurr = cons;
```

L'utilisateur rentre au clavier la devise de son choix (fonction `cons`) qui est stockée dans la variable `selcurr`.

```
if ((selcurr $/= "fin") and (selcurr $/= ""));
  selstmt = "SELECT code from devises where devise = '" $+ cons $+ "';";
  print /flush selstmt;
  ODBCExecuteDirect(1,selstmt);
  codecurr = ODBCReadComplete(1);
  print codecurr;
```

On construit la chaîne de caractères `selstmt`. Cette chaîne est écrite en language SQL et correspond à la commande de sélection du code de cette devise. On exécute cette fonction dans le canal 1 à l'aide de `ODBCExecuteDirect`. Le résultat est placé dans le canal 1 qui est alors lu avec `ODBCPReadComplete` et placé dans la variable numérique `codecurr`.

```
  codecurr = ftos(codecurr,"\%*.*lf",3,0);
  selstmt = "SELECT siren from eccelot1 where devise = " $+ codecurr $+ ";";
  print /flush selstmt;
  ODBCExecuteDirect(1,selstmt);
  firmes = ODBCReadComplete(1);
```

La variable numérique `codecurr` est transformée en variable caractère (fonction `ftos`) a l'aide de laquelle on construit une seconde requête qui est à son tour soumise et dont le résultat est stocké dans `firmes`.

```
  ODBCExecuteDirect(1,"create table feuro (firme char(8));");
```

On crée une table `feuro` contenant une variable caractère de longueur 8 nommée `firme`. Cette table est vide pour l'instant.

```
  selstmt = "INSERT INTO feuro (firme) values(?);";
  ODBCPrepareSQL(1,selstmt);
  ODBCBindParameter(1,firmes,1);
```

On construit une commande d'insertion de nouvelles observations dans cette table. On précise la variable pour laquelle des observations vont être rajoutées mais pas les valeurs de ces observations, remplacées par `?`. L'instruction `ODBCPrepareSQL` (et non pas `ODBCExecuteDirect`) envoie cette commande. Elle exécutée par l'instruction suivante, `ODBCBindParameters`, dont le deuxième argument est justement l'ensemble des observations à ajouter. Le premier argument désigne comme d'habitude le canal de communication et le troisième le numéro correspondant au type des données transférées. La correspondance type-numéro est disponible dans le fichier *Help/DCK/Column_ Types.html*.

```
endif;

ODBCClose(1);
ret = ODBCDelDataSource("Microsoft Access Driver (*.mdb)","tutor");
end;
```

On ferme le canal de communication et l'on retire *ecce.mdb* de la liste des tables avec lesquelles une connection est établie.

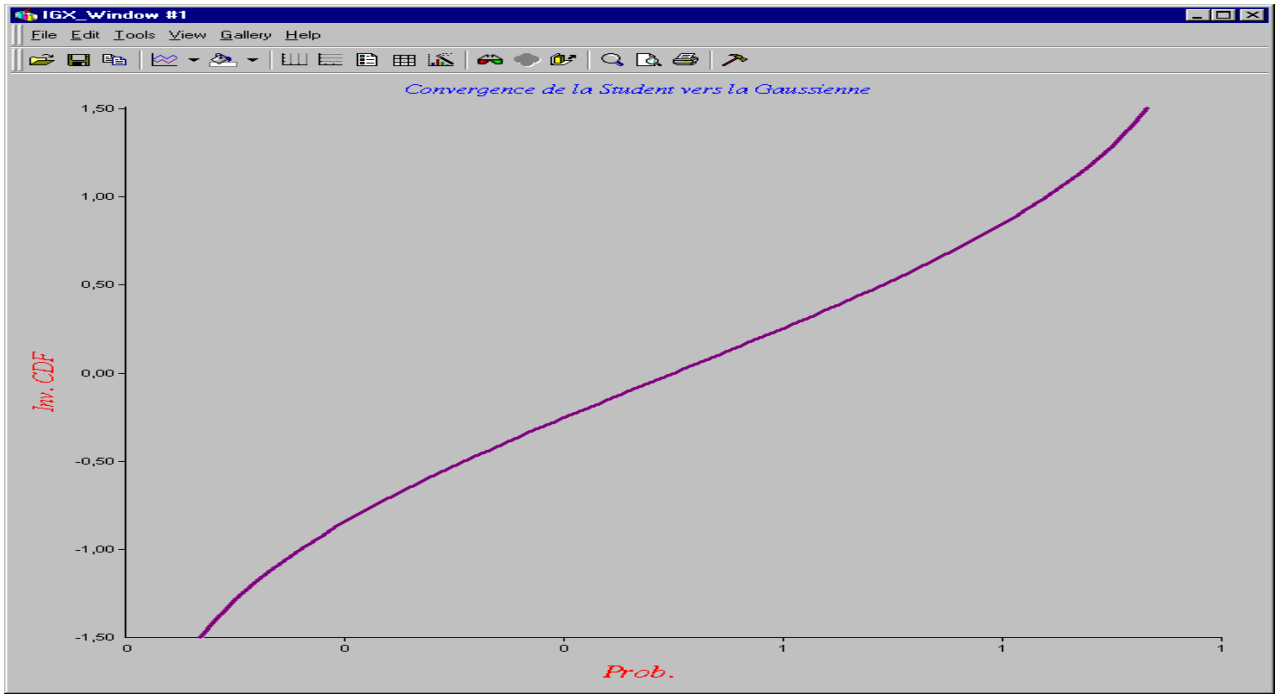
Remarque 1 *Si jamais une commande ou une série de commande SQL est très longue, il est possible de l'écrire dans un fichier de commande commande.cmd, puis, dans le programme GAUSS, d'écrire :*

selstmt = getf("commande.cmd",0) ;
ODBCExecuteDirect(1,selstmt) ;

Bibliothèque IGX

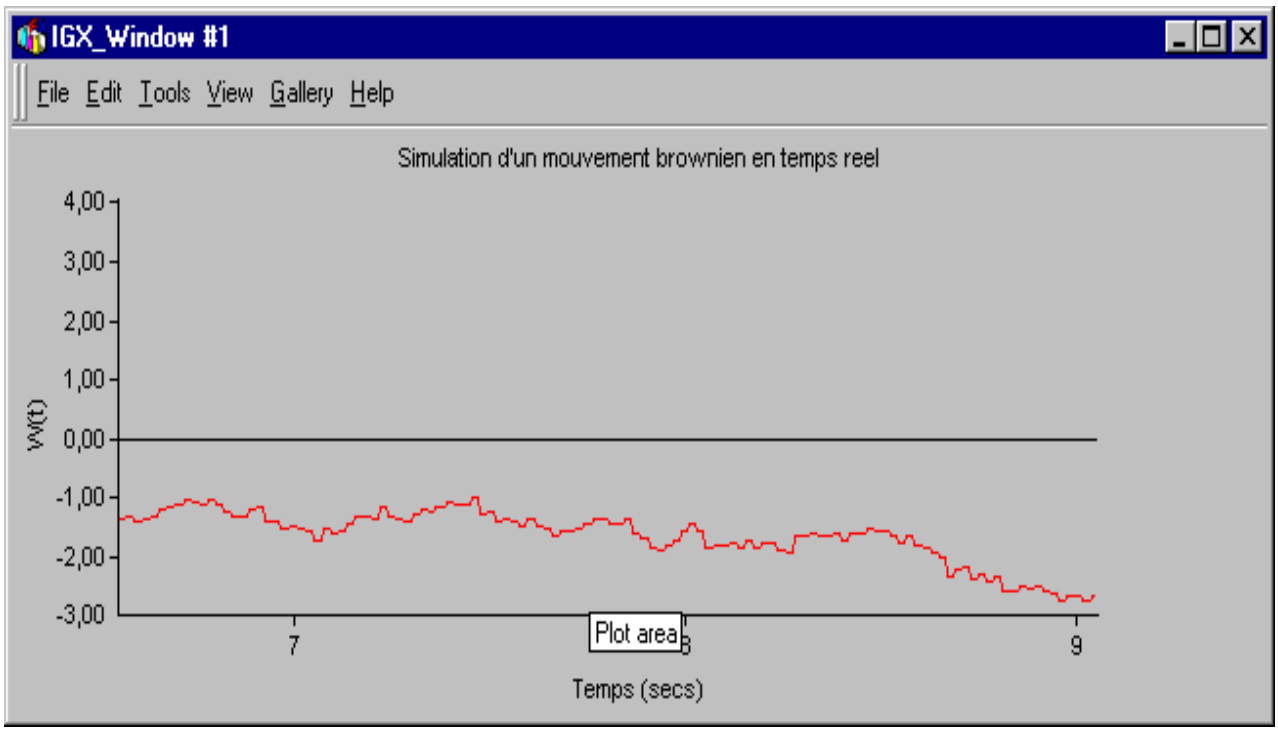
```
new;
library graphix;

df = 1;
p = seqa(0.01,0.01,99);   /* Vecteur des probabilités        */
```

```
x = cdftci(1-p,df);       /* Inverse de la Student          */
n = cdfni(p);             /* Inverse de la Gaussienne       */

gx(p,x~n);
gx_linewidth(series(2),3);
gx_linewidth(series(1),2);

gx_axis(xaxis,0,1);
gx_step(xtick,0.2);
gx_axis(yaxis,-1.5,1.5);
gx_step(ytick,0.5);
gx_xtitle("Prob.","techmath 16",red);
gx_ytitle("Inv. CDF","techmath 14",red);

gx_title("Convergence de la Student vers la Gaussienne","techmath 12", blue);
gx_window(100,50,800,800);
gx_serlegbox(off);
gx_view;


df = 1;
maxit = 50;
timer(&fcalc,0,50,maxit);

proc fcalc(param);
  local x,txt;

  df = df + 1;            /* Incrémente le degré de liberté  */
  x = cdftci(1-p,df);     /* Calcule l'inverse de la student */
  gx_value(series(1),x);  /* Update la première série        */

  retp("");
endp;
```



Bibliothèque IGX

```
new;
library graphix;
rndseed 123;

W = 0;
maxval =  500;
gx_text(xtitle,''Temps (secs)'');

gx_text(title,''Simulation d'un mouvement brownien en temps reel'');
gx_text(ytitle,''W(t)'');

gx_step(xaxis,200);
gx_color(series(1),red);
gx_constant(yaxis,W,'''');
gx_toolbar(off);
gx_annotebar(off);
gx_window(50,50,650,300);
d0 = date;
first = d0;
gx_setrealtime(maxval,1);
timer(&fwrite,0,1,3000);

proc fwrite(x);
  local dur,et;
  et = ethsec(first,date)/100;
  dur = '''' $+ ftocv(ethsec(d0,date)/100,1,0);
  W = W + rndn(1,1)*sqrt(et);
  first = date;
  gx(dur,W);
  retp('''');
endp;
```

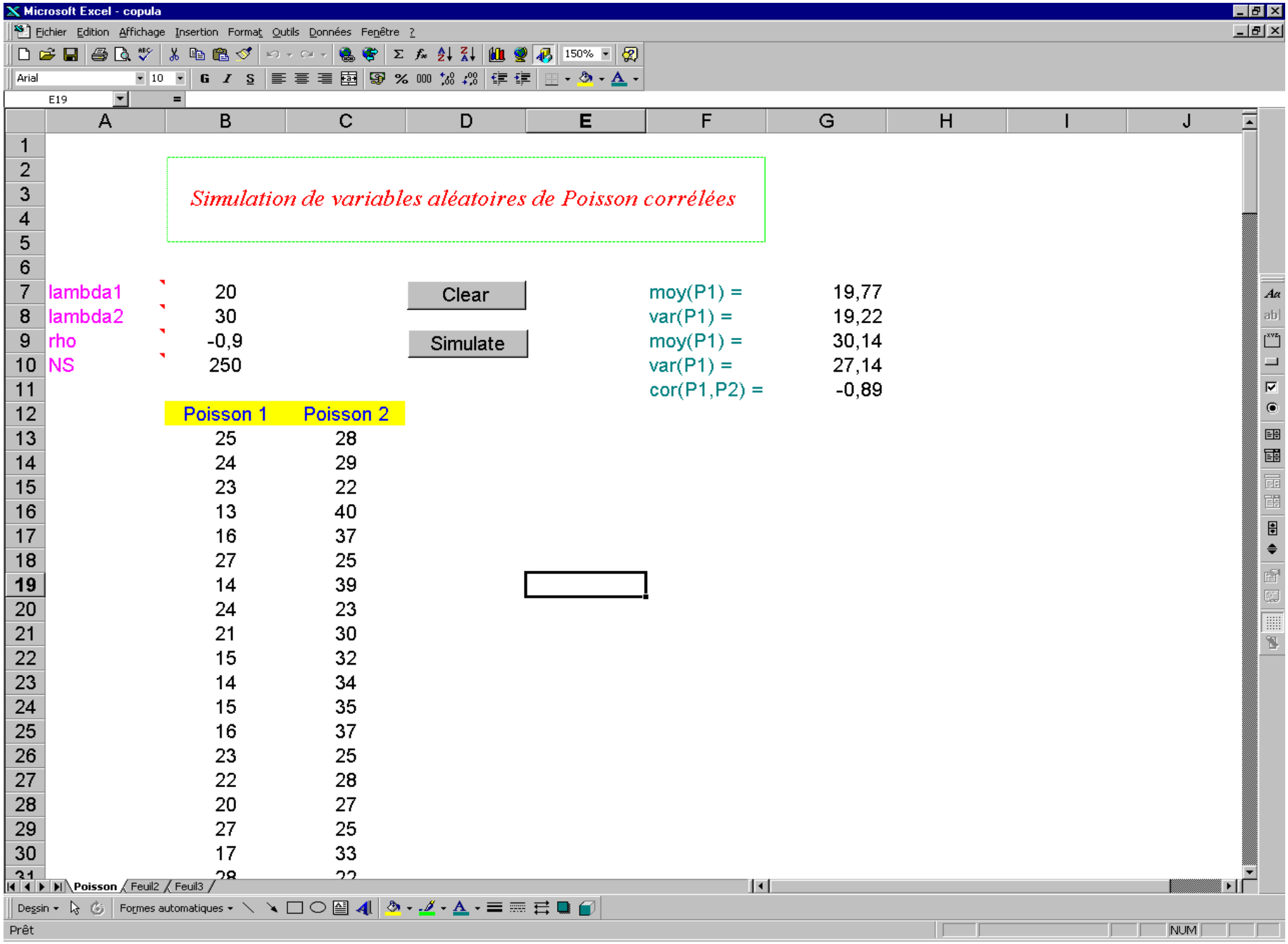


1.3 Le moteur de calcul GAUSS ENGINE

Le fichier rndp2.src

```
/*
** Simulation d'un Poisson bi-dimensionnel
** à partir d'une copule Gaussienne
**
*/

proc rndp2(lambda1,lambda2,rho,N);
  local u;

  u = rndGaussianCopula(xpnd(1|rho|1),N);
  u[.,1] = cdfpi(u[.,1],lambda1);
  u[.,2] = cdfpi(u[.,2],lambda2);

  retp(u);
endp;

proc rndGaussianCopula(rho,N);
  local u;

  u = cdfn(rndmn(zeros(rows(rho),1),rho,N));

  retp(u);
endp;

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

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

  Pchol = chol(SIGMA)';
  y = mu + Pchol*u;

  retp(y');
endp;

proc cdfpi(u,lambda) ;
  local Limit,z,p;
  local x,i;

  Limit = 500;   /* Limit = 500 */

  z = seqa(0,1,Limit);
  p = cdfchic(2.*lambda,2.*(z+1));
  p = (1 + (p - 1).*(p .< 1)).*(p .> 0);

  x =  zeros(rows(u),1);
  i = 1;
  do until i > rows(u);
    x[i] = sumc(p .< u[i]);
    i = i + 1;
  endo;

  retp(x);
endp;
```

Le code VBA


Microsoft Excel - copula
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Arial 10
E19
Simulation de variables aléatoires de Poisson corrélées
lambda1 20
lambda2 30
rho -0,9
NS 250
Clear
Simulate
moy(P1) = 19,77
var(P1) = 19,22
moy(P1) = 30,14
var(P1) = 27,14
cor(P1,P2) = -0,89
Poisson 1 Poisson 2
25 28
24 29
23 22
13 40
16 37
27 25
14 39
24 23
21 30
15 32
14 34
15 35
16 37
23 25
22 28
20 27
27 25
17 33
Poisson Feuil2 Feuil3
Dessin Formes automatiques
Prêt
NUM

```
Public NS As Integer

Sub Simulate()
    Dim lambda1() As Double
    Dim lambda2() As Double
    Dim rho() As Double
    Dim N() As Double
    Dim Poisson As Mat2D
    Dim stats As Mat2D
    Dim cmdFile As String
    Dim cmdString As String
    Dim I, J As Integer

    ' scalaire
    ReDim lambda1(0, 0)
    ReDim lambda2(0, 0)
    ReDim rho(0, 0)
    ReDim N(0, 0)

    ' sélectionne la feuille
    Sheets("Poisson").Select

    ' initialise les paramètres
    lambda1(0, 0) = Range("B7").Value
    lambda2(0, 0) = Range("B8").Value
    rho(0, 0) = Range("B9").Value
    N(0, 0) = Range("B10").Value
    NS = N(0, 0)

    ' initialise le GAUSS ENGINE
    If GAUSS_O_Initialize_VB <> 0 Then End

    ' transfère les données dans le GAUSS ENGINE
    If GAUSS_O_Set2DMatrix_VB("lambda1", 1, 1, 0, lambda1) Then End
    If GAUSS_O_Set2DMatrix_VB("lambda2", 1, 1, 0, lambda2) Then End
    If GAUSS_O_Set2DMatrix_VB("rho", 1, 1, 0, rho) Then End
    If GAUSS_O_Set2DMatrix_VB("N", 1, 1, 0, N) Then End

    ' compile le fichier source rndp2.src
    cmdFile = "d:\gauss\gro\banque\rndp2.src"
    If GAUSS_O_CompileFile_VB(cmdFile) Then End

    ' exécute le code compilé
    If GAUSS_O_Execute_VB Then End

    ' compile une ligne de programme
    cmdString = "u = rndp2(lambda1,lambda2,rho,N); " + _
                "m = meanc(u); s = stdc(u); " + _
                "cor = submat(corrx(u),1,2); " + _
                "stats = m|s^2|cor; "
    If GAUSS_O_CompileString_VB(cmdString) Then End

    ' exécute le code compilé
    If GAUSS_O_Execute_VB Then End

    ' lit la table des symboles du GAUSS ENGINE
    If GAUSS_O_Get2DMatrix_VB(Poisson, "u") Then End
    If GAUSS_O_Copy2DData_VB(Poisson) Then End
    If GAUSS_O_Get2DMatrix_VB(stats, "stats") Then End
```

```
    If GAUSS_O_Copy2DData_VB(stats) Then End

    ' ferme le GAUSS ENGINE
    GAUSS_O_Shutdown_VB

    ' affiche les nombres aléatoires
    For I = 0 To Poisson.Rows - 1
        For J = 0 To Poisson.Cols - 1
            Worksheets("Poisson").Cells(I + 13, J + 2).Value = Poisson.Data(I, J)
        Next J
    Next I

    ' affiche les statistiques empiriques
    Worksheets("Poisson").Cells(7, 7).Value = stats.Data(0, 0)
    Worksheets("Poisson").Cells(8, 7).Value = stats.Data(2, 0)
    Worksheets("Poisson").Cells(9, 7).Value = stats.Data(1, 0)
    Worksheets("Poisson").Cells(10, 7).Value = stats.Data(3, 0)
    Worksheets("Poisson").Cells(11, 7).Value = stats.Data(4, 0)

End Sub


Sub Clear()
    Dim I, J As Integer
    For I = 0 To NS - 1
        For J = 0 To 1
            Worksheets("Poisson").Cells(I + 13, J + 2).Value = ""
        Next J
    Next I
End Sub
```

1.4 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/

1.4.1 Un premier exemple

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 *gro1.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 dllDurbin(double *r,double *x,double *n,double *ndp)
{
 long int i,j,jm;
 double alpha,beta=1;
```

```
 double *y;

 jm=(long int)(*n);
 y=(double*)malloc(jm*sizeof(double));
 alpha=x[0]=-r[0];

 for(j=0;j<jm-1;j++) {
  beta*=1-alpha*alpha;
  if (beta==0) {
   *ndp=1;
   return(0);
  }
  alpha=-r[j+1];
  for(i=0;i<=j;i++)
   alpha-=(y[i]=x[j-i])*r[i];
  alpha/=beta;
  for(i=0;i<=j;i++)
   x[i]+=alpha*y[i];
  x[j+1]=alpha;
 }
 free(y);
 return(0);
}

__declspec(dllexport) int dllLevinson(double *r,double *b,double *x,double *n,double *ndp)
{
 double *y,*ry;
 double alpha,beta,mu;
 long int i,j,jm;

 jm=(long int)(*n);
 y=(double*)malloc(jm*sizeof(double));
 ry=(double*)malloc(jm*sizeof(double));
 jm--;

 alpha=y[0]=-r[0];
 x[0]=b[0];
 beta=1;
 for(j=0;j<jm;j++) {
  beta*=1-alpha*alpha;
  if (beta==0) {
   *ndp=1;
   return(0);
  }
  mu=b[j+1];
  for(i=0;i<=j;i++)
   mu-=x[j-i]*r[i];
  mu/=beta;
  for(i=0;i<=j;i++)
   x[i]+=mu*(ry[i]=y[j-i]);
  x[j+1]=mu;
  if (j==jm-1)
   break;
  alpha=-r[j+1];
  for(i=0;i<=j;i++)
   alpha-=ry[i]*r[i];
  alpha/=beta;
  for(i=0;i<=j;i++)
   y[i]+=alpha*ry[i];
```

```
  y[j+1]=alpha;
 }
 free(y);
 free(ry);
 return(0);
}

__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;
}
```

$\Longrightarrow$ Application : résolution des EDP et modèle de Dupire.

1.4.2 L'utilisation des variables statiques

Celles-ci sont très intéressantes pour construire des générateurs de nombres aléatoires. Dans ce cas, les valeurs des variables de la DLL sont conservées en mémoire.

Fichier *sobol.c*

```
#include <windows.h>


#define IMIN(a,b) ((a)<(b)?a:b)


/* Antonov-Saleev variant of Sobol'sequence */
#define MAXBIT 30
#define MAXDIM 6


/* shared by both algorithms */
#define NTAB 32
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
```

```
BOOL WINAPI __declspec(dllexport) LibMain(HINSTANCE hDLLInst, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH:


            break;
        case DLL_PROCESS_DETACH:


            break;
        case DLL_THREAD_ATTACH:

            break;
        case DLL_THREAD_DETACH:

            break;
    }
    return TRUE;
}


__declspec(dllexport) int dllsobol(double *n,double *x)
{
 int j,k,l;
 unsigned long i,im,ipp;
 static double fac;
 static unsigned long in,ix[MAXDIM+1],*iu[MAXBIT+1];
 static unsigned long mdeg[MAXDIM+1]={0,1,2,3,3,4,4};
 static unsigned long ip[MAXDIM+1]={0,0,1,1,2,1,4};
 static unsigned long iv[MAXDIM*MAXBIT+1]={
  0,1,1,1,1,1,1,3,1,3,3,1,1,5,7,7,3,3,5,15,11,5,15,13,9};

 *n=(int)(*n);
 if (*n < 0) {
  for(k=1;k<=MAXDIM;k++)
   ix[k]=0;
  in=0;
  if (iv[1] != 1)
   return(1);
  fac=1.0/(1L << MAXBIT);
  for(j=1,k=0;j<=MAXBIT;j++,k+=MAXDIM)
   iu[j] = &iv[k];
  for(k=1;k<=MAXDIM;k++) {
   for(j=1;j<=mdeg[k];j++)
    iu[j][k] <<= (MAXBIT-j);
   for(j=mdeg[k]+1;j<=MAXBIT;j++) {
    ipp=ip[k];
    i=iu[j-mdeg[k]][k];
    i ^= (i >> mdeg[k]);
    for(l=mdeg[k]-1;l>=1;l--) {
     if (ipp & 1)
      i ^= iu[j-l][k];
     ipp >>=1;
    }
    iu[j][k]=i;
   }
```

```
  }
 } else {
  im=in++;
  for(j=1;j<=MAXBIT;j++) {
   if (!(im & 1))
    break;
   im >>= 1;
  }
  if (j > MAXBIT) {
/*   fprintf(stderr,''MAXBIT too small in sobseq\n'');*/
   return(1);
  }
  im=(j-1)*MAXDIM;
  for(k=1;k<=IMIN(*n,MAXDIM);k++) {
   ix[k] ^=iv[im+k];
   x[k-1]=ix[k]*fac;
  }
 }
 return(0);
}
```

1.4.3 Un exemple plus abouti : *Multivariate Adapatative Regression Spline*

- Le code fortan est disponible à l'adresse

  http ://www-stat.stanford.edu/~jhf/ftp/progs/

- Transformer le code fortran en C avec *f2c*.
- Construire une *wrapper function* et ajouter un *entry point*.
- Créer la procédure GAUSS.

Fichier *mars.c*

```
/* --- The following code comes from d:\lcc\lib\wizard\dll.tpl. */
#include <windows.h>
#include <malloc.h>
#include ''f2c.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;
}
/* mars36.f -- translated by f2c (version 19980831 for lcc-win32).
   You must link the resulting object file with the library:
 libf77.lib
*/




.....................................
....environ 13.000 lignes de code C.....
.....................................




/**********************************
**                               **
**         DLL POUR GAUSS        **
**                               **
**********************************/



__declspec(dllexport) int dll_mars36(
double *n,
double *p,
double *x,    /* matrice de taille n*p */
double *y,    /* vecteur de taille n   */
double *w,    /* vecteur de taille n   */
      double *nk,
double *mi,
double *lx,   /* vecteur de taille p   */
double *fm,
double *im,
```

```
double *row_fm,
double *row_im,
double *row_sp,
double *row_dp,
double *row_mm,
double *is,
double *il,
double *df,
double *ix,
double *seedGauss,
    double *fv,
double *ic,
double *ms)

{
/* arguments simples */
integer _n,_p,_nk,_mi;
integer _row_fm,_row_im,_row_sp,_row_dp,_row_mm;
integer _is,_il,_ix,_seedGauss,_ic,_ms;
real _df,_fv;

/* vecteurs */
integer *_lx,*_im;
real *_x,*_y,*_w,*_fm,*_sp,*_mm;
doublereal *_dp;

/* compteur de boucle */
int i;


/* cast des arguments simples */
_n  = (integer) *n;
_p  = (integer) *p;
_nk = (integer) *nk;
_mi = (integer) *mi;
_row_fm = (integer) *row_fm;
_row_im = (integer) *row_im;
_row_sp = (integer) *row_sp;
_row_dp = (integer) *row_dp;
_row_mm = (integer) *row_mm;
_is = (integer) *is;
_il = (integer) *il;
_ix = (integer) *ix;
_seedGauss = (integer) *seedGauss;
_ic = (integer) *ic;
_ms = (integer) *ms;
_df = (real) *df;
_fv = (real) *fv;


/* allocation et cast des vecteurs arguments */
_x  = (real*) malloc(sizeof(real)*_n*_p);
_y  = (real*) malloc(sizeof(real)*_n);
_w  = (real*) malloc(sizeof(real)*_n);
_lx = (integer*) malloc(sizeof(integer)*_p);
_fm = (real*) malloc(sizeof(real)*_row_fm);
_sp = (real*) malloc(sizeof(real)*_row_sp);
_dp = (doublereal*) malloc(sizeof(doublereal)*_row_dp);
_mm = (real*) malloc(sizeof(real)*_row_mm);
```

```
_im = (integer*) malloc(sizeof(integer)*_row_im);

for (i=0;i<_n*_p;i++)
_x[i] = (real) x[i];

for (i=0;i<_n;i++)
{
_y[i] = (real) y[i];
_w[i] = (real) w[i];
}

for (i=0;i<_p;i++)
_lx[i] = (integer) lx[i];



/*****************************/
/*                           */
/*       APPEL DE mars_      */
/*                           */
/*****************************/

/*
**
**  int mars_(integer *n, integer *p, real *x, real *y, real *w,
** integer *nk, integer *mi, integer *lx, real *fm, integer *im, real *
** sp, doublereal *dp, integer *mm)
**
*/


print_(&c_0);
speed_(&_is);
logit_(&_il);
setdf_(&_df);
xvalid_(&_ix);
stseed_(&_seedGauss);
setfv_(&_fv);
setic_(&_ic);
setms_(&_ms);
    mars_(&_n,&_p,_x,_y,_w,&_nk,&_mi,_lx,_fm,_im,_sp,_dp,_mm);


/* cast et sauvegarde des résultats */

for (i=0;i<_row_fm;i++)
fm[i] = (double) _fm[i];

for (i=0;i<_row_im;i++)
im[i] = (double) _im[i];


/* libération de l'espace alloué */
free(_x);
free(_y);
free(_w);
free(_lx);
free(_fm);
free(_sp);
```

```
free(_dp);
free(_mm);
free(_im);

return(0);
}
```

Fichier *mars.src*

```
dlibrary d:\gauss\gro\tmp\mars36.dll;

declare matrix _mars_speed = 4.0;
declare matrix _mars_logit = 0;
declare matrix _mars_df = 3.0;
declare matrix _mars_estimate_df = 0;
declare matrix _mars_seed = 0;
declare matrix _mars_penalty = 0;
declare matrix _mars_categorical = 0;
declare matrix _mars_minimum_span = 0;

/*
**> mars_estimate
**
** Objet : Estimation du modèle MARS.
**
** Format : {fm,im} = mars_estimate(x,y,w,nB,nI,Lx);
**
** Entrées :  y - vecteur N*1, variable expliquée
**            x - matrice N*P, variables explicatives
**            w - vecteur N*1, poids des observations
**                -- ou --
**                0 pour un poids égal à 1 pour toutes les observations
**           nB - scalaire, nombre maximal de fonctions de base
**           nI - scalaire, nombre maximal de variables par fonction de base
**           Lx - vecteur P*1, mode de traitement de chaque variable :
**                              0 : la variable n'est pas prise en compte dans le modèle
**                              1 : variable numérique (aucune restriction)
**                              2 : variable numérique qui ne peut intervenir que de façon additive,
**                                  sans interaction avec d'autres variables
**                              3 : variable numérique qui ne peut intervenir que de façon linéaire
**                             -1 : variable qualitative (aucune restriction)
**                             -2 : variable qualitative qui ne peut intervenir que de façon additive,
**                                  sans interaction avec d'autres variables
**                              (si Lx = 0, alors Lx = ones(K,1);)
**
** Sorties : fm - vecteur im[1]*1, modèle MARS
**           im - vecteur im[2]*1, modèle MARS
**
** Globales :
**            _mars_speed - scalaire, facteur d'accélération (1-5)
**                          défaut = 4
**            _mars_logit - scalaire,
**                          0 pour la régression OLS (défaut)
**                          1 pour la régression logistique
**               _mars_df - scalaire, dégré de liberté (défaut = 3.0)
**          _mars_penalty - scalaire, estimation du degré de liberté (défaut = 0)
**      _mars_categorical - mode de traitement des variables qualitatives :
**                             0 : pas d'effet (défaut)
**                             1 : pas d'interaction entre les variables quantitatives et qualitatives
```

```
**                          2 : l'interaction entre les variables quantitatives et qualitatives est
**                              limitée à 2 variables qualitatives
**     _mars_minimum_span - scalaire, nombre minimum d'observations entre chaque noeud (défaut = 0)
**
** Remarques : Voici les correspondances entre les variables GAUSS et les variables du programme FORTRAN
**             du Friedman :
**
**             n,p,x,y,w,lx,fm,im = mêmes variables que celles du programme MARS
**
**             nB = nk
**                  maximum number of basis functions (Ref[2] Sec. 3.6, Ref[3] Sec. 2.3)
**
**             nI = mi
**                  maximum number of variables per basis function (interaction level).
**                        mi = 1 => additive modeling (main effects only);
**                        mi > 1 => up to mi-variable interactions allowed.
**
**             Lx = lx
**                  predictor variable flags; lx(i) corresponds to the ith variable:
**                        lx(i) =  0 : exclude variable from model.
**                                 1 : ordinal variable - no restriction.
**                                 2 : ordinal variable that can only enter additively;
**                                     no interactions with other variables.
**                                 3 : ordinal variable that can enter only linearly.
**                                -1 : categorical variable - no restriction.
**                                -2 : categorical variable that can only enter additively;
**                                     no interactions with other variables.
**
**             fm(3+nk*(5*mi+nmcv+6)+2*p+ntcv),im(21+nk*(3*mi+8)) = mars model.
**                  (nmcv = maximum number of distinct values for any categorical variable;
**                   ntcv = total number of distinct values over all categorical variables.)
**
**             note: upon return im(1) and im(2) contain the lengths of the fm and im
**                    arrays (respectively) actually used by the program.
**
**
**             _mars_speed = is
**                           call speed(is):
**                           is = speed acceleration factor (1-5).
**                           larger values progressively sacrifice optimization thuroughness for
**                           computational speed advantage. this usually results in marked decrease
**                           in computing time with little or no effect on resulting approximation
**                           accuracy (especially useful for exploratory work).
**                           is = 1 => no acceleration.
**                           is = 5 => maximum speed advantage.
**                           (default: is=4) (Ref [4] Secs. 3.0 - 4.0)
**
**             _mars_logit = il
**                           call logit(il):
**                           il = ordinary/logistic regression flag.
**                           il = 0 => ordinary least-squares regression.
**                           il = 1 => logistic regression.
**                           (default: il=0). If logistic regression is selected (il=1) then
**                           the response variable is assumed to take on only the values 0/1 and
**                           the mars model is for the log-odds: f(x) = log (Pr(Y=1:x)/Pr(Y=0:x)).
**                           (Ref[2] Sec. 4.5)
**
**             _mars_df = df
**                           call setdf(df):
```

```
**                          df = number of degrees-of-freedom charged for (unrestricted)
**                          knot optimization. (default: df=3.0)
**                          (Ref[2] Sec. 3.6, Ref[3] Sec. 2.3)
**
**             _mars_estimate_df = ix
**                          call xvalid(ix):
**                          ix = control parameter for sample reuse technique used to automatically
**                          estimate smoothing parameter df (see above) from the data.
**                          ix = 0 => no effect (default). value used for df is set by user if
**                          setdf(df) is called (see above), otherwise default value
**                          (df=3.0) is used.
**                          ix > 0 => ix - fold cross-validation.
**                          ix < 0 => single validation pass using every (-ix)th (randomly selected)
**                          observation as an independent test set.
**                          if ix.ne.0 then call setdf(df) (see above) has no effect. if ix > 0,
**                          computation increases roughly by a factor of ix over that for ix = 0.
**                          for ix < 0 computation increases approximately by a factor of two.
**                          (Ref[3] Sec. 2.3)
**
**             _mars_seed = is
**                          call stseed(is):
**                          is = seed for internal random number generator used to group observation
**                          subsets for validation (ix.ne.0). (default: is=987654321).
**
**
**             _mars_penalty = fv
**                          call setfv(fv):
**                          fv = (fractional) incremental penalty for increasing the number of variables
**                          in the mars model. sometimes useful with highly collinear designs as it
**                          may produce nearly equivalent models with fewer predictor variables,
**                          aiding in interpretation. (fv .ge. 0)
**                          fv = 0.0  => no penalty (default).
**                          fv = 0.05 => moderate penalty.
**                          fv = 0.1  => heavy penality.
**                          the best value depends on the specific situation and some user
**                          experimentation using different values is usually required. this option
**                          should be used with some care. (Ref[2] Sec. 5.3)
**
**             _mars_categorical = ic
**                          call setic(ic):
**                          ic = flag restricting categorical - ordinal interactions.
**                          ic = 0 => no effect (default).
**                          ic = 1 => interactions between categorical and ordinal variables prohibited.
**                          ic = 2 => maximum number of ordinal variables participating in any
**                          interaction is restricted to two. categorical interactions are
**                          unrestricted.
**                          the restrictions associated with a value of ic are imposed in addition
**                          to those that are controlled by the mi and lx flags (see above).
**
**             _mars_minimum_span = ms
**                          call setms(ms):
**                          ms = minimum span (minimum number of observations between each knot).
**                          ms .le. 0 => default value (depending on n and p) is used.
**                          (default: ms=0). (Ref[2] Sec. 3.8)
**
** Références :
**
**  [1] Friedman, J. H. (1988). Fitting functions to noisy data in high
**      dimensions. Proc., Twentyth Symposium on the Interface, Wegman, Gantz,
```

```
**      and Miller, eds. American Statistical Association, Alexandria, VA. 3-43.
**
**  [2] Friedman, J. H. (1991a). Multivariate adaptive regression splines
**      (with discussion).  Annals of Statistics, 19, 1-141 (March).
**
**  [3] Friedman, J. H. (1991b). Estimating functions of mixed ordinal and
**      categorical variables using adaptive splines. Department of Statistics,
**      Stanford University, Tech. Report LCS108.
**
**  [4] Friedman, J. H. (1993). Fast MARS. Department of Statistics,
**      Stanford University, Tech. Report LCS110.
**
**  [5] Friedman, J. H. and Silverman, B. W. (1989). Flexible parsimonious
**      smoothing and additive modeling (with discussion). TECHNOMETRICS, 31,
**      3-39 (Feburary).
**
*/


proc (2) = mars_estimate(y,x,w,nB,nI,Lx);
  local fm,row_fm,im,row_im;
  local row_sp,row_dp,row_mm;
  local N,P,nmcv,ntcv;
  local fmt,omat,basfn,data,sgn,indx;
  local parent,side,vars,knot,coef;
  local sa,i,aux;
  local is,il,df,ix,seed,fv,ic,ms;

  N = rows(x);
  P = cols(x);

  nmcv = 0;
  ntcv = 0;

  row_fm = 3 + nB*(5*nI+nmcv+6) + 2*P + ntcv;
  row_im = 21 + nB*(3*nI+8);

  row_sp = N * (_mars_max(nB+1,2) + 3) +
           _mars_max(_mars_max(3*N+5*nB+P,2*P),4*N) +
           2*P + 4*nB;

  row_dp = _mars_max(N*nB,(nB+1)*(nB+1)) + _mars_max((nB+2)*(nmcv+3),4*nB);
  row_mm = N*P + 2*_mars_max(nmcv,nI);

  row_sp = row_sp + 100;
  row_dp = row_dp + 100;
  row_mm = row_mm + 100;
  row_fm = row_fm + 100;
  row_im = row_im + 100;

  fm = zeros(row_fm,1);
  im = zeros(row_im,1);

  is = _mars_speed;
  il = _mars_logit;
  df = _mars_df;
  ix = _mars_estimate_df;
  seed = _mars_seed;
  fv = _mars_penalty;
```

```
ic = _mars_categorical;
ms = _mars_minimum_span;

x = x';                  /* f2c */

if rows(y) /= N or cols(y) /= 1;
  errorlog "erreur : x et y ne sont pas compatibles.";
  end;
endif;

if ismiss(x) or ismiss(y);
  errorlog "erreur : non implémenté pour les valeurs manquantes.";
  end;
endif;

if _mars_minimum_span >= N;
  errorlog "erreur : _mars_minimum_span est plus grand que le nombre d'observations !";
  end;
endif;

if _mars_minimum_span < 0;
  _mars_minimum_span = 0;
endif;



if nI < 1;
  nI = 1;
endif;

if Lx == 0 or Lx == 1;
  Lx = ones(P,1);
endif;

if rows(Lx) /= P or cols(Lx) /= 1;
  errorlog "erreur : mauvaise dimension du vecteur Lx.";
  end;
endif;

if w == 0 or w == 1;
  w = ones(N,1);
endif;

if rows(w) /= N or cols(w) /= 1;
  errorlog "erreur : mauvaise dimension du vecteur w.";
  end;
endif;

if _mars_logit /= 0;
  _mars_logit = 1;
endif;

if __output;
  call header("MARS - Multivariate Adaptative Regression Splines","",0);
  print;
  print "Input parameters:";
  print "-----------------";
  print "  Number of     Number of       Number of      ";
  print "observations    variables    basis functions   ";
```

```
  let fmt[3,3] = "%lf" 8 0 "%lf" 13 0 "%lf" 17 0;
  call printfm(N~P~nB,1,fmt); print;
  print ;
  print "Mars modeling parameters:";
  print "-------------------------";
  print "Interaction     Degrees    Regression  Incremental  Restricting    Speed   Minimum";
  print "   level      of freedom                 penalty    categorical    factor    span  ";
  let fmt[7,3] = "%lf" 6 0 "%lf" 15 2 "*.*lf" 13 8 "%lf" 13 3 "%lf" 11 0 "%lf" 12 0 "%lf" 10 0;
  omat = nI~_mars_df;
  if _mars_logit == 1;
    omat = omat ~ "LOGIT";
  else;
    omat = omat ~ " OLS ";
  endif;
  omat = omat ~ _mars_penalty ~ _mars_categorical ~ _mars_speed ~ _mars_minimum_span;
  call printfm(omat,1~1~0~1~1~1~1,fmt); print;
  print ;
  print "Predictor variable flag:";
  print "------------------------";
  let fmt[1,3] = "*.*lf" 8 6;
  omat = (0 $+"var:  ")~(0 $+ "#" $+ ftocv(seqa(1,1,P)',1,0));
  call printfm(omat,0,fmt); print;
  omat = (0 $+"flag:  ")~(0 $+ ftocv(Lx',1,0));
  call printfm(omat,0,fmt); print;
  print ;
  print ;
  print "                      ============================";
  print ;
  print ;
  print /flush "Forward stepwise knot placement...";
  print;
  print "Basis functions      Parent      Variable     sided basis      knot value";
  print /flush "-------------------------------------------------------------------------------";
endif;

dllcall /r _dll_mars36(N,P,x,y,w,nB,nI,Lx,fm,im,
                       row_fm,row_im,row_sp,row_dp,row_mm,
                       is,il,df,ix,seed,fv,ic,ms);

fm = fm[1:im[1]];
im = im[1:im[2]];

if __output;
  basfn = 0 $+ "#" $+ ftocv(seqa(0,1,nB+1),1,0);
  data  = fm[1]~miss(zeros(1,4),0)  |
          reshape(fm[2:5*nB+1],nB,5);

  parent = data[.,4];
  side = data[.,2] .> 0;
  vars = abs(data[.,2]);
  knot = data[.,3];
  coef = data[.,1];
  omat = basfn~(error(0)|basfn[1+parent[2:nB+1]]);

  indx = indexcat(side,0);
  sgn = ones(nB+1,1) .* (0 $+ "+");
  sgn[1] = 0 $+ "";
  if indx /= error(0);
    sgn[indx] = ones(rows(indx),1) .* (0 $+ "-");
```

```
  endif;
  omat = omat~ftocv(vars,1,0)~sgn~knot;

  omat[1,3] = "C";
  let fmt[5,3] = "*.*lf" 10 8 "*.*lf" 15 8 "*.*lf" 12 8 "*.*lf" 15 8 "*.*lf" 21 3 ;
  call printfm(omat,0~0~0~0~1,fmt); print; print;

  sa = "" $+ zeros(nB+1,1);
  i = 2;
  do until i > nB + 1;
    if Parent[i] == 0;
      sa[i] = _mars_string_basfn(vars[i],knot[i],side[i]) ;
    else;
      sa[i] = sa[parent[i]+1] $+ " * " $+ _mars_string_basfn(vars[i],knot[i],side[i]) ;
    endif;
    i = i + 1;
  endo;
  sa[1] = "1.000";
  aux = "        " $+ ("" $+ basfn) $+ " :   " $+ sa;
  i = 1;
  do until i > nB + 1;
    print aux[i];
    i = i + 1;
  endo;
  print;
  print ;
  print "                      ============================";
  print ;
  print ;

  print /flush "Final model after backward stepwise elimination..."; print;
  omat = basfn~coef;
  let fmt[2,3] = "*.*lf" 10 8 "*.*lf" 22 5;
  print "Basis functions      Coefficient";
  print /flush "-------------------------------";
  indx = selif(seqa(1,1,nB+1),coef ./= 0);
  omat = omat[indx,.];
  call printfm(omat,0~1,fmt); print;

  print /flush "Response model:"; print;

  aux = ftosa(abs(coef),"%- *.*lf",10,5,1);
  indx = indexcat(coef .> 0,0);
  sgn = ones(nB+1,1) .* (0 $+ " + ");
  if indx /= error(0);
    sgn[indx] = ones(rows(indx),1) .* (0 $+ " - ");
  endif;
  sa = "    " $+ ("" $+ sgn) $+ aux $+ " * " $+ sa ;
  sa[1] = ftosa(coef[1],"%- *.*lf",10,5,1);
  indx = indexcat(coef ./= 0,1);
  aux = sa[indx];
  aux[1] = "Y =  " $+ aux[1];
  i = 1;
  do until i > rows(aux);
    print aux[i];
    i = i + 1;
  endo;

endif;
```

```
  retp(fm,im);
endp;

proc _mars_max(x,y);
  local max;

  max = x .>= y;
  max = x .* max + y .* (1-max);

  retp(max);
endp;

proc _mars_string_basfn(var,knot,side);
  local sa;

  var = ftos(var,"#%*.*lf",1,0);
  if side == 1;
    if knot < 0;
      sa = " + " $+ trimBlank(ftos(-knot,"%- *.*lf",10,3));
    else;
      sa = " - " $+ trimBlank(ftos(knot,"%- *.*lf",10,3));
    endif;
    sa = var $+ sa;
  else;
      sa = trimBlank(ftos(knot,"%- *.*lf",10,3)) $+ " - " $+ var;
  endif;
  sa = "max(0," $+ sa $+ ")";

  retp(sa);
endp;
```

L'exemple de Friedman

```
new;
library gro,pgraph,optmum;
Libgro;

cls;

#include mars36.src;

load data[50,6] = mars.data;
y = data[.,6];
x = data[.,1:5];

_mars_df = 3;
_mars_penalty = 0.0;
_mars_speed = 5;
_mars_minimum_span = 3;
{fm,im} = mars_estimate(y,x,1,15,5,0);
```

Les résultats

```
================================================================================
 MARS - Multivariate Adaptative Regression Splines         4/06/2000   4:55 pm
================================================================================

Input parameters:
-----------------
  Number of     Number of       Number of
```

```
observations    variables    basis functions
      50            5              15

Mars modeling parameters:
-------------------------
Interaction     Degrees    Regression  Incremental  Restricting    Speed   Minimum
   level      of freedom                 penalty    categorical    factor    span
     5           3.00         OLS         0.000          0           5        3

Predictor variable flag:
------------------------
  var:        #1      #2      #3      #4      #5
  flag:        1       1       1       1       1

                    ============================


Forward stepwise knot placement...

Basis functions      Parent      Variable     sided basis      knot value
-------------------------------------------------------------------------
        #0             .           C             -                  .
        #1            #0           4             +              0.003
        #2            #0           2             +              0.444
        #3            #0           2             -              0.444
        #4            #0           1             +              0.594
        #5            #0           1             -              0.594
        #6            #0           3             +              0.608
        #7            #0           3             -              0.608
        #8            #0           5             +              0.008
        #9            #8           1             +              0.703
       #10            #8           1             -              0.703
       #11           #10           2             +              0.630
       #12           #10           2             -              0.630
       #13            #2           1             +              0.625
       #14            #2           1             -              0.625
       #15           #13           5             +              0.008


        #0 :   1.000
        #1 :   max(0,#4 - 0.003)
        #2 :   max(0,#2 - 0.444)
        #3 :   max(0,0.444 - #2)
        #4 :   max(0,#1 - 0.594)
        #5 :   max(0,0.594 - #1)
        #6 :   max(0,#3 - 0.608)
        #7 :   max(0,0.608 - #3)
        #8 :   max(0,#5 - 0.008)
        #9 :   max(0,#5 - 0.008) * max(0,#1 - 0.703)
        #10 :   max(0,#5 - 0.008) * max(0,0.703 - #1)
        #11 :   max(0,#5 - 0.008) * max(0,0.703 - #1) * max(0,#2 - 0.630)
        #12 :   max(0,#5 - 0.008) * max(0,0.703 - #1) * max(0,0.630 - #2)
        #13 :   max(0,#2 - 0.444) * max(0,#1 - 0.625)
        #14 :   max(0,#2 - 0.444) * max(0,0.625 - #1)
        #15 :   max(0,#2 - 0.444) * max(0,#1 - 0.625) * max(0,#5 - 0.008)

                    ============================
```

```
Final model after backward stepwise elimination...

Basis functions      Coefficient
-------------------------------
        #0               9.97377
        #1               9.95269
        #2               7.18855
        #3             -21.15881
        #5             -15.02795
        #6              13.30074
        #7               6.52126
        #9              28.71405
       #12              50.57082
       #13             -58.77685

Response model:

Y =  9.97377
     + 9.95269 * max(0,#4 - 0.003)
     + 7.18855 * max(0,#2 - 0.444)
     - 21.15881 * max(0,0.444 - #2)
     - 15.02795 * max(0,0.594 - #1)
     + 13.30074 * max(0,#3 - 0.608)
     + 6.52126 * max(0,0.608 - #3)
     + 28.71405 * max(0,#5 - 0.008) * max(0,#1 - 0.703)
     + 50.57082 * max(0,#5 - 0.008) * max(0,0.703 - #1) * max(0,0.630 - #2)
     - 58.77685 * max(0,#2 - 0.444) * max(0,#1 - 0.625)
(gauss)
```

1.5 Les ressources sur Internet

- General ressources and information
  - http ://gurukul.ucc.american.edu/econ/gaussres/GAUSSIDX.HTM — American University Software Archive
  - http ://www.aptech.com — Aptech Systems, INC. GAUSS Home Page
  - http ://www.scientificweb.com/sciencee.html — Stefan Steinhaus
  - http ://www.rhkoning.com/gauss/ — Ruud Koning
  - http ://netec.wustl.edu/CodEc/Gauss.html — Gauss at CodEc
- GAUSS distributors
  - http ://www.additive-net.de — ADDITIVE GmbH
  - http ://www.mathstat.com.au — Mathstat Computing
  - http ://www.ravenholm.com — Ravenholm Computing
  - http ://www.ritme.com — Ritme Informatique
  - http ://www.timberlake.co.uk —- Timberlake Consulting
- GAUSS notes
  - http ://www.cirano.umontreal.ca/publication/guides/gauss/gauss.html — Guide sur l'utilisation du logiciel GAUSS by Benoit Durocher and Normand Ranger, CIRANO
  - http ://scottie.stir.ac.uk/software/gaussbook/contents.html — GAUSS an Introduction Online seminar coursebook by David Bell and Felix Ritchie, Stirling University

  - http ://www.ecn.ulaval.ca/~dbol/cours/ecn61694/index.html — Denis Bolduc (Page d'acceuil pour le cours ECN-61694 Économétrie)
  - http ://www.ecn.ulaval.ca/aide/gauss/gauss.html — Denis Bolduc (Le système Gauss)
  - http ://www.few.eur.nl/few/people/draisma/ — Gerrit Draisma (Introduction to Gauss)
  - http ://www.econ.bbk.ac.uk/faculty/kenc/mscfcf.html — Turalay Kenc (Computational Finance)
  - http ://www.columbia.edu/~tl124/Gauss/Content+N.htm — Tsz-Cheong Lai (GAUSS Tutorial)
  - http ://www.uni-koeln.de/themen/Statistik/gauss/gauss-nag.html — An Example of the "Dynamic Library Interface (Nag Call in Gauss, German)
  - http ://www.mimas.ac.uk/stats/gauss/ — GAUSS Home Page on MIMAS
  - http ://statlab.stat.yale.edu/statlab/software/GAUSS.html — GAUSS Home Page at Yale Statlab
- Econometrics
  - http ://www.econotron.com — John Breslaw
  - http ://www.econ.ucla.edu/dwyer/231/syll99.htm — Mark Dwyer (Gauss programs of the course "Introduction to Linear, Multivariate Methods for Serially Dependent Data")
  - http ://elsa.berkeley.edu/gauss_progs.html — Econometrics Laboratory Software Archive
  - http ://swopec.hhs.se/hastef/abs/hastef0171.htm — Jan Eklöf and Sune Karlsson
  - http ://www.econ.umn.edu/~geweke/ — John Geweke (Bayesian econometrics and statistics)
  - http ://econ.la.psu.edu/~eghysels/eric_frm.htm — Eric Ghysels
  - http ://pgiot.core.ucl.ac.be/ — Pierre Giot
  - http ://www.ssc.wisc.edu/~bhansen/progs/progs.htm — Bruce E. Hansen
  - http ://pages.nyu.edu/~bh32/programs_frame.html — Bart Hobijin
  - http ://webware.princeton.edu/econometrics/programs/ — Bo E. Honoré
  - http ://www.biz.uiowa.edu/faculty/horowitz/ — Joel L. Horowitz
  - http ://troi.cc.rochester.edu/~dshn/Progs.html — D.J. Hodgson (Adaptative estimation)
  - http ://www.maxwell.syr.edu/maxpages/faculty/cdkao/working/w.html — Chihwa Kao
  - http ://jhunix.hcf.jhu.edu/~de_lima/gauss.html — Pedro de Lima
  - http ://www.agecon.ucdavis.edu/homepages/Mohapatra/hompg/codelist.html — Sandeep Mohapatra
  - http ://www.eur.nl/few/ei/links/ooms/#programs — Marius Ooms (Parfima program)
  - http ://eclab.econ.pdx.edu/gpe/ — Kuan-Pin Lin and Lani Pennington
  - http ://econweb.sscnet.ucla.edu/potter/232b/gauss232programs.html — Simon M. Potter
  - http ://scottie.stir.ac.uk/~fri01/gauss/gauss.html — Felix Ritchie
  - http ://research.mpls.frb.fed.us/~drunkle/software/GKR/mmp.html — David E. Runkle
  - (GAUSS Programs for Multiperiod Multinomial Probit)
  - http ://faculty.washington.edu/rons/ — Ron Schoenberg
  - http ://www.hhs.se/personal/Psoderlind/ — Paul Söderlind
  - http ://www.iies.su.se/leosven/ — Lars E.O. Svensson
  - http ://www.shh.fi/~vaihekos/mv_econ.htm — Mika Vaihekoski (Generalized Method of Moments)
  - http ://kafuwong.econ.cuhk.edu.hk/research/gausscode.htm — Ka-fu Wong
- Panel Data
  - http ://www.cemfi.es/~arellano/ — Manuel Arellano (DPD98)
  - http ://www.siasr.unisg.ch/lechner/research/p2_nlpd.html — Michael Lechner (Gauss programs of the research project : Econometric Methods for Nonlinear Panel Data Models)
  - http ://www.nek.lu.se/nekcwe/ — Curt Wells

- Garch Models

http ://www.bm.ust.hk/~jcduan/ — Jin-Chuan Duan (GARCH option pricing model)

- – http ://www.duke.edu/~sg12/software/soft.htm — Stephen F. Gray
  - http ://www.cba.uh.edu/~rsusmel/Academic/workp.htm — Rauli Susmel
  - http ://weber.ucsd.edu/Depts/Econ/Software/ARCH.html — UCSD Department of Economics (Gauss programs of K. Kroner and V. Ng)
- Time Series, State Space Models and Kalman Filte
  - http ://weber.ucsd.edu/~jhamilto/software.htm — James Hamilton
  - http ://weber.u.washington.edu/~ezivot/econ512/econ512.htm — Chang-Jin Kim
  - http ://www.econ.washington.edu/user/cnelson/markov/prgmlist.htm — Website for Kim and Nelson (1998) (Computer programs and data to accompany "State-Space Models with Regime-Switching : Classical and Gibbs-Sampling Approaches with Applications")
  - http ://ise.wiwi.hu-berlin.de/~luetke/ — Helmut Lütkepohl (MulTi)
  - http ://www.hec.ca/~p280/code/codepage.html — Simon van Norden
  - http ://cep.lse.ac.uk/fmg/people/shuetrim/gauss.htm — Geoffrey Shuetrim (Multivariate Kalman filtering code)
  - http ://www.socsci.umn.edu/~stimson/dymimic.prg — J. Stimson (Watson-Engle Kalman filter DYMIMIC model)
  - http ://www.few.eur.nl/few/people/djvandijk/ — Dick van Dijk (Gauss programs of the book
  - "Nonlinear Time Series Models in Empirical Finance")
  - http ://www.wws.princeton.edu/~mwatson/publi.html — Mark Watson
  - http ://www.nek.lu.se/nekcwe/ — Curt Wells (programs of the book "The Kalman Filter in Finance")
- Simulation
  - http ://www.polsci.wvu.edu/faculty/mooney/homepage.htm — Christopher Z. Mooney
  - http ://www.arec.umd.edu/mnerlove/mnerlove.htm — Marc Nerlove
- Statistics
  - ftp ://ftp.cict.fr/pub/lstatprob/sir — Y. Aragon (routines for SIR and multivariate SIR - Sliced Inverse Regression)
  - http ://www.econ.ucdavis.edu/~cameron/gauss.html — A. Colin Cameron
  - http ://www.econ.ucdavis.edu/~cameron/racd/racddata.html — A. Colin Cameron and Pravin K. Trivedi (Regression Analysis of Count Data)
  - http ://darwin.cwru.edu/~witte/gauss/ — Sander Greenland (bio-statistics)
  - http ://w1.463.telia.com/~u46304521/riemann/ — Anders Kallen (the Riemann library)
  - http ://GKing.Harvard.Edu/stats.shtml — Gary King
  - http ://www.la.utexas.edu/research/faculty/dpowers/programs.html — Dan Powers
  - http ://www.la.utexas.edu/research/faculty/dpowers/book/ — Dan Powers and Yu Xie
  - (Statistical Methods for Categorical Data Analysis)
  - ftp ://ftp.cict.fr/pub/lstatprob/compar — S. Viguier-Pla (routines of the article "Factor-Based comparison of several populations using the COMPAR software", Computational Statistics, 9, 135-163)
- Finance
  - http ://www.hec.fr/profs/dept/fe/rockinge/home/index.htm — Michael Rockinger
  - http ://www.goodnet.com/~dh74673/gcode.htm — Cameron Rookley
  - http ://www.city.ac.uk/cubs/ferc/thierry/gauss.html — Thierry Roncalli

- Macroeconomics
  - http ://www.uni-siegen.de/~vwlv/ehlgen/software/index.htm — Jürgen Ehlgen
  - http ://weber.ucsd.edu/~wdenhaan/soft.html — Wouter den Haan's software collection (Parameterized Expectations Algorithms)
  - http ://www.eco.utexas.edu/faculty/Kendrick/frontpg/GAUSS.htm — David Andrew Kendrick (macroeconometric model and riccati equations)
  - http ://www.econ.ohio-state.edu/jhm/jhm.html — J. Huston McCulloch
  - http ://ideas.uqam.ca/QMRBC/codes.html — QM&RBC Codes Online
  - http ://bonvent.upf.es/~ravn/morsoft.html — Morten O. Ravn (Business cycle models)
  - http ://www.wws.princeton.edu/~mwatson/publi.html — Mark Watson
- Games Theory
  - http ://aida.econ.yale.edu/~ariel/ — Ariel S. Pakes
  - http ://www.gov.harvard.edu/graduate/tercer/papers/papers.htm — James Honaker
- Others
  - http ://www.informatik.uni-frankfurt.de/~stst/DCK13 — DCK module

2 La bibliothèque GRO du Groupe de Recherche Opérationnelle

Cette bibliothèque a été développé au GRO depuis Avril 1999. Elle est gérée par une équipe de 2 ou 3 ingénieurs. Quelques chiffres :

- **500 procédures**
- **1.76 Mo de fichiers sources GAUSS**
- **990 Ko de fichiers DLLs**
- **1.13 Mo de programmes d'utilisation**

Elle a pour vocation d'assister les ingénieurs dans toutes les études quantitatives. Dès qu'un problème numérique se pose, celui-ci est résolu par l'ingénieur ou par l'équipe de la bibliothèque de manière la plus générale possible. La procédure correspondante est ensuite incorporée dans la bibliothèque afin d'être utilisée ultérieurement. Dans ce qui suit, nous développons deux aspects utilitaires de la bibliothèque :

1. la gestion des bases de données ;
2. l'exploitation rationnelle des graphiques.

2.1 Gestion des bases de données

2.1.1 Format et traitement des bases de données sous GAUSS

Les bases de données GAUSS sont au format `*.dat`. Il s'agit d'un format aux temps d'accès extrêmement rapide, puisque le système à la connaissance de l'emplacement de chaque ligne de la base, contrairement aux formats de certains autres logiciels statistiques. Ainsi, déplacer un pointeur du début à la fin d'une base de données est quasi-instantané, alors que cette opération peut prendre plusieurs minutes sur d'autres systèmes.

GAUSS laisse le choix à l'utilisateur du mode de traitement de la base. Si cette dernière est de taille raisonnable, et que l'on dispose d'assez de RAM (bases de données de quelques dizaines de Mo), il est possible de la charger intégralement en mémoire. Si tel n'est pas le cas, nous sommes alors obligés de travailler *bloc par bloc* :

Il est en effet possible de charger en mémoire un bloc de plusieurs lignes de la base de données grâce à la commande `readr`. Nous travaillons alors séquentiellement. Pour des traitements techniques tels que des procédures de maximum de vraisemblance, de calcul de matrices de corrélations, etc... nous employons des méthodes d'analyse numérique par blocs.

Illustrons le traitement des bases par l'exemple ci-dessous. Il s'agit de calculer la somme des variables d'une base de données. Sur une matrice chargée intégralement en mémoire, la commande `sumc` suffirait. Sur une base de données, la syntaxe est la suivante :

```
open f1 = dataset;
sum = 0;
do until eof(f1);
  y = readr(f1,100);    /* chargement d'un bloc
                        :: de 100 lignes dans y */
  sum = sum + sumc(y);
endo;
f1 = close(f1);
```

Ce traitement peut sembler assez lourd, mais il a l'avantage de laisser une totale liberté au développeur, et de plus, **il peut être considérablement allégé grâce aux procédures de la librairie** GRO.

2.1.2 Les procédures de traitement des bases de données avec la librairie GRO

Notre objectif dans la librairie GRO est de **décharger l'ingénieur des aspects les plus techniques** de la gestion des bases de données, **tout en laissant à sa disposition la puissance du traitement par blocs**, notamment par l'intermédiaire des opérateurs $E \times E$ de GAUSS.

Certaines procédures qui fonctionnent sur bases de données sont directement opérationnelles (calculs simples de sommes, moyennes..., maximum de vraisemblance, régressions statistiques diverses, tris...), et elles présentent généralement toujours la même syntaxe. Voici par exemple le format de la procédure `dat_sumc` qui calcule la somme des colonnes d'une base :

```
/*
**> dat_sumc
**
** Objet : fonction sumc pour bases de données.
**
** Format : s = dat_sumc(dataset,vars)
**
** Entrées :  dataset - chaînes de caractères, nom de la base de données
**               vars - vecteur K*1, noms ou indices des variables
**
** Sorties :        s - vecteur K*1, la somme des K colonnes
**
*/
```

Remarquons qu'il est possible de faire référence aux variables de la base par l'intermédiaire de leur nom ou de leur indice. Dans cette procédure, nous automatisons le choix de la taille optimale des blocs à charger[1], la gestion la référence aux variables, et nous gérons de plus les éventuelles erreurs (mauvaise référence à la base...).

Dans le cas de calculs plus spécifiques, l'ingénieur a entre autres à sa disposition les procédures `dat_applyfn` et `dat_addcols`, qui permettent respectivement de créer de nouvelles bases et de nouvelles variables. Prenons l'exemple de `dat_applyfn` dont le format est le suivant :

```
/*
**> dat_ApplyFn
**
** Objet : Applique une fonction f(x) à une base de données.
**
** Format :  call dat_ApplyFn(dataset,&fct,newdataset,newvars,vtype);
**
** Entrées :   dataset - chaîne de caractères, nom de la base de données
```

[1] L'ingénieur peut toutefois régler ce paramètre à sa guise.

```
**               &fct - pointeur d'une fonction f(x)
**                      de transformation
**         newdataset - chaîne de caractères, nom de la nouvelle base de données
**            newvars - vecteur de caractères, nom des nouvelles variables
**                      0 pour utiliser les variables de la base de départ
**              vtype - vecteur, type des variables
**                      1 pour une variable numérique
**                      0 pour une variable caractère
**                     -1 pour l'utilisation de la convention v89
**
*/
```

Dans cette procédure, la fonction `fct` que l'on passe en argument doit pouvoir prendre en entrée un bloc de la base **d'un nombre de lignes quelconque**, et renvoyer le bloc de la nouvelle base de données correspondant. Le travail de l'ingénieur se trouve alors réduit à l'implémentation de la fonction `fct`, et sa seule contrainte technique consiste à travailler sur des blocs de taille quelconque. Cette procédure est particulièrement efficace si le traitement à effectuer s'exprime à l'aide d'opérateurs $E \times E$. Dans ce cas, la production du bloc de la nouvelle base est quasi-instantané. Toutefois, rien ne l'empêche de travailler ligne par ligne à l'aide d'une boucle, ou encore de séparer son travail sur le bloc en une partie $E \times E$ et une partie *ligne par ligne*.

Cette procédure à été employée en utilisant cette dernière méthode pour calculer des encours théoriques et les comparer à des encours réels d'un produit d'épargne du Crédit Lyonnais. La procédure utilisait des opérateurs $E \times E$ pour effectuer une correspondance entre les dates renseignées de la base et les nouvelles variables, et calculait *ligne par ligne* les encours théoriques. L'année précédente, le même calcul avait pris plusieurs dizaines d'heures sur une station UNIX munie d'un autre logiciel statistique. Un gain très significatif a également été fait en termes de temps de développement et de maintenance du code.

Ainsi, le travail sur les bases de données gagne en souplesse, et le nombre d'étapes de travail est considérablement réduit. En effet, en un voire deux appels des fonctions `dat_applyfn` et `dat_addcols` nous avons à notre disposition de nouvelles variables prêtes à être traitées.

2.1.3 Exemples

2.1.3.1 Les procédures de manipulation des bases de données Dans cet exemple, nous créons une base de données "banque1.dat", nous la visualisons, puis nous créons une nouvelle base de données "banque1_.dat" à partir de "banque1.dat". Le calcul de "banque1_.dat" s'effectue par l'intermédiaire de l'appel de `dat_applyfn` qui appelle elle-même la fonction `fct`. Remarquons que la fonction `fct` traîte les blocs de la base initiale à la fois en $E \times E$ et ligne par ligne.

```
/*
** Exemple d'utilisation
** de dat_savedata, dat_viewdata
** et dat_applyfn
*/

new;
library gro,optmum,pgraph;
Libgro;

target banque;
dataset     = "banque1";
let vnames  = A B C D;

newdataset  = "banque1_";
let newvars = TA GA DA;

/*
** Création de la base de données
```

```
*/


X = rndn(3000,2)~floor(10*rndu(3000,2));
call dat_SaveData(X,dataset,vnames,-1);


/*
** Visualisation de la base
*/

dat_viewdata(dataset,0,8);

/*
** Calcul farfelu
*/

proc (1) = fct(x);
  local y,ta,ga,da,i;

  /* calcul E*E de ta et ga */
  ta = (x[.,1].<=2.3) .* (x[.,3]-x[.,2]);
  ga = ta^3;

  /* calcul ligne par ligne de da */
  da = error(0) * ones(rows(x),1);
  i  = 1;
  do while i <= rows(x);
    if ga[i]>1;
      da[i] = 12;
    endif;
    i = i + 1;
  endo;

  y = ta~ga~da;

  retp(y);
endp;

call dat_ApplyFn(dataset,&fct,newdataset,newvars,-1);

/*
** Visualisation de la nouvelle base
*/

dat_viewdata(newdataset,0,8);
```

2.1.3.2 Maximum de vraisemblance : `regML` Nous cherchons ici à ajuster un modèle linéaire avec des perturbations normales homoscédastiques :

$$\begin{aligned} Y &= X\beta + U \\ U &\rightsquigarrow \mathcal{N}\left(0, \sigma^2\right) \end{aligned}$$

Nous produisons aléatoirement des données dans une base. La procédure `regML` emploie des méthodes d'analyse numérique par blocs afin de calculer un maximum de vraisemblance :

```
/*
** Exemple d'utilisation de regML
*/
```

```
new;
library gro,pgraph,optmum;
Libgro;

target banque;



/*
** création des données
** Nobs * Nblocs observations
*/

rndseed 123456;

Nblocs = 50;
Nobs   = 10;
Nreg   = 5;
sigma  = 0.25;

beta = seqa(1,0.5,Nreg);


dataset   = "regml1";

let vnames = Y X0 X1 X2 X3 X4;

create fh = ^dataset with ^vnames,Nreg+1,8;

  i = 1;
  do while i <= Nblocs;

    x = ones(Nobs,1) ~ rndu(Nobs,Nreg-1);
    y = x*beta + rndn(Nobs,1)*sigma;
    call writer(fh,y~x);

    i = i + 1;
  endo;

call close(fh);


dat_viewdata(dataset,0,8);


/*
** Utilisation de regML
*/

proc ml(theta,data);
  local b,sigma2,u,logl;
  local y,x;

  y = data[.,1];
  x = data[.,2:cols(data)];

  b = theta[1:Nreg];
  sigma2 = theta[Nreg+1]^2;
```

```
  u = y - x*b;

  logl = -0.5*ln(2*pi) - 0.5*ln(sigma2) - 0.5*u^2/sigma2;

  retp(logl);
endp;

sv = zeros(Nreg,1)|1;
_reg_PrintIters = 2;

cls;
{theta,stderr,Mcov,u} = regML(dataset,0,&ml,sv,0,0);
```

La procédure `regML` fournit la sortie suivante :

```
================================================================================
 regML - Maximum Likelihood                              4/06/2000   5:27 pm
================================================================================


Total observations:                                           500
Missing observations:                                           0
Usable observations:                                          500
Valid cases:                                                  500

Number of parameters:                                           6
Number of unrestricted parameters:                              6
Degrees of freedom:                                           494

Value of the maximized log-likelihood function:           3.06945
Mean log-likelihood:                                      0.00614


Parameters      estimates       std.err.      t-statistic         p-value
--------------------------------------------------------------------------
P01              0.991841       0.037756        26.270012        0.000000
P02              1.528265       0.037765        40.467732        0.000000
P03              2.001471       0.038546        51.924449        0.000000
P04              2.485146       0.036548        67.996807        0.000000
P05              3.000942       0.036749        81.661178        0.000000
P06             -0.240490       0.007605       -31.622279        0.000000
```

2.2 Les objets graphiques `gWizard` de la librairie GRO

2.2.1 Motivation

GAUSS permet de produire rapidement des graphes d'une grande qualité, mais n'offre malheureusement pas directement la possibilité de les sauvegarder sous un format à partir duquel ils sont modifiables. Ce problème, d'autant plus important pour le développement de procédures graphiques, est réglé grâce aux objets `gWizard` de la librairie GRO, les `gBuffers`.

Détaillons tout d'abord la structure d'un graphe GAUSS construit grâce à la librairie `pgraph`. Un graphe est composé de différentes fenêtres, variables en taille et en nombre que l'on définit entre autres à l'aide des procédures `window` et `makewind`. Pour chaque fenêtre des variables définissent l'aspect du graphe (position des axes, titre, légende, dessins additionnels, date). Enfin, pour générer le graphe on utilise une certaine commande en fonction du type de graphique désiré qui prend en entrée les données à traiter (`xy` pour une courbe et/ou un nuage de points, `surface` pour une surface, `contour` pour des lignes de niveau). Les objets graphiques `gWizard` permettent de sauvegarder ces trois types de données sous la forme d'un buffer matriciel[2] (éventuellement sans

[2] Un buffer matriciel est un objet GAUSS permettant de suvegarder plusieurs matrices de tailles différentes au sein d'un même objet.

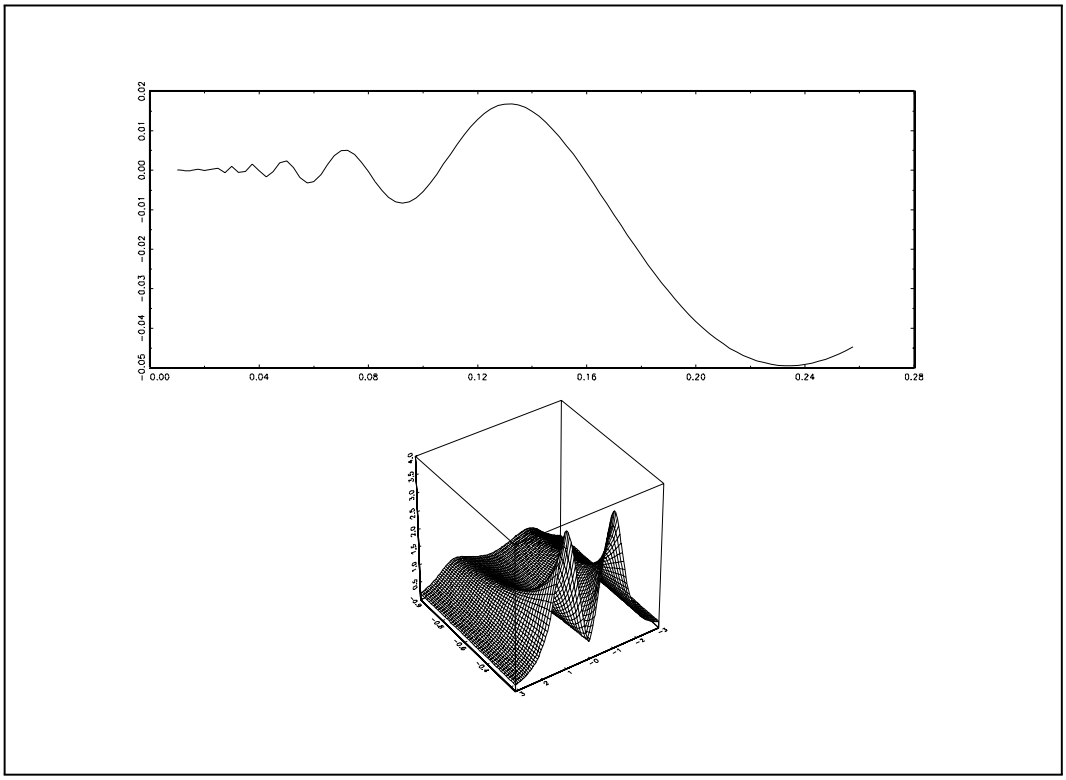


FIG. 16: Le graphe initial

lancer l'affichage du graphe), puis de les récupérer pour modifier éventuellement le graphe (par exemple pour changer les données, les titres, ajouter des commentaires ou encore pour le sauvegarder sous un format d'export).

Prenons l'exemple suivant, et créons tout d'abord un graphe quelconque sur deux fenêtres (le graphe est représenté figure 16) :

```
/*
** Exemple d'utilisation des objets
** graphiques gWizard
*/

new;
library gro,optmum,pgraph;
Libgro;

target banque;


/*
** Affichage de la première version
** du graphique et sauvegarde dans un
** buffer graphique
*/

graphset;

  _pdate = "";
  _psurf = { 0, 0 };
  _pcross = 0;
```

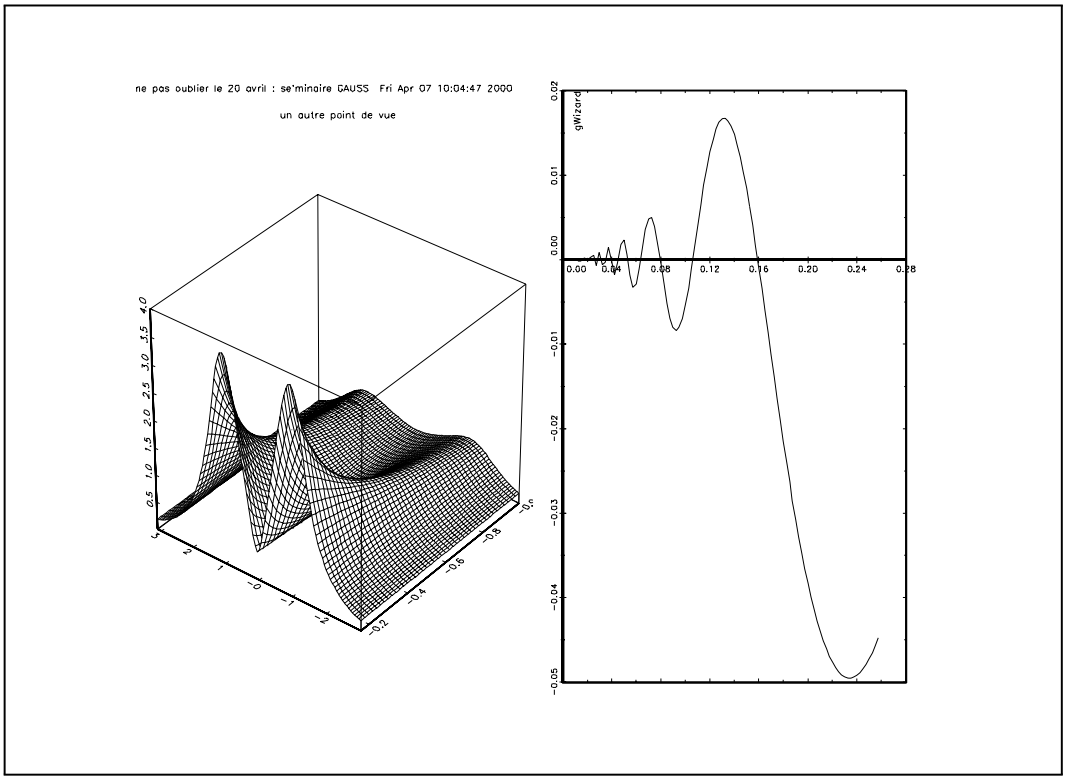


FIG. 17: Le graphe modifié grâce à `gWizard`

```
  begwind;
  window(2,1,0);

  gBuffer = gWizardSavewind(0);

  setwind(1);
    x = seqa(0.01,0.0025,100);
    y = sin(1./x) .* x^2;
    xy(x,y);

    gBuffer = gWizardBuild(gBuffer,gWizardCatch("xy",x,y,0));

  setwind(2);
    _psurf = { 0, 3 }; _pzclr = { -.3 4, .9 12, 1.7 13 };
    x = seqa(-3.05,.06,103)';
    y = seqa(-.95,.02,43);
    z = sin(sqrt((x/2)^2+(y/2)^2)) ./ sqrt((x/2)^8+y^2);
    volume(1,1,1);
    view(7,10,9);
    surface(x,y,z);

    gBuffer = gWizardBuild(gBuffer,gWizardCatch("surface",x,y,z));

endwind;


/*
```

```
** Sauvegarde du buffer
*/
save gwizard1 = gBuffer;
```

Nous chargeons à présent le buffer en mémoire, et nous réaffichons les graphes en leur apportant quelques modifications. Nous sauvegardons également la fenêtre graphique au format `postscript`, afin de l'inclure dans le présent document, figure 17.

```
/*
** chargement du buffer
*/
load gBuffer = gWizard1;


/*
** Affichage d'informations
** sur le buffer
*/
call gWizardInfo(gBuffer);


/*
** Affichage du graphique
** et modifications
*/
graphset;

  begwind;

  /* on ne reprend pas l'ancienne */
  /* configuration des fenêtres   */
  window(1,2,0);

  setwind(1);

    /* chargement de l'aspect précédent de cette fenêtre */
    gWizardSet(gBuffer,2);

    /* modifications */
    _pdate = "ne pas oublier le 20 avril : se'minaire GAUSS  ";
    title("un autre point de vue");
    view(-7,10,9);
    _pzclr = 4|13|12;

    /* affichage du graphe */
    gWizardDraw(gBuffer,2);

  setwind(2);
    gWizardSet(gBuffer,1);
    ylabel("gWizard");
    _pcross = 1;
    gWizardDraw(gBuffer,1);


  /* on sauvegarde le graphe */
  /* au format postcript     */
  graphprt("-c=1 -cf=gwizard2.ps");

endwind;
```

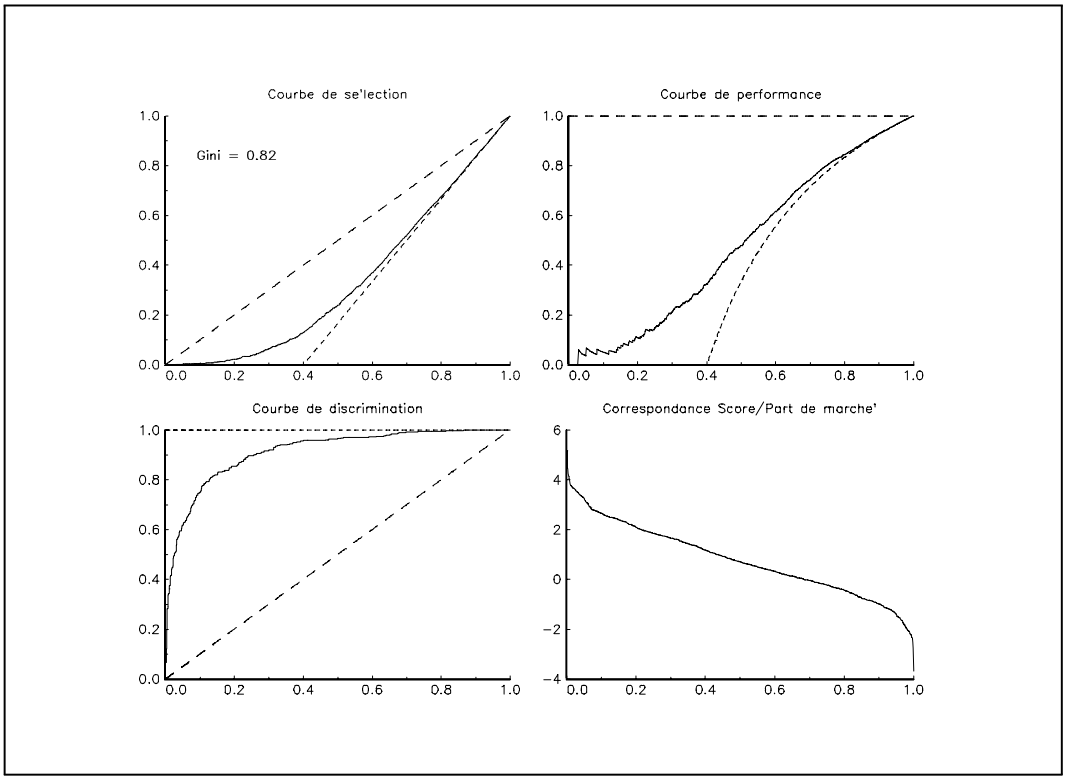


FIG. 18: Performances du premier score

2.2.2 Exemple

La procédure `ScoringCurve` permet de tracer les courbes de performance, sélection et discrimination d'un score afin de mesurer ses performances. Afin de comparer deux scores, il est utile de représenter leurs courbes sur la même fenêtre. Ceci est possible grâce au `gBuffers`. Dans l'exemple ci-dessous, nous produisons deux scores "artificiels" de performances inégales. Nous reproduisons figure 18 la performance du premier score, et figure 19 la comparaison des deux scores sur les courbes de sélection :

Nous employons un échantillon de 1000 observations, avec 400 non-défaillants ($Y = 1$) et 600 défaillants ($Y = 0$). Nos scores sont produits par des lois normales de moyennes différentes pour les défaillants et les non-défaillants :

```
/*
** Exemple d'utilisation
** de ScoringCurve
*/

new;
library gro,optmum,pgraph;
Libgro;

target banque;


N1 = 400; /* nombre de non-défaillants */
N2 = 600; /* nombre de défaillants     */


/* variable expliquée */
```

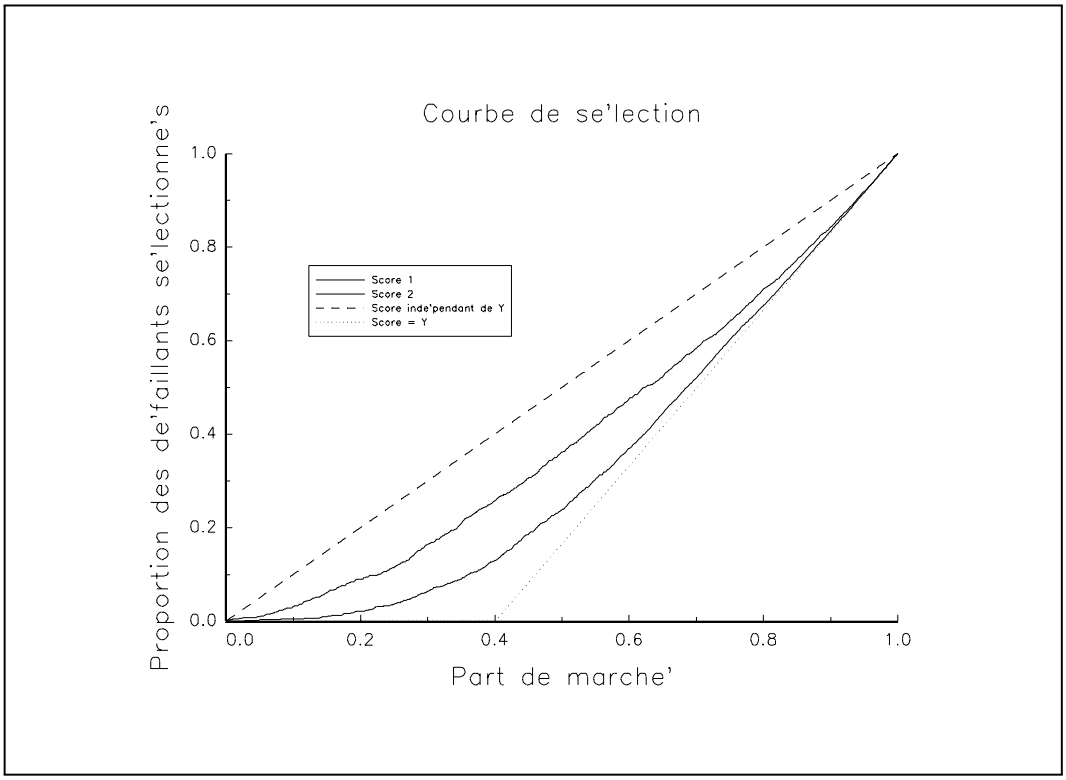


FIG. 19: Comparaison des performances des deux scores

```
Y  = ones(N1,1) | zeros(N2,1);

/* premier score      */
S1 = ( rndn(N1,1) + 2)| rndn(N2,1);

/* deuxième score     */
/* (moins efficace)   */
S2 = ( rndn(N1,1) + 1)| rndn(N2,1);
```

Nous récupérons les graphes des scores dans des `gBuffers` :

```
dataset = 0; /* on ne travaille pas sur une base */
w       = 0; /* pondération uniforme             */
pdraw   = 0; /* ne pas afficher les graphes      */


/* génération des gBuffers */
{gBuffer1,data1,Gini1} = ScoringCurve(dataset,Y,S1,w,pdraw);
{gBuffer2,data2,Gini2} = ScoringCurve(dataset,Y,S2,w,pdraw);
```

Nous traçons les graphes du premier score :

```
{no,gproc,gtype,IsWindow} = gWizardInfo(gBuffer1);
{no,gproc,gtype,IsWindow} = gWizardInfo(gBuffer2);


/* affichage du graphe par défaut */
graphset;

  begwind;
```

```
    gWizardLoadwind(gBuffer1);

  setwind(1);
    gWizardSet(gBuffer1,1);
    gWizardDraw(gBuffer1,1);

  setwind(2);
    gWizardSet(gBuffer1,2);
    gWizardDraw(gBuffer1,2);

  setwind(3);

    gWizardSet(gBuffer1,3);
    gWizardDraw(gBuffer1,3);

  setwind(4);
    gWizardSet(gBuffer1,4);
    gWizardDraw(gBuffer1,4);

  graphprt("-c=1 -cf=score1.ps");

endwind;
```

Enfin, nous récupérons les données des graphes des courbes de sélection des deux scores, et nous les affichons dans la même fenêtre afin de les comparer :

```
/* modifications des graphes précédents */

graphset;

  /* on récupère depuis les
  ** 2 gBuffers les données
  ** des courbes de sélection
  **
  ** (on aurait pu utiliser
  ** la sortie "data" de
  ** ScoringCurve)
  */

  {xsel1,ysel1} = gWizard2D(gBuffer1,1);
  {xsel2,ysel2} = gWizard2D(gBuffer2,1);

  gWizardSet(gBuffer1,1);
  _pmsgstr = "";
  _plegstr = "Score 1\000Score 2\000Score inde'pendant de Y\000Score = Y";
  _plegctl = {2 3 2 4};
  _pltype  = 6|6|3|4;
  _pnumht  = 0;
  xlabel("Part de marche'");
  yLabel("Proportion des de'faillants se'lectionne's");

  graphprt("-c=1 -cf=score2.ps");

  xy(xsel1,ysel1[.,1]~ysel2[.,1]~ysel1[.,2:3]);

endwind;
```

Extraits du séminaire de Londres

3.2 TSM

TSM is a **GAUSS** library for Time Series Modelling. It is primarily designed for the analysis and estimation of ARMA, VARX processes, state space models, fractional processes and structural models. For studying such models, special tools have been developed such as procedures for simulation, spectral analysis, Hankel matrices, etc. Estimation is based on the Maximum Likelihood principle. Linear restrictions may be easily imposed. More than 250 examples illustrate the **TSM** routines. These examples are not just applications, but should be viewed as extensions of the library. They concern the computations of the Exact Maximum Likelihood for vector ARMA models, of the optimal order of VAR models, of Kolmogorov-Smirnov statistic in the frequency domain, of CUSUM and CUSUMsq tests, and many others.

TSM contains the procedures whose list is given bellow. See the command reference part for their full description.

1. ARMA processes
   (a) **arma_ML**: Conditional maximum likelihood for Vector ARMA models
   (b) **arma_CML**: Conditional maximum likelihood for Vector ARMA models under linear restrictions
   (c) **arma_to_VAR1**: VAR(1) representation of a Vector ARMA process
   (d) **arma_roots**: roots of the VAR(1) representation of a Vector ARMA process
   (e) **canonical_arma**: Canonical representation of a Vector ARMA process (infinite AR and MA orders)
   (f) **arma_autocov**: Autocovariances and autocorrelations of a Vector ARMA process
   (g) **arma_impulse**: Responses to Forecast Errors of a Vector ARMA process
   (h) **arma_orthogonal**: Responses to Orthogonal Impulses of a Vector ARMA process
   (i) **arma_fevd**: Forecast Error Variance Decomposition of a Vector ARMA process
   (j) **arma_to_SSM**: State space form of a Vector ARMA model
   (k) **Hankel**: Hankel matrix for multivariate time series
   (l) **SSM_to_arma**: ARMA coefficients of a Vector ARMA state space model
2. VARX processes
   (a) **varx_LS**: Multivariate Least Squares Estimation of VARX processes
   (b) **varx_CLS**: Multivariate Least Squares Estimation of VARX processes under linear restrictions
   (c) **varx_ML**: Maximum Likelihood of VARX processes
   (d) **varx_CML**: Maximum Likelihood of VARX processes under linear restrictions
3. Spectral analysis
   (a) **fourier**: Fourier transform
   (b) **inverse_fourier**: Inverse Fourier transform
   (c) **fourier2**: Fourier transform of two real time series
   (d) **PDGM**: Periodogram of a univariate time series
   (e) **PDGM2**: Periodogram of a multivariate time series
   (f) **CPDGM**: Cross-periodogram
   (g) **CSpectrum**: Coherency, cross-amplitude spectra and phase spectra
   (h) **Smoothing**: Data windowing in the frequency domain
4. Maximum Likelihood Estimation
   (a) Time domain estimation
      i. **TD_ml**: Estimation in the time domain
      ii. **TD_cml**: Estimation in the time domain under linear restrictions

iii. **TDml_derivatives**: Computes the Jacobian, the gradient, the Hessian and the Information matrices in the time domain

(b) Frequency domain estimation for univariate processes

i. **FD_ml**: Estimation in the frequency domain
ii. **FD_cml**: Estimation in the frequency domain under linear restrictions
iii. **FDml_derivatives**: Computes the Jacobian, the gradient, the Hessian and the Information matrices in the frequency domain

5. Univariate Models

(a) **sm_LL**: Local level/random walk plus noise model
(b) **sm_LLT**: Local linear trend model
(c) **BSM**: Basic structural model
(d) **sm_cycle**: Cycle model
(e) **arfima**: Fractional ARMA model with constraints
(f) **canonical_arfima**: Canonical representation of a fractional ARMA process
(g) **sgf_arfima**: Spectral generating function of a fractional ARMA process

6. State space models and the Kalman filter

(a) **SSM**: Print the state space model
(b) **SSM_build**: Build the state space model
(c) **SSM_ic**: Initial conditions for the state space model
(d) **KFiltering**: Kalman filtering
(e) **KF_matrix**: Matrices defined by the Kalman Filter
(f) **KF_gain**: Compute the gain matrices K_t
(g) **KF_ml**: Maximum likelihood of the innovations process
(h) **KSmoothing**: Smoothing
(i) **KForecasting**: Forecasting
(j) **ARE**: Algebraic Riccati equation
(k) **sgf_SSM**: Spectral generating function of a time-invariant state space model
(l) **SSM_autocov**: Autocovariances and autocorrelations of a time-invariant state space model
(m) **SSM_impulse**: Responses to Forecast Errors of a time-invariant state space model
(n) **SSM_orthogonal**: Responses to Orthogonal Impulses of a time-invariant state space model
(o) **SSM_fevd**: Forecast Error Variance Decomposition of a time-invariant state space model
(p) **SSM_Hankel**: Hankel matrix of a time-invariant state space model

7. Resampling and simulation

(a) **Bootstrap**: Boot-strapping a matrix
(b) **bootstrap_SSM**: Bootstrapping state space models
(c) **surrogate**: FT Surrogate data technique
(d) **Kernel**: Density estimation with the Kernel method
(e) **RND_arma**: Simulation of Vector ARMA processes
(f) **RND_arfima**: Simulation of fractional ARMA processes
(g) **RND_SSM**: Simulation of state space models

8. Estimation tools for time series analysis

(a) **FLS**: Flexible least squares
(b) **GFLS**: Generalized flexible least squares of Kalaba and Tesfatsion [1990]
(c) **GFLS2**: Generalized flexible least squares of Lütkepohl and Herwartz [1996]
(d) **GMM**: Generalized method of moments
(e) **RLS**: Recursive least squares

9. Time-Frequency analysis
   (a) Quadrature mirror filters
      i. **Coiflet**: Coiflet filters
      ii. **Daubechies**: Daubechies filters
      iii. **Haar**: Haar filters
      iv. **Pollen**: Pollen filters
   (b) Wavelet analysis
      i. Periodic discrete wavelet transform
         A. **iwt**: Inverse wavelet transform of a vector
         B. **iwt_matrix**: matrix associated with the inverse wavelet transform
         C. **wt**: Wavelet transform of a vector
         D. **wt_matrix**: matrix associated with the wavelet transform
      ii. Wavelet Tools
         A. **extract** : Wavelet decomposition coefficients subband extraction
         B. **insert**: Wavelet decomposition coefficients subband insertion
         C. **Scalogram**: Scalogram of the wavelet decomposition coefficients
         D. **select**: Wavelet decomposition coefficients subband selection
         E. **split**: Wavelet decomposition coefficients subband split
         F. **wPlot**: Wavelet decomposition coefficients plot
   (c) Wavelet packet analysis
      i. Wavelet packet transform
         A. **iwpkt**: Inverse wavelet packet transform
         B. **wpkPlot**: Wavelet packet table plot
         C. **wpkt**: Wavelet packet transform
      ii. Wavelet packet basis
         A. **Basis**: Wavelet packet basis selection
         B. **BasisPlot**: *Time-frequency* plane tilings plot
         C. **BestBasis**: Best basis selection (pruning algorithm)
         D. **BestLevel**: Best level selection
         E. **Entropy**: Shannon entropy cost function
         F. **isBasis**: check whether $\mathcal{B}$ is a basis
         G. **LogEnergy**: Log-energy cost function
         H. **LpNorm**: ℓ^p norm cost function
   (d) Thresholding methods
      i. **SemiSoft**: Semi-soft shrinkage
      ii. **Thresholding**: Quantile thresholding
      iii. **VisuShrink**: Visu shrinkage (or universal thresholding)
      iv. **WaveShrink**: Wavelet shrinkage (hard and soft shrinkages)

10. Filters
   (a) **arma_Filter**: ARMA filtering

(b) **fractional_Filter**: Fractional filtering
(c) **garch_Filter**: GARCH filtering
(d) **Linear_Filter**: Linear filtering
(e) **Savitzky_Golay**: Savitzky-Golay smoothing filter

11. TSM tools

(a) Matrix operators
i. **Commutation_**: Commutation matrix
ii. **Duplication_**: Duplication matrix
iii. **Elimination_**: Elimination matrix
iv. **vech_**: vech operator
v. **xpnd_**: xpnd operator
vi. **xpnd2**: Procedure for coding square matrices

(b) Optimization under linear restrictions
i. **Explicit_to_Implicit**: Convert explicit linear restrictions $C\theta = c$ to implicit linear restrictions $\theta = R\gamma + r$
ii. **Implicit_to_Explicit**: Convert implicit linear restrictions $\theta = R\gamma + r$ to explicit linear restrictions $C\theta = c$
iii. **optmum2**: General nonlinear optimization under linear restrictions

3.2.1 TSM examples

The examples use the following data bases:

- *frfdem.asc* — Ascii file which contains daily quotations of the FRF/DEM exchange rate since 1987.
- *gnp.asc* — table B (page 515) of HARVEY [1990]. The ascii file consists of US Real Gross National Product (GNP) (annual data for the period 1910-1970).
- *lutkepohl.asc* — table E.1 (page 498) of LÜTKEPOHL [1991]. The ascii file consists of three series: German Fixed Investment, Disposable Income and Consumption Expenditures (quarterly, seasonally adjusted for the period 1960-1982).
- *lynx.asc* — series G (page 557) of BROCKWELL and DAVIS or data (appendix 3, page 470) of TONG [1990] or data (page 322) of JANACEK and SWIFT [1993]. The ascii file consists of annual Canadian lynx trappings for the period 1821-1934.
- *purse*.asc — table C (page 516) of HARVEY [1990] or data (page 324) of JANACEK and SWIFT [1993]. The ascii file consists of Purses snatched in the Hyde Park area of Chicago (28-day-period from January 1968).
- *rainfall.asc* — table D (page 517) of HARVEY [1990]. The ascii file consists of Rainfall in Fortaleza, North-East Brazil (annual data for the period 1849-1984).
- *reinsel.asc* — table A.2 (page 227) of REINSEL [1993]. The ascii file consists of two series: U.S. Fixed Investment and Changes in Business Inventories (quarterly, seasonally adjusted for the period 1947-1971).
- *sunspots.asc* — data (page 327) of JANACEK and SWIFT [1993]. The ascii file consists of Wolfer sun spot numbers.

1. **arfima1.prg**
We simulate an ARIMA(1,0,1) process

$$(1 - 0.95L)\, y_t = (1 - 0.5L)\, \varepsilon_t \tag{3.1}$$

with $\varepsilon_t \sim \mathcal{N}(0, 2)$. Then, we estimate the following ARFIMA model in the frequency domain

$$(1 - \phi_1 L)\,(1 - L)^d\, y_t = (1 - \theta_1 L)\, \varepsilon_t \tag{3.2}$$

2. **arfima2.prg**
   We examine the problem of several maxima when a fractional process is estimated in the frequency domain. To do this, we use the simulated process (3.1) and estimate the model (3.2) using two algorithms: the scoring algorithm and the BFGS algorithm of **OPTMUM**.

3. **arfima3.prg**
   In certain cases, the problem of several maxima comes from the estimation of the fractional d coefficient. If we impose the restriction $d = 0$, we notice that we get only one maximum. This suggests first using the Geweke-Porter Hudak (GPH) estimator and then fixing the fractional d coefficient to the GPH estimator to estimate completely the ARFIMA process.

4. **arfima4.prg**
   We simulate the following ARFIMA process with the procedure `RND_arfima`

$$(1 - 0.8L)(1 - L)^{0.25} y_t = (1 - 0.4L)\varepsilon_t \tag{3.3}$$

   with $\varepsilon_t \sim \mathcal{N}(0, 2)$. Then, we estimate the following ARFIMA model in the frequency domain

$$(1 - \phi_1 L)(1 - L)^{d} y_t = (1 - \theta_1 L)\varepsilon_t \tag{3.4}$$

   Firstly, we estimate the unrestricted model. Secondly, we estimate the model under the restriction $d = 0.25$. Thirdly, we impose the restrictions $d = 0.25$ and $\theta_1 = 0.4$. Finally we test the two hypotheses $\mathrm{H}_1 : d = 0.25$ and $\mathrm{H}_2 : (d, \theta_1) = (0.25, 0.4)$ with the likelihood ratio statistic.

5. **arfima5.prg**
   Simulation of fractional processes with $d = -0.25$ and $d = 0.75$.

6. **arma1a.prg**
   Let $y_{1,t}$ be the variation in investment and $y_{2,t}$ the inventories level. We estimate the following vector ARMA(1,1) model

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} - \Phi_1 \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} = \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} - \Theta_1 \begin{bmatrix} \varepsilon_{1,t-1} \\ \varepsilon_{2,t-1} \end{bmatrix} \tag{3.5}$$

   with $\varepsilon_t \sim \mathcal{N}(\mathbf{0}_2, \Sigma)$. We use the Newton-Raphson algorithm to obtain the estimates $\beta = \mathrm{vec}\begin{bmatrix} \Phi_1 & \Theta_1 \end{bmatrix}$. The external variables `_arma_sigma` and `_arma_epsilon` correspond to the estimate of Σ and to the residuals $\hat{\varepsilon}$ respectively.

7. **arma1b.prg**
   We estimate the model (3.5) by exact maximum likelihood. For this, we use the Kalman Filter. To obtain the initial conditions, we use both the estimates of the Conditional Maximum Likelihood and the procedure `SSM_ic`. Given the procedure `KF_ml`, we construct the log-likelihood function. Then, we employ `TD_ml` to obtain the exact ML estimates. Note that the estimates θ correspond to the vector $\mathrm{vec}\begin{bmatrix} \beta & \mathbb{P}^\star \end{bmatrix}$ with $\mathbb{P}^\star = \mathrm{vech}(\mathbb{P})$ and $\mathbb{P}$ the Cholesky decomposition of Σ, that is $\Sigma = \mathbb{P}\mathbb{P}^\top$.

8. **arma1c.prg**
   The model (3.5) is estimated by conditional maximum likelihood with linear restrictions of the form $\beta = R\gamma + r$. We impose $\beta_1 = 1$ (that is $\Phi_{1,11} = 1$). We have

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{1\times 7} \\ \mathbf{I}_7 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \\ \gamma_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

   To construct the matrix R, we employ the `design` procedure. Because the argument `sv` in `arma_CML` is 0, the procedure computes the starting values for the optimization algorithm.

9. **arma1d.prg**
Estimates the model (3.5) by conditional maximum likelihood under the restriction $\Phi_{1,11} = \Phi_{1,21}$ (or $\beta_1 = \beta_2$). We put this linear restriction into the form

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}_{1\times 6} \\ 1 & \mathbf{0}_{1\times 6} \\ \mathbf{0}_{6\times 1} & \mathbf{I}_6 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \\ \gamma_7 \end{bmatrix} + \mathbf{0}_{8\times 1}$$

10. **arma1e.prg**
Estimates the model (3.5) by conditional maximum likelihood with the restrictions $\Phi_{1,12} = \Theta_{1,11} = \Theta_{1,21} = 0$ (or $\beta_3 = \beta_5 = \beta_6 = 0$). These restrictions are motivated because these coefficients are not significantly different from zero. We have

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \end{bmatrix} + \mathbf{0}_{8\times 1}$$

11. **arma1f.prg**
In this example, we impose the restriction that the model (3.5) corresponds to two univariate ARMA(1,1) processes. That is, the matrices Φ_1 and Θ_1 are of the form

$$\begin{bmatrix} \cdot & 0 \\ 0 & \cdot \end{bmatrix}$$

12. **arma1g.prg**
With the command `vread`, we read from the external variables `_ml_derivatives` the Jacobian and gradient vectors and the Hessian and information matrices of the log-likelihood function evaluated at the estimates $\hat{\beta}$ corresponding to the ML estimates under the preceding restrictions (*arma1f.prg*).

13. **arma1h.prg**
We test whether the restrictions in the program *arma1f.prg* are accepted. To this end, we use the likelihood ratio, the Lagrange multiplier or the Wald test. Note that the Lagrange multiplier is evaluated by using the vectors and matrices given by `_ml_derivatives`.

14. **arma1i.prg**
Stability analysis of the model (3.5).

15. **arma1j.prg**
Forecast Error Variance Decomposition of the model (3.5).

16. **arma1k.prg**
Impulse Responses of the model (3.5).

17. **arma2a.prg**
We consider the univariate AR(1) model

$$y_t = 0.5y_{t-1} + \varepsilon_t$$

In its state space form, the vector of state variables is $\begin{bmatrix} y_t & \varepsilon_t \end{bmatrix}^\top$. The covariance matrix corresponds to

$$\begin{bmatrix} E[y_t y_t] & E[y_t \varepsilon_t] \\ E[\varepsilon_t y_t] & E[\varepsilon_t \varepsilon_t] \end{bmatrix}$$

Computing this covariance can be achieved with the **SSM_ic** procedure.

18. **arma2b.prg**
Same program as *arma2a.prg* but with a univariate MA(1) model.

19. **arma2c.prg**
Same program as *arma2a.prg* but with the vector model (3.5).

20. **arma2d.prg**
Exact maximum likelihood estimation of a univariate ARMA(1,1) model by Kalman filter (KOHN and ANSLEY [1983]). The results are compared with those obtained from ANSLEY's [1979] algorithm (**arima** library).

21. **arma2e.prg**
Exact maximum likelihood estimation of the vector ARMA(1,1) model (3.5) by Kalman filter (KOHN and ANSLEY [1983]). The difference with the *arma1b.prg* program is that the initial conditions are computed at each iteration (`SSM_ic` is included in the `ml` procedure). *arma2e.prg* computes the Exact MLE (*arma1b.prg* computes an approximation of the Exact MLE).

22. **autocov1.prg**
Computes the theoretical autocovariances and autocorrelations of the following VAR(1) process

$$Y_t - \begin{bmatrix} .5 & 0 & 0 \\ .1 & .1 & .3 \\ 0 & .2 & .3 \end{bmatrix} Y_{t-1} = \varepsilon_t \tag{3.6}$$

with $\varepsilon_t \sim \mathcal{N}(\mathbf{0}_3, \Sigma)$ and

$$\Sigma = \begin{bmatrix} 2.25 & 0 & 0 \\ 0 & 1 & .5 \\ 0 & .5 & .74 \end{bmatrix}$$

To read the matrices, we use the `varget` procedure.

23. **autocov2.prg**
Computes the theoretical autocovariances and autocorrelations of the following VAR(2) process

$$Y_t - \begin{bmatrix} .5 & .1 \\ .4 & .5 \end{bmatrix} Y_{t-1} - \begin{bmatrix} 0 & 0 \\ .25 & 0 \end{bmatrix} Y_{t-2} = \varepsilon_t \tag{3.7}$$

with $\varepsilon_t \sim \mathcal{N}(\mathbf{0}_2, \Sigma)$ and

$$\Sigma = \begin{bmatrix} .09 & 0 \\ 0 & .04 \end{bmatrix}$$

24. **autocov3.prg**
Computes the theoretical autocovariances and autocorrelations of the vector ARMA model (3.5).

25. **autocov4.prg**
Same program as *autocov2.prg*, but autocovariances are computed with the `SSM_autocov` procedure.

26. **autocov5.prg**
Same program as *autocov3.prg*, but autocovariances are computed with the `SSM_autocov` procedure.

27. **autocov6.prg**
Computes autocovariances and autocorrelations matrices of a time-invariant state space model.

28. **band1a-1d.prg**
Ad hoc examples to show the use of the subband wavelet procedures: `split`, `extract`, `select` and `insert`.

29. **basis1.prg**
We use the procedure `isBasis` to verify that we have a wavelet packet basis.

30. **basis2.prg**
We use the procedure `BasisPlot` to obtain the Time-frequency plane tilings plot of several bases. We can also see the localization in time and in frequency. For the basis *base0*, we have a good localization in time, but not in frequency. For the basis *base9*, this is the opposite.

31. **basis3.prg**
We use the `BestLevel` procedure to select a basis with the log-energy cost function. Then, we verify that the selected basis has effectively the minimal cost value.

32. **basis4.prg**
Same program as *basis3.prg* but with the BestBasis procedure and different cost functions (entropy, ℓ^p norm and log-energy).

33. **boot1-3.prg**
Illustration of the `bootstrap_SMM` procedure.

34. **bsm1.prg**
We study the Basic Structural Model presented by HARVEY [1990]. The measurement equation is

$$y_t = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \eta_t \\ \zeta_t \\ \omega_t \\ \omega_{t-1} \\ \omega_{t-2} \end{bmatrix} + \varepsilon_t$$

with $\varepsilon_t \sim \mathcal{N}(0, H)$ and the transition equation is

$$\begin{bmatrix} \eta_t \\ \zeta_t \\ \omega_t \\ \omega_{t-1} \\ \omega_{t-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \eta_{t-1} \\ \zeta_{t-1} \\ \omega_{t-1} \\ \omega_{t-2} \\ \omega_{t-3} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_3 \\ \mathbf{0}_{2\times 3} \end{bmatrix} \nu_t$$

with $\nu_t \sim \mathcal{N}(0, Q)$. We have

$$H = \sigma_\varepsilon^2$$

and

$$Q = \begin{bmatrix} \sigma_\eta^2 & 0 & 0 \\ 0 & \sigma_\zeta^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{bmatrix}$$

Using the numerical values $\left(\sigma_\varepsilon^2, \sigma_\eta^2, \sigma_\zeta^2, \sigma_\omega^2\right) = (1, 0.25, 1.25, 3)$, we simulate the BSM model with the initial position $\begin{bmatrix} 100 & 4 & 4 & 2 & 3 \end{bmatrix}^\top$. Then we estimate the BSM model with ML in the frequency domain. In our case, we set s equal to 4.

35. **bsm2.prg**
We perform Monte Carlo experiments to investigate the power of the FDML of the BSM model.

36. **bsm3.prg**
In this example, we estimate the basic structural model in the frequency and in the time domains. Because the model is not stable, we cannot use the procedure `SSM_ic`. There are several ways to initialize the Kalman filter. The Kalman filter can be used to obtain unobservable components, for example the seasonal factor.

37. **canon1.prg**
Computes the moving average and autoregressive representations of the VAR(1) process described in equation (3.6).

38. **canon2.prg**
Computes the moving average and autoregressive representations of the VAR(2) process described in equation (3.7).

39. **canon3.prg**
Computes the infinite moving average and autoregressive representations of the vector ARMA process (3.5).

40. **canon4.prg**
For a univariate ARMA(p,q) model, we can use the `canonical_arfima` or `canonical_arma` procedures. The model is

$$y_t - 0.5y_{t-1} - 0.25y_{t-2} = u_t - 0.4u_{t-1} + 0.3u_{t-2} \tag{3.8}$$

41. **canon5.prg**
Computes the impulse responses and the accumulated impulse responses (or the interim multipliers) of the ARMA model (3.8).

42. **canon6.prg**
Computes the impulse responses and the accumulated impulse responses (or the interim multipliers) of the ARFIMA process

$$\left(1 - 0.5L + 0.25L^2\right)(1-L)^d y_t = (1 - 0.3L)\, u_t \tag{3.9}$$

The fractional operator d takes different values: -0.5, -0.25, 0, 0.25, and 0.5.

43. **canon7.prg**
Computes the autocovariances, autocorrelations and partial autocorrelations of the ARFIMA process (3.9). The `AUTOCOV` procedure uses the fact that if the process allows an infinite moving average representation

$$y_t = \sum_{i=0}^{\infty} \theta_i u_{t-i}$$

then the autocovariances γ_i of the process (if we assume that $\text{var}(u_t) = 1$) are equal to

$$\gamma_i = \sum_{j=0}^{\infty} \theta_j \theta_{j+i}$$

The autocorrelations correspond to

$$\rho_i = \frac{\gamma_i}{\gamma_0}$$

and the partial autocorrelations are obtained as the solution of the Toeplitz system

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{i-1} \\ \gamma_{i-1} & \gamma_0 & \gamma_1 & \cdots & \gamma_{i-2} \\ & & & \ddots & \\ \gamma_{i-1} & \gamma_{i-2} & \gamma_{i-3} & \cdots & \gamma_0 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_i \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_i \end{bmatrix}$$

44. **chirp1a.prg**
We define the linear chirp $x_t = \sin\left(100\pi t^2\right)$. Using the Coiflet #2 filters, we compute the difference between the original signal and the reconstructed signal by the inverse wavelet transform.

45. **chirp1b.prg**
We use the precedent linear chirp. We plot the wavelet packet table of the signal. Using the basis $\mathcal{B} = (1,2,3,3)$, we show that the reconstructed signal by applying the inverse wavelet packet transform is the same as the original signal.

46. **cml1-5.prg**
Illustration of the use of the **CML** package with **TSM**.

47. **cpdgm1.prg**
Illustration of the `CPDGM` procedure.

48. **cspect1-2.prg**
Illustration of the `CSpectrum` procedure.

49. **cspect3.prg**
Example 11.7.1 in BROCKWELL and DAVIS [1991].

50. **cusum1.prg**
Estimates the Local Level model (or random walk plus noise) with the data *purse*. The model corresponds to

$$\left\{ \begin{array}{rcl} y_t & = & \mu_t + \varepsilon_t \\ \mu_t & = & \mu_{t-1} + \eta_t \end{array} \right.$$

Using the Kalman filter, we can construct the standardized innovations

$$w_t = \frac{v_t}{\sqrt{f_t}}$$

Then we build the CUSUM statistic

$$W_t = \frac{1}{s} \sum_{i=1}^{t} w_i$$

with s the standard deviation of the standardized innovations w_t and the CUSUMsq statistic

$$W_t^{\bullet} = \frac{\sum_{i=1}^{t} w_i^2}{\sum_{i=1}^{T} w_i^2}$$

51. **cusum2.prg**
Computes the CUSUM and CUSUMsq statistics. The model is a MA(2) process and is estimated by exact maximum likelihood using the Kalman filter.

52. **cusum3.prg**
This is the same thing as the *cusum2.prg* example, except that a leverage point is introduced in the MA(2) process.

53. **cycle.src**
Spectral generating functions for the trend + cycle model and the cyclical trend model.

54. **cycle1.prg**
Represents the power spectra for stochastic cycles.

55. **cycle2.prg**
HARVEY [1989] uses a stochastic cycle plus noise model to explain the *rainfall* data. The spectral generating function for a stochastic cycle plus noise model is the sum of the s.g.f. of the stochastic cycle and the s.g.f. of the noise. The s.g.f. of the stochastic cycle is given by the `_cycle_sgf` procedure. The model is estimated in the frequency domain with the `FD_ml` procedure. We observe that we obtain the same results as in HARVEY [1989].

56. **cycle3.prg**
In this example, we compare the periodogram of the *Rainfall* data with the estimated spectral generating function.

57. **denois1a-1d.prg**
We consider the generated series

$$x_t = \sin(t) + \sin(2t) + u_t$$

with u_t a white noise process. Denoising a series could be done by using the wavelet shrinkage. In a first step, we calculate the wavelet coefficients with the `wt` procedure. In a second step, we use a thresholding technique. Finally, we reconstruct the series by applying the `iwt` procedure to the thresholding coefficients.

58. **denois2a-2b.prg**
In the examples below, the wavelet shrinkage is applied to all the coefficients. But, we can use thresholding techniques just for some coefficients, for example the coefficients of some subband of the wavelet transform or of the wavelet packet transform.

59. **fdml1a.prg**
We simulate an AR(2) process. Then, we use the Bloomfield exponential spectral density. The corresponding spectral generating function is given in Dzhaparidze [1986] on page 125:

$$g(\lambda_j) = \sigma^2 \exp\left(2\sum_{i=1}^{r}\gamma_i \cos(i\lambda_j)\right)$$

We may estimate the vector of parameters $\theta = \begin{bmatrix} \gamma_1 & \cdots & \gamma_r & \sigma \end{bmatrix}^\top$ with the `FD_ml` procedure. In this example, we have set r equal to 4.

60. **fdml1b.prg**
We test now $r = 4$ against $r = 5$. To compute the spectral LM test, we can employ the data buffer `_ml_derivatives` or the procedure `FDml_derivatives`.

61. **fdml2a.prg**
We consider the model z_t, defined by

$$\begin{cases} z_t &= x_t + y_t \\ x_t &= \phi_1 x_{t-1} + u_t \\ y_t &= v_t - \theta_1 v_{t-1} \end{cases}$$

with $u_t \sim N\left(0, \sigma_u^2\right)$ and $v_t \sim N\left(0, \sigma_v^2\right)$. The corresponding spectral generating function is

$$\begin{aligned} g(\lambda_j) &= \sigma_u^2 \frac{1}{\left|1-\phi_1 e^{i\lambda_j}\right|^2} + \sigma_v^2 \left|1-\theta_1 e^{i\lambda_j}\right|^2 \\ &= \sigma_u^2 \frac{1}{\left(1-2\phi_1\cos\lambda_j+\phi_1^2\right)} + \sigma_v^2\left(1-2\theta_1\cos\lambda_j+\theta_1^2\right) \end{aligned}$$

The vector of parameters is set to $\begin{bmatrix} \phi_1 & \sigma_u & \theta_1 & \sigma_v \end{bmatrix}^\top$.

62. **fdml2b.prg**
To see if $\theta_1 = 0.7$ in the above model, we use the LM and LR tests in the frequency domain.

63. **fdml3.prg**
The model is

$$\begin{cases} z_t &= x_t + y_t \\ x_t &= x_{t-1} + u_t - \theta_1 u_{t-1} \\ y_t &= v_t \end{cases}$$

with $u_t \sim N\left(0, \sigma_u^2\right)$ and $v_t \sim N\left(0, \sigma_v^2\right)$. The stationary form is

$$z_t - z_{t-1} = (1-\theta_1 L)\, u_t + (1-L)\, v_t$$

The spectral generating function *for the stationary form* is

$$\begin{aligned} g(\lambda_j) &= \sigma_u^2 \left|1-\theta_1 e^{i\lambda_j}\right|^2 + \sigma_v^2 \left|1-e^{i\lambda_j}\right|^2 \\ &= \left(1-2\theta_1\cos\lambda_j+\theta_1^2\right)\sigma_u^2 + 2\left(1-\cos\lambda_j\right)\sigma_v^2 \end{aligned}$$

Because the stationary form is $z_t - z_{t-1}$, the data used in the `FD_ml` procedure are `z-lag1(z)`.

64. **fdml4.prg**
Same example as *kalman4c.prg*, but the spectral generating function is computed by the `sgf_SSM` procedure.

65. **fdml5-6.prg**
Examples of Maximum Likelihood of multivariate processes in the frequency domain.

66. **fft.prg**
An example to illustrate the problem of the scaled factor. For any time series x_t, we must verify that the Fourier transform for the first frequency $\lambda_0 = 0$ equals the mean of x_t:

$$f(\lambda_0) = \bar{x}$$

67. **filter1a.prg**
Univariate ARMA process estimation with the `arma_Filter` procedure.

68. **filter1b-1c.prg**
Univariate ARMA-GARCH process estimation with the `arma_Filter` and `garch_Filter` procedures.

69. **filter2a.prg**
Estimation of a Fractional ARMA(2,1) process with the `fractional_Filter` procedure.

70. **fls1.prg**
We compare the FLS and OLS methods by applying them to the following model for $t = 1, \ldots, N$

$$y_t = \beta_{1,t} x_{1,t} + \beta_{2,t} x_{2,t} + \beta_{3,t} x_{3,t} + u_t$$

with

$$\beta_{1,t} = 1$$

$$\beta_{2,t} = \sin\left(\frac{2\pi}{N} t\right) + v_{2,t}$$

$$\beta_{3,t} = 0.9\beta_{3,t-1} + v_{3,t}$$

u_t, $v_{2,t}$ and $v_{3,t}$ are Gaussian processes. For the FLS regression, we pose

$$\mu = \begin{bmatrix} 10000 \\ 1 \\ 1 \end{bmatrix}$$

71. **fls2.prg**
We graph the residual efficiency frontier $\left\{ \left(r_D^2(\mu), r_M^2(\mu) \right), \mu \in \mathbb{R}_+ \right\}$ of the preceding model.

72. **fls3.prg**
We consider the model

$$y_t = x_t \beta_t + u_t$$

with

$$\beta_t = \begin{cases} z & \text{if } t \leq S \\ w & \text{if } t > S \end{cases}$$

We estimate β_t with the FLS, RLS and OLS methods. We show that FLS can detect an unanticipated shift from z to w.

73. **fls4.prg**
An example to illustrate the convergence of the FLS estimates to the OLS estimates as μ tends to $+\infty$. We consider different values for μ: 10^4, 10^6, 10^7, 10^8, 4×10^8 and 5×10^9.

74. **fractal1-4.prg — fractal.src**
Different examples to illustrate the estimation of the fractional parameter using wavelets. In *fractal1.prg*, we estimate the d parameter for a white noise process. The method proposed by Wornell and Oppenheim [1992] is based on the complete wavelet coefficients. But, we can use coefficients for just some levels (and not for all the scales). In *fractal2.prg*, we consider a fractional process with $d = 0.25$. The examples *fractal3.prg* and *fractal4.prg* compute the empirical density of the wavelet and GPH estimators.

75. **gfls1.prg**
We compare the GFLS and FLS methods with each other on the following model for $t = 1, \ldots, N$

$$y_t = \beta_{1,t} x_{1,t} + \beta_{2,t} x_{2,t} + \beta_{3,t} x_{3,t} + u_t$$

with $\beta_{1,t}$ a constant, $\beta_{2,t}$ a parameter with seasonal path and $\beta_{3,t}$ a time-varying parameter.

76. **gfls2.prg**
An example to show that the FLS method is a special case of the GFLS method.

77. **gfls3.prg**
We consider the following multi-dimensional process

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} x_{1,t} & 0 & x_{3,t} \\ 0 & x_{2,t} & x_{3,t} \end{bmatrix} \begin{bmatrix} \beta_{1,t} \\ \beta_{2,t} \\ \beta_{3,t} \end{bmatrix} + \begin{bmatrix} u_{1,t} \\ u_{2,t} \end{bmatrix}$$

with $u_{1,t}$ and $u_{2,t}$ two white noise processes and $\beta_{1,t}$, $\beta_{2,t}$ and $\beta_{3,t}$ three time-varying parameters. The corresponding approximately linear system is

$$\begin{cases} \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} & \simeq & \begin{bmatrix} x_{1,t} & 0 & x_{3,t} \\ 0 & x_{2,t} & x_{3,t} \end{bmatrix} \begin{bmatrix} \beta_{1,t} \\ \beta_{2,t} \\ \beta_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \beta_{1,t+1} \\ \beta_{2,t+1} \\ \beta_{3,t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{1,t} \\ \beta_{2,t} \\ \beta_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{cases}$$

We can also estimate $\beta_{1,t}$, $\beta_{2,t}$ and $\beta_{3,t}$ with the GFLS filter. For the first estimation, we set $D_t = I_3$, $M_t = I_2$, $Q_0 = I_3$, $\mathbf{p}_0 = \mathbf{0}_3$ and $\mu = 1$. For the second estimation, D_t is equal to

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{10} & 0 \\ 0 & 0 & \frac{1}{10} \end{bmatrix}$$

and μ is set to 10.

78. **gfls4.prg**
In this example, we show the use of GFLS for the estimation of specific and common components. Suppose a two-dimensional process with

$$\begin{cases} y_t &= s_t^y + c_t + u_t^y \\ x_t &= s_t^x + c_t + u_t^x \end{cases}$$

with c_t the common component of y_t and x_t while s_t^y and s_t^x are the two specific components. Let us consider the approximately linear system

$$\begin{cases} \begin{bmatrix} y_t \\ x_t \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \\ \alpha_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \alpha_{1,t+1} \\ \alpha_{2,t+1} \\ \alpha_{3,t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \\ \alpha_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{cases}$$

Then, $\alpha_{1,t}$ and $\alpha_{2,t}$ can be wiewed as the specific components and we can interpret $\alpha_{3,t}$ as the common factor. Note that the choice of Q_0 and $\mathbf{p}_0$ are not very important in this example, because it does not affect the curve form of the estimates (we obtain the same estimates, but with a slight translation).

79. **gfls5.prg**
The local level model takes the approximately linear form:

$$\begin{cases} y_t & \simeq & \beta_t \\ \beta_{t+1} & \simeq & \beta_t \end{cases}$$

We compare the estimation of the state vector process obtained with the Kalman filter with that given by the GFLS filter (see *ll2.prg* example).

80. **gfls6.prg**
The local linear trend model takes the approximately linear form:

$$\begin{cases} y_t & \simeq & \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \delta_{t+1} \\ \beta_{t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{cases}$$

We compare the estimation of the state vector process obtained with the Kalman filter with the resulting one through the GFLS filter (see *llt2.prg* example).

81. **gmm1a.prg**
We consider the linear model

$$y_t = x_t\beta + u_t \tag{3.10}$$

with $u_t \sim \mathcal{N}(0, \sigma^2)$ and β a 4×1 vector. Let $\theta = \text{vec}\begin{bmatrix} \beta & \sigma \end{bmatrix}$ be the vector of parameters. We estimate θ with GMM by considering the moment conditions

$$\begin{cases} E[u_t] = 0 \\ E\left[u_t^2 - \sigma^2\right] = 0 \\ E[u_t x_{t,i}] = 0 \qquad \forall\, i = 1, \ldots, 4 \end{cases}$$

Note that we use the analytical gradient to perform GMM.

82. **gmm1b.prg**
Constrained GMM of the model (3.10) with $\beta_1 = \beta_2$.

83. **gmm2a.prg**
The model is

$$\begin{cases} y_t & = & \beta_1 + \beta_2 x_t + u_t \\ u_t & \sim & \mathcal{N}(0, h_t^2) \\ h_t^2 & = & \alpha_0^2 + \alpha_1^2 u_{t-1}^2 \end{cases}$$

Let $\theta = \text{vec}\begin{bmatrix} \beta_1 & \beta_2 & \alpha_0 & \alpha_1 \end{bmatrix}$ be the vector of parameters. We estimate θ by the ML and GMM methods. For the GMM estimation, we consider the moment conditions

$$\begin{cases} E_t[u_t] = 0 \\ E_t\left[u_t^2 - h_t^2\right] = 0 \\ E_t[u_t x_t] = 0 \\ E_t\left[\left(u_t^2 - h_t^2\right) u_{t-1}^2\right] = 0 \end{cases}$$

84. **gmm2b.prg**
This is the same program as gmm2a.prg, but we impose that $\alpha_1 = 0$ (no ARCH effect).

85. **gmm3a.prg**
We consider a geometric Brownian motion process

$$\begin{cases} dx_t & = & \mu x_t\, dt + \sigma x_t\, dW_t \\ x(t_0) & = & x_0 \end{cases} \tag{3.11}$$

where W_t is a Wiener process. The solution of the stochastic differential equation (3.11) is

$$x(t) = x_0 \exp\left[\left(\mu - \frac{1}{2}\sigma^2\right)(t - t_0) + \sigma\left(W(t) - W(t_0)\right)\right]$$

Let h be the sampling interval of the discrete-time data. We set

$$\varepsilon_t = \ln\frac{x_t}{x_{t-1}} - \left(\mu - \frac{1}{2}\sigma^2\right)h$$

We can estimate the vector of parameters $\theta = \text{vec}\left[\begin{array}{ccc} \mu & \sigma \end{array}\right]$ by maximum likelihood or by the generalized method of moments. For the ML estimation, we have

$$\ell_t = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln\left(\sigma^2 h\right) - \frac{1}{2}\frac{\varepsilon_t^2}{\sigma^2 h}$$

For the GMM estimation, we consider the two moment conditions

$$\left\{\begin{array}{l} E_{t-1}\left[\varepsilon_t\right] = 0 \\ E_{t-1}\left[\varepsilon_t^2 - \sigma^2 h\right] = 0 \end{array}\right.$$

86. **gmm3b.prg**
We consider an Ornstein-Uhlenbeck process

$$\left\{\begin{array}{rcl} dx_t & = & a\left(b - x_t\right)\, dt + \sigma\, dW_t \\ x\left(t_0\right) & = & x_0 \end{array}\right. \tag{3.12}$$

The solution of the stochastic differential equation (3.12) is

$$x(t) = x_0 e^{-a(t-t_0)} + b\left(1 - e^{-a(t-t_0)}\right) + \sigma\int_{t_0}^{t} e^{a(\theta - t)}\, dW\left(\theta\right)$$

We define

$$\varepsilon_t = x_t - e^{-ah}x_{t-1} - b\left(1 - e^{-ah}\right)$$

Let $\theta = \text{vec}\left[\begin{array}{ccc} a & b & \sigma \end{array}\right]$ be the vector of parameters. The expression of the log-likelihood is

$$\ell_t = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln\left(\sigma^2\left(\frac{1 - e^{-2ah}}{2a}\right)\right) - \frac{1}{2}\frac{\varepsilon_t^2}{\sigma^2\left(\frac{1-e^{-2ah}}{2a}\right)}$$

GMM estimation of θ can be performed by considering the following moment conditions

$$\left\{\begin{array}{l} E_{t-1}\left[\varepsilon_t\right] = 0 \\ E_{t-1}\left[\varepsilon_t^2 - \sigma^2\left(\frac{1-e^{-2ah}}{2a}\right)\right] = 0 \\ E_{t-1}\left[\varepsilon_t x_{t-1}\right] = 0 \end{array}\right.$$

87. **gmm3c.prg**
Chan, Karolyi, Longstaff and Sanders [1992] consider the following stochastic differential equation

$$\left\{\begin{array}{rcl} dy_t & = & \left(\alpha + \beta y_t\right)\, dt + \sigma\left|y_t\right|^{\gamma}\, dW_t \\ y\left(t_0\right) & = & x_0 \end{array}\right. \tag{3.13}$$

To estimate the vector of parameters $\theta = \text{vec}\left[\begin{array}{cccc} \alpha & \beta & \gamma & \sigma \end{array}\right]$, they use the discrete-time model

$$y_{t+1} - y_t = \left(\alpha + \beta y_t\right)h + \varepsilon_{t+1}$$

with $\varepsilon_{t+1} \sim \mathcal{N}(0, \sigma^2\left|y_t\right|^{2\gamma} h)$. They consider the following moment conditions

$$\left\{\begin{array}{l} E_t\left[\varepsilon_{t+1}\right] = 0 \\ E_t\left[\varepsilon_{t+1}^2 - \sigma^2\left|y_t\right|^{2\gamma} h\right] = 0 \\ E_t\left[\varepsilon_{t+1} y_t\right] = 0 \\ E_t\left[\left(\varepsilon_{t+1}^2 - \sigma^2\left|y_t\right|^{2\gamma} h\right) y_t\right] = 0 \end{array}\right.$$

to estimate θ with GMM. In this example, we simulate an Ornstein-Ulhenbeck. Then, we estimate the parameters of the stochastic differential equation (3.13). The Ornstein-Uhlenbeck is a special case of the model (3.13) by imposing $\gamma = 0$. In this case, we have the following correspondence

$$\left\{\begin{array}{rcl} \alpha & = & ab \\ \beta & = & -a \end{array}\right.$$

88. **gmm4a-4i.prg**
Parameters estimation of the Bernoulli, Binomial, Negative Binomial, Poisson, Gamma, Beta, Laplace-Gauss, Log-normal and Exponential distributions.

89. **gmm5a-5b.prg**
Parameters estimation of univariate ARMA processes.

90. **gmm6a-6c.prg**
Parameters estimation of state space models.

91. **golay1.prg**
The program computes the coefficients of the Savitzky-Golay filter for different values of M, n_L and n_R. It replicates the table given on page 646 in PRESS, TEUKOLSKY, VETTERLING and FLANNERY [1992].

92. **golay2.prg**
An illustration of the `Savitzky_Golay` procedure applied to noisy data.

93. **gph1.prg**
GEWEKE and PORTER-HUDAK [1983] suggested the following regression to estimate the fractional integration order d of a time series

$$\ln I(\lambda_j) = c - d \sin^2 \frac{\lambda_j}{2} + u_t \tag{3.14}$$

We employ this method to estimate the fractional root of a white noise process. We test $d = 0$.

94. **gph2.prg**
REISEN [1994] proposes employing the smoothed periodogram in regression (3.14). The series used is a random walk process. We compare the estimates of d based on the periodogram and those based on the smoothed periodogram with the Parzen lag window generator. Then, we test the null hypothesis $d = 1$.

95. **gph3.prg**
We estimate the fractional root of a white noise process by using different smoothed periodograms and then we test if the hypothesis $d = 0$ cannot be rejected.

96. **gph4.prg**
This example is a Monte Carlo investigation of the power of the GPH estimator and the estimator based on a smoothed periodogram (Bartlett and Tukey with the parameter equal to 0.20). To obtain the density of the different estimators, we use the kernel estimator.

97. **gph5.prg**
In the frequency domain, we estimate an ARFIMA process in two ways. The first one consists of estimating all the parameters by maximizing the log-likelihood function. In the second method, we use the GPH estimator to estimate the fractional part of the ARFIMA model and we estimate the ARMA part of the ARFIMA model.

98. **gph6.prg**
Monte Carlo experiments of the standard errors of the GPH estimator and those based on the smoothing periodogram.

99. **hankel1.prg**
Hankel matrix of a univariate time series.

100. **hankel2.prg**
Hankel matrix of a multivariate time series.

101. **hankel3.prg**
Monte Carlo experiments of the singular value decomposition of the Hankel matrix for white noise and AR(1) processes.

102. **hankel4.prg**
Monte Carlo experiments of the singular value decomposition of the Hankel matrix for an AR(2) process.

103. **hankel5.prg**
McMillan order of a VAR process.

104. **hankel6.prg**
Computes the theoretical and the empirical Hankel matrices of the ARMA(1,1) model (3.5).

105. **hankel7.prg**
In this example, we check for the McMillan order of various state space models to be equal to the number of state variables.

106. **hurst1.prg — hurst.src**
R/S statistic and Hurst exponent with a white noise process.

107. **hurst2.prg — hurst.src**
R/S statistic and Hurst exponent with a fractional process.

108. **hurst3.prg — hurst2.src**
Estimates the Hurst exponent with the method described in TAQQU, TERERovsky and WILLINGER [1995].

109. **icss1-2.prg — icss.src**
Detection of changes of variance by the ICSS algorithm.

110. **impuls1a-2b.prg — impuls.txt**
Computes the standard errors of the impulse responses by simulation techniques.

111. **jump.prg**
An example of jump and sharp cust detection by wavelets.

112. **kalman1a.prg**
We consider the following state space model

$$\begin{cases} \begin{bmatrix} y_{1,t} \\ y_2, t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} 10 \\ 0 \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 \\ 0 & 0.2 \end{bmatrix} \begin{bmatrix} \alpha_{t-1} \\ \beta_{t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \eta_t \end{cases} \tag{3.15}$$

with

$$H = E\left[\varepsilon_t \varepsilon_t^\top\right] = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.16}$$

and $Q = E[\eta_t \eta_t] = 1$. We build the state space model in a time-invariant form.

113. **Kalman1b.prg**
We build the state space model (3.15) in a time-variant form.

114. **kalman1c.prg**
We simulate the state space model (3.15).

115. **kalman1d.prg**
Kalman filtering of the state space model (3.15) in its time-invariant form. $\mathbf{a}_0$ and P_0 are computed using the `SSM_ic` procedure.

116. **kalman1e.prg**
Kalman filtering of the state space model (3.15) in its time-variant form.

117. **kalman1f.prg**
Graphical representation of the estimated value of α_t with its 95% confidence interval.

118. **kalman1g.prg**
Graphical representation of the log-likelihood vector.

119. **kalman1h.prg**
Exact Maximum likelihood estimation of the model

$$\begin{cases} \begin{bmatrix} y_{1,t} \\ y_2,t \end{bmatrix} = \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} \theta_3 \\ 0 \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} = \begin{bmatrix} \theta_6 & \theta_7 \\ 0 & \theta_8 \end{bmatrix} \begin{bmatrix} \alpha_{t-1} \\ \beta_{t-1} \end{bmatrix} + \begin{bmatrix} \theta_9 \\ \theta_{10} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \eta_t \end{cases} \tag{3.17}$$

with

$$H = \begin{bmatrix} \theta_4 & 0 \\ 0 & \theta_5 \end{bmatrix} \tag{3.18}$$

and $Q = \theta_{11}$.

120. **kalman1i.prg**
Conditional MLE with $\mathbf{a}_0 = \mathbf{0}$ and $P_0 = 0_{2\times 2}$ in the time-variant form. Note the use of external variables.

121. **kalman1j.prg**
Smoothing of the estimated model.

122. **kalman1k.prg**
Forecasting of the estimated model.

123. **kalman2a.prg**
Maximum likelihood estimation of the state space model

$$\begin{cases} y_t = \begin{bmatrix} 1 & x_t & t \end{bmatrix} \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_{0,t} \\ \eta_{1,t} \\ \eta_{2,t} \end{bmatrix} \end{cases} \tag{3.19}$$

The unknown parameters θ are $H = \theta_1^2$ and

$$Q = \begin{bmatrix} \theta_2^2 & 0 & 0 \\ 0 & \theta_3^2 & 0 \\ 0 & 0 & \theta_4^2 \end{bmatrix} \tag{3.20}$$

$\mathbf{a}_0$ and P_0 are set to the null vector and matrix respectively.

124. **kalman2b.prg**
Same program as *kalman2a.prg*, but $\mathbf{a}_0$ and P_0 are fixed differently. This program shows the initialization problem of the Kalman filter.

125. **kalman3a.prg**
We simulate a linear process with ARMA parameters.

126. **kalman3b.prg**
Conditional maximum likelihood of the corresponding state space model

$$\begin{cases} y_t = \begin{bmatrix} x_t & 0 \end{bmatrix} \begin{bmatrix} \beta_t \\ \eta_t \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \beta_t \\ \eta_t \end{bmatrix} = \begin{bmatrix} \phi_1 & -\theta_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_{t-1} \\ \eta_{t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \eta_t \end{cases} \tag{3.21}$$

The estimated parameters are ϕ_1, θ_1, σ_ε and σ_η.

127. **kalman3c.prg**
Conditional maximum likelihood of another representation of the above state space model

$$\begin{cases} y_t = \begin{bmatrix} x_t & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_t \\ \eta_t \\ \varepsilon_t \end{bmatrix} \\ \begin{bmatrix} \beta_t \\ \eta_t \\ \varepsilon_t \end{bmatrix} = \begin{bmatrix} \phi_1 & -\theta_1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_{t-1} \\ \eta_{t-1} \\ \varepsilon_{t-1} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_t \\ \varepsilon_t \end{bmatrix} \end{cases}$$

This program is an illustration of the identification problem.

128. **kalman4a.prg**
Suppose that we observe a process y_t with a measurement error ε_t. We note z_t the observed process. We have

$$z_t = y_t + \varepsilon_t \tag{3.22}$$

We suppose that y_t is an ARMA(1,1) process

$$y_t = \phi_1 y_{t-1} + u_t - \theta_1 u_{t-1} \tag{3.23}$$

The state space form of this model is

$$\begin{cases} z_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ u_t \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} y_t \\ u_t \end{bmatrix} = \begin{bmatrix} \phi_1 & -\theta_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ u_{t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_t \end{cases} \tag{3.24}$$

We simulate the ARMA plus noise process and then we use the Kalman filter to obtain the estimate of the unobserved component y_t.

129. **kalman4b.prg**
In this example, we estimate the coefficients ϕ_1, θ_1, σ_u and σ_ε of the model (3.24) by maximum likelihood in the time domain.

130. **kalman4c.prg**
We estimate the ARMA plus noise model by maximum likelihood in the frequency domain. The corresponding spectral generating function is

$$g(\lambda_j) = \sigma_u^2 \frac{1 - 2\theta_1 \cos\lambda_j + \theta_1^2}{1 - 2\phi_1 \cos\lambda_j + \phi_1^2} + \sigma_\varepsilon^2 \tag{3.25}$$

131. **kalman4d.prg**
We estimate the model (3.24) under the restriction $\theta_1 = 0$. This restriction can be written as:

$$\begin{bmatrix} \phi_1 \\ \theta_1 \\ \sigma_u \\ \sigma_\varepsilon \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \sigma_u \\ \sigma_\varepsilon \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.26}$$

The restricted ML estimates are obtained both in the frequency domain (with the `FD_cml` procedure) and in the time domain (with the `TD_cml` procedure).

132. **kalman4e.prg**
Illustrate the `KForecasting` procedure to obtain forecasts of a process.

133. **kalman4f.prg**
In the model (3.24), we compute the smoothed component $\mathbf{a}_{t|T}$. If the variance of ε_t is zero, then we must verify that the first component of $\mathbf{a}_{t|T}$ is just equal to z_t or y_t.

134. **kalman4g.prg**
Smoothing the model (3.24) with the Kalman filter.

135. **kalman5a.prg**
Modelling the *lutkepohl* data as a VAR(2) process

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_2 \end{bmatrix} + \Phi_1 \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \\ y_{3,t-1} \end{bmatrix} + \Phi_2 \begin{bmatrix} y_{1,t-2} \\ y_{2,t-2} \\ y_{3,t-2} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \\ \varepsilon_{3,t} \end{bmatrix} \tag{3.27}$$

with $\varepsilon_t \sim \mathcal{N}(\mathbf{0}, \Sigma)$. We estimate this model in the time domain with the Kalman filter. We use the Cholesky decomposition to define the matrix Q, that is $Q = \mathbb{P}\mathbb{P}^\top$. The estimated vector θ corresponds to

$$\begin{bmatrix} \text{vec}(\Phi_1) \\ \text{vec}(\Phi_2) \\ \mu \\ \text{vech}(\mathbb{P}) \end{bmatrix}$$

with vech the operator in Lütkepohl sense.

136. **kalman5b.prg**
Does not income/consumption $(y_{2,t} - y_{3,t})$ cause investment $(y_{1,t})$? We can test this hypothesis by using the Wald statistic. Note that this hypothesis corresponds to the fact that the matrices Φ_1 and Φ_2 are of the form

$$\begin{bmatrix} \cdot & 0 & 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

This is equivalent to test $\theta_4 = \theta_7 = \theta_{13} = \theta_{16} = 0$.

137. **kalman5c.prg**
Using the results of the t-statistics in the *kalman5a.prg* example, we impose that the following coefficients are zero:

$$\{\theta_2, \theta_3, \theta_4, \theta_5, \theta_7, \theta_{10}, \theta_{11}, \theta_{12}, \theta_{13}, \theta_{14}, \theta_{16}, \theta_{17}, \theta_{18}, \theta_{19}, \theta_{23}\}$$

The restricted model is estimated by maximum likelihood in the time domain.

138. **kalman5d.prg**
We check the accuracy of the above restrictions. To this end, we use the Likelihood Ratio (LR) and the Lagrange Multiplier (LM) statistics. The LM test is computed using the different matrices of the `_ml_derivatives` external variable.

139. **kalman5e.prg**
Another way to compute the LM tests with the `TDml_derivatives` procedure.

140. **kalman5f.prg**
Computes the LM test with an OPG artificial regression.

141. **kalman6a.prg**
We study a time-variant model

$$y_t = \beta_{0,t} x_{0,t} + \beta_{1,t} x_{1,t} + u_t \tag{3.28}$$

with $u_t \sim \mathcal{N}\left(0, \sigma_u^2\right)$ and

$$\begin{cases} \beta_{0,t} &= \beta_{0,t-1} + v_0 \\ \beta_{1,t} &= \beta_{1,t-1} + v_1 \end{cases} \tag{3.29}$$

with $\begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \Sigma_v)$. We suppose that

$$\Sigma_v = \begin{bmatrix} \sigma_0^2 & 0 \\ 0 & \sigma_1^2 \end{bmatrix}$$

The state space form of the model (3.28-3.29) is

$$\left\{ \begin{array}{rcl} y_t & = & \left[\begin{array}{cc} x_{0,t} & x_{1,t} \end{array} \right] \left[\begin{array}{c} \beta_{0,t} \\ \beta_{1,t} \end{array} \right] + u_t \\ \left[\begin{array}{c} \beta_{0,t} \\ \beta_{1,t} \end{array} \right] & = & \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right] \left[\begin{array}{c} \beta_{0,t-1} \\ \beta_{1,t-1} \end{array} \right] + \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right] \left[\begin{array}{c} v_0 \\ v_1 \end{array} \right] \end{array} \right. \tag{3.30}$$

This example shows how to construct a time-variant state space model. Next, we use the `KFiltering` and the `KSmoothing` procedures to estimate the unobservable components $\beta_{0,t}$ and $\beta_{1,t}$.

142. **kalman6b.prg**
Maximum Likelihood of the model (3.30). Note the declaration of `sigma` as an external variable and the definition of `sigma` in the `ml` procedure.

143. **kfgain1-2.prg**
Illustration of the `KF_gain` procedure.

144. **kernel1.prg**
Density estimation (normal random number).

145. **kernel2.prg**
Density estimation (χ_2 random number) with the truncated (at left) normal kernel.

146. **kernel3.prg**
Density estimation (uniform random number) with the trunacted normal (at left and right) kernel.

147. **kernel4.prg**
We investigate the empirical probability density of the FRF/DEM return for different scales : 1, 2, 5, 10 and 30 days. We use the thresholding method to compare the "noisy" density with the "denoised" density.

148. **kpss.prg — kpss.src**
KPSS statistic.

149. **ks1.prg**
Computes the Kolmogorov-Smirnov test for a white noise process presented in Brockwell and Davis [1991].

150. **ks2.prg**
Computes the Kolmogorov-Smirnov test for a unit root process.

151. **ks3.prg**
Computes the Kolmogorov-Smirnov test for the random walk plus noise model applied to the *purse* data.

152. **ll1.prg**
Estimates the Local Level model for the *purse* data in the frequency domain with the method of scoring and the BFGS algorithm.

153. **ll2.prg**
Estimates the unobserved component of the *purse* Local Level model.

154. **ll3.prg**
Estimates the *purse* Local Level model in the time domain. This program shows the importance of the choice of the initial conditions.

155. **llt1.prg**
Estimates the Local Linear model for the *gnp* data in the frequency domain with the method of scoring and the BFGS algorithm.

156. **llt2.prg**
Estimates the unobserved component of the *gnp* Local Linear model.

157. **llt3.prg**
Estimates the *gnp* Local Linear model in the time domain with the BHHH algorithm.

158. **matrix1.prg**
Computes the matrices $\mathbf{L}_4$, $\mathbf{D}_4$ and $\mathbf{K}_{4,3}$.

159. **matrix2.prg**
Shows the difference between the `vech` and `vech_` operators.

160. **matrix3.prg**
Verifies the following propositions (LÜTKEPOHL [1991]) for $m = 1, \ldots, 10$

$$\mathbf{L}_m \mathbf{D}_m = I_{m(m+1)/2}$$

$$\mathbf{K}_{m,m} \mathbf{D}_m = \mathbf{D}_m$$

$$\mathbf{K}_{m,1} = \mathbf{K}_{1,m} = I_m$$

$$\text{trace}\left(\mathbf{K}_{m,m}\right) = m$$

$$\text{trace}\left(\mathbf{D}_m^\top \mathbf{D}_m\right) = m^2$$

$$\mathbf{L}_m \mathbf{L}_m^\top = I_{m(m+1)/2}$$

$$\text{trace}\left(\mathbf{D}_m^\top \mathbf{D}_m\right)^{-1} = \frac{m\left(m+3\right)}{4}$$

161. **matrix4-5.prg**
An illustration of the `xpnd2` procedure with real and complex matrices.

162. **matrix6a-6d.prg**
An illustration of the `Explicit_to_Implicit` and `Implicit_to_Explicit` procedures.

163. **missing1.prg**
Illustration of the `Missing` procedure.

164. **nw.prg — nw.src**
Newey and West estimator of the variance.

165. **optmum2a-2c.prg**
Some examples with the `optmum2` procedure.

166. **pdgm1.prg**
Computes the periodogram of the *Lynx* data and the Fisher's g statistic.

167. **pdgm2.prg**
Smoothed periodogram of a stable AR(2) process.

168. **pdgm3.prg**
Periodogram of the *sunspots* data.

169. **pdgm4.prg**
Covariogram of a time series using the `PDGM` and `inverse_fourier` procedures.

170. **pdgm5.prg**
We consider the state space model

$$\begin{cases} \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 \\ 0 & -0.5 \end{bmatrix} \begin{bmatrix} \alpha_{1,t-1} \\ \alpha_{2,t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \eta_t \end{cases} \tag{3.31}$$

with $\eta_t \sim \mathcal{N}(0, 0.25)$ and

$$\varepsilon_t \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.2 & 0 \\ 0 & 0.1 \end{bmatrix})$$

We compare the spectral density of the state space model with the smoothed periodogram of a realization of the model.

171. **pdgm6.prg**
We compare the theoretical covariances and the cross-covariances of the model (3.31) with the empirical covariances and the cross-covariances of a realization.

172. **pdgm7.prg**
Spectral density of a univariate ARMA(2,1) process.

173. **qmf1.prg**
In this example, we verify the following properties of quadrature mirror filters:

$$\begin{cases} \sum_{k=0}^{p} h_k &= \sqrt{2} \\ \sum_{k=0}^{p} g_k &= 0 \\ \sum_{k=0}^{p} h_k^2 &= 1 \\ \sum_{k=0}^{p} h_k g_k &= 0 \end{cases}$$

174. **qmf2.prg**
Same program as *qmf1.prg* with Pollen filters.

175. **riccati1.prg**
Solves the Algebraic Riccati Equation for the state space model given in exercise 4.8 by HARVEY [1990].

176. **rls1.prg**
We apply the `RLS` procedure to a time-invariant model.

177. **rls2.prg**
We apply the `RLS` procedure to a model whose coefficients follow a random walk process.

178. **robinson.prg — robinson.src**
Estimates the Hurst exponent with ROBINSON's [1995] method in the frequency domain.

179. **scalogrm.prg**
This example is taken from ARINO and VIDAKOVIC [1995]. The scalogram can be used to decompose a time series into different time series. We consider a time series x_t which is the sum of two time series y_t and z_t, that is we have

$$x_t = y_t + z_t$$

The first component y_t is a trend and the second component z_t is a cycle. Wavelet analysis is useful for describing the time series x_t because the trend is better localized for high scales (the cycle is better localized for the first scales).

180. **spectrum.prg**
There are several techniques to estimate the power spectrum with wavelet or wavelet packet. Firstly, we compute the periodogram. Secondly, we calculate the coefficients of the wavelet transform. Thirdly, we transform the coefficients. Finally, we compute the inverse wavelet transform. We can use thresholding techniques to perform the transformation. In this example, we transform the wavelet coefficients by extracting some subbands.

181. **ssm1-5.prg**
Printing state space models.

182. **ssm6a.prg**
Same program as *varx1e.prg*, but responses to forecast errors are computed with the *SSM_impulse* procedure.

183. **ssm6b.prg**
Same program as *varx1e.prg*, but responses to orthogonal impulses are computed with the *SSM_orthogonal* procedure.

184. **ssm6c.prg**
Same program as *varx1d.prg*, but we compute the forecast error variance decomposition with the *SSM_fevd* procedure.

185. **ssm7a.prg**
Same program as *arma1k.prg*, but responses to forecast errors are computed with the *SSM_impulse* procedure.

186. **ssm7b.prg**
Same program as *arma1k.prg*, but responses to orthogonal impulses are computed with the *SSM_orthogonal* procedure.

187. **ssm7c.prg**
Same program as *arma1j.prg*, but we compute the forecast error variance decomposition with the *SSM_fevd* procedure.

188. **ssm8a.prg**
We consider the state space model

$$\begin{cases} \begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 4 & 2 \\ 2 & -3 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \end{bmatrix} = \begin{bmatrix} .5 & .45 \\ -.5 & .8 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_{1,t} \\ \eta_{2,t} \end{bmatrix} \end{cases} \tag{3.32}$$

with

$$\varepsilon_t \sim \mathcal{N}\left(\mathbf{0}_3, \begin{bmatrix} 5 & 1 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}\right)$$

and

$$\begin{bmatrix} \eta_{1,t} \\ \eta_{2,t} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}_2, \begin{bmatrix} 2 & 0.5 \\ 0.5 & 1 \end{bmatrix}\right)$$

We compute the responses to the forecast error $\mathbf{e} = \begin{bmatrix} 1 & 0 \end{bmatrix}^\top$.

189. **ssm8b.prg**
We compute the responses to the forecast error $\mathbf{e} = \begin{bmatrix} 1 & -1 \end{bmatrix}^\top$ for the state space model (3.32).

190. **ssm9a.prg**
We compute the responses to the orthogonal impulse $\mathbf{e} = \begin{bmatrix} 1 & 0 \end{bmatrix}^\top$ for the state space model (3.32).

191. **ssm9b.prg**
We compute the responses to the orthogonal impulse $\mathbf{e} = \begin{bmatrix} 1 & -1 \end{bmatrix}^\top$ for the state space model (3.32).

192. **ssm10.prg**
We compute the forecast error variance decomposition for the state space model (3.32).

193. **surrog1.prg**
Surrogate data in the univariate case.

194. **surrog2.prg**
Surrogate data in the multivariate case.

195. **surrog3.prg — rk4.src**
Surrogate data can be used to detect non-linearities. In this example, we use the Lorenz model defined by

$$\left\{ \begin{array}{lcl} \frac{d\,x}{d\,t} & = & \sigma\,(y-x) \\ \frac{d\,y}{d\,t} & = & -xz + Rx - y \\ \frac{d\,z}{d\,t} & = & xy - \beta z \end{array} \right.$$

196. **tdml1a.prg**
Teräsvirta [1994] suggests a LSTAR model to fit the *lynx* data

$$\begin{array}{rcl} x_t & = & \beta_1 x_{t-1} + [\beta_2 x_{t-2} + \beta_3 x_{t-3} + \beta_4 x_{t-4} + \beta_5 x_{t-9} + \beta_6 x_{t-11}] \\ & & \times [1 + \exp(\rho \times 1.8\,(x_{t-3} - \theta))]^{-1} + u_t \end{array} \tag{3.33}$$

This program estimates the model (3.33). Note the use of the external variable `_tsm_parnm` for the names of the estimated coefficients.

197. **tdml2a.prg**
To model the *lynx* data, Ozaki [1982] suggests to use the EXPAR model

$$x_t = \begin{array}{l} \left[\beta_1 + (\beta_2 + \beta_3 x_{t-1}) \exp\left(-\delta x_{t-1}^2\right)\right] x_{t-1} \\ \left[\beta_4 + (\beta_5 + \beta_6 x_{t-1}) \exp\left(-\delta x_{t-2}^2\right)\right] x_{t-2} + u_t \end{array}$$

with $u_t \sim \mathcal{N}\left(0, \sigma^2\right)$. With the `TD_ml` procedure, we estimate the coefficients $\theta = \left[\begin{array}{ccc} \beta^\top & \delta & \sigma \end{array}\right]^\top$.

198. **tdml2b.prg**
This is a modification of the *tdml2a.prg* example by setting δ to 3.89.

199. **tdml3a.prg**
Maximum Likelihood estimation of a linear model.

200. **tdml3b.prg**
Maximum Likelihood estimation of a linear model under linear restrictions.

201. **tdml4a.prg**
Maximum Likelihood of the linear model with AR(1) errors:

$$\left\{ \begin{array}{lcl} y_t & = & x_t \beta + u_t \\ u_t & = & \rho u_{t-1} + \varepsilon_t \end{array} \right.$$

with $\varepsilon_t \sim N\left(0, \sigma^2\right)$. The parameter vector θ is $\left[\begin{array}{ccc} \beta^\top & \rho & \sigma \end{array}\right]^\top$. The ML function is based on Beach and MacKinnon [1978]. We test also $\rho = 0$ with LM and LR statistics.

202. **tdml4b.prg**
Maximum Likelihood of a PROBIT model. The program contains the normality test for PROBIT models of Bera, Jarque and Lee [1984]. Note that the ML procedure uses the analytical Jacobian.

203. **twofft.prg**
An illustration of the `fourier2` procedure.

204. **varx1a.prg**
Define the following series with the *Lutkepohl* data

$$\begin{array}{rcl} y_{1,t} & = & INV(t) - INV(t-1) \\ y_{2,t} & = & INC(t) - INC(t-1) \\ y_{3,t} & = & CONS(t) - CONS(t-1) \end{array}$$

The program estimates the VAR(2) process

$$\left[\begin{array}{c} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{array}\right] = \Phi_1 \left[\begin{array}{c} y_{1,t-1} \\ y_{2,t-1} \\ y_{3,t-1} \end{array}\right] + \Phi_2 \left[\begin{array}{c} y_{1,t-2} \\ y_{2,t-2} \\ y_{3,t-2} \end{array}\right] + \left[\begin{array}{c} \mu_1 \\ \mu_2 \\ \mu_3 \end{array}\right] + \varepsilon_t \tag{3.34}$$

and performs a stability analysis. The θ vector of coefficients corresponds to

$$\begin{bmatrix} \text{vec}(\Phi_1) \\ \text{vec}(\Phi_2) \\ \mu \end{bmatrix}$$

205. **varx1b.prg**
Computes the Wald test for no Granger-causality from INC/CONS to INV.

206. **varx1c.prg**
Computes the Wald test for no Instantaneous-causality between INC/CONS and INV.

207. **varx1d.prg**
Forecast Error Variance Decomposition of the above VAR(2) model.

208. **varx1e.prg**
Impulses Responses of the above VAR(2) model.

209. **varx1f.prg**
Impulses Responses of the above VAR(2) model (graphical representation).

210. **varx1g.prg**
VAR order selection with the BIC, AIC alpha, SIC, FPE, AIC and HQ criteria.

211. **varx1h.prg**
Estimates the model (3.34) with the restrictions

$$\Phi_1 = \begin{bmatrix} \cdot & 0 & 0 \\ 0 & 0 & \cdot \\ 0 & \cdot & \cdot \end{bmatrix}$$

$$\Phi_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \cdot & 0 \end{bmatrix}$$

and

$$\mu = \begin{bmatrix} 0 \\ \cdot \\ \cdot \end{bmatrix}$$

212. **varx2a.prg**
Estimation of the Dynamic Simultaneous Equations

$$\begin{bmatrix} \text{INC}_t \\ \text{CONS}_t \end{bmatrix} = \Phi_1 \begin{bmatrix} \text{INC}_{t-1} \\ \text{CONS}_{t-1} \end{bmatrix} + \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \text{INV}_{t-1} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} \tag{3.35}$$

213. **varx2b.prg**
Estimation of the Constrained Dynamic Simultaneous Equations

$$\begin{bmatrix} \text{INC}_t \\ \text{CONS}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} \text{INC}_{t-1} \\ \text{CONS}_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \mu_2 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ 0 \end{bmatrix} \text{INV}_{t-1} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} \tag{3.36}$$

214. **varx2c.prg**
Estimation of the model (3.36) by Maximum Likelihood.

215. **varx3a.prg**
Use of the `varx_ls` procedure to compute OLS estimates. The results are compared with those calculated with the `ols` procedure.

216. **varx3b.prg**
We use the `varx_cls` procedure to compute the SUR estimator. The example is taken from JUDGE, HILL, GRIFFITHS, LÜTKEPOHL and LEE [1988] pages 453 and 454.

217. **varx3c.prg**
We use the `varx_cls` procedure to compute the restricted SUR estimator. The example is taken from JUDGE, HILL, GRIFFITHS, LÜTKEPOHL and LEE [1988] pages 460 to 462.

218. **varx3d.prg**
We use the `varx_cls` procedure to estimate a system of simultaneous equations. The example is taken from JUDGE, HILL, GRIFFITHS, LÜTKEPOHL and LEE [1988] pages 656 to 663.

219. **window2a-2b.prg**
Somes examples to show the use of the `window2` procedure.

220. **wn1.prg**
Estimates the white noise model in the frequency domain

$$y_t = \varepsilon_t$$

with $\varepsilon_t \sim \mathcal{N}\left(0, \sigma^2\right)$. Then we plot the empirical distribution of $2\frac{I(\lambda_j)}{g(\lambda_j)}$ and the theoretical χ^2_2 distribution.

221. **wn2.prg**
This is the same program as *wn1.prg* applied to a unit root process.

222. **wn3.prg**
We check if the FRF/DEM is a unit root process.

223. **wpkt1.prg**
We select a wavelet packet basis for a time series with the `BestBasis` and the `BestLevel` procedures based on the entropy cost function.

224. **wpkt2.prg**
We simulate a fractional process. Then, we draw the wavelet packet table of this process. We illustrate the fact that the wavelet transform is a special case of the wavelet packet transform with a special basis.

225. **wt1a.prg-wt1b.prg**
We evaluate the inverse wavelet transform for different unit vectors $\mathbf{e}$ to see how wavelets look like (see PRESS, TEUKOLSKY, VETTERLING and FLANNERY [1992], page 591).

226. **wt2.prg**
Reconstruction of an ARMA process by the quantile thresholding method with different values of p.

227. **wt3.prg**
An important result is that the "mother" coefficient of the wavelet transform of a time series x_t of length $N = 2^M$ is equal to

$$\mathbf{c}_0 = \sum_{t=1}^{N} x_t \Big/ 2^{\frac{M}{2}}$$

3.2.2 Time series

3.2.2.1 Arfima process

```
new;
library tsm,optmum;

output file = tsm1a.out reset;

rndseed 123456;

p = 1; q = 1; beta = 0.8|0.4|0.25; sigma = 2;          /* d = 0.25 */
Nobs = 500;
{y,retcode} = RND_arfima(beta,p,q,sigma,1000,Nobs,1);  /* ARFIMA process simulation */
```

```
sv = 0.8|0.4|0.25|2;

@<--- first estimation --->@

cnt = 0|0|0|0;                                          /* No fixed parameters */

_tsm_Mcov = 1;    /* Hessian Matrix Covariance */
_fourier = 0;     /* Fast fourier transform    */
_tsm_optmum = 0;  /* scoring algorithm         */
__output = 1;
_print = 1;

{beta1,stderr,Mcov,Logl1} = arfima(y,1,1,sv,cnt);

_tsm_optmum = 1;  /* BFGS algorithm         */
__output = 0;
_print = 0;

@<--- second estimation --->@

sv = 0.8|0.4|0.25|2;
cnt = 0|0|1|0;                                          /* d is fixed */

{beta2,stderr,Mcov,Logl2} = arfima(y,1,1,sv,cnt);

@<--- third estimation --->@

sv = 0.8|0.4|0.25|2;
cnt = 0|1|1|0;                                          /* d and MA parameters are fixed */

{beta3,stderr,Mcov,Logl3} = arfima(y,1,1,sv,cnt);

@<--- likelihood ratio tests --->@

LR = 2*(logl1-logl2);      /* Likelihood ratio                              */
pvalue = cdfchic(LR,1);    /* Approximating the noncentral chi-squared CDF */

print;
print ''Testing d = 0.25''; print chrs(45*ones(40,1));
print ftos(LR,''Likelihood ratio statistic: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);
print; print;

LR = 2*(logl1-logl3);      /* Likelihood ratio                              */
pvalue = cdfchic(LR,2);    /* Approximating the noncentral chi-squared CDF */

print ''Testing d = 0.25 and theta1 = 0.4''; print chrs(45*ones(40,1));
print ftos(LR,''Likelihood ratio statistic: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);

output off;


Total observations:                                    500
Usable observations:                                   500
Number of parameters to be estimated:                    4
Degrees of freedom:                                    496
Value of the maximized log-likelihood function:  -1093.83332


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------
phi1            0.886292        0.069672        12.720846        0.000000
theta1          0.354264        0.147613         2.399952        0.016765
d               0.095775        0.211379         0.453095        0.650678
sigma           2.049915        0.064080        31.989992        0.000000

Covariance matrix: inverse of the negative Hessian.

Testing d = 0.25
----------------------------------------
Likelihood ratio statistic:    0.52048
p-value:    0.47064


Testing d = 0.25 and theta1 = 0.4
----------------------------------------
Likelihood ratio statistic:    1.23108
p-value:    0.54035
```

3.2.2.2 Varma process

- Estimation of a Varma model and impulse functions.

```
/*
**  Estimation of a Vector ARMA(1,1) process
**  Conditional Maximum Likelihood
**
**  Reinsel, G.C. [1993], Elements of Multivariate Time Series Analysis,
**  Springer-Verlag, New York
*/

new;
library tsm,optmum;

output file = tsm1b.out reset;

@<--- data --->@

load reinsel[100,2] = reinsel.asc;

invest = reinsel[.,1];
invent = reinsel[.,2];
di = invest - lag1(invest);
y = di~invent;

_print = 1;
_tsm_optmum = 0;   /* Analytical Newton-Raphson algorithm */
_tsm_gtol = 0.001;


@<--- Unrestricted model --->@

sv = ''HR2''; /* Hannan-Rissanen method (second form) */

{beta1,stderr1,Mcov1,LogL1} = arma_ML(y,1,1,''HR2'');
SIGMA1 = _arma_SIGMA;


@<--- Restricted model (implicit form) --->@

/*
**  4 restrictions:
**
**  AR1_21 = 0
**  AR1_12 = 0
**  MA1_21 = 0
**  MA1_12 = 0
*/

RR = design(1|0|0|2|3|0|0|4); r = zeros(8,1);

sv = 0; /* Lutkepohl method */

{beta2,stderr2,Mcov2,LogL2} = arma_CML(y,1,1,sv,RR,r);
SIGMA2 = _arma_SIGMA;


@<--- Testing restrictions --->@

H = vread(_ml_derivatives,''H_matrix'');  /* Hessian matrix (restricted model) */
G = vread(_ml_derivatives,''G_matrix'');  /* Gradient matrix (restricted model) */

I = - H;                                  /* Approximation of the Information matrix */


/*
**  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,4);    /* Approximating the noncentral chi-squared CDF */

print;
print ftos(LR,''Likelihood ratio statistic: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);


/*
**  Lagrange multiplier
**
**  DAVIDSON and MACKINNON [1993], Estimation and Inference in Econometrics
**  Oxford University Press, page 437, formula (13.04)
*/
```

```
LM = G'*inv(I)*G;
pvalue = cdfchic(LM,4);

print;
print ftos(LM,''Lagrange multiplier: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);


/*
**  Wald test
**
**  DAVIDSON and MACKINNON [1993], Estimation and Inference in Econometrics
**  Oxford University Press, page 437, formula (13.05)
*/

proc restrict(beta);
  local r;
  r = beta[2]|beta[3]|beta[6]|beta[7];
  retp(r);
endp;

Dr = gradp(&restrict,beta1);
r = restrict(beta1);

wald = r'*invpd(Dr*Mcov1*Dr')*r;
pvalue = cdfchic(wald,4);

print;
print ftos(wald,''Wald statistic: %lf'',10,5);
print ftos(pvalue,''p-value: %lf'',10,5);
print; print;


@<--- Responses to Forecast Errors (unrestricted model) --->@

call arma_impulse(beta1,1,1,8);

z = miss(zeros(2,1),0);

mask = ones(1,5);
let fmt[5,3]=
    ''*.*lf'' 10 3
    ''*.*lf'' 10 3
    ''*.*lf'' 25 0
    ''*.*lf'' 10 3
    ''*.*lf'' 10 3;

i = 0;
do until i>8;
  x = varget(''IMPULSE''$+ftos(i,''%lf'',1,0));
  y = varget(''_IMPULSE''$+ftos(i,''%lf'',1,0));
  w = x~z~y;

  print chrs(45*ones(79,1));
  print ftos(i,''Periods: %lf'',1,0);
  print chrs(45*ones(79,1));
  print ''        Estimated responses                  Accumulated responses'';
  print ''            PHI matrix                           PSI matrix      '';
  print chrs(45*ones(79,1));
  call printfm(w,mask,fmt);
  print;
  i = i+1;
endo;

print; print;


@<--- Responses to Orthogonal Impulses (unrestricted model) --->@

let fmt[5,3]=
    ''*.*le'' 10 2
    ''*.*le'' 10 2
    ''*.*lf'' 25 0
    ''*.*le'' 10 2
    ''*.*le'' 10 2;

call arma_orthogonal(beta1,1,1,SIGMA1,8);

i = 0;
do until i>8;
  x = varget(''IMPULSE''$+ftos(i,''%lf'',1,0));
  y = varget(''_IMPULSE''$+ftos(i,''%lf'',1,0));
  w = x~z~y;

  print chrs(45*ones(79,1));
```

```
  print ftos(i,''Periods: %lf'',1,0);
  print chrs(45*ones(79,1));
  print ''        Estimated responses                  Accumulated responses'';
  print ''           THETA matrix                          KSI matrix      '';
  print chrs(45*ones(79,1));
  call printfm(w,mask,fmt);
  print;
  i = i+1;
endo;


output off;


Total observations:                                      100
Usable observations:                                      99
Number of parameters to be estimated:                      8
Degrees of freedom:                                       90
Value of the maximized log-likelihood function:       -501.39229


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
AR1_11          0.348233        0.166432         2.092351        0.039223
AR1_21          0.660844        0.229693         2.877072        0.005012
AR1_12         -0.018264        0.045996        -0.397081        0.692248
AR1_22          0.845953        0.047469        17.821190        0.000000
MA1_11         -0.222574        0.174794        -1.273349        0.206173
MA1_21          0.253697        0.316177         0.802391        0.424441
MA1_12         -0.177884        0.069941        -2.543330        0.012687
MA1_22          0.239739        0.111043         2.158965        0.033513

Covariance matrix: inverse of the negative Hessian.

Total observations:                                      100
Usable observations:                                      99
Number of parameters to be estimated:                      4
Degrees of freedom:                                       94
Value of the maximized log-likelihood function:       -513.38787


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
AR1_11          0.284297        0.182115         1.561083        0.121865
AR1_21          0.000000        0.000000                .               .
AR1_12          0.000000        0.000000                .               .
AR1_22          0.895241        0.051069        17.529942        0.000000
MA1_11         -0.266600        0.187651        -1.420724        0.158706
MA1_21          0.000000        0.000000                .               .
MA1_12          0.000000        0.000000                .               .
MA1_22          0.198175        0.114039         1.737787        0.085523

Covariance matrix: inverse of the negative Hessian.

Likelihood ratio statistic:   23.99116
p-value:    0.00008

Lagrange multiplier:   22.77949
p-value:    0.00014

Wald statistic:   25.75092
p-value:    0.00004


-------------------------------------------------------------------------------
Periods: 0
-------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
            PHI matrix                           PSI matrix
-------------------------------------------------------------------------------
     1.000     0.000                       .     1.000     0.000
     0.000     1.000                       .     0.000     1.000

-------------------------------------------------------------------------------
Periods: 1
-------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
            PHI matrix                           PSI matrix
-------------------------------------------------------------------------------
     0.571     0.160                       .     1.571     0.160
     0.407     0.606                       .     0.407     1.606

-------------------------------------------------------------------------------
Periods: 2
-------------------------------------------------------------------------------
```

```
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
     0.191     0.045                          .     1.762     0.204
     0.722     0.618                          .     1.129     2.225

--------------------------------------------------------------------------------
Periods: 3
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
     0.053     0.004                          .     1.816     0.208
     0.737     0.552                          .     1.866     2.777

--------------------------------------------------------------------------------
Periods: 4
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
     0.005    -0.009                          .     1.821     0.200
     0.659     0.470                          .     2.524     3.247

--------------------------------------------------------------------------------
Periods: 5
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
    -0.010    -0.012                          .     1.811     0.188
     0.561     0.392                          .     3.085     3.639

--------------------------------------------------------------------------------
Periods: 6
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
    -0.014    -0.011                          .     1.797     0.177
     0.468     0.324                          .     3.553     3.963

--------------------------------------------------------------------------------
Periods: 7
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
    -0.013    -0.010                          .     1.783     0.167
     0.386     0.267                          .     3.939     4.230

--------------------------------------------------------------------------------
Periods: 8
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
            PHI matrix                              PSI matrix
--------------------------------------------------------------------------------
    -0.012    -0.008                          .     1.772     0.159
     0.318     0.219                          .     4.257     4.449



--------------------------------------------------------------------------------
Periods: 0
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
           THETA matrix                             KSI matrix
--------------------------------------------------------------------------------
 2.39e+000 0.00e+000                          . 2.39e+000 0.00e+000
 1.03e+000 4.09e+000                          . 1.03e+000 4.09e+000

--------------------------------------------------------------------------------
Periods: 1
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
           THETA matrix                             KSI matrix
--------------------------------------------------------------------------------
 1.53e+000 6.53e-001                          . 3.91e+000 6.53e-001
 1.59e+000 2.48e+000                          . 2.62e+000 6.57e+000

--------------------------------------------------------------------------------
Periods: 2
--------------------------------------------------------------------------------
        Estimated responses                    Accumulated responses
           THETA matrix                             KSI matrix
```

```
--------------------------------------------------------------------------------
 5.02e-001 1.82e-001                       . 4.42e+000 8.35e-001
 2.36e+000 2.53e+000                       . 4.98e+000 9.09e+000

--------------------------------------------------------------------------------
Periods: 3
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
 1.32e-001 1.72e-002                       . 4.55e+000 8.52e-001
 2.33e+000 2.26e+000                       . 7.30e+000 1.14e+001

--------------------------------------------------------------------------------
Periods: 4
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
 3.45e-003-3.53e-002                       . 4.55e+000 8.17e-001
 2.06e+000 1.92e+000                       . 9.36e+000 1.33e+001

--------------------------------------------------------------------------------
Periods: 5
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
-3.63e-002-4.74e-002                       . 4.52e+000 7.69e-001
 1.74e+000 1.60e+000                       . 1.11e+001 1.49e+001

--------------------------------------------------------------------------------
Periods: 6
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
-4.44e-002-4.58e-002                       . 4.47e+000 7.23e-001
 1.45e+000 1.32e+000                       . 1.25e+001 1.62e+001

--------------------------------------------------------------------------------
Periods: 7
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
-4.19e-002-4.01e-002                       . 4.43e+000 6.83e-001
 1.20e+000 1.09e+000                       . 1.37e+001 1.73e+001

--------------------------------------------------------------------------------
Periods: 8
--------------------------------------------------------------------------------
        Estimated responses                  Accumulated responses
           THETA matrix                           KSI matrix
--------------------------------------------------------------------------------
-3.64e-002-3.39e-002                       . 4.39e+000 6.49e-001
 9.84e-001 8.96e-001                       . 1.47e+001 1.82e+001
```

3.2.2.3 Varx Process

- VAR order selection with the BIC, AIC alpha, SIC, FPE, AIC and HQ criteria.

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  VAR order selection
**  See LUTKEPOHL, chapter 4
**
*/

new;
library tsm,optmum;

output file = tsm1c.out reset;

cls;

@<--- data --->@

load y[92,3] = lutkepoh.asc;

y = ln(y[1:76,.]);
```

```
INVEST = y[.,1];
dinv = INVEST-lag1(INVEST);
INCOME = y[.,2];
dinc = INCOME-lag1(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lag1(CONSUM);
data = dinv~dinc~dcons;

@<--- VARX order selection --->@

_print = 1;

{CR,p} = criteria(data,1,5);

output off;



/*
**  A procedure to select the order of a VARX model
*/

proc (2) = criteria(Y,X,p_max);
  local old,K,T,m,CR,L,D,SIGMA;
  local omat,fmt,mask,p;

  old = _print;
  K = cols(Y);
  CR = zeros(p_max,7);

  m = 1;
  do until m > p_max;

    _print = 0;
    call varx_ML(Y,X,m);
    T = rows(packr(Y)) - m;

    SIGMA = _varx_SIGMA;
    D = det(SIGMA);
    L = ln(D);

    CR[m,1] = L+(K^2)*m*ln(T)/T;                  /* BIC */
    CR[m,2] = L+3*m*(K^2)/T;                      /* AIC alpha = 3 */
    CR[m,3] = L+(1+m*(K^2)/T)/(1-(m*(K^2)-2)/T);  /* AICc */
    CR[m,4] = D*(1+2*(m*(K^2)+1)/T)^K;            /* SIC  */
    CR[m,5] = D*(((T+K*m+1)/(T-K*m-1))^K);        /* MFPE */
    CR[m,6] = L+2*(K^2)*m/T;                      /* AIC  */
    CR[m,7] = L+2*(K^2)*m*ln(ln(T))/T;            /* HQ   */

    m = m + 1;
  endo;

  P = minindc(CR);
  _print = old;

  if _print == 1;

    m = seqa(1,1,p_max);

    print chrs(45*ones(79,1));
    print ''Lags      BIC       AICa      AICc        SIC         FPE       AIC''\
          ''       HQ'';
    print chrs(45*ones(79,1));

    omat = ''Opt.''~p';
    mask = 0~1~1~1~1~1~1;
    let fmt[8,3] =
       ''-*.*s''  4 4
       ''*.*lf'' 8  0
       ''*.*lf'' 10 0
       ''*.*lf'' 10 0
       ''*.*lf'' 12 0
       ''*.*lf'' 12 0
       ''*.*lf'' 10 0
       ''*.*lf'' 10 0;
    call printfm(omat,mask,fmt); print; print chrs(45*ones(79,1));

    omat = m~CR;
    mask = 1;
    let fmt[8,3] =
       ''*.*lf''  4 0
       ''*.*lf'' 10 3
       ''*.*lf'' 10 3
       ''*.*lf'' 10 3
```

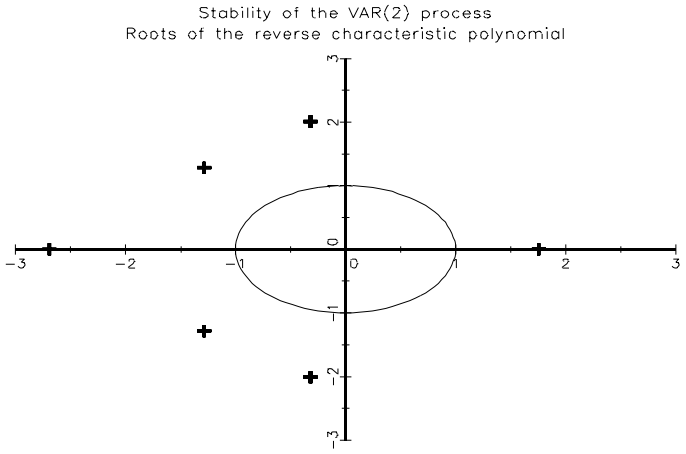
```
      ''*.*lf'' 12 3
      ''*.*lf'' 12 3
      ''*.*lf'' 10 3
      ''*.*lf'' 10 3;
    call printfm(omat,mask,fmt);
  endif;

  retp(CR,p);
endp;
```

Lags	BIC	AICa	AICc	SIC	FPE	AIC	HQ
Opt.	1	2	2	1	2	2	2
1	-24.213	-24.372	-23.498	0.000	0.000	-24.493	-24.382
2	-24.067	-24.385	-23.528	0.000	0.000	-24.632	-24.407
3	-23.575	-24.054	-23.072	0.000	0.000	-24.429	-24.089
4	-23.210	-23.850	-22.480	0.000	0.000	-24.357	-23.901
5	-22.895	-23.697	-21.367	0.000	0.000	-24.340	-23.766

▶ VAR estimation and stability analysis.

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  Estimation of a VAR(2) process and stability analysis
**  See LUTKEPOHL, section 3.2.3
**
**  Note: the constant is the FIRST column of B in LUTKEPOHL
**        and the LAST column of B in this program
*/

new;
library tsm,optmum,pgraph;

output file = tsm1d.out reset;

cls;


@<--- data --->@

load y[92,3] = lutkepoh.asc;

y = ln(y[1:76,.]);

INVEST = y[.,1];
dinv = INVEST-lag1(INVEST);
INCOME = y[.,2];
dinc = INCOME-lag1(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lag1(CONSUM);
data = dinv~dinc~dcons;


@<--- LS estimation --->@

{theta,stderr,Mcov,LOGL1} = varx_LS(data,1,2);  /* Constant and 2 lags */
print;
print ''SIGMA:'';
print _varx_sigma;


@<--- Stability analysis --->@

beta = theta[1:18];   /* Estimates of the AR part */
roots = arma_roots(beta,2,0);

print; print chrs(45*ones(79,1));
print ''                  STABILITY analysis of the VAR(2) process  '';
print chrs(45*ones(79,1));
print ''        Roots of the reverse            Modulus'';
print ''      characteristic polynomial'';
print chrs(45*ones(79,1));
omat = roots~abs(roots);
mask=1~1;
let fmt[2,3]=
    ''*.*lf'' 14 5
    ''*.*lf'' 14 5;
call printfm(omat,mask,fmt);
```

```
graphset;
  _pdate = ''''; _pcross = 1; _pframe = {0,0};
  title(''Stability of the VAR(2) process''\
        ''\LRoots of the reverse characteristic polynomial'');
  scale(-3|3,-3|3);

  r = real(roots); i = imag(roots); Nr = rows(roots);
  _psym = r~i~ones(Nr,5).*(11~5~2~1~1);
  t = seqa(0,2*pi/100,101); x=cos(t); y=sin(t);

  graphprt(''-c=1 -cf=tsm1d.eps'');
  xy(x,y);


output off;


Total observations:                                     76
Usable observations:                                    73
Number of parameters to be estimated:                   21
Degrees of freedom:                                     52
Value of the maximized log-likelihood function:    595.26885


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01            -0.319631        0.125456        -2.547745        0.013837
P02             0.043931        0.031859         1.378910        0.173825
P03            -0.002423        0.025676        -0.094354        0.925190
P04             0.145989        0.545666         0.267543        0.790110
P05            -0.152732        0.138570        -1.102199        0.275450
P06             0.224813        0.111678         2.013052        0.049300
P07             0.961219        0.664310         1.446943        0.153915
P08             0.288502        0.168700         1.710150        0.093199
P09            -0.263968        0.135960        -1.941514        0.057624
P10            -0.160551        0.124907        -1.285368        0.204359
P11             0.050031        0.031720         1.577281        0.120796
P12             0.033880        0.025564         1.325330        0.190856
P13             0.114605        0.534570         0.214387        0.831084
P14             0.019166        0.135752         0.141182        0.888272
P15             0.354912        0.109407         3.243976        0.002062
P16             0.934394        0.665096         1.404900        0.165997
P17            -0.010205        0.168899        -0.060420        0.952053
P18            -0.022230        0.136120        -0.163312        0.870906
P19            -0.016722        0.017226        -0.970720        0.336181
P20             0.015767        0.004375         3.604272        0.000701
P21             0.012926        0.003526         3.666287        0.000579

Covariance matrix: inverse of the negative Hessian.

SIGMA:

    0.0021296289   7.1616667e-005    0.00012324036
  7.1616667e-005    0.00013733773   6.1458668e-005
   0.00012324036   6.1458668e-005   8.9203514e-005

-------------------------------------------------------------------------------
                STABILITY analysis of the VAR(2) process
-------------------------------------------------------------------------------
       Roots of the reverse            Modulus
      characteristic polynomial
-------------------------------------------------------------------------------
       1.75294                         1.75294
      -0.31951 -        2.00842i       2.03368
      -0.31951 +        2.00842i       2.03368
      -2.69403                         2.69403
      -1.28511 -        1.28023i       1.81398
      -1.28511 +        1.28023i       1.81398
```



▶ Estimation of dynamic simultaneous equations.

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  Estimation of a System of Dynamic Simultaneous Equations
**  See LUTKEPOHL, chapter 10
**
*/

new;
library tsm,optmum;

output file = tsm1e.out reset;

cls;


@<--- data --->@

load y[92,3] = lutkepoh.asc;

y = ln(y[1:76,.]);

INVEST = y[.,1];
INCOME = y[.,2];
CONSUMP = y[.,3];

X = ones(76,1)~lag1(invest);
Y = INCOME~CONSUMP;


/*
**  Estimation of the reduced form (10.2.4)
**  See LUTKEPOHL, page 326
*/

{theta,stderr,Mcov,LOGL1} = varx_LS(Y,X,1);

BB = reshape(theta,4,2)';

BB = (''INC(t)''|''CONS(t)'')~BB;
print; print;
print ''             INC(t-1)   CONS(t-1)    Constant    Inv(t-1)'';
let fmt[5,3] = ''-*.*s'' 8 8
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4;
call printfm(BB,0~1~1~1~1,fmt);
print;


/*
**  Estimation of the reduced form (10.2.4)
**  See LUTKEPOHL, page 326
*/
```

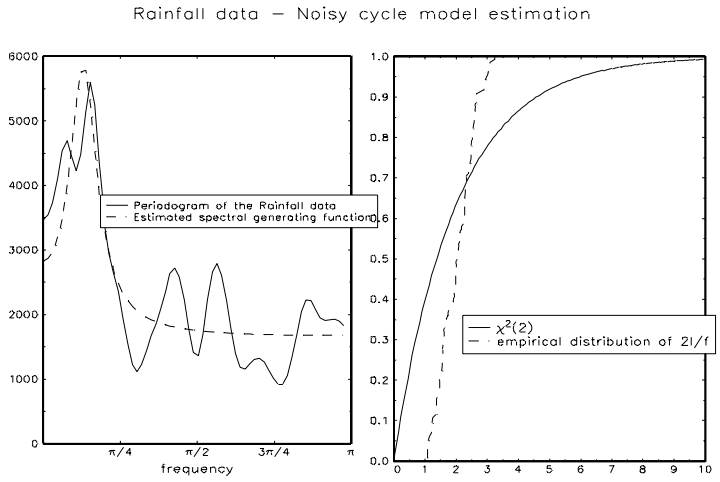
```
/*
**  We impose the following restrictions:
**
**  The INC(t-1) coefficient in the INC(t) equation is 1
**  The CONS(t-1) coefficient in the INC(t) equation is 0
**  The constant in the INC(t) equation is 0
**  The INV(t-1) coefficient in the CONS(t) equation is 0
*/

RR = design(0|1|0|2|0|3|4|0); r = 1|zeros(7,1);

{theta,stderr,Mcov,LOGL1} = varx_CLS(Y,X,1,RR,r);

BB = reshape(theta,4,2)';

BB = (''INC(t)''|''CONS(t)'')~BB;
print; print;
print ''             INC(t-1)   CONS(t-1)    Constant    Inv(t-1)'';
let fmt[5,3] = ''-*.*s'' 8 8
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4
               ''*.*lf'' 12 4;
call printfm(BB,0~1~1~1~1,fmt);
print;


output off;
```

```
Total observations:                                        76
Usable observations:                                       74
Number of parameters to be estimated:                       8
Degrees of freedom:                                        66
Value of the maximized log-likelihood function:        472.49156


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01             0.949956        0.098749         9.619858        0.000000
P02             0.299401        0.079091         3.785501        0.000334
P03             0.021842        0.098227         0.222363        0.824718
P04             0.690517        0.078673         8.777017        0.000000
P05             0.026800        0.026661         1.005209        0.318468
P06             0.068595        0.021354         3.212302        0.002037
P07             0.032432        0.017825         1.819475        0.073375
P08            -0.003768        0.014276        -0.263912        0.792670

Covariance matrix: inverse of the negative Hessian.


             INC(t-1)   CONS(t-1)    Constant    Inv(t-1)
INC(t)        0.9500      0.0218      0.0268      0.0324
CONS(t)       0.2994      0.6905      0.0686     -0.0038


Total observations:                                        76
Usable observations:                                       74
Number of parameters to be estimated:                       4
Degrees of freedom:                                        70
Value of the maximized log-likelihood function:        468.72790


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01             1.000000        0.000000                .               .
P02             0.287130        0.063200         4.543167        0.000023
P03             0.000000        0.000000                .               .
P04             0.702438        0.065889        10.660848        0.000000
P05             0.000000        0.000000                .               .
P06             0.050349        0.017598         2.861154        0.005561
P07             0.003460        0.000245        14.111550        0.000000
P08             0.000000        0.000000                .               .

Covariance matrix: inverse of the negative Hessian.


             INC(t-1)   CONS(t-1)    Constant    Inv(t-1)
INC(t)        1.0000      0.0000      0.0000      0.0035
CONS(t)       0.2871      0.7024      0.0503      0.0000
```

3.2.2.4 Structural models

```
/*
**  HARVEY [1990], Forecasting, Structural Time Series and
**  the Kalman Filter, Cambridge University Press, pages 86-89
**
**  Cycle Model
*/

new;
library tsm,optmum,pgraph;


@<--- data --->@

load rainfall[] = rainfall.asc;

y = rainfall[1:131];
y = y - 142.2;

_cycle_prmt = 0;
_tsm_parnm = ''rho''|''lambda_c''|''sig_kappa''|''sig_epsilon'';
_tsm_optmum = 1;
_print = 1;


@<--- Noisy cycle model estimation --->@

/*
**  Spectral generating function for a stochastic cycle plus noise model
*/

proc sgf(theta,lambda);
  local g_cycle,g_noise,g;
  g_cycle = _cycle_sgf(theta[1:3],lambda);
  g_noise = theta[4]^2;
  g = g_cycle + g_noise;
  retp(g);
endp;


/*
**  Using Harvey's starting values, page 198
*/

rho = 0.5; lambda_c = 0.78; sig_epsilon = 1; sig_kappa = 1;
sv = rho|lambda_c|sig_epsilon|sig_kappa;

{theta,stderr,Mcov,Logl} =  FD_ml(y,&sgf,sv);


@<--- Periodogram and spectral function estimation --->@

{lambda,I} = PDGM(y);
_smoothing = 6|5|0|0.23;   /* Parzen lag window with bandwidth = 6   */
I = smoothing(I);          /* smoothed periodogram                   */
g = sgf(theta,lambda);     /* estimated spectral generating function */
q = trunc(rows(lambda)/2);


/*
**  PAWITAN and O'SULLIVAN [1994], Nonparametric spectral density estimation
**  using penalized whittle likelihood,
**  Journal of the American Statistical Association, pages 600-610
*/

/*
**  I: periodogram
**  f: spectral density function
**
**  The autors use the fact that 2I/f is asymptotically
**  distributed as a chi-squared function with 2 degrees of freedom
**  to plot the observed quantiles of 2I/f against
**  the quantiles of the chi(2) distribution.
*/

z = 2*I./g;

x1 = sortc(z,1); Nobs = rows(x1);  y1 = seqa(1,1,Nobs)/Nobs;
x2 = seqa(0,10/Nobs,Nobs);         y2 = 1 - cdfchic(x2,2);


graphset;
  begwind;
  window2(0,2,1,0.75);
```

```
setwind(1);

  _pnum = 0; _paxes = 0; _ptitlht = 1.5;
  title(''Rainfall data - Noisy cycle model estimation'');
  draw;

graphset;

setwind(2);

  _pdate = ''''; _pnum = 2; _pnumht = 0.20; _paxht = 0.25;
  fonts(''simplex simgrma'');
  _plegstr = ''Periodogram of the Rainfall data''\
           ''\000Estimated spectral generating function'';
  _plegctl = {2 5 2.5 4};
  xlabel(''frequency'');
  xtics(0,pi,pi/4,0);
  lab = '' 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201'';
  asclabel(lab,0);
  xy(lambda[1:q],I[1:q]~g[1:q]);

setwind(3);

  _plegstr = ''\202h\201[2](2)\000empirical distribution of 2I/f'';
  _plegctl = {2 6 2.5 2};
  let w = {.,.};
  scale(0|10,w);
  asclabel(0,0);
  xlabel('''');
  xy(x2~x1,y2~y1);

  graphprt(''-c=1 -cf=tsm1f.eps'');
endwind;
```



3.2.3 Estimation methods

3.2.3.1 Maximum likelihood

- An example with EXPAR model.

```
/*
**  OZAKI [1982], The statistical analysis of perturbed limit cycle process
**  using nonlinear time series models, Journal of Times Series Analysis,
**  3, pages 29-41
**
**  See also:
**  TONG [1990], Non-linear Time Series, Oxford University Press
**
**  Estimate an EXPAR model
**  Note: Tong suspects that there has been a misprint or a computing error.
**        That is why we don't find OZAKI's estimates
**
*/

new;
```

```
library tsm,optmum;

output file = tsm2a.out reset;

@<--- data --->@

load y[] = lynx.asc;
y = log(y);
x = y - meanc(y);
Nobs = rows(y);


@<--- log-likelihood of EXPAR model --->@

proc EXPAR(theta);
  local beta,delta,sigma,resid,i,logL;

  beta = theta[1:6]; delta = theta[7]; sigma = theta[8];
  resid = miss(zeros(Nobs,1),0);
  LogL = miss(zeros(Nobs,1),0);

  i = 3;
  do until i>Nobs;

    resid[i] = x[i]
       -(beta[1] +(beta[2]+beta[3]*x[i-1])*exp(-delta*(x[i-1]^2)))*x[i-1]
       +(beta[4] +(beta[5]+beta[6]*x[i-1])*exp(-delta*(x[i-2]^2)))*x[i-2];
    i=i+1;
  endo;

  LogL[3:Nobs] = -0.5*ln(2*pi) -0.5*ln(sigma^2)
                 -0.5*(resid[3:Nobs]^2)/(sigma^2);

  retp(Logl);
endp;


@<--- estimation --->@

_print = 1;
_tsm_optmum = 1;
_tsm_parnm  = ''beta1''|''beta2''|''beta3''|''beta4''|''beta5''|
              ''beta6''|''delta''|''sigma'';
__title = ''EXPAR model'';

sv = ones(8,1);

{theta,stderr,Mcov,Logl} = TD_ml(&EXPAR,sv);

output off;


Total observations:                                    114
Usable observations:                                   112
Number of parameters to be estimated:                    8
Degrees of freedom:                                    104
Value of the maximized log-likelihood function:    15.89488


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
beta1           1.989202        2.011331         0.988998        0.324960
beta2          -0.569987        2.058277        -0.276924        0.782388
beta3           0.448605        0.182175         2.462498        0.015440
beta4           0.151471        1.404292         0.107863        0.914312
beta5           0.742232        1.414515         0.524725        0.600891
beta6           0.532173        0.206033         2.582944        0.011186
delta           0.113101        0.282295         0.400646        0.689502
sigma           0.209956        0.014029        14.966371        0.000000

Covariance matrix: inverse of the negative Hessian.
```

3.2.3.2 Generalized method of moments

- ▶ An example with Ornstein-Uhlenbeck process.

```
new;
library tsm,optmum,option;
TSMset;

output file = tsm2b.out reset;
```

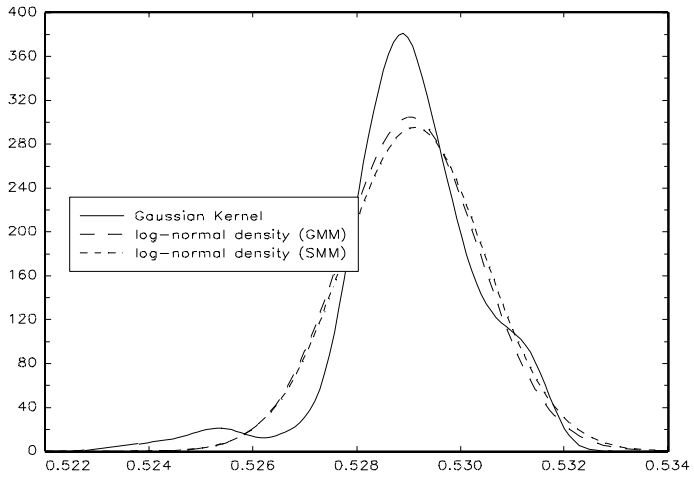
```
@<--- data generation --->@

x0 = 10; a = 0.8; b = 0.1; sigma = 0.06;
t0 = 0; TT = 100; Nobs = 1001;
sv = a|b|sigma;

{t,x} = simulate_OU(x0,a,b,sigma,t0,TT,Nobs,1);


@<--- moments definition --->@

h = 0.1;

proc Moments(theta);
  local a,b,sigma,k1,k2,epsilon,M;

  a = theta[1];
  b = theta[2];
  sigma = theta[3];

  k1 = exp(-a*h);
  k2 = sigma^2*(1-exp(-2*a*h))/(2*a);

  epsilon = x[2:Nobs]-k1.*x[1:Nobs-1]-b*(1-k1);

  M = epsilon~(epsilon^2-k2)~(epsilon.*x[1:Nobs-1]);

  retp(M);
endp;


@<--- GMM estimation --->@

_tsm_parnm = ''a''|''b''|''sigma'';
sv = a|b|sigma;
_ml_Jacobian_proc = 0;
__title = ''Ornstein-Ulhenbeck process'';

{theta,stderr,Mcov,Qmin} = gmm(&Moments,sv);

output off;


Total observations:                                  1000
Usable observations:                                 1000
Number of parameters to be estimated:                   3
Degrees of freedom:                                   997
Number of moment conditions:                            3
Value of the criterion function:                  0.00000


Parameters      estimates       std.err.      t-statistic         p-value
------------------------------------------------------------------------
a               0.795014        0.009004        88.293036        0.000000
b               0.090010        0.007787        11.559073        0.000000
sigma           0.060943        0.001430        42.619105        0.000000
```

3.2.3.3 Simulated method of moments

- Log-normal distribution.

```
new;
library tsm,optmum,pgraph;
TSMset;

#include 2LN.src;

output file = tsm2c.out reset;

load data[] = frfdem.asc;
s = packr(log(data));

_kernel[1:2] = 0.522|0.534;

/* Density estimation by Kernel */

{x,pdfKernel,cdfKernel,retcode} = Kernel(s);

/* Log-Normal */

proc H(theta);
  local mu,sigma,h1,h2;
```

```
  mu = theta[1];
  sigma = sqrt(theta[2]^2);

  /* first moment E[z] = exp(mu+0.5*sigma^2) */
  h1 = s - exp(mu+0.5*sigma^2);

  /* second moment var[z] = exp(2*mu+sigma^2)*(exp(sigma^2)-1) */
  h2 = h1.*h1 - exp(2*mu+sigma^2)*(exp(sigma^2)-1);

  retp(h1~h2);
endp;

_gmm_lags = 0;
_gmm_iter = 2;
__output = 1;
_print = 0;
_tsm_parnm = ''mu''|''sigma'';

sv = ln(meanc(s))|.1;
{theta,stderr,Mcov,Qmin} = gmm(&H,sv);
mu = theta[1];
sigma = sqrt(theta[2]^2);
pdfGMM1 = pdfLN(x,mu,sigma);

proc SimulatedMoments(theta);
  local mu,sigma;
  local N,u;
  local M1,uc,M2;
  local h1,h2;

  mu = theta[1];
  sigma = sqrt(theta[2]^2);

  N = 1000;  /* number of simulations */

  rndseed 123456; /* always use the same random numbers */

  u = rndLN(mu,sigma,N,1);

  M1 = meanc(u);
  uc = u - M1;
  M2 = meanc(uc^2);

  /* first moment */
  h1 = s - M1;

  /* second moment */
  h2 = h1.*h1 - M2;

  retp(h1~h2);
endp;

{theta,stderr,Mcov,Qmin} = gmm(&SimulatedMoments,sv);
mu = theta[1];
sigma = sqrt(theta[2]^2);
pdfGMM2 = pdfLN(x,mu,sigma);

graphset;
  _pdate = ''''; _pnum = 2; _pltype = 6|1|3;
  _plegstr = ''Gaussian Kernel\000log-normal density (GMM)\000log-normal density (SMM)'';
  _plegctl = {2 5 1 3};
  xtics(0.522,0.534,0.002,0);
  graphprt(''-c=1 -cf=tsm2c.eps'');
  xy(x,pdfKernel~pdfGMM1~pdfGMM2);
```



- Parameters of the mixture of two log-normal variables (Not yet done).

3.2.3.4 Whittle method

We consider the model z_t, defined by

$$\left\{\begin{array}{rcl} z_t &=& x_t + y_t \\ x_t &=& \phi_1 x_{t-1} + u_t \\ y_t &=& v_t - \theta_1 v_{t-1} \end{array}\right.$$

with $u_t \sim N\left(0, \sigma_u^2\right)$ and $v_t \sim N\left(0, \sigma_v^2\right)$. The corresponding spectral generating function is

$$\begin{array}{rcl} g\left(\lambda_j\right) &=& \sigma_u^2 \frac{1}{\left|1-\phi_1 e^{i\lambda_j}\right|^2} + \sigma_v^2 \left|1-\theta_1 e^{i\lambda_j}\right|^2 \\ &=& \sigma_u^2 \frac{1}{\left(1-2\phi_1 \cos\lambda_j + \phi_1^2\right)} + \sigma_v^2 \left(1 - 2\theta_1 \cos\lambda_j + \theta_1^2\right) \end{array}$$

The vector of parameters is set to $\left[\begin{array}{cccc} \phi_1 & \sigma_u & \theta_1 & \sigma_v \end{array}\right]^\top$.

```
new;
library tsm,optmum;
TSMset;

output file = tsm2d.out reset;


@<--- data generation --->@

rndseed 123456;

phi1 = 0.5; theta1 = 0.7; sigma_u = 0.25; sigma_v = 1;
Nobs = 1000;

u = rndn(Nobs,1)*sigma_u;
x = recserar(u,0,phi1);

v = rndn(Nobs,1)*sigma_v;
y = v - theta1*(0|trimr(v,0,1));
z = x + y;


@<--- Whittle estimation --->@

sv = phi1|sigma_u|theta1|sigma_v;
_tsm_parnm = ''phi1''|''sigma_u''|''theta1''|''sigma_v'';
_fourier = 0;
_sgf_Jacobian_Proc = &sgf_Jacobian;

{theta,stderr,Mcov,Logl} = FD_ml(z,&sgf,sv);

output off;
```

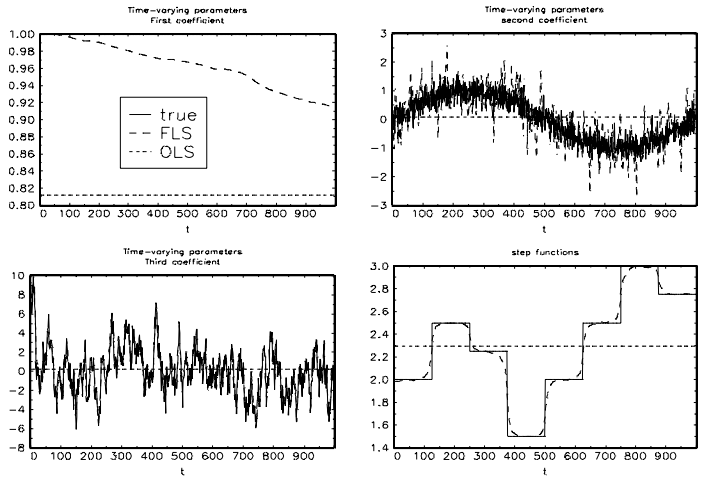
```
proc sgf(coeff,lambda);
  local phi1,theta1,sigma_u,sigma_v;
  local w,g;
  phi1 = coeff[1];
  sigma_u = coeff[2];
  theta1 = coeff[3];
  sigma_v = coeff[4];
  w = cos(lambda);
  g = (sigma_u^2)./(1-2*phi1*w+phi1^2) +
      (sigma_v^2).*(1-2*theta1*w+theta1^2);
  retp(g);
endp;

proc sgf_Jacobian(coeff,lambda);
  local phi1,theta1,sigma_u,sigma_v;
  local w,g;
  local w1,w2,J1,J2,J3,J4,J;

  phi1 = coeff[1];
  sigma_u = coeff[2];
  theta1 = coeff[3];
  sigma_v = coeff[4];
  w = cos(lambda);

  w1 = 1-2*phi1*w+phi1^2;
  w2 = 1-2*theta1*w+theta1^2;

  J1 = 2*(w-phi1).*(sigma_u^2)./(w1^2);
  J2 = 2*sigma_u./w1;
  J3 = 2*(theta1-w).*(sigma_v^2);
  J4 = 2*sigma_v.*w2;
  J = J1~J2~J3~J4;

  retp(J);
endp;
```

```
Total observations:                                  1000
Usable observations:                                 1000
Number of parameters to be estimated:                   4
Degrees of freedom:                                   996
Value of the maximized log-likelihood function:  -1568.95349


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
phi1            0.380003        0.171811         2.211752        0.027210
sigma_u         0.348686        0.084475         4.127688        0.000040
theta1          0.999331        0.814324         1.227191        0.220041
sigma_v         0.856431        0.349596         2.449773        0.014466

Covariance matrix: inverse of the negative Hessian.
```

3.2.3.5 Generalized flexible least squares

```
new;
library tsm,optmum,pgraph;


/*
================== First example: Time-varying parameters ==================
*/

@<--- data generation --->@

rndseed 456;

Nobs = 1000;

s = seqa(1,1,Nobs);

x = floor(rndu(Nobs,3)*100);

b1 = ones(Nobs,1);                                  /* constant parameter */
b2 = sin(seqa(0,2*pi/Nobs,Nobs)) + rndn(Nobs,1)*0.25;   /* sine parameter     */
b3 = recserar(rndn(Nobs,1),10,0.9);                 /* AR parameter       */
B = b1~b2~b3;

y = sumc(x'.*B') + rndn(Nobs,1);


/* Flexible Least Squares with mu = 1
** (and imposing constancy for the first parameter)
```

```
*/

mu = 100000|1|1;
{Bfls,u,r2_M,r2_D} = FLS(y,x,mu);          /* FLS estimation */

Bols = (invpd(x'x)*x'y )' .*ones(Nobs,1);  /* OLS estimation */

xy1 = b1~Bfls[.,1]~Bols[.,1];
xy2 = b2~Bfls[.,2]~Bols[.,2];
xy3 = b3~Bfls[.,3]~Bols[.,3];


/*
================== Second example: step function ==================
*/

StepValues = 2|2.5|2.25|1.5|2|2.5|3|2.75;

x1 = floor(rndu(Nobs,1)*100); x2 = floor(rndu(Nobs,2)*100);
b1 = {}; b2 = 2|3;
i = 1;
do until i > 8;
  b1 = b1 | StepValues[i]*ones(125,1);
  i = i + 1;
endo;

y = 4 + x1.*b1 + x2*b2 + rndn(Nobs,1);
x = ones(Nobs,1)~x1~x2;

{Bfls,u,r2M,r2D} = FLS(y,x,100000);          /* FLS estimation */

Bols = (invpd(x'x)*x'y )' .*ones(Nobs,1);  /* OLS estimation */

xy4 = b1~Bfls[.,2]~Bols[.,2];

graphset;
  begwind;
  window(2,2,0);

    _pdate = '''';  _pnum = 2; _pnumht = 0.20; _paxht = 0.20;
    _pltype = 6|1|3;
    _plegstr = ''true\0FLS\0OLS''; _plegctl = {2 8 3 2};
    xlabel(''t'');

  setwind(1);
    title(''Time-varying parameters\LFirst coefficient'');
    xy(s,xy1);

    _plegctl = 0;

  setwind(2);
    title(''Time-varying parameters\Lsecond coefficient'');
    xy(s,xy2);

  setwind(3);
    title(''Time-varying parameters\LThird coefficient'');
    xy(s,xy3);

  setwind(4);
    title(''step functions'');
    xy(s,xy4);

  graphprt(''-c=1 -cf=tsm2e.eps'');

  endwind;
```



3.2.4 State-space modelling

3.2.4.1 Structural models

▶ Local linerar trend model. The estimation is performed in the **frequency** domain.

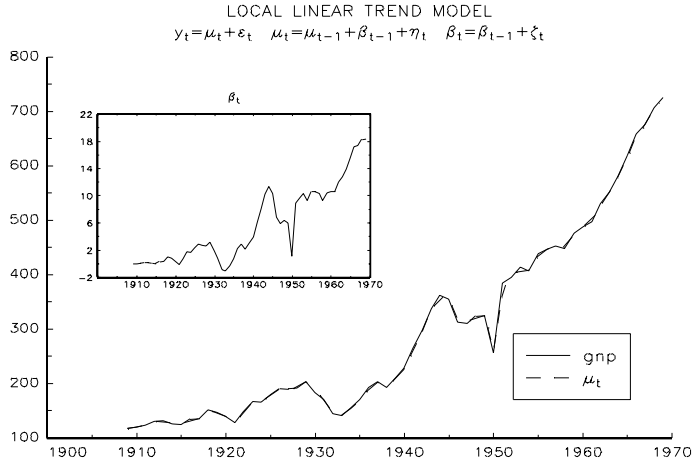
```
/*
**  HARVEY [1990], Forecasting, Structural Time Series and
**  the Kalman Filter, Cambridge University Press, pages 90-93
**
**  Kalman filtering and the Local Linear Trend Model
*/

new;
library tsm,optmum,pgraph;
TSMset;

output file = tsm3a.out reset;


@<--- data --->@

load gnp[] = gnp.asc;
Nobs = rows(gnp);


@<--- frequency domain estimation --->@

_tsm_optmum = 0;                              /* Method of scoring */
{beta,stderr,Mcov,Logl} =  sm_LLT(gnp,1|1|1);


@<--- Associated state-space model --->@

theta = beta^2;  /* sigma^2 */

Z = 1~0; d = 0; H = theta[3];
T = {1 1,0 1}; c = 0|0; R = {1 0,0 1}; Q = (theta[1]~0)|(0~theta[2]);

call SSM_build(Z,d,H,T,c,R,Q,0);

a0 = gnp[1]|0; P0 = zeros(2,2);               /* Kalman filter initialization */
call KFiltering(gnp,a0,P0);                   /* Running the Kalman filter    */

a = KF_matrix(3);                             /* a[t]                         */

t_ = seqa(1909,1,61);

output off;

graphset;
  fonts(''simplex simgrma'');
  _pdate = ''''; _pnum = 2;
  begwind;
  makewind(9,6.855,0,0,0);
  makewind(4,3,1,2.5,1);
  setwind(1);
```

```
    _pframe = 0;
    title(''LOCAL LINEAR TREND MODEL''\
          ''\Ly]t[=\202m\201]t[+\202e\201]t[  ''\
          ''\202m\201]t[=\202m\201]t-1[+\202b\201]t-1[+\202c\201]t[  ''\
          ''\202b\201]t[=\202b\201]t-1[+\202z\201]t['');
    _plegctl = {2 6 6.5 1};
    _plegstr = ''gnp\000\202m\201]t['';
    xy(t_,gnp~a[.,1]);
  nextwind;
    _pnum = 2; _pframe = 1|1; _paxes = 1;
    _ptitlht = 0.25; _pnumht = 0.20;
    title(''\202b\201]t['');
    _plegstr = ''''; _plegctl = 0;
    xy(t_,a[.,2]);

  graphprt(''-c=1 -cf=tsm3a.eps'');

  endwind;
```

```
Total observations:                                    61
Usable observations:                                   59
Number of parameters to be estimated:                   3
Degrees of freedom:                                    56
Value of the maximized log-likelihood function:  -273.00725
```

Parameters	estimates	std.err.	t-statistic	p-value
sig_eta	21.732567	3.864839	5.623149	0.000001
sig_zeta	1.601397	0.826497	1.937572	0.057725
sig_epsi	7.112369	5.288299	1.344926	0.184071

```
Covariance matrix: inverse of the negative Hessian.
```

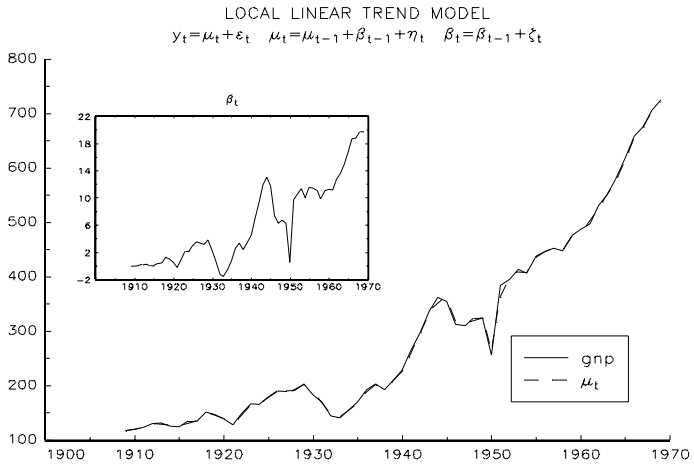


▶ Local linerar trend model. The estimation is performed in the **time** domain.

```
/*
**  HARVEY [1990], Forecasting, Structural Time Series and
**  the Kalman Filter, Cambridge University Press, pages 90-93
**
**  Kalman filtering and the Local Linear Trend Model
*/

new;
library tsm,optmum,pgraph;
TSMset;

output file = tsm3b.out reset;


@<--- data --->@

load gnp[] = gnp.asc;
Nobs = rows(gnp);


@<--- time domain estimation --->@
```

```
Z = 1~0; d = 0;
T = {1 1,0 1}; c = 0|0; R = {1 0,0 1};
a0 = gnp[1]|0; P0 = zeros(2,2);

proc ml(theta);
  local H,Q,LogL;
  theta = theta^2;  /* sigma^2 */
  H = theta[3];
  Q = (theta[1]~0)|(0~theta[2]);
  call SSM_build(Z,d,H,T,c,R,Q,0);
  call KFiltering(gnp,a0,P0);
  LogL = KF_ml;
  retp(LogL);
endp;


_tsm_optmum = 0;    /* BHHH algorithm  */
_tsm_Mcov = 3;      /* Heteroskedasticity covariance */

{beta,stderr,Mcov,Logl} = TD_ml(&ml,5|5|5);


@<--- Associated state-space model --->@

theta = beta^2;  /* sigma^2 */

H = theta[3];
Q = (theta[1]~0)|(0~theta[2]);

call SSM_build(Z,d,H,T,c,R,Q,0);

call KFiltering(gnp,a0,P0);                    /* Running the Kalman filter    */

a = KF_matrix(3);                              /* a[t]                         */

t_ = seqa(1909,1,61);

output off;

graphset;
  fonts(''simplex simgrma'');
  _pdate = ''''; _pnum = 2;
  begwind;
  makewind(9,6.855,0,0,0);
  makewind(4,3,1,2.5,1);
  setwind(1);
    _pframe = 0;
    title(''LOCAL LINEAR TREND MODEL''\
          ''\Ly]t[=\202m\201]t[+\202e\201]t[  ''\
          ''\202m\201]t[=\202m\201]t-1[+\202b\201]t-1[+\202c\201]t[  ''\
          ''\202b\201]t[=\202b\201]t-1[+\202z\201]t['');
    _plegctl = {2 6 6.5 1};
    _plegstr = ''gnp\000\202m\201]t['';
    xy(t_,gnp~a[.,1]);
  nextwind;
    _pnum = 2; _pframe = 1|1; _paxes = 1;
    _ptitlht = 0.25; _pnumht = 0.20;
    title(''\202b\201]t['');
    _plegstr = ''''; _plegctl = 0;
    xy(t_,a[.,2]);

  graphprt(''-c=1 -cf=tsm3b.eps'');

  endwind;
```

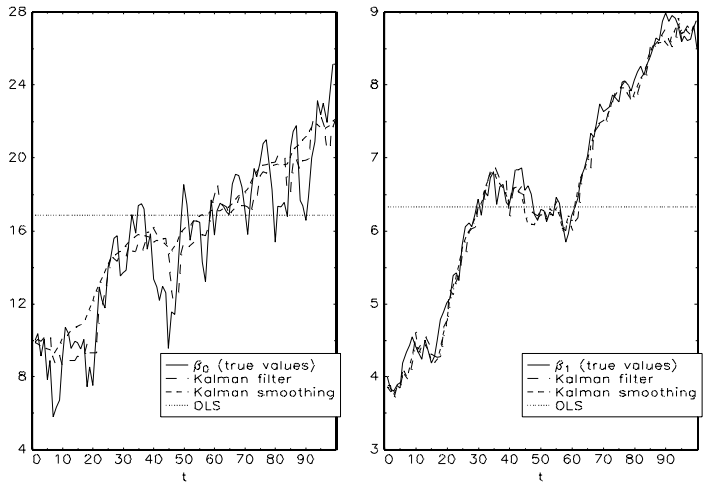
```
Total observations:                                      61
Usable observations:                                     61
Number of parameters to be estimated:                     3
Degrees of freedom:                                      58
Value of the maximized log-likelihood function:      -281.05421


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01            20.935307        3.830210         5.465837        0.000001
P02             1.865098        0.598496         3.116310        0.002847
P03             7.544804       10.783561         0.699658        0.486938

Covariance matrix: White Heteroskedastic matrix.
```



3.2.4.2 Time-varying parameters

```
/*
**  Time-Varying Model
**
**  y(t) = b0*x0(t) + b1*x1(t) + u(t)
**  b0(t) = b0(t-1) + v0(t)
**  b1(t) = b1(t-1) + v1(t)
**
**  Maximum Likelihood Estimation
*/

new;
library tsm,optmum,pgraph;
TSMset;

declare external sigma;


@<--- Time-Varying Model simulation --->@

rndseed 123;

s = seqa(1,1,100);

b0 = recserar(rndn(100,1)*1.5,10,1);
b1 = recserar(rndn(100,1)*0.2,4,1);

x0 = ones(100,1);
x1 = rndu(100,1)*25;

u = rndn(100,1)*2;
X = x0~x1;
Y = X0.*b0 + X1.*b1 + u;


@<--- Associated state-space model --->@

proc Z(i); local w; w = X[i,.]; retp(w); endp;

proc d(i); local w; w = 0; retp(w); endp;

proc T(i); local w; w = eye(2); retp(w); endp;

proc c(i); local w; w = 0|0; retp(w); endp;

proc R(i); local w; w = eye(2); retp(w); endp;

proc H(i); local w; w = sigma[1]^2; retp(w); endp;

proc Q(i); local w; w = eye(2).*sigma[2:3]'; w = w^2; retp(w); endp;


@<--- log-likelihood function --->@

proc ml(theta);
  local a0,P0,LogL;
  sigma = theta[1:3];
```

```
  call SSM_build(&Z,&d,&H,&T,&c,&R,&Q,1);
  a0 = b0[1]|b1[1];
  P0 = zeros(2,2);
  call KFiltering(Y,a0,P0);
  LogL = KF_ml;
  retp(LogL);
endp;


@<--- ML estimation --->@

_tsm_optmum = 0;                          /* BHHH algorithm */
_tsm_parnm = ''sig_u''|''sigma0''|''sigma1'';

{theta,stderr,Mcov,LogL} = TD_ml(&ml,2|1.5|0.2);

beta = invpd(X'X)*X'Y;                    /* OLS estimation */

output off;


@<--- Kalman filtering --->@

sigma = theta[1:3];
a0 = b0[1]|b1[1];
P0 = zeros(2,2);

call SSM_build(&Z,&d,&H,&T,&c,&R,&Q,1);
call KFiltering(Y,a0,P0);

yc = KF_matrix(1);     /* y[t|t-1]      */
v = KF_matrix(2);      /* v[t]          */
a = KF_matrix(3);      /* a[t]          */
P = KF_matrix(4);      /* P[t]          */
ac = KF_matrix(5);     /* a[t|t-1]      */
Pc = KF_matrix(6);     /* P[t|t-1]      */
F = KF_matrix(7);      /* F[t|t-1]      */
invF = KF_matrix(8);   /* F[t|t-1]^(-1) */


@<--- Kalman smoothing --->@

{as,Ps} = Ksmoothing;


graphset;
   begwind;
   window(1,2,1);

   _pdate = ''''; _pnum = 2; _paxht = 0.25; _pnumht = 0.25;
   fonts(''simplex simgrma'');
   xlabel(''t'');
   _pltype = 6|1|3|4;
   _plegctl = {2 5 4 1};

   setwind(1);

     _plegstr = ''\202b\201]0[ (true values)''\
                ''\000Kalman filter''\
                ''\000Kalman smoothing''\
                ''\000OLS'';
     xy(s,b0~a[.,1]~as[.,1]~(beta[1]*ones(100,1)));

   setwind(2);

     _plegstr = ''\202b\201]1[ (true values)''\
                ''\000Kalman filter''\
                ''\000Kalman smoothing''\
                ''\000OLS'';
     xy(s,b1~a[.,2]~as[.,2]~(beta[2]*ones(100,1)));

   graphprt(''-c=1 -cf=tsm3c.eps'');

   endwind;
```



3.2.4.3 Multivariate model

- Not yet done.

3.2.4.4 Exact maximum likelihood

- Univariate ARMA example.

```
/*
**  Exact Maximum Likelihood Estimation of an ARMA(1,1) model
**  with the Kalman Filter
*/

new;
library tsm,optmum,pgraph;
TSMset;

output file = tsm3e.out reset;

/* Generate an ARMA(1,1) process */

rndseed 123456;

Nobs = 100;
et = rndn(Nobs+1,1)*1.5;
yt = recserar(et[2:Nobs+1]+0.6*et[1:Nobs],0,0.8);

/* Build a procedure for the likelihood function */

proc ml(coeff);
  local beta,sigma,Pchol,Z,d,H,T,c,R,Q;
  local a0,P0,Logl;

  beta = coeff[1:2];
  SIGMA = coeff[3]^2;

  {Z,d,H,T,c,R,Q} = arma_to_SSM(beta,1,1,SIGMA);
  call SSM_build(Z,d,H,T,c,R,Q,0);
  {a0,P0} = SSM_ic;
  call KFiltering(yt,a0,P0);
  Logl = KF_ml;

  retp(Logl);
endp;

/* exact MLE */

_print = 1;
_tsm_optmum = 0;
_tsm_gtol = 0.001;
_tsm_Mcov = 1;

{theta,stderr,Mcov,Logl} = TD_ml(&ml,0.8|-0.6|1.5);

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

```
/* SSM */

{Z,d,H,T,c,R,Q} = arma_to_SSM(beta,1,1,SIGMA);
call SSM_build(Z,d,H,T,c,R,Q,0);
{a0,P0} = SSM_ic;
call KFiltering(yt,a0,P0);

/* Forecasting */

np = 10;

{Forecasts,mse,aF,PF} = KForecasting(np);
s = seqa(Nobs+1,1,np);
Fstderr = sqrt(mse);
LCL = Forecasts - 1.96*Fstderr;
UCL = Forecasts + 1.96*Fstderr;

omat = LCL~Forecasts~UCL~Fstderr;

print;
print ''   Period        LCL         Forecasts        UCL''\
      ''        Forecast Std. Err.'';
print chrs(45*ones(75,1));
let fmt[5,3] = ''*.*lf'' 8 0
               ''*.*lf'' 15 6
               ''*.*lf'' 15 6
               ''*.*lf'' 15 6
               ''*.*lf'' 15 6;
call printfm(s~omat,1~1~1~1~1,fmt);

output off;

/*
**
**  If you have the ARIMA library, you could compare with Ansley method:
**
**      {b,l,e,cv,aic,sbc} = arima(0,yt,1,0,1,0);
**      f = forecast(b,yt,1,0,1,0,e,10);
**
**  The results are:
**
**  Model:  ARIMA(1,0,1)
**
**
**  Final Results:
**
**  Iterations Until Convergence:   4
**
**  Log Likelihood:   -183.223266         Number of Residuals: 100
**  AIC           :    370.446532         Error Variance     : 2.294675936
**  SBC           :    375.656872         Standard Error     : 1.514818780
**
**  DF: 98      Adj. SSE: 228.550784245        SSE: 224.878241685
**
**               Coefficients     Std. Errors      T-Ratio    Approx. Prob.
**  AR1            0.71161142      0.07773737      9.15405          0.00000
**  MA1           -0.50727789      0.09713635     -5.22233          0.00000
**
**
**  Total Computation Time: 0.11 (seconds)
**
**  AR Root:   1.40526
**
**  MA Root:  -1.97131
**
**
**  Forecasts for ARIMA(1,0,1) Model.   95% Confidence Interval Computed.
**
**   Period        LCL        Forecasts        UCL     Forecast Std. Err.
**     101     -2.070068      0.898923      3.867913      1.514819
**     102     -4.041248      0.639684      5.320616      2.388275
**     103     -4.887352      0.455206      5.797765      2.725845
**     104     -5.324186      0.323930      5.972046      2.881745
**     105     -5.566196      0.230512      6.027220      2.957559
**     106     -5.706484      0.164035      6.034554      2.995218
**     107     -5.790815      0.116729      6.024274      3.014109
**     108     -5.843140      0.083066      6.009272      3.023630
**     109     -5.876523      0.059111      5.994744      3.028440
**     110     -5.898338      0.042064      5.982465      3.030873
*/


Total observations:                                    100
Usable observations:                                   100
```

```
Number of parameters to be estimated:                          3
Degrees of freedom:                                           97
Value of the maximized log-likelihood function:       -183.22327


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01              0.711615       0.076031         9.359545        0.000000
P02             -0.507273       0.093996        -5.396750        0.000000
P03              1.499598       0.106059        14.139286        0.000000

Covariance matrix: inverse of the negative Hessian.

   Period        LCL         Forecasts        UCL        Forecast Std. Err.
-------------------------------------------------------------------------
     101      -2.040286       0.898927       3.838139       1.499598
     102      -3.994291       0.639690       5.273670       2.364276
     103      -4.833764       0.455213       5.744189       2.698457
     104      -5.267538       0.323936       5.915411       2.852793
     105      -5.508062       0.230518       5.969097       2.927847
     106      -5.647613       0.164040       5.975693       2.965129
     107      -5.731575       0.116733       5.965042       2.983831
     108      -5.783715       0.083069       5.949853       2.993257
     109      -5.817004       0.059113       5.935230       2.998019
     110      -5.838772       0.042066       5.922904       3.000428
```

- Multivariate ARMA example.

```
/*
**  Estimation of a Vector ARMA(1,1) process
**  Exact Maximum Likelihood
**
*/

new;
library tsm,optmum;

output file = tsm3f.out reset;

@<--- Data --->@

load reinsel[100,2] = reinsel.asc;

invest = reinsel[.,1];
invent = reinsel[.,2];
di = invest - lag1(invest);
y = di~invent;


@<--- ML function --->@

proc ml(coeff);
  local beta,sigma,Pchol,Z,d,H,T,c,R,Q;
  local a0,P0,Logl;

  beta = coeff[1:8];
  Pchol = (coeff[9]~0)|(coeff[10]~coeff[11]);
  SIGMA = Pchol*Pchol';

  {Z,d,H,T,c,R,Q} = arma_to_SSM(beta,1,1,SIGMA);
  call SSM_build(Z,d,H,T,c,R,Q,0);

  {a0,P0} = SSM_ic;         /* <=====  THIS LINE IS THE MOST IMPORTANT
                                       FOR EXACT MLE
                            */

  if ismiss(a0);
    cls;
    print ''Not implemented for non-stationnary ARMA models.'';
    end;
  endif;

  call KFiltering(y,a0,P0);
  Logl = KF_ml;

  retp(Logl);
endp;

_print = 1;
_tsm_optmum = 0;
_tsm_gtol = 0.001;

load arma1, arma2;
```

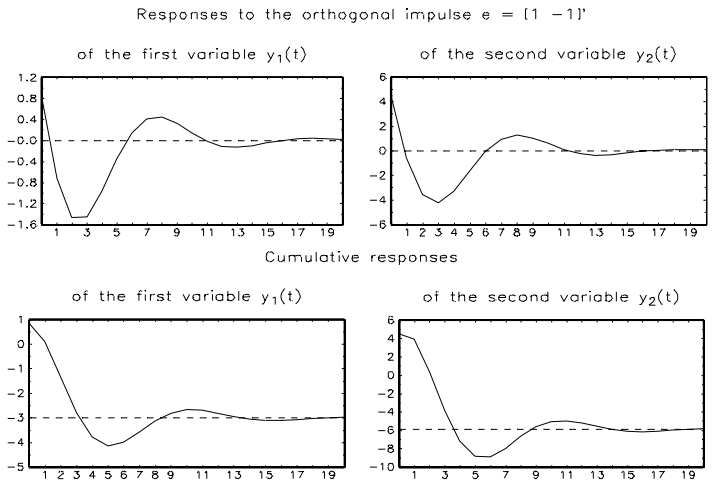
```
sv = arma1|vech_(chol(arma2));    /* Starting values = CMLE parameters */

{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;


Total observations:                                          100
Usable observations:                                          99
Number of parameters to be estimated:                         11
Degrees of freedom:                                           88
Value of the maximized log-likelihood function:       -506.82688


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01              0.358733        0.176666         2.030576        0.045317
P02              0.648398        0.230546         2.812441        0.006062
P03             -0.023065        0.044917        -0.513508        0.608883
P04              0.841344        0.047614        17.670001        0.000000
P05             -0.208480        0.183373        -1.136917        0.258659
P06              0.237817        0.317614         0.748760        0.455999
P07             -0.181840        0.069227        -2.626708        0.010169
P08              0.239144        0.117736         2.031191        0.045253
P09              2.377602        0.169044        14.065018        0.000000
P10              1.000585        0.418358         2.391694        0.018900
P11              4.077719        0.290700        14.027226        0.000000

Covariance matrix: inverse of the negative Hessian.
```

3.2.4.5 Impulse functions

- An example with a SSM model.

```
new;
library tsm,optmum,pgraph;
TSMset;

Z = {1 1,4 2,2 -3}; d = 0|0|0; H = {5 1 0,1 4 0,0 0 8};
T = {0.5 0.45,-0.5 0.8};
c = {0,0}; R = eye(2); Q = {2 0.5,0.5 1};

call SSM_build(Z,d,H,T,c,R,Q,0);

Nr = 20;
s = seqa(0,1,Nr+1);

e = 1|-1;
{Delta,Ksi,zeta} = SSM_orthogonal(e,Nr);

graphset;
  begwind;
  makewind(9,0.5,0,6.855-0.5,0);
  makewind(9/2,5.855/2,0,6.855/2,0);
  makewind(9/2,5.855/2,4.5,6.855/2,0);
  makewind(9,0.5,0,5.855/2,0);
  makewind(9/2,5.855/2,0,0,0);
  makewind(9/2,5.855/2,4.5,0,0);

  setwind(1);
    _paxes = 0; _pnum = 0; _ptitlht = 2;
    title(''Responses to the orthogonal impulse e = @[1 -1@]'''');
    draw;

  setwind(2);
    graphset;
    _ptitlht = 0.35; _paxht = 0.25; _pnumht = 0.25; _pnum = 2;
    _pline = 1~1~0~0~Nr~0~1~7~0;
    title(''of the first variable y]1[(t)'');
    xtics(0,Nr,1,0);
    xy(s,Delta[.,1]);

  setwind(3);
    title(''of the second variable y]2[(t)'');
    xtics(0,Nr,1,0);
    xy(s,Delta[.,2]);

  setwind(4);
    graphset;
    _paxes = 0; _pnum = 0; _ptitlht = 2;
    title(''Cumulative responses'');
    draw;
```

```
  setwind(5);
    graphset;
    _ptitlht = 0.35; _paxht = 0.25; _pnumht = 0.25;  _pnum = 2;
    title(''of the first variable y]1[(t)'');
    xtics(0,Nr,1,0);
    _pline = 1~1~0~zeta[1]~Nr~zeta[1]~1~7~0;
    xy(s,Ksi[.,1]);

  setwind(6);
    title(''of the second variable y]2[(t)'');
    xtics(0,Nr,1,0);
    _pline = 1~1~0~zeta[2]~Nr~zeta[2]~1~7~0;
    xy(s,Ksi[.,2]);

    graphprt(''-c=1 -cf=tsm3g.eps'');

  endwind;
```



- An example with a VAR model.

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  Forecast Error Variance Decomposition
**  See LUTKEPOHL, section 3.7.3
**
*/

new;
library tsm,optmum,pgraph;

output file = tsm3h.out reset;

load y[92,3] = lutkepoh.asc;

y = ln(y[1:76,.]);

INVEST = y[.,1];
dinv = INVEST-lag1(INVEST);
INCOME = y[.,2];
dinc = INCOME-lag1(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lag1(CONSUM);
data = dinv~dinc~dcons;

_print = 0;
{theta,stderr,Mcov,LOGL} = varx_LS(data,1,2);  /* Constant and 2 lags */

beta = theta[1:18];                            /* Estimates of the AR part */
SIGMA = _varx_SIGMA;

{Z,d,H,T,c,R,Q} = arma_to_SSM(beta,2,0,SIGMA);
call SSM_build(Z,d,H,T,c,R,Q,0);

Omega = SSM_fevd(8);

i = seqa(1,1,8);
```

```
mask = 1~1~1~1;
let fmt[4,3]=  ''*.*lf'' 13 0 ''*.*lf'' 18 4 ''*.*lf'' 18 4 ''*.*lf'' 18 4;

print;
print ''FORECAST ERROR VARIANCE DECOMPOSITION in INVESTMENT (percent)'';
print;
print chrs(45*ones(79,1));
print ''                        Investment           Income      Consumption'';
print ''         periods      innovations      innovations      innovations'';
print chrs(45*ones(79,1));
i = 1;
do until i > 8;
  rsp = xpnd2(Omega,i);
  rsp = rsp[1,.];
  call printfm(i~(rsp*100),mask,fmt); print;
  i = i +1;
endo;

print; print;

print;
print ''FORECAST ERROR VARIANCE DECOMPOSITION in INCOME (percent)'';
print;
print chrs(45*ones(79,1));
print ''                        Investment           Income      Consumption'';
print ''         periods      innovations      innovations      innovations'';
print chrs(45*ones(79,1));
i = 1;
do until i > 8;
  rsp = xpnd2(omega,i);
  rsp = rsp[2,.];
  call printfm(i~(rsp*100),mask,fmt); print;
  i = i +1;
endo;

print; print;

print;
print ''FORECAST ERROR VARIANCE DECOMPOSITION in CONSUMPTION (percent)'';
print;
print chrs(45*ones(79,1));
print ''                        Investment           Income      Consumption'';
print ''         periods      innovations      innovations      innovations'';
print chrs(45*ones(79,1));
i = 1;
do until i > 8;
  rsp = xpnd2(Omega,i);
  rsp = rsp[3,.];
  call printfm(i~(rsp*100),mask,fmt); print;
  i = i +1;
endo;

output off;
```

FORECAST ERROR VARIANCE DECOMPOSITION in INVESTMENT (percent)

periods	Investment innovations	Income innovations	Consumption innovations
1	100.0000	0.0000	0.0000
2	95.9960	1.7511	2.2529
3	94.5649	2.8021	2.6330
4	94.0792	2.9361	2.9847
5	93.8464	3.0181	3.1356
6	93.8308	3.0246	3.1446
7	93.7784	3.0737	3.1479
8	93.7751	3.0739	3.1510

FORECAST ERROR VARIANCE DECOMPOSITION in INCOME (percent)

periods	Investment innovations	Income innovations	Consumption innovations
1	1.7536	98.2464	0.0000
2	6.0245	90.7470	3.2285
3	6.9592	89.5762	3.4645
4	6.8313	89.2321	3.9366
5	6.8501	89.2120	3.9379
6	6.9243	89.1408	3.9348

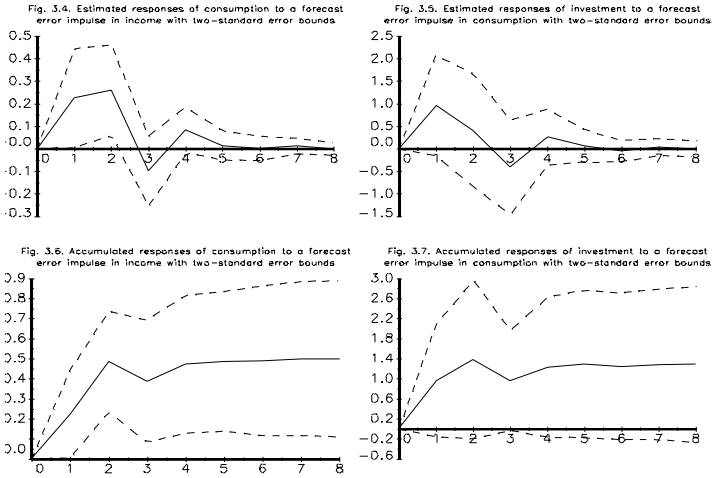
```
           7           6.9222           89.1158           3.9620
           8           6.9228           89.1149           3.9623


FORECAST ERROR VARIANCE DECOMPOSITION in CONSUMPTION (percent)

------------------------------------------------------------------------------
                      Investment           Income      Consumption
        periods      innovations      innovations      innovations
------------------------------------------------------------------------------
           1           7.9950           27.2921           64.7129
           2           7.7248           27.3848           64.8904
           3          12.9729           33.3641           53.6630
           4          12.8703           33.4988           53.6309
           5          12.8588           33.9244           53.2168
           6          12.8522           33.9630           53.1848
           7          12.8702           33.9562           53.1736
           8          12.8704           33.9682           53.1614
```

3.2.4.6 Bootstrap methods

▶ Computes the standard errors of the impulse responses by SSM bootstrap techniques.

```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  This program reproduces the figures 3.4-3.7 of LUTKEPOHL [1991].
*/

new;
library tsm,optmum,pgraph;
TSMset;

load y[92,3] = lutkepoh.asc;

y = ln(y[1:76,.]);

INVEST = y[.,1];
dinv = INVEST-lag1(INVEST);
INCOME = y[.,2];
dinc = INCOME-lag1(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lag1(CONSUM);
{data,retcode} = Missing(dinv~dinc~dcons,0);
Nobs = rows(data);

/* First, estimate the coefficients */

_print = 0;
{theta,stderr,Mcov,LOGL} = varx_LS(data,1,2);  /* Constant and 2 lags */

/* Take the AR part of the VARX model */

beta = theta[1:18];
const = theta[19:21];  /* constant estimates */
SIGMA = _varx_SIGMA;

Nr = 8;        /* Number of responses */
Ns = 100;       /* Number of bootstrap */

rep1 = zeros(1+Nr,1);  /* Responses of Consumption to an
                          impulse in Income */
rep2 = zeros(1+Nr,1);  /* Responses of Investment to an
                          impulse in Consumption */
rep3 = zeros(1+Nr,1);  /* Accumulated responses of Consumption to an
                          impulse in Income */
rep4 = zeros(1+Nr,1);  /* Accumulated responses of Investment to an
                          impulse in Consumption */

/* Compute the responses */

call arma_impulse(beta,2,0,Nr);

j = 0;
do until j > Nr;
  z = varget(''IMPULSE''$+ftos(j,''%lf'',1,0));
  rep1[1+j,1] = z[3,2];
  rep2[1+j,1] = z[1,3];
  z = varget(''_IMPULSE''$+ftos(j,''%lf'',1,0));
  rep3[1+j,1] = z[3,2];
  rep4[1+j,1] = z[1,3];
```

```
  j = j + 1;
endo;

/* Build the corresponding state space model */

{Z,d,H,T,c,R,Q} = arma_to_SSM(beta,2,0,SIGMA);
c[1:3] = const;   /* Add the constant */
call SSM_build(Z,d,H,T,c,R,Q,0);
a0 = data[1,.]'|data[2,.]'|zeros(3,1);
P0 = zeros(9,9);
call KFiltering(data[3:Nobs,.],a0,P0);

/* Compute the standard errors with the Bootstrap Method */

rep1s = zeros(1+Nr,Ns);
rep2s = zeros(1+Nr,Ns);
rep3s = zeros(1+Nr,Ns);
rep4s = zeros(1+Nr,Ns);

i = 1;
do until i > Ns;

  {ys,as} = bootstrap_SSM(a0);         /* SSM bootstrap */
  datas = data[1,.]|data[2,.]|ys;      /* Add the presample values */
  {thetas,stderr,Mcov,LOGL} = varx_LS(datas,1,2); /* VAR estimation */
  betas = thetas[1:18];
  call arma_impulse(betas,2,0,Nr);

  j = 0;
  do until j > Nr;
    z =  varget(''IMPULSE''$+ftos(j,''%lf'',1,0));
    rep1s[1+j,i] = z[3,2];
    rep2s[1+j,i] = z[1,3];
    z =  varget(''_IMPULSE''$+ftos(j,''%lf'',1,0));
    rep3s[1+j,i] = z[3,2];
    rep4s[1+j,i] = z[1,3];
    j = j + 1;
  endo;

  i = i + 1;
endo;

stderr1 = stdc(rep1s');
stderr2 = stdc(rep2s');
stderr3 = stdc(rep3s');
stderr4 = stdc(rep4s');

period = seqa(0,1,Nr+1);


graphset;
  begwind;
  window(2,2,0);

  _pnum = 2; _pdate = ''''; _pltype = 1|6|1; _pframe = {0,0}; _pcross = 1;
  _pcolor = 12|11|12; _paxht = 0.25; _pnumht = 0.25; _ptitlht = 0.17;

  setwind(1);
    bound = rep1 + (-2~0~2).*stderr1;
    title(''Fig. 3.4. Estimated responses of consumption to a forecast''\
          ''\Lerror impulse in income with two-standard error bounds'');
    xy(period,bound);

  setwind(2);
    bound = rep2 + (-2~0~2).*stderr2;
    title(''Fig. 3.5. Estimated responses of investment to a forecast''\
          ''\Lerror impulse in consumption with two-standard error bounds'');
    xy(period,bound);

  setwind(3);
    bound = rep3 + (-2~0~2).*stderr3;
    title(''Fig. 3.6. Accumulated responses of consumption to a forecast''\
          ''\Lerror impulse in income with two-standard error bounds'');
    xy(period,bound);

  setwind(4);
    bound = rep4 + (-2~0~2).*stderr4;
    title(''Fig. 3.7. Accumulated responses of investment to a forecast''\
          ''\Lerror impulse in consumption with two-standard error bounds'');
    xy(period,bound);

  graphprt(''-c=1 -cf=tsm3i.eps'');

  endwind;
```

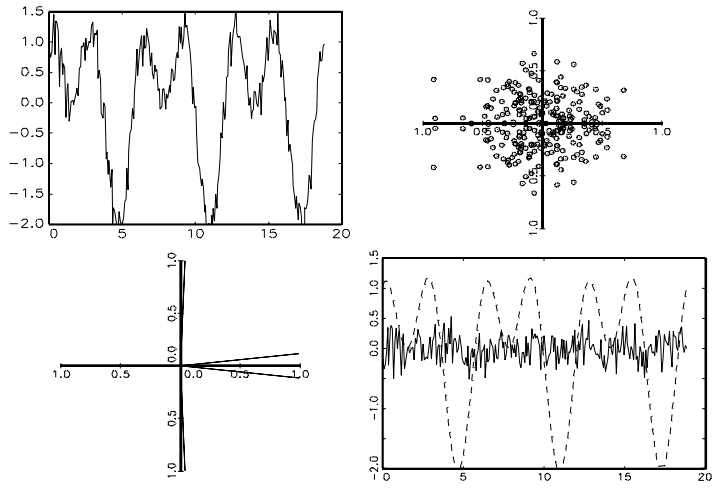


3.2.5 Spectral methods

3.2.5.1 Understanding the Fourier transform

```
new;
library tsm,optmum,pgraph;
TSMset;


@<--- Data --->@

N = 248;
t = seqa(0,6*pi/N,N+1);
x1 = sin(t);
x2 = cos(2*t);
x3 = rndn(N+1,1)*0.20;

y = x1 + x2 + x3;


@<--- Fourier transform --->@

d = fourier(y);

{r,theta} = topolar(d);


@<--- Inverse Fourier transform --->@

r1 = substute(r, r .> 100, 0);
d1 = tocart(r1,theta);
y1 = real(inverse_fourier(d1));

r2 = substute(r, r .< 100, 0);
d2 = tocart(r2,theta);
y2 = real(inverse_fourier(d2));


graphset;
  begwind;
  window(2,2,0);
    _pdate = ""; _pnum = 2; _pnumht = 0.25;

  setwind(1);
    ytics(-2,1.5,0.5,0);
    xtics(0,20,5,0);
    xy(t,y);

  setwind(2);
    graphset; _pnumht = 0.20; _plctrl = -1;
    polar(r1,theta);

  setwind(3);
    _plctrl = 0;
    polar(r2,theta);

  setwind(4);
```

```
    ytics(-2,1.5,0.5,0);
    xtics(0,20,5,0);
    xy(t,y1~y2);

    graphprt("-c=1 -cf=tsm4a.eps");

  endwind;
```



3.2.5.2 Periodogram, spectral estimation and cross-spectrum analysis

▶ Not yet done.

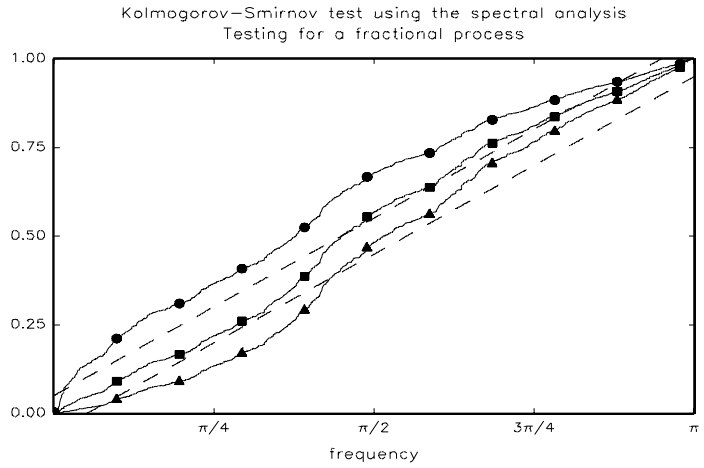
3.2.5.3 Surrogate data

▶ Not yet done.

3.2.5.4 Kolmogorov-Smirnov test

```
/*
**  Spectral Kolmogorov-Smirnov test
**
**  BROCKWELL and DAVIS [1991], Times Series: Theory and Methods,
**  Springer-Verlag, New York, pages 339-342
*/

new;
library tsm,optmum,pgraph;

load data[] = frfdem.asc;

r = packr(log(data./lag1(data))); /* Returns */

y1 = fractional_filter(r,-0.20);
y2 = fractional_filter(r,0);
y3 = fractional_filter(r,0.20);

_fourier = 0;
{lambda,I1} = PDGM(y1);
{lambda,I2} = PDGM(y2);
{lambda,I3} = PDGM(y3);

q = rows(lambda)/2;

C1 = cumsumc(I1[1:q])/sumc(I1[1:q]);
C2 = cumsumc(I2[1:q])/sumc(I2[1:q]);
C3 = cumsumc(I3[1:q])/sumc(I3[1:q]);

x = seqa(1,1,q);

k_alpha = -1.63~1.63;                          /* 1% level */
C_bound = (x-1)/(q-1) + k_alpha*(q-1)^(-0.5);

graphset;
  title(''Kolmogorov-Smirnov test using the spectral analysis''\
```

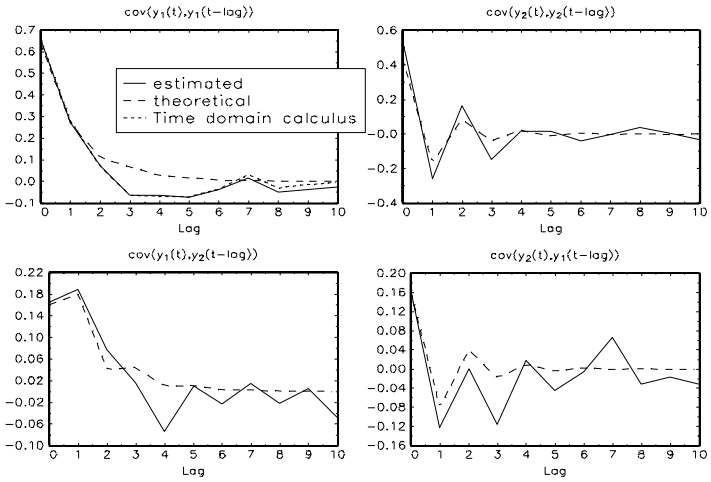
```
    ''\LTesting for a fractional process'');
  _pdate = ''''; _pnum = 2;
  _pltype = 6|6|6|1|1;
  _pstype = 8|9|10|1|1;
  _plctrl = 100|100|100|0|0;
  fonts(''simplex simgrma'');
  xtics(0,pi,pi/4,0); ytics(0,1,0.25,0);
  lab = '' 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201'';
  asclabel(lab,0);
  xlabel(''frequency'');
  graphprt(''-c=1 -cf=tsm4d.eps'');
  xy(x/q*pi,C1~C2~C3~C_bound);
```



3.2.5.5 Covariance functions

```
new;
library tsm,optmum,pgraph;
TSMset;

_fourier = 1;

rndseed 123456;

Z = eye(2); d = 0|0;
let H[2,2] = 0.2 0 0 0.1;
let T[2,2] = 0.5 0.3 0 -0.5;
c = 0|0; R = 1|1; Q = 0.25;

call SSM_build(Z,d,H,T,c,R,Q,0);
{y,a} = RND_SSM(0|0,100);
y = y - meanc(y)';

{lambda,I} = PDGM2(y);
g = sgf_SSM(lambda);

/* first component */

CV1a = real(inverse_fourier(I[.,1]));
CV1b = real(inverse_fourier(g[.,1]));
covTD = autoc(y[.,1],100);                /* Autocovariances     */

proc autoc(x,k);
  local t,rho;
  x=packr(x); t=rows(x);
  rho=rev(conv(x,rev(x),t-k,t));
  retp(rho/t);
endp;

/* second component */

CV2a = real(inverse_fourier(I[.,4]));
CV2b = real(inverse_fourier(g[.,4]));

/* first component/second component */

CV3a = real(inverse_fourier(I[.,3]));
CV3b = real(inverse_fourier(g[.,3]));
```

```
/* second component/first component */

CV4a = real(inverse_fourier(I[.,2]));
CV4b = real(inverse_fourier(g[.,2]));


@<--- Plots --->@

graphset;
  begwind;
  window(2,2,1);
    _pdate = ''''; _pltype = 6|1|3; _pnum = 2; _pnumht = 0.22; _ptitlht = 0.22; _paxht = 0.22;
    _plegstr = ''estimated\000theoretical\000Time domain calculus''; _plegctl = {2 7 3 3};
    xlabel(''Lag'');

  setwind(1);
    title(''cov(y]1[(t),y]1[(t-lag))'');
    xy(seqa(0,1,11),CV1a[1:11]~CV1b[1:11]~covTD[1:11]);

    _plegctl = 0;

  setwind(2);
    title(''cov(y]2[(t),y]2[(t-lag))'');
    xy(seqa(0,1,11),CV2a[1:11]~CV2b[1:11]);

  setwind(3);
    title(''cov(y]1[(t),y]2[(t-lag))'');
    xy(seqa(0,1,11),CV3a[1:11]~CV3b[1:11]);

  setwind(4);
    title(''cov(y]2[(t),y]1[(t-lag))'');
    xy(seqa(0,1,11),CV4a[1:11]~CV4b[1:11]);

  graphprt(''-c=1 -cf=tsm4e.eps'');

  endwind;
```



3.2.5.6 Varma parameters estimation in the frequency domain

```
new;
library tsm,optmum,pgraph;
TSMset;

output file = tsm4f.out reset;

load reinsel[100,2] = reinsel.asc;

invest = reinsel[.,1];
invent = reinsel[.,2];
di = invest - lag1(invest);
y = di~invent;

{lambda,Iy} = PDGM2(y);         /* Multidimensional periodogram */

/*
** Procedure to compute the multidimensional sgf of the SSM
```

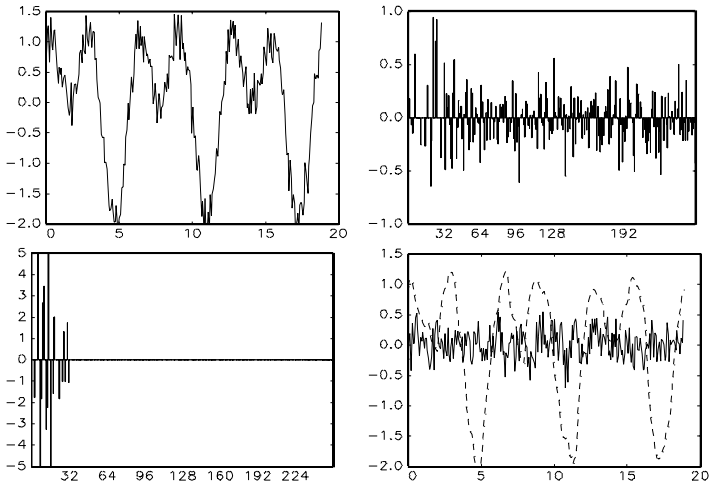
```
*/

proc sgf(theta,lambda);
  local beta,Pchol,SIGMA,T,Q,H,Z,d,c,R;
  local Gy;

  beta = theta[1:8];
  Pchol = (theta[9]~0)|(theta[10]~theta[11]);
  SIGMA = Pchol*Pchol';

  {Z,d,H,T,c,R,Q} = arma_to_SSM(beta,1,1,SIGMA);
  call SSM_build(Z,d,H,T,c,R,Q,0);
  Gy = sgf_SSM(lambda);

  retp(Gy);
endp;


/*
** Procedure to compute the log-likelihood function in frequensy domain
*/


proc ml(theta);
  local Gy,Nstar,logl,j,Ij,Gj;

  Gy = sgf(theta,lambda);
  Nstar = rows(lambda);

  logl = zeros(Nstar,1);

  j = 1;
  do until j > Nstar;
   Ij = xpnd2(Iy,j);
   Gj = xpnd2(Gy,j);
   logl[j] = -0.5*ln(det(Gj)) -0.5*sumc(diag((inv(Gj)*Ij)));
   j = j + 1;
  endo;

  logl = real(logl);

  retp(logl);
endp;


_tsm_Mcov = 1;
load arma1, arma2;
sv = arma1|vech_(chol(arma2));    /* Starting values = CMLE parameters */

{theta,stderr,Mcov,Logl} = TD_ml(&ml,sv);

output off;


Total observations:                                    99
Usable observations:                                   99
Number of parameters to be estimated:                  11
Degrees of freedom:                                    88
Value of the maximized log-likelihood function:       -327.01770


Parameters      estimates       std.err.      t-statistic         p-value
-------------------------------------------------------------------------
P01              0.341743        0.192283         1.777287        0.078976
P02              0.606935        0.259933         2.334965        0.021821
P03             -0.019407        0.048163        -0.402937        0.687972
P04              0.839086        0.051853        16.181872        0.000000
P05             -0.202206        0.201916        -1.001434        0.319363
P06              0.218497        0.342886         0.637229        0.525630
P07             -0.175699        0.071158        -2.469131        0.015476
P08              0.194652        0.111825         1.740682        0.085235
P09              2.422907        0.172510        14.044984        0.000000
P10              1.053449        0.425994         2.472921        0.015323
P11              4.129853        0.294031        14.045634        0.000000

Covariance matrix: inverse of the negative Hessian.
```

3.2.6 Wavelets analysis

3.2.6.1 Understanding the Wavelet and Wavelet Packets transforms

```
new;
library tsm,optmum,pgraph;
```

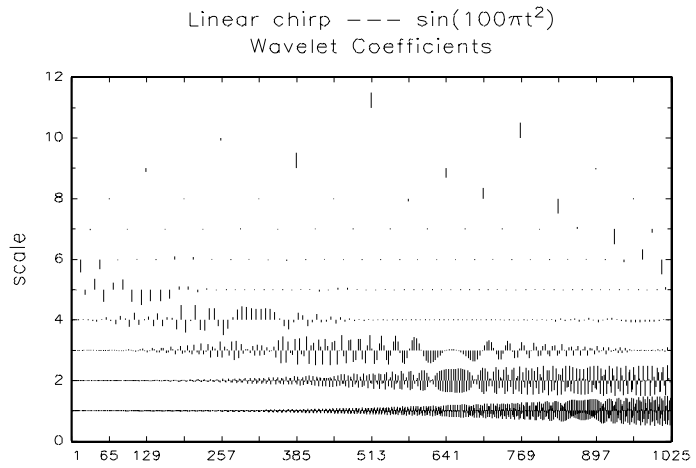
```
TSMset;


@<--- Data --->@

N = 255;
t = seqa(0,6*pi/N,N+1);
x1 = sin(t);
x2 = cos(2*t);
x3 = rndn(N+1,1)*0.20;

y = x1 + x2 + x3;


@<--- wavelet transform --->@

{H,G,Htilde,Gtilde} = Coiflet(2);

_wcenter = 0;
w = wt(y,H,G,0);

w1 = substute(w, abs(w) .> 1, 0);
y1 = iwt(w1,Htilde,Gtilde,0);

w2 = substute(w, abs(w) .< 1, 0);
y2 = iwt(w2,Htilde,Gtilde,0);


graphset;
  begwind;
  window(2,2,0);
    _pdate = ''''; _pnum = 2; _pnumht = 0.25;

  setwind(1);
    ytics(-2,1.5,0.5,0);
    xtics(0,20,5,0);
    xy(t,y);

  setwind(2);
    xtics(0,rows(w),32,0);
    ytics(-1,1,0.5,0);
    bar(seqa(1,1,rows(w)),w1);

  setwind(3);
    ytics(-5,5,1,0);
    bar(seqa(1,1,rows(w)),w2);

  setwind(4);
    ytics(-2,1.5,0.5,0);
    xtics(0,20,5,0);
    xy(t,y1~y2);

    graphprt(''-c=1 -cf=tsm5a.eps'');

  endwind;
```

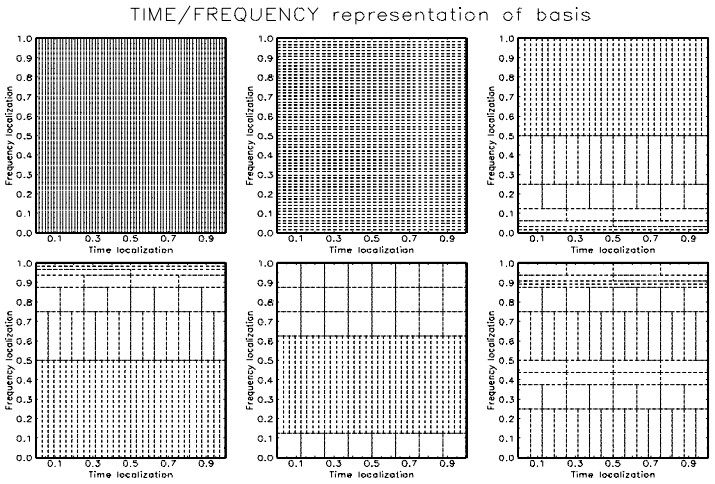


3.2.6.2 Wavelet and Wavelet Packets representation

```
new;
library tsm,optmum,pgraph;

M = 11;
N = 2^M;
t = seqa(0,2/N,N);
chirp = sin(100*pi*t^2);

{H,G,Htilde,Gtilde} = Coiflet(2);

_wcenter = 0;
w = wt(chirp,H,G,0);          /* Wavelet transform */

pkt = wpkt(chirp,H,G,0);      /* Wavelet Packet transform */


graphset;
  call wplot(w,0,1);
  _pnum = 2; _pdate = ''''; _ptitlht = 0.20; _paxht = 0.20;
  fonts(''simplex simgrma'');
  title(''Linear chirp --- sin(100\202p\201t[2])\LWavelet Coefficients'');
  ylabel(''scale'');
  xtics(1,N/2,64,0);
  graphprt(''-c=1 -cf=tsm5b.eps'');
  draw;

graphset;
  _wgrid = 0; _wcolor = seqa(1,1,6);
  call wpkPlot(pkt,1);
  _pnum = 2; _pdate = ''''; _ptitlht = 0.20; _paxht = 0.20;
  fonts(''simplex simgrma'');
  title(''Linear chirp --- sin(100\202p\201t[2])''\
        ''\LWavelet Packet Coefficients'');
  xtics(0,N-1,128,0);
  draw;


M = 6;
Nobs = 2^M;

call varput(0,''Base0'');                /* Time domain */
call varput(M*ones(Nobs,1),''Base1'');   /* Frequency domain */

call varput(M|seqa(M,-1,M),''Base2'');   /* Wavelet transform
                                            (better frequency localization at
                                             lower frequencies)
                                         */

call varput(seqa(1,1,M)|M,''Base3'');    /* opposite of the wavelet basis
                                            (better frequency localization at
                                             higher frequencies)
                                          */

call varput(3|1|3|3|3,''Base4'');           /* Wavelet packet transform */
call varput(2|3|4|4|2|3|6|6|5|4,''Base5''); /* Wavelet packet transform */


graphset;
  begwind;
  window2(2,3,0,0.5);

  setwind(1);

    _pdate = ''''; _pnum = 0; _paxes = 0; _ptitlht = 2.5;
    title(''TIME/FREQUENCY representation of basis'');
    draw;

    _pnum = 2; _paxes = 1; _ptitlht = 0.20; _pnumht = 0.25; _paxht = 0.25;
    title('''');

i = 0;
do until i > 5;

  str = ''Base'' $+ ftos(i,''%lf'',1,0);
  B = varget(str);

  setwind(2 + i);

    call BasisPlot(B,M);
    xlabel(''Time localization'');
    ylabel(''Frequency localization'');
    draw;
```

```
  i = i + 1;
endo;

  graphprt(''-c=1 -cf=tsm5b2.eps'');

  endwind;
```

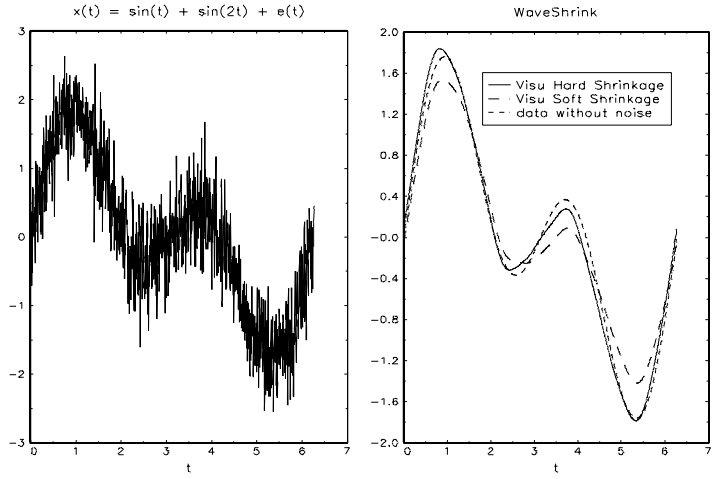




3.2.6.3 Data denoising

```
new;
library tsm,optmum,pgraph;

rndseed 123;

Nobs = 2^10;

t = seqa(0,2*pi/Nobs,Nobs);
x_ = sin(t) + sin(2*t);
x = x_ + rndn(Nobs,1)*0.4;

{H,G,Htilde,Gtilde} = Daubechies(12);

_wcenter = 0;
w = wt(x,H,G,0);

w1 = VisuShrink(w,''Hard'');
w2 = VisuShrink(w,''Soft'');

y1 = iwt(w1,Htilde,Gtilde,0);
y2 = iwt(w2,Htilde,Gtilde,0);
```

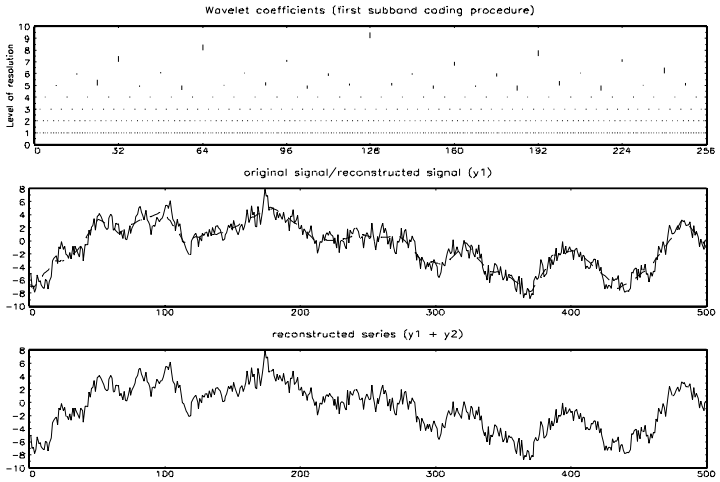
```
y = y1~y2;

graphset;
  fonts(''simplex simgrma'');
  _pdate = ''''; _pnum = 2;
  begwind;
  window(1,2,0);

  setwind(1);
    _pnumht = 0.20;
    _paxht = 0.25;
    _ptitlht = 0.25;
    title(''x(t) = sin(t) + sin(2t) + e(t)'');
    xlabel(''t'');
    xy(t,x);

  setwind(2);
    _pltype = 6|1|3|4;
    _plwidth = 0;
    _plegstr = ''Visu Hard Shrinkage''\
               ''\0Visu Soft Shrinkage''\
               ''\0data without noise'';
    _plegctl = {2 6 3 5};
    title(''WaveShrink'');
    xy(t,y~x_);

    graphprt(''-c=1 -cf=tsm5c.eps'');
  endwind;
```



3.2.6.4 Subbands coding

```
new;
library pgraph,tsm,optmum;

rndseed 1234;

Nobs = 2^9;

s = seqa(1,1,Nobs);
sigma = 1;
x0 = miss(0,0);
x = RND_arma(0.62|0.35,2,0,sigma,x0,Nobs);

{H,G,Htilde,Gtilde} = Daubechies(6);

_wcenter = 2;
w = wt(x,H,G,0);

L1 = 5|6|7|8|9|10;  /* lower frequencies */
L2 = 1|2|3|4;    /* higher frequencies */

bnd1 = Extract(w,L1);
bnd2 = Extract(W,L2);
y1 = iwt(bnd1,Htilde,Gtilde,0);
y2 = iwt(bnd2,Htilde,Gtilde,0);
y3 = y1 + y2;
```

```
  begwind;
  window(3,1,0);
  setwind(1);
    graphset;
    _pdate = ''''; _pnum = 2; _pnumht = 0.25; _ptitlht = 0.30; _paxht = 0.25;
    title(''Wavelet coefficients (first subband coding procedure)'');
    _wcolor = seqa(1,1,9); _wline = 0;
    call wplot(w,0,bnd1);
    xtics(0,Nobs/2,32,0);
    draw;
  setwind(2);
    graphset;
    _pdate = ''''; _pnum = 2; _pnumht = 0.25; _ptitlht = 0.30; _paxht = 0.25;
    xtics(0,500,100,0);
    title(''original signal/reconstructed signal (y1)'');
    xy(s,x~y1);
  setwind(3);
    title(''reconstructed series (y1 + y2)'');
    xy(s,y3);

    graphprt(''-c=1 -cf=tsm5d.eps'');
  endwind;
```



3.2.6.5 Scalogram

```
/*
**  ARINO and VIDAKOVIC [1995], On wavelet scalograms and their applications
**  in economic time series, DP 95-21, ISDS, Duke University
**
**  ARINO [1995], Time series forecasts via wavelets: an application
**  to car sales in the spanish market, DP 94-30, ISDS, Duke University
**
*/

new;
library tsm,optmum,pgraph;

C1 = intquad1(&gfunc,(8*pi)|0)/(8*pi);
C2 = intquad1(&hfunc,(8*pi)|0)/(8*pi);

t = seqa(1,1,2^12);
yt = gfunc(8*pi*t/4096) - C1;
zt = hfunc(8*pi*t/4096) - C2;
xt = yt+zt;

{H,G,Htilde,Gtilde} = Daubechies(12);
w = wt(xt,H,G,0);

output file = tsm5e.out reset;

E = scalogram(w);

output off;

bnd = split(w,7);

x1 = iwt(bnd[.,1],Htilde,Gtilde,0);
```

```
x2 = iwt(bnd[.,2],Htilde,Gtilde,0);

graphset;
  begwind;
  window(2,2,0);
  _pnum = 2; _pdate = ''''; _ptitlht = 0.25; _pnumht = 0.20;

  setwind(1);
    title(''data'');
    xy(t,xt);

  nextwind;
    _pnumht = 0.16;
    lab = '']c]0[[2][ ]R(0)[ ]E(1)[ ]E(2)[ ]E(3)[ ]E(4)[ ]E(5)[ ]E(6)[ ''\
          '']E(7)[ ]E(8)[ ]E(9)[ ]E(10)[ ]E(11)['';
    xtics(0,13,1,0);
    asclabel(lab,0);
    title(''scalogram'');
    bar(0,E);

  nextwind;
    graphset; _ptitlht = 0.25; _pnumht = 0.20; _pnum = 2;
    title(''trend'');
    xy(t,x2);

  nextwind;
    title(''cycle'');
    xy(t,x1);

  endwind;


proc (1) = gfunc(x);
  local y;
  y = 2*sin(x);
  retp(y);
endp;

proc (1) = hfunc(x);
  local y;
  y = abs(arcsin(sin(16*x/pi)));
  retp(y);
endp;
```

```
-------------------------------
         Scalogram
-------------------------------
  c(0)^2         112.01955
  E(0)             0.02141
  E(1)             0.05578
  E(2)          4968.64820
  E(3)          3208.63411
  E(4)            19.69621
  E(5)            55.86363
  E(6)           748.18123
  E(7)            29.73965
  E(8)             6.21492
  E(9)             0.91120
  E(10)            0.16022
  E(11)            0.09792
```

3.2.6.6 A financial example

- Not yet done.

Chapter 4

Developing professional applications using Gauss programs

4.1 The Gauss Engine

The Gauss Engine is a dynamic library that can be linked in with any program written in C, C++, Visual Basic, Delphi, Java or many other development environments, that allows your application to compile and execute Gauss programs and pass data between it and the Gauss workspace.

4.1.1 What is the Gauss Engine?

Gauss Engine is a DLL file:

target_path	*gauss.dll & gseng.dll*
*target_path***lib**	gauss.lcg
*target_path***src**	source code files of the Gauss library

Remark 3 *There are no executable programs in Gauss Engine and we do not need Gauss to use Gauss Engine.*

Gauss Engine consists of 34 functions (Initialization and Shutdown, Compilation and Execution, Data Handling, Normal I/O, Critical and Error I/O). The main Gauss Engine API functions are:

- **GAUSS_O_Initialize**: Initializes the engine.
- **GAUSS_O_Shutdown**: Shuts the engine down.
- **GAUSS_O_CompileFile**: Compiles a file.
- **GAUSS_O_CompileString**: Compiles a string buffer.
- **GAUSS_O_Execute**: Executes the last compiled program.
- **GAUSS_O_Get2DMatrix**: Gets matrices from the GAUSS symbol table.
- **GAUSS_O_Set2DMatrix**: Sets matrices in the Gauss symbol table.

4.1.2 Understanding the Gauss Engine

▶ **Gauss Engine is a DLL file.**

```
new;
dlibrary c:\engine\gseng.dll;

str = ''output file = ge1.out reset;''\
      ''rndseed 123; x = rndu(200,200);''\
      ''t0 = hsec; y = inv(x); t1 = hsec;''\
      ''print /flush (t1-t0); output off;'';

dllcall GAUSS_O_Initialize;
```

```
dllcall /r GAUSS_O_CompileString(str);
dllcall /r GAUSS_O_Execute;
dllcall GAUSS_O_Shutdown;

output file = ge1.out on;

print ''========================================'';

rndseed 123;
z = rndu(200,200);
t0 = hsec;
y = inv(z);
t1 = hsec;
print /flush (t1-t0);

output off;


       28.000000
======================================
       27.000000
```

- **Gauss Engine executes Gauss programs, but Gauss Engine and Gauss are two differents products. In the example below, GE has its proper symbol tables.**

```
new;
dlibrary c:\engine\gseng.dll;

output file = ge2.out reset;

dllcall /r GAUSS_O_Initialize;

str = ''y = rndu(3,3); x = y[1,1];'';

dllcall /r GAUSS_O_CompileString(str);
dllcall /r GAUSS_O_Execute;

show;
print ''===================================='';

x = rndn(3,3);

show;
print ''===================================='';

output off;

str = ''output file = ge2.out on;''\
      ''x = x^2;''\
      ''show; output off;'';

dllcall /r GAUSS_O_CompileString(str);
dllcall /r GAUSS_O_Execute;
dllcall GAUSS_O_Shutdown;


STR
      32 bytes at [01320020]      26 char    STRING
X
       0 bytes at [00000000]   uninitialized MATRIX

65536 bytes program space, 1% used
62849008 bytes workspace, 62848976 bytes free
2 global symbols, 1500 maximum, 2 shown
====================================

STR
      32 bytes at [01320020]      26 char    STRING
X
      72 bytes at [01320040]       3,3       MATRIX

65536 bytes program space, 1% used
62849008 bytes workspace, 62848904 bytes free
2 global symbols, 1500 maximum, 2 shown
====================================

       8 bytes at [06d00068]  x              1,1          MATRIX
      72 bytes at [06d00020]  y              3,3          MATRIX

256000 bytes program space, 0.039% used
10718136 bytes workspace, 10718056 bytes free
2 global symbols, 2000 maximum, 2 shown
0 active locals, 2000 maximum
```

▶ **Gauss Engine could run Gauss files.**

```
new;
dlibrary c:\engine\gseng.dll;

dllcall /r GAUSS_0_Initialize;

prg = ''d:\\gauss\\conf3\\var1.prg'';

dllcall /r GAUSS_0_CompileFile(prg);
dllcall /r GAUSS_0_Execute;
dllcall GAUSS_0_Shutdown;
```

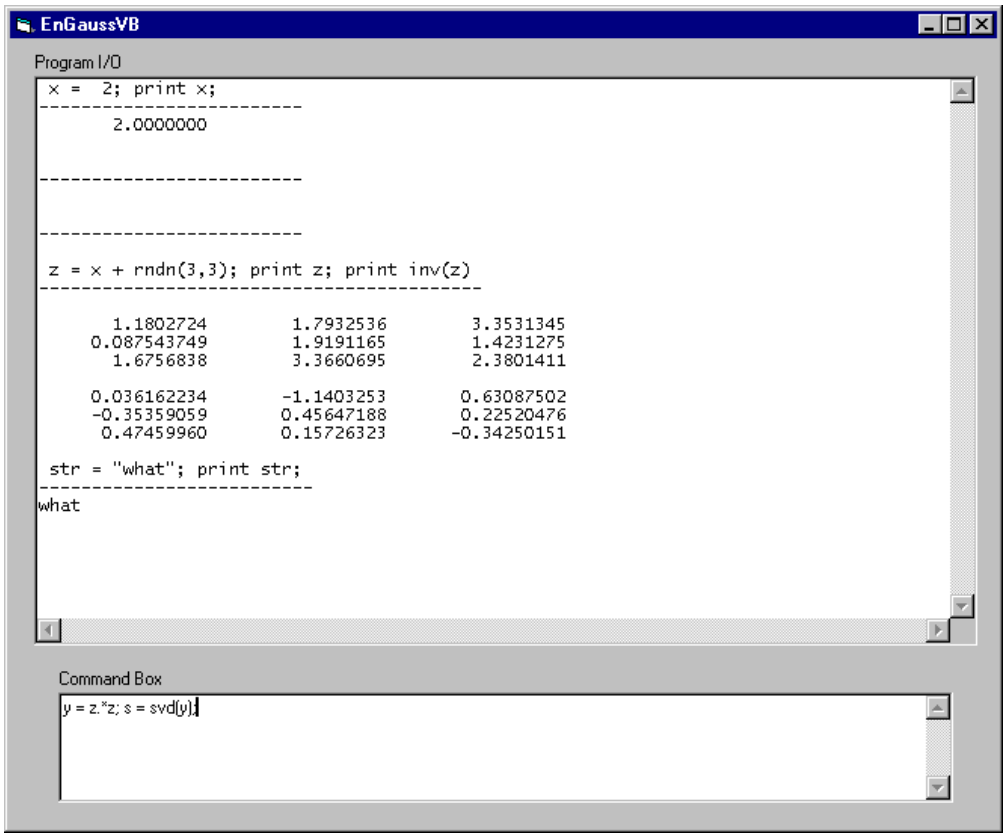
▶ **Gauss Engine is an open system and allows communication with the application.**

```
#include <stdio.h>
#include <stdlib.h>

#include ''gseng.h''

#define FALSE 0
#define TRUE (!FALSE)

#define XR 4
#define XC 4

double x[XR*XC] =
{
   0.281130110612139,   0.333434643689543,   0.536355309421197,   0.057482290081680,
   0.944966180948541,   0.068411342334002,   0.625666477950290,   0.730278179049492,
   0.522050328319892,   0.058814641553909,   0.677300492534414,   0.838403818197548,
   0.351553247077391,   0.404659466352314,   0.034298057900742,   0.210895204916596,
};

#define YR 4
#define YC 4

double y[YR*YC] =
{
   0.577031770488247,   0.043839922640473,   0.883301105583087,   0.008838660083711,
   0.406743812141940,   0.479973535519093,   0.185277891578153,   0.918547097593546,
   0.843689869856462,   0.116690800059587,   0.990037156501785,   0.833994106389582,
   0.276006092084572,   0.276495044585317,   0.150156348245218,   0.231630844995379,
};

double *z;


int main(int argc,char *argv[])
{
 Mat2D zDesc;
 int i,j,zR,zC;


 GAUSS_0_Initialize();
 GAUSS_0_Set2DMatrix( ''x'', XR, XC, 0, x );
 GAUSS_0_Set2DMatrix( ''y'', YR, YC, 0, y );
 GAUSS_0_CompileString(''format /rds 20,15; print y/x; z = y/x;'');
 GAUSS_0_Execute();
 GAUSS_0_Get2DMatrix( &zDesc, ''z'' );


 z = zDesc.address;
        zR = zDesc.rows;
        zC = zDesc.cols;


 for ( i = 0; i < zR; i++ )
 {
  printf(''\n'');
  for ( j = 0; j < zC; j++ ) printf(''%20.15lf '', *(z + i*zC + j) );
 }
 printf(''\n'');


 GAUSS_0_Shutdown();

 return 0;
}
```

4.1.3 Some examples with Excel

▶ Not yet done.

4.1.4 Some examples with Visual Basic



4.2 Mercury_GE

Mercury_GE consists of a set of functions that enable programmers to interact with the Gauss Engine, as part of an application. These tools consist of a library (DLL) of core APIs that are called from an external application. Support files for this library are provided for Excel, Visual Basic, Visual J++, and C++; however these libraries can be ported to any program that can access the Windows APIs - for example, Delphi, Powerbuilder, Toolbook, etc. These routines permit sending strings, values and data between the external application and the Gauss Engine, as well as routines for managing the Gauss Engine and status checking. Demonstration projects for Excel, VB, VJ and C applications are included and show how these calls are implemented.

4.2.1 What is Mercury_GE?

The Mercury_GE API functions are:

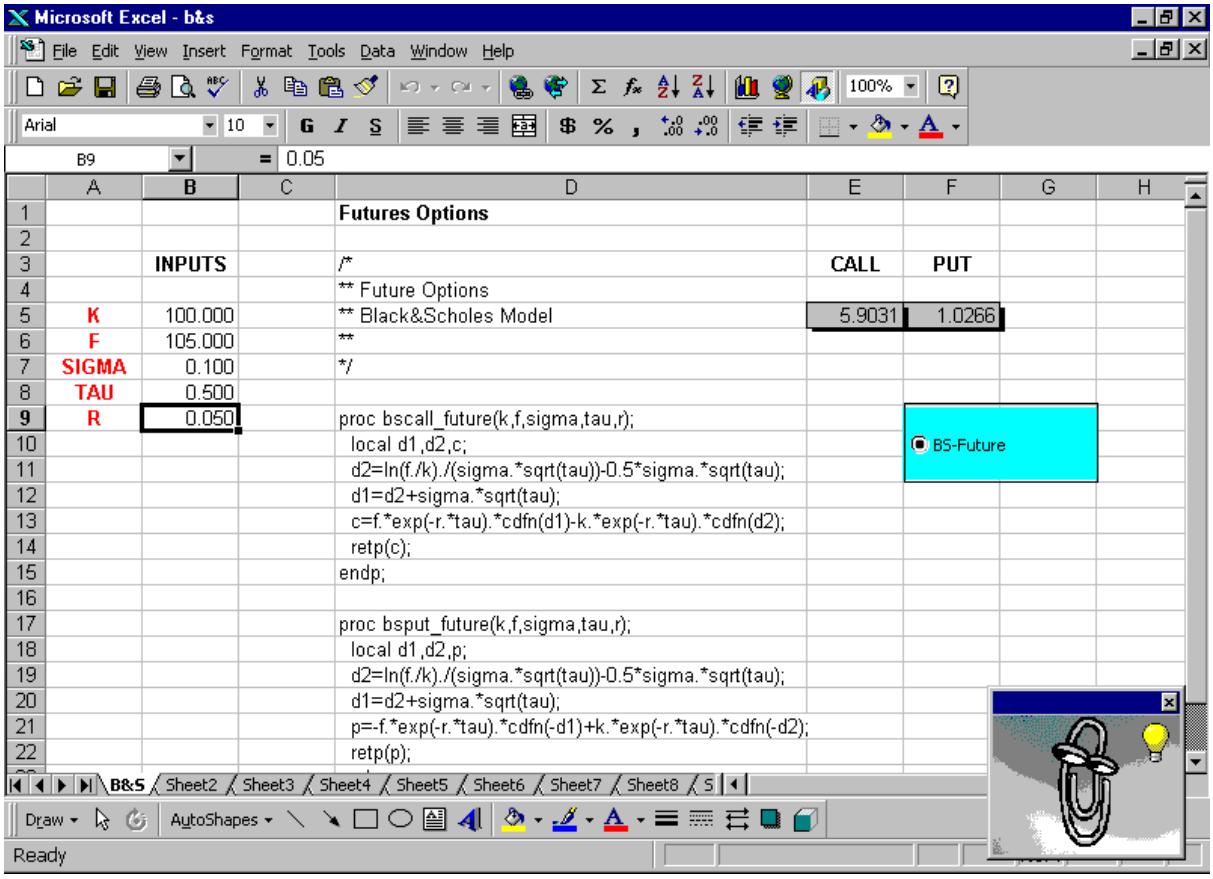
- **geclose()**: Closes the Gauss Engine.
- **geexec(ByVal strn As String)**: Compiles and executes a string buffer or a file (or cells of the spreadsheet).
- **gematget(ByVal strng As String, ByRef target() As Double)**: Gets matrices from the Gauss symbol table.
- **gematput(ByVal strng As String, matr() As Double)**: Sets matrices in the Gauss symbol table.
- **gematsize(ByVal strng As String, lr, lc)**: Size of a matrix (Gauss symbol).
- **geopen()**: Opens the Gauss Engine.
- **geopencheck()**: Checks if the Gauss Engine is opened.
- **geoutput(mode As String)**: Displays the output with NotePad.

- **gestrget(ByVal strng As String, target As String)**: Gets strings from the Gauss symbol table.
- **gestrput(ByVal str As String, ByVal source As String)**: Sets matrices in the Gauss symbol table.
- **gestrsize(ByVal strng As String, slen As Long)**: Length of a string (Gauss symbol).
- **gevalget(ByVal strng As String, value)**: Gets scalars from the Gauss symbol table.
- **gevalput(ByVal strng As String, value)**: Sets scalars in the Gauss symbol table.

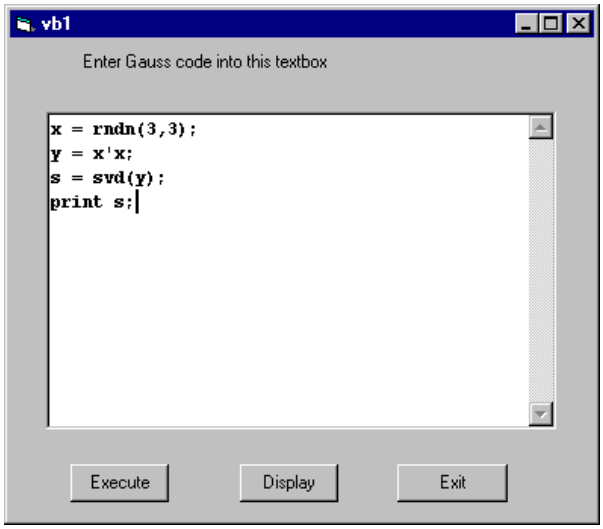
4.2.2 Some Mercury_GE examples

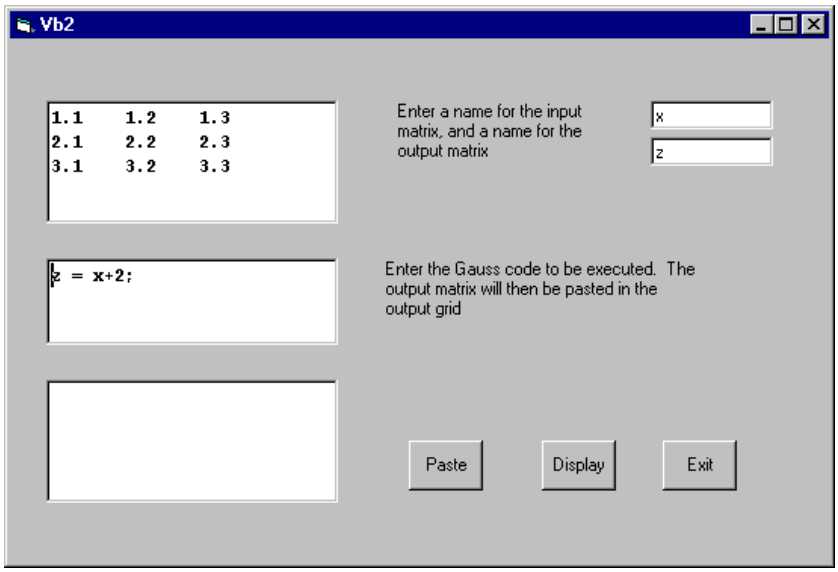
4.2.2.1 Excel example

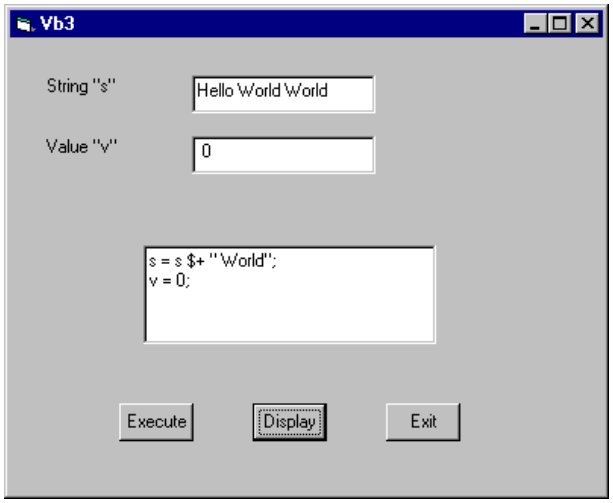
```
'*******************************************************

'Black&Scholes model


Sub bs_future()

   Sheets(''B&S'').Select
   gematput ''k'', ''b5''
   gematput ''f'', ''b6''
   gematput ''sigma'', ''b7''
   gematput ''tau'', ''b8''
   gematput ''r'', ''b9''

   'call
   geexec ''d9:d15''
   geexec ''c = bscall_future(k,f,sigma,tau,r);''
   gematget ''c'', ''e5''

   'put
   geexec ''d17:d23''
   geexec ''p = bsput_future(k,f,sigma,tau,r);''
   gematget ''p'', ''f5''

End Sub
```



VB example





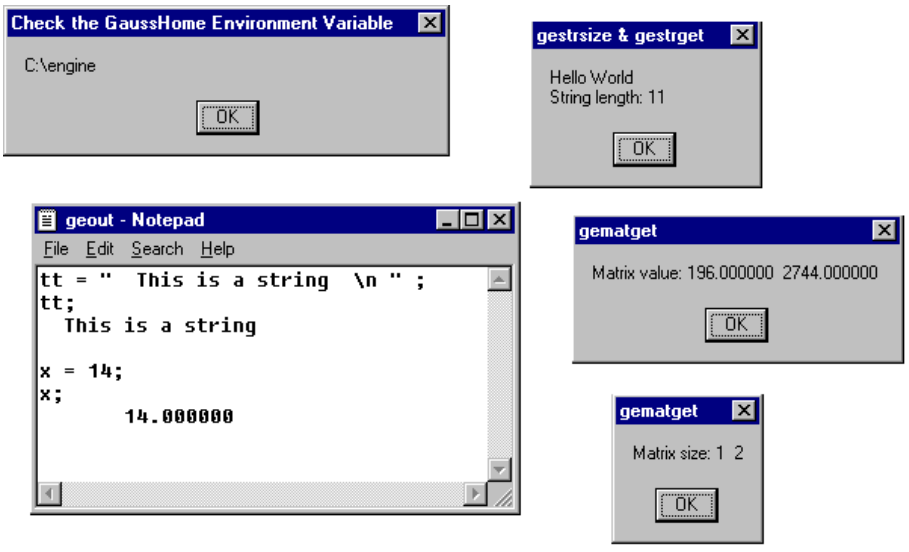


4.2.2.2 VC++ example

```
// cprg1.cpp : Defines the entry point for the application.
//
```

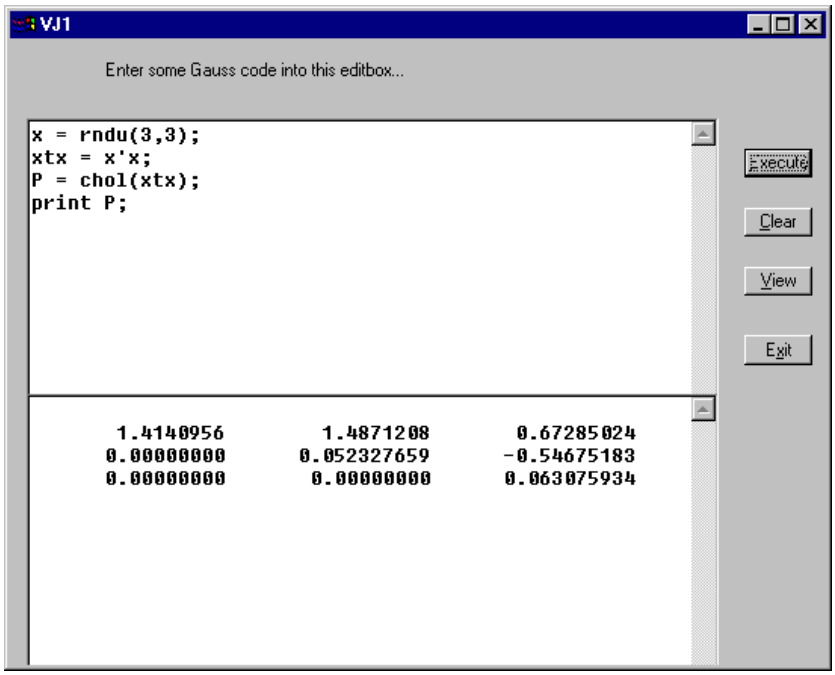
```
#include ''stdafx.h''


int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
  // TODO: Place code here.

 Mat2D zzDesc;
 char buffer[256];
 int ii, jj;
 int slen[1];
 double zz[1];
 double *z;
 int pval = 0;
 char  *sname = ''GAUSSHOME'';
 long nsize=256 ;
 char sbuff[256];
 int len = 256;

    if(GetEnvironmentVariable(sname, sbuff,nsize)== 0)
 {
  MessageBox(0,''The GAUSSHOME environment variable was not found'',''Mercury_GE'',0);
  return 1;
 }
 else
  MessageBox(0,sbuff,''Check the GaussHome Environment Variable'',0);

 if (strlen(lpCmdLine) != 0)
 {
  MessageBox(0,lpCmdLine,''Execute File'',0);
       geexec(lpCmdLine);
       geoutput(''show'');
       return 0;
 }

 pval = geopen();
 if (pval == 1)
 {
  MessageBox(0,''Failed to open'',''cprg'',0); return 1;
 }

//  Create a gauss exec
 geexec('' tt = \''  This is a string  \\n \'' ; tt; '');
 geexec (''x = 14; x;'');
 geoutput(''show'');
 geoutput(''reset'');

//  Create a gauss string
 strcpy(buffer,''The fox went out on a chilly night'');
 gestrput(''str'', buffer);
 geexec (''str; '');
 geoutput(''show'');
 geoutput(''reset'');


//  Create a Gauss matrix
 double q[4*3] =  {   0.281130110612139,   0.333434643689543,   0.536355309421197,
                      0.944966180948541,   0.068411342334002,   0.625666477950290,
                           0.522050328319892,   0.058814641553909,   0.677300492534414,
                           0.351553247077391,   0.404659466352314,   0.034298057900742,
 };
 zzDesc.rows = 4;
 zzDesc.cols = 3;
 zzDesc.address = q;
 gematput(''qmat'', &zzDesc);
 geexec ('' qmat;'');
 geoutput(''show'');
 geoutput(''reset'');


//  Create a Gauss scalar
 zz[0] = 1.2;
 gevalput(''zz'',zz);
 geexec ('' zz;'');
 geoutput(''show'');
 geoutput(''reset'');

//  Execute Gauss code
 geexec(''x = 14; yy = x^2 ~x^3;'' );

//  Retrieve Gauss matrix
```

```
 gematget(''yy'', &zzDesc);
 z = zzDesc.address;
 ii = zzDesc.rows;
 jj = zzDesc.cols;
 sprintf(buffer,''Matrix size: %d  %d'', ii,jj);
 MessageBox(0, buffer, ''gematget'', 0);
 sprintf(buffer,''Matrix value: %f  %f '', *z, *(z+1) );
 MessageBox(0, buffer, ''gematget'', 0);

//  Retrieve Gauss scalar
 geexec('' z = -14.7;'');
 gevalget(''z'', zz);
 sprintf(buffer,''Scalar value: %f '', zz[0] );
 MessageBox(0, buffer, ''gevalget'', 0);

//  Retrieve Gauss string
 gestrput(''hw'',''Hello World'');
 gestrsize(''hw'',slen);
 gestrget(''hw'', buffer);
 sprintf(sbuff,''\nString length: %i '', slen[0] );
 strcat(buffer,sbuff);
 MessageBox(0, buffer, ''gestrsize & gestrget'', 0);
 geoutput(''reset'');

// clipboard support

 geexec(''x = rndn(3,2); x'');
 gecopy(''x'');
 gestrput(''tt'', ''x is copied to the clipboard '');
 geexec('' tt;'');
 geexec('' new '');
 gepaste(''z'');
 gestrput(''tt'', '' z is pasted from the clipboard '');
 geexec(''tt; z;'');
 geoutput(''show'');
 geoutput(''reset'');

 return 0;
}
```



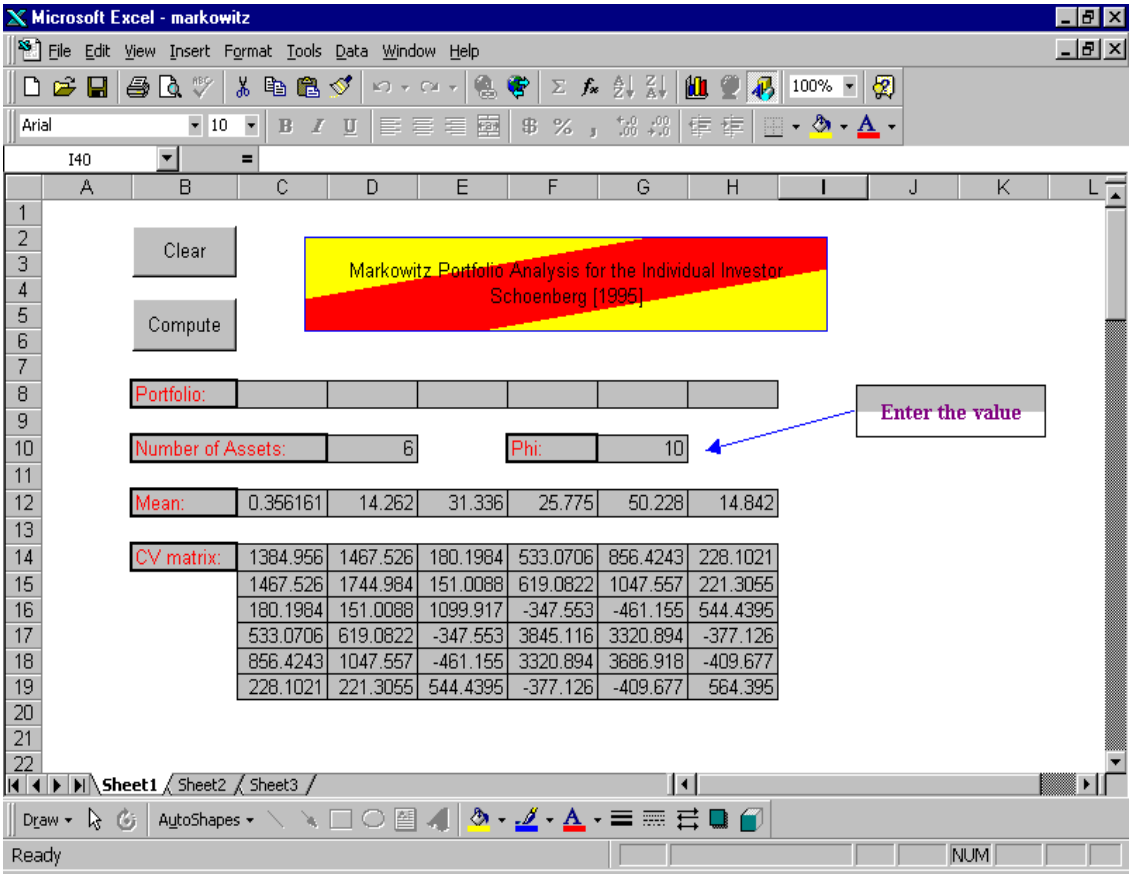
4.2.2.3 VJ++ example

- Needs VM Java component.



4.2.3 Three explained examples

4.2.3.1 Markowitz portfolio and Excel

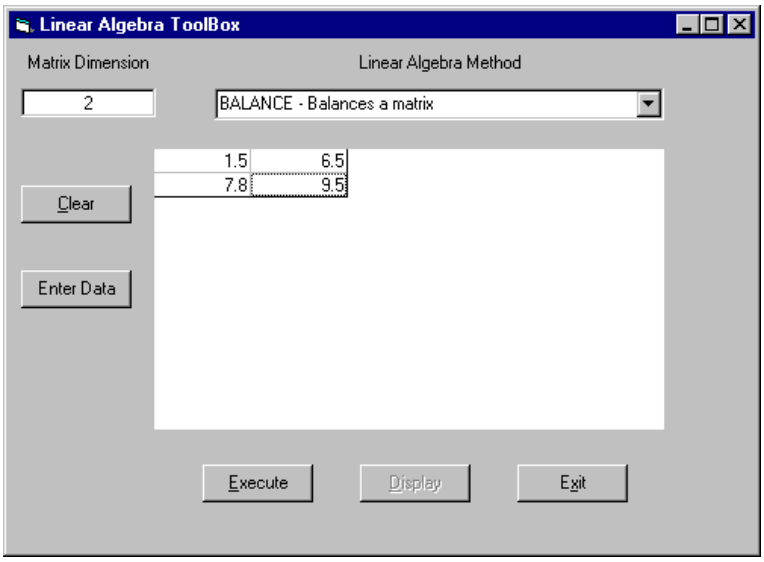


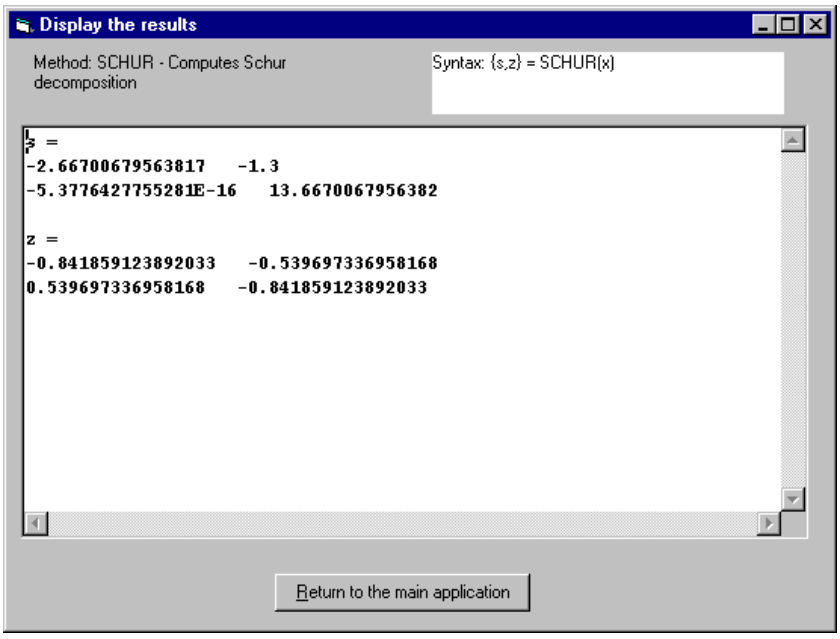
- The VBA code is:

```
Sub compute()
    Dim GaussProgram As String
```

```
    Sheets(''sheet1'').Select

    'geopen

    gematput ''phi'', ''g10''
    gematput ''mu'', ''c12:h12''
    gematput ''Mcov'', ''c14:h19''
    GaussProgram = ''N = rows(Mcov); sv = ones(N,1)/N; Q = 2*Mcov; '' + _
                   ''R = phi * mu '; A = ones(1,N); B = 1; '' + _
                   ''C = eye(N); D = zeros(N,1); '' + _
                   ''{theta,u1,u2,u3,u4,retcode} = Qprog(sv,Q,R,A,B,C,D,0); '' + _
                   ''theta = theta' .* dotfne(theta',0);''
    geexec GaussProgram
    gematget ''theta'', ''c8''

    'geclose
End Sub
-------------------------------------------
Sub Clear()
    Sheets(''sheet1'').Select
    Range(''g10'').Value = ''''
    Range(''c8:h8'').Value = ''''
End Sub
```

4.2.3.2 Linear algebra and Visual Basic





▶ The VB code of the first window is:

```
Private Sub cmdClear_Click()
    grdMatrix.Clear
    cmdExecute.Enabled = False
End Sub
-------------------------------------------
Private Sub cmdDisplay_Click()
    frmVB1.Hide
    frmVB2.cmdDisplay
    frmVB2.Show
End Sub
-------------------------------------------
Private Sub cmdExecute_Click()
    Dim InputMatrix() As Double
    Dim N As Integer
    Dim i, j As Integer
    Dim InputMatrixName As String
    Dim GaussProgram As String

    N = Val(txtDimension.Text)
    ReDim InputMatrix(1 To N, 1 To N)

    For i = 1 To N Step 1
        grdMatrix.Row = i - 1
        For j = 1 To N Step 1
            grdMatrix.Col = j - 1
            InputMatrix(i, j) = Val(grdMatrix.Text)
        Next j
    Next i

    InputMatrixName = ''x''
    gematput InputMatrixName, InputMatrix()

    Select Case cboLinearAlgebra.ListIndex
        Case 0
            GaussProgram = ''c:\engine\prg\balance.prg''
            geexec GaussProgram
        Case 1
            GaussProgram = ''c:\engine\prg\cond.prg''
            geexec GaussProgram
        Case 2
            GaussProgram = ''c:\engine\prg\crout.prg''
            geexec GaussProgram
        Case 3
            GaussProgram = ''c:\engine\prg\hess.prg''
            geexec GaussProgram
        Case 4
            GaussProgram = ''c:\engine\prg\lu.prg''
            geexec GaussProgram
        Case 5
            GaussProgram = ''c:\engine\prg\null.prg''
            geexec GaussProgram
        Case 6
            GaussProgram = ''c:\engine\prg\pinv.prg''
            geexec GaussProgram
        Case 7
            GaussProgram = ''c:\engine\prg\qqr.prg''
            geexec GaussProgram
        Case 8
            GaussProgram = ''c:\engine\prg\rank.prg''
            geexec GaussProgram
        Case 9
            GaussProgram = ''c:\engine\prg\schur.prg''
            geexec GaussProgram
        Case 10
            GaussProgram = ''c:\engine\prg\svd.prg''
            geexec GaussProgram
    End Select

    cmdDisplay.Enabled = True

End Sub
-------------------------------------------
Private Sub cmdExit_Click()
    geclose
    End
End Sub
-------------------------------------------
Private Sub cmdData_Click()
    Dim N As Integer
    Dim i, j As Integer
    Dim Prompt As String
    Dim Mij As String
```

```
    N = Val(txtDimension.Text)

    For i = 1 To N Step 1
        grdMatrix.Row = i - 1
        For j = 1 To N Step 1
            Prompt = ''Row #'' + str(i) + '' - Col #'' + str(j)
            Mij = InputBox(Prompt, ''Enter Data'', '''', 500, 5000)
            grdMatrix.Col = j - 1
            grdMatrix.Text = Mij
        Next j
    Next i

    cmdExecute.Enabled = True

End Sub
-------------------------------------------
Private Sub Form_Load()
    cboLinearAlgebra.AddItem ''BALANCE - Balances a matrix''
    cboLinearAlgebra.AddItem ''COND - Computes condition number''
    cboLinearAlgebra.AddItem ''CROUT - Computes Crout Decomposition''
    cboLinearAlgebra.AddItem ''HESS - Computes upper Hessenberg form''
    cboLinearAlgebra.AddItem ''LU - Computes LU Decomposition with row pivoting''
    cboLinearAlgebra.AddItem ''NULL - Computes orthonormal basis for right null space''
    cboLinearAlgebra.AddItem ''PINV - Computes Moore-Penrose pseudo-inverse''
    cboLinearAlgebra.AddItem ''QQR - Computes QR decomposition''
    cboLinearAlgebra.AddItem ''RANK - Computes rank of a matrix''
    cboLinearAlgebra.AddItem ''SCHUR - Computes Schur decomposition''
    cboLinearAlgebra.AddItem ''SVD - Computes the singular values''
    cboLinearAlgebra.ListIndex = 0

    cmdDisplay.Enabled = False
    cmdExecute.Enabled = False
    geopen
End Sub
-------------------------------------------
Private Sub txtDimension_Change()
    grdMatrix.Clear
    cmdExecute.Enabled = False
    grdMatrix.cols = Val(txtDimension.Text)
    grdMatrix.rows = Val(txtDimension.Text)
End Sub
```

- The VB code of the second window is:

```
Private Sub cmdExit_Click()
    frmVB2.Hide
    frmVB1.Show
End Sub
-------------------------------------------
Public Sub cmdDisplay()
    Dim OutputMatrix() As Double
    Dim OutputString As String
    Dim OutputMatrixName As String
    Dim OutputStringName As String
    Dim txt As String

    Dim nr, nc As Integer
    Dim i, j As Integer

    txtDisplay.Text = ''''

    Select Case frmVB1.cboLinearAlgebra.ListIndex

        Case 0
            txt = ''Method: ''
            OutputStringName = ''method''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblMethod.Caption = txt

            txt = ''Syntax: ''
            OutputStringName = ''syntax''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblSyntax.Caption = txt

            txt = ''b = '' & Chr(13) & Chr(10)
            OutputMatrixName = ''b''
            gematget OutputMatrixName, OutputMatrix()
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
```

```
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i
        txt = txt & Chr(13) & Chr(10)
        txt = txt & ''z = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''z''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 1
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''c = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''c''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 2
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''L = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''l''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i
        txt = txt & Chr(13) & Chr(10)
        txt = txt & ''U = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''u''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 3
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt
```

```
        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''h = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''h''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i
        txt = txt & Chr(13) & Chr(10)
        txt = txt & ''z = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''z''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 4
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''l = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''l''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i
        txt = txt & Chr(13) & Chr(10)
        txt = txt & ''u = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''u''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 5
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''b = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''b''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
```

```
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 6
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''y = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''y''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 7
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''q1 = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''q1''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i
        txt = txt & Chr(13) & Chr(10)
        txt = txt & ''r = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''r''
        gematget OutputMatrixName, OutputMatrix()
        nr = UBound(OutputMatrix, 1)
        nc = UBound(OutputMatrix, 2)
        For i = 1 To nr Step 1
            For j = 1 To nc Step 1
                txt = txt & OutputMatrix(i, j) & ''   ''
            Next j
            txt = txt & Chr(13) & Chr(10)
        Next i

    Case 8
        txt = ''Method: ''
        OutputStringName = ''method''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblMethod.Caption = txt

        txt = ''Syntax: ''
        OutputStringName = ''syntax''
        gestrget OutputStringName, OutputString
        txt = txt & OutputString
        lblSyntax.Caption = txt

        txt = ''k = '' & Chr(13) & Chr(10)
        OutputMatrixName = ''k''
        gematget OutputMatrixName, OutputMatrix()
```

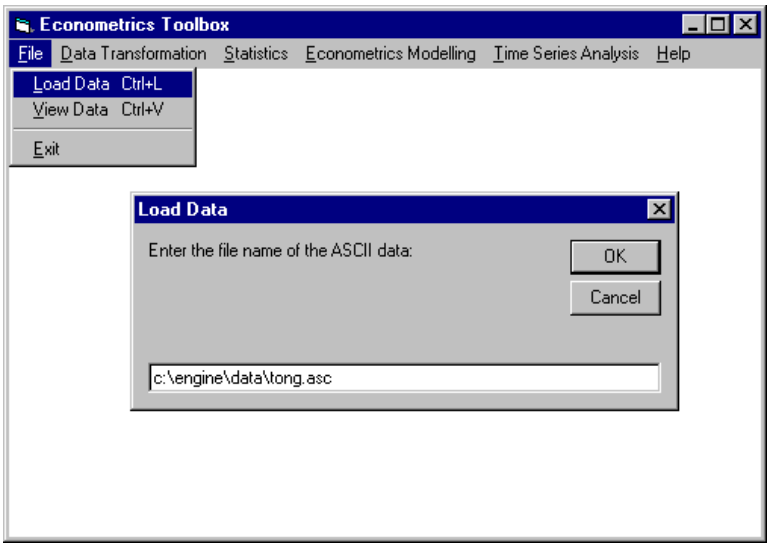
```
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
                Next j
                txt = txt & Chr(13) & Chr(10)
            Next i

        Case 9
            txt = ''Method: ''
            OutputStringName = ''method''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblMethod.Caption = txt

            txt = ''Syntax: ''
            OutputStringName = ''syntax''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblSyntax.Caption = txt

            txt = ''s = '' & Chr(13) & Chr(10)
            OutputMatrixName = ''s''
            gematget OutputMatrixName, OutputMatrix()
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
                Next j
                txt = txt & Chr(13) & Chr(10)
            Next i
            txt = txt & Chr(13) & Chr(10)
            txt = txt & ''z = '' & Chr(13) & Chr(10)
            OutputMatrixName = ''z''
            gematget OutputMatrixName, OutputMatrix()
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
                Next j
                txt = txt & Chr(13) & Chr(10)
            Next i

        Case 10
            txt = ''Method: ''
            OutputStringName = ''method''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblMethod.Caption = txt

            txt = ''Syntax: ''
            OutputStringName = ''syntax''
            gestrget OutputStringName, OutputString
            txt = txt & OutputString
            lblSyntax.Caption = txt

            txt = ''s = '' & Chr(13) & Chr(10)
            OutputMatrixName = ''s''
            gematget OutputMatrixName, OutputMatrix()
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
                Next j
                txt = txt & Chr(13) & Chr(10)
            Next i

    End Select

    txtDisplay.Text = txt
End Sub
```

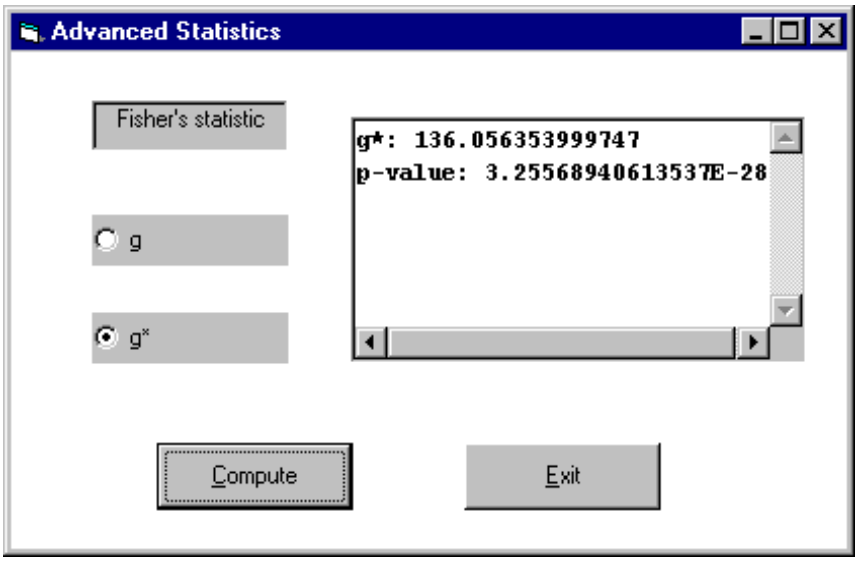
▶ The Gauss programs are:

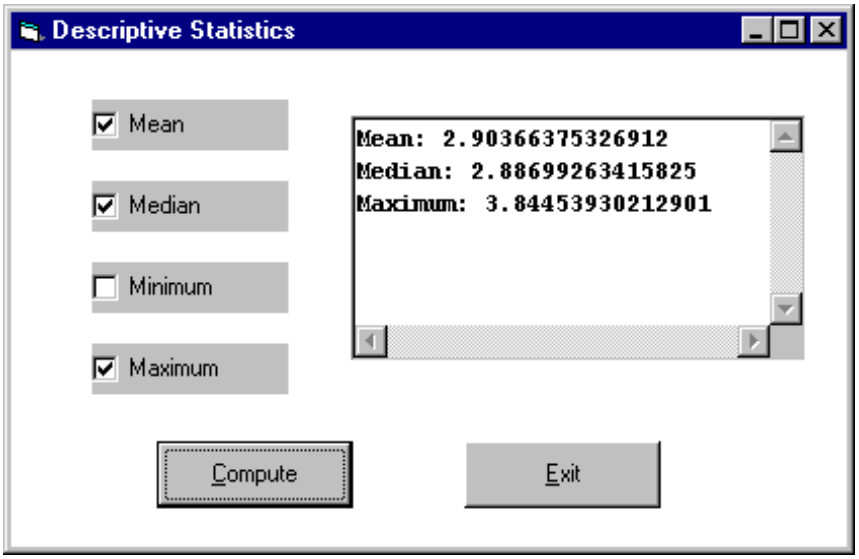
```
-----------------------------------------
-> BALANCE.PRG

method = ''BALANCE - Balances a matrix'';
syntax = ''{b,z} = BALANCE(x)'';
```

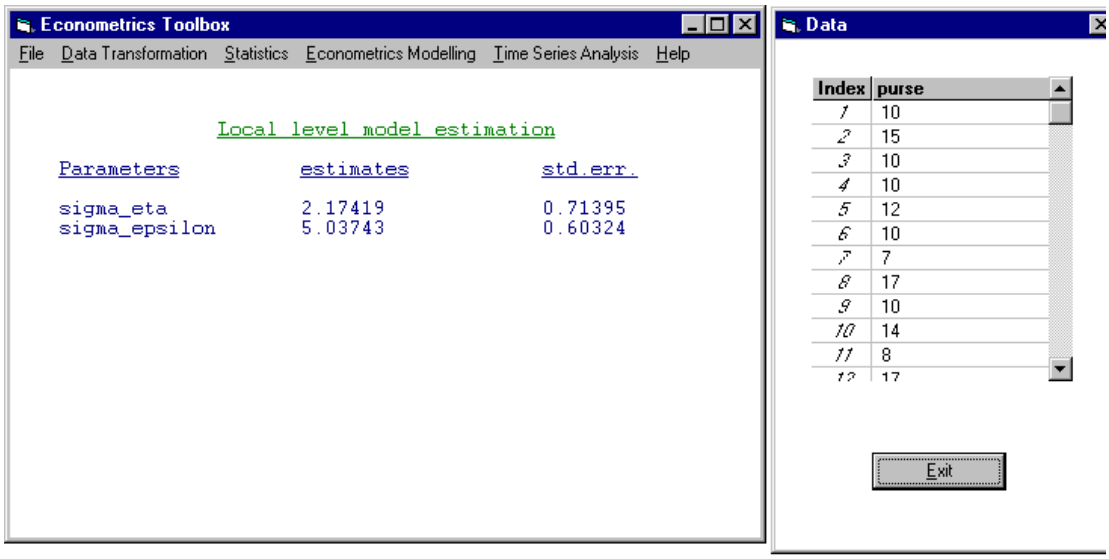
```
{b,z} = BALANCE(x);
-------------------------------------------
-> COND.PRG

method = ''COND - Computes condition number'';
syntax = ''c = COND(x)'';

c = COND(x);
-------------------------------------------
-> CROUT.PRG

method = ''CROUT - Computes Crout Decomposition'';
syntax = ''y = CROUT(x)'';

y = CROUT(x);
L = lowmat(y);
U = upmat1(y);
-------------------------------------------
-> HESS.PRG

method = ''HESS - Computes upper Hessenberg form'';
syntax = ''{h,z} = HESS(x)'';

{h,z} = HESS(x);
-------------------------------------------
-> LU.PRG

method = ''LU - Computes LU Decomposition with row pivoting'';
syntax = ''{l,u} = LU(x)'';

{l,u} = LU(x);
-------------------------------------------
-> NULL.PRG

method = ''NULL - Computes orthonormal basis for right null space'';
syntax = ''b = NULL(x)'';

b = NULL(x);
-------------------------------------------
-> PINV.PRG

method = ''PINV - Computes Moore-Penrose pseudo-inverse'';
syntax = ''y = PINV(x)'';

y = PINV(x);
-------------------------------------------
-> QQR.PRG

method = ''QQR - Computes QR decomposition'';
syntax = ''{q1,r} = QQR(x)'';

{q1,r} = QQR(x);
-------------------------------------------
-> RANK.PRG

method = ''RANK - Computes rank of a matrix'';
syntax = ''k = RANK(x)'';

k = RANK(x);
-------------------------------------------
-> SCHUR.PRG

method = ''SCHUR - Computes Schur decomposition'';
syntax = ''{s,z} = SCHUR(x)'';

{s,z} = SCHUR(x);
-------------------------------------------
-> SVD.PRG

method = ''SVD - Computes the singular values'';
syntax = ''s = SVD(x)'';

s = SVD(x);
```

4.2.3.3 Creating an Econometrics ToolBox








Econometrics Toolbox
File Data Transformation Statistics Econometrics Modelling Time Series Analysis Help
Local level model estimation
Parameters estimates std.err.
sigma_eta 2.17419 0.71395
sigma_epsilon 5.03743 0.60324
Data
Index purse
1 10
2 15
3 10
4 10
5 12
6 10
7 7
8 17
9 10
10 14
11 8
Exit