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

1 Introduction to GAUSS  
  1.1 What is Gauss? ........................................... 1  
  1.1.1 The Language ........................................ 1  
  1.1.2 The Graphics ....................................... 1  
  1.2 What is new in the NT/95 version? ....................... 1  
  1.2.1 PlayW ............................................... 3  
  1.2.2 On-line help ....................................... 4  
  1.3 Value at Risk examples ................................ 5  
  1.3.1 Asset portfolio .................................... 5  
  1.3.2 Simulating methods ................................ 5  
  1.3.3 VaR on options .................................... 6  
  1.3.4 Derivatives portfolio ............................... 7  
  1.3.5 Non-linearity in VaR ............................... 8  
  1.4 Creating DLLs for Gauss ................................. 10  
  1.4.1 C/C++ source code ................................ 10  
    1.4.1.1 lcc compiler (LCC-WIN32) ..................... 10  
    1.4.1.2 gcc/g++ compilers (GNU-WIN32) ............... 13  
    1.4.1.3 gcc compiler (RSX NT) ........................ 13  
  1.4.2 Fortran code ....................................... 13  
    1.4.2.1 dvf/fps compilers ............................. 13  
    1.4.2.2 g77 compiler (RSX NT) ....................... 13  
  1.5 Win32 APIs access .................................... 14  
  1.5.1 The C code ....................................... 14  
  1.5.2 The Gauss win32api.src file ....................... 15  
  1.5.3 An example ....................................... 16  
  1.6 DDE functionality ................................... 16  
    1.6.1 Using Mercury and Visual Basic ................... 16  
    1.6.2 Using temporary files ............................. 19  
      1.6.2.1 A very simple example ...................... 19  
      1.6.2.2 Working with matrices ...................... 21  
      1.6.2.3 An option pricer ........................... 23  
    1.6.3 Using a disk buffer ................................ 23  
      1.6.3.1 A very simple example ...................... 28  
      1.6.3.2 Risk neutral density estimation .......... 30  

2 Financial Modelling ........................................ 35  
  2.1 The PDE2D library .................................... 35  
    2.1.1 What is PDE2D? .................................. 35  
    2.1.2 Some examples .................................. 35  
      2.1.2.1 Parabolic problems ......................... 35  
      2.1.2.2 Elliptic problems ........................... 37  
      2.1.2.3 Computing Greeks ......................... 40  
      2.1.2.4 Computing stochastic volatility option greeks coefficients .................... 41  
    2.1.2.5 A Dixit-Pindyck example ..................... 44  
    2.1.2.6 Other financial problems .................... 46
2.2 The OPTION library ...................................................... 46
  2.2.1 What is OPTION? .................................................. 46
  2.2.2 Some examples .................................................... 49
    2.2.2.1 Cox, Ross and Rubinstein model ......................... 49
    2.2.2.2 Black-Scholes Gamma computing ....................... 50
    2.2.2.3 Option pricing with jump-diffusion processes ....... 51
    2.2.2.4 Stochastic volatility option models .................. 53
    2.2.2.5 Option pricing with subordinated stochastic processes 54
    2.2.2.6 Implied volatility and transaction frequency - An application to Libor 3 months future options .......... 55

3 Financial Econometrics ................................................. 57
  3.1 FANPAC .................................................................. 57
    3.1.1 FANPAC syntax .................................................. 57
      3.1.1.1 Programming syntax ..................................... 57
      3.1.1.2 Keywords syntax ....................................... 58
    3.1.2 Nelson and Cao Constraints and calculation of confidence limits by inversion of the Wald statistic ........................................ 59
    3.1.3 Univariate models .............................................. 62
    3.1.4 Multivariate models ........................................... 64
  3.2 TSM .................................................................. 65
    3.2.1 TSM examples ................................................... 68
    3.2.2 Time series ...................................................... 91
      3.2.2.1 Arfima process .............................................. 91
      3.2.2.2 Varma process ............................................. 92
      3.2.2.3 Varx Process ............................................. 97
      3.2.2.4 Structural models ......................................... 103
    3.2.3 Estimation methods ............................................ 104
      3.2.3.1 Maximum likelihood ..................................... 104
      3.2.3.2 Generalized method of moments ...................... 105
      3.2.3.3 Simulated method of moments ......................... 106
      3.2.3.4 Whittle method .......................................... 108
      3.2.3.5 Generalized flexible least squares .................. 109
    3.2.4 State-space modelling ........................................ 111
      3.2.4.1 Structural models ....................................... 111
      3.2.4.2 Time-varying parameters ................................ 114
      3.2.4.3 Multivariate model ..................................... 116
      3.2.4.4 Exact maximum likelihood ............................. 116
      3.2.4.5 Impulse functions ....................................... 119
      3.2.4.6 Bootstrap methods ....................................... 122
    3.2.5 Spectral methods .............................................. 124
      3.2.5.1 Understanding the Fourier transform .................. 124
      3.2.5.2 Periodogram, spectral estimation and cross-spectrum analysis ........................................ 125
      3.2.5.3 Surrogate data ........................................... 125
      3.2.5.4 Kolmogorov-Smirnov test ................................ 125
      3.2.5.5 Covariance functions ................................... 126
      3.2.5.6 Varma parameters estimation in the frequency domain .................. 127
    3.2.6 Wavelets analysis .............................................. 128
      3.2.6.1 Understanding the Wavelet and Wavelet Packets transforms .......... 128
      3.2.6.2 Wavelet and Wavelet Packets representation .......... 130
      3.2.6.3 Data denoising .......................................... 131
      3.2.6.4 Subbands coding ....................................... 132
      3.2.6.5 Scalogram .............................................. 133
      3.2.6.6 A financial example ..................................... 134
4 Developing professional applications using Gauss programs
   4.1 The Gauss Engine .................................................. 135
      4.1.1 What is the Gauss Engine? .................................... 135
      4.1.2 Understanding the Gauss Engine ......................... 135
      4.1.3 Some examples with Excel ................................. 137
      4.1.4 Some examples with Visual Basic ....................... 138
   4.2 Mercury_GE ......................................................... 138
      4.2.1 What is Mercury_GE? ......................................... 138
      4.2.2 Some Mercury_GE examples ............................... 139
         4.2.2.1 Excel example ...................................... 139
         4.2.2.2 VC++ example ...................................... 140
         4.2.2.3 VJ++ example ...................................... 142
      4.2.3 Three explained examples ................................ 143
         4.2.3.1 Markowitz portfolio and Excel .................. 143
         4.2.3.2 Linear algebra and Visual Basic ................ 144
         4.2.3.3 Creating an Econometrics ToolBox ............. 152
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 Exchange feature for exchanging data with most Windows spreadsheets and databases. GAUSS matrices and data sets can be exported in a variety of file formats. Files generated by other software can be imported as matrices 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 non-zero 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 \texttt{cdftvn( )}. Now they are only limited by computational time and RAM. Functions are included for the following:

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

- Other New Features

  1. New Random Number Generators
1.2. WHAT IS NEW IN THE NT/95 VERSION?

(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


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) `inand(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) `ishift(a,s)` Shift bits s positions (+ve or -ve)

1.2.1 PlayW

PlayW is a new graphic utility for converting Gauss .gdf 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

![Gauss Browser](image)

- 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 asset 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 = [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 fton(VaR,'Value at Risk (0.05) = %f 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; S = -1;
Portfolio_t0 = 4;

Nobs = 2500;
Portfolio_t1 = rndm(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, 1);)

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

proc (1) = rndm(mu, SIGMA, Ns);
local dim, u, Pchol;
local oldTran;
dim = rows(mu);
oldTran = transchk(1);
trap 1, 1;
Pchol = chol(SIGMA);'
trap oldTran, 1;
if scalar(Pchol);'
ERROR100 'error: SIGMA is not a positive definite matrix.';
retl(error(0));
endif;
u = mu + Pchol*rndn(dim, Ns);
retu(u);
endp;

Estimated Quantiles
------------------------
1% -5.4215644
2.5% -5.5141676
5% -4.911986
Median -3.6000266
95% -2.330682
97.5% -2.080657
99% -1.789327

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 mmProc(t, S);
  retp(b*S);
endp;
proc sigmaProc(t, S);
  retp(sigma*S);
endp;
C0 = EuropeanBS(S0, K, sigma, tau, b, r);
Ns = 2000;
1.3. VALUE AT RISK EXAMPLES

```c
{t,S} = Simulate_SDE(S0, &muProc, &sigmaProc, 0, abs(deltas Tau), 20, Ns);
S1 = S[20 , ];
Cl = EuropeanBS(S1, K, sigma, tau, b, r);
deltaC = Cl - C0;
alpha = 0.05;
output file = var3.out reset;

/* Long position */
deltaC = Cl - C0;
VaR = abs(quantile(deltaC, alpha));
print ('Long Position');
print ('------------------');
print fts(var, 'Value at Risk (0.05) = %lf Francs', 5, 4);
print;

/* Short Position */
deltaC = -(Cl - C0);
alpha = 0.05;
VaR = abs(quantile(deltaC, alpha));
print ('Short Position');
print ('------------------');
print fts(var, 'Value at Risk (0.05) = %lf Francs', 5, 4);
print;

/* Long Position --- DELTA-GAMMA approach */
dels = 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 fts(var, 'Value at Risk (0.05) = %lf Francs', 5, 4);
output off;
```

---

<table>
<thead>
<tr>
<th>Long Position</th>
<th>Value at Risk (0.05) = 3.1836 Francs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Position</td>
<td>Value at Risk (0.05) = 3.1924 Francs</td>
</tr>
<tr>
<td>Long Position --- DELTA-GAMMA approach</td>
<td>Value at Risk (0.05) = 3.1961 Francs</td>
</tr>
</tbody>
</table>

1.3.4 Derivatives portfolio

- An example with derivatives portfolio.

```c
new;
library option, tmm, optrmm, pgraph;
rndseed 123;
S0 = 100; K = 100; sigma = 0.10; r = 0.08; b = r; N = 4;
deltas Tau = -7/365; /* 7 days */
theta = 6 | -2 ;
/* Call American */
tau1 = 90/365;
/* Put Lock-back */
tau2 = 160/365;
proc mProc(t, S);
  retp( b*S );
endP;
```
1.3.5 Non-linearity in VaR

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

```gauss
proc sigmProc(t,S);
   retP ( sigma*S );
endp;

/* t0 */
_type_Option = 'call';
Call = AmericanCRN(S0,K,sigma,tau1,b,r,N);

_type_Option = 'put';
PutLookBack = LookBackCRN(S0,sigma,tau2,b,r,N);
Portfolio_t0 = theta[1]*Call + theta[2]*PutLookBack;

/* t1 */
Nt = 1000;
{t,S} = Simulate_SDE(S0,AmProc,sigProc,0,abs(deltaTau),S0,Nt);
S1 = S[200,:];

_type_Option = 'call';
Call = AmericanCRN(S1,K,sigma,tau1,b,r,N);

_type_Option = 'put';
PutLookBack = LookBackCRN(S1,sigma,tau2,b,r,N);
Portfolio_t1 = theta[1]*Call + theta[2]*PutLookBack;

deltaP = Portfolio_t1 - Portfolio_t0;
{x,d,f,retcode} = Kernel(deltasP);

getplot;
   _plate = ''; _pnum = 2;
   fonts('simplex simgnum');
   title('Q01PDF\$Q02DQ01P$');
   graphprt('out -CF -CF=var4.eps');
   xy(x,d);
```

![PDF(DP)](image-url)
1.3. VALUE AT RISK EXAMPLES

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

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

/* t0 */
_type_option = 'call';
Call0 = 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,mmProc,sigmaProc,0,abs(deltaTau),20,Ns);
S1 = S[20, ];

_type_option = 'call';
Call = AmericanCRR(S1,K,sigma,tau1,b,r,N);

_type_option = 'put';
PutLookback = LookbackCRR(S1,sigma,tau2,b,r,N);
theta2 = seqs(-10,1,21);
alpha = 0.10|0.05;
VaR1 = {};
VaR2 = {};
i = -10;
do until i > 10;
  Portfolio_t0 = i*Call + theta2 . * PutLookback0;
  Portfolio_t1 = i*Call + theta2 . * PutLookback;
  deltaP = Portfolio_t1 - Portfolio_t0;
  VaR = abs(quantile(deltaP,alphas));
  VaR1 = VaR1 | VaR[1, ];
  VaR2 = VaR2 | VaR[2, ];
i = i + 1;
enddo;

graphset;
  begwind;
  window(2,2,0);
    _plate = ' ';
    _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);
    _pnum = 0;
    _pframe = 0;
    _pnum = 0;
    _ptitleht = 0.50;
    title('VaR at 10%');
    draw;
  setwind(3);
    _pnum = 0;
    _pframe = 0;
    title('VaR at 5% -- >');
    draw;
  setwind(4);
    _pnum = 2;
    _pnumht = 0.25;
    _paxht = 0.25;
    xlab('Look-Back Put');
    ylab('American Call');
    surface(theta2,theta2,VaR2);
  graphprt('c=s -cf=var5.eps');
  endwind;
```
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:


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

$$\begin{align*}
\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= d
\end{align*}
$$

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[0] = b[0]; dprime[0] = d[0];

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;
end;

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;
end;
retp(x);
endp;
The corresponding C subroutine is:

```c
/* --- The following code comes from c:\cc\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
// lpReserved - 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 lpReserved)
{
    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, much 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;
}
```

The file `c:\cc\lib\wizard\dll.tpl` is added to the C file, because the file .dll needs an entry point to load/unload DLLs.
Remark 1 Arguments are passed to \texttt{dllTDGSolve} by reference.

- We create the \texttt{tridiag.dll} file with the following commands

  
  ```
  loc tridiag.c
  lec tridiag.obj
  ```

- The \texttt{tridiag.ex} 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 \texttt{_dllTDGSolve} in Gauss, we declare the \texttt{tridiag.dll} file with \texttt{dlibrary}. Because the arguments are passed by reference, it is very important to declare them correctly before calling the subroutine with the \texttt{dllcall} command.

  ```
  new;
  library mlib;
  dlibrary tridiag.dll;
  N = 5;
  a = rmdn(N,1); b = rmdn(N,1); c = rmdn(N,1);
  abc = TDGmatrix_to_Dmatrix(a,b,c);
  d = abc*seq(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 = tridiag.dat reset;
  print x1' x2;
  output off;
  ```
1.4. CREATING DLLS FOR GAUSS

\[
\begin{array}{ll}
1.0000000 & 1.0000000 \\
2.0000000 & 2.0000000 \\
3.0000000 & 3.0000000 \\
4.0000000 & 4.0000000 \\
5.0000000 & 5.0000000 \\
\end{array}
\]

A better solution is to create a Gauss procedure:

```c
/*
** d1lTDGSolve
*/

proc d1lTDGSolve(a,b,c,d);
  local N,x,bprime,dprime;
  N = rows(d);
  x = zeros(N,1);
  bprime = zeros(N,1);
  dprime = zeros(N,1);
  d1lcall_d1lTDGSolve(a,b,c,d,N,x,bprime,dprime);
  retp(x);
endp;
```

An example of using the Gauss d1lTDGSolve procedure.

```c
def;
library mlib, tridiag;
dlibrary tridiag.dll;

N = 5;
a = randn(N,1); b = randn(N,1); c = randn(N,1);
abc = TDMatrix_to_DMatrix(a,b,c);
d = abc+seq(1,1,N);
x = d1lTDGSolve(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 d8f/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 address:

http://weber.u.washington.edu/~rons/f2c.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 CREATE_DIRECTORY function, delete/move files with the kernel32.dll DELETE_FILE/MOVE_FILE function, scroll windows (ScrollWindow) or play sounds with the winmm.dll PLAY_SOUND function.

1.5.1 The C code

```c
#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
// lpReserved - 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.
// boolean WINAPI LibMain(HINSTANCE hDLLInst, DWORD fdwReason, LPVOID lpReserved)
{
    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, much 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;
    hWnd = FindWindowEx(0, _hWnd, _lpszClassName, NULL);
    if (hwnd)
    {
        return TRUE;
    }
    return FALSE;
}
```
1.5. WIN32 APIs Access

FlashWindow(hwnd, bInvert);
return 0;
}

declspec(dllexport) int dl1MessageBox(double *ret, char *lpText, char *lpCaption, double *wType)
{
    char _lpText[256];
    char _lpCaption[256];
    strcpy(_lpText, lpText);
    strcpy(_lpCaption, lpCaption);
    *ret = MessageBox(HWND, _lpText, _lpCaption, wType);
    return 0;
}

declspec(dllexport) int dl1MoveWindow(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 dl1ShowWindow(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);
1.5.3 An example

new;
dlibrary win32spi;
library win32spi;
ret = MessageBox('Do you want to quit Gauss ?','GAUSS Messenger 1',1);
if ret == 1;
call MessageBox('Sorry, but the program will continue','GAUSS Messenger 2',0);
else;
call MessageBox('Ok, the program will continue','GAUSS Messenger 2',0);
endif;
call MessageBox('MoveWindow Example','GAUSS Messenger 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 cmdstring As String
    Dim hWnd As Long
    GaussPath = txtGaussPath.Text
    FileName = txtProgramPath.Text
    cmdstring = GaussPath & " & FileName
    Call gauss_load(cmdstring)
    hWnd = winhandle(\"GAUSS\")
    If hWnd > 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 Sub

Private Sub Form_Load()
    ChoTypeOption.AddItem "call"
    ChoTypeOption.AddItem "put"
    ChoTypeOption.ListIndex = 0
    cmdSendQuery.Enabled = False
    cmdDisplay.Enabled = False
    numSendQuery.Enabled = False
    numDisplay.Enabled = False
End Sub

Private Sub numAbout_Click()
    About.Show
End Sub

Private Sub numDisplay_Click()
    cmdDisplay = True
End Sub

Private Sub numExit_Click()
    cmdExit = True
End Sub

Private Sub numLoadGauss_Click()
    cmdLoadGauss = True
End Sub

Private Sub numSendQuery_Click()
    cmdSendQuery = True
End Sub

Private Sub numTheModel_Click()
    TheModel.Show
End Sub

Private Sub tmrStatus_Timer()
    Dim hwid As Long
    Dim gaussState As String
    hwid = winhandle("GAUSS")
    If hwid = 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 hwid As Long
    hwid = winhandle("GAUSS")
    If hwid > 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_sppname = &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");
1.6. DDE FUNCTIONALITY

\[
\begin{align*}
  \text{w} &= \sigma \cdot \exp(\text{tau}); \\
  \text{d1} &= \left( \ln(S_0/K) + b \cdot \text{tau} \right)/\text{w} + 0.5 \cdot \text{w}; \\
  \text{d2} &= \text{d1} - \text{w}; \\
  \text{if lower(OptionType) } &\text{ then} ' \text{put'}; \\
  \text{Premium} &= -S_0 \cdot \exp((-r) \cdot \text{tau}) \cdot \Phi(\text{d1}) + K \cdot \exp(-r \cdot \text{tau}) \cdot \Phi(\text{d2}); \\
  \text{else; } \\
  \text{Premium} &= S_0 \cdot \exp((-r) \cdot \text{tau}) \cdot \Phi(\text{d1}) - K \cdot \exp(-r \cdot \text{tau}) \cdot \Phi(\text{d2}); \\
  \text{endif; }
\end{align*}
\]

```vba
if lower(OptionType) = 'put'
  Premium = -s0 * exp((-r) * tau) * phi(d1) + k * exp(-r * tau) * phi(d2)
else
  Premium = s0 * exp((-r) * tau) * phi(d1) - k * exp(-r * tau) * phi(d2)
endif
```

call sendvalue('BS_premium', Premium)
ret;
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 $\mathcal{N}(\mu, \sigma^2)$. We compute them with Gauss from an Excel interface. The sheet looks like the picture below:

![Excel sheet](image)

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

- Execute Gauss $\iff$ Execute_Gauss()
- Display Results $\iff$ Display_Results()

- The VBA code is the following:

```vba
Dim GaussPath As String
Sub Execute_Gauss()
  Dim GaussProgram As String
  GaussPath = "C:\gauss"'
  GaussProgram = GaussPath + '\gauss.exe' -
  + GaussPath + '\conf3\all.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 adj 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 = mysetate(2,0);
cmd = ChangeDir(GaussPath $+$ 'conf3');
tmpfile = '\xl1_in.tmp';
load data[] = tmpfile;
u = data[1]; sigma = data[2];
Nobs = data[3];
output file = tmpfile reset;
screen off;
print u;
output off;
system;

Remark 2 The program could be improved by using the SHELEXECUTE 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:

```vba
Dim GaussPath As String
Sub Execute_Gauss()
    Dim GaussProgram As String
    GaussPath = "d:\gauss"
    GaussProgram = GaussPath & "\gauss.exe " + GaussPath & "\conf3\x12.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\x12_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\x12_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 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 = mysetate(2, 0);
cml = ChangeDir(GaussPath $"conf3");
tmpfile = '"x12_in.tmp"';
load data[] = 'tmpfile';

Nrows = data[1];
Ncols = data[2];
M = reshape(tri2tri(data, 2, 0), Nrows, Ncols);
old = trapchk(1);
trap 1,1;
Mchol = chol(M);
trap old,1;
tmpfile = '"x12_out.tmp"';
output file = 'tmpfile reset;
screen off;
if scalar(Mchol);
print '"-999"';
else;
print vec(Mchol);
endif;
output off;
system;
1.6. DDE FUNCTIONNALITY

1.6.2.3 An option pricer

![Excel interface with European BS formulas and data]

1.6.3 Using a disk buffer

When you are working with different matrices, 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 **VB gauss.src** file is

```vbnet
proc (1) = VBclear (diskbuffer);
    local file1, file2, f1, f2;
    if string (diskbuffer, ":", 1) == 0;
        diskbuffer = ChDir (0) $":" $"\" $+ diskbuffer;
    endif;

    file1 = diskbuffer $"\", "inf";
    file2 = diskbuffer $"\", "asc";

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

    if (f1 == 0) or (f2 == 0);
        ERROR 1""error: could not clear the disk buffer""
        retp (0);
    endif;

    fclose (f1);
    fclose (f2);

    retp (1);
endp;
```

```vbnet
proc (1) = VBput (diskbuffer, x, name);
```
local file1, file2, f1, f2;
local r, c, N, str, i;

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

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

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

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

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

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

call fseek(f1, 0, 2);
call fputstr(f1, str);
call fseek(f2, 0, 2);
x = vec(x);
i = 1;
do until i > N;
   str = ftostr(x[i], '%lf', 20, 10);
call fputstr(f2, str);
i = i + 1;
enddo;

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

ret(1);
ewoo;

proc (1) = VRead(diskbuffer, name);
local file1, file2, f1, f2;
local pos, i, str, name_r, c, x;
local errCode, erString;

if strstr(buffer, '::', 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 end(f1);
   str = fgets(f1, 1);
   {name_r, c} = VToken(str);
   if name_r $== name;
      errCode = 0;
      break;
   endif;
   pos = pos + r*c;
enddo;

if errCode;
   erString = 'error: ' $+ name $+ ' not found';
   ERRORLOG erString;
endif;
1.6. DDE FUNCTIONNALITY

```text
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) = VBiniFile(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 = strindex(str, '', '', 1);
str = strput('""""", str, pos);
pos = strindex(str, '', '', 1);
str = strput('""""", str, pos);
pos = strindex(str, '"\"\"\", 1);
str = strput('""""", str, pos);
pos = strindex(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 VBGaussBas file is ---
Public Function GaussDir() As String
GaussDir = "d:\gauss\"
End Function

Public Sub ExecuteGauss(ByVal GaussProgam 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

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

Public Sub VbPut(ByVal diskbuffer As String, ByVal 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 

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

End Sub 

Public Sub VbRead(ByVal diskbuffer As String, ByVal name As String, _
    ByVal 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

    file1 = FreeFile
1.6. DDE FUNCTIONALITY

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, _
    ByVal r As Integer, _
    ByVal c 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

errorMsg:
MsgBox "error: MatrixToCells"
Exit Sub
End Sub

Public Sub ClearCells(ByVal Sheet As String, ByVal minRow As Integer, ByVal minCol As Integer, ByVal r As Integer, ByVal c As Integer)
    Dim i, j As Integer
    On Error GoTo errMsg
    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
    errMsg:
    MsgBox "error: ClearCells"
    Exit Sub
End Sub

1.6.3.1 A very simple example

![Excel Sheet with a linear system example]

- This is the VBA code:

```vba
Sub Execute_Gauss()
```

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

```vba
Public Sub ClearCells(ByVal Sheet As String, ByVal minRow As Integer, ByVal minCol As Integer, ByVal r As Integer, ByVal c As Integer)
    Dim i, j As Integer
    On Error GoTo errMsg
    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
    errMsg:
    MsgBox "error: ClearCells"
    Exit Sub
End Sub
```

```
```
1.6. **DDE FUNCTIONNALITY**

```vbnet
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, 5, A()
CellsToMatrix "Sheet1", 4, 6, 3, 1, b()

VBclear GaussData
VPut GaussData, A, "'A'"
VPut GaussData, b, "'b'"

End Sub

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

GaussData = GaussDir + '\Vbgauss\ex2'

VPread GaussData, "'x'", x()
MatrixToCells "Sheet1", 4, 8, x()

End Sub
```

▶ This is the Gauss program

```vbnet
new;
library Vbgauss;
GaussPath = system(2, 0);
cmd = ChangeDir(GaussPath $+'Vbgauss');
a = VPread("'ex2', 'A'");
b = VPread("'ex2', 'b'");
x = b/a;
call VPut("'ex2', x, 'x'");
```

system;
1.6.3.2 Risk neutral density estimation

<table>
<thead>
<tr>
<th>Date</th>
<th>Option type</th>
<th>Maturity date</th>
<th>K</th>
<th>Premium</th>
<th>FD</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>920102</td>
<td>call</td>
<td>9203</td>
<td>90.3</td>
<td>0.15</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9203</td>
<td>90.4</td>
<td>0.11</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9203</td>
<td>90.5</td>
<td>0.075</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9203</td>
<td>90.7</td>
<td>0.035</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9203</td>
<td>91.1</td>
<td>0.01</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9206</td>
<td>90.4</td>
<td>0.37</td>
<td>90.59</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9206</td>
<td>90.5</td>
<td>0.31</td>
<td>90.59</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9206</td>
<td>91.5</td>
<td>0.025</td>
<td>90.59</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>call</td>
<td>9209</td>
<td>91.2</td>
<td>0.2</td>
<td>90.95</td>
<td>0.101875</td>
</tr>
<tr>
<td>920102</td>
<td>put</td>
<td>9203</td>
<td>89.9</td>
<td>0.066</td>
<td>90.25</td>
<td>0.101875</td>
</tr>
</tbody>
</table>

**Microsoft Excel - Pibor (RND Estimation).xls**

**Options on Pibor 3 Months Future Malif S.A.**

**Microsoft Excel - Pibor (RND Estimation).xls**

**Current date**: 990317
**Maturity date**: 9506
**m(Future)**: 86
**man(Future)**: 96
**N(Future)**: 161
This is the VBA code:

```vba
Sub Execute_Gauss()
    Dim idx As Integer
    Dim GaussProgram As String
    Find_Data idx
    MsgBox "Data sorting finished"
    Save_Data idx
    GaussProgram = GaussDir + "\Vegas\rand.prg"
    Execute Gauss GaussProgram
    ClearCells "RND plot", 3, 2, 1000, 2
End Sub

Sub Save_Data(ByRef idx 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 + "\Vegas\rand"
    r = UBound(idx, 1)
    ReDim data(1 To r, 1 To 5)
    sheet = "Pibor"
    For i = 1 To r
        TypeOption = Worksheets(sheet).Cells(idx(i, 1), 2).Value
        If TypeOption = "call" Then
            data(i, 1) = 10
        Else
            data(i, 1) = 0
        End If
        For j = 2 To 5
            data(i, j) = Worksheets(sheet).Cells(idx(i, 1), j + 2).Value
        Next j
    Next i
End Sub
```

![Risk Neutral Density Estimation](image)
Next i

V8Clear GaussData
V8Put GaussData, data(), 'data'

sheet = 'Estimation'
CellsToMatrix sheet, 4, 3, 5, 1, inf()  
V8Put 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 = 'Fibon'
  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\rdn''

  V8Read GaussData, 'F + PDF', x()
  MatrixToCells '"RND plot''', 3, 2, x()

  Worksheets("'RND plot''").Activate
  Charts.Add
  ActiveChart.ChartType = xYSScatterSmoothNoMarkers
  ActiveChart.SetSourceData Source:=Sheets("'RND plot''").Range("B3:C421!"), PlotBy:=xColumns
  ActiveChart.Location Where:=xLocationAsObject, 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 optimum, V8gauss;

randseed 123; /* stability problem */

GaussPath = mystate(2,0);
rand = ChangeDir(GaussPath $'+ 'VBgauss'');

#include d:\gauss\vbgauss\rnd.src;
data = V8read(''rdn'',''data'');
OptionType = data[1,1];
K = data[.,2];
Premium = data[.,3];
F0 = data[.,4];
r = data[..,6];
inf = WBrad('rnd',''inf'');
d = inf[1];
dtm = 100*inf[2] + 15;
imP = inf[3];
maxP = inf[4];
nF = inf[6];
dF = (maxP - minP) / (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]));
sv = sv / 0.4;
output file = rnd.out reset;
theta = estimateRND(Premium,K,tau,r,OptionType,F0,sv);
print theta;
output off;
F = seq(minF,dF,nF);
PDF = pif2LN(F,theta[1],theta[2],theta[3],theta[4],theta[5]);
WPut('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} = optim({theta,f,g,retcode}_{estimateRND},sv);
theta = sqrt(theta[2]);
theta[6] = 0.65 + 0.6 * exp(-theta[6])/(1-exp(-theta[6]));
ret(theta);
endp;
proc _estimateRND(theta);
local mu1,mu2,sigma1,sigma2,alpha,m;
local Premium,K,tau,OptionType,F0;
local TheoreticalPremium,u,v;
theta = sqrt(theta[2]); /* positivity */
/* 0.05 <= alpha <= 0.45 */
theta[6] = 0.65 + 0.8 * exp(-theta[6])/(1-exp(-theta[6]));
mu1 = theta[1];
sigma1 = theta[2];
mu2 = theta[3];
sigma2 = theta[4];
alpha = theta[6];
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) * (alphas * (a1 * b1 - K * c1) + (1 - alpha) * (a2 * b2 - K * c2));

Put_ = exp(-r * tau) * (alphas * (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" $ stof(t, "%4f", 6, 0);

yyyy = stof(strstr(t, 1.4));
mm = stof(strstr(t, 5.2));
dd = stof(strstr(t, 7.2));
retP(yyyy, mm, dd);
endP;

proc edays_(tstart, tend);
retP( edays(convDates(tstart), convDates(tend))/360 );
endP;

proc pdf_2LN(x, mu1, sigma1, mu2, sigma2, alpha);
retP( alpha * pdfLN(x, mu1, sigma1) + (1-alpha) * pdfLN(x, mu2, sigma2) );
endP;

proc cdf_2LN(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 University 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 \( u^n_{i,j} \) 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

\[
\begin{align*}
  a(t, x, y) &= a \\
  b(t, x, y) &= b \\
  c(t, x, y) &= c \\
  d(t, x, y) &= 0 \\
  e(t, x, y) &= 0 \\
  f(t, x, y) &= 0 \\
  g(t, x, y) &= 0
\end{align*}
\]

\( \mathbb{R} \) is set to \([0, 1] \times [0, 1]\) and we have

\[
\begin{align*}
  u(t, 0, y) &= e^{-\left(a+2b+c\right)t}\sin(y) \\
  u(t, 1, y) &= e^{-\left(a+2b+c\right)t}\sin(1+y) \\
  u(t, x, 0) &= e^{-\left(a+2b+c\right)t}\sin(x) \\
  u(t, x, 1) &= e^{-\left(a+2b+c\right)t}\sin(1+x)
\end{align*}
\]
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;
PDE2Dnet;

a = 0.1; b = 0.05; c = 0.15;
proc eProc(t,x,y);
    rep( a*ones(rows(x),cols(y)) );
endp;
proc hProc(t,x,y);
    rep( b*ones(rows(x),cols(y)) );
endp;
proc cProc(t,x,y);
    rep( c*ones(rows(x),cols(y)) );
endp;
proc dProc(t,x,y);
    rep( zeros(rows(x),cols(y)) );
endp;
proc eProc(t,x,y);
    rep( zeros(rows(x),cols(y)) );
endp;
proc fProc(t,x,y);
    rep( zeros(rows(x),cols(y)) );
endp;
proc gProc(t,x,y);
    rep( zeros(rows(x),cols(y)) );
endp;
proc tminBound(t,x,y);
    rep( sin(xy) );
endp;
proc xminBound(t,x,y);
    rep( exp(-(a+2b+c)*t)*sin(y) );
endp;
proc xmaxBound(t,x,y);
    rep( exp(-(a+2b+c)*t)*sin(1+ty) );
endp;
proc yminBound(t,x,y);
    rep( exp(-(a+2b+c)*t)*sin(x) );
endp;
proc ymaxBound(t,x,y);
    rep( exp(-(a+2b+c)*t)*sin(1+x) );
endp;

call PDE2D(&aProc, &hProc, &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_PrintItems = 2;
call Hopscotch(tmin, tmax, Nx, xmin, xmax, Ny, ymin, ymax, 1, 1, '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(xy)
**
2.1. THE PDE2D LIBRARY

/*

proc solutionProc(t,x,y);
   ret(exp((-st+2*bb*c)*t)*sin(x*y))
endp;

sol = solutionProc(tmax,x,y);

graphset;

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

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

graphset;
   _plate = ' ***'; _pnum = 2; _ptitlht = 0.25; _paxes = 0.25; _pnumht = 0.20;
   xlabel('x'); ylabel('y'); zlabel('u(t,x,y)');
setwind(2);
   title('Numerical solution')
   surface(x,y,U);

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

Gourlay and McKee [1977] - Example 1 page 203

Numerical solution

Exact solution

2.1.2.2 Elliptic problems

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

\[
\begin{align*}
   a(x,y) &= 1 \\
   b(x,y) &= -\frac{1}{2} \\
   c(x,y) &= 1 \\
   d(x,y) &= 0 \\
   e(x,y) &= 0 \\
   f(x,y) &= 0 \\
   g(x,y) &= 0 
\end{align*}
\]
with the boundary conditions

\[
\begin{align*}
u(0, y) &= 0 \\
u(1, y) &= y + y^2 \\
u(x, 0) &= 0 \\
u(x, 1) &= x + x^2
\end{align*}
\]

The solution of this problem is

\[u(x, y) = x^2 y + xy^2\]

For the initial guess solution, we use random numbers.

```c
/* 
 *  pde2d2.prg 
 */
new;
library PDE2D,pgraph;
PDE2Dset;
proc sProc(t,x,y);
   retp( ones(rows(x),cols(y)) );
endp;
proc hProc(t,x,y);
   retp( -0.5*ones(rows(x),cols(y)) );
endp;
proc pProc(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( rand(rows(x),cols(y)) );
endp;
proc solutionProc(t,x,y);
   retp( (x^2).*y + x.*(y^2) );
endp;
rndseed 123;
```
2.1. THE PDE2D LIBRARY

call PDE2D(&Proc,&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);
_plt = ' '; _pnum = 0; _paxes = 0; _pframe = 0; _psurf = {0,0};
_ptilh = 0.45;

i = 1;
do until i > 14;
  U = ExtractSolution(['pde2d2','t'][3*i]);
  setwind(i);
  str = ftcos(3*i - 1,'iter no %lf',1,0);
  title(str);
  surface(x,y,U);
  i = i + 1;
end;

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

setwind(15);
  str = ftcos(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 $\Delta$, $\Gamma$ and $\Theta$ coefficients of the call options. We remark that the differences between these numerical values and those obtained by exact formulas are very small.

```c
/*
** pde2d3.prg
***/

new;
library PDE2D, option, pgraph;
PDE2Dnet;

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

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

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

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

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

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

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

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

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

_Hopscotch_PrintItems = 10;
call Hopscotch(tmin, tmax, NT, xmin, xmax, Nx, ysim, ymax, Ny, ' pde2d3 '');

_dt = vread(_Hopscotch_MeshRation, 'k' );
_dS = vread(_Hopscotch_MeshRation, 'hx' );

indexT = seq(10, 15, 30);
indexS = seq(1, 10, 21);

graphset;
```
2.1. THE PDE2D LIBRARY

begwind;
window(2,2,0);
_pdate = ''; _pnum = 2; _pnumt = 0.25; _ptitlht = 0.25; _paxht = 0.25;
font(('simplex simgima'));
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('Gammam');
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('20T\,ST');
ylabel('S0');
xtics(0,0.20,0,10,0);
ytics(80,120,10,0);
xtics(0,50,10,0);
surface(submat(t',0,indxT),submat(theta',indxS0,0),
submat(theta',indxS0,indxT));
setwind(4);
surface(submat(t',0,indxT),submat(theta',indxS0,0),
submat(EuropeanBS_Theta(S0,K,sigma,t',b,r),indxS0,indxT));
graphpt('-c1 -cf-pde2d3.eps');
endwind;

2.1.2.4 Computing stochastic volatility option greeks coefficients

----------

/*
** Kurpiel, A. and T. Roncalli [1996], Hopscotch methods for two-state
** financial models, FERC working paper, City University Business School */

new;
library option,FME2D;
FME2Dset;

K = 100;
S0 = norm(80,5,9);
b = -0.04;
epsilon = 0.08;
Vo = 0.20^2;
tau = 0.25;

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

proc dProc(t,x,y);
  ret( 0.5 * y .* (x^2) );
endp;

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

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

proc gProc(t,x,y);
  ret( b .* times(1,cols(y)) );
endp;

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

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

proc macProc(t,x,y);
  ret( zeros(rows(x),cols(y)) );
endp;

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

proc hProc(t,x,y,U);
  local Payff,cmd;
  Payff = ( x > K ) .* ( x - K ) .* ones(1,cols(y));
  cmd = Payff .> U;
  ret( cmd .* Payff + (1-cmd).*U );
endp;

G<------------------------------------------->

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

proc xmaxBound(t,x,y);
  ret( zeros(rows(x),cols(y)) );
endp;

proc DmaxBound(t,x,y);
  ret( ones(rows(x),cols(y)) );
endp;

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

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

call FME2D(&dProc, &bProc, &cProc, &gProc, &vProc, &macProc, &hProc, &xminBound, &xmaxBound, 0, 0, 0, 0, 0, 0, &DmaxBound, 0, 0);
2.1. THE PDE2D LIBRARY

\[ \text{xmin} = 50; \text{xmax} = 150; \text{Nx} = 101; \]
\[ \text{ymin} = 0.002; \text{ymax} = 0.122; \text{Ny} = 61; \]
\[ \text{tmin} = 0; \text{tmax} = \text{tau}; \text{NT} = \text{floor}(1825*\text{tau}); \]
\_Hopscotch\_SaveLastIter = 1;
\_Hopscotch\_PrintIters = 1;
call Hopscotch(tmin, tmax, NT, xmin, xmax, Nx, ymin, ymax, Ny, 'pde2d');
save ratios = \_Hopscotch\_MeshRatios;

-------------

/*
** Kurpiel, A. and T. Roncalli [1994], Stochastic volatility and contingent
** claims pricing --- the American option case, FERG working paper,
** City University Business School
*/

c\_\text{u};
library option\_PDE2D\_pgraph;
PDE2Dnet;
K = 100;
S0 = seq(80,5,8);
b = -0.04;
r = 0.08;
V0 = 0.20^2;
\tau = 0.25;
\rho = -0.5;
Kappa = 0.5;
\theta = 0.20^2;
lambda = 0.0;
sigmaV = 0.10;
kappaStar = kapp + lambda;
thetaStar = kappaStar/(kapp+lambda);
load MeshRatios = ratios;
S = ExtractSolution('pde2d','x');
V = ExtractSolution('pde2d','y');
tau = ExtractSolution('pde2d','t');
U = ExtractSolution('pde2d','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);

\text{graphset};
\_pdate = ' '; \_pnum = 2;
\_pilype = 6;
\text{fonts}('simplex simgima');
\text{title}('\text{\textbackslash 202D}\text{\textbackslash 2014}\text{am,Hopscotch}''');
xy(S,Delta1);
\text{graphset};
\_pdate = ' '; \_pnum = 2;
\_pilype = 6;
\text{fonts}('simplex simgima');
\text{title}('\text{\textbackslash 202D}\text{\textbackslash 2014}\text{am,Hopscotch} - \text{\textbackslash 202D}\text{\textbackslash 2014}e,SS''');
\text{xlabel('S')}0[''];
\text{ylabel('V')}0[''];
\text{surface(trim(V',5,5)',S,trimr((Delta1-Delta2)',5,5)');}
\text{graphset};
\_pdate = ' '; \_pnum = 2;
\text{title('VEGA\_am,Hopscotch''');
\text{ztics(0,0.05,25,0);
\text{xlabel('S')}0[''];
\text{ylabel('V')}0[''];
\text{graphprf(''-c1 -cf-pde2d5.eps'');
\text{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 University Business School".

```plaintext
/*
** Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state
** financial models, FERC working paper
** *
new;
library pde2d, graph;
PDE2Dnet;
declare matrix e;
rseed 123;
sigma = 0.20;
r = 0.04;
mu = 0.08;
I = 1;
eta = 0.1;
Vbar = 1.5;
proc aProc(t,x,y);
   rtp( 0.5*(sigma^2) * (x^2) * ones(rows(x),cols(y)) );
endp;
proc bProc(t,x,y);
   rtp( 0. * ones(rows(x),cols(y)) );
endp;
proc cProc(t,x,y);
   rtp( 0. * ones(rows(x),cols(y)) );
endp;
proc dProc(t,x,y);
   rtp( (r - mu + eta*(Vbar-x)) * x * ones(rows(x),cols(y)) );
endp;
proc eProc(t,x,y);
   rtp( 0. * ones(rows(x),cols(y)) );
endp;
proc fProc(t,x,y);
   rtp( 0. * ones(rows(x),cols(y)) );
endp;
proc gProc(t,x,y);
   rtp( zeros(rows(x),cols(y)) );
endp;
proc lminBound(t,x,y);
   rtp( rmu(rows(x),cols(y)) );
```
2.1. THE PDE2D LIBRARY

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

proc xmaxBound(t,x,y);
    rtp( (x - 1) .* ones(rows(x),cols(y)) );
endp;

proc DxminBound(t,x,y);
    rtp( ones(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
    rtp( 0.*ones(rows(x),cols(y)) );
endp;

_output = 1;
_Hopscotch_SavelastTIter = 1;
_Hopscotch_Elliptic = 1;
_fctmptol = 1e-4;
_Hopscotch_PrintIter = 10;
output file = wp12.out reset;
V = 1.585[1,6];
sol = {};
x = {};
e = 1;
do until e > 2;
    Vstar = V[e];
    call PDE2D(@Proc,&Proc,&dProc,&aProc,&gProc,&fProc,&dProc,&gProc,&fProc,&aProc,0,
            &xminBound,&xmaxBound,0,0,0,0,0,0,0,0);
xmin = 0.000; xmax = Vstar; Nx = 51;
ymin = 0.000; ymax = t; 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','U');
x = vrend(Hopscotch,MeshRation,'hx');
sol = sol'U[,]3];
e = e + 1;
enddo;
V = seqs(1.0,0.065,51);
graphset;
_pdate = 'J'; _pnum = 2; _plttype = 1[3]6;
xlabel('V'); ylabel('C(V)'); ytics(0,0.8,0.1,2);
xtics(0,0.2,0.0,0.2);
_plgext = 'Mean Reversion I\000Mean Reversion II\000V - I';
_plgctol = [5 5 5 8 4];
_pline = 1.5 1.655 0.1 0.655 1.75 1.65 0.655 1.75 1.65 0.655 1.75 ;
graphprtr('c=1 -cfs=pde2d,eps'); xy[x,1,2,1]V,sol[2,1](V-1);```
2.1.2.6 Other financial problems


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
  - AmericanBS
  - AmericanBS_Delta
  - AmericanBS_Gamma
  - AmericanBS_impVol
  - AmericanBS_Omega
  - AmericanBS_SmileCurve
  - AmericanBS_Theta
  - AmericanBS_Vega

- EuropeanBS
  - EuropeanBS_Delta
  - EuropeanBS_Gamma
  - EuropeanBS_impVol
  - EuropeanBS_Omega
  - EuropeanBS_SmileCurve
  - EuropeanBS_Theta
  - EuropeanBS_Vega

2.2. THE OPTION LIBRARY

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

- **Heston [1993]** model
  - EuropeanHeston

- **Bates [1996]** model
  - EuropeanBates
2.2. THE OPTION LIBRARY

- Chang, Chang and Lim [1998] model
  - AmericanCCL
  - AmericanCCL_Delta
  - AmericanCCL_Gamma
  - AmericanCCL_impVol
  - AmericanCCL_Omega
  - AmericanCCL_Theta
  - AmericanCCL_Vega
  - 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.

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

<table>
<thead>
<tr>
<th>Price</th>
<th>80.00000</th>
<th>76.821334</th>
<th>83.310191</th>
<th>86.757348</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00000</td>
<td>68.23287</td>
<td>70.857861</td>
<td>83.310191</td>
<td>94.085468</td>
</tr>
<tr>
<td>112.00000</td>
<td>65.320467</td>
<td>73.769668</td>
<td>86.757348</td>
<td>97.974779</td>
</tr>
<tr>
<td>120.00000</td>
<td>62.725688</td>
<td>77.980701</td>
<td>94.085468</td>
<td>102.03287</td>
</tr>
</tbody>
</table>

Option Price Tree

<table>
<thead>
<tr>
<th>Price</th>
<th>0.00000000</th>
<th>0.056401446</th>
<th>0.11551579</th>
<th>0.23668786</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20000</td>
<td>0.000000001</td>
<td>0.000000002</td>
<td>0.000000003</td>
<td>0.000000004</td>
</tr>
<tr>
<td>0.40000</td>
<td>0.000000001</td>
<td>0.000000002</td>
<td>0.000000003</td>
<td>0.000000004</td>
</tr>
<tr>
<td>0.60000</td>
<td>0.000000001</td>
<td>0.000000002</td>
<td>0.000000003</td>
<td>0.000000004</td>
</tr>
<tr>
<td>0.80000</td>
<td>0.000000001</td>
<td>0.000000002</td>
<td>0.000000003</td>
<td>0.000000004</td>
</tr>
</tbody>
</table>

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.
2.2. THE OPTION LIBRARY

\[ \text{SO} = \text{seq}(70, 0.5, 121); \]
\[ \text{K} = 100; \]
\[ \text{sigma} = 0.20; \]
\[ \text{tau} = \text{seq}(90/365, -1/365, 89); \]
\[ r = 0.08; \]
\[ \text{b} = 0; \]
\[ \text{Option_Type} = '\text{'put'}'; \]

\[ \text{Gamma} = \text{EuropeanMerton(SO, K, sigma, tau, b, r);} \]

\[ \text{graphset; } \]
\[ \text{\_plsize = '11'; \_pnum = 2; \_paxht = 0.30; \_ptitlht = 0.30; \_pframe = 0; } \]
\[ \text{font('', 'times', 'times');} \]
\[ \text{title('2021Put');} \]
\[ \text{xlabel('S');} \]
\[ \text{ytics(70, 130, 1, 0);} \]
\[ \text{ztics(0, 0.01, 0, 0);} \]
\[ \text{graphpr('c=1 cf=option2.eps');} \]
\[ \text{surface(tau, SO, Gamma);} \]

2.2.2.3 Option pricing with jump-diffusion processes

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

/*
** Bates91.prg
***/

new;
library option;
output file = option3.out reset;

\[ \text{SO} = 260; \]
\[ \text{b} = 0; \]
\[ r = 0.10; \]
\[ \text{tau} = 0.25; \]
\[ \text{K} = \text{seq}(220, 15, 5); \]
\[ \text{let sigma[1,5] = 0.1414 0.10 0.10 0.10 0.10}; \]
\[ \text{let lambdaStar[1,5] = 0 10 10 0.25 0.25}; \]
\[ \text{let deltaStar[1,5] = 0 0.01 -0.01 0.20 -0.20}; \]
\[ \text{let kbarStar = exp(gammaStar) - 1}; \]
\[ \text{Option_Type = 'call'}; \]
\[ \text{C = \text{EuropeanMerton(SO, K, sigma, tau, b, r, kbarStar, deltaStar, lambdaStar);} \]
\[ \text{Option_Type = 'put'}; \]
P = EuropeanMerton(S0,K,sigma,tau,b,r,kStar,delStar,ldStar);

print \"European option prices \";
call printf(C,1,\".*lf\" \"8 \"2); print;
call printf(P,1,\".*lf\" \"8 \"2); print;

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

e = ones(5,1);
sigma = sigma .* e;
kStar = kStar .* e;
delStar = delStar .* e;
ldStar = ldStar .* e;
K = e .* K;

_Option_Type = \"call\";
C = EuropeanMerton(S0,K,sigma,tau,b,r,kStar,delStar,ldStar);

_Option_Type = \"put\";
P = EuropeanMerton(S0,K,sigma,tau,b,r,kStar,delStar,ldStar);

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

print \"European option prices \";
call printf(C,1,\".*lf\" \"8 \"2); print;
call printf(P,1,\".*lf\" \"8 \"2); print;

_Option_Type = \"call\";
C = AmericanMerton(S0,K,sigma,tau,b,r,kStar,delStar,ldStar);

_Option_Type = \"put\";
P = AmericanMerton(S0,K,sigma,tau,b,r,kStar,delStar,ldStar);

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

print \"American option prices \";
call printf(C,1,\".*lf\" \"8 \"2); print;
call printf(P,1,\".*lf\" \"8 \"2); print;

output off;

European option prices :
29.49 29.45 29.58 29.30 30.14
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 16.61 17.28 16.74
29.68 29.82 29.61 30.68 29.35

European option prices :
29.49 29.45 29.58 29.30 30.14
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 16.61 17.28 16.74
29.68 29.82 29.61 30.68 29.35

American option prices :
30.36 30.15 30.08 30.00 30.25
6.92 6.83 6.34 6.54 6.02
2.07 2.19 1.89 2.77 1.11
0.43 0.57 0.35 1.46 0.09
0.23 0.19 0.33 0.04 0.92
1.76 1.63 1.89 0.99 2.19
2.2. THE OPTION LIBRARY

2.2.2.4 Stochastic volatility option models

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

```r
new;
library option.pgraph;

S0 = eqSim(65,1,31);
K = 100;
sigma = 0.20;
sigmaV = 2.5;
theta = 0.08;
rho = 0;
b = 0;
_OPTION_Type = 'plot';
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_WAIT = 1e-4;
_OPTION_NumTime = 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);

grapheat;
肇date = '11';_pnum = 2;_pctrl = 3;_pstype = 8|8|10|11|10;_psymex = 2.5;
肇type = 6|8|1|1|1|3;
肇ntol('simplex siqngcm');
title('Heston model');
肇labl(['S']0);'
肇labl('Put');
肇legxtr = '85\000\2000\2000\2000 = 0.2[2] -- \2002\2002 = 0''
肇\000\2002\2002 = 0.3[2] -- \2002\2002 = 0''
肇\000\2002\2002 = 0.3[2] -- \2002\2002 = 0.9''
肇\000\2002\2002 = 0.3[2] -- \2002\2002 = 0.9''
肇\000\2002\2002 = 0.1[2] -- \2002\2002 = 0.9''
肇pegctl1 = [2 5 6 4];
肇tics(85,115,5,0);
肇graphpr('col=1 -cf=option4.epso');
xy(20,'P2 P3 P4 P5');
```
Influence of $\rho$ 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).

```r
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;
```
2.2. THE OPTION LIBRARY

\[ C_2 = \text{AmericanCCL}(S_0, K, \sigma, \tau, b, r, \lambda); \]
\[ \lambda = 4; \]
\[ C_3 = \text{AmericanCCL}(S_0, K, \sigma, \tau, b, r, \lambda); \]
\[ \lambda = 8; \]
\[ C_4 = \text{AmericanCCL}(S_0, K, \sigma, \tau, b, r, \lambda); \]
\[ \text{graphnet; fonte('simplex simgma');} \]
\[ \_pdeTr = '1'; \_pnum = 2; \]
\[ \_plectr = 'BS\000CCL (1)\000CCL (2)\000CCL (3)\000CCL (4)'; \]
\[ \_plect1 = [2 5 3 4]; \]
\[ \text{xlabel('S0');} \]
\[ \text{ylabel('call');} \]
\[ \text{graphpr('c=1 -cf=option6.eps');} \]
\[ \text{xy}(S_0, C_0C_1C_2C_3C_4); \]

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 (??).

\[ \text{library option.pgraph;} \]
\[ \text{load oam.c;} \]
\[ \text{Nobs = rowv(oam_c);} \]
\[ b = 0; \]
\[ \_Option_impVol = 'NR'; /^(\_Option_impVol = 'bi-section'; */ \]
\[ \text{tau = zeros(Nobs,1);} \]
\[ i = 1; \]
\[ \text{do until i > Nobs;} \]
\[ \text{tau[i] = option_estdays(oam_c[i,1], oam_c[i,3]);} \]
\[ i = i + 1; \]
\[ \text{end;} \]
\[ F0 = \text{oam_c[1,6]}; \]
\[ K = \text{oam_c[1,4]}; \]
\[ C = \text{oam_c[1,6]}; \]
\[ r = \text{oam_c[7]} / 100; \]
\[ \text{impVol} = \text{EuropeanBS_impVol}(F0, K, tau, b, r, C); \]

\[ \sigma = 0.006; \]
\[ \text{implambda = zeros(Nobs,1);} \]
\[ _\text{cp.PrintIter} = 0; \]
\[ _\text{output} = 0; \]
i = 1;
do until i > Nobs;

proc rrs(lambda);
   local u;
   u = EuropeanCCL(omc_c[i, 6], omc_c[i, 4], sigma, 
                    tau[i], beta, omc_c[i, 7]/100, lambda) - omc_c[i, 8];
   ret(u^2);
endp;

{impllambda[i, f0, g0, retcode] = \$Newton(&rrs, 0);

i = i + 1;
enddo;

graphset;
   _pnum = 2; _pdate = ''; _plotype = 6|4;
   fonts('simplex simprras');
   title('2021\2031 implied [\(\text{CCL}\)]');
   acxlabel('lab', 6);
   acylabel('lab', 6);
   acxtics(1,Nobs,Nobs/4,11);
   graphpr('c=1 -cf=option6a.epd');
   xy(seqa(1,1,Nobs), impllambda);

   title('2021\2031 implied [\(\text{BS}\)]');
   graphpr('c=1 -cf=option6b.epd');
   xy(seqa(1,1,Nobs), impllambda);

\begin{flushleft}
\includegraphics[width=\textwidth]{implBS.png}
\end{flushleft}

\begin{flushleft}
\includegraphics[width=\textwidth]{implCCL.png}
\end{flushleft}
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

```r
ew;
library fanpac;
fanset;

/* loading data */
load y[320,7] = wolverine.asc;
_fan_series = y[,4];
Nobs = rows(_fan_series);

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

/* stationarity constraint */
_nlp_IneqProc = &garch_ineq;

/* bounds for GARCH parameters */
_nlp_Bounds = { .001 1e250, /* omega > 0 */
               0 1,        /* garch_1 >= 0 */
```

57
-1e260 1e260,
-1e260 1e260,
-1e260 1e260;

/* Analytical gradient */
\_nlp\_GradProc = &arch\_n\_grd;
\_nlp\_IterInfo = 1;
sv = \{.2, .3, .2, .1, 1\};

output file = fanpac1.out reset;
\{coef, f, g, retcode\} = \_nlp(&arch\_n, sv);
print coef;
output off;

0.93066861
0.84091192
0.010302615
0.11585207
0.99287855

3.1.1.2 Keywords syntax

new;
library fanpac;

session Exemple1 'wilshire';

setVarNames date cpri ce cvdiv cwt cvvri ce cvdiv cvaret;
setDataSet wilshire.asc;
setSeries cvaret;

setInferenceType InvWald;
estimate model1 garch(1,2);
output file = fanpac2.out reset;
showResults;
output off;

-----------------------------------------------
Session: Exemple1
-----------------------------------------------

-----------------------------------------------
FANPAC Version 1.0.0 Data Set: wilshire 11/30/1998 9:45:16
-----------------------------------------------

-----------------------------------------------
Run: model1
-----------------------------------------------

return code = 0
normal convergence

Model: GARCH(1,2)

Number of Observations : 320
Observations in likelihood : 316
Degrees of Freedom : 313

AIC 1852.20
BIC 1871.01
LRS 1842.20

roots
-------
-8.3924913
1.0364427
1.1650608

Abs(roots)
-------
3.1. FANPAC

8.3924913
1.0304427
1.1690008

unconditional variance

28.149469

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

Series: cvret

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Standard Errors</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>0.831</td>
<td>0.003</td>
<td>0.822</td>
<td>0.841</td>
</tr>
<tr>
<td>Garch1</td>
<td>0.041</td>
<td>0.000</td>
<td>0.040</td>
<td>0.042</td>
</tr>
<tr>
<td>Arch1</td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.010</td>
</tr>
<tr>
<td>Arch2</td>
<td>0.116</td>
<td>0.000</td>
<td>0.115</td>
<td>0.117</td>
</tr>
<tr>
<td>Const</td>
<td>0.253</td>
<td>0.000</td>
<td>0.253</td>
<td>0.255</td>
</tr>
</tbody>
</table>

Correlation Matrix of Parameters

<table>
<thead>
<tr>
<th></th>
<th>omega</th>
<th>Garch1</th>
<th>Arch1</th>
<th>Arch2</th>
<th>Const</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>1.000</td>
<td>0.871</td>
<td>-0.214</td>
<td>-0.079</td>
<td></td>
</tr>
<tr>
<td>Garch1</td>
<td>-0.867</td>
<td>1.000</td>
<td>0.006</td>
<td>-0.334</td>
<td></td>
</tr>
<tr>
<td>Arch1</td>
<td>0.131</td>
<td>0.006</td>
<td>1.000</td>
<td>-0.586</td>
<td></td>
</tr>
<tr>
<td>Arch2</td>
<td>-0.214</td>
<td>-0.534</td>
<td>0.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>Const</td>
<td>0.079</td>
<td>0.142</td>
<td>0.180</td>
<td>-0.274</td>
<td></td>
</tr>
</tbody>
</table>

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 fannac;

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

output file = fannac3.out reset;

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

_fan_series = load('''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,
};

{ coefs, fct, grad, rcode } = nlp(&garch_n, start);
print;
print';' Nelson & Cao constraints';'
print;

if rcode < 2;
  /* covariance matrix of parameters */
  h = _nlp_CovPar(coefs, &garch_n, &garch_n_grd, nhb, 1);
else;
  h = error(0);
endif;
format /rd 12.4;
if not scalmiss(h);
  alpha = .95; /* 95% cl's */
  dv = cdfci(0.5*alpha, nhb-rews(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 */
  se1 = 0; /* selection vector, if zero, all cl's computed */
  cl = _nlp_climit(coefs, h, nhb, alpha, se1);
else;
  print', Coefficients';
  print coefs;
endif;

/*
** standard constraints
*/

_nlp_IneqProc = { . };

_nlp_c = { 0 -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';' standard constraints';
if rcode < 2;
    /* covariance matrix of parameters */
    h = _nlp_CovPar(coefs, &garch_n, &garch_n_gnd, nobs, 1);
else;
    h = error(0);
endif;

format /rd 12.4;
if not scalmiss(h);
    alpha = .05; /* 95% c1's */
    dv = cdfci(0.5*alpha, nobs-rews(coefs))&rmdiag(h);
    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_limits(coefs, h, nobs, alpha, sel);
endif;
format /rd 12.4;
if not scalmiss(cl) and not scalmiss(h);
    print;                  /* 
    print 'Coefficients   lower cl   upper cl';
    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   0.9187
    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   0.1364   0.4180
-0.0670   -0.2056   -0.1294
0.5435   0.4399   0.6472

standard constraints

confidence limits from standard errors

Coefficients   lower cl   upper cl
    0.1770   0.0058   0.3463
    0.5687   0.3315   0.6969
    0.3011   0.1019   0.5063
    0.0000   .       .
0.5437   0.4424   0.6490

confidence limits from inversion of Wald statistic

Coefficients   lower cl   upper cl
    0.1770   0.0334   0.3463
    0.5687   0.3315   0.6969
    0.3011   0.1341   0.4313
    0.0000   .       .
3.1.3 Univariate models

```r
new;
library fampac.pgraph;
session Example 'wilshire';
setVarNames date exprice exdiv ecrime exprice exdiv ecrime;
setDataSet wilshire.asc;
setSeries ecrime;
plotSeries;
plotSeriesACF;
plotSeriespACF;
setInferenceType InvWald;
estimate model1 garch(1,2);
estimate model2 lgarch(1,2);
estimate model3 garchm(1,1);
showResults model2;
testSR;
plotQQ model1 model3;
plotUV model1 model2;
plotSR;
```

---

**Session: Example**
---

**wilshire**
---

**FAMPAC 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 1829.20
---

**roots**
---

-12.7474
1.0967
1.2476
---

**Abs(roots)**
---

12.7474
1.0967
1.2476
---

**unconditional variance**
---

11.9517
---

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Standard Errors</th>
<th>Lower Limits</th>
<th>Upper Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>1.991</td>
<td>1.807</td>
<td>0.000</td>
<td>4.956</td>
</tr>
<tr>
<td>Garch1</td>
<td>0.801</td>
<td>0.102</td>
<td>0.601</td>
<td>0.867</td>
</tr>
<tr>
<td>Arch1</td>
<td>0.032</td>
<td>0.048</td>
<td>0.000</td>
<td>0.111</td>
</tr>
<tr>
<td>Arch2</td>
<td>0.072</td>
<td>0.084</td>
<td>-0.026</td>
<td>0.161</td>
</tr>
<tr>
<td>Const</td>
<td>1.176</td>
<td>0.225</td>
<td>0.733</td>
<td>1.619</td>
</tr>
<tr>
<td>Nu</td>
<td>6.242</td>
<td>1.988</td>
<td>2.948</td>
<td>10.173</td>
</tr>
</tbody>
</table>

Correlation Matrix of Parameters

<table>
<thead>
<tr>
<th></th>
<th>omega</th>
<th>Garch1</th>
<th>Arch1</th>
<th>Arch2</th>
<th>Const</th>
<th>Nu</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>1.000</td>
<td>-0.840</td>
<td>0.059</td>
<td>0.100</td>
<td>-0.036</td>
<td>-0.134</td>
</tr>
<tr>
<td>Garch1</td>
<td>-0.840</td>
<td>1.000</td>
<td>0.014</td>
<td>-0.501</td>
<td>0.086</td>
<td>-0.007</td>
</tr>
<tr>
<td>Arch1</td>
<td>0.059</td>
<td>0.014</td>
<td>1.000</td>
<td>-0.636</td>
<td>0.133</td>
<td>-0.086</td>
</tr>
<tr>
<td>Arch2</td>
<td>0.100</td>
<td>-0.501</td>
<td>-0.636</td>
<td>1.000</td>
<td>-0.154</td>
<td>0.037</td>
</tr>
<tr>
<td>Const</td>
<td>-0.036</td>
<td>0.086</td>
<td>0.133</td>
<td>-0.154</td>
<td>1.000</td>
<td>-0.064</td>
</tr>
<tr>
<td>Nu</td>
<td>-0.134</td>
<td>-0.007</td>
<td>-0.086</td>
<td>0.037</td>
<td>0.064</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Session: Example

wilshire

Time Series

Series: curre

| skew | 266.1720 | pr  | 0.000 |
| kurtosis | 8558.5534 | pr  | 0.000 |
| heteroskedastic-consistent | Ljung/Box | 39.0081 | pr  | 0.124 |

Residuals

model1: GARCH(1,2)

| skew | -4.2151 | pr  | 0.040 |
| kurtosis | 8.6365 | pr  | 0.003 |
| heteroskedastic-consistent | Ljung/Box | 17.5867 | pr  | 0.965 |

model2: TARCH(1,2)

| skew | -4.4598 | pr  | 0.034 |
| kurtosis | 11.2486 | 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 below. See the command reference part for their full description.

1. ARMA processes
   (a) arma_ML: Conditional maximum likelihood for Vector ARMA model
   (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) CPDGDM: 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. **TDmle_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. **FDmle_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) **KFFiltering**: 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
3.2. TSM

(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 B is a basis
      G. **LogEnergy**: Log-energy cost function
      H. **LpNorm**: $\ell^p$ 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 \( \theta = R\gamma + \gamma \)
   ii. **Implicit_to_Explicit**: Convert implicit linear restrictions \( \theta = R\gamma + \gamma \) to explicit linear restrictions \( C\theta = c \)
   iii. **optimum2**: General nonlinear optimization under linear restrictions

### 3.2.1 TSM examples

The examples use the following data bases:

- **fr/dem.asc** — Ascii file which contains daily quotations of the FRF/DEM exchange rate since 1987.
- **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.
- **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).
- **sunspos.asc** — data (page 327) of Janacek and Swift [1993]. The ascii file consists of Wolfer sun spot numbers.

1. **arima1.prg**
   We simulate an ARIMA(1,0,1) process
   \[(1 - 0.95L) y_t = (1 - 0.5L) \varepsilon_t \]  
   with \( \varepsilon_t \sim N(0, 2) \). Then, we estimate the following ARFIMA model in the frequency domain
   \[(1 - \phi_1 L)(1 - L)^d y_t = (1 - \theta_1 L) \varepsilon_t \]  

2. \texttt{arfima2.png}
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 \texttt{OPTMUM}.

3. \texttt{arfima3.png}
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. \texttt{arfima4.png}
We simulate the following ARFIMA process with the procedure \texttt{RMD_arfima}

\[
(1 - 0.8L)(1 - L)^{0.25} y_t = (1 - 0.4L) \varepsilon_t
\]  

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

\[
(1 - \phi_1 L)(1 - L)^d y_t = (1 - \theta_1 L) \varepsilon_t
\]  

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 \(H_1 : d = 0.25\) and \(H_2 : (d, \theta_1) = (0.25, 0.4)\) with the likelihood ratio statistic.

5. \texttt{arfima5.png}
Simulation of fractional processes with \(d = -0.25\) and \(d = 0.75\).

6. \texttt{arma1a.png}
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

\[
\begin{bmatrix}
  y_{1,t} \\
  y_{2,t}
\end{bmatrix} - \Phi_1 \begin{bmatrix}
  y_{1,t-1} \\
  y_{2,t-1}
\end{bmatrix} = \begin{bmatrix}
  \varepsilon_{1,t} \\
  \varepsilon_{2,t}
\end{bmatrix} - \Theta_1 \begin{bmatrix}
  \varepsilon_{1,t-1} \\
  \varepsilon_{2,t-1}
\end{bmatrix}
\]  

with \(\varepsilon_t \sim N(\mathbf{0}_2, \Sigma)\). We use the Newton-Raphson algorithm to obtain the estimates \(\beta = \text{vec} \begin{bmatrix} \Phi_1 & \Theta_1 \end{bmatrix}\). The external variables \_arma\_sigma and \_arma\_epsilon correspond to the estimate of \(\Sigma\) and to the residuals \(\varepsilon\) respectively.

7. \texttt{arma1b.png}
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 \texttt{SSM.ic}. Given the procedure \texttt{PF.ml}, we construct the log-likelihood function. Then, we employ \texttt{TD.ml} to obtain the exact ML estimates. Note that the estimates \(\theta\) correspond to the vector \(\text{vec} \begin{bmatrix} \beta & \mathbb{P}^* \end{bmatrix}\) with \(\mathbb{P}^* = \text{vech}(\mathbb{P})\) and \(\mathbb{P}\) the Cholesky decomposition of \(\Sigma\), that is \(\Sigma = \mathbb{P}\mathbb{P}^\top\).

8. \texttt{arma1c.png}
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

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

To construct the matrix \(R\), we employ the \texttt{design} procedure. Because the argument \texttt{sv} in \texttt{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
\[
\begin{bmatrix}
\beta_1 \\
\beta_2 \\
\beta_3 \\
\beta_4 \\
\beta_5 \\
\beta_6 \\
\beta_7 \\
\beta_8
\end{bmatrix}
= \begin{bmatrix}
1 & 0_{1 \times 6} \\
1 & 0_{1 \times 6} \\
0_{6 \times 1} & I_{6}
\end{bmatrix}
\begin{bmatrix}
\gamma_1 \\
\gamma_2 \\
\gamma_3 \\
\gamma_4 \\
\gamma_5 \\
\gamma_6 \\
\gamma_7
\end{bmatrix}
+ 0_{8 \times 1}
\]

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

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 \( \Phi_1 \) and \( \Theta_1 \) are of the form
\[
\begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

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 \( \beta \) corresponding to the ML estimates under the preceding restrictions (arma1f.prg).

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

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

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

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

17. arma2a.prg
We consider the univariate AR(1) model
\[
y_t = 0.5y_{t-1} + \varepsilon_t
\]

In its state space form, the vector of state variables is \( \begin{bmatrix} y_t & \varepsilon_t \end{bmatrix}^\top \). The covariance matrix corresponds to
\[
\begin{bmatrix}
E[y_t \varepsilon_t] & E[y_t \varepsilon_t] \\
E[\varepsilon_t y_t] & E[\varepsilon_t \varepsilon_t]
\end{bmatrix}
\]
Computing this covariance can be achieved with the SSM_p 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 m1 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
   
   $$ Y_t - \begin{bmatrix} .5 & 0 & 0 \\ .1 & .1 & .3 \\ 0 & .2 & .3 \end{bmatrix} Y_{t-1} = \varepsilon_t $$  

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

   $$ \Sigma = \begin{bmatrix} 2.25 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.74 \end{bmatrix} $$

   To read the matrices, we use the *varect* procedure.

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

   $$ Y_t - \begin{bmatrix} .5 & .1 \\ .4 & .5 \end{bmatrix} Y_{t-1} - \begin{bmatrix} 0 & 0 \\ 0.25 & 0 \end{bmatrix} Y_{t-2} = \varepsilon_t $$  

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

   $$ \Sigma = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.04 \end{bmatrix} $$

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.ld.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 *basis0*, we have a good localization in time, but not in frequency. For the basis *basis9*, 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, \( \ell^p \) 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

\[
y_t = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \eta_t \\ \zeta_t \\ \omega_{t-1} \\ \omega_{t-2} \end{bmatrix} + \epsilon_t
\]

with \( \epsilon_t \sim N(0,H) \) and the transition equation is

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

with \( \nu_t \sim N(0,Q) \). We have

\[ H = \sigma_t^2 \]

and

\[ Q = \begin{bmatrix} \sigma_t^2 & 0 & 0 \\ 0 & \sigma_\zeta^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{bmatrix} \]

Using the numerical values \( (\sigma_t^2, \sigma_\zeta^2, \sigma_\omega^2) = (1, 0.25, 1.25, 3) \), 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_i.c. 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
\[ y_t - 0.5y_{t-1} - 0.25y_{t-2} = u_t - 0.4u_{t-1} + 0.3u_{t-2} \] (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
\[ (1 - 0.5L + 0.25L^2) (1 - L)^d y_t = (1 - 0.3L) u_t \] (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
\[ y_t = \sum_{j=1}^{\infty} \theta_j u_{t-j} \]
then the autocovariances \( \gamma_i \) of the process (if we assume that \( \text{var}(u_t) = 1 \)) are equal to
\[ \gamma_i = \sum_{j=0}^{\infty} \theta_j \theta_{j+i} \]
The autocorrelations correspond to
\[ \rho_i = \frac{\gamma_i}{\gamma_0} \]
and the partial autocorrelations are obtained as the solution of the Toeplitz system
\[
\begin{bmatrix}
\gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{i-1} \\
\gamma_{i-1} & \gamma_0 & \gamma_1 & \cdots & \gamma_{i-2} \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\gamma_{i-1} & \gamma_{i-2} & \gamma_{i-3} & \cdots & \gamma_0
\end{bmatrix}
\begin{bmatrix}
\xi_1 \\
\xi_2 \\
\vdots \\
\xi_i
\end{bmatrix} =
\begin{bmatrix}
\gamma_1 \\
\gamma_2 \\
\vdots \\
\gamma_i
\end{bmatrix}
\]

44. **chirp1a.prg**
We define the linear chirp \( x_t = \sin(100\pi t^2) \). 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 \( B = (1, 2, 3, 3) \), we show that the reconstructed signal by applying the inverse wavelet packet transform is the same as the original signal.

46. **cm11-5.prg**
Illustration of the use of the CML package with TSM.

47. **cpgem1.prg**
Illustration of the CPGEM 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 **parse**. The model corresponds to
   \[
   \begin{align*}
   y_t &= \mu_t + \varepsilon_t \\
   \mu_t &= \mu_{t-1} + \eta_t
   \end{align*}
   \]
   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
   \[ W_t^* = \frac{\sum_{i=1}^{t} w_i^2}{\sum_{i=1}^{T} w_i^2} \]

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 _FD2d1 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 \texttt{wv} procedure. In a second step, we use a thresholding technique. Finally, we reconstruct the series by applying the \texttt{itrw} 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. **film1a.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 \( \theta = [\gamma_1 \cdots \gamma_r \sigma]^T \) with the FD_{ml} procedure. In this example, we have set \( r \) equal to 4.

60. **film1b.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. **film2a.prg**

We consider the model \( z_t \), defined by

\[
\begin{align*}
z_t &= x_t + y_t \\
x_t &= \phi_1 x_{t-1} + u_t \\
y_t &= v_t - \theta_1 v_{t-1}
\end{align*}
\]

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

\[
g(\lambda_j) = \sigma_u^2 \left| 1 - \frac{1}{1 - \phi_1 e^{i\lambda_j}} \right|^2 + \sigma_v^2 \left| 1 - \theta_1 e^{i\lambda_j} \right|^2
\]

\[
= \sigma_u^2 \left( 1 - 2\phi_1 \cos \lambda_j + \phi_1^2 \right) + \sigma_v^2 \left( 1 - 2\theta_1 \cos \lambda_j + \theta_1^2 \right)
\]

The vector of parameters is set to \( [\phi_1 \sigma_u \theta_1 \sigma_v]^T \).

62. **film2b.prg**

To see if \( \theta_1 = 0.7 \) in the above model, we use the LM and LR tests in the frequency domain.

63. **film3.prg**

The model is

\[
\begin{align*}
z_t &= x_t + y_t \\
x_t &= x_{t-1} + u_t - \theta_1 u_{t-1} \\
y_t &= v_t
\end{align*}
\]

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

\[
g(\lambda_j) = \sigma_u^2 \left| 1 - \theta_1 e^{i\lambda_j} \right|^2 + \sigma_v^2 \left| 1 - e^{i\lambda_j} \right|^2
\]

\[
= \left( 1 - 2\theta_1 \cos \lambda_j + \theta_1^2 \right) \sigma_u^2 + 2 (1 - \cos \lambda_j) \sigma_v^2
\]

Because the stationary form is \( z_t - z_{t-1} \), the data used in the FD_{ml} procedure are \( z - \text{lag}1(z) \).

64. **film4.prg**

Same example as kalman4.c.prg, but the spectral generating function is computed by the sgif_SSM procedure.

65. **film5-6.prg**

Examples of Maximum Likelihood of multivariate processes in the frequency domain.
66. filter1a.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. filter1b.prg
   Univariate ARMA process estimation with the \texttt{arma\_Filter} procedure.

68. filter1b-1c.prg
   Univariate ARMA-GARCH process estimation with the \texttt{arma\_Filter} and \texttt{garch\_Filter} procedures.

69. filter2a.prg
   Estimation of a Fractional ARMA(2,1) process with the \texttt{fractional\_Filter} procedure.

70. fsl1.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
   \[
   \beta_{1,t} = 1
   \]
   \[
   \beta_{2,t} = \sin \left( \frac{2\pi t}{N} \right) + v_{2,t}
   \]
   \[
   \beta_{3,t} = 0.9 \beta_{3,t-1} + v_{3,t}
   \]
   \( u_t, v_{2,t} \) and \( v_{3,t} \) are Gaussian processes. For the FLS regression, we pose
   \[
   \mu = \begin{bmatrix} 10000 \\ 1 \\ 1 \end{bmatrix}
   \]

71. fsl2.prg
   We graph the residual efficiency frontier \( \{(r_D^2(\mu), r_M^2(\mu)) : \mu \in \mathbb{R}_+\} \) of the preceding model.

72. fsl3.prg
   We consider the model
   \[
   y_t = x_t \beta_t + u_t
   \]
   with
   \[
   \beta_t = \begin{cases} 
   z & \text{if } t \leq S \\
   w & \text{if } t > S 
   \end{cases}
   \]
   \( u_t \) and \( \beta_t \) are Gaussian processes. For the FLS regression, we pose
   \[
   \mu = \begin{bmatrix} 10000 \\ 1 \\ 1 \end{bmatrix}
   \]

73. fsl4.prg
   An example to illustrate the convergence of the FLS estimates to the OLS estimates as \( \mu \) tends to \( +\infty \). We consider different values for \( \mu \): 10^4, 10^6, 10^7, 10^8, 4 \times 10^8 and 5 \times 10^8.

74. fractal1-4.prg — fractal.src
   Different examples to illustrate the estimation of the fractional parameter using wavelets. In \texttt{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 \texttt{fractal2.prg}, we consider a fractional process with \( d = 0.25 \). The examples \texttt{fractal3.prg} and \texttt{fractal4.prg} compute the empirical density of the wavelet and GH estimators.
75. **gfs1.png**
We compare the GFLS and FLS methods with each other on the following model for \( t = 1, \ldots, N \):

\[
y_t = \beta_1 t x_1 + \beta_2 t x_2 + \beta_3 t x_3 + u_t
\]

with \( \beta_1 \), a constant, \( \beta_2 \), a parameter with seasonal path and \( \beta_3 \), a time-varying parameter.

76. **gfs2.png**
An example to show that the FLS method is a special case of the GFLS method.

77. **gfs3.png**
We consider the following multi-dimensional process:

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

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

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

We can also estimate \( \beta_{1,t}, \beta_{2,t} \) and \( \beta_{3,t} \), with the GFLS filter. For the first estimation, we set \( D_i = I_3 \), \( M_i = I_2 \), \( Q_0 = I_3 \), \( P_0 = 0_3 \) and \( \mu = 1 \). For the second estimation, \( D_i \) is equal to:

\[
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & \frac{1}{10} & 0 \\
  0 & 0 & \frac{1}{10}
\end{bmatrix}
\]

and \( \mu \) is set to 10.

78. **gfs4.png**
In this example, we show the use of GFLS for the estimation of specific and common components. Suppose a two-dimensional process with:

\[
\begin{align*}
y_t &= s_{1t} + c_t + u_{1t} \\
x_t &= s_{2t} + c_t + u_{2t}
\end{align*}
\]

with \( c_t \) the common component of \( y_t \) and \( x_t \) while \( s_{1t} \) and \( s_{2t} \) are the two specific components. Let us consider the approximately linear system:

\[
\begin{align*}
\begin{bmatrix}
  y_t \\
  x_t
\end{bmatrix} & \simeq \begin{bmatrix}
  1 & 0 & 1 \\
  0 & 1 & 1
\end{bmatrix} \begin{bmatrix}
  \alpha_{1,t} \\
  \alpha_{2,t} \\
  \alpha_{3,t}
\end{bmatrix} + \begin{bmatrix}
  0 \\
  0
\end{bmatrix} \\
\begin{bmatrix}
  \alpha_{1,t+1} \\
  \alpha_{2,t+1} \\
  \alpha_{3,t+1}
\end{bmatrix} & \simeq \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  \alpha_{1,t} \\
  \alpha_{2,t} \\
  \alpha_{3,t}
\end{bmatrix} + \begin{bmatrix}
  0 \\
  0
\end{bmatrix}
\end{align*}
\]

Then, \( \alpha_{1,t} \) and \( \alpha_{2,t} \) can be viewed as the specific components and we can interpret \( \alpha_{3,t} \) as the common factor. Note that the choice of \( Q_0 \) and \( P_0 \) 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. **gfs5.png**
The local level model takes the approximately linear form:

\[
\begin{align*}
y_t & \simeq \beta_t \\
\beta_{t+1} & \simeq \beta_t
\end{align*}
\]

We compare the estimation of the state vector process obtained with the Kalman filter with that given by the GFLS filter (see **ill2.png** example).
80. gfls6.prg
The local linear trend model takes the approximately linear form:

\[
\begin{align*}
\begin{bmatrix}
y_t \\
\delta_{t+1} \\
\beta_{t+1}
\end{bmatrix} & \simeq \begin{bmatrix}
1 & 0 \\
0 & 1 \\
0 & 0
\end{bmatrix} \begin{bmatrix}
\delta_t \\
\beta_t
\end{bmatrix} + \begin{bmatrix}
0 \\
0
\end{bmatrix}
\end{align*}
\]

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

\[y_t = x_t \beta + u_t\]  \hspace{1cm} (3.10)

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

\[
\begin{align*}
E [u_t] &= 0 \\
E [u_t^2 - \sigma^2] &= 0 \\
E [u_t x_{t,i}] &= 0 \quad \forall \ i = 1, \ldots, 4 
\end{align*}
\]

Note that we use the analytical gradient to perform GMM.

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

83. gmm2a.prg
The model is

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

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

\[
\begin{align*}
E_t [u_t] &= 0 \\
E_t [u_t^2 - h_t^2] &= 0 \\
E_t [u_t x_{t,i}] &= 0 \\
E_t [(u_t^2 - h_t^2) u_{t-1}^2] &= 0
\end{align*}
\]

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

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

\[
\begin{align*}
\begin{cases}
\quad dx_t &= \mu x_t \ dt + \sigma x_t \ dW_t \\
\quad x(t_0) &= x_0
\end{cases}
\end{align*}\]  \hspace{1cm} (3.11)

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

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

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

\[\varepsilon_t = \ln \frac{x_t}{x_{t-1}} - \left( \mu - \frac{1}{2} \sigma^2 \right) h\]
We can estimate the vector of parameters $\theta = \text{vec} \begin{bmatrix} \mu & \sigma \end{bmatrix}$ 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

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

86. gmm3b.prg

We consider an Ornstein-Uhlenbeck process

$$
\begin{align*}
\begin{cases}
\frac{dx_t}{dt} &= a(b - x_t) \, dt + \sigma \, dW_t \\
x(t_0) &= x_0
\end{cases}
\end{align*}
\quad (3.12)
$$

The solution of the stochastic differential equation (3.12) is

$$
x(t) = x_0 e^{-a(t-t_0)} + b \left( 1 - e^{-a(t-t_0)} \right) + \sigma \int_{t_0}^{t} e^{a(t-t')} \, dW(t')
$$

We define

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

Let $\theta = \text{vec} \begin{bmatrix} a & b & \sigma \end{bmatrix}$ 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 $\theta$ can be performed by considering the following moment conditions

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

87. gmm3c.prg

Chan, Karolyi, Longstaff and Sanders [1992] consider the following stochastic differential equation

$$
\begin{align*}
\begin{cases}
\frac{dy_t}{dt} &= \left( a + \beta y_t \right) \, dt + \sigma |y_t| \, dW_t \\
y(t_0) &= x_0
\end{cases}
\end{align*}
\quad (3.13)
$$

To estimate the vector of parameters $\theta = \text{vec} \begin{bmatrix} a & \beta & \gamma & \sigma \end{bmatrix}$, they use the discrete-time model

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

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

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

to estimate $\theta$ with GMM. In this example, we simulate an Ornstein-Uhlenbeck. 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.png**
Parameters estimation of the Bernoulli, Binomial, Negative Binomial, Poisson, Gamma, Beta, Laplace-Gauss, Log-normal and Exponential distributions.

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

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

91. **golay1.png**
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.png**
An illustration of the Savitzky-Golay procedure applied to noisy data.

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

\[ \ln I(\lambda_j) = c - d \sin^2 \frac{\lambda_j}{2} + u_t \]  

(3.14)

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

94. **gph2.png**
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.png**
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.png**
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.png**
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.png**
Monte Carlo experiments of the standard errors of the GPH estimator and those based on the smoothing periodogram.

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

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

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

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

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, Teresovsky and Willinger* [1995].

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

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

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

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

\[
\begin{align*}
\begin{bmatrix}
    y_{1,t} \\
    y_{2,t} \\
    \alpha_t \\
    \beta_t
\end{bmatrix} &= \begin{bmatrix}
    1 & 0 \\
    0 & 1 \\
    0.5 & 0.3 \\
    0 & 0.2
\end{bmatrix} \begin{bmatrix}
    \alpha_t \\
    \beta_t
\end{bmatrix} + \begin{bmatrix}
    10 \\
    0 \\
    \beta_{t-1} \\
    \beta_{t-1}
\end{bmatrix} + \begin{bmatrix}
    1 \\
    1
\end{bmatrix} \eta_t \\
\end{align*}
\]

(3.15)

with

\[
H = E[\varepsilon_t \varepsilon_t'] = \begin{bmatrix}
    2 & 0 \\
    0 & 1
\end{bmatrix}
\]

(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. \( a_0 \) 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 \( \alpha_t \) with its 95% confidence interval.

118. **kalman1g.prg**
Graphical representation of the log-likelihood vector.
119. kalman1h.prg
Exact Maximum likelihood estimation of the model
\[
\begin{bmatrix}
y_{1,t} \\
y_{2,t} \\
\alpha_t \\
\beta_t 
\end{bmatrix}
= \begin{bmatrix}
\theta_1 & 0 \\
0 & \theta_2 \\
\theta_6 & \theta_7 \\
0 & \theta_8 
\end{bmatrix}
\begin{bmatrix}
\alpha_{t-1} \\
\beta_{t-1} 
\end{bmatrix}
+ \begin{bmatrix}
\theta_3 \\
0 \\
\theta_9 \\
0 
\end{bmatrix}
\epsilon_t + \begin{bmatrix}
1 \\
1 
\end{bmatrix}
\eta_t
\]
(3.17)

with
\[
H = \begin{bmatrix}
\theta_4 & 0 \\
0 & \theta_5 
\end{bmatrix}
\]
(3.18)
and \( Q = \theta_{11} \).

120. kalman1i.prg
Conditional MLE with \( a_0 = 0 \) and \( P_0 = 0_{2 \times 2} \) in the time-variant form. Note the use of external variables.

121. kalman1j.prg
Smoothing of the estimated model.

122. kalman1k.prg
Forecasting of the estimated model.

123. kalman2a.prg
Maximum likelihood estimation of the state space model
\[
\begin{bmatrix}
y_t \\
\beta_0_t \\
\beta_1_t \\
\beta_2_t 
\end{bmatrix}
= \begin{bmatrix}
1 \\
0 \\
0 \\
0 
\end{bmatrix}
\begin{bmatrix}
\beta_{0,t-1} \\
\beta_{1,t-1} \\
\beta_{2,t-1} 
\end{bmatrix}
+ \begin{bmatrix}
\epsilon_t \\
\eta_t \\
\eta_t \\
\eta_t 
\end{bmatrix}
\]
(3.19)
The unknown parameters \( \theta \) are \( H = \theta_1^2 \) and
\[
Q = \begin{bmatrix}
\theta_2^2 & 0 & 0 \\
0 & \theta_3^2 & 0 \\
0 & 0 & \theta_4^2 
\end{bmatrix}
\]
(3.20)
a_0 and \( P_0 \) are set to the null vector and matrix respectively.

124. kalman2b.prg
Same program as kalman2a.prg, but \( a_0 \) and \( P_0 \) are fixed differently. This program shows the initialization problem of the Kalman filter.

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

126. kalman3b.prg
Conditional maximum likelihood of the corresponding state space model
\[
\begin{bmatrix}
y_t \\
\beta_t \\
\eta_t 
\end{bmatrix}
= \begin{bmatrix}
x_t \\
\epsilon_t \\
\eta_t 
\end{bmatrix}
\]
(3.21)
The estimated parameters are \( \phi_1 \), \( \theta_1 \), \( \sigma_\epsilon \) and \( \sigma_\eta \).
127. kalman3c.prg
Conditional maximum likelihood of another representation of the above state space model

\[
\begin{align*}
\begin{bmatrix}
y_t \\
\eta_t \\
\varepsilon_t
\end{bmatrix} &= \begin{bmatrix}
\beta_t \\
\theta_t \\
\varepsilon_t
\end{bmatrix} \\
\begin{bmatrix}
\beta_t \\
\theta_t \\
\varepsilon_t
\end{bmatrix} &= \begin{bmatrix}
\phi_1 & -\theta_1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
\beta_{t-1} \\
\theta_{t-1} \\
\varepsilon_{t-1}
\end{bmatrix} + \begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix} \begin{bmatrix}
\eta_t \\
\varepsilon_t
\end{bmatrix}
\end{align*}
\]

This program is an illustration of the identification problem.

128. kalman4a.prg
Suppose that we observe a process \( y_t \) with a measurement error \( \varepsilon_t \). We note \( z_t \) the observed process. We have

\[
\begin{align*}
z_t &= y_t + \varepsilon_t \\
\end{align*}
\]

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

\[
\begin{align*}
y_t &= \phi_1 y_{t-1} + u_t - \theta_1 u_{t-1}
\end{align*}
\]

The state space form of this model is

\[
\begin{align*}
\begin{bmatrix}
z_t \\
y_t \\
u_t
\end{bmatrix} &= \begin{bmatrix}
1 & 0 \\
\phi_1 & -\theta_1 \\
0 & 0
\end{bmatrix} \begin{bmatrix}
y_{t-1} \\
u_{t-1}
\end{bmatrix} + \begin{bmatrix}
1 \\
0
\end{bmatrix} u_t + \varepsilon_t \\
\end{align*}
\]

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 \( \phi_1, \theta_1, \sigma_u \) and \( \sigma_\varepsilon \) 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

\[
\begin{align*}
g(\lambda_j) &= \sigma_u^2 \frac{1 - 2\theta_1 \cos \lambda_j + \theta_1^2}{1 - 2\phi_1 \cos \lambda_j + \phi_1^2 + \sigma_\varepsilon^2}
\end{align*}
\]

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

\[
\begin{align*}
\begin{bmatrix}
\phi_1 \\
\theta_1 \\
\sigma_u \\
\sigma_\varepsilon
\end{bmatrix} &= \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\phi_1 \\
\sigma_u \\
\sigma_\varepsilon
\end{bmatrix} + \begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\end{align*}
\]

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 \( a_{1T} \). If the variance of \( \varepsilon_1 \) is zero, then we must verify that the first component of \( a_{1T} \) is just equal to \( z_t \) or \( y_t \).
134. **kalman4.prg**  
Smoothing the model (3.24) with the Kalman filter.

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

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

(3.27)

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

\[
\begin{bmatrix}
  \text{vec}(\Phi_1) \\
  \text{vec}(\Phi_2) \\
  \mu \\
  \text{vech}(P)
\end{bmatrix}
\]

with \( \text{vech} \) 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 \( \Phi_1 \) and \( \Phi_2 \) are of the form

\[
\begin{bmatrix}
  0 & 0 & 0 \\
  \vdots & \ddots & \vdots \\
  0 & 0 & 0
\end{bmatrix}
\]

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:

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

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 TD_ml_derivatives procedure.

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

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

\[
y_t = \beta_0,t x_{0,t} + \beta_1,t x_{1,t} + u_t
\]

(3.28)

with \( u_t \sim \mathcal{N}(0, \sigma^2_u) \) and

\[
\begin{cases}
  \beta_{0,t} = \beta_{0,t-1} + \nu_0 \\
  \beta_{1,t} = \beta_{1,t-1} + \nu_1
\end{cases}
\]

(3.29)

with \( \begin{bmatrix} \nu_0 \\
  \nu_1 \end{bmatrix} \sim \mathcal{N}(0, \Sigma_v) \). We suppose that

\[
\Sigma_v = \begin{bmatrix}
  \sigma^2_0 & 0 \\
  0 & \sigma^2_1
\end{bmatrix}
\]
The state space form of the model (3.28-3.29) is

\[
\begin{align*}
    y_t &= \begin{bmatrix} x_{0,t} \\ x_{1,t} \end{bmatrix} \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \end{bmatrix} + u_t \\
    \begin{bmatrix} \beta_{0,t} \\ \beta_{1,t} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{0,t-1} \\ \beta_{1,t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} 
\end{align*}
\]  

(3.30)

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

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

143. \texttt{kfgain1-2.prg}
Illustration of the \texttt{KF_gain} procedure.

144. \texttt{kernel1.prg}
Density estimation (normal random number).

145. \texttt{kernel2.prg}
Density estimation ($\chi_2$ random number) with the truncated (at left) normal kernel.

146. \texttt{kernel3.prg}
Density estimation (uniform random number) with the truncated normal (at left and right) kernel.

147. \texttt{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. \texttt{kpss.prg — kpss.src}
KPSS statistic.

149. \texttt{ks1.prg}
Computes the Kolmogorov-Smirnov test for a white noise process presented in \textsc{Brockwell} and \textsc{Davis} [1991].

150. \texttt{ks2.prg}
Computes the Kolmogorov-Smirnov test for a unit root process.

151. \texttt{ks3.prg}
Computes the Kolmogorov-Smirnov test for the random walk plus noise model applied to the \texttt{parse} data.

152. \texttt{ll1.prg}
Estimates the Local Level model for the \texttt{parse} data in the frequency domain with the method of scoring and the BFGS algorithm.

153. \texttt{ll2.prg}
Estimates the unobserved component of the \texttt{parse} Local Level model.

154. \texttt{ll3.prg}
Estimates the \texttt{parse} Local Level model in the time domain. This program shows the importance of the choice of the initial conditions.

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

156. \texttt{llt2.prg}
Estimates the unobserved component of the \texttt{gnp} Local Linear model.

157. \texttt{llt3.prg}
Estimates the \texttt{gnp} Local Linear model in the time domain with the BHHH algorithm.
158. **matrix1.prg**
Computes the matrices $L_4$, $D_4$ and $K_{4,3}$.

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

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

\[
L_mD_m = I_{m(m+1)/2}
\]

\[
K_{m,m}D_m = D_m
\]

\[
K_{m,1} = K_{1,m} = I_m
\]

\[
\text{trace}(K_{m,m}) = m
\]

\[
\text{trace} (D_m^T D_m) = m^2
\]

\[
L^*_mL_m = I_{m(m+1)/2}
\]

\[
\text{trace} (D_m^T D_m)^{-1} = \frac{m(m+3)}{4}
\]

161. **matrix4-5.prg**
An illustration of the $xpml2$ 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. **optimum2a-2c.prg**
Some examples with the $optimum2$ 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**
Covarogram of a time series using the $PDGM$ and $inverse_fourier$ procedures.

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

\[
\begin{bmatrix}
  y_{1,t} \\
  y_{2,t} \\
  \alpha_{1,t} \\
  \alpha_{2,t}
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 \\
  0 & 1 \\
  0.5 & 0.3 \\
  0 & -0.5
\end{bmatrix}
\begin{bmatrix}
  \alpha_{1,t} \\
  \alpha_{2,t}
\end{bmatrix}
+ \begin{bmatrix}
  \varepsilon_t \\
  \varepsilon_t
\end{bmatrix}
\]

\[
\begin{bmatrix}
  \alpha_{1,t-1} \\
  \alpha_{2,t-1}
\end{bmatrix}
+ \begin{bmatrix}
  1 \\
  1
\end{bmatrix} \eta_t
\]

(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.png
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.png
Spectral density of a univariate ARMA(2,1) process.

173. qmfl.png
In this example, we verify the following properties of quadrature mirror filters:
\[
\begin{align*}
\sum_{k=0}^{P} h_k &= \sqrt{2} \\
\sum_{k=0}^{P} g_k &= 0 \\
\sum_{k=0}^{P} h_k^2 &= 1 \\
\sum_{k=0}^{P} h_k g_k &= 0
\end{align*}
\]

174. qmfl2.png
Same program as qmfl.png with Pollen filters.

175. riccat1.png
Solves the Algebraic Riccati Equation for the state space model given in exercise 4.8 by Harvey [1990].

176. rls1.png
We apply the RLS procedure to a time-invariant model.

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

178. robinson.png — robinson.src

179. scalogram.png
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.png
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.png
Printing state space models.
182. ss\textit{m6a.prg}
Same program as \textit{varxl.e.prg}, but responses to forecast errors are computed with the \textit{SSM\_impulse} procedure.

183. ss\textit{m6b.prg}
Same program as \textit{varxl.e.prg}, but responses to orthogonal impulses are computed with the \textit{SSM\_orthogonal} procedure.

184. ss\textit{m6c.prg}
Same program as \textit{varxl.d.prg}, but we compute the forecast error variance decomposition with the \textit{SSM\_fevd} procedure.

185. ss\textit{m7a.prg}
Same program as \textit{arma1k.prg}, but responses to forecast errors are computed with the \textit{SSM\_impulse} procedure.

186. ss\textit{m7b.prg}
Same program as \textit{arma1k.prg}, but responses to orthogonal impulses are computed with the \textit{SSM\_orthogonal} procedure.

187. ss\textit{m7c.prg}
Same program as \textit{arma1j.prg}, but we compute the forecast error variance decomposition with the \textit{SSM\_fevd} procedure.

188. ss\textit{m8a.prg}
We consider the state space model
\begin{equation}
\begin{bmatrix}
  y_{1,t} \\
  y_{2,t} \\
  y_{3,t}
\end{bmatrix} =
\begin{bmatrix}
  1 & 1 & 0 \\
  4 & 2 & 0 \\
  2 & 3 & 0 
\end{bmatrix}
\begin{bmatrix}
  \alpha_{1,t} \\
  \alpha_{2,t}
\end{bmatrix}
+ \varepsilon_t
\end{equation}
with
\[\varepsilon_t \sim \mathcal{N}\left(\mathbf{0}_3, \begin{bmatrix}
  5 & 1 & 0 \\
  1 & 4 & 0 \\
  0 & 0 & 8
\end{bmatrix}\right)\]
and
\[\begin{bmatrix}
  \eta_{1,t} \\
  \eta_{2,t}
\end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}_2, \begin{bmatrix}
  2 & 0.5 \\
  0.5 & 1
\end{bmatrix}\right)\]
We compute the responses to the forecast error \(\mathbf{e} = [1 \quad 0]^\top\).

189. ss\textit{m8b.prg}
We compute the responses to the forecast error \(\mathbf{e} = [1 \quad -1]^\top\) for the state space model (3.32).

190. ss\textit{m9a.prg}
We compute the responses to the orthogonal impulse \(\mathbf{e} = [1 \quad 0]^\top\) for the state space model (3.32).

191. ss\textit{m9b.prg}
We compute the responses to the orthogonal impulse \(\mathbf{e} = [1 \quad -1]^\top\) for the state space model (3.32).

192. ss\textit{m10.prg}
We compute the forecast error variance decomposition for the state space model (3.32).

193. \textit{surrog1.prg}
Surrogate data in the univariate case.

194. \textit{surrog2.prg}
Surrogate data in the multivariate case.
195. **surrog3.prn — rk4.src**

Surrogate data can be used to detect non-linearities. In this example, we use the Lorenz model defined by

\[
\begin{align*}
\frac{dx}{dt} &= \sigma (y - x) \\
\frac{dy}{dt} &= -xz + Rz - y \\
\frac{dz}{dt} &= xy - \beta z
\end{align*}
\]

196. **tdm11a.prn**

TERÄSVIRTA [1994] suggests a LSTAR model to fit the lynx data

\[
x_t = \beta_1 x_{t-1} + [\beta_2 x_{t-2} + \beta_3 x_{t-3} + \beta_4 x_{t-4} + \beta_5 x_{t-5} + \beta_6 x_{t-11}] \\
\times [1 + \exp (\rho \times 1.8 (x_{t-3} - \theta))]^{-1} + u_t
\]

This program estimates the model (3.33). Note the use of the external variable `tsm_parm` for the names of the estimated coefficients.

197. **tdm12a.prn**

To model the lynx data, OZAKI [1982] suggests to use the EXPAR model

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

with \( u_t \sim \mathcal{N}(0, \sigma^2) \). With the `TD ML` procedure, we estimate the coefficients \( \theta = \left[ \begin{array}{ccc} \beta^T & \delta & \sigma \end{array} \right]^T \).

198. **tdm12b.prn**

This is a modification of the `tdm12a.prn` example by setting \( \delta \) to 3.89.

199. **tdm13a.prn**

Maximum Likelihood estimation of a linear model.

200. **tdm13b.prn**

Maximum Likelihood estimation of a linear model under linear restrictions.

201. **tdm14a.prn**

Maximum Likelihood of the linear model with AR(1) errors:

\[
\begin{align*}
y_t &= x_t \beta + u_t \\
u_t &= \rho u_{t-1} + \varepsilon_t
\end{align*}
\]

with \( \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \). The parameter vector \( \theta \) is \( \left[ \begin{array}{c} \beta^T \\rho \\sigma \end{array} \right]^T \). The ML function is based on BEACH and MACKINNON [1978]. We test also \( \rho = 0 \) with LM and LR statistics.

202. **tdm14b.prn**

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.prn**

An illustration of the `fourier2` procedure.

204. **varx1a.prn**

Define the following series with the Lutkepohl data

\[
\begin{align*}
y_{1,t} &= INV(t) - INV(t - 1) \\
y_{2,t} &= INC(t) - INC(t - 1) \\
y_{3,t} &= CONS(t) - CONS(t - 1)
\end{align*}
\]

The program estimates the VAR(2) process

\[
\begin{bmatrix}
y_{1,t} \\
y_{2,t} \\
y_{3,t}
\end{bmatrix} = \Phi_1 \begin{bmatrix}
y_{1,t-1} \\
y_{2,t-1} \\
y_{3,t-1}
\end{bmatrix} + \Phi_2 \begin{bmatrix}
y_{1,t-2} \\
y_{2,t-2} \\
y_{3,t-2}
\end{bmatrix} + \begin{bmatrix}
\mu_1 \\
\mu_2 \\
\mu_3
\end{bmatrix} + \varepsilon_t
\]

(3.34)
and performs a stability analysis. The $\theta$ vector of coefficients corresponds to

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

205. varx1b.prg
Computes the Wald test for no Granger-causality from INC/CONS to INV.

206. varx1c.prg
Computes the Wald test for no Instantaneous-causality between INC/CONS and INV.

207. varx1d.prg
Forecast Error Variance Decomposition of the above VAR(2) model.

208. varx1e.prg
Impulse Responses of the above VAR(2) model.

209. varx1f.prg
Impulse 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

$$
\mu = \begin{bmatrix}
0 \\
\cdot
\end{bmatrix}
$$

212. varx2a.prg
Estimation of the Dynamic Simultaneous Equations

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

(3.35)

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

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

(3.36)

214. varx2c.prg
Estimation of the model (3.36) by Maximum Likelihood.

215. varx3a.prg
Use of the varx_1s procedure to compute OLS estimates. The results are compared with those calculated with the ols procedure.
3.2  TSM

216. varx3b.prg
We use the *varx.cls* procedure to compute the SUR estimator. The example is taken from JUDGE, HILL,

217. varx3c.prg
We use the *varx.cls* procedure to compute the restricted SUR estimator. The example is taken from JUDGE,

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
Some 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 \( \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \). Then we plot the empirical distribution of \( 2\frac{f(\lambda_t)}{g(\lambda_t)} \) and the theoretical \( \chi^2_\nu \) distribution.

221. wn2.prg
This is the same program as *wn1.prg* applied to a unit root process.

222. wn3.prg
We check if the FRF/DEM is a unit root process.

223. wpkt1.prg
We select a wavelet packet basis for a time series with the *BestBasis* and the *BestLevel* procedures based
on the entropy cost function.

224. wpkt2.prg
We simulate a fractional process. Then, we draw the wavelet packet table of this process. We illustrate the
fact that the wavelet transform is a special case of the wavelet packet transform with a special basis.

225. wt1a.prg-wt1b.prg
We evaluate the inverse wavelet transform for different unit vectors \( \mathbf{e} \) to see how wavelets look like (see PRESS,
TEUKOLSKY, VETTERLING and FLANNERY [1992], page 591).

226. wt2.prg
Reconstruction of an ARMA process by the quantile thresholding method with different values of \( p \).

227. wt3.prg
An important result is that the “mother” coefficient of the wavelet transform of a time series \( x_t \) of length
\( N = 2^M \) is equal to

\[ c_0 = \sum_{i=1}^{N} x_i \sqrt{2^{-i/2}} \]

3.2.2  Time series

3.2.2.1  Arfima process

```plaintext
new;
library tsml, optimum;
output file = tsm1a.out reset;
rnseed 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];
@----- first estimation --->@

cnt = 0[0|1|0];
/* No fixed parameters */
	/_tom_Mcov = 1; /* Hessian Matrix Covariance */
	/_fourier = 0; /* Fast fourier transform */
	/_tom_optnum = 0; /* scoring algorithm */
	/_output = 1;
	/_print = 1;

{beta1, stderr, Mcov, Logll} = arfima(y,1,1,sv,cnt);
	/_tom_optnum = 1; /* BFGS algorithm */
	/_output = 0;
	/_print = 0;

@----- second estimation --->@

cnt = 0[0|1|0];
/* d is fixed */

{beta2, stderr, Mcov, Logll2} = arfima(y,1,1,sv,cnt);

@----- third estimation --->@

cnt = 0[0|1|0];
/* d and MA parameters are fixed */

@----- likelihood ratio tests --->@

LR = 2*(Log11-log12); /* Likelihood ratio */

pvalue = omvchi2(LR,1); /* Approximating the noncentral chi-squared CDF */

print;
print 'Testing d = 0.25'; print chs(45*ones(40,1));
print ftest(LR,'Likelihood ratio statistic: %lf',10,5);
print ftest(pvalue,'p-value: %lf',10,5);
print;

LR = 2*(Logll-log13); /* Likelihood ratio */

pvalue = omvchi2(LR,2); /* Approximating the noncentral chi-squared CDF */

print 'Testing d = 0.25 and theta1 = 0.4'; print chs(45*ones(40,1));
print ftest(LR,'Likelihood ratio statistic: %lf',10,5);
print ftest(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: -1093.835392

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>phi1</td>
<td>0.860292</td>
<td>0.069672</td>
<td>12.720846</td>
<td>0.000000</td>
</tr>
<tr>
<td>theta1</td>
<td>0.354294</td>
<td>0.147613</td>
<td>2.399862</td>
<td>0.016765</td>
</tr>
<tr>
<td>d</td>
<td>0.695775</td>
<td>0.211379</td>
<td>3.302955</td>
<td>0.000928</td>
</tr>
<tr>
<td>sigma</td>
<td>2.048615</td>
<td>0.064920</td>
<td>31.869922</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

Testing d = 0.25
----------------------------------------
Likelihood ratio statistic: 0.52048
p-value: 0.47084

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.
3.2. TSM

/*
** 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 tm_optum;

output file = tsm_b.out reset;
@--- data --->>

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

invent = reinsel[.,1];
inv = reinsel[.,2];
di = invent - log1(inv);
y = di*inv;

_print = 1;
_tsm_optum = 0; /* Analytical Newton-Raphson algorithm */
tsm_tol = 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|1|0|4); r = zeros(8,1);

sv = 0; /* Lutkepohl method */

{beta2, stderr2, Mcov2, LogL2} = arma_GML(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 = cdfchi(LR,4); /* Approximating the noncentral chi-squared CDF */

print;
print format(LR,'likelihood ratio statistic: \%lf',10,5);
print format(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)
*/
\begin{verbatim}
LM = G'inv(I)\*G;
pvalue = offchic(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];
ret[p(r)];
endp;

Dr = gradp(restrict,beta);
r = restrict(beta);
wald = r@invpd(\*Dr@cov(\*Dr));
pvalue = offchic(wald,4);
print;
print ftos(wald,'"Wald statistic: \%lf"',10,5);
print ftos(pvalue,'"p-value: \%lf"',10,5);
print;

@<--- Responses to Forecast Errors (unrestricted model) --->@

call arma_impulse(beta1,1,1,8);

x = miss(zeros(2,1),0);
mask = ones(1,5);
let fmt[5,3]=
  '1*+',  10 3
  '1*+',  10 3
  '1*+',  10 3
  '1*+',  10 3
  '1*+',  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@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 printf(w,mask,fmt);
print;
i = i+1;
enddo;

print;
print;

@<--- Responses to Orthogonal Impulses (unrestricted model) --->@

let fmt[5,3]=
  '1*+',  10 2
  '1*+',  10 2
  '1*+',  10 2
  '1*+',  10 2
  '1*+',  10 2;
call arma_orthogonal(beta1,1,1,SIGMA,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@y;
  print chrs(45@ones(79,1));
\end{verbatim}
3.2. TSM

```
print fo(i,'Periods: %d',1,0);
print chs(45*mes(79,1));
print 'Estimated responses
THETA matrix
KSI matrix
print chs(45*mes(79,1));
call printf(w,mask,fmt);
print;
i = i+1;
end;
```

output off;

Total observations: 100
Usable observations: 99
Number of parameters to be estimated: 8
Degrees of freedom: 50
Value of the maximized log-likelihood function: -501.39229

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR1_1</td>
<td>0.348233</td>
<td>0.168432</td>
<td>2.062351</td>
<td>0.039239</td>
</tr>
<tr>
<td>AR1_2</td>
<td>0.609844</td>
<td>0.226863</td>
<td>2.771672</td>
<td>0.006012</td>
</tr>
<tr>
<td>MA1_1</td>
<td>-0.512994</td>
<td>0.046649</td>
<td>-3.97591</td>
<td>0.049284</td>
</tr>
<tr>
<td>MA1_2</td>
<td>-0.222674</td>
<td>0.174879</td>
<td>-1.27311</td>
<td>0.206735</td>
</tr>
<tr>
<td>MA1_1</td>
<td>0.229667</td>
<td>0.148177</td>
<td>0.802141</td>
<td>0.424841</td>
</tr>
<tr>
<td>MA1_2</td>
<td>-0.177838</td>
<td>0.069981</td>
<td>-2.543321</td>
<td>0.051667</td>
</tr>
<tr>
<td>MA1_2</td>
<td>0.429738</td>
<td>0.113043</td>
<td>1.368975</td>
<td>0.053513</td>
</tr>
</tbody>
</table>

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: -531.86787

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR1_1</td>
<td>0.284297</td>
<td>0.182115</td>
<td>1.561083</td>
<td>0.121865</td>
</tr>
<tr>
<td>AR1_2</td>
<td>0.000000</td>
<td>0.000000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>AR1_2</td>
<td>0.000000</td>
<td>0.000000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>MA1_1</td>
<td>0.181657</td>
<td>0.051069</td>
<td>1.733942</td>
<td>0.041000</td>
</tr>
<tr>
<td>MA1_2</td>
<td>0.000000</td>
<td>0.000000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>MA1_2</td>
<td>0.198715</td>
<td>0.114039</td>
<td>1.737877</td>
<td>0.086028</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

Likelihood ratio statistic: 23.99116
p-value: 0.00008

Lagrange multiplier: 22.77046
p-value: 0.00014

Wald statistic: 26.75902
p-value: 0.00004

```
```

Periods: 0

<table>
<thead>
<tr>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Periods: 1

<table>
<thead>
<tr>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>0.571</td>
<td>0.160</td>
</tr>
<tr>
<td>0.407</td>
<td>0.606</td>
</tr>
</tbody>
</table>

Periods: 2
### CHAPTER 3. FINANCIAL ECONOMETRICS

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>0.191</td>
<td>0.045</td>
<td>1.762</td>
</tr>
<tr>
<td>0.722</td>
<td>0.618</td>
<td>1.129</td>
</tr>
</tbody>
</table>

Periods: 3

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>0.053</td>
<td>0.004</td>
<td>1.616</td>
</tr>
<tr>
<td>0.737</td>
<td>0.552</td>
<td>1.666</td>
</tr>
</tbody>
</table>

Periods: 4

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>0.005</td>
<td>-0.009</td>
<td>1.621</td>
</tr>
<tr>
<td>0.009</td>
<td>0.470</td>
<td>2.524</td>
</tr>
</tbody>
</table>

Periods: 5

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>-0.010</td>
<td>-0.012</td>
<td>1.611</td>
</tr>
<tr>
<td>0.561</td>
<td>0.392</td>
<td>3.085</td>
</tr>
</tbody>
</table>

Periods: 6

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>-0.014</td>
<td>-0.011</td>
<td>1.797</td>
</tr>
<tr>
<td>0.468</td>
<td>0.354</td>
<td>3.553</td>
</tr>
</tbody>
</table>

Periods: 7

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>-0.013</td>
<td>-0.010</td>
<td>1.783</td>
</tr>
<tr>
<td>0.386</td>
<td>0.267</td>
<td>3.989</td>
</tr>
</tbody>
</table>

Periods: 8

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PHI matrix</td>
<td>PSI matrix</td>
</tr>
<tr>
<td>-0.012</td>
<td>-0.008</td>
<td>1.772</td>
</tr>
<tr>
<td>0.318</td>
<td>0.219</td>
<td>4.257</td>
</tr>
</tbody>
</table>

Periods: 9

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>THETA matrix</td>
<td>KSI matrix</td>
</tr>
<tr>
<td>2.39e+000</td>
<td>0.000e+000</td>
<td>2.39e+000</td>
</tr>
<tr>
<td>1.03e+000</td>
<td>4.09e+000</td>
<td>1.03e+000</td>
</tr>
</tbody>
</table>

Periods: 1

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>THETA matrix</td>
<td>KSI matrix</td>
</tr>
<tr>
<td>1.53e+000</td>
<td>6.53e+000</td>
<td>3.91e+000</td>
</tr>
<tr>
<td>1.59e+000</td>
<td>2.48e+000</td>
<td>2.62e+000</td>
</tr>
</tbody>
</table>

Periods: 2

<table>
<thead>
<tr>
<th></th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>THETA matrix</td>
<td>KSI matrix</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
3.2. TSM

<table>
<thead>
<tr>
<th>Periods: 3</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>1.32e-001 1.72e-02</td>
<td>4.66e-000 8.52e-001</td>
<td></td>
</tr>
<tr>
<td>2.33e+001 2.26e+000</td>
<td>7.30e+000 1.14e+01</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Periods: 4</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>3.45e-003 3.53e-02</td>
<td>4.56e-000 8.17e-001</td>
<td></td>
</tr>
<tr>
<td>2.06e+000 1.82e+000</td>
<td>9.36e+000 1.33e+01</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Periods: 5</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>-3.63e-002 4.74e-02</td>
<td>4.52e+000 7.69e-001</td>
<td></td>
</tr>
<tr>
<td>1.74e+000 1.60e+000</td>
<td>1.11e+001 1.49e+001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Periods: 6</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>-4.44e-002 4.58e-02</td>
<td>4.47e+000 7.23e-001</td>
<td></td>
</tr>
<tr>
<td>1.45e+000 1.35e+000</td>
<td>1.26e+001 1.62e+001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Periods: 7</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>-4.19e-002 4.01e-02</td>
<td>4.43e+000 6.83e-001</td>
<td></td>
</tr>
<tr>
<td>1.20e+000 1.08e+000</td>
<td>1.37e+001 1.73e+001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Periods: 8</th>
<th>Estimated responses</th>
<th>Accumulated responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA matrix</td>
<td>KSI matrix</td>
<td></td>
</tr>
<tr>
<td>-3.64e-002 3.38e-02</td>
<td>4.38e+000 6.49e-001</td>
<td></td>
</tr>
<tr>
<td>9.64e+001 8.95e+001</td>
<td>1.47e+01 1.82e+01</td>
<td></td>
</tr>
</tbody>
</table>

3.2.2.3 VarX Process

- VAR order selection with the BIC, AIC alpha, SIC, FPE, AIC and HQ criteria.

```c
/*
 ** LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
 ** Springer-Verlag, Berlin-Heidelberg
 **
 ** VAR order selection
 ** See LUTKEPOHL, chapter 4
 ** */

new;
lifetime tm, optima;
output file = tsm.c.out reset;
cls;
@<<< data --->>
load y[R,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;
@end
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] = 1*(K-2)*m*ln(T)/T;   /* BIC */
   CR[m,2] = 1+3*m*(K-2)/T;      /* AIC alpha = 3 */
   CR[m,3] = ln(1+m*(K-2)/T)/(1-m*(K-2)/2)/T; /* AICc */
   CR[m,4] = D*(1+2*m*(K-2)/T)^K;    /* SIC */
   CR[m,5] = D*((T*K+1)/(T*K-1))^K;   /* MPFE */
   CR[m,6] = 1+2*(K-2)*m/T;         /* AIC */
   CR[m,7] = 1+2*(K-2)*m*ln(T)/T;   /* HQ */

   m = m + 1;
end;
P = min(ind(CR));
_print = old;
if _print == 1;
   m = seq(1,1,p_max);
   print chrs(45*omes(79,1));
   print 'Lags' BIC AICa AICc SIC FPE AIC'' \  
   'HQ''
   print chrs(45*omes(79,1));
   omat = ''Gp1'' 'p';
   mask = 0''1''1''1''1''1; 1;
   let fmt[6,3] =
   1''*''44 4 4
   1''*''2f' 8 0
   1''*''2f' 10 0
   1''*''2f' 10 0
   1''*''2f' 12 0
   1''*''2f' 12 0
   1''*''2f' 10 0
   call printf(fmt,omat,mask,fmt); print; print chrs(45*omes(79,1));
   omat = m'CR;
   mask = 1;
   let fmt[6,3] =
   1''*''2f' 4 0
   1''*''2f' 10 3
   1''*''2f' 10 3
   1''*''2f' 10 3
3.2. **TSM**

```
'111.1211' 12 3
'111.1211' 12 3
'111.1211' 10 3
'111.1211' 10 3;
call printf(cmat,mask,fmt);
endif;
return (GR,p);
endp;

<table>
<thead>
<tr>
<th>Lags</th>
<th>BIC</th>
<th>AICa</th>
<th>AICc</th>
<th>SIC</th>
<th>PFE</th>
<th>AIC</th>
<th>HQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-24.213</td>
<td>-24.372</td>
<td>-23.498</td>
<td>0.000</td>
<td>0.000</td>
<td>-24.493</td>
<td>-24.382</td>
</tr>
<tr>
<td>2</td>
<td>-24.067</td>
<td>-24.365</td>
<td>-23.526</td>
<td>0.000</td>
<td>0.000</td>
<td>-24.632</td>
<td>-24.407</td>
</tr>
<tr>
<td>3</td>
<td>-23.876</td>
<td>-24.054</td>
<td>-23.072</td>
<td>0.000</td>
<td>0.000</td>
<td>-24.425</td>
<td>-24.085</td>
</tr>
<tr>
<td>4</td>
<td>-23.210</td>
<td>-23.850</td>
<td>-22.480</td>
<td>0.000</td>
<td>0.000</td>
<td>-24.357</td>
<td>-23.901</td>
</tr>
<tr>
<td>5</td>
<td>-22.895</td>
<td>-23.697</td>
<td>-21.367</td>
<td>0.000</td>
<td>0.000</td>
<td>-23.340</td>
<td>-23.766</td>
</tr>
</tbody>
</table>

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, optsum, pgraph;

output file = tsmd.out reset;
cls;

@<<< data ==> data
load y[92,3] = lutkepoh.asc;
y = ln(y[1:76,1]);
INVEST = y[.,1];
dinv = INVEST-lag(1)(INVEST);
INCOME = y[.,2];
dinc = INCOME-lag(1)(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lag(1)(CONSUM);
data = dinv'.dinc'dcons;

@<<< LS estimation ==> print
{theta, stderr, Mcov, L0D11} = varx_LS(data,1,2); /* Constant and 2 lags */
print;
print '';''SIGMA:'';
print _varx_sigma;

@<<< Stability analysis ==> print
beta = theta[1:16]; /* 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));
cmat = roots'
mask = 1';
let fmt[2,3] =
'111.1211' 14 5
'111.1211' 14 5;
call printf(cmat,mask,fmt);
```
CHAPTER 3. FINANCIAL ECONOMETRICS

graphset;
_plt = ' ', _prr = 1, _pframe = {0,0};
title('Stability of the VAR(2) process');
'\nRoots of the reverse characteristic polynomial');
scale(-3|3, -3|3);

r = real(roots); i = imag(roots); N = rows(roots);
_psym = r+i*ones(Nr,5).*[(11.5 21.1)];
t = seqa(0,2*pi/100,101); x = cos(t); y = sin(t);
grphplt('c-1 -cf=tms1d.eps');

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: 595.2686

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>-0.316631</td>
<td>0.125456</td>
<td>-2.547746</td>
<td>0.013837</td>
</tr>
<tr>
<td>P02</td>
<td>0.043931</td>
<td>0.031659</td>
<td>1.378910</td>
<td>0.173832</td>
</tr>
<tr>
<td>P03</td>
<td>-0.002929</td>
<td>0.025676</td>
<td>-0.094354</td>
<td>0.928180</td>
</tr>
<tr>
<td>P04</td>
<td>0.145829</td>
<td>0.586666</td>
<td>0.257543</td>
<td>0.790110</td>
</tr>
<tr>
<td>P05</td>
<td>-0.152732</td>
<td>0.138570</td>
<td>-1.102199</td>
<td>0.275650</td>
</tr>
<tr>
<td>P06</td>
<td>0.224813</td>
<td>0.111678</td>
<td>2.012652</td>
<td>0.048360</td>
</tr>
<tr>
<td>P07</td>
<td>0.361219</td>
<td>0.664310</td>
<td>1.446943</td>
<td>0.153915</td>
</tr>
<tr>
<td>P08</td>
<td>0.288502</td>
<td>0.168700</td>
<td>1.710150</td>
<td>0.093199</td>
</tr>
<tr>
<td>P09</td>
<td>-0.263968</td>
<td>0.135960</td>
<td>-1.941514</td>
<td>0.057624</td>
</tr>
<tr>
<td>P10</td>
<td>-0.160551</td>
<td>0.124070</td>
<td>-1.285538</td>
<td>0.204389</td>
</tr>
<tr>
<td>P11</td>
<td>0.050031</td>
<td>0.031720</td>
<td>1.577281</td>
<td>0.120796</td>
</tr>
<tr>
<td>P12</td>
<td>0.033880</td>
<td>0.025564</td>
<td>1.325330</td>
<td>0.190256</td>
</tr>
<tr>
<td>P13</td>
<td>0.114605</td>
<td>0.534570</td>
<td>0.214387</td>
<td>0.831046</td>
</tr>
<tr>
<td>P14</td>
<td>0.019066</td>
<td>0.136782</td>
<td>0.141182</td>
<td>0.886272</td>
</tr>
<tr>
<td>P15</td>
<td>0.354912</td>
<td>0.169407</td>
<td>2.025376</td>
<td>0.043262</td>
</tr>
<tr>
<td>P16</td>
<td>0.343994</td>
<td>0.665096</td>
<td>1.046900</td>
<td>0.165897</td>
</tr>
<tr>
<td>P17</td>
<td>-0.010205</td>
<td>0.168999</td>
<td>-0.060420</td>
<td>0.952033</td>
</tr>
<tr>
<td>P18</td>
<td>-0.022530</td>
<td>0.136120</td>
<td>-0.183312</td>
<td>0.860664</td>
</tr>
<tr>
<td>P19</td>
<td>-0.016722</td>
<td>0.017226</td>
<td>-0.976720</td>
<td>0.333681</td>
</tr>
<tr>
<td>P20</td>
<td>0.017767</td>
<td>0.004375</td>
<td>3.604572</td>
<td>0.000701</td>
</tr>
<tr>
<td>P21</td>
<td>0.012926</td>
<td>0.005262</td>
<td>3.666287</td>
<td>0.000579</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

SIGMA:

0.0021292628 7.1616667e-005 0.00012324036
7.1616667e-005 0.00013733773 6.1458668e-005
0.00012324036 6.1458668e-005 8.9203614e-005

--------------------------------------------------------

STABILITY analysis of the VAR(2) process

<table>
<thead>
<tr>
<th>Roots of the reverse characteristic polynomial</th>
<th>Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.75294</td>
<td>1.75294</td>
</tr>
<tr>
<td>-0.31661</td>
<td>2.00842i</td>
</tr>
<tr>
<td>-0.31661</td>
<td>2.00842i</td>
</tr>
<tr>
<td>-2.69403</td>
<td>2.69403</td>
</tr>
<tr>
<td>-1.28611</td>
<td>1.28623i</td>
</tr>
<tr>
<td>-1.28611</td>
<td>1.28623i</td>
</tr>
</tbody>
</table>
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, optimize;
output file = tsm1e.out reset;
clear;

<<--- data --->>
load y[2,3] = lutkeph.asc;
y = ln(y[1:76,1]);
INVEST = y[..,1];
INCOME = y[..,2];
CONSUMP = y[..,3];
X = ones(76,1)' lag(1, invest);
Y = INCOME CONSUMP;

/*/ Estimation of the reduced form (10.2.4)
 ** See LUTKEPOHL, page 326
 */
{theta, stderr, Mcov, LGLH1} = varx_LS(Y, X, 1);
BB = reshape(theta, 4, 2)';
BB = ('IN(t)'' | 'CONS(t)''')BB;
print; print;
print ''; IN(t-1) CONS(t-1) Constant Inv(t-1)'';
let fmt[5,3] = '***.*' 8 8
'***.***' 12 4
'***.**' 12 4
'***.' 12 4
'***.###' 12 4;
call printfm(BB, 0'1'1'1', fmt);
print;

/*/ Estimation of the reduced form (10.2.4)
 ** See LUTKEPOHL, page 326
 */
```c
/*
** We impose the following restrictions:
**
** The INC(t-1) coefficient in the INC(t) equation is 1
** The CHNS(t-1) coefficient in the INC(t) equation is 0
** The constant in the INC(t) equation is 0
**
*/

RR = design(0:1|1|2|0|3|4|0); r = 1|zeros(7,1);
(theta,stderr,Ncov,LOGL1) = varxCLS(Y,X,1,RR,r);
BB = reshape(theta,4,2)';
BB = (1|INC(t-1) |'CHNS(t-1)').BB;
print; print;
print ' INC(t-1) CHNS(t-1) Constant Inv(t-1)';
let fmt[5,3] = '1'.e, .e' 8 8
'*.+if' 12 4
'*.+if' 12 4
'*.+if' 12 4
'*.+if' 12 4

call printfm(BB,0',1',1',fmt);
print;

Total observations: 76
Usable observations: 74
Number of parameters to be estimated: 8
Degrees of freedom: 66
Value of the maximized log-likelihood function: 472.49186

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.948656</td>
<td>0.086749</td>
<td>8.819858</td>
<td>0.00000</td>
</tr>
<tr>
<td>P02</td>
<td>0.293001</td>
<td>0.079091</td>
<td>3.75001</td>
<td>0.000334</td>
</tr>
<tr>
<td>P03</td>
<td>0.021842</td>
<td>0.082227</td>
<td>0.222363</td>
<td>0.82478</td>
</tr>
<tr>
<td>P04</td>
<td>0.690517</td>
<td>0.078673</td>
<td>8.777017</td>
<td>0.00000</td>
</tr>
<tr>
<td>P05</td>
<td>0.026000</td>
<td>0.026661</td>
<td>1.005209</td>
<td>0.31848</td>
</tr>
<tr>
<td>P06</td>
<td>0.068935</td>
<td>0.021354</td>
<td>3.212302</td>
<td>0.002037</td>
</tr>
<tr>
<td>P07</td>
<td>0.033432</td>
<td>0.017625</td>
<td>1.819475</td>
<td>0.073375</td>
</tr>
<tr>
<td>P08</td>
<td>-0.033768</td>
<td>0.014276</td>
<td>-0.263912</td>
<td>0.792670</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

<table>
<thead>
<tr>
<th>INC(t-1)</th>
<th>CHNS(t-1)</th>
<th>Constant</th>
<th>Inv(t-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC(t)</td>
<td>0.9600</td>
<td>0.0218</td>
<td>0.0268</td>
</tr>
<tr>
<td>CHNS(t)</td>
<td>0.2994</td>
<td>0.6905</td>
<td>0.0866</td>
</tr>
</tbody>
</table>

Total observations: 76
Usable observations: 74
Number of parameters to be estimated: 4
Degrees of freedom: 70
Value of the maximized log-likelihood function: 468.72789

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>1.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P02</td>
<td>0.287130</td>
<td>0.023200</td>
<td>4.543167</td>
<td>0.000023</td>
</tr>
<tr>
<td>P03</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P04</td>
<td>0.050439</td>
<td>0.017786</td>
<td>2.861154</td>
<td>0.000561</td>
</tr>
<tr>
<td>P05</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P06</td>
<td>0.005460</td>
<td>0.000045</td>
<td>14.111550</td>
<td>0.000000</td>
</tr>
<tr>
<td>P07</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

<table>
<thead>
<tr>
<th>INC(t-1)</th>
<th>CHNS(t-1)</th>
<th>Constant</th>
<th>Inv(t-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC(t)</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>CHNS(t)</td>
<td>0.2671</td>
<td>0.7024</td>
<td>0.0003</td>
</tr>
</tbody>
</table>
```

CHAPTER 3. FINANCIAL ECONOMETRICS
3.2. TSM

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, optum, pgraph;

@<<< data --->>

load rainfall[] = rainfall.asc;
y = rainfall[1:131];
y = y - 142.2;
_cycle_prm = 6;
_tsm_parm = "'rho'|'lambda_c'|'sig_epsilon'|'sig_kappa';
_tsm_optim = 1;
_print = 0;

@<<< 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 = 0.76; sig_epsilon = 1; sig_kappa = 1;
sv = rho|lambda_c|sig_epsilon|sig_kappa;
{theta, stderr, Hcov, Logl} = TD_ml(y, &sgf, sv);

@<<< Periodogram and spectral function estimation --->>

{lambda, I} = PDXM(y);
  _smoothing = 6[0][0][0][0.23];   /*Perzen lag window with bandwidth = 6 */
I = smoothing(I);             /*smoothed periodogram*/
g = sgf(theta, lambda);      /*estimated spectral generating function*/
q = trunc(rows(lambda)/2);

/*
** PAVITAN 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 authors 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(x,1); Nobs = rows(x1); y1 = seqa(1,1,Nobs)/Nobs;
x2 = seqa(0.10,Nobs,Nobs); y2 = 1 - cdflchi(x2,2);

graphnet;
begwind;
window2(0,2,1,0.75);
3.2.3 Estimation methods

3.2.3.1 Maximum likelihood

An example with EXPAR model.

```plaintext
/*
** 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
**
*/
```
library tsm, optimum;
output file = tsm2a.out reset;
@<--- data --->@
load y[] = lynx.asc;
y = log(y);
x = y - mean(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]*x[i-1]*exp[-delta*y[i-1]] + beta[4] + beta[5]*x[i-1]*exp[-delta*y[i-1]]*x[i-1])
i = i+1;
end;
LogL[3:Nobs] = -0.5*ln(2*pi) - 0.5*ln(sigma^2)
-0.5*(resid[3:Nobs]^2)/(sigma^2);
retv(LogL);
endp;

@<--- estimation --->@
._print = 1;
._tsm_optnum = 1;
._tsm_parms = "beta1" | "beta2" | "beta3" | "beta4" | "beta5" | "delta" | "sigma";
._title = "EXPAR model";
sv = ones(6, 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std. err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta1</td>
<td>1.989202</td>
<td>2.011331</td>
<td>0.988598</td>
<td>0.324690</td>
</tr>
<tr>
<td>beta2</td>
<td>-0.669867</td>
<td>2.056277</td>
<td>-0.326924</td>
<td>0.742388</td>
</tr>
<tr>
<td>beta3</td>
<td>0.446605</td>
<td>0.182175</td>
<td>2.462498</td>
<td>0.015440</td>
</tr>
<tr>
<td>beta4</td>
<td>0.151471</td>
<td>1.404292</td>
<td>0.107853</td>
<td>0.914312</td>
</tr>
<tr>
<td>beta5</td>
<td>0.742232</td>
<td>1.414615</td>
<td>0.524725</td>
<td>0.600891</td>
</tr>
<tr>
<td>delta</td>
<td>0.533173</td>
<td>0.206333</td>
<td>2.582944</td>
<td>0.011166</td>
</tr>
<tr>
<td>sigma</td>
<td>0.113011</td>
<td>0.282295</td>
<td>0.400248</td>
<td>0.688502</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

3.2.3.2 Generalized method of moments

▶ An example with Ornstein-Uhlenbeck process.

new;
library tsm, optimum, option;
TSMset;
output file = tsm2b.out reset;
\[ \begin{align*}
\text{\#} &\text{ data generation } \rightarrow \text{#} \\
\text{\#} & x_0 = 10; \quad a = 0.8; \quad b = 0.1; \quad \text{sigma} = 0.06; \\
\text{\#} & t_0 = 0; \quad \text{TT} = 100; \quad \text{Nobs} = 1001; \\
\text{\#} & \text{sv} = \text{sb}/\sigma; \\
\text{\#} & (t, x) = \text{simulate.DU}(x_0, a, \text{sigma}, t_0, \text{TT}, \text{Nobs}, 1); \\
\text{\#} & \text{#} \text{ moments definition } \rightarrow \text{#} \\
\text{\#} & \text{h} = 0.1; \\
\text{\#} & \text{proc} \text{ Moments(theta);} \\
\text{\#} & \quad \text{local} \ a, \ b, \ \text{sigma}, \ k_1, \ k_2, \ \text{epsilon}, \ M; \\
\text{\#} & \quad a = \text{theta}[1]; \\
\text{\#} & \quad b = \text{theta}[2]; \\
\text{\#} & \quad \text{sigma} = \text{theta}[3]; \\
\text{\#} & \quad k_1 = \exp(-a*h); \\
\text{\#} & \quad k_2 = \text{sigma}^2*(1-\exp(-2*a*h))/(2*a); \\
\text{\#} & \quad \text{epsilon} = x[2:\text{Nobs}]-k_1.*x[1:\text{Nobs}]-b.*(1-k_1); \\
\text{\#} & \quad M = \text{epsilon}^2(\text{epsilon}^2-k_2)^2(\text{epsilon}.*x[1:\text{Nobs}]-1); \\
\text{\#} & \quad \text{retv(M);} \\
\text{\#} & \text{#} \text{ GMM estimation } \rightarrow \text{#} \\
\text{\#} & \_\text{pem}_\text{parm} = \{'a'='a', 'b'='b', 'sigma='sigma'}; \\
\text{\#} & \_\text{sv} = \text{sb}/\sigma; \\
\text{\#} & \_\text{ml}_\text{jacobian$_\text{proc}$ = 0;} \\
\text{\#} & \_\text{title} = \{'Gleason-\text{Uhlenbeck process'}\}; \\
\text{\#} & (\text{theta}, \text{stderr}, \text{Mcov}, \text{qmin}) = \text{gmm(Moments, sv)}; \\
\text{\#} & \text{output} \ \text{off}; \\
\text{\#} & \text{Total observations:} \quad 1000 \\
\text{\#} & \text{Usable observations:} \quad 1000 \\
\text{\#} & \text{Number of parameters to be estimated:} \quad 3 \\
\text{\#} & \text{Degrees of freedom:} \quad 997 \\
\text{\#} & \text{Number of moment conditions:} \quad 3 \\
\text{\#} & \text{Value of the criterion function:} \quad 0.000000 \\
\end{align*} \]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.799014</td>
<td>0.009004</td>
<td>88.258036</td>
<td>0.000000</td>
</tr>
<tr>
<td>b</td>
<td>0.050010</td>
<td>0.007767</td>
<td>11.559073</td>
<td>0.000000</td>
</tr>
<tr>
<td>sigma</td>
<td>0.609433</td>
<td>0.001430</td>
<td>42.619105</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

### 3.2.3.3 Simulated method of moments

- Log-normal distribution.

\begin{verbatim}
\text{\#} \text{gmm;} \\
\text{\#} \text{library \ text, \optnum, \pgraph;} \\
\text{\#} \text{TSMset;} \\
\text{\#} \text{#include \ \text{GLN.src};} \\
\text{\#} \text{output file = \text{tsm2e.out \ \reset;}} \\
\text{\#} \text{load data[1] = \text{frfden.src;}} \\
\text{\#} \text{s = packr(log(data));} \\
\text{\#} \text{\_kernel[1:2] = 0.5210.534; \}} \\
\text{\#} \text{/* Density estimation by Kernel */} \\
\text{\#} \text{\{x, pdfKernel, cdfKernel, retcode\} = Kernel(s);} \\
\text{\#} \text{/* Log-Normal */} \\
\text{\#} \text{proc H(theta);} \\
\text{\#} \text{local mu, sigma, h1, h2;}
\end{verbatim}
mu = theta[1];
sigma = sqrt(theta[2]^2);

/* first moment E[x] = exp(mu+0.5*sigma^2) */
h1 = n - exp(mu+0.5*sigma^2);

/* second moment var[x] = exp(2*mu+sigma^2)*{exp(sigma^2)-1} */
h2 = h1.*h1 - exp(2*mu+sigma^2)*{exp(sigma^2)-1};

retv(h1*h2);
endp;
_gmm_logn = 0;
_gmm_iter = 2;
_output = 1;
_print = 0;
_gmm_parms = '"mu'"',"sigma'";

sv = ln(mean(x)).'.1;
(theta,stderr,Mcov,Qmin) = gmm(MH,sv);
mu = theta[1];
sigma = sqrt(theta[2]^2);
pdfGMM = 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 = mean(u);
uc = u - M1;
M2 = mean(uc^2);

/* first moment */
h1 = n - M1;

/* second moment */
h2 = h1.*h1 - M2;

retv(h1*h2);
endp;

(theta,stderr,Mcov,Qmin) = gmm(&SimulatedMoments,sv);
mu = theta[1];
sigma = sqrt(theta[2]^2);
pdfGMM2 = pdfLN(x, mu, sigma);

graphset;
_pltype = 13;
_plegstr = 'Gaussian Kernel\001log-normal density (GMM)\001log-normal density (GMM)';
_plgcl = (2 1 3);
xtics(0.522, 0.634, 0.602, 0);
graphprv('c1 = -cf=tm2c.dat');
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

\[
\begin{align*}
    z_t &= x_t + y_t \\
    x_t &= \phi_1 x_{t-1} + u_t \\
    y_t &= v_t - \theta_1 v_{t-1}
\end{align*}
\]

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

\[
g(\lambda) = \sigma_u^2 \left| 1 - \phi_1 e^{i\lambda} \right|^2 + \sigma_v^2 \left| 1 - \theta_1 e^{i\lambda} \right|^2 \\
= \frac{\sigma_u^2}{1 - 2\phi_1 \cos \lambda + \phi_1^2} + \frac{\sigma_v^2}{1 - 2\theta_1 \cos \lambda + \theta_1^2}
\]

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

```r
#--- data generation ---#
rndseed 123456;
phi1 = 0.5; theta1 = 0.7; sigma_u = 0.25; sigma_v = 1;
Nobs = 1000;
u = rnorm(Nobs, 1) * sigma_u;
x = recserar(u, 0, phi1);
v = rnorm(Nobs, 1) * sigma_v;
y = v + theta1 * (0 | trimr(v, 0, 1));
x = x + y;

#--- Whittle estimation ---#
sv = phi1 | sigma_u | theta1 | sigma_v;
_umat = \text{umat}(phi1, sigma_u, theta1, sigma_v);
_fourier = 0;
_egg_JacobianProc = &egg_Jacobian;
{theta, stderr, Mcov, Logl} = FD_m3(x, _egg, sv);
output off;
```
3.2. TSM

```
proc mgf (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 = cons(lambda);
  g = (sigma_u^2)/(1-2*phi1*w*phi1^2) -
   (sigma_v^2)/(1-2*theta1*w*theta1^2); return(g);
endp;

proc mgf_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 = cons(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;
  return(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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>phi1</td>
<td>0.380033</td>
<td>0.171811</td>
<td>2.211752</td>
<td>0.027210</td>
</tr>
<tr>
<td>sigma_u</td>
<td>0.346656</td>
<td>0.084475</td>
<td>4.127668</td>
<td>0.000040</td>
</tr>
<tr>
<td>theta1</td>
<td>0.599331</td>
<td>0.144324</td>
<td>4.247191</td>
<td>0.000040</td>
</tr>
<tr>
<td>sigma_v</td>
<td>0.856431</td>
<td>0.349566</td>
<td>2.449773</td>
<td>0.014466</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

3.2.3.5 Generalized flexible least squares

new;
library ts, otsum, pgarch;

/------------- First example: Time-varying parameters -------------/
@<--- data generation ---@

randseed 456;
Nobs = 1000;
s = seq(1,1,Nobs);
x = floor(rndu(Nobs,3)*100);

b1 = ones(Nobs,1); /* constant parameter */
b2 = sin(seq(0,2*pi/Nobs,Nobs)) + rnd(Nobs,1)*0.26; /* sine parameter */
b3 = regress(rndu(Nobs,1),10,0.9); /* AR parameter */
B = b1' b2' b3;
y = sum(x'.^1) + rnd(Nobs,1);

/* Flexible Least Squares with mu = 1 */
**(and imposing constancy for the first parameter)**
*/
mu = 100000*[1];
{Bf1s,u,r2,M,r2,D} = FLS(y,x,mu);  /* FLS estimation */
Bols = (invpd(x'x)*x'y) .* ones(Nobs,1);  /* OLS estimation */

xy1 = b1' • Bf1s[.,1]' • Bols[.,1];
xy2 = b2' • Bf1s[.,2]' • Bols[.,2];
xy3 = b3' • Bf1s[.,3]' • Bols[.,3];

/
----------------- Second example: step function -----------------
*/
StepValues = 2[2.5;2.5;1.5;2;2.5;3;2.75];
x1 = floor(rndu(Nobs,1)*100); x2 = floor(rndu(Nobs,2)*100);
b1 = []; b2 = 23;
i = 1;
do until i > 8;
b1 = b1 | StepValues[i] • ones(125,1);
i = i + 1;
enddo;

y = 4 + x1.*b1 + x2.*b2 + rndu(Nobs,1);
x = ones(Nobs,1)' • x1' • x2;

{Bf1s,u,r2,M} = FLS(y,x,100000);  /* FLS estimation */
Bols = (invpd(x'x)*x'y) .* ones(Nobs,1);  /* OLS estimation */

xy4 = b1' • Bf1s[.,2]' • Bols[.,2];

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

  _pdate = ''; _pnum = 2; _pnumht = 0.20; _paxht = 0.20;
  _ptype = 3; _plegstr = 'true\0FLS\0OLS'; _plegctl = (2 8 3 2);
  xlabel('t');
  setwind(1);
title('Time-varying parameters\First coefficient');
  xy(a,xy1);

  _plegctl = 0;

  setwind(2);
title('Time-varying parameters\Second coefficient');
  xy(a,xy2);

  setwind(3);
title('Time-varying parameters\Third coefficient');
  xy(a,xy3);

  setwind(4);
title('Step functions');
  xy(a,xy4);
  graphprt('-c=1 -cf=tmm2e. eps');
endwind;
3.2.4 State-space modelling

3.2.4.1 Structural models

- Local linear trend model. The estimation is performed in the frequency domain.

```c
/*
** HARVEY [1990], Forecasting, Structural Time Series and
** the Kalman Filter, Cambridge University Press, pages 90-93
**
** Kalman filtering and the Local Linear Trend Model
*/

library tsm, optmum.pgraph;
TSMset;
output file = tsm3a.out reset;

@<--- data --->@
load gnp1 = gnp.asc;
Nobs = row(gnp);

@<--- frequency domain estimation --->@
_tsm_optum = 0; /* Method of scoring */
{beta, stderr, Mcov, Logl} = smLLT(gnp, I, I, 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]^2)I * (theta[2]);
call SM_build(Z, d, H, T, c, R, Q, 0);

a0 = gnp[1]0; P0 = zeros(2,2); /* Kalman filter initialization */
call KFfiltering(gnp, a0, P0);

z = KF_matrix(3); /* z[t] */
s = seqa(1909,1,61);
output off;

pgraph;
font('simplex simgrem');
_plate = '***'; _pnun = 2;
breakwind;
makewind(9.6, 0.0, 0, 0);
makewind(4, 3.1, 2.5, 1);
setwind(1);
```
Local linear trend model. The estimation is performed in the time domain.

```plaintext
/*
** HARVEY [1990], Forecasting, Structural Time Series and
** the Kalman Filter, Cambridge University Press, pages 90-93
** Kalman filtering and the Local Linear Trend Model
*/

//
library tspm, optsum, graph;
TSSet;
output file = tm3b.out reset;

\@--- data -->@
load gnp[] = gnp.asc;
Nobs = row(gnp);

\@--- time domain estimation -->@
3.2. TSM

\[ Z = 1^0; \quad d = 0; \]
\[ T = (1, 1) \times 1; \quad c = 0; \quad R = (1, 0, 0); \]
\[ a_0 = \text{gp}[1]; \quad P_0 = \text{zeros}(2, 2); \]
\[ \text{proc ml(theta);} \]
\[ \quad \text{local N,Q,LogL; theta = theta}\_2; /* \text{sigma}^2 */ \]
\[ \quad \text{H = theta[3]; Q = (theta[1]);} \]
\[ \quad \text{call} \text{Kfiltering}(\text{Z,d,H,T,c,R,Q,0); \}
\[ \quad \text{call} \text{LSM}(\text{Z,d,H,T,c,R,Q,0); \}
\[ \quad \text{LogL = KF_matrix(3); s = KF_matrix(3); \}
\[ \quad \text{t_} = \text{seq}(\text{t_}, 1, \text{st}(t_)); \]
\[ \quad \text{output off; \}
\[ \quad \text{graphset; \}
\[ \quad \text{font(1); \}
\[ \quad \text{title('"LOCAL LINEAR TREND MODEL"'); \}
\[ \quad \text{pframe = 0; \}
\[ \quad \text{pnum = 2; \}
\[ \quad \text{ptiltht = 0.26; \}
\[ \quad \text{plnumt = 0.20; \}
\[ \quad \text{title('"\text{Estimated Results}"'); \}
\[ \quad \text{pframe = 1; \}
\[ \quad \text{pnum = 2; \}
\[ \quad \text{title('"\text{Estimated Results}"'); \}
\[ \quad \text{graphprt('"\text{Estimated Results}"'); \}
\[ \quad \text{end; \}
\]
\text{Total observations: \quad 61; \}
\text{Usable observations: \quad 61; \}
\text{Number of parameters to be estimated: \quad 3; \}
\text{Degrees of freedom: \quad 66; \}
\text{Value of the maximized log-likelihood function: \quad -261.05421; \}
\text{Parameters estimates std.err. t-statistic p-value \}
\begin{array}{cccc}
\text{P01} & 20.930307 & 3.630210 & 5.645237 & 0.000001 \\
\text{P02} & 1.858483 & 0.589496 & 3.136310 & 0.000001 \\
\text{P03} & 7.548049 & 10.783561 & 0.699668 & 0.486338 \\
\end{array}
\text{Covariance matrix: White Heteroskedastic matrix.}
3.2.4.2 Time-varying parameters

```c
/**
 ** 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 tm, opthum, pgplot;
TSset;

declare external sigma;

@<<--- Time-Varying Model simulation --->>
rndseed 123;
s = seq(1,1,100);
b0 = reccosa(rndm(100,1)*1.5,10,1);
b1 = reccosa(rndm(100,1)*0.5,4,1);
x0 = ones(100,1);
x1 = rndu(100,1)*25;
u = rndm(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 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 s0,P0,LogL;
  sigma = theta[1:3];
```
3.2. TSM

call SSM_build(&Z, &d, &H, &T, &c, &R, &Q, 1);
s0 = b0[i][1][1];
P0 = zeros(2, 2);
call KFfiltering(Y, s0, P0);
LogL = KF_ml;
retv(LogL);
endp;

@<--- ML estimation --->@

_tsm_optnum = 0;  /* BHSH algorithm */
_tsm_parms = "'sig_v'"  "'sigma0'"  "'sigma1'";
{theta, stderr, Mcov, LogL} = TD_ml(&ml, 2|1.6|0.2);

beta = invpd(X'X)*X'Y;  /* GLS estimation */
output off;

@<--- Kalman filtering --->@

sigma = theta[1:3];
s0 = b0[1][1];
P0 = zeros(2, 2);
call SSM_build(&Z, &d, &H, &T, &c, &R, &Q, 1);
call KFfiltering(Y, s0, P0);

yc = KF_matrix(1);  /* y[t-1] */
v = KF_matrix(2);  /* v[t] */
s = KF_matrix(3);  /* a[t] */
P = KF_matrix(4);  /* P[t] */
s = KF_matrix(4);  /* a[t] */
Pc = KF_matrix(6);  /* F[t-1] */
P = KF_matrix(7);  /* F[t-1] */
invF = KF_matrix(8);  /* P[t-1] */

@<--- Kalman smoothing --->@

{as, Ps} = Ksmoothing;

graphset;
beginw;
window(1, 2, 1);

_plotc = "'"; _plotn = 2; _paxht = 0.25; _pnumht = 0.25;
fonts("simplex simplexgamma");
xlabel("'t'");
_pType = 6[1][3][4];
_plotc1 = [2 5 4 1];
setwind(1);

_plotc = "'202b'201l'0[ (true values) '"
   "'000Kalman filter'
   "'000Kalman smoothing'
   "'000GLS'
   xy(s, b0[1][1]-as[1][1]+(beta[1]*ones(100,1)));

setwind(2);

_plotc = "'202b'201l'0[ (true values) '"
   "'000Kalman filter'
   "'000Kalman smoothing'
   "'000GLS'
   xy(s, b1[1][2]-as[2][2]+(beta[2]*ones(100,1)));

graphpr(1-c=1 -cf=tsm3c.epa');
endw;
3.2.4.3 Multivariate model

- Not yet done.

3.2.4.4 Exact maximum likelihood

- Univariate ARMA example

```latex
/*
** Exact Maximum Likelihood Estimation of an ARMA(1,1) model
** with the Kalman filter
*/
new;
library ts, optma, pgraph;
TSMset;
output file = tsm.be.out reset;

/* Generate an ARMA(1,1) process */
randseed 133456;

Nobs = 100;
et = rnorm(100+1,1.5);
yt = recvar(et[2:Nobs]+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, 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_optnum = 0;
_TSM_gtol = 0.001;
_TSM_Mcov = 1;
{theta, stderr, Mcov, Logl} = TD_ml({a0, P0, Z}, 1.5);

beta = theta[1:2];
sigma = theta[3]^2;
```
3.2. TSM

/* SSMM */
{z,d,H,T,c,R,Q} = arma_to_ssm(betas,1,1,SIGMA);
call SSMM_build(z,d,H,T,c,R,Q,0);
{a0,P0} = SSMM_ic;
call KFfiltering(yt,a0,P0);

/* Forecasting */

mp = 10;
{Forecasts,mse,aF,PF} = KForecasting(mp);
eps = seq(Nobs+1,1,mp);
Fsderr = sqrt(mse);
LCL = Forecasts - 1.96*Fsderr;
UCL = Forecasts + 1.96*Fsderr;

call printf("LCL" Forecasts" UCL\"
"Forecast Std. Err.\"
print chr$(45+mse(75,1));
let fmt[5,3] = '*.*f' 6 0
   '.*f' 6 0
   '.*f' 15 6
   '.*f' 15 6
   '.*f' 15 6;
call printf5(m$ omc,1'1'1',fmt);

output off;

/*
** If you have the ARIMA library, you could compare with Ansley method:
**
** {b,l,e,cv,sic,abc} = 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: -183.229266 Number of Residuals: 100
** AIC : 370.446532 Error Variance : 2.294675836
** SBC : 375.666772 Standard Error : 1.514818786
**
** DF: 98 Adj. SSE: 226.550784245 SSE: 224.878416685
**
** Coefficients Std. Errors T-Ratio Approx. Prob.
** AR1 0.71619142 0.07757377 9.15405 0.00000
** MA1 -0.00727789 0.03713339 -5.22253 0.00000
**
** Total Computation Time: 0.11 (seconds)
**
** AR Root: 1.46026
**
** MA Root: -1.97131
**
** Forecasts for ARIMA(1,0,1) Model. 95% Confidence Interval Computed.
**
** Period LCL Forecasts UCL Forecast Std. Err.
** 101 -2.076066 0.899923 3.867613 1.514819
** 102 -4.041248 0.639084 5.320616 2.388275
** 103 -4.887352 0.455266 5.797765 2.726645
** 104 -5.324196 0.323830 6.072068 2.881746
** 105 -5.566196 0.230512 6.027220 2.967659
** 106 -5.709454 0.164035 6.034545 2.996218
** 107 -5.752815 0.116729 6.024274 3.014169
** 108 -5.843140 0.083086 6.009272 3.023630
** 109 -5.876523 0.059111 5.994744 3.028440
** 110 -5.898338 0.042064 5.982165 3.030873
*/

Total observations: 100
Unable observations: 100
Number of parameters to be estimated: 3
Degrees of freedom: 87
Value of the maximized log-likelihood function: -183.22327

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std. err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.71615</td>
<td>0.07638</td>
<td>9.35945</td>
<td>0.00000</td>
</tr>
<tr>
<td>P02</td>
<td>-0.597273</td>
<td>0.093996</td>
<td>-6.396750</td>
<td>0.00000</td>
</tr>
<tr>
<td>P03</td>
<td>1.499696</td>
<td>0.108069</td>
<td>14.139286</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

<table>
<thead>
<tr>
<th>Period</th>
<th>LCL</th>
<th>Forecast</th>
<th>UCL</th>
<th>Forecast Std. Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>-5.040286</td>
<td>0.899297</td>
<td>3.639139</td>
<td>1.499698</td>
</tr>
<tr>
<td>102</td>
<td>-3.984991</td>
<td>0.639890</td>
<td>5.273670</td>
<td>2.364276</td>
</tr>
<tr>
<td>103</td>
<td>-4.835764</td>
<td>0.455213</td>
<td>5.744169</td>
<td>2.668467</td>
</tr>
<tr>
<td>104</td>
<td>-5.297358</td>
<td>0.353286</td>
<td>5.915411</td>
<td>2.857603</td>
</tr>
<tr>
<td>105</td>
<td>-5.508662</td>
<td>0.230518</td>
<td>5.986097</td>
<td>2.976647</td>
</tr>
<tr>
<td>106</td>
<td>-5.647613</td>
<td>0.164040</td>
<td>5.976683</td>
<td>2.865129</td>
</tr>
<tr>
<td>107</td>
<td>-5.731675</td>
<td>0.116733</td>
<td>5.865842</td>
<td>2.853631</td>
</tr>
<tr>
<td>108</td>
<td>-5.976375</td>
<td>0.083082</td>
<td>5.945853</td>
<td>2.833527</td>
</tr>
<tr>
<td>109</td>
<td>-5.847004</td>
<td>0.059113</td>
<td>5.923230</td>
<td>2.998019</td>
</tr>
<tr>
<td>110</td>
<td>-5.838772</td>
<td>0.042066</td>
<td>5.922904</td>
<td>3.000428</td>
</tr>
</tbody>
</table>

---

Multivariate ARMA example.

/*
  ** Estimation of a Vector ARMA(1,1) process
  ** Exact Maximum Likelihood
  ** */

new;
library tm_optum;
output file = tm5f.out reset;

@--- Data ---->
load reinsel[100,2] = reinsel.asc;
invest = reinsel[.,1];
i inv = rein sel[.,2];
di = invest - log(invest);
y = di*invest;

@--- ML function ---->
proc ml(coef);
local beta,sigma,Pchol,Z,d,H,T,c,R,Q;
local a0,P0,Logl;

beta = coef[1:6];
SIGMA = Pchol*Pchol';

{Z,d,H,T,c,R,Q} = arma_to_SM(betas,1,1,SIGMA);
call SM_build(Z,d,H,T,c,R,Q,0);
{a0,P0} = SM_ic; /* <= THIS LINE IS THE MOST IMPORTANT
               FOR EXACT MLE
               */

if imiss(a0);
  cli;
  print 'Not implemented for non-stationary ARMA models.';
  end;
endif;

call KFiltering(y,a0,P0);
Logl = -ml; 
retP(Logl);
endp;

_print = 1;
_tm_optum = 0;
_tm_pchol = 0.001;
load arma1, arma2;
3.2. TSM

sv = armaIvec(chol(arma2));  /* Starting values = CMLE parameters */

{theta, stderr, Ncov, 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>estimates</th>
<th>std.err.</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>0.356733</td>
<td>0.176666</td>
<td>2.030576</td>
<td>0.045317</td>
</tr>
<tr>
<td>P02</td>
<td>0.946566</td>
<td>0.230546</td>
<td>2.612441</td>
<td>0.006062</td>
</tr>
<tr>
<td>P03</td>
<td>-0.0253065</td>
<td>0.044817</td>
<td>-0.553508</td>
<td>0.578893</td>
</tr>
<tr>
<td>P04</td>
<td>0.841344</td>
<td>0.047814</td>
<td>17.670001</td>
<td>0.000000</td>
</tr>
<tr>
<td>P05</td>
<td>-0.208480</td>
<td>0.183373</td>
<td>-1.136917</td>
<td>0.256669</td>
</tr>
<tr>
<td>P06</td>
<td>0.237817</td>
<td>0.317614</td>
<td>0.748760</td>
<td>0.455999</td>
</tr>
<tr>
<td>P07</td>
<td>-0.181840</td>
<td>0.065227</td>
<td>-2.626708</td>
<td>0.010169</td>
</tr>
<tr>
<td>P08</td>
<td>2.377602</td>
<td>0.165904</td>
<td>14.065018</td>
<td>0.000000</td>
</tr>
<tr>
<td>P10</td>
<td>1.308656</td>
<td>0.418356</td>
<td>2.931694</td>
<td>0.018900</td>
</tr>
<tr>
<td>Pr</td>
<td>4.0777719</td>
<td>0.290760</td>
<td>14.027226</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

Covariance matrix: inverse of the negative Hessian.

3.2.4.5 Impulse functions

> An example with a SSM model.

ew;
library ts, optsum, pgraph;
TSMset;
Z = {11 4 2 2 -3}; d = 0100; H = {1 1 1 1 0 0 0 0 8};
T = {6 5 0.45, -0.5 0.6};
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 = seq(01, Nr+1);
e = 1:1;
{Delta, K, psi, beta} = SSM_orthogonal(e, Nr);
pgraphet;
begwind;
makewind(9, 0.5, 0.6, 0.565-0.5, 0);
makewind(9/2, 5.855/2, 0, 6.665/2, 0);
makewind(9/2, 5.855/2, 4.5, 8.855/2, 0);
makewind(9/2, 5.855/2, 0, 5.855/2, 0);
makewind(9/2, 5.855/2, 2, 0, 0);
makewind(9/2, 5.855/2, 4.5, 0, 0);
setwind(1);
_ paxes = 0; _ pnum = 0; _ ptitltht = 2;
title('Responses to the orthogonal impulse e = 01 -t 0)'');
draw;
setwind(2);
pgraphet;
_ ptitltht = 0.35; _ pnum = 0.25; _ pnum = 0.25; _ pnum = 2;
_ line = 't 0 0 0; _ ttitle = 't' 0;
title('of the first variable y1[t']');
xtics(0, Nr, 1, 0);
xy(a, Delta[, 1]);
setwind(3);
title('of the second variable y12[t']');
xtics(0, Nr, 1, 0);
xy(a, Delta[, 2]);
setwind(4);
pgraphet;
_ paxes = 0; _ pnum = 0; _ ptitltht = 2;
title('Cumulative responses'));
draw;
An example with a VAR model.

```plaintext
/*
** LUTEKPEHL [1994], Introduction to Multiple Time Series Analysis,
** Springer-Verlag, Berlin-Heidelberg
**
** Forecast Error Variance Decomposition
** See LUTEKPEHL, section 3.7.3
**
*/

now;
library tsm, optsum, pgraph;
output file = tsm3h. out reset;
load y[92,3] = lutkeph.asc;
y = ln(y1[1:76, 1]);
INVEST = y[.,1];
dinv = INVEST-lagt(INVEST);
INCOME = y[.,2];
dinc = INCOME-lagt(INCOME);
CONSUM = y[.,3];
dcons = CONSUM-lagt(CONSUM);
data = dinv’dinc’dcons;

_pnum = 0;
{theta, stderr, Mcov, LGEL} = varx LS(data, 1,2);  /* Constant and 2 lags */
theta = theta[1:16];  /* Estimates of the AR part */
SIGMA = _varx_SIGMA;
{Z,d,H,T,c,R,q} = arma_to_SSM(theta, 2, 0, SIGMA);
call SSN_build(Z,d,H,T,c,R,q,0);
Omega = SSN_fewd(8);
i = nseq(1,1,8);```
3.2. TSM

mask = 1"1"i"1; let fmt[4,3]= ’*.*lf’ 13 0 ’*.*lf’ 16 4 ’*.*lf’ 18 4 ’*.*lf’ 18 4;

print;
print ’FIRECAST ERROR VARIANCE DECISION in INVESTMENT (percent)’;
print;
print chs(4)mes(79,1));
print ’ periods investment income consumption’;
print chs(4)mes(79,1));
i = 1;
do until i > 0;
   r = xpm2(tomesa, i);
   r = r[1, 1];
call printf(1’*’(r*100),mask,fmt); print;
i = i+1;
end;

print;
print;
print ’FIRECAST ERROR VARIANCE DECISION in INCOME (percent)’;
print;
print chs(4)mes(79,1));
print ’ periods investment income consumption’;
print chs(4)mes(79,1));
i = 1;
do until i > 0;
   r = xpm2(tomesa, i);
   r = r[2, 1];
call printf(1’*’(r*100),mask,fmt); print;
i = i+1;
end;

print;
print;
print ’FIRECAST ERROR VARIANCE DECISION in CONSUMPTION (percent)’;
print;
print chs(4)mes(79,1));
print ’ periods investment income consumption’;
print chs(4)mes(79,1));
i = 1;
do until i > 0;
   r = xpm2(tomesa, i);
   r = r[3, 1];
call printf(1’*’(r*100),mask,fmt); print;
i = i+1;
end;

output off;

--------------------------------------------------------------------
FORECAST ERROR VARIANCE DECISION in INVESTMENT (percent)
--------------------------------------------------------------------
<table>
<thead>
<tr>
<th>periods</th>
<th>Investment innovations</th>
<th>Income innovations</th>
<th>Consumption innovations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>95.9960</td>
<td>1.7611</td>
<td>2.2929</td>
</tr>
<tr>
<td>3</td>
<td>94.5946</td>
<td>2.0821</td>
<td>2.6330</td>
</tr>
<tr>
<td>4</td>
<td>92.8722</td>
<td>2.3561</td>
<td>2.8847</td>
</tr>
<tr>
<td>5</td>
<td>93.8464</td>
<td>3.0181</td>
<td>3.1356</td>
</tr>
<tr>
<td>6</td>
<td>93.8308</td>
<td>3.0946</td>
<td>3.1446</td>
</tr>
<tr>
<td>7</td>
<td>93.7764</td>
<td>3.0737</td>
<td>3.1479</td>
</tr>
<tr>
<td>8</td>
<td>93.7761</td>
<td>3.0739</td>
<td>3.1510</td>
</tr>
</tbody>
</table>

--------------------------------------------------------------------
FORECAST ERROR VARIANCE DECISION in INCOME (percent)
--------------------------------------------------------------------
<table>
<thead>
<tr>
<th>periods</th>
<th>Investment innovations</th>
<th>Income innovations</th>
<th>Consumption innovations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.7636</td>
<td>98.2464</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>6.0845</td>
<td>90.7470</td>
<td>3.2285</td>
</tr>
<tr>
<td>3</td>
<td>6.9592</td>
<td>89.5762</td>
<td>3.4845</td>
</tr>
<tr>
<td>4</td>
<td>6.8313</td>
<td>89.2321</td>
<td>3.9366</td>
</tr>
<tr>
<td>5</td>
<td>6.8651</td>
<td>89.2120</td>
<td>3.9379</td>
</tr>
<tr>
<td>6</td>
<td>6.9243</td>
<td>89.1486</td>
<td>3.9348</td>
</tr>
</tbody>
</table>
CHAPTER 3. FINANCIAL ECONOMETRICS

7 6.9222 69.1158 3.9620
8 6.9228 69.1149 3.9623

FORECAST ERROR VARIANCE DECOMPOSITION in CONSUMPTION (percent)

<table>
<thead>
<tr>
<th>periods</th>
<th>Investment innovations</th>
<th>Income innovations</th>
<th>Consumption innovations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.9650</td>
<td>27.2921</td>
<td>64.7129</td>
</tr>
<tr>
<td>2</td>
<td>7.7348</td>
<td>27.3946</td>
<td>64.8904</td>
</tr>
<tr>
<td>3</td>
<td>12.9728</td>
<td>33.3941</td>
<td>53.6830</td>
</tr>
<tr>
<td>4</td>
<td>12.8703</td>
<td>33.4988</td>
<td>53.6985</td>
</tr>
<tr>
<td>5</td>
<td>12.8668</td>
<td>33.9944</td>
<td>53.2168</td>
</tr>
<tr>
<td>6</td>
<td>12.8622</td>
<td>33.9630</td>
<td>53.1946</td>
</tr>
<tr>
<td>7</td>
<td>12.8702</td>
<td>33.9682</td>
<td>53.1736</td>
</tr>
<tr>
<td>8</td>
<td>12.8704</td>
<td>33.9682</td>
<td>53.1614</td>
</tr>
</tbody>
</table>

3.2.4.6 Bootstrap methods

► Computes the standard errors of the impulse responses by SSM bootstrap techniques.

```bash
/*
** LUTKEPOHL [1991], Introduction to Multiple Time Series Analysis,
** Springer-Verlag, Berlin-Heidelberg
**
** This program reproduces the figures 3.4-3.7 of LUTKEPOHL [1991].
*/

cmp;
library tsm, opusum, pgraph;
TSMset;
load y[92,3] = lutkeohl.asc;
y = ln(y[1:76,1]);
INVEST = y[..1];
dinc = INVEST-lag1(INVEST);
INCOME = y[..2];
dinc = INCOME-lag1(INCOME);
CONSUM = y[..3];
dcon = CONSUM-lag1(CONSUM);
{data,retcode} = Missing(dinc",dcon",0);
Nobs = row(data);

/* First, estimate the coefficients */

_print = 0;
{theta,stderr,Mcov,LDL} = 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 */
Nss = 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 Income */
rep3 = zeros(1+Nr,1); /* Responses of Consumption to an impulse in Consumption */
rep4 = zeros(1+Nr,1); /* Accumulated responses of Consumption to an impulse in Income */

/* Compute the responses */
call arma_impulse(beta,2,0,Nr);
```

3.2. TSM

```c
j = j + 1;
endo;

/* Build the corresponding state space model */
{Z, d, H, T, c, R, Q} = arma_to_SSM(betas, 2, 0, SIGMA);
c[1:3] = const; /* Add the constant */
call SSMBuild({Z, d, H, T, c, R, Q, 0},
a0 = data[1,1]'data[2,1]'zeros(3,1),
P0 = zeros(9,9);
call KFiltering(data[Nobs, ], a0, P0);

/* Compute the standard errors with the Bootstrap Method */
rep1s = zeros(1+Nr,Na);
rep2s = zeros(1+Nr,Na);
rep3s = zeros(1+Nr,Na);
rep4s = zeros(1+Nr,Na);
i = 1;
do until i > Na;

{ys,as} = bootstrap_SSM(a0); /* LSM bootstrap */
datas = data[1,1]'data[2,1]'ys; /* Add the pre-sample values */
{thetas,stderr_Mcov,LOGL} = varc_Ls(datas,1,2); /* VAR estimation */
betas = thetas[1:10];
call arma_impulse(betas, 2, 0, Nr);

j = 0;
do until j > Nr;

z = varget('IMPULSE''$f(yos([j,'''1f''',1,0));
rep1s[j+1,1] = z[3,3];
rep2s[j+1,1] = z[1,3];
rep3s[j+1,1] = z[3,3];
rep4s[j+1,1] = z[1,3];
j = j + 1;
end;

i = i + 1;
endo;
stderr1 = stderr(rep1s);
stderr2 = stderr(rep2s);
stderr3 = stderr(rep3s);
stderr4 = stderr(rep4s);
period = seq(0,1,Nr+1);

dgraph;
beginwind;
window(2,2,0);
_plnum = 2;_pdate = '';_pltype = 1_6111_pframe = {0,0}_pcross = 1;
_pcolor = 12_111121_pszht = 0.251_pnumht = 0.251_pstilht = 0.17;
setwind(1);
 bound = rep1 + (-2^0.2) _stderr1;
title('Fig. 3.4. Estimated responses of consumption to a forecast'\
'\Error 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'\
'\Error 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'\
'\Error 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'\
'\Error impulse in consumption with two-standard error bounds');
xy(period, bound);
graphpr('c-=1 -cf=sm3i.epsp');
endwind;
```
3.2.5 Spectral methods

3.2.5.1 Understanding the Fourier transform

```r
#--- Data --->
N = 248;
t = seq(0,2*pi/N,N+1);
x1 = sin(t);
x2 = cos(2*t);
x3 = rnorm(N+1,1)*0.2;
y = x1 + x2 + x3;

#--- Fourier transform --->
#d = fourier(y);
(r,theta) = topole(r);

#--- Inverse Fourier transform --->
r1 = msubstr(r, r > 100, 0);
d1 = tocart(r1,theta);
y1 = real(inverse_fourier(d1));
r2 = msubstr(r, r < 100, 0);
d2 = tocart(r2,theta);
y2 = real(inverse_fourier(d2));
```

```r
#--- Graphs --->
graphset;
begwind;
window(2,2,0);
_plate = "";
_pnum = 2;
_pnumbt = 0.25;
setwind(1); ytics(-2:1.5,0.5,0);
xtics(0,20,5,0);
xy(t,y);
setwind(2);
graphset;
_plmht = 0.20;
_plct1 = -1;
polar(r1,theta);
setwind(3);
_plct1 = 0;
polar(r2,theta);
setwind(4);
```
3.2. TSM

```c
ytics(-2.1.5,0.5,0);
xtics(0,20,5,0);
wy(t1,y1,y2);

graphpsr("-c1 -cf=tsm1a.eps");
endwin;
```

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

```c
/**
 ** Spectral Kolmogorov-Smirnov test
 ** BROOKWELL and DAVIS [1991], Times Series: Theory and Methods,
 ** Springer-Verlag, New York, pages 339-342
 */

row;
library tsm, optum,pgraph;

load data[] = frfdom.asc;

r = packr(log(data./lag1(data))); /* Returns */

t = fractional_filter(r,-0.20);

fig = fractional_filter(r,0.2);

lambda = sort(lambda) / 2;

C1 = summary[I1[1:q]]/numc[I1[1:q]];
C2 = summary[I2[1:q]]/numc[I2[1:q]];
C3 = summary[I3[1:q]]/numc[I3[1:q]];

x = seq(1,q);

k_alpha = -1.645 / 0.5;  /* 1% level */

graphpsr;
title('Kolmogorov-Smirnov test using the spectral analysis');
```
3.2.5.5 Covariance functions

new;
library tm, optim,pgraph;
TSMset;

_fourier = 1;

readed 123456;

Z = eye(2); d = 0;0;
let H[2,2] = 0.2 0 0.1;
let T[2,2] = 0.5 0.3 0.5 0.5;
c = 0.0; R = 111; Q = 0.26;

call SSM_build[Z,d,H,T,c,R,Q,0];
[y,a] = RND_SSM(0,100);
y = y - mean(y);
{lambda,1} = POOM2(y);
g = nlgf_SSM(lambda);

/* first component */

CV1a = real(inverse_fourier([.,1]));
CV1b = real(inverse_fourier([g[.,1]]));
covTD = autoc(y[.,1],100);

/* Autocovariances */

proc autoc(x,k);
local t,rho;
pack(x); t=rows(x);
rho = conv(x,rev(x),t-k,t);
resrho=rho/t; endp;

/* second component */

CV2a = real(inverse_fourier([.,4]));
CV2b = real(inverse_fourier([g[.,4]]));

/* first component/second component */

CV3a = real(inverse_fourier([.,3]));
CV3b = real(inverse_fourier([g[.,3]]));
3.2. TSM

/* second component/first component */
CV4a = real(inverse_fourier([1,2]));
CV4b = real(inverse_fourier([g[1,2]]));

@--- Plots --> @

graph6;
begin;
window(2,2,1);
 picturesque = "estimated\000theoretical\000Time domain calculus";
 pleqtr = "estimated\000theoretical\000Time domain calculus";
 xlabel('t(Lag)');

setwindow(1);
title('cov(y[t],y[t-(t-Lag)])');
xy(seq(0,1,11),CV1a[1:11]-CV1b[1:11]'-covTD[1:11]);

pleqtr = 0;

setwindow(2);
title('cov(y[t],y[t-(t-Lag)])');
xy(seq(0,1,11),CV2a[1:11]-CV2b[1:11]);

setwindow(3);
title('cov(y[t],y[t-(t-Lag)])');
xy(seq(0,1,11),CV3a[1:11]-CV3b[1:11]);

setwindow(4);
title('cov(y[t],y[t-(t-Lag)])');
xy(seq(0,1,11),CV4a[1:11]-CV4b[1:11]);

graphpt('c=1 -cf=tms4e.eps');
end;

3.2.5.6 Varma parameters estimation in the frequency domain

new;
library tsm, optim,pgraph;
TSMset;
output file = tms4f.out reset;
load reinsei[100,2] = reinsei.asc;
invent = reinsei[,1];
invent = reinsei[,2];
di = invent - log(invent);
y = di'\invent;
(lambd,ly) = PDSMDV(y);         /* Multidimensional periodogram */

/*
** Procedure to compute the multidimensional mgf of the SSM
*/
3.2.6 Wavelets analysis

3.2.6.1 Understanding the Wavelet and Wavelet Packets transforms

```r
new;
library tsm, optsum, pgraph;
```
3.2. TSM

TSMset;

@--- Data --->@
N = 255;
t = seq(0.6*pi/N, N+1);
x1 = sin(t);
x2 = cos(2*t);
x3 = runif(N+1,1)*0.20;
y = x1 + x2 + x3;

@--- wavelet transform --->@
{H,G,Htilde,Gtilde} = Coiflet(2);
_wcenter = 0;
w = wts(y,H,G,0);
w1 = mbstute(w, abs(w) > 1, 0);
w1 = iwt(w1,Htilde,Gtilde,0);
w2 = mbstute(w, abs(w) < 1, 0);
w2 = iwt(w2,Htilde,Gtilde,0);

graphset;
begwind;
  window(2,2,0);
  _plate = '***'; _psnum = 2; _psnumht = 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(seq(1,1,rows(w)),w1);
  setwind(3);
    ytics(-0.5,0.5,0);
    bar(seq(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.ep3');
endwind;
3.2.6.2 Wavelet and Wavelet Packets representation

```r
library tmv, optimum, pgraph;

M = 11;
N = 2^M;
t = seq(0.2/N,N);
chirp = sin(100*pi*t^2);

{H,G,tilde,G,tilde} = Coiflet(2);

_wcenter = 0;
w = wt(chirp,H,G,0);  /* Wavelet transform */

pkt = wpkt(chirp,H,G,0);  /* Wavelet Packet transform */

graphset;
call wpplot(v,0,1);
   _pnum = 2; _plate = '''
   _pnum = 0; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''
   _pnum = 2; _plate = '''

M = 6;
Nobs = 2^M;

call varput(0,''Base0'');  /* Time domain */
call varput(M+1:en(Nobs,1),''Base1'');  /* Frequency domain */
call varput(M+1:obs(M,-1,1),''Base2'');  /* Wavelet transform */
   (better frequency localization at lower frequencies)

   /*
call varput(seqs(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|2:6|6|6,4,''Base5'');  /* Wavelet packet transform */

graphset;
begwind;
window2(2,3,0,0.5);

setwind(1);
   _plate = '''
   _pnum = 0; _paxes = 0; _ptitle = 2.5;
   title(''TIME/FREQUENCY representation of basis'');
draw;

   _pnum = 2; _paxes = 1; _ptitle = 0.20; _pnumtxt = 0.25; _paxt = 0.25;
   title(''''

i = 0;
do until i > 5;

str = 'Base'' $+ ftn(i,''Max''',1,0);
B = varget(str);

setwind(2 + i);

call BasimPlot(B,M);
xlabel(''Time localization'');
ylabel(''Frequency localization'');
draw;
```

3.2. TSM

\[ i = i + 1; \]
\[ \text{enddo;} \]
\[ \text{graphplt(''-c=1 -cf=tms5b2.epsl');} \]
\[ \text{endwind;} \]

![Linear chirp \( \sin(100\pi t^2) \) Wavelet Coefficients](image)

![TIME/FREQUENCY representation of basis](image)

3.2.6.3 Data denoising

\[ \text{new;} \]
\[ \text{library tms, optsym, pgraph;} \]
\[ \text{rmdseed 123;} \]
\[ \text{Nobs = 2^10;} \]
\[ t = \text{seq}(0,2*pi/Nobs,Nobs); \]
\[ x_+ = \sin(t) + \sin(2*t); \]
\[ x = x_+ + \text{rmdn(Nobs,1)}*0.4; \]
\[ \{H,G,Hilde,Gilde\} = \text{Daubechies(12);} \]
\[ \_wcenter = 0; \]
\[ w = \text{wt}(x,H,2,0); \]
\[ w1 = \text{ViusShrink}(w,'\text{'Hari'}'); \]
\[ w2 = \text{ViusShrink}(w,'\text{'Sof'}'); \]
\[ y1 = \text{iwt}(w1,Hilde,Gilde,0); \]
\[ y2 = \text{iwt}(w2,Hilde,Gilde,0); \]
3.2.6.4 Subbands coding

```r
new;
library pgraph,tam,optmm;

rnsseed 1234;
Nobs = 2^9;
s = seq(1,1,Nobs);
sigma = 1;
x0 = miss(0,0);
x = RN:d:arma(0.62)[0.35,2,0,sigma,x0,Nobs);
{H,G,tilde,Gtilde} = Daubechies(6);

_wcenter = 2;
w = wt(x,H,G,0);
L1 = 5[6:7:6]9;10; /* lower frequencies */
L2 = 4[2:3]4; /* higher frequencies */

bd1 = Extract[x,L1];
bd2 = Extract[x,L2];
y1 = iwt(bd1,H,tilde,Gtilde,0);
y2 = iwt(bd2,H,tilde,Gtilde,0);
y3 = y1 + y2;
```
3.2. TSM

begwind;
window(3,1,0);
setwind(1);
graphset;
_plate = '##'; _pnum = 2; _psnumht = 0.25; _ptitlht = 0.30; _pexht = 0.25;
title("Wavelet coefficients (first subband coding procedure)'');
_wcolor = seqn(1,1,0); _xline = 0;
call wplot(w,0,bndl);
xtics(0,Nobs/2,32,0);
draw;
setwind(2);
graphset;
_plate = '##'; _pnum = 2; _psnumht = 0.25; _ptitlht = 0.30; _pexht = 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);
graphpt(''-c1 -cf=tsm5d.eps'');
endwind;

3.2.6.5 Scalogram

/*
 ** ARIND and VIDAKOVIC [1996]. On wavelet scalograms and their applications
 ** in economic time series, DP 95-21, ISDS, Duke University
 ** ARIND [1995], Time series forecasts via wavelets: an application
 ** to car sales in the Spanish market, DP 94-30, ISDS, Duke University
 ** */

new;
library tsm, optsum, pgraph;

C1 = intquad1(&gfunc,(8*pi)|0)/(8*pi);
C2 = intquad1(&hfunc,(8*pi)|0)/(8*pi);

t = seqn(1,1,2*12);
yt = gfunc(8*pi*t/4096) - C1;
xh = hfunc(8*pi*t/4096) - C2;
x = yt+xh;

{H,Htilda;Otilda} = Daubechies(12);

w = wt(wt,H,Htilda,Gtilda,0);
output file = tsm5e.out reset;
E = scalogram(w);
output off;

bnd = split(w,7);
x1 = iwt(bnd[..1],Htilda,Gtilda,0);

2D wavelet coefficients of the time series

The wavelet coefficients of the time series are shown in the top graph. The second graph shows the original signal and the reconstructed signal from the wavelet coefficients (y1). The bottom graph displays the reconstructed series after adding the original signal to the reconstructed signal (y1 + y2).
x2 = iwt(hm(.,2),Stilde,Stilde,0);

graphnet;
beginwind;
    window(2,2,0);
    _pnum = 2; _pdate = '1'; _ptitleht = 0.26; _pnumht = 0.20;
    setwind(1);
    title('data');
    xy(t,t);
    nextwind;
    _pnumht = 0.16;
    xtics(0,13,1,0);
    asclabel(lab,0);
    title('scalogram');
    bar(0,1);
    nextwind;
    graphnet; _ptitleht = 0.26; _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(sin(x/pi));
    retp(y);
endp;

-------------------------------
Scalogram
-------------------------------
c(0)' 2 112.01985
E(0)    0.02141
E(1)    0.05878
E(2)    4988.64820
E(3)    3208.65411
E(4)    19.69621
E(5)    85.69263
E(6)    748.16123
E(7)    29.73985
E(8)    6.21492
E(9)    0.9120
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:

\[
\begin{align*}
\text{target\_path} & \quad \text{gauss.dll \\& gsend.dll} \\
\text{target\_path\lib} & \quad \text{gauss.kg} \\
\text{target\_path\src} & \quad \text{source code files of the Gauss library}
\end{align*}
\]

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_Enable**: 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.

ew;
dllibrary c:\engine\gseng.dll;

str = '$output file = gel.out reset;'
  \'\'random 123, x = rndv(200,200);\'\'
  \'\'x0 = bsec; y = inv(x); tl = bsec;\'\'
  \'\'print /flush (tl-t0); output off;\'\';
dllcall GAUSS_O_Initialize;
Gauss Engine executes Gauss programs, but Gauss Engine and Gauss are two different 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_GInitialize;
str = '''y = rmdn(3,3); x = y[1,1];''';
dllcall /r GAUSS_GCompileString(str);
dllcall /r GAUSS_GExecute;
show;
print '''----------------------------------------''';
x = rmdn(3,3);
show;
print '''----------------------------------------''';
output off;
str = '''output file = ge2.out on;
   ''''x = x^2;'''
   ''show; output off;''';
dllcall /r GAUSS_GCompileString(str);
dllcall /r GAUSS_GExecute;
dllcall GAUSS_GShutdown;

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>32</td>
<td>[01320020]</td>
</tr>
<tr>
<td>STRING</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>[00000000]</td>
</tr>
<tr>
<td>MATRIX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>32</td>
<td>[01320020]</td>
</tr>
<tr>
<td>STRING</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>72</td>
<td>[01320040]</td>
</tr>
<tr>
<td>MATRIX</td>
<td>3,3</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>32</td>
<td>[01320020]</td>
</tr>
<tr>
<td>STRING</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>72</td>
<td>[01320040]</td>
</tr>
<tr>
<td>MATRIX</td>
<td>3,3</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>32</td>
<td>[01320020]</td>
</tr>
<tr>
<td>STRING</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>72</td>
<td>[01320040]</td>
</tr>
<tr>
<td>MATRIX</td>
<td>3,3</td>
<td></td>
</tr>
</tbody>
</table>

---

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
4.1. THE GAUSS ENGINE

- Gauss Engine could run Gauss files.

```c
new;
dllibrary c:\engine\gseng.dll;
dllcall /r GAUSS_G_Initialize;

prg = "d:\gauss\conf3\var1.prg";
dllcall /r GAUSS_G_CompileFile(prg);
dllcall /r GAUSS_G_Execute;
dllcall GAUSS_G_Shutdown;
```

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

```c
#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] =
{
  0.281130116112139, 0.333434643689543, 0.53653653041197, 0.05748229081660,
  0.9449661804648541, 0.60641532834002, 0.8258564772550390, 0.75027175949492,
  0.52250332316292, 0.0568114641553905, 0.677504625234414, 0.853463818197564,
  0.36155324797391, 0.404659466352314, 0.034288057909742, 0.210895204916596,
};
#define YR 4
#define YC 4

double y[YR+YC] =
{
  0.577037780468247, 0.043889922644073, 0.863501355638071, 0.00838666083711,
  0.406743612149460, 0.476973535410903, 0.18527781575153, 0.892547697593646,
  0.9456666686868622, 0.116668860065867, 0.960371665001785, 0.833994163898582,
  0.760000020804672, 0.274980544885317, 0.15035482452318, 0.231698946653765,
};

double *x;

int main(int argc, char *argv[])
{
    Mat2D zDesc;
    int i, j, zR, zC;

    GAUSS_G_Initialize();
    GAUSS_G_Set2DMatrix("x", XR, XC, 0, x);
    GAUSS_G_Set2DMatrix("y", YR, YC, 0, y);
    GAUSS_G_CompileString("format /ds 20,16; print y/x; z = y/x;");
    GAUSS_G_Execute();
    GAUSS_G_Get2DMatrix(&zDesc, "z");

    z = zDesc.address;
    zR = zDesc.rows;
    zC = zDesc.cols;

    for (i = 0; i < zR; i++)
    {
        printf("%s", z);
        for (j = 0; j < zC; j++) printf("%20.15lf", *(z+i*zC+j));
    }
    printf("\n");
    GAUSS_G_Shutdown();
    return 0;
}
```

4.1.3 Some examples with Excel

- Not yet done.
4.1.4 Some examples with Visual Basic

![Visual Basic Example]

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, le):** 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.
4.2. MERCURY_GE

- **gestrget(ByVal str 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 str As String, slen As Long):** Length of a string (Gauss symbol).
- **gevalget(ByVal str As String, value):** Gets scalars from the Gauss symbol table.
- **gevalput(ByVal str 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
    gestrput "'k'\', "'b6'
    gestrput "'f'\', "'b6'
    gestrput "'sigma'\', "'b7'
    gestrput "'tau'\', "'b6'
    gestrput "'r'\', "'b6'

    'call
    gexec '"d9:d15';
    gexec '"c = bs_call_future(k,f,tau,r);''
    gestrget "'c'\', "'e5''

    'put
    gexec '"d17:d23'
    gexec '"p = bs_put_future(k,f,tau,r);''
    gestrget "'p'\', "'f6''
End Sub
```

![Microsoft Excel - b&s](image-url)
4.2.2.2 VC++ example

// cpregl.cpp : Defines the entry point for the application.

VB example

```vbnet
x = xa[3,3]:
y = y'y:
s = svd(y):
print s:
```

```
1.1 1.2 1.3
2.1 2.2 2.3
3.1 3.2 3.3
```

```
x = x+2:
```
4.2. *MERCURY* _GE

#include "stdafx.h"

int WINAPI WinMain(HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpCmdLine,
  int nCmdShow)
{
    // TODO: Place code here.

    Mat2D mzDesc;
    char buffer[256];
    int ii, jj;
    int elem[1];
    double mz[1];
    double *m;
    int pval = 0;
    char *name = "GAUSSHOME";
    long nsize = 256;
    char sbuf[256];
    int len = 256;
    
    if (GetEnvironmentVariable(name, sbuf, nsize)==0)
    {
        MessageBox(0, "The GAUSSHOME environment variable was not found!", "MERCURY_GE", 0);
        return 1;
    }
    else
    MessageBox(0, sbuf, '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", "error", 0); return 1;
    }
    
    // Create a gauss exec
    geexec("' tt = ' This is a string \n \n'; tt, ' ');
    geexec("'x = 14; x'; ' ');
    geoutput("show");
    geoutput("reset");
    
    // Create a gauss string
    strcpy(buffer, "The fox went out on a chilly night");
    geoutput("str", buffer);
    geexec("'str'; ' ');
    geoutput("show");
    geoutput("reset");
    
    // Create a Gauss matrix
    double q[4*3] = { 0.261130110612139, 0.333434643689543, 0.5363553689421197,
                      0.944666104946541, 0.0684133434003, 0.625666477950292,
                      0.522050329318982, 0.6561441553909, 0.677300492534414,
                      0.351553247077391, 0.40465946332314, 0.03429057960742, },
    mzDesc.rows = 4;
    mzDesc.cols = 3;
    mzDesc.address = q;
    geoutput("qmat", &mzDesc);
    geexec("'qmat'; ' ');
    geoutput("show");
    geoutput("reset");
    
    // Create a Gauss scalar
    mz[0] = 1.2;
    geoutput("'mz', mz");
    geexec("'mz'; ' ');
    geoutput("show");
    geoutput("reset");
    
    // Execute Gauss code
    geexec("'x = 14; yy = x^2 *x^3; ' ' ");
    
    // Retrieve Gauss matrix
gematget('yy', &mmDesc);
pi = mmDesc.address;
nj = mmDesc.cols;
printf(buffer, 'Matrix size: %d %d', ii, jj);
MessageBox(0, buffer, 'gematget', 0);
printf(buffer, 'Matrix value: %zf %zf', *z, *(z+1));
MessageBox(0, buffer, 'gematget', 0);

// Retrieve Gauss scalar
gexec('z = -14.7;' );
gevalget('z', zz);
printf(buffer, 'Scalar value: %f', zz[0]);
MessageBox(0, buffer, 'gevalget', 0);

// Retrieve Gauss string
gstrput('hw', 'Hello World');
gestrlen('hw', &elem);
gstrget('hw', buffer);
printf(buffer, '\nString length: %d', &elem[0]);
strcat(buffer, buffer);
MessageBox(0, buffer, 'gstrlen & gstrget', 0);
geput('reset');

// clipboard support
gexec('x = rmdn(3,2); x');
gecopy('x');
gstrput('tt' , 'x is copied to the clipboard');
geexec('tt' );
geexec(' new ');
gepaste('z' );
gstrput('tt' , 'x is pasted from the clipboard');
geexec('tt' , 'x');
geput('show');
geput('reset');

return 0;

4.2.2.3 VJJ++ example

- Needs VM Java component.
4.2.3 Three explained examples

4.2.3.1 Markowitz portfolio and Excel

The VBA code is:

```vba
Sub compute()
    Dim GaussProgram As String
```
Sheets("'sheet!'").Select
'geopen

gemput 'phi', 'g10'
gemput 'mu', 'c12_h12'
gemput 'Mcov', 'c14_h19'
GausProgram = '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); ' + _
'(theta1, theta2, theta3, theta4, recode) = Gprg(sv, Q, R, A, B, C, D, 0); ' + _
'theta = theta + .dotfn(theta', 0);'
gexec GausProgram

gemget 'theta', 'c8'
'geclose
End Sub

Sub Clear()
    Sheets("'sheet!'").Select
    Range("'g10'").Value = 0
    Range("'c8_h18'").Value = 0
End Sub

### 4.2.3.2 Linear algebra and Visual Basic

![Linear Algebra Toolbox](image1)

![Display the results](image2)
4.2. MERCURY GE

The VB code of the first window is

Private Sub cmdClear_Click()
    grdMatrix.Clear
    cmdExecute.Enabled = False
End Sub

Private Sub cmdDisplay_Click()
    frmWB1.Hide
    frmWB2.cmdDisplay
    frmWB2.Show
End Sub

Private Sub cmdExecute_Click()
    Dim InputMatrix() As Double
    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
        For j = 1 To N Step 1
            InputMatrix(i, j) = Val(grdMatrix.Text)
        Next j
    Next i

    InputMatrixName = 'x'
geomatput InputMatrixName, InputMatrix()

Select Case cholLinearAlgebra.ListIndex
    Case 0
        GaussProgram = 'c:\engine\prg\balance.prg'
        gexec GaussProgram
    Case 1
        GaussProgram = 'c:\engine\prg\cond.prg'
        gexec GaussProgram
    Case 2
        GaussProgram = 'c:\engine\prg\crout.prg'
        gexec GaussProgram
    Case 3
        GaussProgram = 'c:\engine\prg\chess.prg'
        gexec GaussProgram
    Case 4
        GaussProgram = 'c:\engine\prg\lub.prg'
        gexec GaussProgram
    Case 5
        GaussProgram = 'c:\engine\prg\null.prg'
        gexec GaussProgram
    Case 6
        GaussProgram = 'c:\engine\prg\pinv.prg'
        gexec GaussProgram
    Case 7
        GaussProgram = 'c:\engine\prg\qqr.prg'
        gexec GaussProgram
    Case 8
        GaussProgram = 'c:\engine\prg\rank.prg'
        gexec GaussProgram
    Case 9
        GaussProgram = 'c:\engine\prg\schur.prg'
        gexec GaussProgram
    Case 10
        GaussProgram = 'c:\engine\prg\svd.prg'
        gexec GaussProgram
End Select

cmdDisplay.Enabled = True

End Sub

Private Sub cmdExit_Click()
gclose
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
    grnMatrix.Row = i - 1
    For j = 1 To N Step 1
        Prompt = 'Row #' + i + ' - Col #' + j
        Mij = InputBox(Prompt, 'Enter Data', '', 500, 5000)
        grnMatrix.Col = j - 1
        grnMatrix.Text = Mij
    Next j
Next i

cmExecute.Enabled = True

End Sub

---------------------------------------------
Private Sub Form_Load()
    chol.linearAlgebra.AddItem "'BALANCE - Balances a matrix'
    chol.linearAlgebra.AddItem "'LU' - Computes LU Decomposition with row pivoting'
    chol.linearAlgebra.AddItem "'CND - Computes condition number'
    chol.linearAlgebra.AddItem "'PIMC' - Computes Moore-Penrose pseudo-inverse'
    chol.linearAlgebra.AddItem "'QR' - Computes QR decomposition'
    chol.linearAlgebra.AddItem "'RANK' - Computes rank of a matrix'
    chol.linearAlgebra.AddItem "'SCHUR' - Computes Schur decomposition'
    chol.linearAlgebra.ListIndex = 0

cmDisplay.Enabled = False

End Sub

---------------------------------------------
Private Sub txtDimension_Change()
    grnMatrix.Clear
    cmExecute.Enabled = False
    grnMatrix.rows = Val(txtDimension.Text)
    grnMatrix.cols = 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 cmDisplay()
    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.chol.linearAlgebra.ListIndex
        Case 0
            txt = "'Method: '" OutputStringName = "'method'
            gestarget OutputStringName, OutputString
            txt = txt & OutputString
            lblMethod.Caption = txt
            OutputMatrixName = "'b'
            gestarget 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
            Next i
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'"
getOutputOutputMatrixName, 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
Next i
Case 1
  txt = "'Method: '
  OutputStringName = 'method'
  getOutputOutputStringName, OutputString
  txt = txt & OutputString
  lbMethod.Caption = txt

  txt = "'Syntax: '
  OutputStringName = 'syntax'
  getOutputOutputStringName, OutputString
  txt = txt & OutputString
  lbSyntax.Caption = txt

  txt = "'c = ' & Chr(13) & Chr(10)
  OutputMatrixName = "'c'"
  getOutputOutputMatrixName, 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
Next i
Case 2
  txt = "'Method: '
  OutputStringName = 'method'
  getOutputOutputStringName, OutputString
  txt = txt & OutputString
  lbMethod.Caption = txt

  txt = "'Syntax: '
  OutputStringName = 'syntax'
  getOutputOutputStringName, OutputString
  txt = txt & OutputString
  lbSyntax.Caption = txt

  txt = "'L = ' & Chr(13) & Chr(10)
  OutputMatrixName = "'L'"
  getOutputOutputMatrixName, 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
Next i
Case 3
  txt = "'Method: '
  OutputStringName = 'method'
  getOutputOutputStringName, OutputString
  txt = txt & OutputString
  lbMethod.Caption = txt
```
txt = "'Syntax: 
OutputStringName = "'syntax"'
generate OutputStringName, OutputString
txt = txt & OutputString
lblSyntax.Caption = txt

txt = "'h = ' & Chr(13) & Chr(10)
OutputMatrixName = "'h'"
generate OutputMatrixName, OutputMatrix()
nr = UBound(OutputMatrix, 1)
cmp = UBound(OutputMatrix, 2)
For i = 1 To nr Step 1
    For j = 1 To nc Step 1
        txt = txt & OutputMatrix(i, j) & "\n"
    Next j
Next i
txt = txt & Chr(13) & Chr(10)
txt = txt & "'z = ' & Chr(13) & Chr(10)
OutputMatrixName = "'z'"
generate OutputMatrixName, OutputMatrix()
nr = UBound(OutputMatrix, 1)
cmp = UBound(OutputMatrix, 2)
For i = 1 To nr Step 1
    For j = 1 To nc Step 1
        txt = txt & OutputMatrix(i, j) & "\n"
    Next j
Next i
txt = txt & Chr(13) & Chr(10)
Next i

Case 4

txt = "'Method: 
OutputStringName = "'method"'
generate OutputStringName, OutputString
txt = txt & OutputString
lblMethod.Caption = txt

txt = "'I = ' & Chr(13) & Chr(10)
OutputMatrixName = "'I'"
generate OutputMatrixName, OutputMatrix()
nr = UBound(OutputMatrix, 1)
cmp = UBound(OutputMatrix, 2)
For i = 1 To nr Step 1
    For j = 1 To nc Step 1
        txt = txt & OutputMatrix(i, j) & "\n"
    Next j
Next i
txt = txt & Chr(13) & Chr(10)
txt = txt & "'u = ' & Chr(13) & Chr(10)
OutputMatrixName = "'u'"
generate OutputMatrixName, OutputMatrix()
nr = UBound(OutputMatrix, 1)
cmp = UBound(OutputMatrix, 2)
For i = 1 To nr Step 1
    For j = 1 To nc Step 1
        txt = txt & OutputMatrix(i, j) & "\n"
    Next j
Next i
txt = txt & Chr(13) & Chr(10)
Next i

Case 5

txt = "'Method: 
OutputStringName = "'method"'
generate OutputStringName, OutputString
txt = txt & OutputString
lblMethod.Caption = txt

txt = "'b = ' & Chr(13) & Chr(10)
OutputMatrixName = "'b'"
generate OutputMatrixName, OutputMatrix()
nr = UBound(OutputMatrix, 1)
cmp = 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'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblMethod.Caption = txt
    
    txt = "'Syntax: '"
    OutputStringName = "'syntax'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblSyntax.Caption = txt
    
    txt = "'y = ' & Chr(13) & Chr(10)
    OutputMatrixName = "'y'"
    getset 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'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblMethod.Caption = txt
    
    txt = "'Syntax: '"
    OutputStringName = "'syntax'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblSyntax.Caption = txt
    
    txt = "'q_1 = ' & Chr(13) & Chr(10)
    OutputMatrixName = "'q_1'"
    getset 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'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblMethod.Caption = txt
    
    txt = "'Syntax: '"
    OutputStringName = "'syntax'"
    getset OutputStringName, OutputString
    txt = txt & OutputString
    lblSyntax.Caption = txt
    
    txt = "'k = ' & Chr(13) & Chr(10)
    OutputMatrixName = "'k'"
    getset 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"
getget OutputStringName, OutputString
   txt = txt & OutputString
   lblMethod.Caption = txt
   txt = "Syntax: "
   OutputStringName = "syntax"
getget OutputStringName, OutputString
   txt = txt & OutputString
   lblSyntax.Caption = txt
   txt = "'x = ' & Chr(13) & Chr(10)
   OutputMatrixName = "x"
getget 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 & "'x = ' & Chr(13) & Chr(10)
   OutputMatrixName = "x"
getget 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
Next i
Case 10
   txt = "Method: "
   OutputStringName = "method"
getget OutputStringName, OutputString
   txt = txt & OutputString
   lblMethod.Caption = txt
   txt = "Syntax: "
   OutputStringName = "syntax"
getget OutputStringName, OutputString
   txt = txt & OutputString
   lblSyntax.Caption = txt
   txt = "'x = ' & Chr(13) & Chr(10)
   OutputMatrixName = "x"
getget 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
Next i
End Select
Display.Text = txt
End Sub

The Gauss programs are:

---

-> BALANCE.PRG

method = 'BALANCE = Balances a matrix';
syntax = '{b,z) = BALANCE(x)';
4.2. MERCURY_GE

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

method = 'COND - Computes condition number';
syntax = 'c = COND(x)';

c = COND(x);
-> CRUT.CRG

method = 'CRUT - Computes Crout Decomposition';
syntax = 'y = CRUT(x)';

y = CRUT(x);
L = lomat(y);
U = upmat(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);
-> QRR.PRG

method = 'QRR - Computes QR decomposition';
syntax = '{q,r} = QRR(x)';

{q,r} = QRR(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 = '{a,z} = SCHUR(x)';

{a,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**
- **Advanced Statistics**
- **Descriptive Statistics**
### Local level model estimation

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Estimates</th>
<th>Std. Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma_eta</td>
<td>2.17413</td>
<td>0.71395</td>
</tr>
<tr>
<td>sigma_omega</td>
<td>5.03743</td>
<td>0.60324</td>
</tr>
</tbody>
</table>

### Data

<table>
<thead>
<tr>
<th>Index</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
</tr>
</tbody>
</table>