Financial Applications with Gauss

Financial Econometric Research Centre
City University Business School, London, 9 december 1998

Thierry Roncalli

Financial Econometric Research Centre
City University Business School
Frobisher Crescent
Barbican Centre
London

e-mail: t.roncalli@city.ac.uk
web: http://www.city.ac.uk/cubs/ferc/thierry

December 2, 1998

Contents

Chapter 1

Introduction to GAUSS

1.1 What is Gauss?

The Gauss Mathematical and Statistical System is a fast matrix programming language widely used by scientists, engineers, statisticians, biometricians, econometricians, and financial analysts. Designed for computationally intensive tasks, the Gauss system is ideally suited for the researcher who does not have the time required to develop programs in C or FORTRAN but finds that most statistical or mathematical "packages" are not flexible or powerful enough to perform complicated analysis or to work on large problems.

1.1.1 The Language

As a complete programming language, the Gauss system is both flexible and powerful. Immediately available to the Gauss user is a wide variety of statistical, mathematical and matrix handling routines. Gauss can be used in either command mode (interactively) or in edit mode. In command mode one-line commands, or small screen-resident programs, are issued and the results of calculations seen immediately. In edit mode you can write complex programs and store them in files. Gauss has over 400 functions built in, including LINPACK, EISPACK and BLAS routines. In addition you can add your own functions to this library. Powerful data handling capabilities including a data loop allow transformations in a data set by directly using variable names in expressions. This greatly simplifies data transformations and makes for shorter more readable programs. Gauss supports complex numbers. you do not have to keep track of the real and imaginary parts of a matrix. Complex numbers are handled automatically, greatly simplifying program development.

1.1.2 The Graphics

The Gauss system includes complete high resolution 2D and 3D Publication Quality Graphics. Praised by users and reviewers alike, Gauss graphics provide comprehensive data visualization capabilities. You can create XY plots, polar plots, bar charts, histograms, surface plots, XYZ plots; polar, log and box graphs. Graphs can be placed in individual overlapping or tiled windows on a single page. There is no limit to the number of graphs that you can combine, and no cutting or pasting is required.

1.2 What is new in the NT/95 version?

GAUSS is now available in 32-bit native versions for OS/2, Windows 95 and Windows NT. This new version of GAUSS runs in multiple moveable, resizeable windows, and you can:

1. Have multiple graphics open at one time.
2. Cut and paste text between GAUSS command and edit windows, and to and from other programs such as word processors and editors.
3. Run multiple sessions of GAUSS concurrently.
4. Run long GAUSS sessions in the background, freeing the computer for other tasks.

5. Pull-down menus simplify configuring fonts, window selection, file selection and other options.

- New Data Exchange Facility for Win NT/95
  GAUSS for Windows now includes a Data Excahange feature for exchanging data with most Windows spreadsheets and databases. GAUSS matricies and data sets can be exported in a variety of file formats. Files generated by other software can be imported as matricies or data sheets. File formats include: Excel, Lotus, Quatro, Symphony, Dbase, Paradox, FoxPro, Clipper and ASCII.

- Legacy DOS applications
  GAUSS for Windows can now run your DOS programs in a standard 80 x 25 window. Supports: Foreground/background color at the individual character level · ANSI.SYS escape code sequences for color, cursor, and screen control.

- Run and Edit file selection lists
  In the top portion of GAUSS's Command and Edit windows are two separate file selection boxes containing lists of the files you are currently running and the files you are editing. With one click you can bring a file in to the editor and with one click you can save it and rerun your main program. This simplification of the edit run cycle when combined with the new make facility makes development efficient, and easier than ever before. File names can be copied between these lists with one mouse click.

- "Make" facility
  GAUSS for OS/2 and Windows keeps track of which files have been changed. This eliminates the need for you to keep track of all your changes when editing and running complex programs using multiple files. This also can significantly speed up consecutive runs of complex programs because you no longer need to clear the workspace with the new command each time you make a change to a secondary file.

- Foreign Language Interface
  Using the Foreign Language Interface is much simpler in this new version. GAUSS for OS/2 and Windows supports any compiler that will generate .dll files. There are those times when you need a function that GAUSS doesn't have, but for which you already have Fortran or C code available. Incorporating FORTRAN or C in to GAUSS programs used to require "startup" code which is only available with certain compilers. Now all you need is a .dll (dynamically linked library) file, which most UNIX, Windows NT/95 and OS/2 compilers can generate.

- Sparse Matrix Routines
  Sparse solve and matrix operations in the OS/2 and Windows versions efficiently handle large sparse matrices. GAUSS's CPU storage and times for computing sparse solve and sparse matrix products are on the order of the number of nonzero elements, and therefore are far more efficient than their dense analogues for large matrices.

- Cumulative Distribution Functions
  Previously GAUSS users were limited to 3 dimensions with `cdftvn( )`. Now they are only limited by computational time and RAM. Functions are included for the following:

  1. `cdfmvn( )` multivariate Normal cdf
  2. `cdfn2( )` compute interval cdf: cdfn(x+dx) - cdfn(x). cdfn2( ) increases accuracy for a common use of `cdfn( )`. Further gains in accuracy are achieved when taking the log of an interval of the Normal cdf with the `lncdfn2( )` function below.
  3. `lncdfn( )` log of Normal cdf
  4. `lncdfnc( )` log of complement of Normal cdf
  5. `lncdfbvn( )` log of bivariate Normal cdf
  6. `lncdfmvn( )` log of multivariate Normal cdf
  7. `lncdfn2( )` log of Normal interval cdf.

- Other New Features

  1. New Random Number Generators

   (a) Gamma pseudo-random numbers
   (b) Poisson pseudo-random numbers
   (c) Negative binomial pseudo-random numbers
   (d) Beta pseudo-random numbers

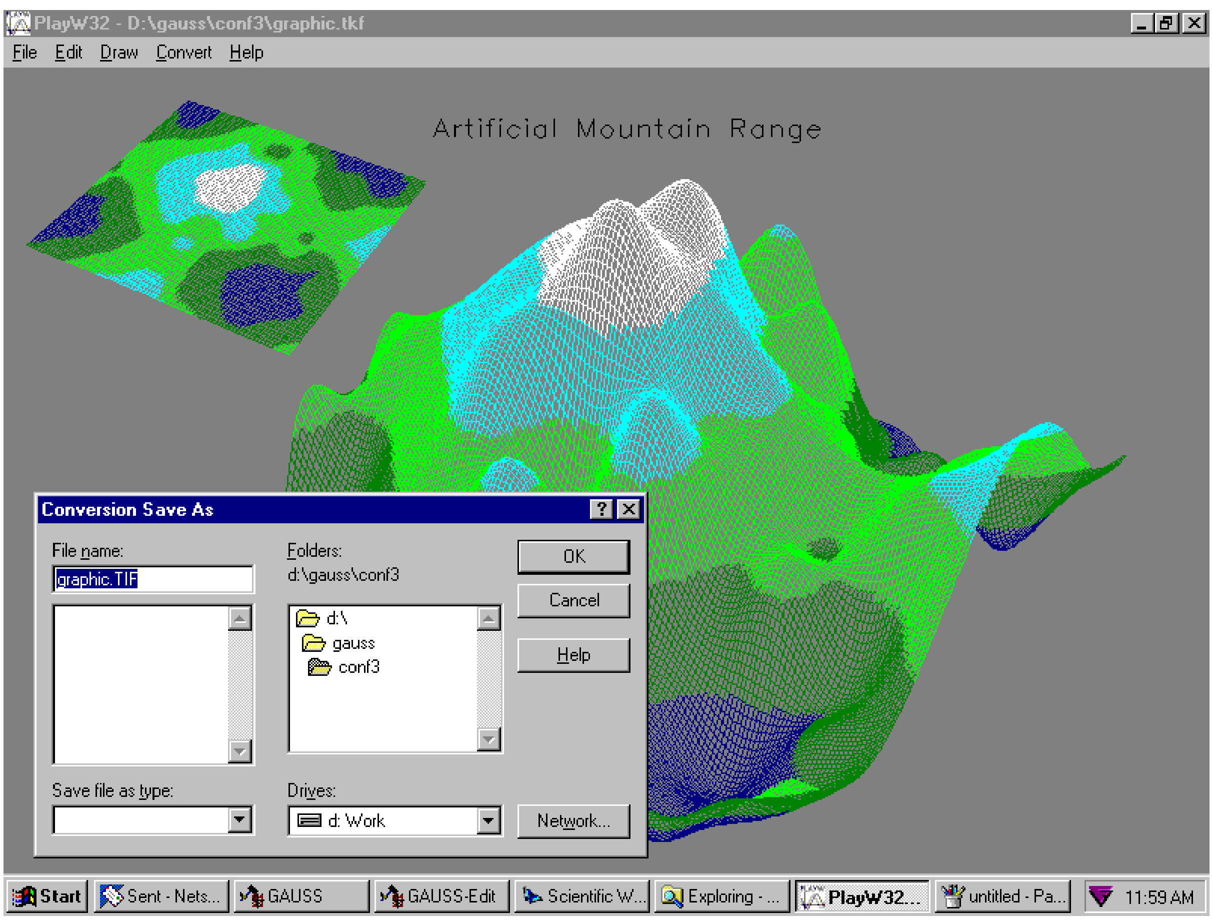
2. New Spline functions
   (a) 2-D spline interpolation under tension
   (b) 1-D smoothing spline under tension

3. New Loess Regression function computes robust locally weighted regression using the method of William S. Cleveland ("Robust Locally Weighted Regression and Smoothing Scatterplots", JASA, 74:829-836.

4. Bitwise arithmetic
   (a) `radix(x,b)` Convert x from decimal to base b
   (b) `radixi(x,b)` Convert x from base b to decimal
   (c) `iand(a,b)` Bitwise a and b
   (d) `ior(a,b)` Bitwise a or b
   (e) `ieqv(a,b)` Bitwise a eqv b
   (f) `ixor(a,b)` Bitwise a xor b
   (g) `inot(a)` Bitwise complement a
   (h) `ishft(a,s)` Shift bits s positions (+ve or -ve)

1.2.1 PlayW

PlayW is a new graphic utility for converting Gauss *.tkf* graphics files to WMF, BMP, TIFF and CMYK PostScript (also includes clipboard support for pasting graphics into other applications, and various new color and formatting options).



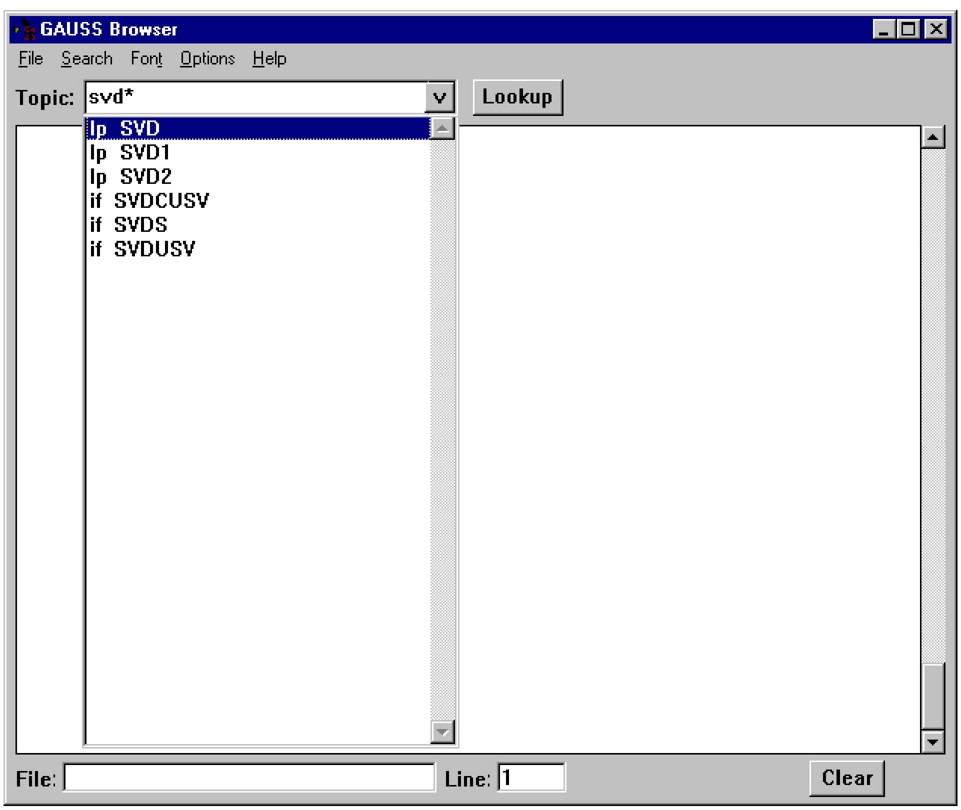
1.2.2 On-line help

There are a number of ways to specify the pattern to be used in a search:

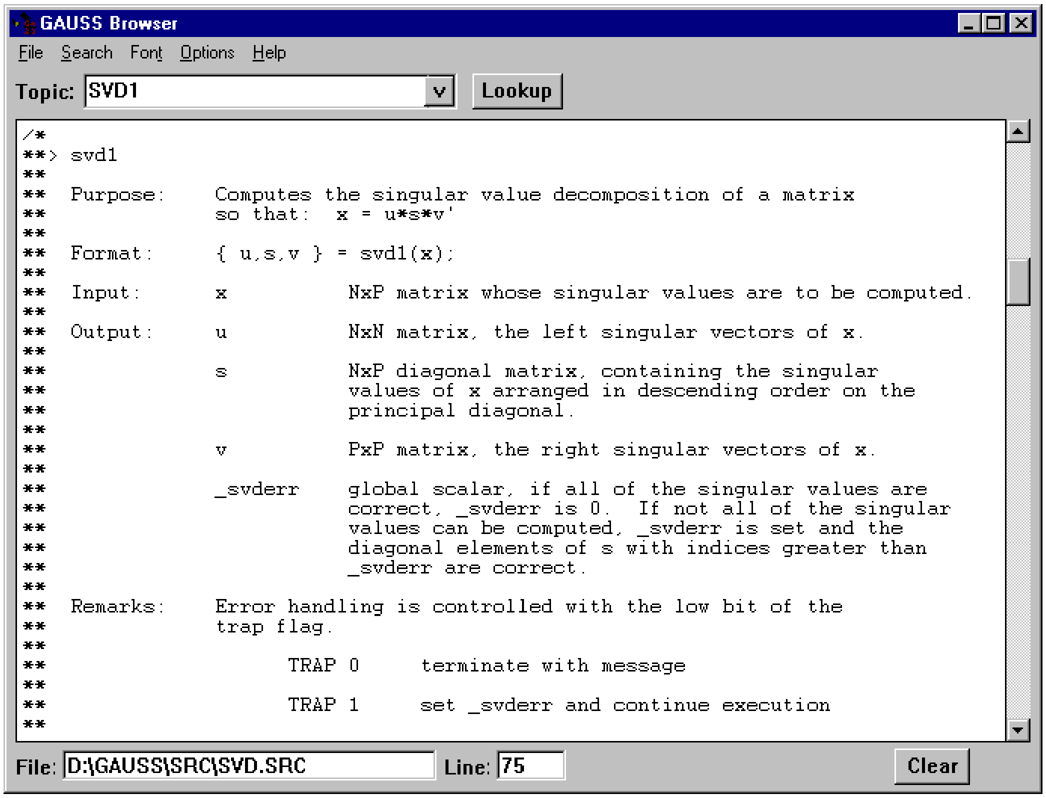
1. Select (mark) text in the parent window before opening the browser.
2. Enter or modify text in the Topics box at the top of the window.
3. Select from the Topics box drop-down list of recently successful searches.
4. Select (mark) text in the file currently being displayed.
5. Select from the drop-down list of possibilities generated by a wildcard search.

Search patterns may contain the wildcards "*" or "?". "*" represents any number and combination of characters. "?" represents any single character. To execute a search, click the Lookup button or press Enter. In drop-down lists you can also double-click an entry. You can also search for files, by entering the name in the Topics box or File box, or by using the File Open dialog box (menu File|Open...). File searches, however, may not contain wildcards. If you include a path, the browser will look only in the indicated directory for the file. If you enter just the file name, the browser will search the Gauss source path (src_path) for it. When a unique pattern is found and it represents a library symbol, the file containing it is loaded and scrolled to the topic. If the unique pattern is an intrinsic, the Gauss help system is opened to the selected subject. If a pattern search results in more than one match, the matches are displayed in a drop-down list. For example: searching with a pattern of "*n" results in a list containing all the intrinsic operators, keywords and functions ending in "n" as well as all library functions, keywords, procedures, matrices, and strings ending in "n".

▶ For example, if we specify `svd*`, we obtain



▶ With `svd1`, we have



1.3 Value at Risk examples

1.3.1 Asset portfolio

- This example shows how to compute the *VaR* of an assets portfolio in Gauss. It is done very easily with the inverse Gaussian function `cdfni`.

```
new;

let MU = .1 .2 .3;
let SIGMA[3,3] =  .04  .02 -.01
                  .02  .06  .02
                 -.01  .02  .03;
theta = 1|3|-1;
Portfolio_t0 = 4;

MeanP = theta'MU - Portfolio_t0;
VarP = theta'SIGMA*theta;

alpha = 0.05;
VaR = abs(MeanP + cdfni(alpha)*sqrt(VarP));

output file = var1.out reset;

print ftos(VaR,''Value at Risk (0.05) = %lf Francs'',5,4);

output off;


Value at Risk (0.05) = 4.9056 Francs
```

1.3.2 Simulating methods

- This is the same problem as above, but the *VaR* is computed with the Monte Carlo method.

```
new;

let MU = .1 .2 .3;
let SIGMA[3,3] =  .04  .02 -.01
                  .02  .06  .02
                 -.01  .02  .03;
```

```
theta = 1|3|-1;
Portfolio_t0 = 4;

Nobs = 2500;
Portfolio_t1 = rndmn(MU,SIGMA,Nobs)*theta;

deltaP = Portfolio_t1 - Portfolio_t0;

output file = var2.out reset;

e = {0.01, 0.025, 0.05, 0.5, 0.95, 0.975, 0.99};

q = quantile(deltaP,e);
name = ''1%'' | ''2.5%'' | ''5%'' | ''Median'' | ''95%'' | ''97.5%'' | ''99%'';

print; print;
print ''Estimated Quantiles'';
print ''-----------------'';
call printfmt(name~q,0~1);

alpha = 0.05;
VaR = abs(quantile(deltaP,alpha));
print;
print ftos(VaR,''Value at Risk (0.05) = %lf Francs'',5,4);

output off;


proc (1) = rndmn(mu,SIGMA,Ns);
  local dim,u,Pchol;
  local oldtrap;

  dim = rows(mu);

  oldtrap = trapchk(1);
  trap 1,1;
  Pchol = chol(SIGMA)';
  trap oldtrap,1;
  if scalerr(Pchol);
    ERRORLOG ''error: SIGMA is not a positive definite matrix.'';
    retp(error(0));
  endif;

  u = mu + Pchol*rndn(dim,Ns);

  retp(u');
endp;
```

```
Estimated Quantiles
-----------------
      1%       -5.4215844
    2.5%       -5.1541676
      5%       -4.9111995
  Median       -3.6080266
     95%        -2.330682
   97.5%       -2.0806537
     99%       -1.7853267

Value at Risk (0.05) = 4.9112 Francs
```

1.3.3 *VaR* on options

- An example with derivatives.

```
new;
library option,pgraph;

S0 = 100; K = 100; sigma = 0.20; tau = 90/365; r = 0.08; b = r;

deltaTau = -7/365;  /* 7 days */

proc muProc(t,S);
  retp( b*S );
endp;

proc sigmaProc(t,S);
  retp( sigma*S );
endp;

C0 = EuropeanBS(S0,K,sigma,tau,b,r);

Ns = 2000;
```

```
{t,S} = Simulate_SDE(S0,&muProc,&sigmaProc,0,abs(deltaTau),20,Ns);

S1 = S[20,.]';
C1 = EuropeanBS(S1,K,sigma,tau,b,r);

deltaC = C1 - C0;

alpha = 0.05;

output file = var3.out reset;

/* Long position*/
deltaC = C1 - C0;
VaR = abs(quantile(deltaC,alpha));

print ''Long Position'';
print ''---------------'';
print ftos(VaR,''Value at Risk (0.05) = %lf Francs'',5,4);
print;

/* Short Position */

deltaC = - (C1 - C0);
alpha = 0.05;
VaR = abs(quantile(deltaC,alpha));

print ''Short Position'';
print ''---------------'';
print ftos(VaR,''Value at Risk (0.05) = %lf Francs'',5,4);
print;

/* Long Position --- DELTA-GAMMA approach */

deltaS = S1 - S0;
DELTA = EuropeanBS_Delta(S0,K,sigma,tau,b,r);
GAMMA_ = EuropeanBS_Gamma(S0,K,sigma,tau,b,r);
deltaC = deltaS * DELTA + 0.5 * (deltaS^2) * GAMMA_;
VaR = abs(quantile(deltaC,alpha));

print ''Long Position --- DELTA-GAMMA approach'';
print ''--------------------------------------'';
print ftos(VaR,''Value at Risk (0.05) = %lf Francs'',5,4);

output off;
```

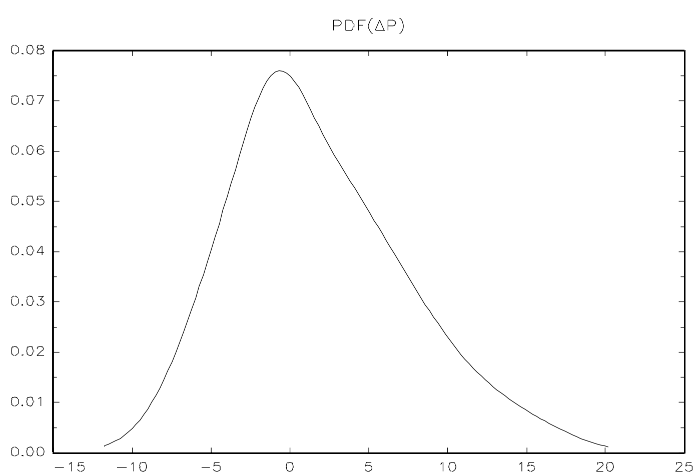
```
Long Position
---------------
Value at Risk (0.05) = 2.1836 Francs

Short Position
---------------
Value at Risk (0.05) = 3.1924 Francs

Long Position --- DELTA-GAMMA approach
--------------------------------------
Value at Risk (0.05) = 2.1961 Francs
```

1.3.4 Derivatives portfolio

▶ An example with derivatives portfolio.

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

rndseed 123;

S0 = 100; K = 100; sigma = 0.10; r = 0.08; b = r; N = 4;

deltaTau = -7/365;  /* 7 days */

theta = 6 | -2 ;

/* Call American */
tau1 = 90/365;

/* Put Look-back */
tau2 = 160/365;


proc muProc(t,S);
  retp( b*S );
endp;
```

```
proc sigmaProc(t,S);
  retp( sigma*S );
endp;

/* t0 */

_type_Option = ''call'';
Cam = AmericanCRR(S0,K,sigma,tau1,b,r,N);

_type_Option = ''put'';
PutLookBack = LookBackCRR(S0,sigma,tau2,b,r,N);

Portfolio_t0 = theta[1]*Cam + theta[2]*PutLookBack;

/* t1 */

Ns = 1000;
{t,S} = Simulate_SDE(S0,&muProc,&sigmaProc,0,abs(deltaTau),20,Ns);
S1 = S[20,.]';

_type_Option = ''call'';
Cam = AmericanCRR(S1,K,sigma,tau1,b,r,N);

_type_Option = ''put'';
PutLookBack = LookBackCRR(S1,sigma,tau2,b,r,N);

Portfolio_t1 = theta[1]*Cam + theta[2]*PutLookBack;


deltaP = Portfolio_t1 - Portfolio_t0;

{x,d,f,retcode} = Kernel(deltaP);

graphset;
  _pdate = ''''; _pnum = 2;
  fonts(''simplex simgrma'');
  title(''\201PDF(\202D\201P)'');
  graphprt(''-c=1 -cf=var4.eps'');
  xy(x,d);
```

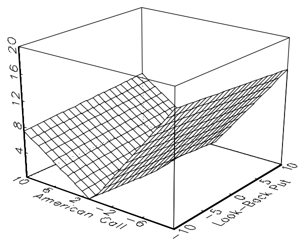


1.3.5 Non-linearity in *VaR*

▶ An illustration of non-linearity in *VaR* with derivatives portfolio.

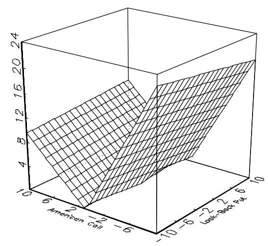
```
new;
library option,tsm,optmum,pgraph;

rndseed 123;

S0 = 100; K = 100; sigma = 0.10; r = 0.08; b = r; N = 4;

deltaTau = -7/365;  /* 7 days */

/* Call American */
tau1 = 90/365;
```

```
/* Put Look-back */
tau2 = 120/365;


proc muProc(t,S);
  retp( b*S );
endp;

proc sigmaProc(t,S);
  retp( sigma*S );
endp;

/* t0 */

_type_Option = ''call'';
Cam0 = AmericanCRR(S0,K,sigma,tau1,b,r,N);

_type_Option = ''put'';
PutLookBack0 = LookBackCRR(S0,sigma,tau2,b,r,N);


/* t1 */

Ns = 1000;
{t,S} = Simulate_SDE(S0,&muProc,&sigmaProc,0,abs(deltaTau),20,Ns);
S1 = S[20,.]';

_type_Option = ''call'';
Cam = AmericanCRR(S1,K,sigma,tau1,b,r,N);

_type_Option = ''put'';
PutLookBack = LookBackCRR(S1,sigma,tau2,b,r,N);

theta2 = seqa(-10,1,21)';
alpha = 0.10|0.05;
VaR1 = {};
VaR2 = {};

i = -10;
do until i > 10;
  Portfolio_t0 = i*Cam0 + theta2 .* PutLookBack0;
  Portfolio_t1 = i*Cam + theta2 .* PutLookBack;
  deltaP = Portfolio_t1 - Portfolio_t0;
  VaR = abs(quantile(deltaP,alpha));
  VaR1 = VaR1 | VaR[1,.];
  VaR2 = VaR2 | VaR[2,.];
  i = i + 1;
endo;


graphset;
  begwind;
  window(2,2,0);
    _pdate = ''''; _pnum = 2; _pnumht = 0.25; _paxht = 0.25;

  setwind(1);

    xtics(-10,10,5,0);
    xlabel(''Look-Back Put'');
    ylabel(''American Call'');
    surface(theta2,theta2',VaR1);
  setwind(2);
    graphset;
    _paxes = 0; _pframe = 0; _pnum = 0; _ptitlht = 0.50;
    title(''\L\L<== VaR at 10%'');
    draw;
  setwind(3);
    _paxes = 0; _pframe = 0;
    title(''\L\LVaR at 5% ==>'');
    draw;
  setwind(4);
    graphset;
    _pnum = 2; _pnumht = 0.25;
    xlabel(''Look-Back Put'');
    ylabel(''American Call'');

    surface(theta2,theta2',VaR2);

  graphprt(''-c=1 -cf=var5.eps'');

  endwind;
```



<== VaR at 10%

VaR at 5% ==>



1.4 Creating DLLs for Gauss

Dynamically-linked libraries (DLL's, also known as shared libraries or shared objects) can be linked at run-time with Gauss, making the functions contained in them available to Gauss programs. DLL's are linked in with `dlibrary`, and the functions contained in them are called using `dllcall`.

1.4.1 C/C++ source code

1.4.1.1 lcc compiler (LCC-WIN32)

We consider a very simple example with the free C compiler LCC-WIN32 available at this following adress:

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

The procedure `TDGsolve` is taken from the **Mlib** library available at the url:

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

This procedure could be used to solve the tridiagonal system

$$[a; b; c]\, x = d$$

```
proc (1) = TDGsolve(a,b,c,d);
  local N,x,bprime,dprime,i,w;

  N = rows(b);

  x = zeros(N,1); bprime = zeros(N,1); dprime = zeros(N,1);
  bprime[N] = b[N]; dprime[N] = d[N];

  i = N - 1;
  do while i > 0;
    w = c[i]/bprime[i+1];
    bprime[i] = b[i] - w*a[i+1];
    dprime[i] = d[i] - w*dprime[i+1];
    i = i - 1;
  endo;

  x[1] = dprime[1]/bprime[1];

  i = 2;
  do until i > N;
    x[i] = (dprime[i]-a[i]*x[i-1])/bprime[i];
    i = i + 1;
  endo;

  retp(x);
endp;
```

▶ The corresponding C subroutine is[1]

```
/* --- The following code comes from c:\lcc\lib\wizard\dll.tpl. */

#include <string.h>
#include <windows.h>
#include <stdio.h>

//
//  FUNCTION: DLLMain(HINSTANCE, DWORD, LPVOID)
//  PURPOSE:  Called when DLL is loaded by a process, and when new
//    threads are created by a process that has already loaded the
//    DLL.  Also called when threads of a process that has loaded the
//    DLL exit cleanly and when the process itself unloads the DLL.
//  PARAMETERS:
//    hDLLInst    - Instance handle of the DLL
//    fdwReason   - Process attach/detach or thread attach/detach
//    lpvReserved - Reserved and not used
//  RETURN VALUE:  (Used only when fdwReason == DLL_PROCESS_ATTACH)
//    TRUE  -  Used to signify that the DLL should remain loaded.
//    FALSE -  Used to signify that the DLL should be immediately unloaded.
//

BOOL WINAPI LibMain(HINSTANCE hDLLInst, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH:
            // The DLL is being loaded for the first time by a given process.
            // Perform per-process initialization here.  If the initialization
            // is successful, return TRUE; if unsuccessful, return FALSE.


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


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

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

            break;
    }
    return TRUE;
}

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

 N = (int) *n;

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

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

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

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

[1]The file *\lcc\lib\wizard\dll.tpl* is added to the C file, because the file *.dll* needs an entry point to load/unload DLLs.

```
 return 0;
}
```

Remark 1 *Arguments are passed to* `dllTDGSolve` *by reference.*

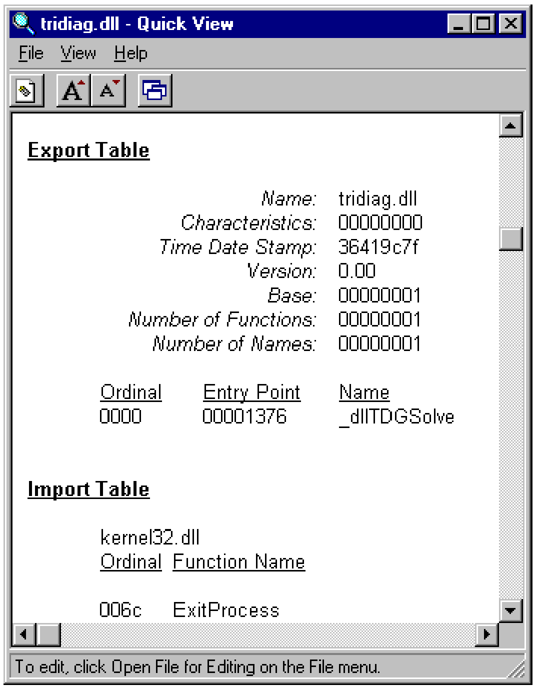
- We create the *tridiag.dll* file with the following commands

  lcc tridiag.c
  lcclnk -dll tridiag.obj

- The *tridiag.exp* is an ascii file containing all the exported names. We obtain

  ```
  tridiag.dll
  _dllTDGSolve   _dllTDGSolve
  ```

  We could also "view" the DLL file with QuickView:



- To use the `_dllTDGSolve` in Gauss, we declare the *tridiag.dll* file with `dlibrary`. **Because the arguments are passed by reference, it is very important to declare them correctly before calling the subroutine with the `dllcall` command.**

```
new;

library mlib;
dlibrary tridiag.dll;

N = 5;

a = rndn(N,1); b = rndn(N,1); c = rndn(N,1);
abc = TDGmatrix_to_Dmatrix(a,b,c);
d = abc*seqa(1,1,N);

x1 = TDGsolve(a,b,c,d);

x2 = zeros(N,1); bprime = zeros(N,1); dprime = zeros(N,1);

dllcall _dllTDGSolve(a,b,c,d,N,x2,bprime,dprime);

output file = tridiag1.out reset;

print x1~x2;

output off;
```

```
      1.0000000        1.0000000
      2.0000000        2.0000000
      3.0000000        3.0000000
      4.0000000        4.0000000
      5.0000000        5.0000000
```

▶ A better solution is to create a Gauss procedure:

```
/*
**> dllTDGSolve
*/


proc dllTDGSolve(a,b,c,d);
  local N,x,bprime,dprime;

  N = rows(d);
  x = zeros(N,1);
  bprime = zeros(N,1);
  dprime = zeros(N,1);

  dllcall _dllTDGSolve(a,b,c,d,N,x,bprime,dprime);

  retp(x);
endp;
```

▶ An example of using the Gauss `dllTDGsolve` procedure.

```
new;

library mlib,tridiag;
dlibrary tridiag.dll;

N = 5;

a = rndn(N,1); b = rndn(N,1); c = rndn(N,1);
abc = TDGmatrix_to_Dmatrix(a,b,c);
d = abc*seqa(1,1,N);

x = dllTDGsolve(a,b,c,d);
```

1.4.1.2 gcc/g++ compilers (GNU-WIN32)

▶ Not yet done.

1.4.1.3 gcc compiler (RSX NT)

▶ Not yet done.

1.4.2 Fortran code

1.4.2.1 dvf/fps compilers

We could also combine Fortran and Gauss. The following url contains examples with Microsoft FORTRAN Powerstation 4.0 (MSFP4) and Digital Visual Fortran 5.0 Professional Edition (DVF5):

http://www.hhs.se/personal/Psoderlind/Software/Software.htm

In this case, we have access to the Fortran code archived in Netlib

http://www.netlib.org/

We could also use these routines by translating them into C with the free *f2c* software. You will find an example at the following adress:

http://weber.u.washington.edu/~rons/fli.html

1.4.2.2 g77 compiler (RSX NT)

▶ Not yet done.

1.5 Win32 APIs access

By using DLLs, we have access to the Windows APIs in Gauss. We could for example create directories with the *kernel32.dll* CreateDirectory function, delete/move files with the *kernel32.dll* DeleteFile/MoveFile function, scroll windows (ScrollWindow) or play sounds with the *winmm.dll* PlaySound function.

1.5.1 The C code

```
/* --- The following code comes from c:\lcc\lib\wizard\dll.tpl. */

#include <string.h>
#include <windows.h>
#include <stdio.h>

//
//  FUNCTION: DLLMain(HINSTANCE, DWORD, LPVOID)
//  PURPOSE:  Called when DLL is loaded by a process, and when new
//    threads are created by a process that has already loaded the
//    DLL.  Also called when threads of a process that has loaded the
//    DLL exit cleanly and when the process itself unloads the DLL.
//  PARAMETERS:
//    hDLLInst    - Instance handle of the DLL
//    fdwReason   - Process attach/detach or thread attach/detach
//    lpvReserved - Reserved and not used
//  RETURN VALUE:  (Used only when fdwReason == DLL_PROCESS_ATTACH)
//    TRUE  -  Used to signify that the DLL should remain loaded.
//    FALSE -  Used to signify that the DLL should be immediately unloaded.
//

BOOL WINAPI 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 dllDestroyWindow(char *lpWindowName)
{
 char _lpWindowName[256];
 long *hwnd;
 strcpy(_lpWindowName,lpWindowName);
 hwnd = FindWindow(0,_lpWindowName);
 DestroyWindow(hwnd);
 return 0;
}

__declspec(dllexport) int dllFlashWindow(char *lpWindowName,double *bInvert)
{
 char _lpWindowName[256];
 long *hwnd;
 long _bInvert;
 strcpy(_lpWindowName,lpWindowName);
 hwnd = FindWindow(0,_lpWindowName);
 _bInvert = (long) *bInvert;
```

```
 FlashWindow(hwnd,_bInvert);
 return 0;
}

__declspec(dllexport) int dllMessageBox(double *ret,char *lpText,char *lpCaption,double *wType)
{
 char _lpText[255];
 char _lpCaption[255];
 strcpy(_lpText,lpText);
 strcpy(_lpCaption,lpCaption);
 *ret = MessageBox(NULL,_lpText,_lpCaption,*wType);
 return 0;
}

__declspec(dllexport) int dllMoveWindow(char *lpWindowName,double *x,double *y,double *nWidth,
                                        double *nHeight,double *bRepaint)
{
 char _lpWindowName[256];
 long *hwnd;
 int _x;
 int _y;
 int _nWidth;
 int _nHeight;
 int _bRepaint;
 strcpy(_lpWindowName,lpWindowName);
 hwnd = FindWindow(0,_lpWindowName);
 _x = (int) *x;
 _y = (int) *y;
 _nWidth = (int) *nWidth;
 _nHeight = (int) *nHeight;
 _bRepaint = (int) *bRepaint;
 MoveWindow(hwnd,_x,_y,_nWidth,_nHeight,_bRepaint);
 return 0;
}


__declspec(dllexport) int dllShowWindow(char *lpWindowName,double *nCmdShow)
{
 char _lpWindowName[256];
 long *hwnd;
 long _nCmdShow;
 strcpy(_lpWindowName,lpWindowName);
 hwnd = FindWindow(0,_lpWindowName);
 _nCmdShow = (long) *nCmdShow;
 ShowWindow(hwnd,_nCmdShow);
 return 0;
}
```

1.5.2 The Gauss win32api.src file

```
/*
**> FlashWindow
*/

proc (0) = FlashWindow(lpWindowName,bInvert);

  dllcall _dllFlashWindow(lpWindowName,bInvert);

  retp;
endp;


/*
**> MessageBox
*/

proc (1) = MessageBox(lpText,lpCaption,wtype);
  local ret;

  ret = 0;

  dllcall _dllMessageBox(ret,lpText,lpCaption,wType);

  retp(ret);
endp;

/*
**> MoveWindow
*/

proc (0) = MoveWindow(lpWindowName,x,y,nWidth,nHeight,bRepaint);

  dllcall _dllMoveWindow(lpWindowName,x,y,nWidth,nHeight,bRepaint);
```

```
  retp;
endp;

/*
**> ShowWindow
*/

proc (0) = ShowWindow(lpWindowName,nCmdShow);

  dllcall _dllShowWindow(lpWindowName,nCmdShow);

  retp;
endp;
```

1.5.3 An example

```
new;

dlibrary win32api;
library win32api;

ret = MessageBox(''Do you want to quit Gauss ?'',''GAUSS MessageBox 1'',1);

if ret == 1;
  call MessageBox(''Sorry, but the program will continue'',''GAUSS MessageBox 2'',0);
else;
  call MessageBox(''Ok, the program will continue'',''GAUSS MessageBox 2'',0);
endif;

call MessageBox(''MoveWindow Example'',''GAUSS MessageBox 3'',3);

call MoveWindow(''Gauss'',10,10,500,500,1);

call pause(1);

call MoveWindow(''Gauss'',10,10,900,500,1);

call pause(1);

call ShowWindow(''Gauss'',7);
call ShowWindow(''Gauss-Edit'',7);

call FlashWindow(''Gauss'',0);
```

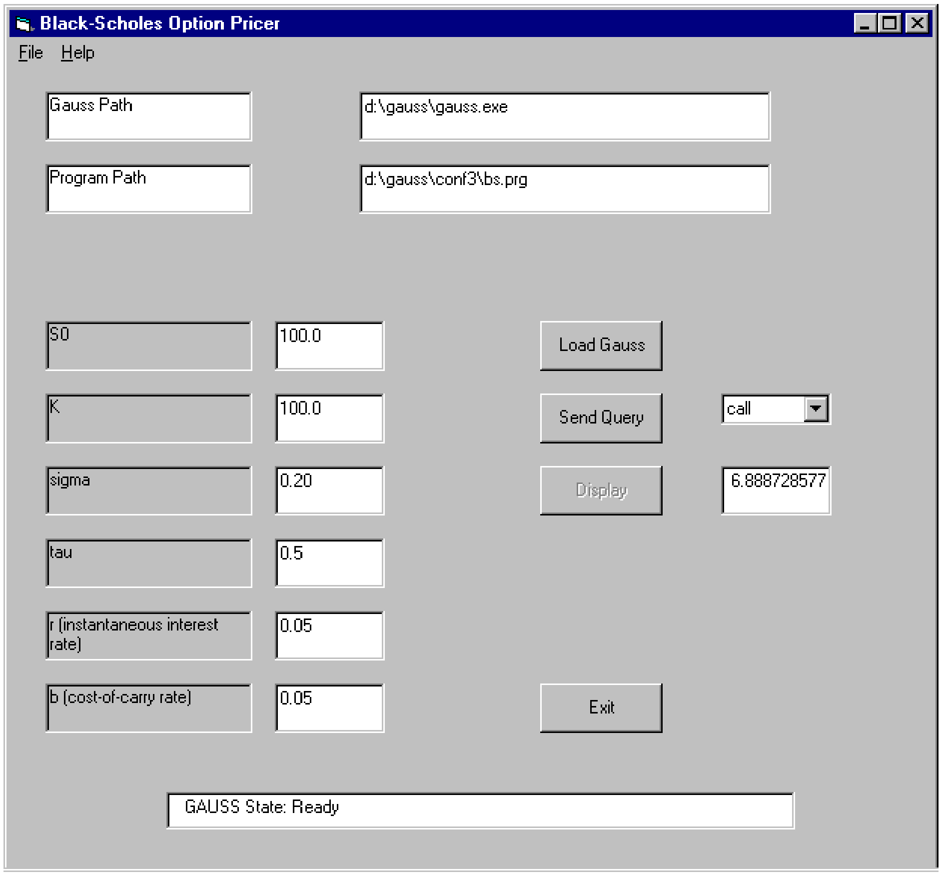
1.6 DDE functionnality

1.6.1 Using Mercury and Visual Basic

Gauss users will find some explanations about the following example at the following url:

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

This is a Visual Basic application for computing Black-Scholes option prices. It uses the VB module **Mercury** to allows communication with Gauss.



- The information are sent to Gauss with the **`send_value`** command. The cmdDisplay_Click() retrieves the option price using the **`get_value`** command.

```
Private Sub cmdLoadGauss_Click()
    Dim GaussPath As String
    Dim FileName As String
    Dim cmdstng As String
    Dim hwind As Long

    GaussPath = txtGaussPath.Text
    FileName = txtProgramPath.Text
    cmdstng = GaussPath & '' '' & FileName
    Call gauss_load(cmdstng)

    hwind = winhandle(''GAUSS'')
    If hwind > 0 Then
        cmdSendQuery.Enabled = True
        mnuSendQuery.Enabled = True
    End If
End Sub
---------------------------------------------
Private Sub cmdSendQuery_Click()
    Call send_value(''BS_S0'', Val(txtS0.Text))
    Call send_value(''BS_K'', Val(txtK.Text))
    Call send_value(''BS_sigma'', Val(txtSigma.Text))
    Call send_value(''BS_tau'', Val(txtTau.Text))
    Call send_value(''BS_r'', Val(txtR.Text))
    Call send_value(''BS_b'', Val(txtB.Text))
    Call send_string(''BS_type'', cboTypeOption.Text)
    Call gauss_run
    cmdDisplay.Enabled = True
    mnuDisplay.Enabled = True
End Sub
---------------------------------------------
Private Sub cmdDisplay_Click()
    Dim premium As Double
    premium = get_value(''BS_premium'')
    txtPrice.Text = Str$(premium)
    cmdDisplay.Enabled = False
    mnuDisplay.Enabled = False
End Sub
---------------------------------------------
```

```
Private Sub cmdExit_Click()
    Call gauss_unload
    End
End Sub
---------------------------------------------
Private Sub Form_Load()
   cboTypeOption.AddItem ''call''
   cboTypeOption.AddItem ''put''
   cboTypeOption.ListIndex = 0
   cmdSendQuery.Enabled = False
   cmdDisplay.Enabled = False
   mnuSendQuery.Enabled = False
   mnuDisplay.Enabled = False
End Sub
---------------------------------------------
Private Sub mnuAbout_Click()
    About.Show
End Sub
---------------------------------------------
Private Sub mnuDisplay_Click()
    cmdDisplay = True
End Sub
---------------------------------------------
Private Sub mnuExit_Click()
    cmdExit = True
End Sub
---------------------------------------------
Private Sub mnuLoadGauss_Click()
    cmdLoadGauss = True
End Sub
---------------------------------------------
Private Sub mnuSendQuery_Click()
    cmdSendQuery = True
End Sub
---------------------------------------------
Private Sub mnuTheModel_Click()
    TheModel.Show
End Sub
---------------------------------------------
Private Sub tmrStatus_Timer()
    Dim hwind As Long
    Dim gaussState As String
    hwind = winhandle(''GAUSS'')
    If hwind = 0 Then
        txtStatus.Text = ''   GAUSS Status: Not Loaded''
    Else
        gaussState = get_state()
        txtStatus.Text = ''   GAUSS State: '' & gaussState
    End If
End Sub
---------------------------------------------
Private Sub TmrHideGauss_Timer()
    Dim hwind As Long
    hwind = winhandle(''GAUSS'')
    If hwind > 0 Then
        Call activewindow(''GAUSS'', 7)
        tmrHideGauss.Enabled = False
    End If
End Sub
```

▶ The Gauss program is:

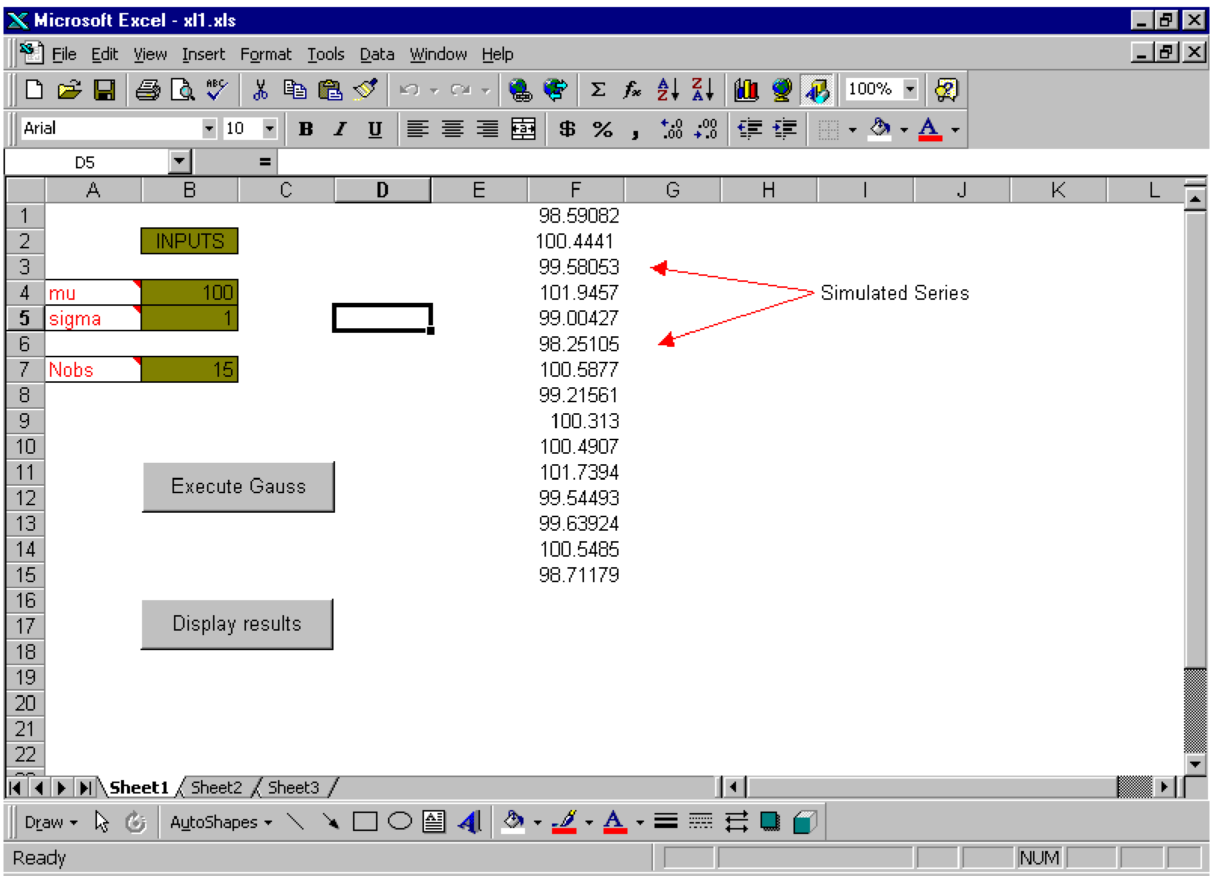
```
/*
** BS.PRG
*/


call sysstate(24,sysstate(2,0));
dlibrary vblink.dll ;
library vblink;
vbl_appname  = &Application;
call vblink;

proc (0) = Application;
  local S0,K,sigma,tau,r,b,OptionType,w,d1,d2,Premium;

  S0 = getvalue(''BS_S0'');
  K = getvalue(''BS_K'');
  sigma = getvalue(''BS_sigma'');
  tau = getvalue(''BS_tau'');
  r = getvalue(''BS_r'');
  b = getvalue(''BS_b'');
  OptionType = getstring(''BS_type'');
```

```
  w = sigma.*sqrt(tau);
  d1 = ( ln(S0./K) + b.*tau )./w + 0.5 * w;
  d2 = d1 - w;

  if lower(OptionType) $== ''put'';
    Premium = -S0.*exp((b-r).*tau).*cdfn(-d1) + K.*exp(-r.*tau).*cdfn(-d2);
  else;
    Premium = S0.*exp((b-r).*tau).*cdfn(d1) - K.*exp(-r.*tau).*cdfn(d2);
  endif;

  call sendvalue(''BS_premium'',Premium);

  retp;
endp;
```

1.6.2 Using temporary files

1.6.2.1 A very simple example

- In the following example, random numbers are simulated from the distribution $\mathcal{N}\left(\mu,\sigma^2\right)$. We compute them with Gauss from an Excel interface. The sheet looks like the picture below:



There are two command buttons. To each button, we have associated one macro:

$\boxed{\text{Execute Gauss}} \Longleftrightarrow$ `Execute_Gauss()`

$\boxed{\text{Display Results}} \Longleftrightarrow$ `Display_Results()`

- The VBA code is the following:

```
Dim GaussPath As String
---------------------------------------------
Sub Execute_Gauss()
    Dim GaussProgram As String
    GaussPath = ''d:\gauss''
    GaussProgram = GaussPath + ''\gauss.exe '' _
                   + GaussPath + ''\conf3\xl1.prg''
```

```
    Save_Data
    Shell GaussProgram, vbMinimizedNoFocus
End Sub
---------------------------------------------
Sub Save_Data()
    Dim GaussData As String
    Dim mu As Double
    Dim sigma As Double
    Dim Nobs As Integer
    Dim fh As Byte

    GaussData = GaussPath + ''\conf3\xl1_in.tmp''

    mu = Worksheets(''Sheet1'').Cells(4, 2).Value
    sigma = Worksheets(''Sheet1'').Cells(5, 2).Value
    Nobs = Worksheets(''Sheet1'').Cells(7, 2).Value

    fh = FreeFile
    Open GaussData For Output As #fh
    Write #fh, mu
    Write #fh, sigma
    Write #fh, Nobs
    Close #fh
End Sub
---------------------------------------------
Sub Display_Results()
    Dim GaussData As String
    Dim Nobs As Integer
    Dim I As Integer
    Dim fh As Byte
    Dim simul As Double

    GaussData = GaussPath + ''\conf3\xl1_out.tmp''

    Nobs = Worksheets(''Sheet1'').Cells(7, 2).Value
    Worksheets(''Sheet1'').Columns(''F'').Value = ''''

    fh = FreeFile
    Open GaussData For Input As #fh

    Input #fh, simul 'skip the first line

    For I = 1 To Nobs
        Input #fh, simul
        Worksheets(''Sheet1'').Cells(I, 6).Value = simul
    Next I

    Close #fh
End Sub
```

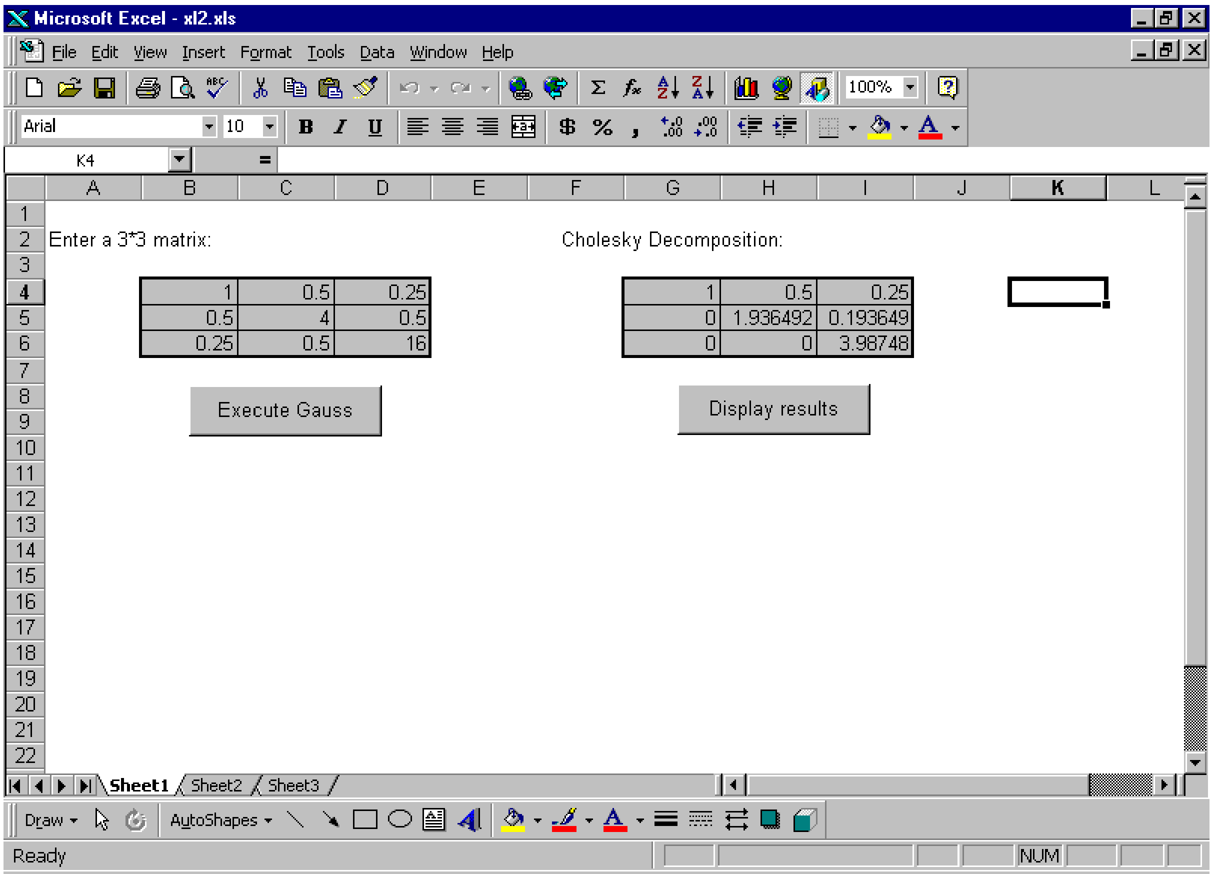
- ▶ The Gauss *xl1.prg* program first loads the data from the temporary file *xl1_ in.tmp* created by the Excel VBA subroutine SAVE_DATA(). Then, it simulates the data and saves them into the temporary file *xl1_ out.tmp*. The VBA DISPLAY_RESULTS() subroutine.reads these random numbers.

```
new;

GaussPath = sysstate(2,0);
cnd = ChangeDir(GaussPath $+''conf3'');

tmpfile = ''xl1_in.tmp'';
load data[] = ^tmpfile;

mu = data[1];
sigma = data[2];
Nobs = data[3];

u = mu + sigma * rndn(Nobs,1);

tmpfile = ''xl1_out.tmp'';

output file = ^tmpfile reset;

screen off;
print u;

output off;

system;
```

Remark 2 *The program could be improved by using the* SHELLEXECUTE *API function in place of the* SHELL *VB function, because we could hide the Gauss application.*

1.6.2.2 Working with matrices

- Suppose that we have to use matrices. In this case, we could employ the free format by indicating the numbers of row and column in the beginning of the ascii file. Then, we could save the matrix in a vector form. With the `reshape` command, it is possible to build the corresponding matrix from the ascii file. Note that the program indicates if the matrix is positive definite.



- This is the VBA code:

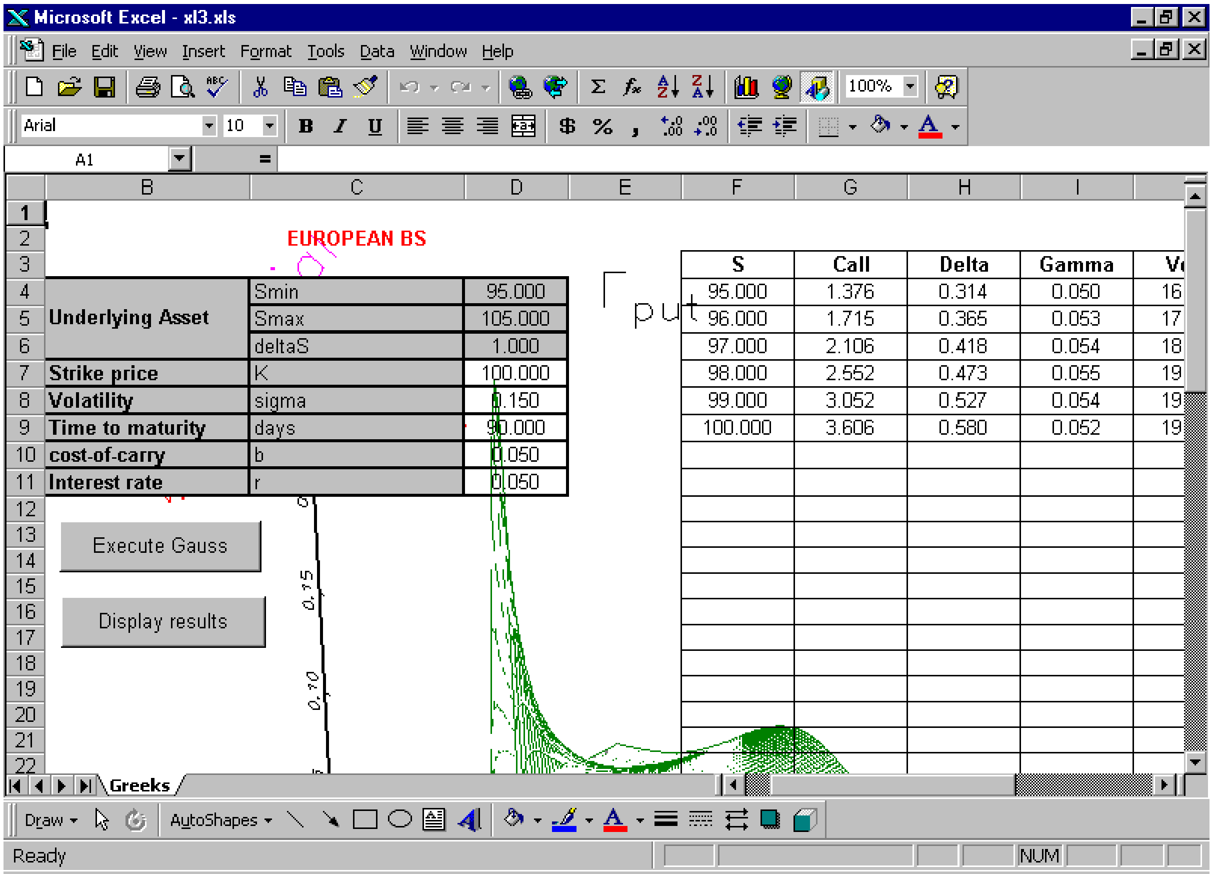
```
Dim GaussPath As String
---------------------------------------------
Sub Execute_Gauss()
    Dim GaussProgram As String
    GaussPath = ''d:\gauss''
    GaussProgram = GaussPath + ''\gauss.exe '' _
                 + GaussPath + ''\conf3\xl2.prg''
    Save_Data
    Shell GaussProgram, vbMinimizedNoFocus
End Sub
---------------------------------------------
Sub Save_Data()
    Dim GaussData As String
    Dim matrix(3, 3) As Double
    Dim Ncols As Integer
    Dim Nrows As Integer
    Dim fh As Byte

    GaussData = GaussPath + ''\conf3\xl2_in.tmp''

    Ncols = 3
    Nrows = 3

    For I = 1 To Ncols
        For J = 1 To Ncols
        matrix(I, J) = Worksheets(''Sheet1'').Cells(3 + I, 1 + J).Value
        Next J
    Next I

    fh = FreeFile
    Open GaussData For Output As #fh
```

```
    Write #fh, Nrows
    Write #fh, Ncols

    For I = 1 To Ncols
        For J = 1 To Nrows
            Write #fh, matrix(I, J)
        Next J
    Next I

    Close #fh

End Sub
---------------------------------------------
Sub Display_Results()
    Dim GaussData As String
    Dim Nrows As Integer
    Dim Ncols As Integer
    Dim I As Integer
    Dim J As Integer
    Dim fh As Byte
    Dim data As Double

    GaussData = GaussPath + ''\conf3\xl2_out.tmp''

    Ncols = 3
    Nrows = 3

    fh = FreeFile
    Open GaussData For Input As #fh

    Input #fh, data 'skip the first line

    If data = -999 Then
        For I = 1 To Nrows
            For J = 1 To Ncols
                Worksheets(''Sheet1'').Cells(3 + I, 6 + J).Value = ''''
            Next J
        Next I
        End
    End If

    For I = 1 To Nrows
        For J = 1 To Ncols
            Input #fh, data
            Worksheets(''Sheet1'').Cells(3 + I, 6 + J).Value = data
        Next J
    Next I

    Close #fh
End Sub
```

▶ This is the Gauss program:

```
new;

GaussPath = sysstate(2,0);
cnd = ChangeDir(GaussPath $+''conf3'');

tmpfile = ''xl2_in.tmp'';
load data[] = ^tmpfile;

Nrows = data[1];
Ncols = data[2];
M = reshape(trimr(data,2,0),Nrows,Ncols);

old = trapchk(1);
trap 1,1;
Mchol = chol(M);
trap old,1;

tmpfile = ''xl2_out.tmp'';
output file = ^tmpfile reset;

screen off;

if scalerr(Mchol);
  print ''-999'';
else;
  print vecr(Mchol);
endif;

output off;

system;
```

1.6.2.3 An option pricer



1.6.3 Using a disk buffer

When you are working with different matrics, you could use a disk buffer. **VBgauss** consists in one Gauss library and one VB module. The main functions are `VBclear` (clears a disk buffer), `VBput` (inserts a matrix into a disk buffer) and `VBread` (reads a matrix from a disk buffer constructed with `VBput`). The VB module contains also a function GaussDir (indicate the Gauss directory), a routine to run Gauss programs (ExecuteGauss), and three routines for working with cells (CellsToMatrix, MatrixToCells and ClearCells).

▶ The VBgauss.src file is

```
proc (1) = VBclear(diskbuffer);
  local file1,file2,f1,f2;

  if strindx(diskbuffer,'':/'',1) == 0;
    diskbuffer = ChangeDir(0) $+ ''\\'' $+ diskbuffer;
  endif;

  file1 = diskbuffer $+ ''.inf'';
  file2 = diskbuffer $+ ''.asc'';

  f1 = fopen(file1,''w'');
  f2 = fopen(file2,''w'');

  if (f1 == 0) or (f2 == 0);
    ERRORLOG ''error: could not clear the disk buffer'';
    retp(0);
  endif;

  f1 = close(f1);
  f2 = close(f2);

  retp(1);
endp;


proc (1) = VBput(diskbuffer,x,name);
```

```
  local file1,file2,f1,f2;
  local r,c,N,str,i;

  if strindx(diskbuffer,'':/'',1) == 0;
    diskbuffer = ChangeDir(0) $+ ''\\'' $+ diskbuffer;
  endif;

  file1 = diskbuffer $+ ''.inf'';
  file2 = diskbuffer $+ ''.asc'';

  if VBisFile(file1) and VBisFile(file2);
    f1 = fopen(file1,''a'');
    f2 = fopen(file2,''a'');
  else;
    f1 = fopen(file1,''w'');
    f2 = fopen(file2,''w'');
  endif;

  if (f1 == 0) or (f2 == 0);
    ERRORLOG ''error: could not open the disk buffer'';
    retp(0);
  endif;

  r = rows(x);
  c = cols(x);
  N = r*c;

  str = ''\'''' $+ name $+ ''\'''' $+ ftos(r,'',%lf'',1,0)
           $+ ftos(c,'',%lf'',1,0);

  call fseek(f1,0,2);
  call fputst(f1,str);

  call fseek(f2,0,2);
  x = vec(x);
  i = 1;
  do until i > N;
    str = ftos(x[i],''%lf'',20,10);
    call fputst(f2,str);
    i = i + 1;
  endo;

  f1 = close(f1);
  f2 = close(f2);

  retp(1);
endp;


proc (1) = VBread(diskbuffer,name);
  local file1,file2,f1,f2;
  local pos,i,str,name_,r,c,x;
  local errCode,errString;

  if strindx(diskbuffer,'':/'',1) == 0;
    diskbuffer = ChangeDir(0) $+ ''\\'' $+ diskbuffer;
  endif;

  file1 = diskbuffer $+ ''.inf'';
  file2 = diskbuffer $+ ''.asc'';

  f1 = fopen(file1,''r'');
  f2 = fopen(file2,''r'');

  pos = 1;

  errCode = 1;

  do until eof(f1);

    str = fgetsa(f1,1);
    {name_,r,c} = VBtoken(str);

    if name_ $== name;
   errCode = 0;
      break;
    endif;

    pos = pos + r*c;

  endo;

  if errCode;
    errString = ''error: '' $+ name $+ '' not found'';
    ERRORLOG errString;
```

```
 retp(error(0));
  endif;

  i = 1;
  do until i >= pos;
    x = fgetsa(f2,1);
    i = i + 1;
  endo;

  x = zeros(r*c,1);
  i = 1;
  do until i > r*c;
    x[i] = stof(fgetsa(f2,1));
    i = i + 1;
  endo;

  x = reshape(x,c,r)';

  f1 = close(f1);
  f2 = close(f2);

  retp(x);
endp;


proc (1) = VBisFile(fname);
  local fnames,finfo;

  {fnames,finfo} = fileinfo(fname);

  if finfo == 0;
    retp(0);
  else;
    retp(1);
  endif;

endp;


proc (3)= VBtoken(str);
  local pos,name,r,c;

  pos = strindx(str,'',''',1);
  str = strput('' '',str,pos);
  pos = strindx(str,'',''',1);
  str = strput('' '',str,pos);

  pos = strindx(str,''\'''',1);
  str = strput('' '',str,pos);
  pos = strindx(str,''\'''',1);
  str = strput('' '',str,pos);

  {name,str} = token(str);

  {r,str} = token(str);
  r = stof(r);

  {c,str} = token(str);
  c = stof(c);

  retp(name,r,c);
endp;
```

▶ The VBgauss.bas file is

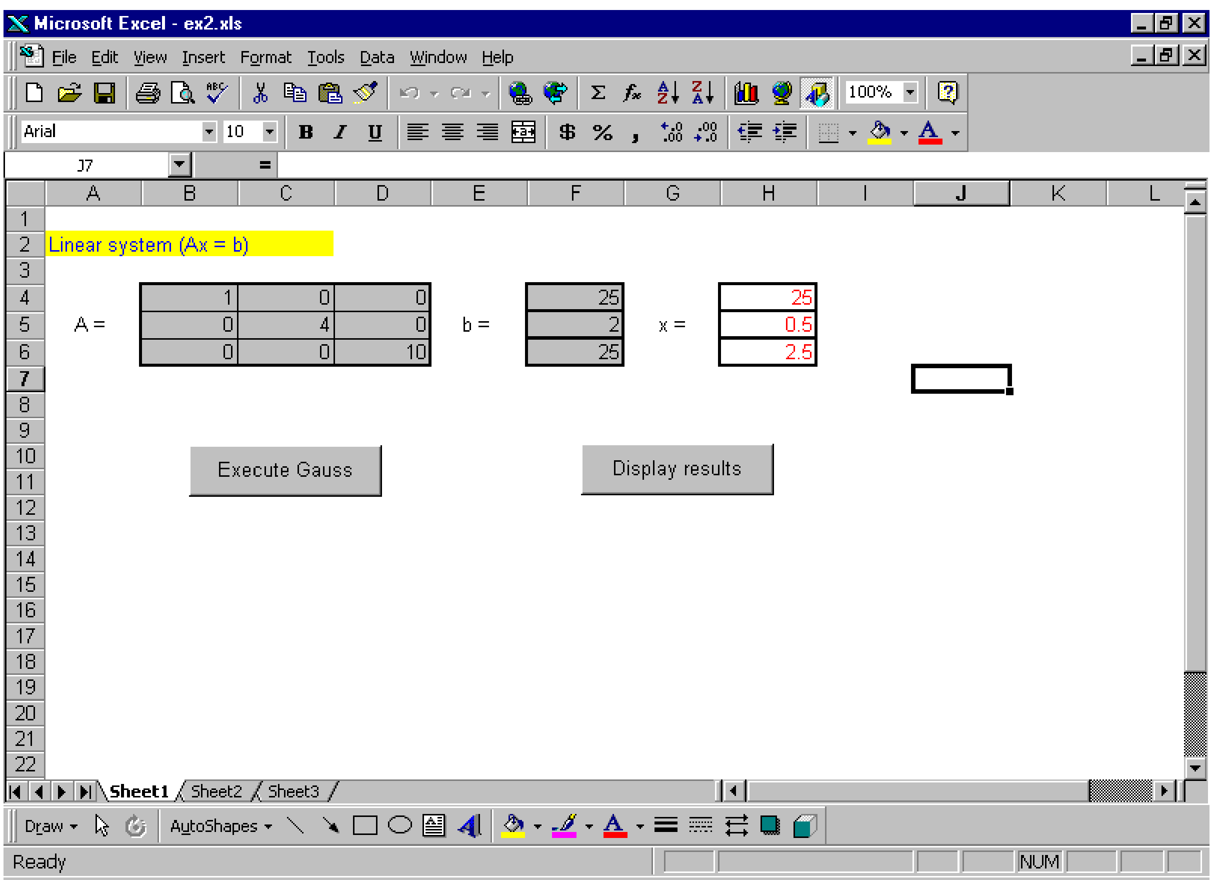
```
Public Function GaussDir() As String
    GaussDir = ''d:\gauss\''
End Function
---------------------------------------------
Public Sub ExecuteGauss(ByVal GaussProgram As String)
    Dim GaussExe, GaussCmd As String

    GaussExe = GaussDir + ''gauss.exe''
    GaussCmd = GaussExe + '' '' + GaussProgram

    Shell GaussCmd, vbHide

End Sub
---------------------------------------------
Public Sub VBclear(ByVal diskbuffer As String)
    Dim file1, file2 As String
    Dim f1, f2 As Byte

    On Error GoTo errMessage
```

```
    If InStr(1, diskbuffer, '':\'', vbTextCompare) = 0 Then
        diskbuffer = CurDir + ''\'' + diskbuffer
    End If

    file1 = diskbuffer + ''.inf''
    file2 = diskbuffer + ''.asc''

    f1 = FreeFile

    Open file1 For Output As #f1
    Close #f1

    f2 = FreeFile
    Open file2 For Output As #f2
    Close #f2

    Exit Sub

errMessage:
    MsgBox ''error: could not clear the disk buffer''
    Exit Sub

End Sub
---------------------------------------------
Public Sub VBput(ByVal diskbuffer As String, ByRef x() As Double, _
                 ByVal name As String)
    Dim file1, file2 As String
    Dim f1, f2 As Byte
    Dim r, c, N As Integer
    Dim i, j As Integer

    On Error GoTo errMessage

    If InStr(1, diskbuffer, '':\'', vbTextCompare) = 0 Then
        diskbuffer = CurDir + ''\'' + diskbuffer
    End If

    file1 = diskbuffer + ''.inf''
    file2 = diskbuffer + ''.asc''

    f1 = FreeFile
    f2 = FreeFile

    r = UBound(x, 1)
    c = UBound(x, 2)
    N = r * c

    Open file1 For Append As #f1
    Write #f1, name, r, c
    Close #f1

    Open file2 For Append As #f2
    For j = 1 To c
        For i = 1 To r
            Write #f2, x(i, j)
        Next i
    Next j
    Close #f2

    Exit Sub

errMessage:
    MsgBox ''error: could not open the disk buffer''
    Exit Sub

End Sub
---------------------------------------------
Public Sub VBread(ByVal diskbuffer As String, ByVal name As String, _
                  ByRef x() As Double)
    Dim file1, file2 As String
    Dim f1, f2 As Byte
    Dim name_ As String
    Dim r, c, pos As Integer
    Dim i, j As Integer
    Dim xi As Double

    On Error GoTo errMessage

    If InStr(1, diskbuffer, '':\'', vbTextCompare) = 0 Then
        diskbuffer = CurDir + ''\'' + diskbuffer
    End If

    file1 = diskbuffer + ''.inf''
    file2 = diskbuffer + ''.asc''
```

```
    f1 = FreeFile
    f2 = FreeFile

    Open file1 For Input As #f1

    pos = 0

    Do While Not EOF(f1)

        Input #f1, name_, r, c

        If name_ = name Then
            Exit Do
        End If

        pos = pos + r * c

    Loop

    Close #f1

    Open file2 For Input As #f2

    For i = 1 To pos
        Input #f2, xi
    Next i

    ReDim x(1 To r, 1 To c)

    For j = 1 To c
        For i = 1 To r
            Input #f2, x(i, j)
        Next i
    Next j

    Close #f2

    Exit Sub

errMessage:
    MsgBox ''error: '' + name + '' not found''
    Exit Sub

End Sub
---------------------------------------------
Public Sub CellsToMatrix(ByVal Sheet As String, _
                         ByVal minX As Integer, _
                         ByVal minY As Integer, _
                         ByVal r As Integer, _
                         ByVal c As Integer, _
                         ByRef x() As Double)
    Dim i, j As Integer

    ReDim x(1 To r, 1 To c)

    On Error GoTo errMessage

    For i = 1 To r
        For j = 1 To c
            x(i, j) = Worksheets(Sheet).Cells(minX + i - 1, minY + j - 1).Value
        Next j
    Next i

    Exit Sub

errMessage:
    MsgBox ''error: CellsToMatrix''
    Exit Sub

End Sub
---------------------------------------------
Public Sub MatrixToCells(ByVal Sheet As String, _
                         ByVal minX As Integer, _
                         ByVal minY As Integer, _
                         ByRef x() As Double)
    Dim r, c As Integer
    Dim i, j As Integer

    r = UBound(x, 1)
    c = UBound(x, 2)

    On Error GoTo errMessage

    For i = 1 To r
```

```
        For j = 1 To c
            Worksheets(Sheet).Cells(minX + i - 1, minY + j - 1).Value = x(i, j)
        Next j
    Next i

    Exit Sub

errMessage:
    MsgBox ''error: MatrixToCells''
    Exit Sub

End Sub
---------------------------------------------
Public Sub ClearCells(ByVal Sheet As String, _
                      ByVal minX As Integer, _
                      ByVal minY As Integer, _
                      ByVal r As Integer, _
                      ByVal c As Integer)
    Dim i, j As Integer

    On Error GoTo errMessage

    For i = 1 To r
        For j = 1 To c
            Worksheets(Sheet).Cells(minX + i - 1, minY + j - 1).Value = ''''
        Next j
    Next i

    Exit Sub

errMessage:
    MsgBox ''error: ClearCells''
    Exit Sub

End Sub
```

1.6.3.1 A very simple example



► This is the VBA code:

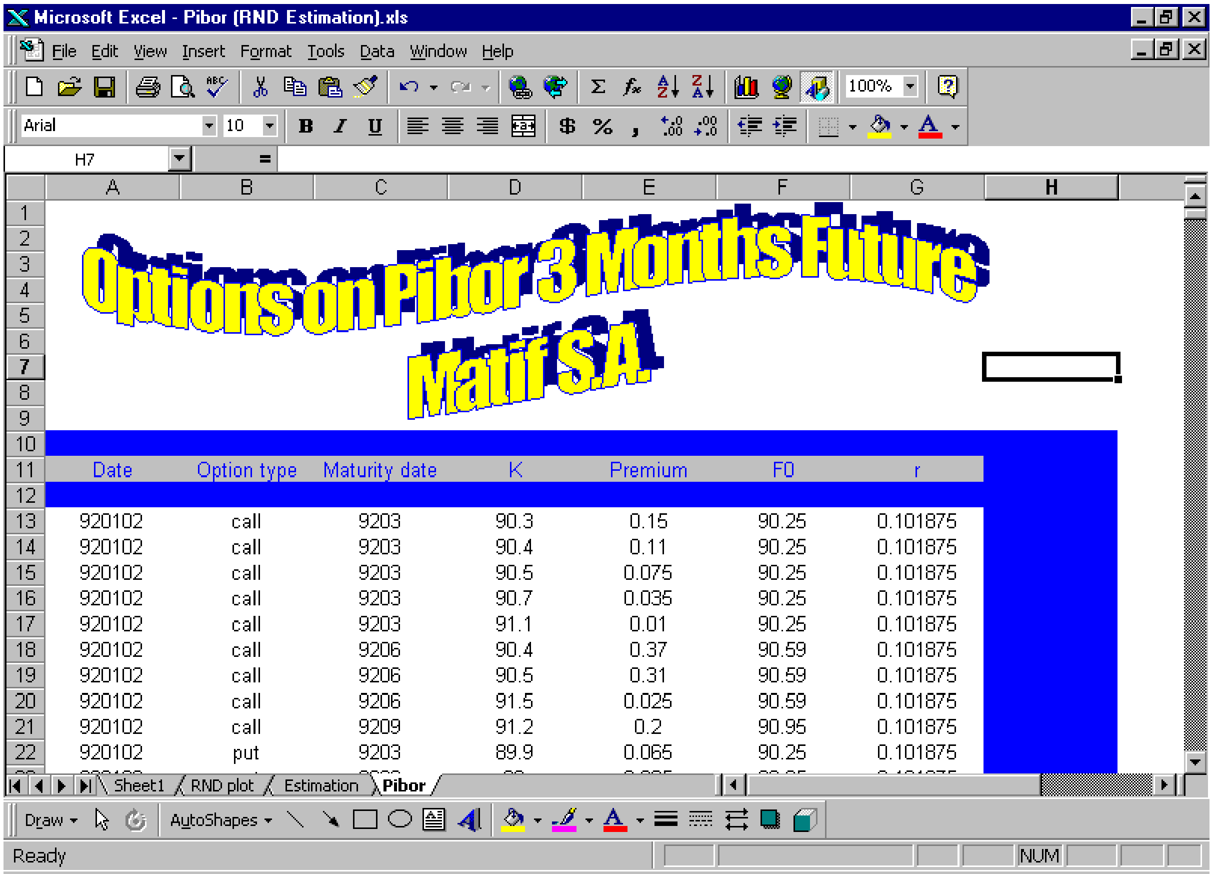
```
Sub Execute_Gauss()
```

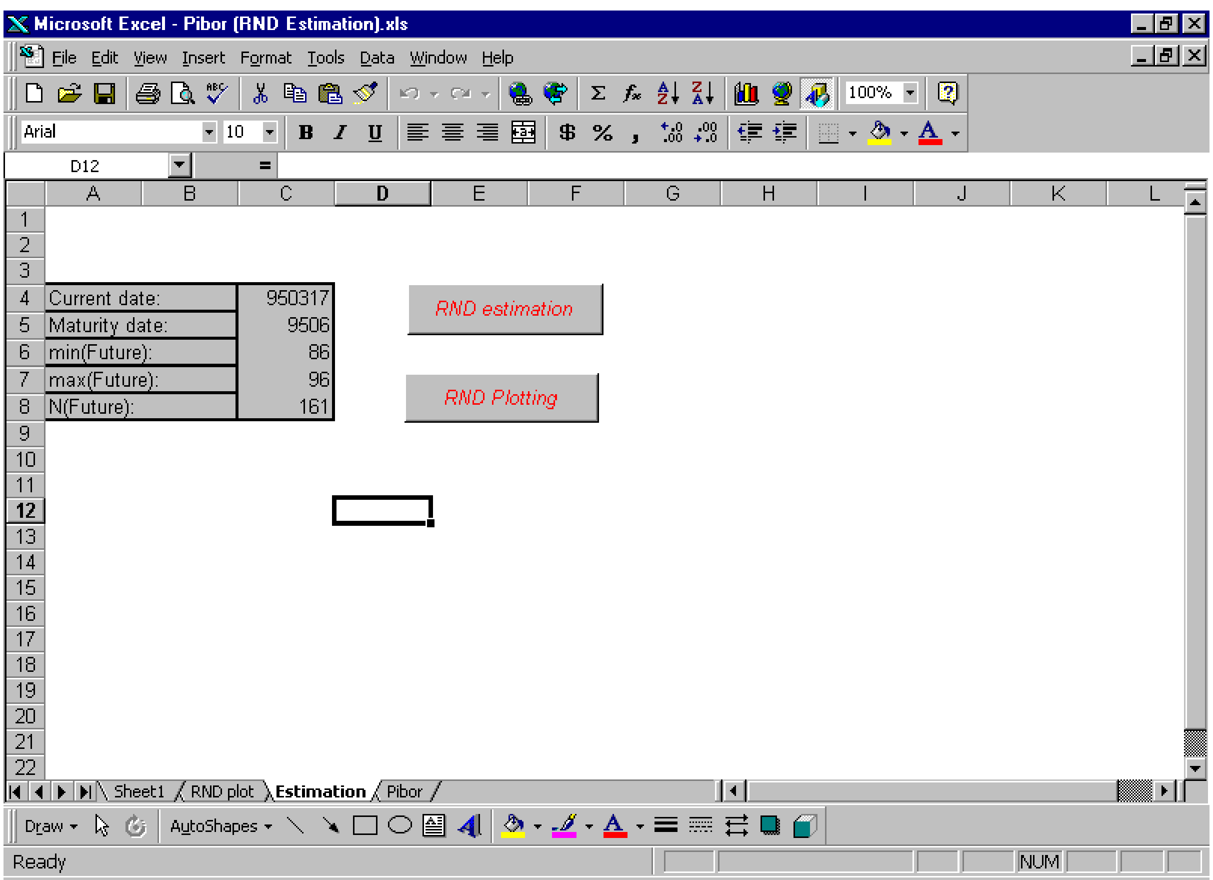
```
    Dim GaussProgram As String

    Save_Data

    GaussProgram = GaussDir + ''Vbgauss\ex2.prg''
    ExecuteGauss GaussProgram
    ClearCells ''Sheet1'', 4, 8, 3, 1

End Sub
---------------------------------------------
Sub Save_Data()
    Dim GaussData As String
    Dim A() As Double
    Dim b() As Double

    GaussData = GaussDir + ''VBGauss\ex2''

    CellsToMatrix ''Sheet1'', 4, 2, 3, 3, A()
    CellsToMatrix ''Sheet1'', 4, 6, 3, 1, b()

    VBclear GaussData
    VBput GaussData, A, ''A''
    VBput GaussData, b, ''b''

End Sub
---------------------------------------------
Sub Display_Results()
    Dim GaussData As String
    Dim x() As Double

    GaussData = GaussDir + ''Vbgauss\ex2''

    VBread GaussData, ''x'', x()
    MatrixToCells ''Sheet1'', 4, 8, x()
End Sub
```

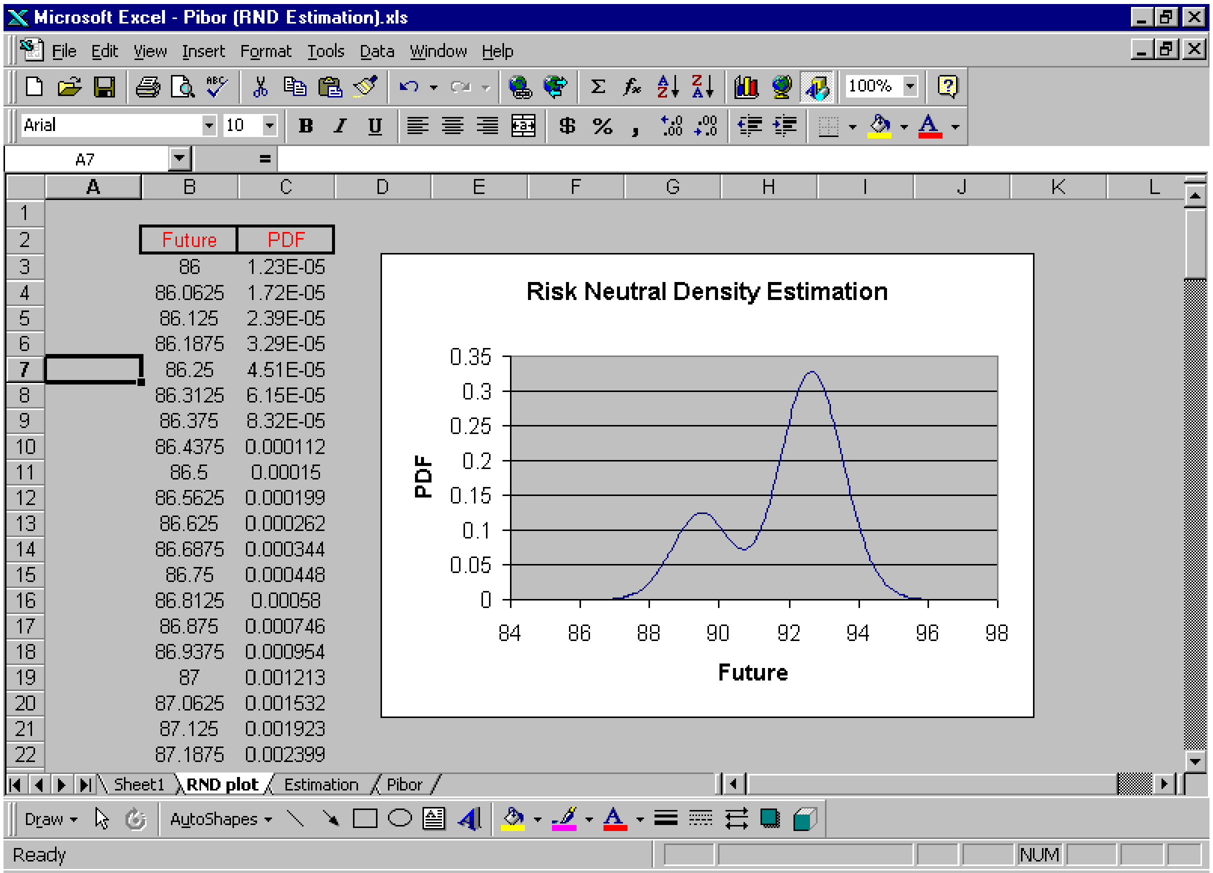
- ▶ This is the Gauss program:

```
new;
library VBgauss;

GaussPath = sysstate(2,0);
cnd = ChangeDir(GaussPath $+''VBgauss'');

a = VBread(''ex2'',''A'');
b = VBread(''ex2'',''b'');
x = b/A;
call VBput(''ex2'',x,''x'');

system;
```

1.6.3.2 Risk neutral density estimation







▶ This is the VBA code:

```
Sub Execute_Gauss()
    Dim indx() As Integer
    Dim GaussProgram As String

    Find_Data indx()
    MsgBox ''Data sorting finished''
    Save_Data indx()

    GaussProgram = GaussDir + ''VBgauss\rnd.prg''
    ExecuteGauss GaussProgram
    ClearCells ''RND plot'', 3, 2, 1000, 2

End Sub
---------------------------------------------
Sub Save_Data(ByRef indx() As Integer)
    Dim GaussData As String
    Dim r As Integer
    Dim TypeOption As String
    Dim data() As Double
    Dim i, j As Integer
    Dim sheet As String
    Dim inf() As Double

    GaussData = GaussDir + ''VBgauss\rnd''

    r = UBound(indx, 1)
    ReDim data(1 To r, 1 To 5)

    sheet = ''Pibor''

    For i = 1 To r
        TypeOption = Worksheets(sheet).Cells(indx(i, 1), 2).Value
        If TypeOption = ''call'' Then
            data(i, 1) = 1#
        Else
            data(i, 1) = 0#
        End If

        For j = 2 To 5
            data(i, j) = Worksheets(sheet).Cells(indx(i, 1), j + 2).Value
        Next j
```

```
    Next i

    VBclear GaussData
    VBput GaussData, data(), ''data''


    sheet = ''Estimation''
    CellsToMatrix sheet, 4, 3, 5, 1, inf()
    VBput GaussData, inf(), ''inf''

End Sub
---------------------------------------------
Sub Find_Data(ByRef indx() As Integer)
    Dim d, dtm As Integer
    Dim i, j As Integer
    Dim temp(1 To 50, 1 To 1) As Integer
    Dim N As Integer

    sheet = ''Estimation''
    d = Worksheets(sheet).Cells(4, 3).Value
    dtm = Worksheets(sheet).Cells(5, 3).Value

    sheet = ''Pibor''

    N = 0

    For i = 13 To 26954
        d_ = Worksheets(sheet).Cells(i, 1).Value
        dtm_ = Worksheets(sheet).Cells(i, 3).Value
        If d_ = d And dtm_ = dtm Then
            N = N + 1
            temp(N, 1) = i
        End If
    Next i

    ReDim indx(1 To N, 1 To 1)

    For i = 1 To N
        indx(i, 1) = temp(i, 1)
    Next i

End Sub
---------------------------------------------
Sub Display_Results()
    Dim GaussData As String
    Dim x() As Double

    GaussData = GaussDir + ''VBgauss\rnd''

    VBread GaussData, ''F + PDF'', x()
    MatrixToCells ''RND plot'', 3, 2, x()

    Worksheets(''RND plot'').Activate
    Charts.Add
    ActiveChart.ChartType = xlXYScatterSmoothNoMarkers
    ActiveChart.SetSourceData Source:=Sheets(''RND plot'').Range(''B3:C421''), PlotBy:=xlColumns
    ActiveChart.Location Where:=xlLocationAsObject, name:=''RND plot''
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = ''Risk Neutral Density Estimation''
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = ''Future''
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = ''PDF''
        .HasLegend = False
    End With

End Sub
```

▶ This is the Gauss program:

```
new;
library optmum,VBgauss;

rndseed 123;  /* stability problem */

GaussPath = sysstate(2,0);
cnd = ChangeDir(GaussPath $+''VBgauss'');

#include d:\gauss\vbgauss\rnd.src;

data = VBread(''rnd'',''data'');

OptionType = data[.,1];
```

```
K = data[.,2];
Premium = data[.,3];
F0 = data[.,4];
r = data[.,5];

inf = VBread(''rnd'',''inf'');

d = inf[1];
dtm = 100*inf[2] + 15;
minF = inf[3];
maxF = inf[4];
nF = inf[5];
dF = (maxF - minF) / (nF - 1);

tau = etdays_(d,dtm) .* ones(rows(Premium),1);

sigma = 0.01;

sv = (ln(F0[1])-0.5*sigma^2*tau[1])|(sigma*sqrt(tau[1])) |
     (ln(F0[1])-0.5*sigma^2*tau[1])|(sigma*sqrt(tau[1])) |
     0.4                                                ;

output file = rnd.out reset;

theta = estimateRND(Premium,K,tau,r,OptionType,F0,sv);

print theta;

output off;

F = seqa(minF,dF,nF);
PDF = pdf2LN(F,theta[1],theta[2],theta[3],theta[4],theta[5]);

VBput(''rnd'',F~PDF,''F + PDF'');

system;
```

▶ This is the source code for RND estimation with two log-normal processes:

```
declare matrix _estimateRND_Data;


proc (1) = estimateRND(Premium,K,tau,r,OptionType,F0,sv);
  local theta,f,g,retcode;

  _estimateRND_Data = Premium ~K ~tau ~r ~OptionType ~F0;

  {theta,f,g,retcode} = optmum(&_estimateRND,sv);
  theta = sqrt(theta^2);
  theta[5] = 0.05 + 0.8 * exp(-theta[5])/(1+exp(-theta[5]));

  retp(theta);
endp;

proc _estimateRND(theta);
  local mu1,mu2,sigma1,sigma2,alpha,m;
  local Premium,K,r,tau,OptionType,F0;
  local TheoreticalPremium,u,v;

  theta = sqrt(theta^2); /* positivity */

  /* 0.05 <= alpha <= 0.45 */
  theta[5] = 0.05 + 0.8 * exp(-theta[5])/(1+exp(-theta[5]));

  mu1 = theta[1];
  sigma1 = theta[2];
  mu2 = theta[3];
  sigma2 = theta[4];
  alpha = theta[5];

  m = alpha * exp( mu1 + 0.5 * (sigma1^2) )
      + (1 - alpha) * exp( mu2 + 0.5 * (sigma2^2) );

  Premium = _estimateRND_Data[.,1];
  K = _estimateRND_Data[.,2];
  tau = _estimateRND_Data[.,3];
  r = _estimateRND_Data[.,4];
  OptionType = _estimateRND_Data[.,5];
  F0 = _estimateRND_Data[.,6];

  TheoreticalPremium =
     _OptionPrice_with2LN(mu1,sigma1,mu2,sigma2,alpha,K,tau,r,OptionType);

  u = Premium - TheoreticalPremium;
```

```
  v = F0 - m;

  retp(u'u + 10 * v'v);
endp;


proc _OptionPrice_with2LN(mu1,sigma1,mu2,sigma2,alpha,K,tau,r,OptionType);
  local a1,b1,c1,a2,b2,c2;
  local Call_,Put_,Premium;

  a1 = exp( mu1 + 0.5 * (sigma1^2) );
  b1 = cdfn( ( mu1 + (sigma1^2) - ln(K) ) ./ sigma1 );
  c1 = cdfn( ( mu1 - ln(K) ) ./ sigma1 );

  a2 = exp( mu2 + 0.5 * (sigma2^2) );
  b2 = cdfn( ( mu2 + (sigma2^2) - ln(K) ) ./ sigma2 );
  c2 = cdfn( ( mu2 - ln(K) ) ./ sigma2 );

  Call_ = exp(-r.*tau) .* (
                           alpha .* ( a1 .* b1 - K .* c1)
                           + (1 - alpha) .* ( a2 .* b2 - K .* c2)
                          );

  Put_ = exp(-r.*tau) .* (
                           alpha .* ( a1 .* (b1 - 1) - K .* (c1 - 1) )
                           + (1 - alpha) .* ( a2 .* (b2 - 1) - K .* (c2 - 1) )
                          );

  Premium = OptionType .* Call_ + ( 1 - OptionType) .* Put_;

  retp(Premium);
endp;


proc convDates(t);
  local yyyy,mm,dd;

  t = ''19'' $+ ftos(t,''%lf'',6,0);

  yyyy = stof(strsect(t,1,4));
  mm = stof(strsect(t,5,2));
  dd = stof(strsect(t,7,2));

  retp(yyyy|mm|dd);
endp;


proc etdays_(tstart,tend);
  retp( etdays(convDates(tstart),convDates(tend))/360 );
endp;


proc pdf2LN(x,mu1,sigma1,mu2,sigma2,alpha);
  retp( alpha*pdfLN(x,mu1,sigma1) + (1-alpha)*pdfLN(x,mu2,sigma2) );
endp;


proc cdf2LN(x,mu1,sigma1,mu2,sigma2,alpha);
  retp( alpha*cdfLN(x,mu1,sigma1) + (1-alpha)*cdfLN(x,mu2,sigma2) );
endp;


proc pdfLN(x,mu,sigma);
  retp( pdfn((ln(x)-mu)./sigma)./(sigma.*x) );
endp;


proc cdfLN(x,mu,sigma);
  retp( cdfn((ln(x)-mu)./sigma));
endp;
```

Chapter 2

Financial Modelling

2.1 The PDE2D library

The PDE2D is a Gauss implementation of Hopscotch methods described in the working paper "KURPIEL, A. and T. RONCALLI [1998], Hopscotch methods for two-state financial models, FERC Working Paper, City Univesity Business School".

2.1.1 What is PDE2D?

PDE2D is a Gauss library for solving Parabolic and Elliptic Partial Differential Equations (PDE) in 2 space dimensions. It includes Hopscotch and θ-schemes algorithms with finite difference methods. It contains the procedures whose list is given below.

- **Derive**: Computes the derivative of the data with respect to the mesh spacing h
- **ExtractSolution**: Extracts solution $u_{i,j}^{m}$ for a specific value of t, x or y
- **FindIndex**: Returns the indices of the elements of a vector x equal to the elements of a vector v
- **Hopscotch**: Solves the PDE problem with Hopscotch methods
- **PDE2D**: Initializes the PDE problem
- **PDE2Dset**: Resets defaults

2.1.2 Some examples

2.1.2.1 Parabolic problems

We consider the linear parabolic PDE problem defined by

$$\begin{aligned}
a(t,x,y) &= a \\
b(t,x,y) &= b \\
c(t,x,y) &= c \\
d(t,x,y) &= 0 \\
e(t,x,y) &= 0 \\
f(t,x,y) &= 0 \\
g(t,x,y) &= 0
\end{aligned}$$

$\mathfrak{R}$ is set to $[0,1] \times [0,1]$ and we have

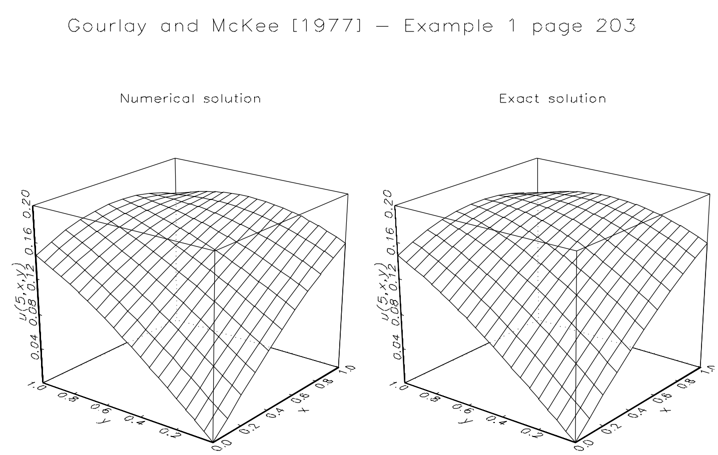
$$\begin{aligned}
u(t,0,y) &= e^{-(a+2b+c)t} \sin(y) \\
u(t,1,y) &= e^{-(a+2b+c)t} \sin(1+y) \\
u(t,x,0) &= e^{-(a+2b+c)t} \sin(x) \\
u(t,x,1) &= e^{-(a+2b+c)t} \sin(1+x)
\end{aligned}$$

The solution of the Cauchy problem with $u(0,x,y) = \sin(x+y)$ is

$$u(t,x,y) = e^{-(a+2b+c)t} \sin(x+y)$$

```
/*
**> pde2d1.prg
**
*/

new;
library PDE2D,pgraph;
PDE2Dset;

a = 0.1; b = 0.05; c = 0.15;

proc aProc(t,x,y);
  retp( a*ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( b*ones(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( c*ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  retp( sin(x+y) );
endp;

proc xminBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(y) );
endp;

proc xmaxBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(1+y) );
endp;

proc yminBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(x) );
endp;

proc ymaxBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(1+x) );
endp;


call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
               0,0,0,0);

xmin = 0; xmax = 1; Nx = 11;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 5; Nt = 51;

_Hopscotch_PrintIters = 2;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''pde2d1'');

x = ExtractSolution(''pde2d1'',''x'');
y = ExtractSolution(''pde2d1'',''y'');
t = ExtractSolution(''pde2d1'',''t'');
U = ExtractSolution(''pde2d1'',''t''|tmax);

/* The exact solution is
**
**          u(t,x,y) = exp(-(a+2b+c)*t)*sin(x+y)
**
```

```
*/

proc solutionProc(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(x+y) );
endp;

sol = solutionProc(tmax,x,y);


graphset;

  begwind;
  makewind(9,1,0,6.855-1,1);
  makewind(9/2,5.855,0,0,1);
  makewind(9/2,5.855,9/2,0,1);

  setwind(1);
    _pdate = ''''; _ptitlht = 1.25; _pnum = 0; _paxes = 0;
    title(''Gourlay and McKee @[1977@] - Example 1 page 203'');
    draw;

  graphset;
    _pdate = ''''; _pnum = 2; _ptitlht = 0.25; _paxht = 0.25; _pnumht = 0.20;
    xlabel(''x'');
    ylabel(''y'');
    zlabel(''u(5,x,y)'');

  setwind(2);
    title(''Numerical solution'')
    surface(x',y',U');

  setwind(3);
    title(''Exact solution'')
    surface(x',y',sol');
    graphprt(''-c=1 -cf=pde2d1.eps'');

  endwind;
```



2.1.2.2 Elliptic problems

We consider the elliptic problem of Gourlay and McKee [1977], page 204:

$$\begin{aligned}
a(x,y) &= 1 \\
b(x,y) &= -\frac{1}{2} \\
c(x,y) &= 1 \\
d(x,y) &= 0 \\
e(x,y) &= 0 \\
f(x,y) &= 0 \\
g(x,y) &= 0
\end{aligned}$$

with the boundary conditions

$$
\begin{aligned}
u(0,y) &= 0 \\
u(1,y) &= y+y^2 \\
u(x,0) &= 0 \\
u(x,1) &= x+x^2
\end{aligned}
$$

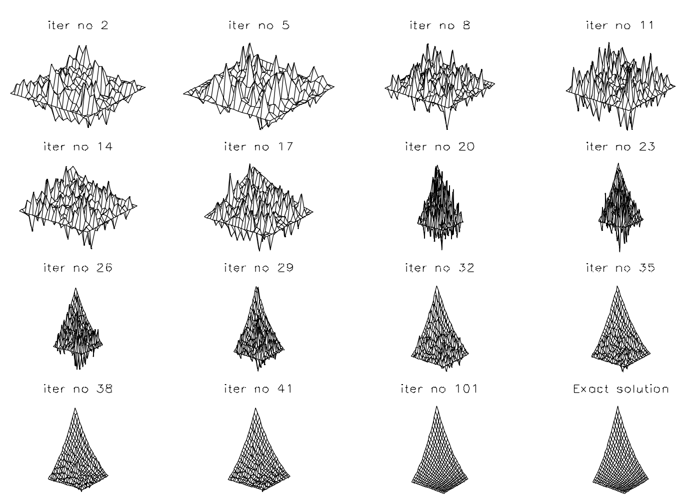
The solution of this problem is

$$u(x,y) = x^2y + xy^2$$

For the initial guess solution, we use random numbers.

```
/*
**> pde2d2.prg
*/

new;
library PDE2D,pgraph;
PDE2Dset;

proc aProc(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( -0.5*ones(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;


proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  retp( ones(rows(x),1) .* (y + y^2) );
endp;

proc yminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc ymaxBound(t,x,y);
  retp( ones(rows(x),1) .* (x + x^2) );
endp;


proc InitialGuessSolution(t,x,y);
  retp( rndn(rows(x),cols(y)) );
endp;


proc solutionProc(t,x,y);
  retp( (x^2).*y + x.*(y^2) );
endp;


rndseed 123;
```

```
call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &InitialGuessSolution,
           &xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.425; Nt = 101;

_Hopscotch_Methods = {2,3}; /* center filling method
                                      &
                               line Hopscotch method */

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''pde2d2'');

x = ExtractSolution(''pde2d2'',''x'');
y = ExtractSolution(''pde2d2'',''y'');
t = ExtractSolution(''pde2d2'',''t'');

sol = solutionProc(tmax,x,y);


graphset;

  begwind;
  window(4,4,0);

  _pdate = ''''; _pnum = 0; _paxes = 0; _pframe = 0; _psurf = {0,0};
  _ptitlht = 0.45;

  i = 1;
  do until i > 14;

    U = ExtractSolution(''pde2d2'',''t''|t[3*i]);

    setwind(i);
      str = ftos(3*i - 1,''iter no %lf'',1,0);
      title(str);
      surface(x',y',U');

    i = i + 1;
  endo;

  U = ExtractSolution(''pde2d2'',''t''|t[Nt]);

  setwind(15);
    str = ftos(Nt,''iter no %lf'',1,0);
    title(str);
    surface(x',y',U');

  setwind(16);
    title(''Exact solution'')
    surface(x',y',sol');

  graphprt(''-c=1 -cf=pde2d2.eps'');

  endwind;
```

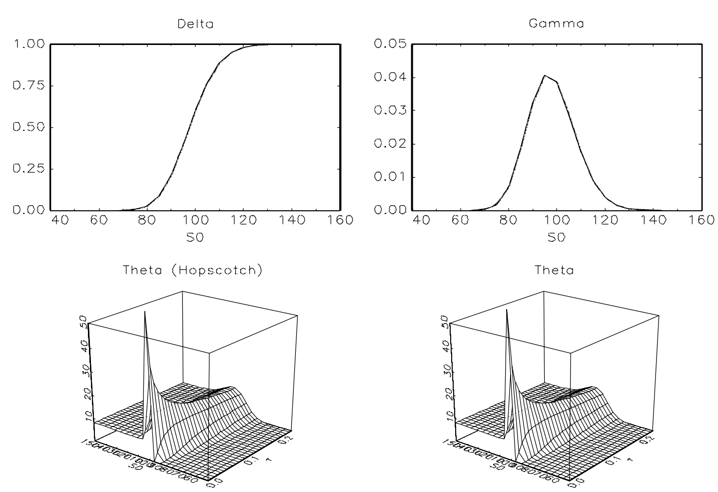


2.1.2.3 Computing Greeks

The following program numerically solves the Black-Scholes model. Then, we use the `Derive` procedure to compute the Δ, Γ and Θ coefficients of the call options. We remark that the differences between these numerical values and these obtained by exact formulas are very small.

```
/*
**> pde2d3.prg
**
*/

new;
library PDE2D,option,pgraph;
PDE2Dset;

K = 100;
sigma = 0.20;
tau = 0.25;
r = 0.08;
b = 0.08;

proc aProc(t,x,y);
  retp( 0.5*(sigma^2) .* (x^2) .* ones(1,cols(y)) );
endp;

proc bProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( b .* x .* ones(1,cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  local PayOff;
  PayOff = x - K;
  PayOff = PayOff .* (PayOff .> 0);
  retp( PayOff .* ones(1,cols(y)) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,0,0,0,
                          0,&DxmaxBound,0,0);

xmin = 50; xmax = 150; Nx = 201;
ymin = 0.10; ymax = 0.30; Ny = 5;
tmin = 0; tmax = tau; Nt = floor(1825*tau);

_Hopscotch_PrintIters = 10;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''pde2d3'');

dt = vread(_Hopscotch_MeshRatios,''k'');
dS = vread(_Hopscotch_MeshRatios,''hx'');

indxT = seqa(10,15,30);
indxS0 = seqa(1,10,21);

graphset;
```

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

_pdate = ''''; _pnum = 2; _pnumht = 0.25; _ptitlht = 0.25; _paxht = 0.25;
fonts(''simplex simgrma'');

setwind(1);

  S0 = ExtractSolution(''pde2d3'',''x'');
  U = ExtractSolution(''pde2d3'',''t''|tau);
  delta = Derive(U,dS,0);

  title(''Delta'');
  xlabel(''S0'');
  ytics(0,1,0.25,0);
  xy(S0[indxS0],submat(delta~EuropeanBS_Delta(S0,K,sigma,tau,b,r),indxS0,0));

setwind(2);

  gamma_ = Derive(Derive(U,dS,-1),dS,1);

  title(''Gamma'');
  ytics(0,0.05,0.01,0);
  xy(S0[indxS0],submat(gamma_~EuropeanBS_Gamma(S0,K,sigma,tau,b,r),indxS0,0));

setwind(3);

  t = ExtractSolution(''pde2d3'',''t'');
  y = ExtractSolution(''pde2d3'',''y'');
  U = ExtractSolution(''pde2d3'',''y''|y[3]);
  theta = Derive(U',dt,0);

  title(''Theta (Hopscotch)'');
  xlabel(''\202t\201'');
  ylabel(''S0'');
  xtics(0,0.20,0.10,0);
  ytics(80,120,10,0);
  ztics(0,50,10,0);
  surface(submat(t',0,indxT),submat(S0,indxS0,0),
          submat(theta',indxS0,indxT));

setwind(4);

  title(''Theta'');
  surface(submat(t',0,indxT),submat(S0,indxS0,0),
          submat(EuropeanBS_Theta(S0,K,sigma,t',b,r),indxS0,indxT));

  graphprt(''-c=1 -cf=pde2d3.eps'');

endwind;
```

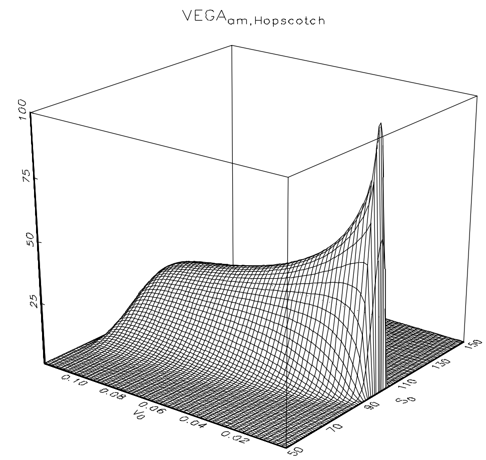


2.1.2.4 Computing stochastic volatility option greeks coefficients

```
==========>

/*
** Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state
```

```
** financial models, FERC working paper, City University Business School
*/

new;
library option,PDE2D;
PDE2Dset;

K = 100;
S0 = seqa(80,5,9);
b = -0.04;
r =  0.08;
V0 = 0.20^2;
tau = 0.25;

rho = -0.5;
Kappa = 0.9;
theta = 0.20^2;
lambda = 0.0;
sigmaV = 0.10;
kappaStar = kappa + lambda;
thetaStar = kappa*theta/(kappa+lambda);

proc aProc(t,x,y);
  retp( 0.5 * y .* (x^2) );
endp;

proc bProc(t,x,y);
  retp( 0.5* rho * sigmaV * y .* x );
endp;

proc cProc(t,x,y);
  retp( 0.5 * (sigmaV^2).*y .* ones(rows(x),1) );
endp;

proc dProc(t,x,y);
  retp( b.*x.*ones(1,cols(y)) );
endp;

proc eProc(t,x,y);
  retp( ( kappaStar*(thetaStar-y) ) .* ones(rows(x),1) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

@<------------------ h(t,x,y,U) ------------------>@

proc hProc(t,x,y,U);
  local PayOff,cnd;
  PayOff = ( x .> K ) .* ( x - K ) .* ones(1,cols(y));
  cnd = PayOff .> U;
  retp( cnd.*PayOff + (1-cnd).*U );
endp;

@<------------------------------------------------>@

proc tminBound(t,x,y);
  x = ones(1,cols(y)) .* x;
  retp( (x .> K) .* ( x - K) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc DyminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DymaxBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,&hProc,
                  &tminBound,&xminBound,0,0,0,
                  0,&DxmaxBound,0,0);
```

```
xmin = 50; xmax = 150; Nx = 101;
ymin = 0.002; ymax = 0.122; Ny = 61;
tmin = 0; tmax = tau; Nt = floor(1825*tau);

_Hopscotch_SaveLastIter = 1;
_Hopscotch_PrintIters = 1;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''pde2d4'');

save ratios = _Hopscotch_MeshRatios;


==========>

/*
** Kurpiel, A. and T. Roncalli [1998], Stochastic volatility and contingent
** claims pricing --- the American option case, FERC working paper,
** City University Business School
*/

new;
library option,PDE2D,pgraph;
PDE2Dset;

K = 100;
S0 = seqa(80,5,9);
b = -0.04;
r =  0.08;
V0 = 0.20^2;
tau = 0.25;

rho = -0.5;
Kappa = 0.9;
theta = 0.20^2;
lambda = 0.0;
sigmaV = 0.10;
kappaStar = kappa + lambda;
thetaStar = kappa*theta/(kappa+lambda);

load MeshRatios = ratios;

S = ExtractSolution(''pde2d4'',''x'');
V = ExtractSolution(''pde2d4'',''y'');
tau = ExtractSolution(''pde2d4'',''t'');
U = ExtractSolution(''pde2d4'',''t''|tau);

dS = vread(MeshRatios,''hx'');
dV = vread(MeshRatios,''hy'');

Delta1 = Derive(U,dS,0);
Vega = Derive(U',dV,0)';
Delta2 = EuropeanBS_Delta(S,K,sqrt(V),tau,b,r);


graphset;
  _pdate = ''''; _pnum = 2;
  _pltype = 6;
  fonts(''simplex simgrma'');
  title(''\202D\201]am,Hopscotch['');
  xy(S,delta1);

graphset;
  _pdate = ''''; _pnum = 2;
  _pltype = 6;
  fonts(''simplex simgrma'');
  title(''\202D\201]am,Hopscotch[ - \202D\201]eu,BS['');
  ylabel(''S]0['');
  xlabel(''V]0['');
  surface(trimr(V',5,5)',S,trimr((Delta1-Delta2)',5,5)');

graphset;
  _pdate = ''''; _pnum = 2;
  title(''VEGA]am,Hopscotch['');
  ztics(0,100,25,0);
  xlabel(''S]0['');
  ylabel(''V]0['');
  graphprt(''-c=1 -cf=pde2d5.eps'');
  surface(S',V',Vega');
```

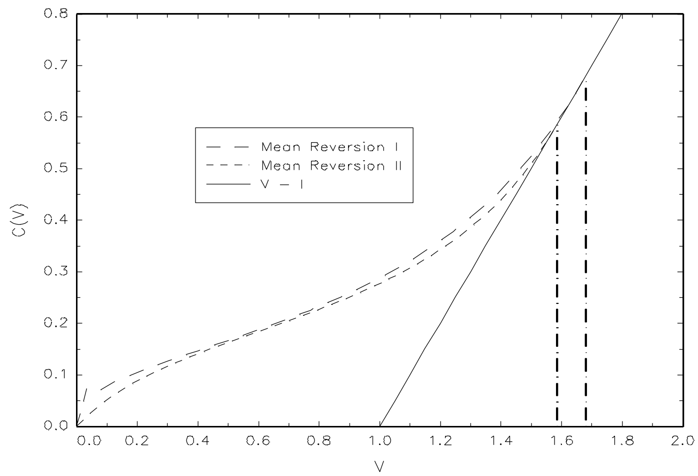


2.1.2.5 A Dixit-Pindyck example

The following program reproduces the figure 7 of "Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state financial models, FERC Working Paper, City Univesity Business School".

```
/*
** Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state
** financial models, FERC working paper
**
*/


new;
library pde2D,pgraph;
PDE2Dset;

declare matrix e;

rndseed 123;

sigma = 0.20;
r = 0.04;
mu = 0.08;
I = 1;
eta = 0.1;
Vbar = 1.5;

proc aProc(t,x,y);
  retp( 0.5*(sigma^2).* (x^e) .* ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( 0 .* ones(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( 0 .* ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( (r - mu + eta*(Vbar-x) ) .* x .* ones(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( 0 .* ones(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  retp( rndu(rows(x),cols(y)) );
```

```
endp;

proc xminBound(t,x,y);
  retp( 0.*ones(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  retp( (x - I) .* ones(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;


proc DyminBound(t,x,y);
  retp( 0*ones(rows(x),cols(y)) );
endp;

proc DymaxBound(t,x,y);
  retp( 0*ones(rows(x),cols(y)) );
endp;

__output = 1;

_Hopscotch_SaveLastIter = 1;
_Hopscotch_Elliptic = 1;
_fcmptol = 1e-4;
_Hopscotch_PrintIters = 10;

output file = wp12.out reset;

V = 1.585|1.68;

sol = {};
x = {};
e = 1;

do until e > 2;

  Vstar = V[e];

  call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
                  &tminBound,&xminBound,&xmaxBound,0,0,
                  0,0,&DyminBound,&DymaxBound);

  xmin = 0.000; xmax = Vstar; Nx = 51;
  ymin = 0.000; ymax = 1; Ny = 5;
  tmin = 0; tmax = 100; Nt = 1001;

  call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');

  x = x ~ExtractSolution(''tmp'',''x'');
  y = ExtractSolution(''tmp'',''y'');
  t = ExtractSolution(''tmp'',''t'');

  U = ExtractSolution(''tmp'',''t''|t);

  hx = vread(_Hopscotch_MeshRatios,''hx'');

  sol = sol~U[.,3];

  e = e + 1;
endo;

V = seqa(1.0,0.05,51);
graphset;
  _pdate = ''''; _pnum = 2; _pltype = 1|3|6;
  xlabel(''V'');
  ylabel(''C(V)'');
  ytics(0,0.8,0.1,2);
  xtics(0,2.0,0.2,2);
  _plegstr = ''Mean Reversion I\000Mean Reversion II\000V - I'';
  _plegctl = {2 5 2.5 4};
  _pline = 1~5~1.585~0~1.585~0.585~1~7~5 |
           1~5~1.680~0~1.680~0.680~1~7~5 ;
  graphprt(''-c=1 -cf=pde2d6.eps'');
  xy(x[.,2 1]~V,sol[.,2 1]~(V-I));
```



2.1.2.6 Other financial problems

▶ See "KURPIEL, A. and T. RONCALLI [1998], Hopscotch methods for two-state financial models, FERC Working Paper, City Univesity Business School".

2.2 The OPTION library

2.2.1 What is OPTION?

OPTION is a Gauss library for option pricing. It includes several models and concerns different option types. It contains the procedures whose list is given below.

- BLACK and SCHOLES [1973] model
  - $\boxed{\textbf{AmericanBS}}$
  - **AmericanBS_Delta**
  - **AmericanBS_Gamma**
  - **AmericanBS_impVol**
  - **AmericanBS_Omega**
  - **AmericanBS_SmileCurve**
  - **AmericanBS_Theta**
  - **AmericanBS_Vega**
  - $\boxed{\textbf{EuropeanBS}}$
  - **EuropeanBS_Delta**
  - **EuropeanBS_Gamma**
  - **EuropeanBS_impVol**
  - **EuropeanBS_Omega**
  - **EuropeanBS_SmileCurve**
  - **EuropeanBS_Theta**
  - **EuropeanBS_Vega**
  - $\boxed{\textbf{EuropeanPayOff}}$
- MERTON [1976] – BATES [1991] model

- **AmericanMerton**
- **AmericanMerton_Delta**
- **AmericanMerton_Gamma**
- **AmericanMerton_impVol**
- **AmericanMerton_Omega**
- **AmericanMerton_Theta**
- **AmericanMerton_Vega**
- **EuropeanMerton**
- **EuropeanMerton_Delta**
- **EuropeanMerton_Gamma**
- **EuropeanMerton_impVol**
- **EuropeanMerton_Omega**
- **EuropeanMerton_Theta**
- **EuropeanMerton_Vega**

• Cox, Ross and Rubinstein [1979] model

- **AmericanCRR**
- **AmericanCRR_Delta**
- **AmericanCRR_Gamma**
- **AmericanCRR_impVol**
- **AmericanCRR_Omega**
- **AmericanCRR_Theta**
- **AmericanCRR_Vega**
- **AsianFixedStrikeCRR**
- **AsianFixedStrikeCRR_Delta**
- **AsianFixedStrikeCRR_Gamma**
- **AsianFixedStrikeCRR_impVol**
- **AsianFixedStrikeCRR_Omega**
- **AsianFixedStrikeCRR_Theta**
- **AsianFixedStrikeCRR_Vega**
- **AsianFloatingStrikeCRR**
- **AsianFloatingStrikeCRR_Delta**
- **AsianFloatingStrikeCRR_Gamma**
- **AsianFloatingStrikeCRR_impVol**
- **AsianFloatingStrikeCRR_Omega**
- **AsianFloatingStrikeCRR_Theta**
- **AsianFloatingStrikeCRR_Vega**
- **EuropeanCRR**
- **EuropeanCRR_Delta**
- **EuropeanCRR_Gamma**
- **EuropeanCRR_impVol**
- **EuropeanCRR_Omega**
- **EuropeanCRR_Theta**

- **EuropeanCRR_Vega**
- $\boxed{\textbf{KnockInDownCRR}}$
- **KnockInDownCRR_Delta**
- **KnockInDownCRR_Gamma**
- **KnockInDownCRR_impVol**
- **KnockInDownCRR_Omega**
- **KnockInDownCRR_Theta**
- **KnockInDownCRR_Vega**
- $\boxed{\textbf{KnockInUpCRR}}$
- **KnockInUpCRR_Delta**
- **KnockInUpCRR_Gamma**
- **KnockInUpCRR_impVol**
- **KnockInUpCRR_Omega**
- **KnockInUpCRR_Theta**
- **KnockInUpCRR_Vega**
- $\boxed{\textbf{KnockOutDownCRR}}$
- **KnockOutDownCRR_Delta**
- **KnockOutDownCRR_Gamma**
- **KnockOutDownCRR_impVol**
- **KnockOutDownCRR_Omega**
- **KnockOutDownCRR_Theta**
- **KnockOutDownCRR_Vega**
- $\boxed{\textbf{KnockOutUpCRR}}$
- **KnockOutUpCRR_Delta**
- **KnockOutUpCRR_Gamma**
- **KnockOutUpCRR_impVol**
- **KnockOutUpCRR_Omega**
- **KnockOutUpCRR_Theta**
- **KnockOutUpCRR_Vega**
- $\boxed{\textbf{LookBackCRR}}$
- **LookBackCRR_Delta**
- **LookBackCRR_Gamma**
- **LookBackCRR_impVol**
- **LookBackCRR_Omega**
- **LookBackCRR_Theta**
- **LookBackCRR_Vega**

• HESTON [1993] model

- $\boxed{\textbf{EuropeanHeston}}$

• BATES [1996] model

- $\boxed{\textbf{EuropeanBates}}$

- Chang, Chang and Lim [1998] model
  - $\boxed{\textbf{AmericanCCL}}$
  - **AmericanCCL_Delta**
  - **AmericanCCL_Gamma**
  - **AmericanCCL_impVol**
  - **AmericanCCL_Omega**
  - **AmericanCCL_Theta**
  - **AmericanCCL_Vega**
  - $\boxed{\textbf{EuropeanCCL}}$
  - **EuropeanCCL_Delta**
  - **EuropeanCCL_Gamma**
  - **EuropeanCCL_impVol**
  - **EuropeanCCL_Omega**
  - **EuropeanCCL_Theta**
  - **EuropeanCCL_Vega**
- Procedures for processes simulating
  - **Simulate_GBM**
  - **Simulate_JumpDiffusion**
  - **Simulate_LevyProcess**
  - **Simulate_mSDE**
  - **Simulate_SDE**
  - **Simulate_SDE2**
  - **Simulate_OU**
- Miscellaneous procedures for option databases managing

2.2.2 Some examples

2.2.2.1 Cox, Ross and Rubinstein model

▶ In the following program, we computes 41 American option prices on futures ($b = 0$) with the Cox, Ross and Rubinstein model. Then, we print the trees with the `PrintTree` procedure.

```
new;
library option;

S0 = seqa(80,1,41);
K = 100;
sigma = 0.20;
tau = 90/365;
r = 0.08;
b = 0;
N = 6;

Cam = AmericanCRR(S0,K,sigma,tau,b,r,N);

output file = option1.out reset;

outwidth 256;
screen off;
PrintTree;
screen on;

output off;


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
OPTION No  1

Underlying Asset Price Tree
```

						102.03257
					97.978479	
				94.085468		94.085468
			90.347140		90.347140	
		86.757348		86.757348		86.757348
	83.310191		83.310191		83.310191	
80.000000		80.000000		80.000000		80.000000
	76.821334		76.821334		76.821334	
		73.768968		73.768968		73.768968
			70.837881		70.837881	
				68.023257		68.023257
					65.320467	
						62.725068

```
Option Price Tree
```

						2.0325715
					0.99241820	
				0.48455558		0.00000000
			0.23658788		0.00000000	
		0.11551579		0.00000000		0.00000000
	0.056401448		0.00000000		0.00000000	
0.027538428		0.00000000		0.00000000		0.00000000
	0.00000000		0.00000000		0.00000000	
		0.00000000		0.00000000		0.00000000
			0.00000000		0.00000000	
				0.00000000		0.00000000
					0.00000000	
						0.00000000

```
              .
              .
              .
              .
              .
              .

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
OPTION No 41

Underlying Asset Price Tree
```

						153.04886
					146.96772	
				141.12820		141.12820
			135.52071		135.52071	
		130.13602		130.13602		130.13602
	124.96529		124.96529		124.96529	
120.00000		120.00000		120.00000		120.00000
	115.23200		115.23200		115.23200	
		110.65345		110.65345		110.65345
			106.25682		106.25682	
				102.03489		102.03489
					97.980701	
						94.087602

```
Option Price Tree
```

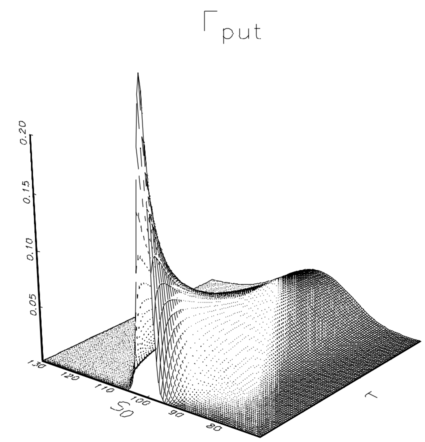
						53.048857
					46.967718	
				41.128202		41.128202
			35.520710		35.520710	
		30.136022		30.136022		30.136022
	24.965286		24.965286		24.965286	
20.000000		20.000000		20.000000		20.000000
	15.359414		15.232001		15.232001	
		11.002363		10.653451		10.653451
			7.0118065		6.2568222	
				3.5601198		2.0348857
					0.99354814	
						0.00000000

2.2.2.2 Black-Scholes Gamma computing

- ▶ The procedures are vectorized in row order. Some of these are vectorized in row **and** column orders. This is the case for the `EuropeanBS_Gamma` procedure.

```
new;
library option,pgraph;
```

```
S0 = seqa(70,0.5,121);
K = 100;
sigma = 0.20;
tau = seqa(90/365,-1/365,89)';
r = 0.08;
b = 0;

_Option_Type = ''put'';


Gamma_ = EuropeanBS_Gamma(S0,K,sigma,tau,b,r);

graphset;
  _pdate = ''''; _pnum = 2; _paxht = 0.30; _ptitlht = 0.30; _pframe = 0;
  fonts(''simplex simgrma'');
  title(''\202G\201]put['');
  xlabel(''\202t\201'');
  ylabel(''S]0]'');
  ytics(70,130,10,0);
  ztics(0,0.20,0.05,0);
  graphprt(''-c=1 -cf=option2.eps'');
  surface(tau,S0,Gamma_);
```



2.2.2.3 Option pricing with jump-diffusion processes

► The following program replicates the results of Bates [1991]. The procedure **EuropeanMerton** is vectorised in row **and** column orders while the procedure **AmericanMerton** is just vectorized in row order. But with Kronecker products...

```
/*
**> Bates91.prg
**
*/

new;
library option;

output file = option3.out reset;

S0 = 250;
b = 0;
r = 0.10;
tau = 0.25;

K = seqa(220,15,5);
let sigma[1,5] =  0.1414   0.10   0.10  0.10  0.10;
let lambdaStar[1,5] = 0       10     10     0.25  0.25;
let gammaStar[1,5] = 0        0.01  -0.01  0.20  -0.20;
let deltaStar[1,5] =  0        0.03   0.03  0      0;

kbarStar = exp(gammaStar) - 1;

_Option_Type = ''call'';
C =  EuropeanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);
_Option_Type = ''put'';
```

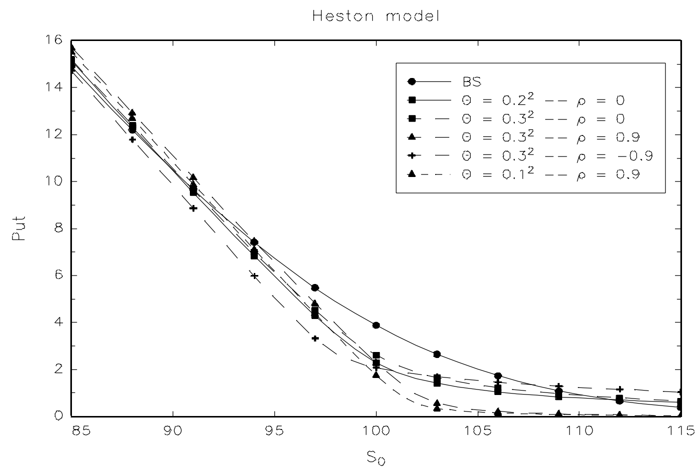
```
P = EuropeanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);

print ''European option prices :'';
call printfm(C,1,''*.*lf'' ~8 ~2);
print;
call printfm(P,1,''*.*lf'' ~8 ~2);
print;

@<-- row and column vectorisation using Kronecker product -->@

e = ones(5,1);
sigma = sigma' .*. e;
kbarStar = kbarStar' .*. e;
deltaStar = deltaStar' .*. e;
lambdaStar = lambdaStar' .*. e;

K = e .*. K;

_Option_Type = ''call'';
C = EuropeanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);
_Option_Type = ''put'';
P = EuropeanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);

C = reshape(C,5,5)';
P = reshape(P,5,5)';

print ''European option prices :'';
call printfm(C,1,''*.*lf'' ~8 ~2);
print;
call printfm(P,1,''*.*lf'' ~8 ~2);
print;


_Option_Type = ''call'';
C = AmericanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);
_Option_Type = ''put'';
P = AmericanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);

C = reshape(C,5,5)';
P = reshape(P,5,5)';

print ''American option prices :'';
call printfm(C,1,''*.*lf'' ~8 ~2);
print;
call printfm(P,1,''*.*lf'' ~8 ~2);
print;

output off;


European option prices :
   29.49   29.45   29.58   29.30   30.14
   16.39   16.25   16.49   15.62   16.71
    6.88    6.81    6.79    6.28    6.02
    2.04    2.17    1.88    2.65    1.11
    0.42    0.56    0.35    1.42    0.09

    0.23    0.19    0.33    0.04    0.88
    1.76    1.62    1.86    0.99    2.08
    6.88    6.81    6.79    6.28    6.02
   16.67   16.79   16.51   17.28   15.74
   29.68   29.82   29.61   30.68   29.35

European option prices :
   29.49   29.45   29.58   29.30   30.14
   16.39   16.25   16.49   15.62   16.71
    6.88    6.81    6.79    6.28    6.02
    2.04    2.17    1.88    2.65    1.11
    0.42    0.56    0.35    1.42    0.09

    0.23    0.19    0.33    0.04    0.88
    1.76    1.62    1.86    0.99    2.08
    6.88    6.81    6.79    6.28    6.02
   16.67   16.79   16.51   17.28   15.74
   29.68   29.82   29.61   30.68   29.35

American option prices :
   30.35   30.18   30.08   30.00   30.25
   16.62   16.47   16.63   16.19   16.73
    6.95    6.88    6.83    6.54    6.02
    2.07    2.19    1.89    2.77    1.11
    0.43    0.57    0.35    1.48    0.09

    0.23    0.19    0.33    0.04    0.92
    1.78    1.63    1.89    0.99    2.19
```

```
   6.95    6.85    6.87    6.29    6.28
  16.92   16.93   16.75   17.30   16.37
  30.46   30.25   30.29   30.76   30.77
```

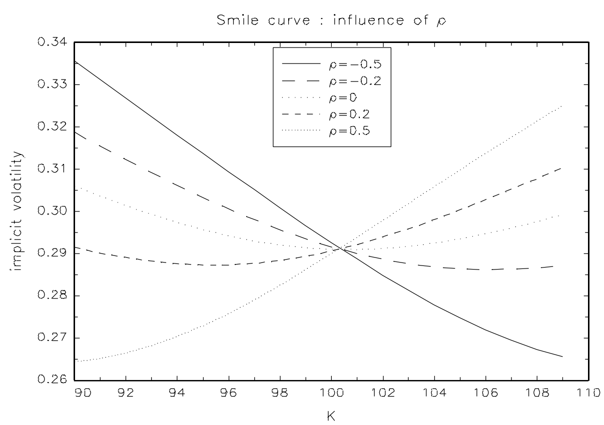
2.2.2.4 Stochastic volatility option models

▶ An example of option pricing with stochastic volatility (Heston model).

```
new;
library option,pgraph;


S0 = seqa(85,1,31);
K = 100;
sigma = 0.20;
tau = 90/365;
r = 0.08;
b = 0;

_Option_Type = ''put'';

P = EuropeanBS(S0,K,sigma,tau,b,r);

V0 = sigma^2; /* V0 = 0.04 */
kappa = 1;
theta = V0;
sigmaV = 2.5;
rho = 0;
lambda = 0;

_Option_Tol = 1e-4;
_Option_NumIters = 10;

{P1,retcode} = EuropeanHeston(S0,K,V0,tau,b,r,
                                       kappa,theta,sigmaV,rho,lambda);

theta = 0.3^2;
rho = 0;

{P2,retcode} = EuropeanHeston(S0,K,V0,tau,b,r,
                                       kappa,theta,sigmaV,rho,lambda);

rho = 0.9;

{P3,retcode} = EuropeanHeston(S0,K,V0,tau,b,r,
                                       kappa,theta,sigmaV,rho,lambda);

rho = -0.9;

{P4,retcode} = EuropeanHeston(S0,K,V0,tau,b,r,
                                       kappa,theta,sigmaV,rho,lambda);

theta = 0.1^2;
rho = 0.9;

{P5,retcode} = EuropeanHeston(S0,K,V0,tau,b,r,
                                       kappa,theta,sigmaV,rho,lambda);

graphset;
  _pdate = ''''; _pnum = 2; _plctrl = 3; _pstype = 8|9|9|10|11|10; _psymsiz = 2.5;
  _pltype = 6|6|1|1|1|3;
  fonts(''simplex simgrma'');
  title(''Heston model'');
  xlabel(''S]0['');
  ylabel(''Put'');
  _plegstr = ''BS\000\202Q\201 = 0.2[2] -- \202r\201 = 0''\
             ''\000\202Q\201 = 0.3[2] -- \202r\201 = 0''\
             ''\000\202Q\201 = 0.3[2] -- \202r\201 = 0.9''\
             ''\000\202Q\201 = 0.3[2] -- \202r\201 = -0.9''\
             ''\000\202Q\201 = 0.1[2] -- \202r\201 = 0.9'';
  _plegctl = {2 5 5 4};
  xtics(85,115,5,0);
  graphprt(''-c=1 -cf=option4.eps'');
  xy(S0,P~P1~P2~P3~P4~P5);
```



► Influence of ρ on smile curve (Kurpiel, A. and T. Roncalli [1998], Option hedging with stochastic volatility, FERC Working Paper, City University Business School).

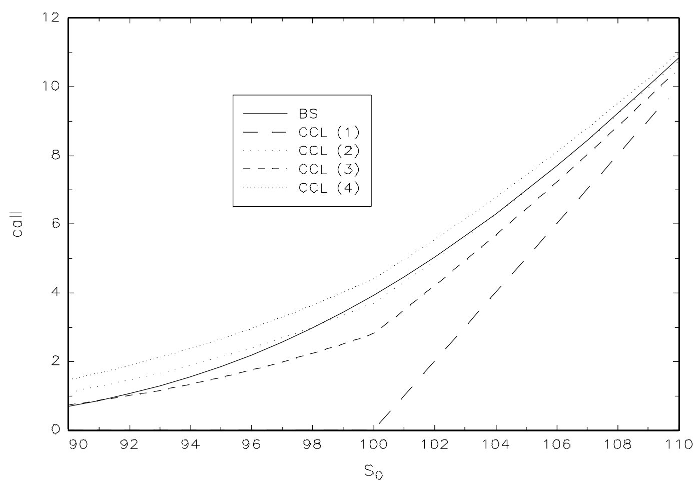


2.2.2.5 Option pricing with subordinated stochastic processes

► An example of option pricing in time-information with subordinated stochastic processes (Chan, Chan and Lim model).

```
new;

library option,pgraph;

K = 100;
r = 0.05;
sigma = 0.20;
tau = 90/365;
b = 0;

S0 = seqa(90,1,21);

_Option_Type = ''call'';

C0 = AmericanBS(S0,K,sigma,tau,b,r);

lambda = 0;
C1 = AmericanCCL(S0,K,sigma,tau,b,r,lambda);

sigma = 0.10;
lambda = 6;
```

```
C2 = AmericanCCL(S0,K,sigma,tau,b,r,lambda);

lambda = 4;
C3 = AmericanCCL(S0,K,sigma,tau,b,r,lambda);

lambda = 8;
C4 = AmericanCCL(S0,K,sigma,tau,b,r,lambda);

graphset;
  fonts(''simplex simgrma'');
  _pdate = ''''; _pnum = 2;
  _plegstr = ''BS\000CCL (1)\000CCL (2)\000CCL (3)\000CCL (4)'';
  _plegctl = {2 5 3 4};
  xlabel(''S]0['');
  ylabel(''call'');
  graphprt(''-c=1 -cf=option5.eps'');
  xy(S0,C0~C1~C2~C3~C4);
```

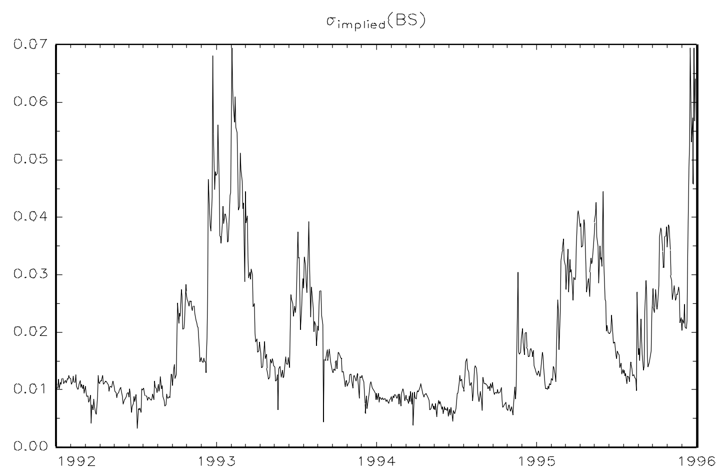


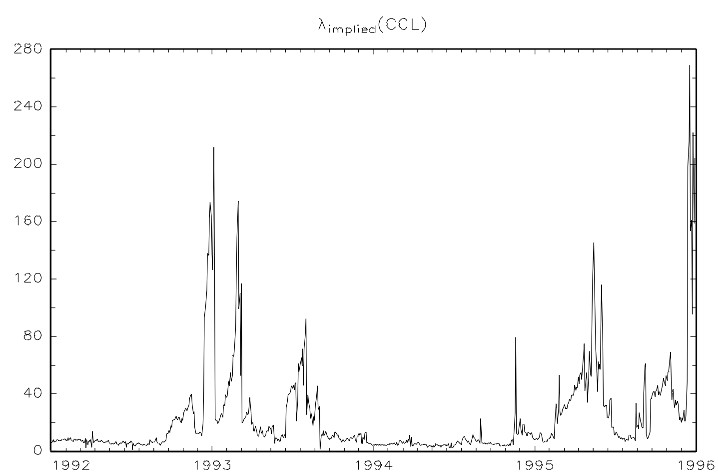
2.2.2.6 Implied volatility and transaction frequency - An application to Pibor 3 months future options

- A very simple example to illustrate the use of subordinated stochastic process model to extract information from derivatives (??).

```
new;
library option,pgraph;


load oam_c;

Nobs = rows(oam_c);
b = 0;

_Option_impVol = ''NR''; /* _Option_impVol = ''bi-section''; */
_Option_Type = ''call'';

tau = zeros(Nobs,1);
i = 1;
do until i > Nobs;
  tau[i] =  option_etdays(oam_c[i,1],oam_c[i,3]);
  i = i + 1;
endo;

F0 = oam_c[.,6];
K = oam_c[.,4];
C = oam_c[.,5];
r = oam_c[.,7]/100;
impVol = EuropeanBS_impVol(F0,K,tau,b,r,C);


sigma = 0.006;
impLambda = zeros(Nobs,1);

_qn_PrintIters = 0;
__output = 0;
```

```
i = 1;
do until i > Nobs;

  proc rss(lambda);
    local u;
    u = EuropeanCCL(oam_c[i,6],oam_c[i,4],sigma,
                tau[i],b,oam_c[i,7]/100,lambda) - oam_c[i,5];
    retp(u^2);
  endp;

  {impLambda[i],f0,g0,retcode} = QNewton(&rss,5);

  i = i + 1;
endo;


graphset;
  _pnum = 2; _pdate = ''''; _pltype = 6|4;
  fonts(''simplex simgrma'');
  title(''\202s\201]implied[(BS)'');
  let lab = ''1992'' ''1993'' ''1994'' ''1995'' ''1996'';
  asclabel(lab,0);
  xtics(1,Nobs,Nobs/4,11);
  graphprt(''-c=1 -cf=option6a.eps'');
  xy(seqa(1,1,Nobs),impVol);

  title(''\202l\201]implied[(CCL)'');
  graphprt(''-c=1 -cf=option6b.eps'');
  xy(seqa(1,1,Nobs),impLambda);
```





Chapter 3

Financial Econometrics

3.1 FANPAC

FANPAC includes:

1. OLS, ARCH, ARIMA, GARCH, Fractionally integrated GARCH, GARCH-in-Mean, EGARCH and IGARCH models, in either Normal or t-distributions.

2. Procedures for computing standardized residuals.

3. Conditional variances, and forecasts as well as confidence limits by inversion of the Wald statistic.

The latest release of FANPAC contains new multivariate models: three basic models, Diagonal Vec, BEKK, and constant correlation Diagonal Vec, are available in ARCH and GARCH configurations, with t-distribution or Normal distribution, and in-mean or not, for a total of 24 new types of time series models. FANPAC v1.1 provides new features for handling independent variables. Independent variables can now be added to the conditional variance equation, and for the multivariate models separate lists of independent variables may be specified for each mean equation. The list of independent variables can include lagged versions of the dependent variable. In the Diagonal Vec model, each element of the condition covariance matrix is modeled in a separate equation. This is efficient computationally and provides for easily interpretable coefficients. An additional efficiency can be gained by constraining the correlational matrix of the conditional covariance matrices to be constant.

3.1.1 FANPAC syntax

3.1.1.1 Programming syntax

```
new;
library fanpac;
fanset;

/* loading data */

load y[320,7] = wilshire.asc;
_fan_series = y[.,4];
Nobs = rows(_fan_series);


/* Garch(1,2) model */

_fan_p = 1;
_fan_q = 2;


/* stationarity contraint */

_nlp_IneqProc = &garch_ineq;


/* bounds for GARCH parameters */

_nlp_Bounds = { .001   1e250,  /* omega > 0    */
                0      1,      /* garch_1 >= 0 */
```

```
                -1e250 1e250,
                -1e250 1e250,
                -1e250 1e250 };

/* Analytical gradient */

_nlp_GradProc = &garch_n_grd;
_nlp_IterInfo = 1;
sv = {.2, .3, .2, .1, 1};

output file = fanpac1.out reset;

{coefs,f,g,retcode} = nlp(&garch_n,sv);

print coefs;

output off;
```

```
      0.93066861
      0.84099192
     0.010302515
      0.11565207
      0.99287855
```

3.1.1.2 Keywords syntax

```
new;
library fanpac;

session Exemple1 'wilshire';

setVarNames date cwprice cwdiv cwret ewprice ewdiv ewret;
setDataSet wilshire.asc;
setSeries cwret;

setInferenceType InvWald;
estimate modele1 garch(1,2);

output file = fanpac2.out reset;

showResults;

output off;
```

```
===============================================================================
                               Session: Exemple1
-------------------------------------------------------------------------------
                                   wilshire
-------------------------------------------------------------------------------
 FANPAC Version 1.0.0         Data Set:  wilshire        11/30/1998  9:45:16
===============================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                 Run: modele1
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

return code =    0
normal convergence


Model:  GARCH(1,2)

Number of Observations     : 320
Observations in likelihood : 318
Degrees of Freedom         : 313

AIC         1852.20
BIC         1871.01
LRS         1842.20

          roots
     ---------------

      -8.3924913
       1.0304427
       1.1890608

       Abs(roots)
     ---------------
```

```
       8.3924913
       1.0304427
       1.1890608


  unconditional variance
----------------------------
       28.149469


Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from inversion of Wald statistic

Series: cwret

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
-----------------------------------------------------------
omega          0.931       0.692      0.009       2.291
Garch1         0.841       0.048      0.747       0.862
Arch1          0.010       0.030      0.000       0.043
Arch2          0.116       0.063      0.012      -0.009
Const          0.993       0.230      0.541       1.445


Correlation Matrix of Parameters

omega          1.000    -0.587     0.131    -0.214     0.079
Garch1        -0.587     1.000     0.005    -0.534     0.142
Arch1          0.131     0.005     1.000    -0.586     0.180
Arch2         -0.214    -0.534    -0.586     1.000    -0.274
Const          0.079     0.142     0.180    -0.274     1.000
```

3.1.2 Nelson and Cao Constraints and calculation of confidence limits by inversion of the Wald statistic

The key features of GARCH modelling in FANPAC are (i) the less restrictive Nelson and Cao constraints ("Inequality Constraints in the Univariate GARCH model", Journal of Business and Economic Statistics, 10:229-235) for enforcing stationarity and the nonnegativity of the conditional variances. Most GARCH packages use unconstrained optimization methods and generally, therefore, apply more restrictive constraints on the model for computation convenience. FANPAC provides for the calculation of confidence limits by inversion of the Wald statistic. For models with a single parameter near a constraint boundary, the method incorporated in FANPAC gives correct results (see Schoenberg, R. "Constrained Maximum Likelihood", in Computational Economics, 10:251-266, 1997). Results for models with more than one parameter in the region of boundaries will be correct provided none of them are correlated more than about 0.6 with each other.

```
/*
**  This example illustrates how to estimate garch models
**  using the procedures rather than the keyword commands.
**  This allows you to provide nonstandard constraints
**  or other conditions to the model.  In this example
**  the Nelson & Cao constraints are compared with the
**  traditional constraints.
*/


   library fanpac;
   fanset;    /* resets globals to default values */
              /* when the command file is re-run  */


   output file = fanpac3.out reset;

   _fan_p = 1;   /*  garch(1,3) model */
   _fan_q = 2;

   _fan_series = loadd(''example'');

   nobs = rows(_fan_Series);


   start = { .2,    /* omega */
             .3,    /* garch_1 */
             .2,    /* arch_1 */
             .1,    /* arch_2 */
              1 };  /* constant */
```

```
 _nlp_GradProc = &garch_n_grd;  /* gradient proc */


/*
**  Nelson and Cao constraints
*/

_nlp_IneqProc = &garch_ineq;  /* inequality constraints */

_nlp_Bounds = { .001   1e250,  /* omega > 0    */
                 0     1,      /* garch_1 >= 0 */
                -1e250 1e250,
                -1e250 1e250,
                -1e250 1e250 };

{ coefs,fct,grad,rcode } = nlp(&garch_n,start);

print;
print;
print ''       Nelson & Cao constraints'';
print;

if rcode <= 2;
    /* covariance matrix of parameters */
    h = _nlp_CovPar(coefs,&garch_n,&garch_n_grd,nobs,1);
else;
    h = error(0);
endif;

format /rd 12,4;
if not scalmiss(h);
    alpha = .05;  /* 95% cl's */

    dv = cdftci(0.5*alpha,nobs-rows(coefs))*sqrt(diag(h));
    print ''confidence limits from standard errors'';
    print;
    print ''  Coefficients    lower cl    upper cl'';
    print coefs~(coefs-dv)~(coefs+dv);
    print;

    /* confidence limits by inversion of Wald statistic   */
    sel = 0;      /* selection vector, if zero, all cl's computed */
    cl = _nlp_climits(coefs,h,nobs,alpha,sel);
endif;

format /rd 12,4;
if not scalmiss(cl) and not scalmiss(h);

    print ''confidence limits from inversion of Wald statistic'';
    print;
    print ''  Coefficients    lower cl    upper cl'';
    print coefs~cl;

else;

    print ''  Coefficients'';
    print coefs;

endif;


/*
** standard constraints
*/


_nlp_IneqProc = {.};

_nlp_c = { 0 -1 -1 -1 0 };   /* garch + arch < 1 */
_nlp_d = { -1 };

_nlp_Bounds = { .001   1, /* omega > 0    */
                 0     1,     /* garch_1 >= 0 */
                 0     1,     /* arch_1 >= 0  */
                 0     1,     /* arch_2 >= 0  */
                -1e250 1e250 };



{ coefs,fct,grad,rcode } = nlp(&garch_n,start);

print;
print;
print ''       standard constraints'';
```

```
   if rcode <= 2;
       /* covariance matrix of parameters */
       h = _nlp_CovPar(coefs,&garch_n,&garch_n_grd,nobs,1);
   else;
       h = error(0);
   endif;


   format /rd 12,4;
   if not scalmiss(h);
       alpha = .05;  /* 95% cl's */

       dv = cdftci(0.5*alpha,nobs-rows(coefs))*sqrt(diag(h));
       print;
       print ''confidence limits from standard errors'';
       print;
       print ''  Coefficients    lower cl    upper cl'';
       print coefs~(coefs-dv)~(coefs+dv);
       print;

       /* confidence limits by inversion of Wald statistic   */
       sel = 0;      /* selection vector, if zero, all cl's computed */
       cl = _nlp_climits(coefs,h,nobs,alpha,sel);
   endif;

   format /rd 12,4;
   if not scalmiss(cl) and not scalmiss(h);
       print;
       print ''confidence limits from inversion of Wald statistic'';
       print ''  Coefficients    lower cl    upper cl'';
       print coefs~cl;

   else;

       print ''  Coefficients'';
       print coefs;

   endif;

   output off;


       Nelson & Cao constraints

confidence limits from standard errors

  Coefficients    lower cl    upper cl

      0.1486      -0.0151       0.3123
      0.6314       0.3430       0.9197
      0.3255       0.1023       0.5488
     -0.0670      -0.3188       0.1848
      0.5435       0.4399       0.6472

confidence limits from inversion of Wald statistic

  Coefficients    lower cl    upper cl

      0.1486       0.0114       0.3123
      0.6314       0.3896       0.6670
      0.3255       0.1384       0.4160
     -0.0670      -0.2055      -0.1234
      0.5435       0.4399       0.6472


       standard constraints

confidence limits from standard errors

  Coefficients    lower cl    upper cl

      0.1770       0.0058       0.3483
      0.5687       0.3315       0.8059
      0.3011       0.1019       0.5003
      0.0000            .            .
      0.5457       0.4424       0.6490


confidence limits from inversion of Wald statistic
  Coefficients    lower cl    upper cl

      0.1770       0.0334       0.3483
      0.5687       0.3315       0.6989
      0.3011       0.1341       0.4313
      0.0000            .            .
```

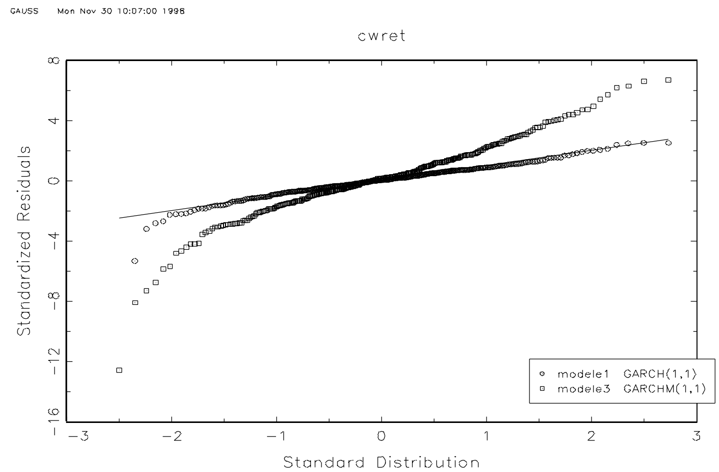
```
     0.5457       0.4424       0.6490
```

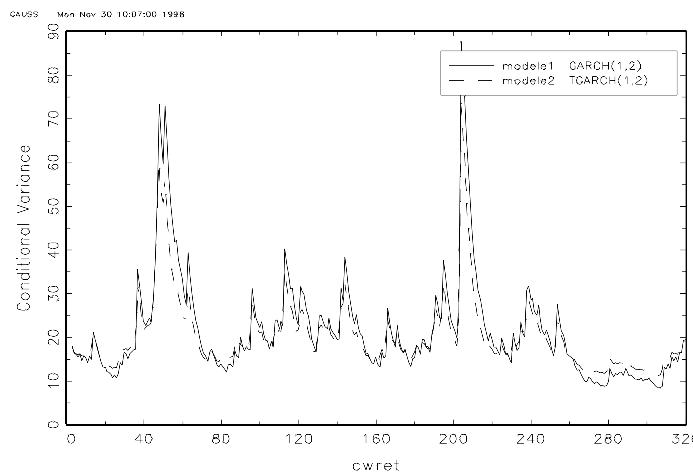
3.1.3 Univariate models

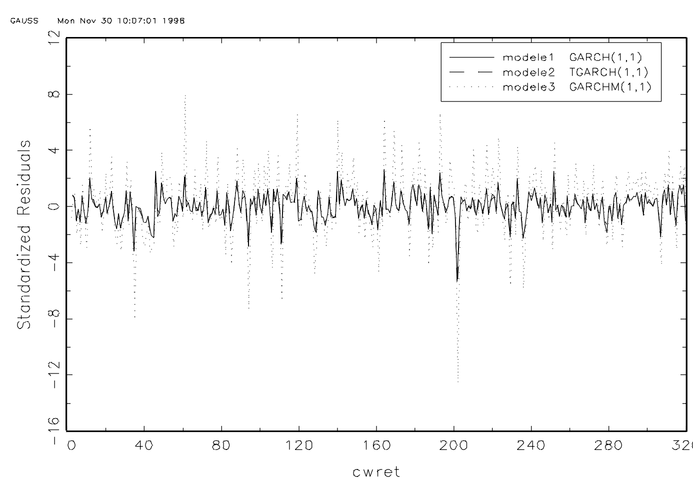
```
new;
library fanpac,pgraph;

session Exemple 'wilshire';

setVarNames date cwprice cwdiv cwret ewprice ewdiv ewret;
setDataSet wilshire.asc;
setSeries cwret;

plotSeries;
plotSeriesACF;
plotSeriesPACF;

setInferenceType InvWald;

estimate model1 garch(1,2);
estimate model2 tgarch(1,2);
estimate model3 garchm(1,1);

ShowResults model2;

testSR;

plotQQ model1 model3;
plotCV model1 model2;
plotSR;
```

```
==============================================================================
                              Session: Exemple
------------------------------------------------------------------------------
                                  wilshire
------------------------------------------------------------------------------
 FANPAC Version 1.0.0         Data Set:  wilshire       11/30/1998 10:59:16
==============================================================================


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                 Run: model2
------------------------------------------------------------------------------
------------------------------------------------------------------------------

return code =    0
normal convergence


Model:  TGARCH(1,2)

Number of Observations     : 320
Observations in likelihood : 318
Degrees of Freedom         : 312

AIC        1835.20
BIC        1857.78
LRS        1823.20

          roots
     ---------------

    -12.7474
      1.0967
      1.2478

       Abs(roots)
     ---------------

     12.7474
      1.0967
      1.2478


  unconditional variance
----------------------------
     11.9517


Maximum likelihood covariance matrix of parameters
0.95 confidence limits computed from inversion of Wald statistic
```

```
Series: cwret

Parameters  Estimates   Standard     Lower       Upper
                         Errors      Limits      Limits
------------------------------------------------------------
omega          1.991      1.507      0.000       4.956
Garch1         0.801      0.102      0.601       0.887
Arch1          0.032      0.048      0.000       0.111
Arch2          0.072      0.084     -0.026       0.161
Const          1.176      0.225      0.733       1.619
Nu             6.242      1.998      2.946      10.173


Correlation Matrix of Parameters

omega        1.000   -0.840    0.059    0.100   -0.036   -0.134
Garch1      -0.840    1.000    0.014   -0.501    0.086   -0.007
Arch1        0.059    0.014    1.000   -0.636    0.133   -0.088
Arch2        0.100   -0.501   -0.636    1.000   -0.154    0.037
Const       -0.036    0.086    0.133   -0.154    1.000   -0.064
Nu          -0.134   -0.007   -0.088    0.037   -0.064    1.000


================================================================================
                               Session: Exemple
--------------------------------------------------------------------------------
                                   wilshire
--------------------------------------------------------------------------------

                   Time Series

==================================================
                  Series: cwret
--------------------------------------------------
       skew      -266.1720   pr =      0.000
   kurtosis      8558.5534   pr =      0.000

   heteroskedastic-consistent
   Ljung/Box       39.0881   pr =      0.124
--------------------------------------------------


                    Residuals

==================================================
              model1: GARCH(1,2)
--------------------------------------------------
       skew        -4.2151   pr =      0.040
   kurtosis         8.8383   pr =      0.003

   heteroskedastic-consistent
   Ljung/Box       17.5967   pr =      0.965

==================================================
              model2: TGARCH(1,2)
--------------------------------------------------
       skew        -4.4968   pr =      0.034
   kurtosis        11.2495   pr =      0.001

   heteroskedastic-consistent
   Ljung/Box       19.1844   pr =      0.936

==================================================
              model3: GARCHM(1,1)
--------------------------------------------------
       skew       -42.3702   pr =      0.000
   kurtosis       698.1698   pr =      0.000

   heteroskedastic-consistent
   Ljung/Box       19.0158   pr =      0.940
--------------------------------------------------
```







3.1.4 Multivariate models

▶ Not yet done.

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 $\beta = \operatorname{vec}\begin{bmatrix} \Phi_1 & \Theta_1 \end{bmatrix}$. 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 $\operatorname{vec}\begin{bmatrix} \beta & \mathbb{P}^\star \end{bmatrix}$ with $\mathbb{P}^\star = \operatorname{vech}(\mathbb{P})$ 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

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

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 $\mathrm{var}(u_t) = 1$) 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

$$\begin{cases} y_t &= \mu_t + \varepsilon_t \\ \mu_t &= \mu_{t-1} + \eta_t \end{cases}$$

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

$$\left\{\begin{array}{lcl} z_t & = & x_t + y_t \\ x_t & = & \phi_1 x_{t-1} + u_t \\ y_t & = & v_t - \theta_1 v_{t-1} \end{array}\right.$$

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

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

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

$$\left\{\begin{array}{lcl} z_t & = & x_t + y_t \\ x_t & = & x_{t-1} + u_t - \theta_1 u_{t-1} \\ y_t & = & v_t \end{array}\right.$$

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

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

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 $t = 1, \dots, N$

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

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

$$\left\{ \begin{array}{ccc} \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} & \simeq & \begin{bmatrix} x_{1,t} & 0 & x_{3,t} \\ 0 & x_{2,t} & x_{3,t} \end{bmatrix} \begin{bmatrix} \beta_{1,t} \\ \beta_{2,t} \\ \beta_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \beta_{1,t+1} \\ \beta_{2,t+1} \\ \beta_{3,t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{1,t} \\ \beta_{2,t} \\ \beta_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{array} \right.$$

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

$$\left\{ \begin{array}{ccc} y_t & = & s_t^y + c_t + u_t^y \\ x_t & = & s_t^x + c_t + u_t^x \end{array} \right.$$

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

$$\left\{ \begin{array}{ccc} \begin{bmatrix} y_t \\ x_t \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \\ \alpha_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \alpha_{1,t+1} \\ \alpha_{2,t+1} \\ \alpha_{3,t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{1,t} \\ \alpha_{2,t} \\ \alpha_{3,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{array} \right.$$

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:

$$\left\{ \begin{array}{ccc} y_t & \simeq & \beta_t \\ \beta_{t+1} & \simeq & \beta_t \end{array} \right.$$

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:

$$\left\{ \begin{array}{rcl} y_t & \simeq & \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} \delta_{t+1} \\ \beta_{t+1} \end{bmatrix} & \simeq & \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{array} \right.$$

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 $\theta = \operatorname{vec} \begin{bmatrix} \beta & \sigma \end{bmatrix}$ be the vector of parameters. We estimate θ with GMM by considering the moment conditions

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

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

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

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

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

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

$$\left\{ \begin{array}{rcl} dx_t & = & \mu x_t \, dt + \sigma x_t \, dW_t \\ x(t_0) & = & x_0 \end{array} \right. \tag{3.11}$$

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 $\theta = \operatorname{vec}\begin{bmatrix} \mu & \sigma \end{bmatrix}$ 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

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

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

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

The solution of the stochastic differential equation (3.12) is

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

We define

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

Let $\theta = \operatorname{vec}\begin{bmatrix} a & b & \sigma \end{bmatrix}$ 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

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

87. **gmm3c.prg**
CHAN, KAROLYI, LONGSTAFF and SANDERS [1992] consider the following stochastic differential equation

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

To estimate the vector of parameters $\theta = \operatorname{vec}\begin{bmatrix} \alpha & \beta & \gamma & \sigma \end{bmatrix}$, 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

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

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

$$\begin{cases} \alpha &= ab \\ \beta &= -a \end{cases}$$

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

$$\left\{\begin{array}{l}\left[\begin{array}{c} y_{1,t} \\ y_2, t \end{array}\right] = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]\left[\begin{array}{c} \alpha_t \\ \beta_t \end{array}\right] + \left[\begin{array}{c} 10 \\ 0 \end{array}\right] + \varepsilon_t \\ \left[\begin{array}{c} \alpha_t \\ \beta_t \end{array}\right] = \left[\begin{array}{cc} 0.5 & 0.3 \\ 0 & 0.2 \end{array}\right]\left[\begin{array}{c} \alpha_{t-1} \\ \beta_{t-1} \end{array}\right] + \left[\begin{array}{c} 1 \\ 4 \end{array}\right] + \left[\begin{array}{c} 1 \\ 1 \end{array}\right]\eta_t \end{array}\right. \tag{3.15}$$

with

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

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

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

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

$$\left\{ \begin{array}{l} y_t = \begin{bmatrix} 1 & x_t & t \end{bmatrix} \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} + \varepsilon_t \\ \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_{0,t} \\ \eta_{1,t} \\ \eta_{2,t} \end{bmatrix} \end{array} \right. \tag{3.19}$$

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

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

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

$$\left\{ \begin{array}{l} y_t = \left[\begin{array}{ccc} x_t & 0 & 1 \end{array} \right] \left[\begin{array}{c} \beta_t \\ \eta_t \\ \varepsilon_t \end{array} \right] \\ \left[\begin{array}{c} \beta_t \\ \eta_t \\ \varepsilon_t \end{array} \right] = \left[\begin{array}{ccc} \phi_1 & -\theta_1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array} \right] \left[\begin{array}{c} \beta_{t-1} \\ \eta_{t-1} \\ \varepsilon_{t-1} \end{array} \right] + \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{array} \right] \left[\begin{array}{c} \eta_t \\ \varepsilon_t \end{array} \right] \end{array} \right.$$

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

$$\left\{ \begin{array}{l} z_t = \left[\begin{array}{cc} 1 & 0 \end{array} \right] \left[\begin{array}{c} y_t \\ u_t \end{array} \right] + \varepsilon_t \\ \left[\begin{array}{c} y_t \\ u_t \end{array} \right] = \left[\begin{array}{cc} \phi_1 & -\theta_1 \\ 0 & 0 \end{array} \right] \left[\begin{array}{c} y_{t-1} \\ u_{t-1} \end{array} \right] + \left[\begin{array}{c} 1 \\ 1 \end{array} \right] u_t \end{array} \right. \tag{3.24}$$

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

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

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

$$\left[\begin{array}{c} \phi_1 \\ \theta_1 \\ \sigma_u \\ \sigma_\varepsilon \end{array} \right] = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] \left[\begin{array}{c} \phi_1 \\ \sigma_u \\ \sigma_\varepsilon \end{array} \right] + \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \tag{3.26}$$

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

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

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 $m = 1, \ldots, 10$

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

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

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

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

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

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

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

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

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

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

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

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:

$$\left\{ \begin{array}{rcl} \sum_{k=0}^{p} h_k & = & \sqrt{2} \\ \sum_{k=0}^{p} g_k & = & 0 \\ \sum_{k=0}^{p} h_k^2 & = & 1 \\ \sum_{k=0}^{p} h_k g_k & = & 0 \end{array} \right.$$

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

$$\left\{\begin{array}{l} \left[\begin{array}{c} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{array}\right]=\left[\begin{array}{cc} 1 & 1 \\ 4 & 2 \\ 2 & -3 \end{array}\right]\left[\begin{array}{c} \alpha_{1,t} \\ \alpha_{2,t} \end{array}\right]+\varepsilon_t \\ \left[\begin{array}{c} \alpha_{1,t} \\ \alpha_{2,t} \end{array}\right]=\left[\begin{array}{cc} .5 & .45 \\ -.5 & .8 \end{array}\right]\left[\begin{array}{c} \alpha_{1,t} \\ \alpha_{2,t} \end{array}\right]+\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]\left[\begin{array}{c} \eta_{1,t} \\ \eta_{2,t} \end{array}\right] \end{array}\right. \tag{3.32}$$

with

$$\varepsilon_t \sim \mathcal{N}\left(\mathbf{0}_3,\left[\begin{array}{ccc} 5 & 1 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 8 \end{array}\right]\right)$$

and

$$\left[\begin{array}{c} \eta_{1,t} \\ \eta_{2,t} \end{array}\right] \sim \mathcal{N}\left(\mathbf{0}_2,\left[\begin{array}{cc} 2 & 0.5 \\ 0.5 & 1 \end{array}\right]\right)$$

We compute the responses to the forecast error $\mathbf{e}=\left[\begin{array}{cc} 1 & 0 \end{array}\right]^{\top}$.

189. **ssm8b.prg**
We compute the responses to the forecast error $\mathbf{e}=\left[\begin{array}{cc} 1 & -1 \end{array}\right]^{\top}$ for the state space model (3.32).

190. **ssm9a.prg**
We compute the responses to the orthogonal impulse $\mathbf{e}=\left[\begin{array}{cc} 1 & 0 \end{array}\right]^{\top}$ for the state space model (3.32).

191. **ssm9b.prg**
We compute the responses to the orthogonal impulse $\mathbf{e}=\left[\begin{array}{cc} 1 & -1 \end{array}\right]^{\top}$ 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

$$\left\{ \begin{array}{lcl} \frac{dx}{dt} & = & \sigma (y - x) \\ \frac{dy}{dt} & = & -xz + Rx - y \\ \frac{dz}{dt} & = & xy - \beta z \end{array} \right.$$

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

$$\begin{array}{rcl} x_t & = & \left[\beta_1 + (\beta_2 + \beta_3 x_{t-1}) \exp\left(-\delta x_{t-1}^2\right)\right] x_{t-1} \\ & & \left[\beta_4 + (\beta_5 + \beta_6 x_{t-1}) \exp\left(-\delta x_{t-2}^2\right)\right] x_{t-2} + u_t \end{array}$$

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

$$\begin{bmatrix} \mathrm{vec}\,(\Phi_1) \\ \mathrm{vec}\,(\Phi_2) \\ \mu \end{bmatrix}$$

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

$$\begin{bmatrix} \mathrm{INC}_t \\ \mathrm{CONS}_t \end{bmatrix} = \Phi_1 \begin{bmatrix} \mathrm{INC}_{t-1} \\ \mathrm{CONS}_{t-1} \end{bmatrix} + \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \mathrm{INV}_{t-1} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} \tag{3.35}$$

213. **varx2b.prg**
Estimation of the Constrained Dynamic Simultaneous Equations

$$\begin{bmatrix} \mathrm{INC}_t \\ \mathrm{CONS}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} \mathrm{INC}_{t-1} \\ \mathrm{CONS}_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \mu_2 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ 0 \end{bmatrix} \mathrm{INV}_{t-1} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} \tag{3.36}$$

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 **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 = 0~1~1~1~1~1~1~1;
    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;
```

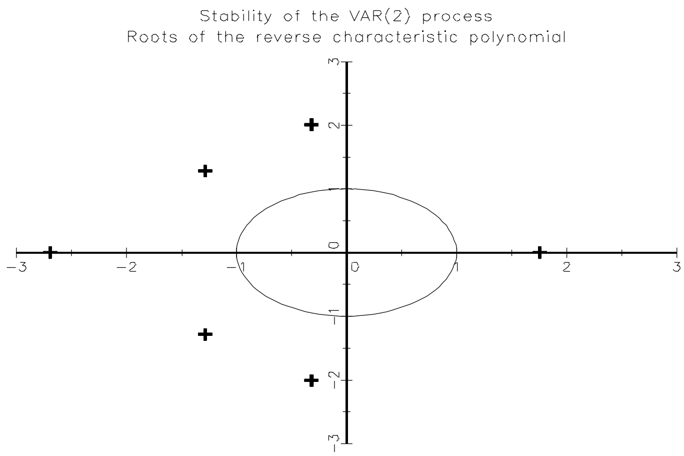
```
------------------------------------------------------------------------------
Lags     BIC      AICa     AICc       SIC        FPE      AIC       HQ
------------------------------------------------------------------------------
Opt.      1         2        2         1          2        2         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.

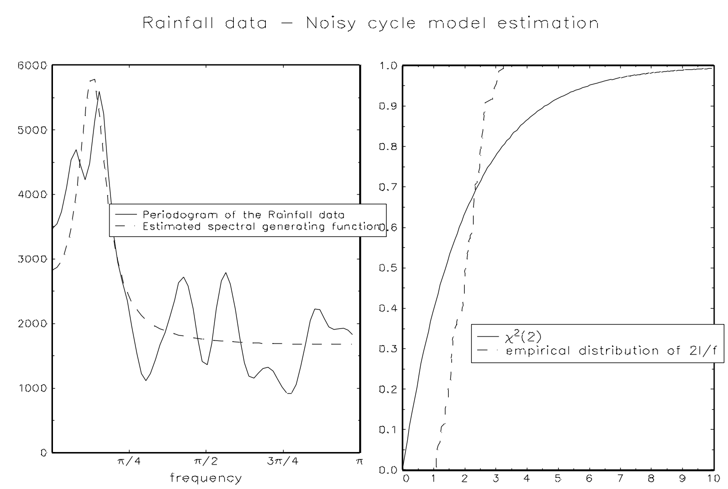
```
/*
**  LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
**  Springer-Verlag, Berlin-Heidelberg
**
**  Estimation of a System of Dynamic Simultaneous Equations
**  See LUTKEPOHL, chapter 10
**
*/

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;
```

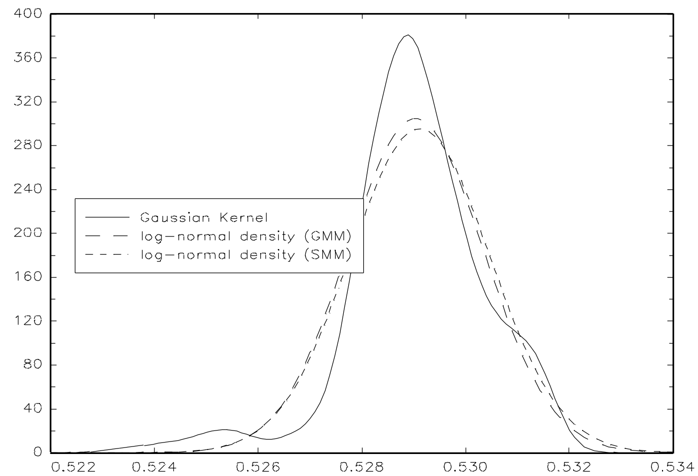
```
@<--- data generation --->@

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

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

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

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

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;
```

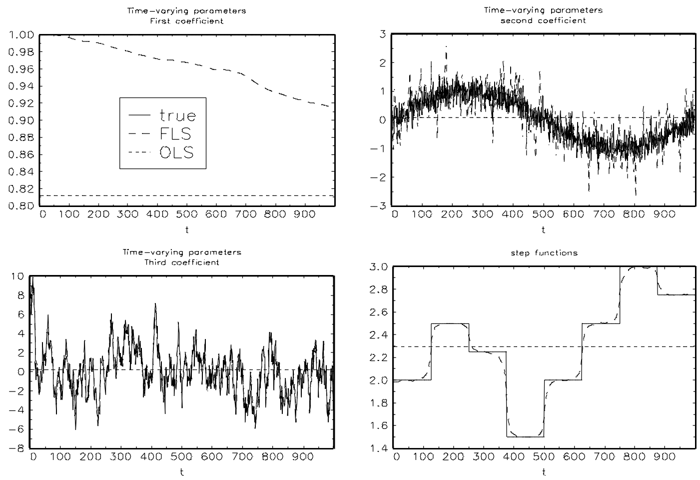
```
proc sgf(coeff,lambda);
  local phi1,theta1,sigma_u,sigma_v;
  local w,g;
  phi1 = coeff[1];
  sigma_u = coeff[2];
  theta1 = coeff[3];
  sigma_v = coeff[4];
  w = cos(lambda);
  g = (sigma_u^2)./(1-2*phi1*w+phi1^2) +
      (sigma_v^2).*(1-2*theta1*w+theta1^2);
  retp(g);
endp;

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;
```



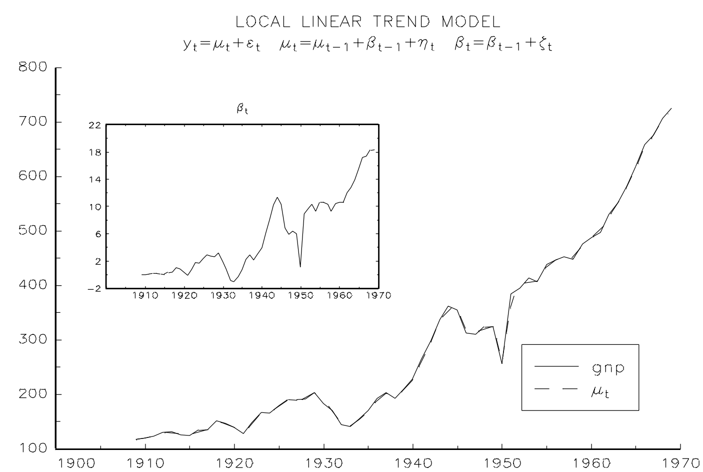
3.2.4 State-space modelling

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);
```

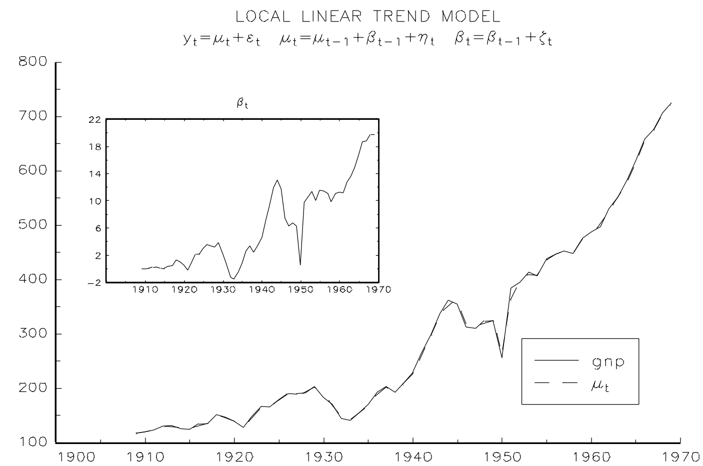
```
    _pframe = 0;
    title(''LOCAL LINEAR TREND MODEL''\
          ''\Ly]t[=\202m\201]t[+\202e\201]t[  ''\
          ''\202m\201]t[=\202m\201]t-1[+\202b\201]t-1[+\202c\201]t[  ''\
          ''\202b\201]t[=\202b\201]t-1[+\202z\201]t['');
    _plegctl = {2 6 6.5 1};
    _plegstr = ''gnp\000\202m\201]t['';
    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 --->@
```

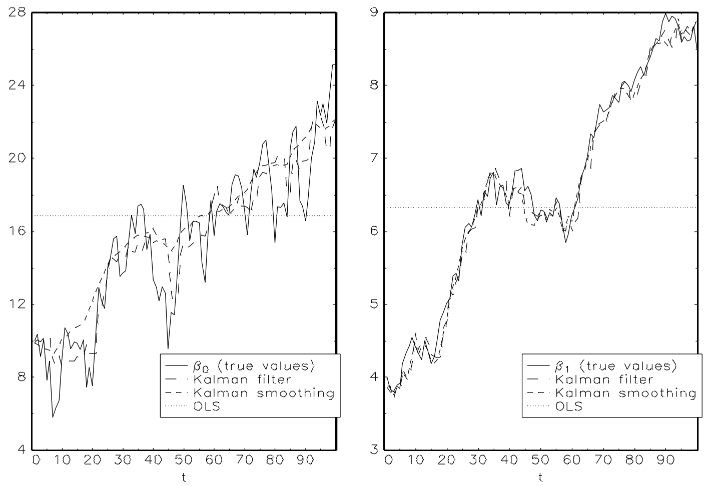
```
Z = 1~0; d = 0;
T = {1 1,0 1}; c = 0|0; R = {1 0,0 1};
a0 = gnp[1]|0; P0 = zeros(2,2);

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;
```

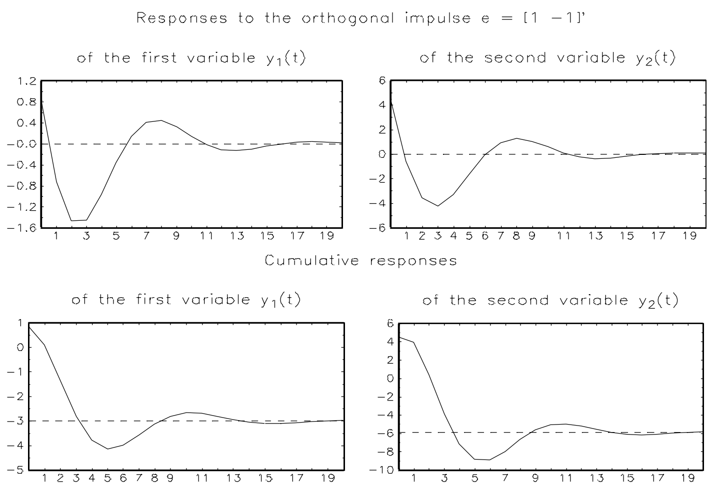
```
sv = arma1|vech_(chol(arma2));    /* Starting values = CMLE parameters */

{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)

-------------------------------------------------------------------------------
                      Investment          Income      Consumption
         periods     innovations     innovations      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)

-------------------------------------------------------------------------------
                      Investment          Income      Consumption
         periods     innovations     innovations      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
```

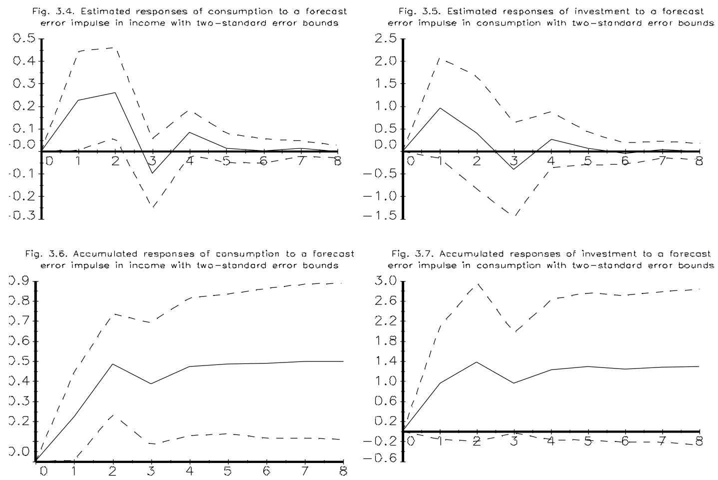
```
           7           6.9222          89.1158           3.9620
           8           6.9228          89.1149           3.9623


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;
```

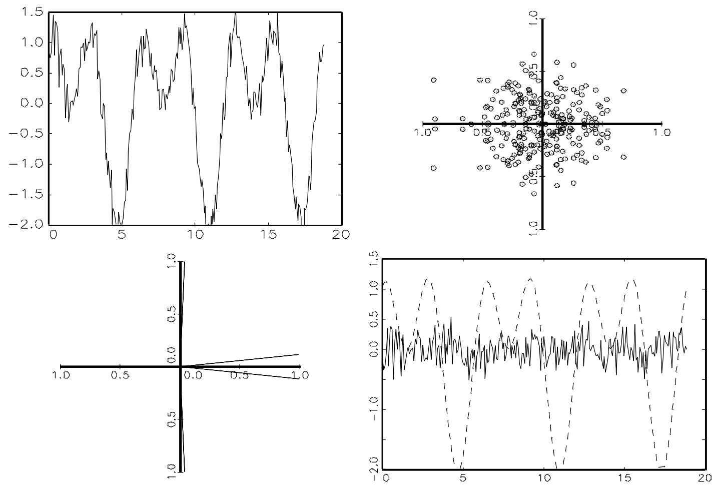


3.2.5 Spectral methods

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.

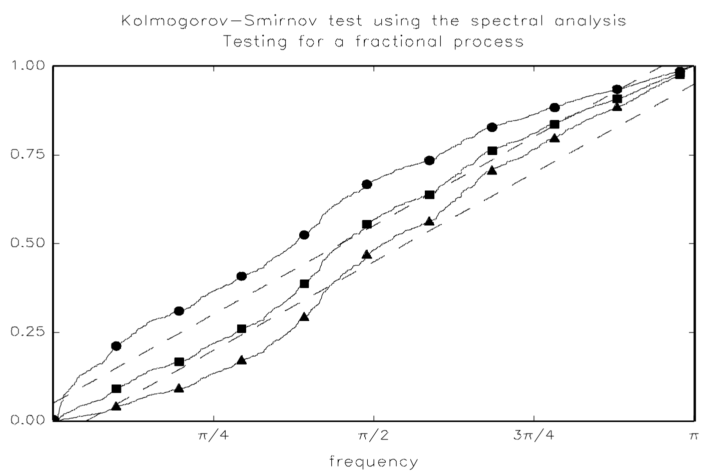
3.2.5.3 Surrogate data

▶ 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''\
```

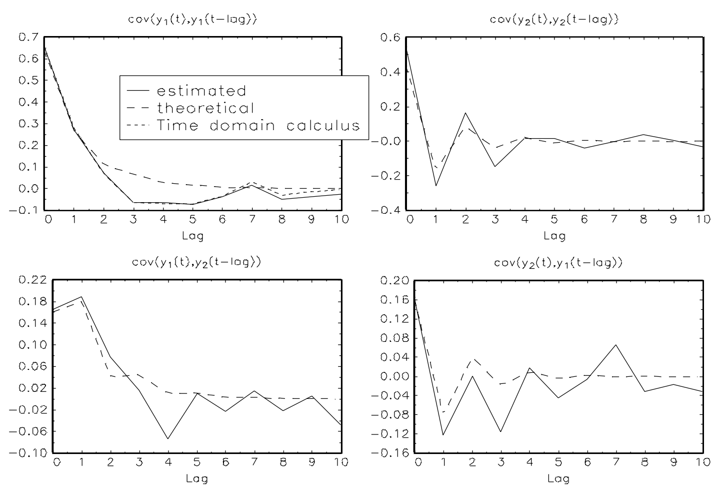
```
      ''\LTesting for a fractional process'');
  _pdate = ''''; _pnum = 2;
  _pltype = 6|6|6|1|1;
  _pstype = 8|9|10|1|1;
  _plctrl = 100|100|100|0|0;
  fonts(''simplex simgrma'');
  xtics(0,pi,pi/4,0); ytics(0,1,0.25,0);
  lab = '' 0 \202p\201/4 \202p\201/2 \2013\202p\201/4 \202p\201'';
  asclabel(lab,0);
  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
```

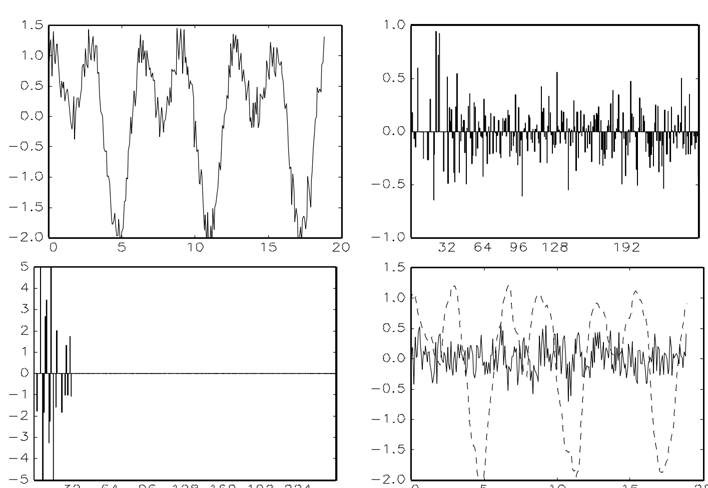
```
*/

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;
```

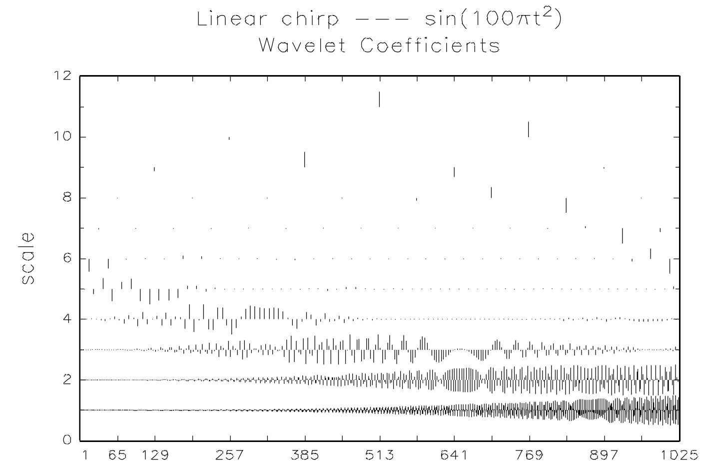
```
TSMset;


@<--- 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;
```

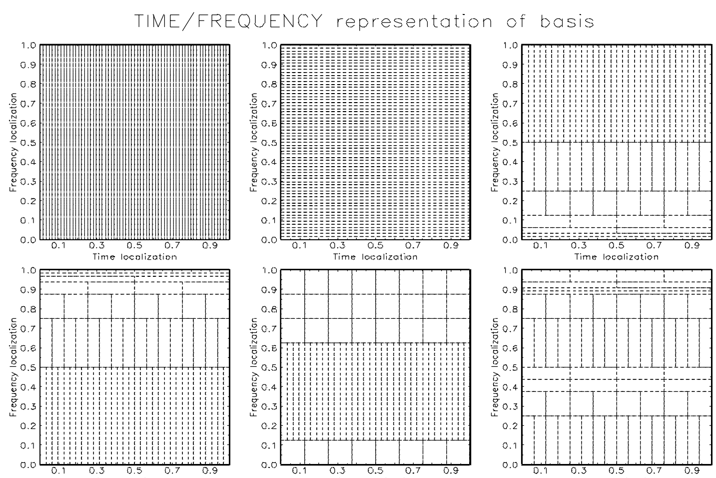


3.2.6.2 Wavelet and Wavelet Packets representation

```
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;
```

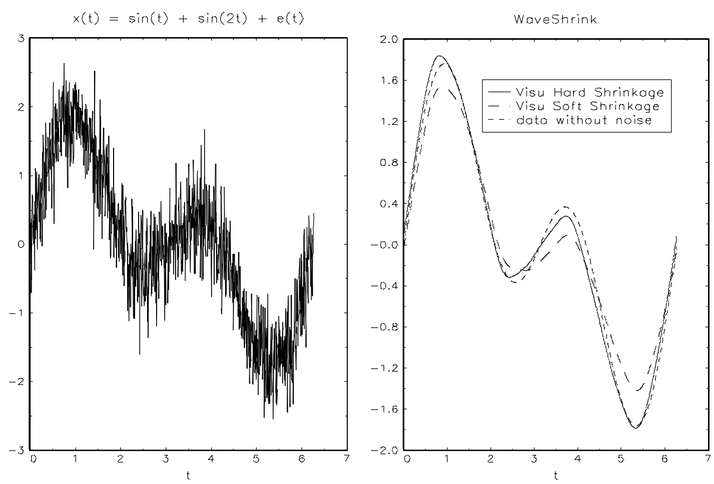




3.2.6.3 Data denoising

```
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);
```

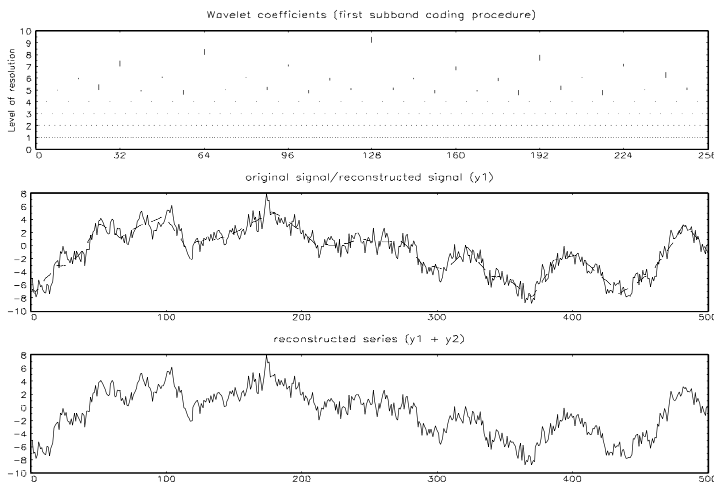
```
y = y1~y2;

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_0_CompileString(str);
dllcall /r GAUSS_0_Execute;
dllcall GAUSS_0_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_0_Initialize;

str = ''y = rndu(3,3); x = y[1,1];'';

dllcall /r GAUSS_0_CompileString(str);
dllcall /r GAUSS_0_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_0_CompileString(str);
dllcall /r GAUSS_0_Execute;
dllcall GAUSS_0_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;
```

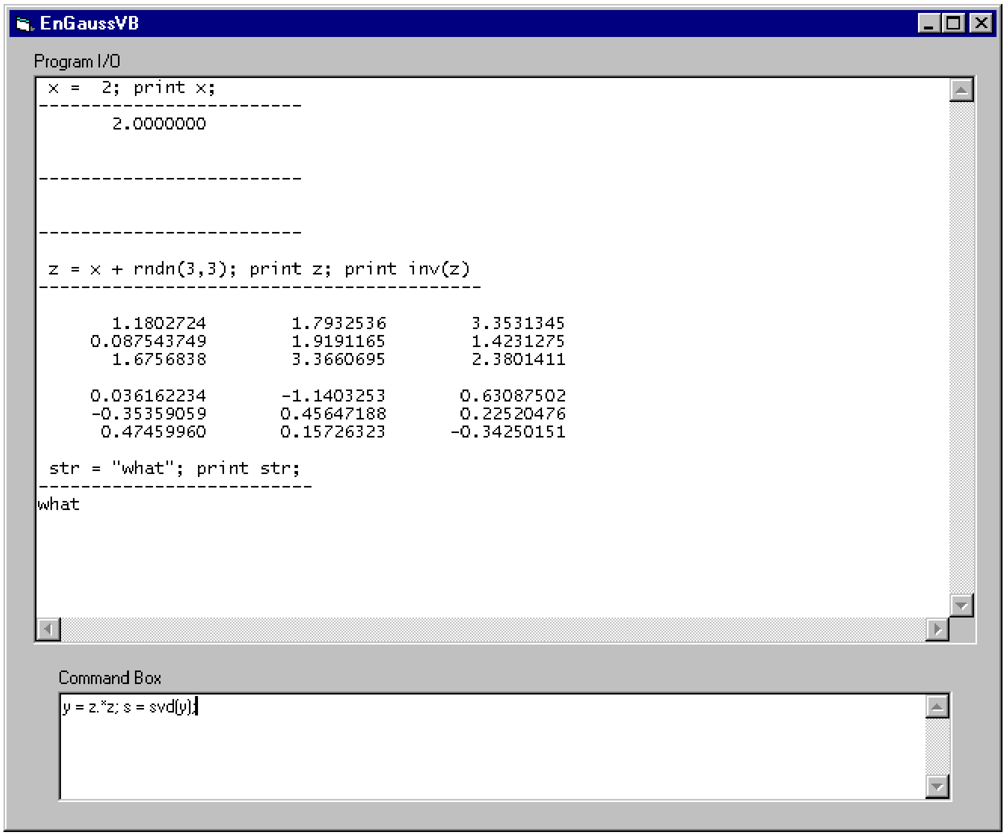
▶ **Gauss Engine is an open system and allows communication with the application.**

```
#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:

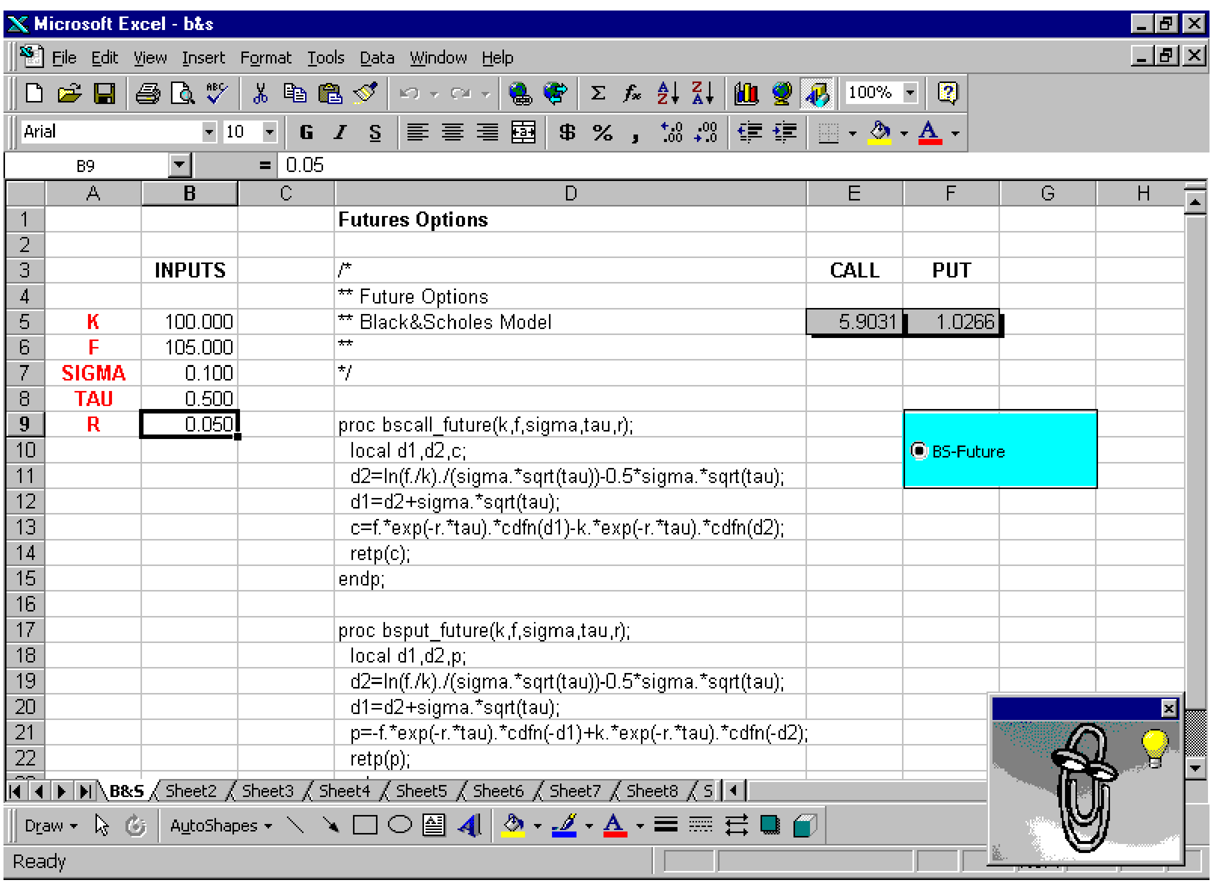
- **geclose()**: Closes the Gauss Engine.
- **geexec(ByVal strn As String)**: Compiles and executes a string buffer or a file (or cells of the spreadsheet).
- **gematget(ByVal strng As String, ByRef target() As Double)**: Gets matrices from the Gauss symbol table.
- **gematput(ByVal strng As String, 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.

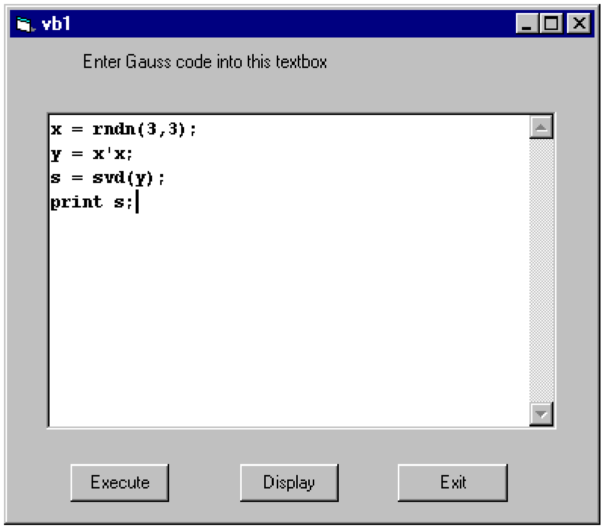
4.2.2 Some Mercury_GE examples

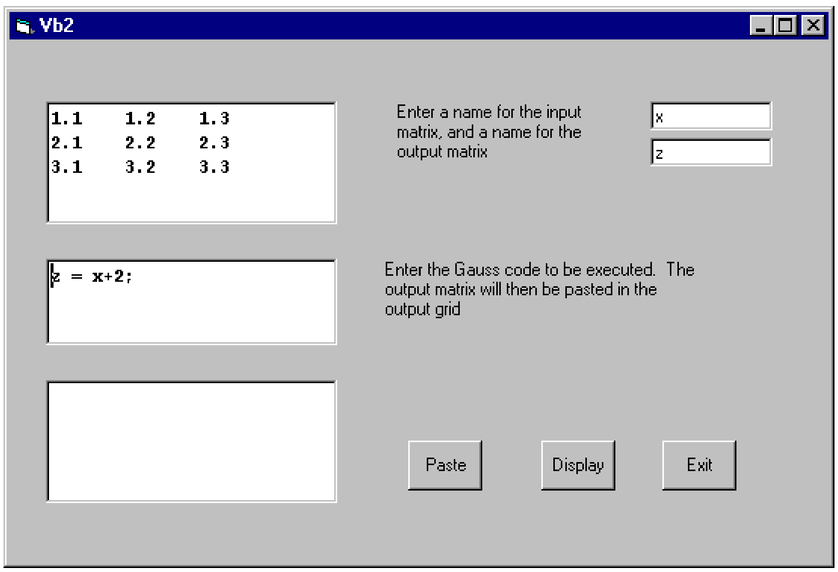
4.2.2.1 Excel example

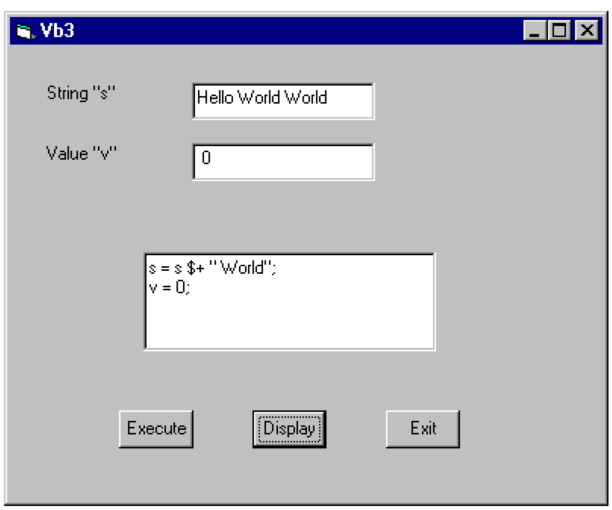
```
'*********************************************************

'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





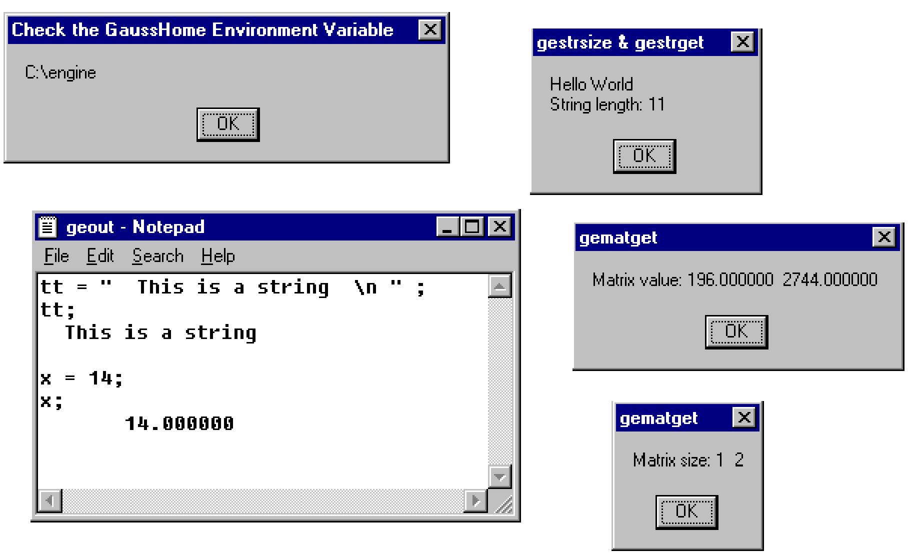


4.2.2.2 VC++ example

```
// cprg1.cpp : Defines the entry point for the application.
//
```

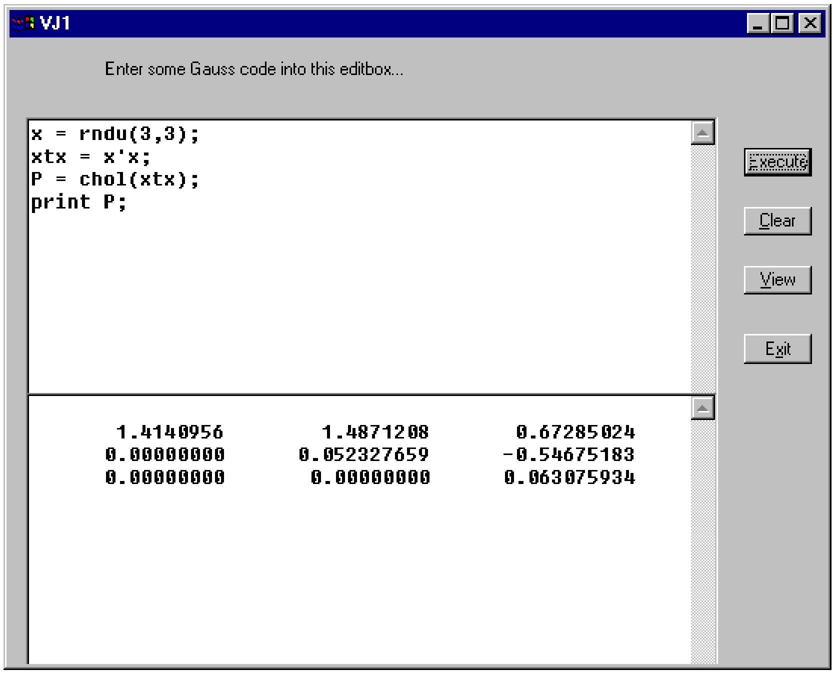
```
#include ''stdafx.h''


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;
}
```



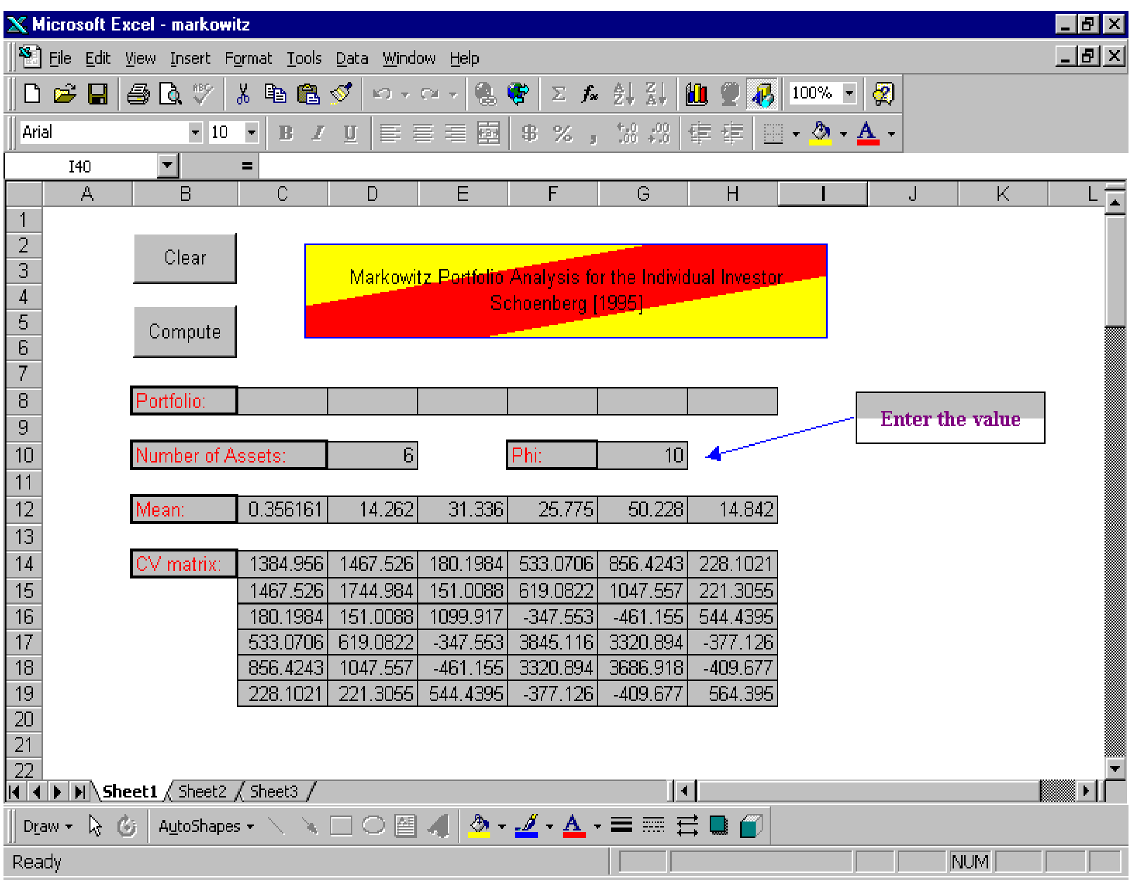
4.2.2.3 VJ++ example

- Needs VM Java component.



4.2.3 Three explained examples

4.2.3.1 Markowitz portfolio and Excel

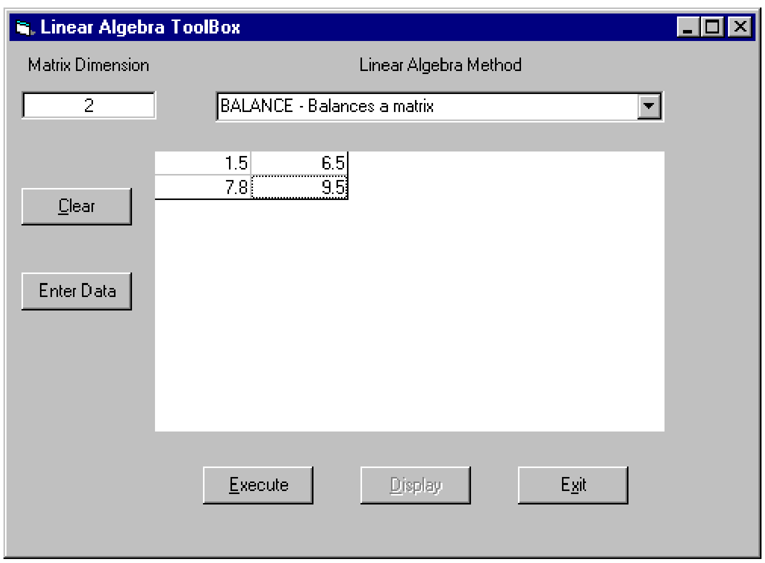


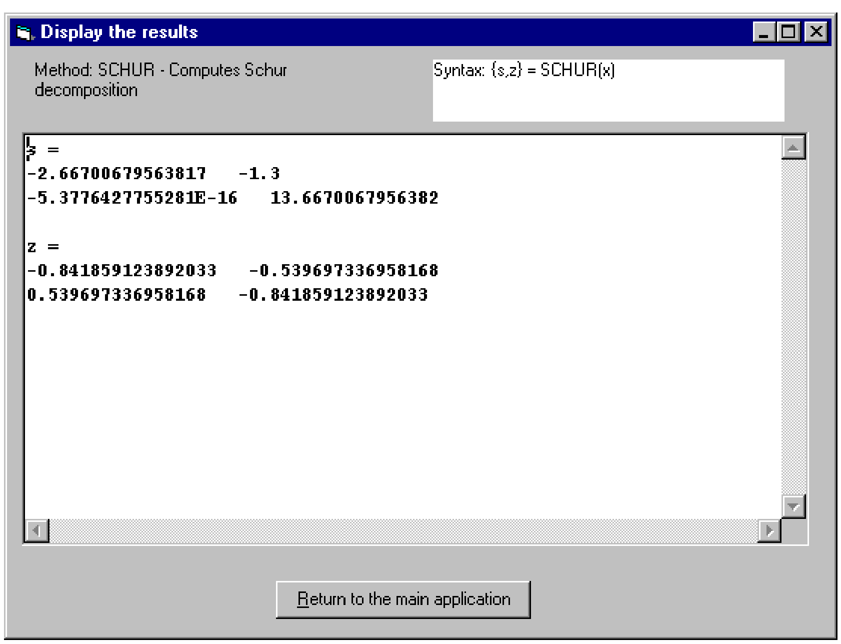
► The VBA code is:

```
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()
```

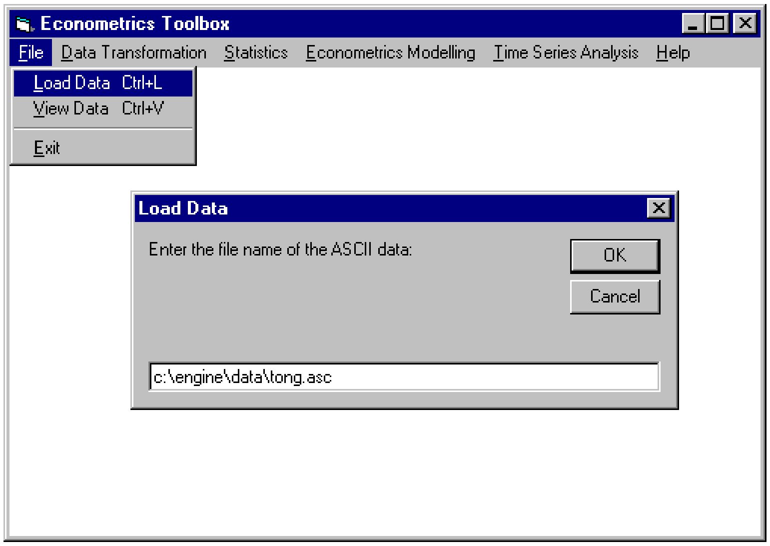
```
            nr = UBound(OutputMatrix, 1)
            nc = UBound(OutputMatrix, 2)
            For i = 1 To nr Step 1
                For j = 1 To nc Step 1
                    txt = txt & OutputMatrix(i, j) & ''   ''
                Next j
                txt = txt & Chr(13) & Chr(10)
            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
```

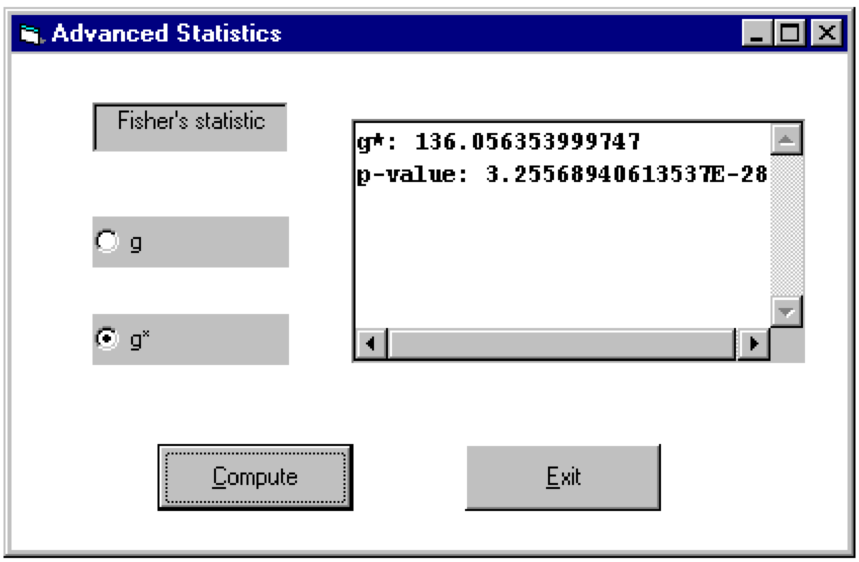
▶ The Gauss programs are:

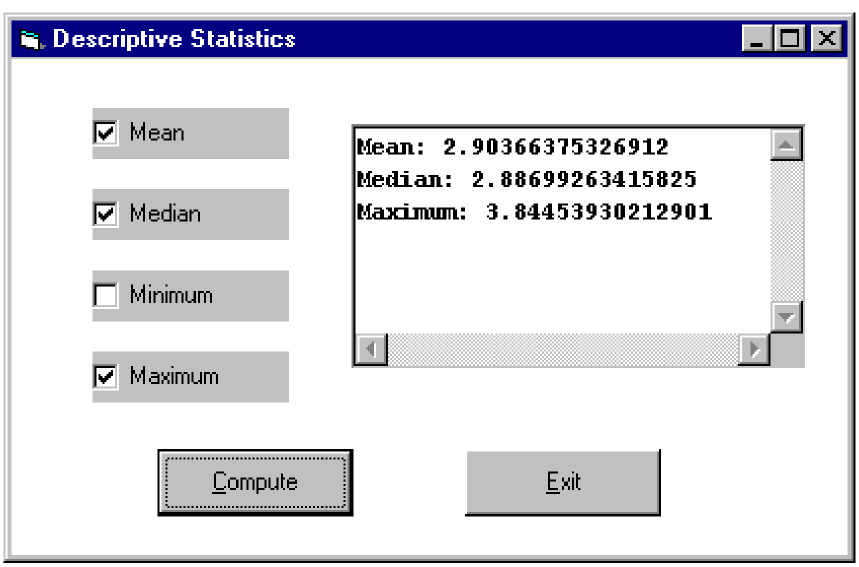
```
---------------------------------------------
-> BALANCE.PRG

method = ''BALANCE - Balances a matrix'';
syntax = ''{b,z} = BALANCE(x)'';
```

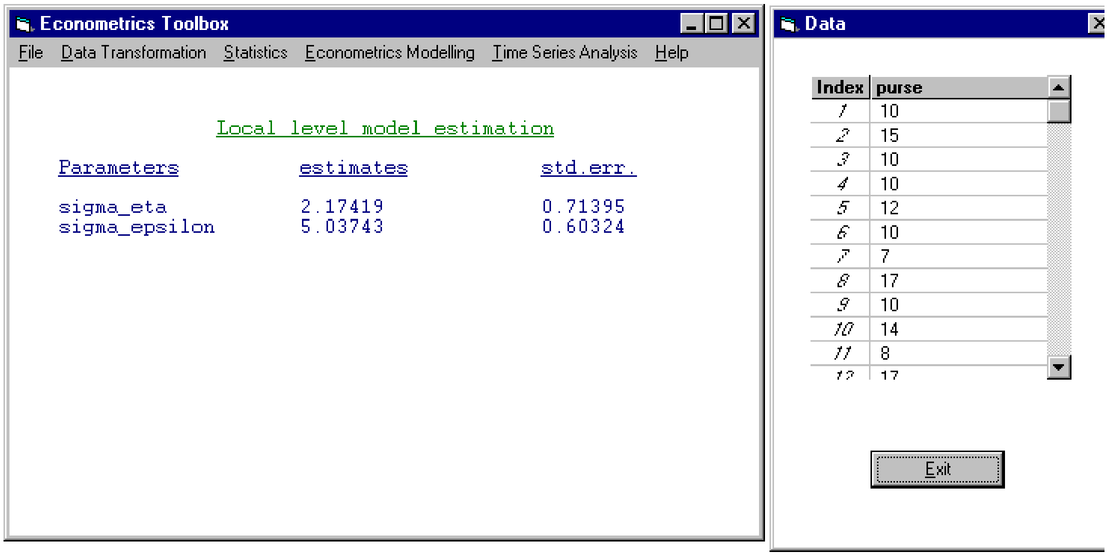
```
{b,z} = BALANCE(x);
--------------------------------------------
-> COND.PRG

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