SSMMATLAB, a Set of MATLAB (OCTAVE) Programs for the Statistical Analysis of State Space Models

Víctor Gómez

Ministerio de Hacienda y A.P.
Madrid, SPAIN

January 2019

Abstract

The purpose of this document is to describe the functions written in MATLAB of the toolbox SSMMATLAB for the statistical analysis of state space models. These functions can also run in the OCTAVE free software platform. The state space model considered is very general. It may have univariate or multivariate observations, time–varying system matrices, exogenous inputs, regression effects, incompletely specified initial conditions, such as those that arise with nonstationary VARMA models, and missing values. There are functions to put frequently used models, such as multiplicative ARIMA or VARMA models, cointegrated VARMA models, VARMAX models in echelon form, transfer function models, and univariate structural or ARIMA model–based unobserved components models, into state space form. Once the model is in state space form, other functions can be used for covariance computation, likelihood evaluation, model estimation, forecasting and smoothing. Functions spectral estimation and for automatic ARIMA and transfer function identification and automatic outlier detection are also provided. A companion book called "Linear Time Series With Matlab and Octave", written by the author, is forthcoming in Springer Verlag, Statistics and Computing Series.

Contents

1 Software Installation

To install SSMMATLAB, uncompress the zip file and copy its contents into a directory, for example SSMMATLAB. There should be five subdirectories with names DATA, RESULTS, SPEC, GRAPHS, and USMSPEC. All the data files used in the demos are in the subdirectory DATA. The subdirectory RESULTS is where all program results are written. The subdirectory GRAPHS is used to eventually write the plots produced by the programs. In the subdirecoty SPEC, you can find all specification files for the different ARIMA and transfer function demos. That is, each specification file contains instructions to read data, generate structures, etc. Finally, the subdirectory USMSPEC has the same function than the subdirectory SPEC but for univariate structural models.

If the user desires to work in a directory different to the one in which the program has been installed, he should **first add to the MATLAB path the directory where SSMMATLAB has been installed**. Then, if the new directory in which the user intends to work is called WORK, for example, the user can start working with SSMMATLAB in that subdirectory. Some of the programs used by SSMMATLAB, for example those that deal with ARIMA, transfer function or univariate structural models, can optionally and automatically create two subdirectories in WORK, called RESULTS and GRAPHS, where the program results and plots are written.

All the functions in SSMMATLAB have been proved to also run under the free software OCTAVE platform.

2 acgf

```
function c=acgf(phi,th,nc)
%
% This function computes the autocovariance function of an ARMA process
% phi(B)z_t=th(B)a_t
% The parameter nc is the number of desired autocovariances plus one,
% because the variance is included: g(0),g(1),...,g(nc-1)
%
%        Input parameters:
%      phi: a (p+1 x 1) array, where p is the degree of phi(z),
%           containing the ocefficients of phi(z) in degree descending
%           order
%      th : a (q+1 x 1) array, where q is the degree of th(z),
%           containing the ocefficients of th(z) in degree descending
%           order
%      nc : an integer, the number of desired covariances plus one
%        Output parameters:
%      c  : a (nc x 1) array containing the autocovariances in the order
%           g(0),g(1),...,g(nc-1)
```

3 akaikessm1

```
function [T,H,Z,ferror]=akaikessm1(phip,thp)
%
%        This function obtains Akaike's state space representation of
%        minimal dimension corresponding to the ARMA model
%
%        y_t = [thp(z)/phip(z)] a_t
%
%        where a_t is (0,1). The model is
%
%        x_{t+1} = T x_t + H*a_t
%           y_t    = Z x_t + a_t,
%
%        where
%            [0 1 0  ... ....   0]        [ Psi_1    ]
%            [0 0 1  ... ....   0]        [ Psi_2    ]
%     T =    [ ...   ...  ...    ],   H = [ ...      ],
%            [0 0 0  ... ....   1]        [ Psi_{r-1}]
%            [-phi_r ...   -phi_1]        [ Psi_r    ]
%
%     Z =    [1 0 0  ...  ... 0],
%
%     r = degree(phip) = degree(thp)  and phi^{1}(z)*theta(z) =
%     Psi_0 + Psi_1*z + Psi_2*z^2+ ...
%
%        Input parameters:
%        phip   : a (1 x np+1) array
%        thp    : a (1 x nt+1) array
%
%        Output parameters:
%        T    : a (r x r) matrix
%        H    : a (r x 1) matrix
%        Z    : a (1 x r) matrix
```

4 akaikessm2

```
function [T,H,Z,ferror]=akaikessm2(phip,thp)
```

```
%
%        This function obtains Akaike's state space representation of
%        nonminimal dimension corresponding to the ARMA model
%
%        y_t = [thp(z)/phip(z)] a_t
%
%        where a_t is (0,1). The model is
%
%        x_t  = T x_{t-1} + H*a_t
%        y_t  = Z x_t ,
%
%        where
%          [0   1   0  ... ....    0]          [   1      ]
%          [0   0   1  ... ....    0]          [ Psi_1    ]
%     T =  [   ...     ... ....     ]  ,   H = [ ...      ],
%          [0   0   0  ... ....    1]          [ Psi_{r-1}]
%          [0 -phi_r   ...    -phi_1]          [ Psi_r    ]
%
%     Z =    [1 0 0  ...  ... 0],
%
%     r = degree(phip) = degree(thp) and phi^{1}(z)*theta(z) =
%     Psi_0 + Psi_1*z + Psi_2*z^2+ ...
%
%        Input parameters:
%        phip   : a (1 x np+1) array
%        thp    : a (1 x nt+1) array
%
%        Output parameters:
%        T    : a ((r+1) x (r+1)) matrix
%        H    : a ((r+1) x 1)     matrix
%        Z    : a (1 x (r+1))     matrix
```

5 arima2rspol

```
function [phir,phis,thr,ths,phirst] = ...
                               arima2rspol(phi,Phi,th,Th,freq,dr,ds)
%*********************************************************************
%
% This function returns the trend, seasonal and stationary polynomials,
% phir, thr, phis, ths and phirst, corresponding to the canonical
% decomposition of an ARIMA model,
%
```

```
%       y_t = [thr(B)/phir(B)]b_t + [ths(B)/phis(B)]c_t + u_t,
%
% where phirst, if it exists, is a stationary factor of phir.
%
% The original ARIMA model is given by the regular and seasonal
% polynomilas, phi, Phi, th and Th, in matrix polynomial format, such
% that
%
%       phi(B)*Phi(B^s)y_t = th(B)*Th(B^s)*a_t
%
% For example, phi(:,:,1)=1; phi(:,:,2)=-1, etc.
%
%*********************************************************************
%     INPUTS :
%        phi : regular autoregressive polynomial
%        Phi : seasonal autoregressive polynomial
%         th : regular moving average polynomial
%         Th : seasonal moving average polynomial
%       freq : frequency of the data
%         dr : number of regular differences
%         ds : number of seasonal differences
%
%    OUTPUTS :
%       phir : autoregressive trend polynomial
%       phis : autoregressive seasonal polynomial
%        thr : moving average trend polynomial
%        ths : moving average seasonal polynomial
%      phirst: stationary autoregressive trend polynomial (factor of
%              phir)
```

6 arimadefval

```
%script file containing ARIMA default values
```

7 arimaeasy

```
function [out,ser] = arimaeasy(y,freq,varargin)
%********************************************************************
%                       EASY ARIMA MODELING
%
%                            USAGE :
% out = arimaeasy(y,freq,'option1',optionvalue1,'option2',optionvalue2,...)
```

```
%
%       INPUTS :
%------------------
%      REQUIRED
%            y : (ly x 1) array containing the series;
%         freq : data frequency (number of observations per year)
%------------------
%       OPTIONS
%  '[bg_year bg_per]': (1 x 2) array containing the initial year and the
%                       initial period. Default [2000 1]
%               'lam': data transformation (logs), = 0 logs, =1 no logs,
%                      default -1 (test for logs)
%          '[p dr q]': (1 x 3) array containing the regular orders
%                      default: [0 1 1]
%        '[ps ds qs]': (1 x 3) array containing the first seasonal orders
%                      default: [0 1 1]
%                 'S': second seasonality. Default 0
%           '[dS qS]': (1 x 2) array containing the second seasonal orders
%                      default: [1 1]
%             'flagm': flag for mean, =1 mean, =0, no mean, default 0
%                      It has not effect with automatic model
%                      identification
%              'pfix': index array for fixed parameters
%              'vfix': array for fixed parameter values
%            'fixdif': flag for fixing the differencing degrees, =1
%                      degrees are fixed, = 0 not fixed, default 0
%            'autmid': flag for automatic model identification, = 1,
%                      perform automatic model identification, = 0, no
%                      automatic model identification, default 1
%                 'Y': array for regression variables, default []
%          'rnamesrg': string matrix for names of regression variables,
%                      default []
%           'nlestim': flag for nonlinear estimation, = 1, nl estimation,
%                      = 0, no nl estimation, default 1
%               'mvx': flag for nl method, = 1, exact maximum likelihood,
%                      = 0, unconditional least squares, default 1
%               'npr': number of forecasts, default 0
%            'olsres': flag for OLS residuals, = 1, OLS residuals are used,
%                      = 0, uncorrelated residuals (transformation of OLS
%                      residuals) are used, default 0
%                'pr': flag for printing in an external file, = 1, printing
%                       = 0, no printing, default 1
%               'gft': flag for graphics, = 1, plot series, = 0, no plots
%                         = 2, plots are saved but not displayed, = 3, plots
%                         are both saved and displayed, default 0
%                 'out': out = 1 perform outlier detection
%                        = 0 do not perform outlier de
%                'omet': omet = 1 use exact ML for model estimation
%                        = 0 use Hannan-Rissanen
%                   'C': critical value for outlier detection
%                        if negative, it is computed depending on the
%                        sample size
%                  'C0': critical value for outlier detection used in the log
%                        test and automatic model identification, default
%                        C0=2.6 + log(log(ny)) (ny = series length)
%                'schr': = 0 outliers of type AO and TC are considered, =1
%                        outliers of type AO, TC and LS are considered,
%                        default 1
%                 'sp1': (sp1,sp2) span for outlier detection, default sp1 =1
%                        default sp2=ny, where ny = series length
%                 'sp2':
%                'trad': = 0 no trading day effect, = 1 TD effect, = -1, test
%                        for TD effect, default 0
%             'tradval': possible number of TD variables (0 is also a value),
%                        default [1 6]
%               'leapy': = 0, no leap year effect, = 1 LP effect, = -1, test
%                        for LP effect, default 0
%               'easte': = 0 no Easter effect, = 1 Easter effect, = -1, test
%                        for Easter effect, default 0
%              'durval': possible days previous to Easter (0 is also a value)
%                        default [4 6]
%               'sname': character array containing the series name
%                        default mseries
%
%
%        OUTPUT : a structure, the output of function arimaestni
%------------------
%
%      Examples:
%
%    [out,ser]=arimaeasy(y,freq,'autmid',1,'out',1)
%    out=arimaeasy(y,freq,'[p dr q]',[0 1 1],'leapy',-1)
```

8 arimaestni

```
function outa = arimaestni(dbname,ser,fidr,ii)
%
% function to identify, estimate and forecast an ARIMA model for one
% series. The series may have up to two seasonalities. The ARIMA model
% is of the form:
%
% phi(B)*phi_s(B^s)*phi_S(B^S)*(delta*delta_s*delta_S*y_t -mu) =
% th(B)*th_s(B^s)*th_S(B^S)*a_t
%
% In the subdirectory spec, there is a specification file where all the
% options for the ARIMA model are defined and returned in the structue
% ser. The name of this file is given in function arimaestos and passed
% to this function.
% These options include, log transformation criteria, automatic
% identification of ARMA model and differencing operators, automatic
% specification of trading day, Easter effect and leap year (for
% quarterly and monthly series only), outlier search and forecasting,
% among other things.
% No automatic model identification is performed for the second
% seasonality (S). This part must be entered by the user. Automatic
% model identification is performed for the regular and the first
% seasonal part. Output is written in an external file in the
% subdirectory results.
%
%     INPUTS:
%     dbname : name of the series
%     ser    : a structure, containing the instructions for this
%              function
%     fidr   : an integer, corresponding to the output file
%     ii     : an integer, corresponding to the series currently
%              handled.
%
%  OUTPUTS:
%      outa  : a structure containing model information for the input
%              with fields:
%       title: a string with the name of series
%      nziyip: a 1 x 3 array with number of obs., initial year, initial
%              per.
%        freq: number of seasons
%        orig: original series
%       model: structre with ARIMA model information. In the case of a
%              transfer function, it is the ARIMA model corresponding
%              to the finite linear approximation to the input filters.
%              It has the following fields:
%          lam: = 0, logs are taken, = 1, no logs
%         mean: = 1, a mean is added, = 0, no mean
%            p: degree of regular AR polynomial
%            d: degree of regular differencing
%            q: degree of regular MA polynomial
%           ps: degree of seasonal AR polynomial
%           ds: degree of seasonal differencing
%           qs: degree of seasonal MA polynomial
%         nreg: the number of regression variables
%       result: a structure containing estimation results
%          phi: an array containing the regular AR polynomial
%               coefficients
%         phis: an array containing the seasonal AR polynomial
%               coefficients
%           th: an array containing the regular MA polynomial
%               coefficients
%          ths: an array containing the seasonal MA polynomial
%               coefficients
%        nrout: number of outliers
%            C: critical value for outlier detection
%         nind: observation numbers of the outliers
%          tip: string containing the outlier types
%       matsis: a structure containing the state space form of the model
%       resinf: a structure containing information about the residuals
%           hb: array containing the regression estimates
%           Mb: matrix containing the covariance matrix of the
%               regression estimates
%            Y: matrix containing the total regression effects
%          seb: array containing the regression standard errors
%           tb: array containing the t-values of the regression
%               estimates
%          Yrg: array containing the regression variables that are not
%               outliers
%        Youtg: array containing the outlier variables
%           se: array containing the standard errors of the estimates
%           tt: array containing the t-values of the estimates
%          npr: number of forecasts
%          pry: array containing the forecasts (transformed scale)
%         spry: array containing the standard errors of the forecasts
%         opry: same as pry but in the original scale
%        ospry: same os spry but in the original scale
```

9 arimaestos

```
function outa = arimaestos(fname,fmeta)
%
% Function for automatic identification, estimation and forecasting of
% ARIMA or transfer function models for one or several series
%
%     INPUTS:
%      fname : If fmeta = 0 or absent, a string such that fname.m is a
%              matlab function in the spec subdirectory that returns
%              the structure ser. In this structure, instruction for
%              this function are given. If fmeta = 1, a string such
%              that fname.txt contains a list of names of matlab
%              functions in the spec subdirectory that will be treated
%              sequentially.
%      fmeta : = 0, fname.m is a matlab function in the spec
%              subdirectory that returns the structure ser; = 1,
%              fname.txt is a file that contains a list of matlab
%              functions in the spec subdirectory that will be treated
%              sequentially. If not input, the program sets by default
%              fmeta = 0,
%
%  OUTPUTS:
%      outa  : a structure containing model information for the input
%              with fields:
%       title: a string with the name of series
%      nziyip: a 1 x 3 array with number of obs., initial year, initial
%              per.
%        freq: number of seasons
%        orig: original series
%       model: structre with ARIMA model information. In the case of a
%              transfer function, it is the ARIMA model corresponding
%              to the finite linear approximation to the input filters.
%              It has the following fields:
%          lam: = 0, logs are taken, = 1, no logs
%         mean: = 1, a mean is added, = 0, no mean
%            p: degree of regular AR polynomial
%            d: degree of regular differencing
%            q: degree of regular MA polynomial
%           ps: degree of seasonal AR polynomial
%            ds: degree of seasonal differencing
%            qs: degree of seasonal MA polynomial
%          nreg: the number of regression variables
%        result: a structure containing estimation results
%           phi: an array containing the regular AR polynomial
%                coefficients
%          phis: an array containing the seasonal AR polynomial
%                coefficients
%            th: an array containing the regular MA polynomial
%                coefficients
%           ths: an array containing the seasonal MA polynomial
%                coefficients
%         nrout: number of outliers
%             C: critical value for outlier detection
%          nind: observation numbers of the outliers
%           tip: string containing the outlier types
%        matsis: a structure containing the state space form of the model
%        resinf: a structure containing information about the residuals
%            hb: array containing the regression estimates
%            Mb: matrix containing the covariance matrix of the
%                regression estimates
%             Y: matrix containing the total regression effects
%           seb: array containing the regression standard errors
%            tb: array containing the t-values of the regression
%                estimates
%           Yrg: array containing the regression variables that are not
%                outliers
%         Youtg: array containing the outlier variables
%            se: array containing the standard errors of the estimates
%            tt: array containing the t-values of the estimates
%           npr: number of forecasts
%           pry: array containing the forecasts (transformed scale)
%          spry: array containing the standard errors of the forecasts
%          opry: same as pry but in the original scale
%         ospry: same os spry but in the original scale
%      tfmodel: structure with transfer function model information.
%        matsis: a structure containing the state space form of the model
%                correponding to the filtered inputs
%        result: a structure containing estimation results
%          nreg: the number of regression variables
%           Yrg: array containing the regression variables that are not
%                outliers
%         Youtg: array containing the outlier variables
%          yci: output corrected by filtered inputs
%        tford: a three column array in which the i-th row has three
%               numbers corresponding to the delay, the numerator
%               degree and the denominator degree of the i-th input
%               filter
%          phi: an array containing the regular AR polynomial
%               coefficients
%         phis: an array containing the seasonal AR polynomial
%                coefficients
%           th: an array containing the regular MA polynomial
%               coefficients
%          ths: an array containing the seasonal MA polynomial
%               coefficients
%          omg: a cell array containing the numerators of the input
%               filters
%          del: a cell array containing the denominators of the input
%               filters
%       resinf: a structure containing information about the residuals
%           se: array containing the standard errors of the estimates
%           tt: array containing the t-values of the estimates
%          npr: number of forecasts
%         dpry: array containing the forecasts of the output corrected
%               by the filtered inputs
%        dspry: array containing the standard errors of the forecasts of
%               the output corrected by the filtered inputs
%          Yin: array containing the input variables
%      modpred: a multiple structure containing the input forecasts if
%               any. The forecasts for each input are given in field
%               .pred
%     modinput: a multiple structure containing the models for the
%               inputsif any (.mod = 0, no model; .mod =1, there is
%               model). The model for each input has fields .alpha,
%               .phi, .theta, .sigma2
%            y: output series in the transformed scale
%          pry: array containing the forecasts (transformed scale)
%         spry: array containing the standard errors of the forecasts
%         opry: same as pry but in the original scale
%        ospry: same os spry but in the original scale
```

10 arimaestwi

```
function outa = arimaestwi(dbname,ser,fidr,ii)
```

```
%
% function to identify, estimate and forecast a transfer function model
% for one series. The method used for automatic model identificaiton is
% described in Gomez (2009), "Transfer Function Model Identification",
% Boletin de Estadistica e Investigacion Operativa, 25, pp. 109-115.
%
%
%     INPUTS:
%     dbname : name of the series
%     ser    : a structure, containing the instructions for this
%              function
%     fidr   : an integer, corresponding to the output file
%     ii     : an integer, corresponding to the series currently
%              handled.
%
%  OUTPUTS:
%      outa  : a structure containing model information for the input
%              with fields:
%       title: a string with the name of series
%      nziyip: a 1 x 3 array with number of obs., initial year, initial
%              per.
%        freq: number of seasons
%        orig: original series
%       model: structre with ARIMA model information. In the case of a
%              transfer function, it is the ARIMA model corresponding
%              to the finite linear approximation to the input filters.
%          lam: = 0, logs are taken, = 1, no logs
%         mean: = 1, a mean is added, = 0, no mean
%            p: degree of regular AR polynomial
%            d: degree of regular differencing
%            q: degree of regular MA polynomial
%           ps: degree of seasonal AR polynomial
%           ds: degree of seasonal differencing
%           qs: degree of seasonal MA polynomial
%         nreg: the number of regression variables
%       result: a structure containing estimation results
%          phi: an array containing the regular AR polynomial
%               coefficients
%         phis: an array containing the seasonal AR polynomial
%               coefficients
%           th: an array containing the regular MA polynomial
%               coefficients
%          ths: an array containing the seasonal MA polynomial
%               coefficients
%         nrout: number of outliers
%             C: critical value for outlier detection
%          nind: observation numbers of the outliers
%           tip: string containing the outlier types
%        matsis: a structure containing the state space form of the model
%        resinf: a structure containing information about the residuals
%            hb: array containing the regression estimates
%            Mb: matrix containing the covariance matrix of the
%               regression estimates
%             Y: matrix containing the total regression effects
%           seb: array containing the regression standard errors
%            tb: array containing the t-values of the regression
%               estimates
%           Yrg: array containing the regression variables that are not
%               outliers
%         Youtg: array containing the outlier variables
%            se: array containing the standard errors of the estimates
%            tt: array containing the t-values of the estimates
%           npr: number of forecasts
%           pry: array containing the forecasts (transformed scale)
%          spry: array containing the standard errors of the forecasts
%          opry: same as pry but in the original scale
%         ospry: same os spry but in the original scale
%      tfmodel: structure with transfer function model information.
%        matsis: a structure containing the state space form of the model
%               correponding to the filtered inputs
%        result: a structure containing estimation results
%          nreg: the number of regression variables
%           Yrg: array containing the regression variables that are not
%               outliers
%         Youtg: array containing the outlier variables
%           yci: output corrected by filtered inputs
%         tford: a three column array in which the i-th row has three
%               numbers corresponding to the delay, the numerator
%               degree and the denominator degree of the i-th input
%               filter
%           phi: an array containing the regular AR polynomial
%               coefficients
%          phis: an array containing the seasonal AR polynomial
%               coefficients
%            th: an array containing the regular MA polynomial
%               coefficients
%          ths: an array containing the seasonal MA polynomial
%               coefficients
%          omg: a cell array containing the numerators of the input
%               filters
%          del: a cell array containing the denominators of the input
%               filters
%       resinf: a structure containing information about the residuals
%           se: array containing the standard errors of the estimates
%           tt: array containing the t-values of the estimates
%          npr: number of forecasts
%         dpry: array containing the forecasts of the output corrected
%               by the filtered inputs
%        dspry: array containing the standard errors of the forecasts of
%               the output corrected by the filtered inputs
%          Yin: array containing the input variables
%      modpred: a multiple structure containing the input forecasts if
%               any. The forecasts for each input are given in field
%               .pred
%     modinput: a multiple structure containing the models for the
%               inputs if any (.mod = 0, no model; .mod =1, there is
%               model). The model for each input has fields .alpha,
%               .phi, .theta, .sigma2
%            y: output series in the transformed scale
%          pry: array containing the forecasts (transformed scale)
%         spry: array containing the standard errors of the forecasts
%         opry: same as pry but in the original scale
%        ospry: same os spry but in the original scale
```

11 arimam

```
% Kalman filter given the ARIMA polynomials. The state space model is
%
%   x_{t} = T x_{t-1} + H a_{t}
%   y_{t} = Z x_{t},
%
% where the initial state vector is
%
%       x_{d+1} = A*\delta + \Xi*c,
%
% and Var(c) = Sigma. See Gomez and Maravall (1994), "Estimation,
% Prediction and Interpolation for Nonstationary Series with the
% Kalman Filter", Journal of the American Statistical Association,
% 89, 611-624. The filter is initialized at time t = d+1, where d is
% the differencing degree, and the first d observations are stacked
% to form the \delta vector.
%
%  INPUTS:
%     phi   : an array containing the AR polynomial
%     alpha : an array containing the differencing polynomial
%     th    : an array containing the MA polynomial
%
%  OUTPUTS:
%        Z: the Z matrix
%        T: the T matrix
%        H: the H matrix
%        A: the A matrix
%    Sigma: the Sigma matrix
%       Xi: the Xi matrix
```

12 arimaopt

```
function   [x,J]=arimaopt(fmarqdt,fid,x0,xv,xf,y,Y,parm,infm,pr)
%
% This function performs the optimization for an ARIMA model
%
%     INPUTS:
%     fmarqdt : = 1 estimation with lsqnonlin (matlab), = 0, estimation
%               with marqdt
%     fid     : an integer, corresponding to an external file
%     x0      : an array containing the initial estimated values
%     xv      : an array containing the variable parameters
%     xf      : an array containing the fixed parameters
%     y       : input series
%     Y       : matrix of regression variables
%     parm: a structure containing model information, where
%       .s:  seasonality
%       .S:  second seasonality
%       .p:  AR order
%      .ps: order of the AR of order s
%       .q:  order of the regular MA
%      .qs: order of the MA of order s (1 at most)
%      .qS: order of the MA of order S (1 at most)
%      .dr: order of regular differencing
%      .ds: order of differencing of order s
%      .dS: order of differencing of order S
%    .pvar:  array containing the indices of variable parameters
%    .pfix:  array containing the indices of fixed parameters
%  .ninput: number of inputs
%  .inputv: array containing the lagged input variables corresponding to
%           the polynomial approximations to the rational input filters
%   .delay: array with the delays of the input filters
%      .ma: array with the ma parameters of the input filters
%      .ar: array with the ar parameters of the input filters
%     .npr: number of forecasts
%   infm      : structure containing function names and optimization options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 x is passed from marqdt to f but not passed from f to marqdt
%           =0 x is passed from marqdt to f and passed from f to marqdt
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian and gradient at the solution is performed
%           =0 no evaluation of jacobian and gradient at the solution is performed
% .maxit:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
%   .chb:   = 1  compute the beta estimate and its MSE
%             0  do not compute the beta estimate and its MSE
%   .inc:   = 0, the initial states in the filter equations to obtain the
%                filtered variables are equal to zero (not estimated)
%           = 1, the initial states in the filter equations are estimated
%     pr      : = 1, print results in an external file, = 0, do not print
%
%  OUTPUTS:
%      x     : an array containing the estimated parameter values
%      J     : a matrix containing the Jacobian at the solution
```

13 arimapol

```
function [phirs,alprsS,thrsS]=arimapol(x,s,S,p,ps,dr,ds,dS,q,qs,qS)
%
% this function computes the different polynomials for an ARIMA model
%
%    INPUTS:
%          x:  an array containing the ARIMA parameter values
%          s:  seasonality
%           S:  second seasonality
%           p:  AR order
%          ps:  order of the AR of order s
%           q:  order of the regular MA
%          qs:  order of the MA of order s (1 at most)
%          qS:  order of the MA of order S (1 at most)
%          dr: order of regular differencing
%          ds: order of differencing of order s
%          dS: order of differencing of order S
%  OUTPUTS:
%     phirs  : an array containing the AR polynomial
%     alprsS : an array containing the differencing polynomial
%     thrsS  : an array containing the MA polynomial
```

14 arimasigex

```
function outa = arimasigex(out,Ycomp)
%
% function to perform the canonical decomposition of an ARIMA model
% previously identified with function arimaestos.
%
% phi(B)*phi_s(B^s)*(delta*delta_s*y_t -mu) =
% th(B)*th_s(B^s)*a_t
%
%     INPUTS:
%     out    : a structure, output of function arimaestos
%     Ycomp  : a cell array, containing the assignment of each
%              regression variable to a component. Possible values are
%              'trend', 'seas', 'tran' and 'irreg'.
%
%  OUTPUTS:
%      outa  : a structure containing model information for the input
%              with fields:
%       title: a string with the name of series
%        orig: original series
%     Ycomp  : same as input
%         gft: flag for graphs, = 0, no graphs, =1 graphs
%         lam: flag for logs, = 0 logs, = 1, no logs
%         npr: number of forecasts
%       sconp: standard deviation of the innovations
%        conp: innovation variance
%        orig: original series
%            Y: array containing the regression variables
%      bg_year: initial year
%       bg_per: initial period
%        datei: date structure
%          str: structure that contains the ARIMA model information. It
%               is the output of function suvarmapqPQ.
%       compcd: structure that is output of function candec
%         strc: structure that contains information about the canonical
%               decomposition model in state space form. It is the
%               output of function sucdm.
%      compmat: structure containing the state space model matrices and
%               initial conditions for the state space model in which
%               the trend-cycle has been decomposed into a smooth trend
%               and a cycle
%       StochCc: matrix containing the stochastic components
%      StochSCc: matrix containing the mse of the stochastic components
%      oStochCc: matrix containing the stochastic components in the
%                original scale
%     oStochSCc: matrix containing the mse of the stochastic components
%                in the original scale
%            Cc: matrix containing the components including
%                deterministic effects
%           SCc: matrix containing the mse of Cc
%           oCc: matrix containing the Cc in the original scale
%          oSCc: matrix containing the mse of the oCc
```

15 arimasigextc

```
function outb = arimasigextc(outa,comp,filter)
%
% function to perform the decomposition of the trend-cycle component of
% a canonical decomposition of an ARIMA model previously identified
% with function arimaestos.
%
% phi(B)*phi_s(B^s)*(delta*delta_s*y_t -mu) =
% th(B)*th_s(B^s)*a_t
%
% the decomposition of the trend-cycle, p_t, is of the form p_t = sp_t +
% c_t, where sp_t is a (smooth) trend and c_t is a (smooth) cycle.
%
%     INPUTS:
%     out    : a structure, output of function arimaestos
%     Ycomp  : a cell array, containing the assignment of each
%              regression variable to a component. Possible values are
%              'trend', 'seas','tran' and 'irreg'.
%     filter : flag for the filter to be applied to the trend-cycle
%              component of the canonical decomposition. Possible values
%              are 'lp' (low-pass) and 'bp' (band-pass).
%
%  OUTPUTS:
%      outb  : a structure containing model information for the input
%              with fields:
%           strc: structure that contains information about the model
%                 y_t = Y_t*beta + sp_t + c_t + s_t + r_t + i_t in
%                 Akaike state space form, where sp_t is the (smooth)
%                 trend and c_t is the (smooth) cycle,  both obtained
%                 from the previous p_t by application of the low pass
%                 or band pass filter. It is the output of function
%                 sucdmpbst (low pass) or sucdmpbp (band pass)
%      StochCctc: matrix containing the stochastic components
%     StochSCctc: matrix containing the mse of the stochastic
%                 components
%     oStochCctc: matrix containing the stochastic components in the
%                 original scale
%    oStochSCctc: matrix containing the mse of the stochastic
%                 components in the original scale
%           Cctc: matrix containing the components including
%                 deterministic effects
%          SCctc: matrix containing the mse of Cctc
%          oCctc: matrix containing the Cctc in the original scale
%         oSCctc: matrix containing the mse of the oCctc
```

16 arimasimeasy

```
function Y = arimasimeasy(freq,varargin)
%**********************************************************************
%                       EASY ARIMA SIMULATION
%
%                            USAGE :
% Y = arimasimeasy(freq,'option1',optionvalue1,'option2',optionvalue2,...)
%
%       INPUTS :
%------------------
%      REQUIRED
%        freq : data frequency (number of observations per year)
%-----------------
%      OPTIONS
%             'phir': (1 x p) array containing the regular AR polynomial
%                     in MATLAB format, for example, [-.4 1] for phir = 1.
%                     - .4B, default 1.
%             'phis': (1 x ps) array containing the seasonal AR polynomial
%                     in MATLAB format, default 1.
%              'thr': (1 x q) array containing the regular MA polynomial
%                     in MATLAB format, default 1.
%              'ths': (1 x q) array containing the seasonal MA polynomial
%                     in MATLAB format, default 1.
%         '[p dr q]': (1 x 3) array containing the regular orders
%                     default: [0 0 0]
%       '[ps ds qs]': (1 x 3) array containing the first seasonal orders
%                     default: [0 0 0]
%                'N': length of the series to be generated, default 100
%               'Ns': number of series to be generated, default 1
%          'discard': number of initial observations to be discarded when
%                     simulating the series, default 50
%              'gft': flag for graphics, = 1, plot series, = 0, no plots
%                      = 2, plots are saved but not displayed, = 3, plots
%                      are both saved and displayed, default 0
%              'drg': regular differencing to be applied to the simulated
%                     series when generating graphs, default 0
%              'dsg': seasonal differencing to be applied to the simulated
%                     series when generating graphs, default 0
%             'mean': mean value for the differenced series, ~=0 mean, =0,
%                      no mean, default 0
%             'stda': standard deviation of the simulated series, default
%                     1.
%             'seed': seed used for the simulation, default 20
%              'lag': number of lags for autocorrelations, default 3*freq
%               'cw': confidence bands coefficient, default 1.96
%
%      OUTPUT : Y   : the simulated (N x Ns) series array
%-----------------
%
%    Examples:
%
%   Y=arimasimeasy(freq,'mean',.5)
%   Y=arimasimeasy(freq,'[p dr q]',[0 1 1],'thr',[-.4 1.],'gft',2)
```

17 arimasplot

```
function arimasplot(Y,dr,dru,s,dsu,lag,rp,pcp,cw)
%*********************************************************************
% Auxiliary function called in arimasimul_d.m to plot the different series
%
%  INPUTS:
%        y : array containing the simulated ARIMA series
%       dr : number of regular differences
%      dru : number of regular differences entered by the user
%        s : seasonal frequency
%      dsu : number of seasonal differences entered by the user
%      lag : number of lags for the autocorrelations and partial autocor.
%       rp : theoretical autocorrelations
%      pcp : theoretical partial autocorrelations
%       cw : confidence interval parameter
```

18 armafil

```
function [z,rx1] = armafil(y,omega,delta,b,inc)
%
% This function filters the series y_t  using the filter nu(z) =
% z^b*omega(z)/delta(z), where omega(z)=omega_0 + omega_1*z +
% omega_2*z^2 + ....+omega_q*z^q and delta(z) = 1 + delta_1*z + ... +
% delta_p*z^p.
%
% Input arguments:
% y: the input series
% omega: a (q+1) array containing the coefficients of omega(z) in
%        ascending order
% delta: a (p+1) array containing the coefficients of delta(z) in
%        ascending order
% b: the delay of the filter
% inc: = 1 initial conditions for the filter are estimated
%      = 0 initial conditions equal to zero
%
% Output arguments:
%   z: the output series
% rx1: the design matrix for the initial state x_1
```

19 armaid

```
function oparm=armaid(y,parm,ols,a,maxpq,maxPQ)
%
% this function automatically identifies an ARMA model for a stationary
% series using the BIC criterion.
%
% Input arguments:
% y: vector containing the data
% parm: astructure containing model information, where
%   s:  seasonality
%   S:  second seasonality
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .p:  initial AR order
% .ps: initial order of the AR of order s
% .q:  initial order of the regular MA
% .qs: initial order of the MA of order s (1 at most)
% .qS: initial order of the MA of order S (1 at most)
% ols: = 0, use the Levinson-Durbin algorithm in the Hannan-Rissanen
%        method
%      = 1, use OLS in  the Hannan-Rissanen method
%   a: the exponent in log(n)^a for the length of the long AR in the
%      Hannan-Rissanen method. By default, a = 1.5.
% maxpq: the maximum orders of the regualr AR and MA polynomials
% maxpq: the maximum orders of the seasonal AR and MA polynomials
%
% Output arguments:
% oparm: a structure containing the same fields as par plus
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
```

20 armaxe2armax

```
function [phi,theta,gamma,ierror] = armaxe2armax(phie,thetae,gammae)
%
% This function computes VARMAX polynomials with \Phi_0=I_s=\Theta_0.
```

```
%---------------------------------------------------
% USAGE: [phi,theta,gamma] = armaxe2armax(phie,thetae,gammae)
% where:    phie   = a k x k polynomial matrix with phi(0) nonsingular
%           thetae = a k x k polynomial matrix
%          gammae  = a k x m polynomial matrix
%---------------------------------------------------
% RETURNS:
%           phi  = the AR polynomial matrix
%          theta = the MA polynomial matrix
%          gamma = the input polynomial matrix
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
```

21 armaxe2sse

```
function str = armaxe2sse(str)
% PURPOSE: given a VARMAX model in echelon form, it computes the state
% space echelon form
%---------------------------------------------------
% USAGE: str = armaxe2sse(str)
% where:    str    = a structure containing the structure of the VARMAX
%                    in echelon form
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the state space echelon form matrices according to the
%                following state space model:
%
%  alpha_{t+1} = Fs*alpha_{t} + Bs*x_t{t} + Ks*a_{t}
%      y_{t}   = Hs*alpha_{t} + Ds*x_{t}  + a_{t}
%---------------------------------------------------
```

22 arpar

```
function y=arpar(x,p,ps,q,qs,qS)
%*********************************************************************
%
% Given the polynomials of a multiplicative ARMA model, this function
% transforms the coefficients of each polynomial as if they were AR
% coefficients into partial correlation coefficients. This may be used
% as a test for stationarity because a polynomial is stable if, and
% only if, all its partial coefficients are less than one in absolute
% value.
%
%  INPUTS:
%      x : coefficients of the polynomials of a multiplicative ARMA
%          model
%      p, ps, q, qs, qS : integers specifying where the coefficients of
%      the ARMA model are in x.
%      More specifically,
%      p : first p are AR coefficients
%     ps : starting with the (p+1)th coefficient, the next ps are AR c.
%      q : starting with the (p+1+ps+1)th coefficient, the next q are
%          MA c.
%     qs : starting with the (p+1+ps+1+q+1)th coefficient,
%          the next qs are MA c.
%     qS : starting with the (p+1+ps+1+q+1+qs+1)th coefficient,
%          the next qS are MA c.
%
% OUTPUTS:
%      y : partial correlation coefficients of all the polynomials of
%          the ARMA model
```

23 aurirvarmapqPQ

```
function [str,ferror] = aurirvarmapqPQ(str,nr,DA)
% PURPOSE: given a structure containing information about a VARMA
% model, it adds the unit roots information given by nr and DA
%---------------------------------------------------
% USAGE: str = fixvarmaxpqPQ(str)
% where:str = a structure created with function suvarmapqPQ
%        nr = number of unit roots in the model
%        DA = matrix of the form [DAr Indxr], where DAr is the
%             parameterization of betaor (the unit root part), and
%             Indxr is an index vector to identify the l.i. rows of DAr.
%---------------------------------------------------
% RETURNS: str = a structure containing model information
%          where the following fields have been added:
%    .nr  : number of unit roots in the model
%    .ns  : number of seasonal unit roots in the model (not used)
%    .xd  : parameter vector for betaor, including fixed and variable
%           parameters
%  .nparmd: number of parameters for the unit root part
%    .xid : array of ones and zeros, as in xi, for the unit root part
%    .xvd : array of parameters to estimate for the unit root part
%    .xfd : array of fixed parameters for the unit root part
%---------------------------------------------------
```

24 aurivarmapqPQ

```
function [str,ferror] = aurivarmapqPQ(str,nr,ns,DA)
% PURPOSE: given a structure containing information about a VARMA
% model, it adds the unit roots information given by nr, ns and DA (
% the output from mcrcreg)
%---------------------------------------------------
% USAGE: str = fixvarmaxpqPQ(str)
% where:   str      = a structure created with function suvarmapqPQ
%---------------------------------------------------
% RETURNS: str = a structure containing model information
%---------------------------------------------------
```

25 autcov

```
function [c0,cv,r]=autcov(y,lag,ic)
%*********************************************************************
%
% This function computes the sample autocovariances and
% autocorrelations of y(t) up to specified lag. The variable has been
% previously demeaned.
%
%   INPUTS:
%       y : input vector
%     lag : integer specifying up to which lag cv and/or r are computed
%      ic = 1: compute autocorrelations
%           0: do not compute autocorrelations
%
%  OUTPUTS:
%      c0 : variance of y
%      cv : autocovariances of y
%       r : autocorrelations of y
```

26 beta_cdf

```
function cdf = beta_cdf(x, a, b)
% PURPOSE: cdf of the beta distribution
```

```
%---------------------------------------------------------------
% USAGE: cdf = beta_cdf(x,a,b)
% where:   x = prob[beta(a,b) <= x], x = vector
%          a = beta distribution parameter, a = scalar
%          b = beta distribution parameter  b = scalar
% NOTE: mean [beta(a,b)], variance = ab/((a+b)*(a+b)*(a+b+1))
%---------------------------------------------------------------
% RETURNS: cdf at each element of x of the beta distribution
%---------------------------------------------------------------
% SEE ALSO: beta_d, beta_pdf, beta_inv, beta_rnd
%---------------------------------------------------------------
% written by:  Anders Holtsberg, 18-11-93
%              Copyright (c) Anders Holtsberg
% documentation modified by LeSage to
% match the format of the econometrics toolbox
```

27 blacktu

```
function [w, m] = blacktu(n, width, a)
%
%        This function computes the weights for the
%        Blackman-Tukey window
%
%     INPUTS:
%         n : lentgh of the series; required input to compute window lag
%             size if m is not input to blacktu
%     width : window width factor
%      wina : "a" parameter for Blackman-Tukey window (0.23 by default)
%             if wina <= 0, it is set to 0.23
%
%    OUTPUTS:
%         w : weights of the Blackman-Tukey window
%         m : window lag size;
```

28 bmols

```
function [beta,M,e]=bmols(y,Y)
%*********************************************************************
%
% This function computes the OLS estimator, its covariance matrix and
% the white noise residuals. The covariance matrix is not multiplied by
% sigma^2.
```

```
%
%   INPUTS:
%       y : data vector
%       Y : matrix with regression variables
%
%  OUTPUTS:
%    beta : OLS estimator
%       M : covariance matrix of beta
%       e : residuals
```

29 bols

```
function beta=bols(y,Y)
%**********************************************************************
%  This function computes the OLS estimator
%
%   INPUTS:
%       y : data vector
%       Y : matrix with regression variables
%
%  OUTPUTS:
%    beta : OLS estimator
```

30 btval

```
function [beta,tv,d]=btval(x,ydf)
%*********************************************************************
%  This function computes the OLS estimator, the t-values and the
% standard errors
%
%   INPUTS:
%       x : vector with parameters
%     ydf : matrix with the data and regression variables
%
%  OUTPUTS:
%    beta : OLS estimator
%      tv : t-value of beta
%       d : standard error of beta
```

31 cal

```
function result = cal(begin_yr,begin_per,freq,obs)
% PURPOSE: create a time-series calendar structure variable that
%          associates a date with an observation #
% -----------------------------------------------------
% USAGE:       result = cal(begin_yr,begin_per,freq,obs)
%         or:  result = cal(cstruc,obs)
% where:    begin_yr  = beginning year, e.g., 1982
%           begin_per = beginning period, e.g., 3
%               freq  = frequency, 1=annual,4=quarterly,12=monthly
%               obs   = optional argument for an observation #
%           cstruc    = a structure returned by cal()
% -----------------------------------------------------
% RETURNS: a structure:
%             result.beg_yr  = begin_yr
%             result.beg_per = begin_period
%             result.freq    = frequency
%             result.obs     = obs            (if input)
%             result.year    = year for obs   (if input)
%             result.period  = period for obs (if input)
% ------------------------------------------------------
% SEE ALSO: ical() an inverse function to find observation #
%           associated with a cal-structure date
%           tsdate() that returns a string for the date associated
%           with observation #
% written by:
% James P. LeSage, Dept of Economics
% University of Toledo
% 2801 W. Bancroft St,
% Toledo, OH 43606
% jpl@jpl.econ.utoledo.edu
```

32 candec

```
function [comp,ierrcandec] = candec(phir,phis,thr,ths,phirst,s,dr,ds,sconp)
%*********************************************************************
% PURPOSE: This function performs the canonical decomposition of the ARIMA
%          model
%          phir(B)*phis(B^s)y_t = thr(B)*ths(B^s)*a_t
%The innovation variance, sigma2a, is assumed to be unity.
%---------------------------------------------------
```

```
% USAGE: [comp,ierrcandec] = candec(phir,phis,thr,ths,phirst,s,dr,ds,sconp)
%
% Inputs: phir : a polynomial containing the regular AR part
%         phis : a polynomial containing the seasonal AR part
%         thr  : a polynomial containing the regular MA part
%         ths  : a polynomial containing the seasonal MA part
%       phirst : a polynomial containing the stationary regular AR part
%            s : a positive integer, the number of seasons
%           dr : a positive integer, the number of regular differences
%           dr : a positive integer, the number of seasonal differences
%        sconp : a positive number, standard deviation of the series model
%                innovations
% Note: all of the previous polynomials are expressed in the same variable.
% The polynomials are given by an array like [ a_n, ... a_1, a_0], where
% the polynomial is a_0 + a_1*z + ... + a_n*z^n.
%
%  Output: comp, a structure containing the following fields
%   .ptnum=thrc;     % trend-cycle numerator
%   .ptden=phir;     % trend-cycle denominator
%   .ptnur=dr+ds;    % number of nonstationary roots in phir
%   .ptvar=sigma2r;  % variance of the trend-cycle innovations (*)
%   .stnum=thsc;     % seasonal numerator
%   .stden=phis;     % seasonal denominator
%   .stnur=(s-1)*ds; % number of nonstationary roots in phis
%   .stvar=sigma2s;  % variance of the seasonal innovations (*)
%   .rt=thtc;        % transitory component (MA term)
%   .rtvar=sigma2t;  % variance of the transitory component innovations (*)
%   .itvar=sigma2i;  % variance of the irregular component (*)
%   .sigmaa=sconp;   % standard deviation of the series model innovations
%   .phi=phirst;     % stationary AR trend polynomial
%(*) in units of the series model innovations
%    ierrcandec : flag for errors
```

33 cascade

```
function [Tsp,Hsp,Zsp,ferror]=cascade(den,Alpha,phip,thp,sigma)
%
%        This function obtains a cascade implementation of the state
%        space form corresponding to the product of filters in the ARMA
%        model
%
%        y_t = [Alpha(z)/den(z)]*[thp(z)/phip(z)] a_t
```

```
%
%        where a_t is (0,sigma^2). In each factor the degree of the
%        numerator has to be equal to that of the denominator.
%        If z_t = [thp(z)/phip(z)] a_t has a state space form
%
%        x^z_{t+1} = T_z x^z_t + H_z*sigma*e_t                  (1)
%           z_t    = Z_z x^z_t + sigma*e_t,
%
%        where Var(e_t)=1,  and if y_t = [Alpha(z)/den(z)] z_t has a
%        state space form
%
%        x^y_{t} = T_y x^y_{t-1} + H_y z_t                      (2)
%           y_t  = Z_y x^y_t,
%
%        then the following state space form for y_t is a cascade
%        implementation
%
%        x_{t}   = [ T_y   H_y*Z_z]x_{t-1}  + [ H_y*sigma ]e_t
%                  [  0    T_z    ]           [ H_z*sigma ]
%          y_t   = [ Z_y     0    ]x_t ,
%
%        where x_t = [x^y_t'  x^z_{t+1}']'.
%        Here, T_z is a square matrix with dimension equal to the
%        degree of phip(z), T_y is a square matrix with dimension equal
%        to the degree of den(z) plus one, and the representations (1)
%        and (2) are Akaike's representations. Note that (1) is minimal
%        while (2) is  not.
%
%        Input parameters:
%        den    : a (1 x nbp) array
%        Alpha  : a (1 x nal) array
%        phip   : a (1 x np+1) array
%        thp    : a (1 x nt+1) array
%        sigma  : a positive constant
%
%        Output parameters:
%        Tsp    : an (nalpha x nalpha) matrix
%        Hsp    : an (nalpha x nepsilon) matrix
%        Zsp    : an (p x nalpha) matrix
```

34 cascadessm1

```
function [Fsp,Gsp,Hsp,Jsp,ferror]=cascadessm1(Fp,Gp,Hp,Fs,Gs,Hs,Js)
%
%        This function obtains a cascade implementation of the state space
%        form corresponding to the product of VARMA filters
%
%        y_t = As(z)*Ap(z) a_t
%
%        where a_t is (0,Sigma).
%        If z_t = Ap(z) a_t has a state space form
%
%        x^p_{t+1} = F_p x^p_t + G_p*a_t                   (1)
%           z_t    = H_p x^p_t + a_t,
%
%        and if y_t = As(z) z_t has a state space form
%
%        x^s_{t+1} = F_s x^s_t + G_s*z_t                       (2)
%           y_t    = H_s x^s_t + J_s*z_t,
%
%        then the following state space form for y_t is a cascade
%        implementation
%
%        x_{t+1} = [ F_s   G_s*H_p]x_t  + [ G_s ]a_t
%                  [  0      F_p  ]       [ G_p ]
%          y_t   = [ H_s   J_s*H_p]x_t  + [ J_s ]a_t,
%
%        where x_t = [x^s_t'  x^p_t']'.
%        The representations (1) and (2) are
%        Akaike's representations. Note that (1) and (2) are minimal.
%
%        Input parameters:
%        Fp     : an (np x np) matrix
%        Gp     : an (np x n) matrix
%        Hp     : an (n x np) matrix
%        Fs     : an (ns x ns) matrix
%        Gs     : an (ns x m) matrix
%        Hs     : an (m x ns) matrix
%        Js     : an (m x ns) matrix
%
%        OuFput parameters:
%        Fsp     : an (nalpha x nalpha) matrix, nalpha = np + ns.
%        Gsp     : an (nalpha x 1) matrix
%        Hsp    : an (m x nalpha) matrix
%        Jsp    : an (m x ns) matrix
```

35 cbic

```
function bic = cbic(x,yd,nd,s,p,ps,q,qs)
%
% this function computes the BIC criterion of an ARMA model
%
% Input arguments:
% x : array containing model parameters
% yd: vector containing the data
% nd: length of yd
% s:  number of seasons
% p:  AR order
% ps: order of the AR of order s
% q:  order of the regular MA
% qs: order of the MA of order s
%
% Output arguments:
% bic: the bic criterion
```

36 chkroots

```
function chk=chkroots(x,p,ps,q,qs,qS)
%*********************************************************************
% This function tests whether all roots of the polynomials of a
% multiplicative ARMA model are outside of the unit circle
%
%  INPUTS:
%      x : coefficients of the polynomials of a multiplicative ARMA
%          model
%      p, ps, q, qs, qS : integers specifying where the coefficients of
%      the ARMA model are in x.
%      More specifically,
%      p : first p are AR coefficients
%     ps : starting with the (p+1)th coefficient, the next ps are AR c.
%      q : starting with the (p+1+ps+1)th coefficient, the next q are
%          MA c.
%     qs : starting with the (p+1+ps+1+q+1)th coefficient,
%          the next qs are MA c.
%     qS : starting with the (p+1+ps+1+q+1+qs+1)th coefficient,
%          the next qS are MA c.
%
%  OUTPUT:
%    chk = 0 : roots are outside the unit circle
%        = 1 : roots are not outside the unit circle
```

37 chkstainv

```
function ierror=chkstainv(Fs)
% new function: 21-1-2011
% This function checks wether the matrix Fs has eigenvalues with modulus
% greater than or equal to one.
```

38 chmarima

```
function [y,Xm,nmiss,idnx]=chmarima(y)
%
%    this function checks whether there are missing values in the series
%
%        INPUTS:
%        y: an array containing the input series
%
%        OUTPUTS:
%        y: an array containing the input series with the missing values
%           replaced with tentative values
%       Xm: a regression matrix whose columns have zeros except for the
%           observation numbers of the missing values in which it has
%           ones
%    nmiss: number of missing observations
%     idxn: index for missing values
```

39 cinest

```
function x0 = cinest(y,Y,parm,est,ols,a,prt,fid)
%
% function to estimate initial parameter values in an ARIMA model
%
%
%     INPUTS:
%     y       : input series
%     Y       : matrix of regression variables
%     parm    : a structure, containing the ARIMA specification. It
%               should have al least the following fields:
%          .s : seasonality
%          .S : second seasonality
%          .p : degree of regular AR polynomial
%          .d : degree of regular differencing
%          .q : degree of regular MA polynomial
%          .ps: degree of seasonal AR polynomial
%          .ds: degree of seasonal differencing
%          .qs: degree of seasonal MA polynomial
%          .dS: degree of second seasonal differencing
%          .qS: degree of second seasonal MA polynomial
%     est = 1 : estimation of regression coefficients
%        = 0  : no estimation of regression coefficients
%        ols  : = 1, perform OLS, = 0, use the Durbin Levinson
%               algorithm in the HR method
%     a       : an integer, the degree of the AR approximation in the
%               first step of the Hanna-Rissanen method.
%     prt     : = 1, print results in an external file, = 0, do not
%               print
%     fid     : an integer, corresponding to an external file
%
%  OUTPUTS:
%      x0     : an array containg the initial estimated values
```

40 cleanpmat

```
function [M, gM] = cleanpmat(M, tol)
%*********************************************************************
% This function cleans a polynomial matrix of small entries (in
% relation to its L1 norm) and if neccesary reduces its order
% afterwards
%
%  INPUTS:
%      M : (n x m x p) polynomial matrix
%    tol : tolerance value
%
% OUTPUTS:
%      M : (n x m x (gM+1)) polynomial matrix after elimination of
%          small entries
%     gM : final order of the polynomial matrix
%*********************************************************************
```

41 cleanpol

```
function p = cleanpol(p, tol)
% This function cleans a polynomial of small entries (in relation to its
% L1 norm) and if neccesary reduces its order afterwards
%  INPUTS:
%      p : (n x 1) polynomial coefficients
%    tol : tolerance value
%
% OUTPUTS:
%      p : polynomial after elimination of small entries
```

42 cohepha

```
function [co,ph,ga]=cohepha(cxy,qxy,fxx,fyy)
%
%        This function computes the coherence, gain and phase
%        (see Granger's book)
%
%     INPUTS:
%------------
%       cxy : cospectrum
%       qxy : quadrature spectrum
%       fxx : (smoothed) periodogram of x
%       fyy : (smoothed) periodogram of y
%
%
%    OUTPUTS:
%------------
%    co     : coherence
%    ph     : phase angle
%    ga     : gain
```

43 coincid

```
function  ncoin=coincid(nind,eind)
%
% this function detects the number of coincidences between the
% present and the past search for outliers
%
% Input arguments:
% nind: array containing the time index of the present outliers
% eind: array containing the time index of the previous outliers
% Output arguments:
% ncoin: number of coincidences
```

44 compresde0

```
function [resid,sigmar]=compresde0(y,x,str)
% PURPOSE: given a structure, it computes the
% model residuals and their covariance matrix using the difference
% equation, starting with zeros.
%---------------------------------------------------
% USAGE: [resid2,sigmar2]=compresde0(y,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%---------------------------------------------------
% RETURNS: resid  = the residuals
%          sigmar = the residuals covariance matrix
%---------------------------------------------------
```

45 compresex

```
function [resid,E,rSigmat]=compresex(y,x,str,tol,maxupdt,Y)
% PURPOSE: given a structure, it computes the model residuals and their
% covariance matrices using the square root CKMS recursions
%---------------------------------------------------
% USAGE: [resid2,sigmar2]=compresex(y,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%           Y      = an (nobs x (neqs x nbeta)) regression matrix
%---------------------------------------------------
% RETURNS: resid  = the residuals
%          E      = the augmented part of the residuals if Y is not
%                   empty
%         rSigmat = the Cholesky factors of the covariance matrix of
%                   residuals
%---------------------------------------------------
```

46 conmedfjac

```
function [fjac,g]=conmedfjac(beta,y,x,str)
% PURPOSE: this function computes the jacobian and the gradient
% corresponding to a VARMAX model
%---------------------------------------------------
% USAGE: [fjac,g]=conmedfjac(beta,y,x,str)
% where:    beta   = a (1 x nparma) vector of parameters
%           y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%---------------------------------------------------
% RETURNS: fjac  = the jacobian
%          g     = the gradient
%---------------------------------------------------
```

47 conmedfv

```
function [residv,beta,str]=conmedfv(beta,y,x,str)
% PURPOSE: given a structure, it computes the model residuals and their
% covariance matrix using the third step parameters of HR method.
%---------------------------------------------------
% USAGE: [residv,beta,str]=conmedfv(beta,y,x,str)
% where:   beta    = an (1 x nparm) vector of parameters
%           y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%---------------------------------------------------
% RETURNS: residv  = the residuals
%           beta   = an [(nparm + s) x 1] vector containing the
%                    parameters, where
%                    nparm : number of parameters
%                        s : number of ouputs
%            str   = updated structure containing the model information
%                    More specifically, if the model corresponding to
%                    str that is input to conmedfv is nonstationary or
%                    noninvertivble, an appropriate transformation is
%                    made to the AR or the MA part of the model to make
%                    them stable. The transformed parameters are in the
%                    updated field vgams.
%---------------------------------------------------
```

48 constant

```
function ct=constant(N,cons,dr,ds,dc,xc,s)
%*****************************************************************
%       This function generates a constant variable
%       for an ARIMA model that can have as nonstationary
%       autoregressive part
%
%       D(B) = (1-B)^dr (1-B^s)^ds (1-2cos(xc)B+B^2)^dc,     (1)
%
%       where 0<=dr<=2, 0<=ds<=1, and 0<=dc<=5.
%
%       The model is
%
%       D(B)z_t = cons + ARMA                                (2)
%
%       and the generated variable is
%
%       ct = cons/D(B)                                       (3)
%
%  INPUTS:
%      N : length of the data vector
%   cons : constant in eq.(2)
%     dr : regular differences
%     ds : seasonal differences
%     dc : integer such that dc*2 is order of (1-2cos(xc)B+B^2)^dc
%     xc : frequency in the polynomial (1-2cos(xc)B+B^2);
%          0 < xc < pi => |cos(xc)| < 1, implying that the polynomial
%          has two complex conjugate roots with modulus one
%      s : frequency of the data
%
% OUTPUTS:
%     ct : constant given by eq.(3)
```

49 constantx

```
function ct=constantx(N,cons,dr,ds,dS,dc,xc,s,S)
%*****************************************************************
%       This function generates a constant variable
%       for an ARIMA model that can have as nonstationary
%       autoregressive part
%
```

```
%    D(B) = (1-B)^dr (1-B^s)^ds (1-B^S)^dS (1-2cos(xc)B+B^2)^dc,     (1)
%
%       where 0<=dr<=2, 0<=ds<=1, 0<=dS<=1, and 0<=dc<=5.
%
%       The model is
%
%    D(B)z_t = cons + ARMA                                            (2)
%
%       and the generated variable is
%
%     t = cons/D(B)                                                   (3)
%
%  INPUTS:
%      N : length of the data vector
%   cons : constant in eq.(2)
%     dr : regular differences
%     ds : first seasonal differences
%     dS : second seasonal differences
%     dc : integer such that dc*2 is order of (1-2cos(xc)B+B^2)^dc
%     xc : frequency in the polynomial (1-2cos(xc)B+B^2);
%          0 < xc < pi => |cos(xc)| < 1, implying that the polynomial
%          has two complex conjugate roots with modulus one
%      s : first frequency of the data
%      S : second frequency of the data
%
% OUTPUTS:
%     ct : constant given by eq.(3)
```

50 copmut

```
function [U, T] = copmut(Um, Tm, ColsUT, da, du, n, m, full)
% this function computes the Unimodular polynomial matrices U and T
% It is used in function pmattrian.
% Author Felix Aparicio-Perez, Instituto Nacional de Estadistica, Spain
```

51 cospqu

```
function [c, q] = cospqu(x, y, win, width, wina)
%
%        This function computes the (smoothed) cross-periodogram
%        and the quadrature spectrum
%
```

```
%     INPUTS:
%------------
%        x,y : series
%        win : window used for smoothing the periodogram;
%             = 0 : no smoothing is performed
%             = 1 : Blackman-Tukey window
%             = 2 : Parzen window
%             = 3 : Tukey-Hanning window
%               if win < 0, it is set to 2
%    width : window width factor (1/3 by default)
%             if width <= 0, it is set to 1/3
%     wina : "a" parameter for Blackman-Tukey window (0.23 by default)
%             if wina <= 0, it is set to 0.23
%
%
%    OUTPUTS:
%------------
%       c    : cospectrum
%       q    : quadrature spectrum
```

52 crcreg

```
function   [nr1,ns1,nr,ns]=crcreg(y,s,maxr)
%
% This function applies the CRC criterion to the y series. It is based
% on the paper "A Strongly Consistent Criterion to Decide Between I(1)
% and I(0) Processes Based on Different Convergence Rates" by
% V\'{\i}ctor G\'{o}mez, (2013), Communications in Statistics -
% Simulation and Computation, 42, pp. 1848-1864.
%
% Input arguments:
% y    : series
% s    : number of seasons
% maxr : maximum regular differencing order considered
%
%
% Output arguments:
% nr1: number of regular differences found in the first step
% ns1: number of seasonal differences found in the first step
% nr: number of regular differences found
% ns: number of seasonal differences found
```

53 crcregr

54 croscor

```
function [cr,stdx,stdy]=croscor(x,y,lag)
%
%
%        This function computes the correlation between x(t) and
%        y(t+lag). Lag can be both positive and negative.
%
%     INPUTS:
%------------
%    REQUIRED
%       x,y : series; y = x, if autocorrelations are to be computed
%
%    OPTIONAL
%       lag : number of lags at which the correlations are to be
%             computed lag = length(y)-1, if lag is not input to
%             croscor or if lag is empty
%
%    OUTPUTS:
%------------
%        cv : correlations between x and y
%      stdx : standard deviation of x
%      stdy : standard deviation of y
```

55 croscov

```
function cv=croscov(x,y,lag)
%
%    This function computes the covariance between x(t) and y(t+lag).
%    Lag can be both positive and negative.
%
%
%     INPUTS:
%------------
%    REQUIRED
%       x,y : series; y = x, if autocovariances are to be computed
%
%    OPTIONAL
%       lag : number of lags at which covariances are to be computed
%             lag = length(y)-1, if lag is not input to croscov or if lag
%             is empty
%
%     OUTPUT:
%------------
%        cv : covariances between x and y
```

56 crosspan

```
function [co, ph, ga, fx, fy, frq] = crosspan(x, y, win, width, wina)
%
%        This function performs a cross spectral analysis.
%        See Granger's book for definitions.
%
%     INPUTS:
%------------
%         x : reference series
%         y : other series
%       win : window used for smoothing the periodogram;
%             = 0 : no smoothing is performed
%             = 1 : Blackman-Tukey window
%             = 2 : Parzen window
%             = 3 : Tukey-Hanning window
%               if win < 0, it is set to 2
%    width : window width factor (1/3 by default)
%             if width <= 0, it is set to 1/3
%     wina : "a" parameter for Blackman-Tukey window (0.23 by default)
%             if wina <= 0, it is set to 0.23
%    OUTPUTS:
%------------
%        co = coherence
%        ph = phase delay
%        ga = gain
%        fx = periodogram for x
%        fy = periodogram for y
%        frq= frequencies
```

57 csigsets

```
function [Sig, ColsUT, Cs, Ncs] = csigsets(SigBar, dtot, du, n, m)
% This routine extracts the Ci sets and, if possible, a triangular
% shape of size m
% Author Felix Aparicio-Perez, Instituto Nacional de Estadistica, Spain
```

58 cTheta

```
function [Theta,D]=cTheta(n,T,Sigma,phi,th)
%
%    this function computes the lower triangular matrix, Theta, such
%    that
%
%     \Phi*L = \Theta
%
%    using the CKMS recursions corresponing to an ARMA model.
%    The series is assumed to be stationary. The state space model is
%
%    x_{t} = T x_{t-1} + H a_{t}
%    y_{t}   = Z x_{t},
%
%    where Var(x_{1}) = Sigma.
%
%        INPUTS:
%        T: the T matrix
%    Sigma: the Sigma matrix
%      phi: an array containing the AR polynomial
%       th: an array containing the MA polynomial
%
%        OUTPUTS:
%    Theta: lower triangular matrix such that \phi*L = \Theta
%        D: diagonal matrix containing the standar errors
```

59 cumnor

```
function [Result,Ccum] = cumnor(Arg)
%     Last change:  BCM  21 Nov 97   10:07 pm
%      SUBROUTINE cumnor(Arg,Result,Ccum)
%      IMPLICIT NONE
%C**********************************************************************
%C
%C      SUBROUINE CUMNOR(X,RESULT,CCUM)
%C
%C
%C                                 Function
%C
%C
%C      Computes the cumulative  of    the  normal   distribution,
%C     i.e., the integral from -infinity to x of
%C          (1/sqrt(2*pi)) exp(-u*u/2) du
%C
%C     X --> Upper limit of integration.
%C                                    X is DOUBLE PRECISION
%C
%C     RESULT <-- Cumulative normal distribution.
%C                                    RESULT is DOUBLE PRECISION
%C
%C     CCUM <-- Compliment of Cumulative normal distribution.
%C                                    CCUM is DOUBLE PRECISION
%C
%C
%C     Renaming of function ANORM from:
%C
%C     Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portabel FORTRAN
%C     Package of Special Function Routines and Test Drivers"
%C     acm Transactions on Mathematical Software. 19, 22-32.
%C
%C     with slight modifications to return ccum and to deal with
%C     machine constants.
%C
%C**********************************************************************
```

60 dbptanbut

```
function [compf,ferror]=dbptanbut(D,Omegap1,Omegap2,Omegas2,Di,Thetac,...
Lambda)
%
% This function obtains the band-pass filter based on the Butterworth
% tangent filter corresponding to the parameters D(1), D(2), Omegap1,
% Omegap2, Omegas2. See "The Use of Bitterworth Filters for Trend and Cycle
% Estimation in Economic Time Series", G\'{o}mez, V. (2001), Journal of
% Business and Economic Statistics, 19, 365-373.
% The filter model is
%
%           z_t = s_t + n_t,
%   Alpha(B)s_t = num(B)b_t,    Var(b_t)=1
%
% where Alpha(z) = (1 - 2*Alph*z + z^2)^Di, num(z) = (1 - z^2)^Di, and n_t
% and b_t are independent white noises.
%
```

```
% The filter numerator is num*(1/sa). The filter denominator is den. Thus,
%
% H(z) = (1/sa)*(num(z)/den(z))
%
% The other filter is
%
% G(z) = (sqrt(Lambda)/sa)*(Alpha(z)/den(z))
%
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and D2.
%               It can be empty.
%     Omegap1 : a number, design frequency Omegap1 divided by pi. Required.
%     Omegap2 : a number, design frequency Omegap2 divided by pi. Required.
%     Omegas2 : a number, design frequency Omegas2 divided by pi. It can be
%               empty
%     Di      : a number, the exponent in Alpha(z) and num(z). It can be
%               empty.
%     Thetac  : a number, the frequency, divided by pi, of gain .5 in the
%               But. tan. filter. It can be empty.
%     Lambda  : a number, the signal to noise ratio (sigma^2_n/sigma^2_b)
%               in the But. tangent filter. It can be empty.
% Note: The usual specification is D, Omegap1, Omegap2 and Omegas2 (Di,
%       Thetac and Lambda empty). Alternatively, the user can enter
%       Omegap1, Omegap2, Di and Thetac (D, Omegas2 and Lambda empty) or
%       Omegap1, Omegap2, Di and Lambda (D, Omegas2 and Thetac empty).
%
% Output parameters: compf, a structure containing the following fields
%     .nterm1 : number of factors of degree 2 in den polynomial below
%     .nterm2 : number of factors of degree 4 in den polynomial below
%     .term1  : factors of degree 2 in den polynomial below
%     .term2  : factors of degree 4 in den polynomial below
%     .num    : a polynomial of degree 2*Di, (1 - z^2)^Di
%     .den    : a polynomial of degree 2*Di
%     .sa     : a positive number (sigmaa)
%     .Alpha  : a polynomial of degree 2*Di, (1 - 2*Alph*z + z^2)^Di
%     .Alph   : a number, in Alpha(z) = (1 - 2*Alph*z + z^2)^Di
%     .Di     : a positive integer
%     .Thetac : a number, the frequency of gain .5 in the But. tan. filter
%     .Lambda : a positive number, the square root of the noise to signal
%               ratio (sigma^2_n/sigma^2_b) in the But. tangent filter.
```

61 deltafil

```
function ct=deltafil(x,dr,ds,dc,xc,s)
%*********************************************************************
%       This function generates a filtered variable
%       for an ARIMA model that can have as filter
%
%       D(B) = (1-B)^dr (1-B^s)^ds (1-2cos(xc)B+B^2)^dc,
%
%       where 0<=dr<=2, 0<=ds<=1, and 0<=dc<=5.
%
%       The filtered variable, ct, satisfies
%
%       D(B)ct_t = x_t
%
%  INPUTS:
%      x : input series
%     dr : regular differences
%     ds : seasonal differences
%     dc : integer such that dc*2 is order of (1-2cos(xc)B+B^2)^dc
%     xc : frequency in the polynomial (1-2cos(xc)B+B^2);
%          0 < xc < pi => |cos(xc)| < 1, implying that the polynomial
%          has two complex conjugate roots with modulus one
%      s : frequency of the data
%
% OUTPUTS:
%     ct : filtered series
```

62 diferm

```
function yd=diferm(y,s)
%*********************************************************************
%  This function computes differences of order s for matrices
%
%  INPUTS:
%      y : data matrix
%      s : order of differencing
%  OUTPUT:
%     yd : data matrix after differencing
```

63 diffest

```
function [yd,beta]=diffest(y,Y,s,S,dr,ds,dS,est)
%*************************************************************************
% This function computes differences of the data vector and the matrix with
% regression variables and optionally performs estimation of regression
% coefficients
%
%  INPUTS:
%      y : data vector
%      Y : matrix with regression variables
%      s : frequency of the data, number of seasons
%      S : number of periods in each season
%     dr : regular differences
%     ds : seasonal differences corresponding to s (1 - B^s)^ds
%     dS : differences corresponding to S (1 - B^S)^dS
%    est = 1 : estimation of regression coefficients
%        = 0 : no estimation of regression coefficients
%
%  OUPUTS:
%     yd : data vector after differencing
%   beta : OLS estimator of the coefficients of the regression variables
```

64 distnj

```
function [pj,qj]=distnj(n,j,p,q)
%      given p and q of a signature matrix J=diag(I_p,-I_q) such that
%      n=p+q and j<=n, this function returns integers pj and qj such
%      that diag(I_pj,-I_qj) is the submatrix of J formed with the last
%      n-j+1 rows and columns. Any of p, q, pj or qj can be zero.
%
%---------------------------------------------------
% USAGE: [pj,qj]=distnj(n,j,p,q)
% where:    n = integer
%           j = integer <= n
%           p,q= integers such that J = diag(I_p,-I-q) is a signature
%           matrix with n=p+q
%
%---------------------------------------------------
% RETURNS: pj and qj,
%---------------------------------------------------
```

65 dlyapsq

```
function v=dlyapsq(a,b)
% Solves the discrete Lyapunov equation AV'VA' - V'V +BB'
% =0
% V is upper triangular with real non-negative diagonal entries
% this is equivalent to v=chol(dlyap(a,b*b')) but better conditioned
% numerically
%      Copyright (C) Mike Brookes 2002
%      Version: $Id: dlyapsq.m 713 2011-10-16 14:45:43Z dmb$
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page:
% http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
```

66 dsinbut

```
function [compf,ferror]=dsinbut(D, Thetap,Thetas,Di,Thetac,Lambda)
%
% This function obtains the sine Butterworth filter corresponding to the
% parameters d (differencing degree) and xc (frequency where gain is 1/2).
% See "The Use of Bitterworth Filters for Trend and Cycle Estimation in
% Economic Time Series", G\'{o}mez, V. (2001), Journal of Business and Economic
% Statistics, 19, 365-373.
% The filter model is
%
%           z_t = s_t + n_t,
%   Alpha(B)s_t = b_t,       Var(b_t)=1
%
% where Alpha(z) = (1 - z)^Di and n_t and b_t are independent white noises.
%
% The filter numerator is (1/sa). The filter denominator is den. Thus,
%
% H(z) = (1/sa)*(1./den(z))
%
% The other filter is
%
% G(z) = (sqrt(Lambda)/sa)*(Alpha(z)/den(z))
%
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and D2.
%               It can be empty.
%     Thetap  : a number, design frequency Thetap divided by pi. It can be
%              empty.
%     Thetas  : a number, design frequency Thetas divided by pi. It can be
%              empty.
%     Di      : a number, the exponent in Alpha(z). It can be empty.
%     Thetac  : a number, the frequency, divided by pi, of gain .5 in the
%              But. sine filter. It can be empty.
%     Lambda  : a number, the signal to noise ratio (sigma^2_n/sigma^2_b)
%              in the But. sine filter. It can be empty.
% Note: The usual specification is D, Thetap and Thetas (Di, Thetac and
%       Lambda empty). Alternatively, the user can enter Di and Thetac (D,
%       Thetap, Thetas and Lambda empty) or Di and Lambda (D, Thetap,
%       Thetas and Thetac empty).
%
% Output parameters: compf, a structure containing the following fields
%     .num    : = 1. (filter numerator)
%     .den    : a polynomial of degree Di (filter denominator)
%     .sa     : a positive number
%     .Alpha  : a polynomial, (1 - z)^Di
%     .Di     : a positive integer
%     .Thetac : a number, the frequency of gain .5 in the But. sine filter
%     .Lambda : a positive number, the square root of the noiese to signal
%              ratio (sigma^2_n/sigma^2_b) in the But. sine filter.
```

67 dtanbut

```
function [compf,ferror]=dtanbut(D, Thetap,Thetas,Di,Thetac,Lambda)
%
% This function obtains the tangent Butterworth filter corresponding to the
% parameters d (differencing degree) and xc (frequency where gain is 1/2).
% See "The Use of Bitterworth Filters for Trend and Cycle Estimation in
% Economic Time Series", G\'{o}mez, V. (2001), Journal of Business and Economic
% Statistics, 19, 365-373.
% The filter model is
%
%          z_t = s_t + n_t,
%   Alpha(B)s_t = num(B) b_t,    Var(b_t)=1
%
% where Alpha(z) = (1 - z)^Di, num(z) = (1 + z)^Di, and n_t and b_t are
% independent white noises.
%
% The filter numerator is (1/sa). The filter denominator is den. Thus,
%
```

```
% H(z) = (1/sa)*(num(z)/den(z))
%
% The other filter is
%
% G(z) = (sqrt(Lambda)/sa)*(Alpha(z)/den(z))
%
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and D2.
%               It can be empty.
%     Thetap  : a number, design frequency Thetap divided by pi. It can be
%               empty.
%     Thetas  : a number, design frequency Thetas divided by pi. It can be
%               empty.
%     Di      : a number, the exponent in Alpha(z). It can be empty.
%     Thetac  : a number, the frequency, divided by pi, of gain .5 in the
%               But. tangent filter. It can be empty.
%     Lambda  : a number, the signal to noise ratio (sigma^2_n/sigma^2_b)
%               in the But. tangent filter. It can be empty.
% Note: The usual specification is D, Thetap and Thetas (Di, Thetac and
%       Lambda empty). Alternatively, the user can enter Di and Thetac (D,
%       Thetap, Thetas and Lambda empty) or Di and Lambda (D, Thetap,
%       Thetas and Thetac empty).
%
% Output parameters: compf, a structure containing the following fields
%     .num    : a polynomial of degree Di, (1 + z)^Di (filter numerator)
%     .den    : a polynomial of degree Di  (filter denominator)
%     .sa     : a positive number
%     .Alpha  : a polynomial, (1 - z)^Di
%     .Di     : a positive integer
%     .Thetac : a number, the frequency of gain .5 in the But. tan. filter
%     .Lambda : a positive number, the square root of the noise to signal
%               ratio (sigma^2_n/sigma^2_b) in the But. tangent filter.
```

68 dtimesy

```
function [yd,ferror]=dtimesy(D,y)
%
%
% This function obtains the series
%        yd_t = D(B)*y_t,
% where D(z)=D_0 + D_1*z + .... + D_r*z^r is a polynomial matrix
% compatible with y_t and B is the backshift operator, By_t = y_{t-1}.
```

```
%
% Input arguments:
%                 D: a n x m x r polynomial matrix
%                 y: an m x s matrix
% Output arguments:
%                 yd: the series D(B)*y_t
```

69 duplication

```
function d = duplication(n)
% duplication(n)
% Returns Magnus and Neudecker's duplication matrix of size n
% Author: Thomas P Minka (tpminka@media.mit.edu)
```

70 durid

```
function oparm=durid(y,Y,infm,parm,ser,ols,a,durval,fid,fmarqdt)
%
% this function automatically estimates the duration period of the
% Easter effect for an ARIMA model with Easter correction
%
% Input arguments:
% y: vector containing the data
% Y: matrix containing regression variables
%   infm     : structure containing function names and optimization
%              options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 x is passed from marqdt to f but not passed from f to
%           marqdt
%           =0 x is passed from marqdt to f and passed from f to marqdt
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian and gradient at the solution is
%           performed
%           =0 no evaluation of jacobian and gradient at the solution
%           is performed
% .maxit:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
% parm: astructure containing model information, where
%  s:  seasonality
%  S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
% ser  : a structure containing the series parameters (the ones
%        specified by the user in the spec file and the default ones)
% ols  : = 1, perform OLS, = 0, use the Durbin Levinson algorithm in
%        the HR method
% a    : an integer, the degree of the AR approximation in the first
%        step of the Hanna-Rissanen method.
% durval : an integer array containing the possible days previous to
%          Easter (0 is also a value)
% fid    : the number of the external output file
% fmarqdt: a parameter for the estimation method
%          = 1 Levenberg-Marquardt method
%          = 0 Lsqnonlin (Matlab)
%
% Output arguments:
% oparm: the input parm structure plus the field
% .dur : the estimated duration of the Easter effect
```

71 durlev

```
function [fi,pc]=durlev(c0,cv)
%*********************************************************************
% This function applies the Durbin-Levinson algorithm to fit an AR
% model of order p = length(cv), given the autocovariances cv and the
% variance c0. As a byproduct, it also gives the partial
% autocorrelation coefficients.
%
%  INPUTS:
%     c0 : variance
%     cv : an (1 x p) vector containing the i-th covariances, i=1,...,p.
%
% OUTPUTS:
```

```
%     fi : an (1 x p) vector containing the AR(p) polynomial
%     pc : an (1 x p) vector containing the partial correlation
%          coefficients
%
% notation of fi is that of Box and Jenkins:
% y(t)-fi(1)*y(t-1)-...-fi(p)*y(t-p) = a(t)
```

72 durwat

```
function Dw = durwat(Res,J,K,Ss)
%*********************************************************************
% This function computes the Durbin-Watson statistic
%
%   INPUTS:
%     Res : residual vector
%       J : number of first residuals in Res not used in the
%           computation of Dw
%       K : integer specifying the last residual in Res used in the
%           computation of Dw
%      Ss : variance of Res
%
%   OUTPUT:
%      Dw : Durbin-Watson statistic
```

73 east

```
function Y=east(Iy,Im,N,Idur,Mq,Yd)
%
% this function generates the variable used to correct for Easter
% effect. Given the initial year Iy, the initial month Im, the desired
% length of the series N, and the duration of the effect Idur, the function
% computes for each month the proportion of the period Idur before Easter
% which falls in that month. It works for the period 1901-2099.
%
% input variables       Iy      : the initial year
%                       Im      : the initial period
%                       N       : the length of the desired vector
%                       Idur    : the length of the period before Easter
%                                 that the effect is thought to prevail
%                       Mq      : the series frequency (=12 for monthly,
%                                                       =4 for quarterly)
%                       Yd      : 199 x 2 array containing the dates of
%                                  Easter
% output variables      Y       : N x 1 array containing the variable%
```

74 eastdate

```
function Y=eastdate
%
% Dates of Easter from 1901 to 2099: first column month, second column day.
% Thus, to obtain the date of Easter for year 2004, first compute
% i=2004-1900=104. Then Y(104,:)= 4, 11 gives the result. The eleventh of
% April is the date of Easter for 2004.
```

75 enfinvp

```
function [str,ierror] = enfinvp(str)
% PURPOSE: enforces invertibility in a VARMAX model in echelon form
%          using polynomial methods
%---------------------------------------------------
% USAGE:  str = enfinv(str)
% where:    str is a structure containing all the information about the
%           model
%---------------------------------------------------
% RETURNS: the same structure with the invertible model
%---------------------------------------------------
```

76 enfstab

```
function   vgam=enfstab(stre,pol)
%
% This function multiplies the parameters of a matrix polynomial by some
% factor of the form .95^n until it becomes stable (all roots outside
% the unit circle).
%
% Input arguments:
% stre: a structure, containing model information. In particular,
% stre.vgams: vector containing the parameters of the polynomial matrix
%       in the form vec(phi_1),...,vec(phi_r),
%       vec(theta_0),vec(theta_1), ..., vec(theta_r),
%       vec(gamma_0),...,vec(gamma_r)
% pol: 'phi' or 'theta'
%
```

```
% Output arguments:
% vgam: the parameter vector containing corresponding to the stable
%       polynomial matrix
```

77 enfstabpol

```
function   polt=enfstabpol(pol)
%
% This function multiplies the parameters of a matrix polynomial by some
% factor of the form .95^n until it becomes stable (all roots outside
% the unit circle). That is, Polynomial P(z) is transformed into
% P(lambda*z)
%
% Input arguments:
% pol: a matrix polynomial
%
% Output arguments:
% polt: where polt(z) = pol(lambda*z)
```

78 enfstap

```
function [str,ierror] = enfstap(str)
% PURPOSE: enforces stationarity in a VARMAX model in echelon form
%          using polynomial methods
%---------------------------------------------------
% USAGE:  str = enfstap(str)
% where:    str is a structure containing all the information about the
%           model
%---------------------------------------------------
% RETURNS: the same structure with the stationary model
%---------------------------------------------------
```

79 estvarmaxkro

```
function [str,ferror] = estvarmaxkro(y,x,seas,kro,hr3,finv2,mstainv,nsig,tsig)
% PURPOSE: estimates a VARMAX model in echelon form using the
%          Hannan-Rissanen method (function MHANRIS). It returns a
%          structure containing the estimated model. The estimated
%          model is forced to be stationary and invertible.
%          After having estimated a VARMAX model with ESTVARMAXKRO, the
%          user can impose some zero restrictions in the model and
%          re-estimate the model using the MHANRIS function.
%---------------------------------------------------
% USAGE: [str,ferror] = ...
%             estvarmaxkro(y,x,seas,kro,hr3,finv2,mstainv,nsig,tsig)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%        seas      = seasonality
%         kro      = an (1 x neqs) array containing the Kronecker
%                    indices
%          hr3     = 1, perform only the first two stages of the HR
%                    method
%                    0, perform the three stages of the HR method, but
%                    only if the second stage model is invertible.
%        finv2     = 1, make model invertible after second stage of HR
%                    0, leave model as it is after second stage of HR
%      mstainv     = 1, use the DARE for enforcing stationarity or
%                    invertibility. This can only be used when there
%                    are no restrictions in the model.
%                  = 0, use multiplication by a small number.
%        nsig      = a (1 x 2) array. If nsig(i)=1, eliminate
%                    nonsignificant parameters after the i-th stage of
%                    the HR method, i=1,2. Default nsig=[0 0];
%        tsig      = a (1 x 2) array. If the t-value is less than
%                    tsig(i), the parameter is eliminated after the
%                    i-th stage of the HR method and the model is
%                    re-estimated, i=1,2.
%                    Default tsig=[.75 1.].
% If the three stages of the Hannan-Rissanen method are performed,
% residuals based on the difference equation are also obtained using the
% third stage estimates .
%---------------------------------------------------
% RETURNS: str = a structure containing the estimated parameters with
%          the following fields (see function MHANRIS)
%         s: number of outputs (neqs)
%         m: number of inputs  (nx)
%       kro: a (1 x s) vector containing the Kronecker indices
%       phi: an (s x s x maxkro) array with NaNs as parameters
%     theta: an (s x s x maxkro) array with NaNs as parameters
%     gamma: an (s x m x maxkro) array with NaNs as parameters
%     nparm: number of parameters
%      npar: an (s x s) array to define the Kronecker indices
%         F: an (n x n) matrix with NaNs as parameters, where n is the
%            McMillan degree = sum of the Kronecker indices
%         H: an (s x n) matrix with NaNs as parameters
%         K: an (n x s) matrix with NaNs as parameters
%         B: an (n x m) matrix with NaNs as parameters
%         D: an (s x m) matrix with NaNs as parameters
%    residv: an (nobs x s) matrix containing the residuals obtained in
%            the first stage
%   sigmarv: an (s x s) covariance matrix of residv
%      vgam: a {[(2*nlag + 1)*s^2 + (nlag + 1)*s*m + neqs] x 1} vector
%            containing the stacks of phi (except the first matrix),
%            theta, gamma, and s NaNs to account for the mean, where
%            nlag = max(kro).
%      bind: an [(nparm + s) x 1] index vector for the parameters in
%            vgam.
%      beta: an [(nparm + s) x 1] vector containing the parameters
%            estimated in the second stage
%        tv: an [(nparm + s) x 1] vector containing the t-values of beta
%     vgams: a vector like vgam but with the NaNs replaced with the
%            parameters estimated in the second stage.
%    vgamtv: a vector like vgam but with the NaNs replaced with the
%            t-values corresponding to the second stage.
%   noninv2: = 1, if model is noninvertible after the second stage
%            = 0, if model is invertible after the second stage
%    nonst2: = 1, if model is nonstationary after the second stage
%            = 0, if model is stationary after the second stage
%    resid2: an [(nobs-nlag) x s] matrix containing the residuals of the
%            second stage regression
%   sigmar2: covariance matrix of resid2
%    musers: mean corresponding to the constant estimated in the second
%            stage
%      phis: same as phi but with the NaNs replaced with the parameters
%            estimated in the second stage
%     phitv: same as phi but with the Nans replaced with the t-values
%            corresponding to the second stage
%    thetas: same as theta but with the NaNs replaced with the
%            parameters estimated in the second stage
%   thetatv: same as theta but with the Nans replaced with the t-values
%            corresponding to the second stage
%    gammas: same as gamma but with the NaNs replaced with the
%            parameters estimated in the second stage
%   gammatv: same as gamma but with the Nans replaced with the t-values
%            corresponding to the second stage
%        mu: an (s x 1) vector containing the constant estimated in the
%            second stage.
%      mutv: an (s x 1) vector containing the t-values of the constant
%            estimated in the second stage.
%     phist: same as phis but with coefficient matrices premultiplied by
%            phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   thetast: same as thetas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   gammast: same as gammas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%        Fs: same as F but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Hs: same as H but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ks: same as K but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Bs: same as B but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ds: same as D but with the NaNs replaced with the parameters
%            estimated in the second stage
%   noninv3: = 1, if model is noninvertible after the third stage
%            = 0, if model is invertible after the third stage
%    nonst3: = 1, if model is nonstationary after the third stage
%            = 0, if model is stationary after the third stage
%   resid23: an (nobs x s) matrix of residuals obtained before the third
%            stage using the VARMAX difference equation estimated in the
%            second stage starting with zeros.
%  sigmar23: covariance matrix of resid23
%     beta3: same as beta but containing the parameters
%            estimated in the third stage
%       tv3: same as tv but containing the t-values corresponding to the
%            third stage
%    vgams3: same as vgams but with the parameters estimated in the
%            third stage
%   vgamtv3: same as vgamtv but with the t-values corresponding to the
%            third stage
%      mus3: same as mu but with the constant estimated in the third
%            stage
%     phis3: same as phis but containing the parameters estimated in the
%            third stage
%   thetas3: same as thetas but containing the parameters estimated in
%            the third stage
%   gammas3: same as gammas but containing the parameters estimated in
%            the third stage
%    phitv3: same as phitv but containing the t-values corresponding to
%            the third stage
%  thetatv3: same as thetatv but containing the t-values corresponding
%            to the third stage
%  gammatv3: same as gammatv but containing the t-values corresponding
%            to the third stage
%     mutv3: same as mutv but containing the t-values corresponding to
%            the third stage
%    phist3: same as phist but containing the parameters estimated in %            the th
%  thetast3: same as thetast but containing the parameters estimated in %            the
%  gammast3: same as gammast but containing the parameters estimated in %            the
%       Fs3: same as Fs but containing the parameters estimated in the
%            third stage
%       Ks3: same as Ks but containing the parameters estimated in the
%            third stage
%       Bs3: same as Bs but containing the parameters estimated in the
%            third stage
%       Ds3: same as Ds but containing the parameters estimated in the
%            third stage
%       Hs3: same as Hs but containing the parameters estimated in the
%            third stage
%    resid3: an [(nobs-nlag) x s] matrix of residuals corresponding to %            the t
%   sigmar3: covariance matrix of resid3
%   resid3m: an (nobs x s) matrix containing the residuals obtained
%            using the VARMAX difference equation estimated in the
%            third stage starting with zeros.
%  sigmar3m: covariance matrix of resid3m
%---------------------------------------------------
```

80 estvarmaxpqrPQR

```
function [str,ferror] = estvarmaxpqrPQR(y,x,seas,ordersr,orderss,hr3,...
finv2,mstainv,nsig,tsig)
% PURPOSE: estimates a seasonal VARMAX model using the Hannan-Rissanen
%          method (function MHANRIS). It returns a structure containing
%          the estimated model. The estimated model is forced to be
%          stationary and invertible.
%          After having estimated a VARMAX model with ESTVARMAXPQRPQR,
%          the user can impose some zero restrictions in the model and
%          r-eestimate using the MHANRIS function.
%---------------------------------------------------
% USAGE: [str,ferror] = ...
%       estvarmaxpqrPQR(y,x,seas,ordersr,orderss,hr3,finv2,mstainv,
%                              nsig,tsig)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%        seas      = seasonality
%        ordersr   = a (1 x 3) array containing the regular VARMAX
%                    orders
%        orderss   = a (1 x 3) array containing the seasonal VARMAX
%                    orders
%          hr3     = 1, perform only the first two stages of the HR
%                    method
%                    0, perform the three stages of the HR method, but
%                    only if the second stage model is invertible.
%        finv2     = 1, make model invertible after second stage of HR
%                    0, leave model as it is after second stage of HR
%      mstainv     = 1, use the DARE for enforcing stationarity or
%                    invertibility. This can only be used when there
%                    are no restrictions in the model.
%                  = 0, use multiplication by a small number.
%        nsig      = a (1 x 2) array. If nsig(i)=1, eliminate
%                    nonsignificant parameters after the i-th stage of
%                    the HR method, i=1,2. Default nsig=[0 0];
%        tsig      = a (1 x 2) array. If the t-value is less than
%                    tsig(i), the parameter is eliminated after the
%                    i-th stage of the HR method and the model is
%                    r-eestimated, i=1,2.
%                    Default tsig=[.75 1.].
% If the three stages of the Hannan-Rissanen method are performed,
% residuals based on the difference equation are also obtained using the
% third stage estimates .
%---------------------------------------------------
% RETURNS: str = a structure containing the estimated parameters with
%          the following fields (see function MHANRIS)
%         s: number of outputs (neqs)
%         m: number of inputs  (nx)
%       kro: a (1 x s) vector containing the Kronecker indices
%       phi: an (s x s x maxkro) array with NaNs as parameters
%     theta: an (s x s x maxkro) array with NaNs as parameters
%     gamma: an (s x m x maxkro) array with NaNs as parameters
%     nparm: number of parameters
%      npar: an (s x s) array to define the Kronecker indices
%         F: an (n x n) matrix with NaNs as parameters, where n is the
%            McMillan degree = sum of the Kronecker indices
%         H: an (s x n) matrix with NaNs as parameters
%         K: an (n x s) matrix with NaNs as parameters
%         B: an (n x m) matrix with NaNs as parameters
%         D: an (s x m) matrix with NaNs as parameters
%    residv: an (nobs x s) matrix containing the residuals obtained in
%            the first stage
%   sigmarv: an (s x s) covariance matrix of residv
%      vgam: a {[(2*nlag + 1)*s^2 + (nlag + 1)*s*m + neqs] x 1} vector
%            containing the stacks of phi (except the first matrix),
%            theta, gamma, and s NaNs to account for the mean, where
%            nlag =  max(kro).
%      bind: an [(nparm + s) x 1] index vector for the parameters in
%            vgam.
%      beta: an [(nparm + s) x 1] vector containing the parameters
%            estimated in the second stage
%        tv: an [(nparm + s) x 1] vector containing the t-values of beta
%     vgams: a vector like vgam but with the NaNs replaced with the
%            parameters estimated in the second stage.
%    vgamtv: a vector like vgam but with the NaNs replaced with the
%            t-values corresponding to the second stage.
%   noninv2: = 1, if model is noninvertible after the second stage
%            = 0, if model is invertible after the second stage
%    nonst2: = 1, if model is nonstationary after the second stage
%            = 0, if model is stationary after the second stage
%    resid2: an [(nobs-nlag) x s] matrix containing the residuals of the
%            second stage regression
%   sigmar2: covariance matrix of resid2
%    musers: mean corresponding to the constant estimated in the second
%            stage
%      phis: same as phi but with the NaNs replaced with the parameters
%            estimated in the second stage
%     phitv: same as phi but with the Nans replaced with the t-values
%            corresponding to the second stage
%    thetas: same as theta but with the NaNs replaced with the
%            parameters estimated in the second stage
%   thetatv: same as theta but with the Nans replaced with the t-values
%            corresponding to the second stage
%    gammas: same as gamma but with the NaNs replaced with the
%            parameters estimated in the second stage
%   gammatv: same as gamma but with the Nans replaced with the t-values
%            corresponding to the second stage
%        mu: an (s x 1) vector containing the constant estimated in the
%            second stage.
%      mutv: an (s x 1) vector containing the t-values of the constant
%            estimated in the second stage.
%     phist: same as phis but with coefficient matrices premultiplied by
%            phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   thetast: same as thetas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   gammast: same as gammas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%        Fs: same as F but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Hs: same as H but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ks: same as K but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Bs: same as B but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ds: same as D but with the NaNs replaced with the parameters
%            estimated in the second stage
%   noninv3: = 1, if model is noninvertible after the third stage
%            = 0, if model is invertible after the third stage
%    nonst3: = 1, if model is nonstationary after the third stage
%            = 0, if model is stationary after the third stage
%   resid23: an (nobs x s) matrix of residuals obtained before the third
%            stage using the VARMAX difference equation estimated in the
%            second stage starting with zeros.
%  sigmar23: covariance matrix of resid23
%     beta3: same as beta but containing the parameters
%            estimated in the third stage
%       tv3: same as tv but containing the t-values corresponding to the
%            third stage
%    vgams3: same as vgams but with the parameters estimated in the %          third st
%   vgamtv3: same as vgamtv but with the t-values corresponding to the
%            third stage
%      mus3: same as mu but with the constant estimated in the third
%            stage
%     phis3: same as phis but containing the parameters estimated in the
%            third stage
%   thetas3: same as thetas but containing the parameters estimated in %          the t
%   gammas3: same as gammas but containing the parameters estimated in %          the t
%    phitv3: same as phitv but containing the t-values corresponding to %          the
%  thetatv3: same as thetatv but containing the t-values corresponding %          the t
%  gammatv3: same as gammatv but containing the t-values corresponding
%            to the third stage
%     mutv3: same as mutv but containing the t-values corresponding to %           the t
%    phist3: same as phist but containing the parameters estimated in %           the th
%  thetast3: same as thetast but containing the parameters estimated in %           the
%  gammast3: same as gammast but containing the parameters estimated in %           the
%       Fs3: same as Fs but containing the parameters estimated in the
%            third stage
%       Ks3: same as Ks but containing the parameters estimated in the
%            third stage
%       Bs3: same as Bs but containing the parameters estimated in the
%            third stage
%       Ds3: same as Ds but containing the parameters estimated in the
%            third stage
%       Hs3: same as Hs but containing the parameters estimated in the
%            third stage
%    resid3: an [(nobs-nlag) x s] matrix of residuals corresponding to %           the t
%   sigmar3: covariance matrix of resid3
%   resid3m: an (nobs x s) matrix containing the residuals obtained
%            using the VARMAX difference equation estimated in the
%            third stage starting with zeros.
%  sigmar3m: covariance matrix of resid3m
%----------------------------------------------------
```

81 eurpi

```
function [nr,ferror]=eurpi(Pi,hm1)
%
%
%   This function estimates the number of unit roots in Pi
%
% Input arguments:
%         Pi : an m x m matrix
%         hm1: a number with which the absolute value of the roots is
%              compared
% Output arguments:
%         nr: an integer, the number of unit roots
```

82 evarma11rurimp

```
function [Pi,alpha,betap,betaor,ferror] = evarma11rurimp(y,x,a,ir,nord)
% PURPOSE: performs VARMAX(1,1) estimation with rank imposed
%          and returns estimated Pi matrix in the form Pi=alpha*betap,
%          where Pi = -phi(1) in the error correction form.
%---------------------------------------------------
% USAGE: [beta,res] = evarma11r11Ru(yd,a,seas,y,x)
% where:    y     = an (nobs x neqs) matrix of observations
%           x     = matrix of input variables (nobs x nx)
%           a     = an (nobs x neqs) matrix of residuals
%           ir    = 0,1, corresponding to I(0) or I(1) case
%          nord   = rank of Pi
%                  (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: Pi     = an (nobs x nobs) matrix, equal to -phi(1), and such
%                    that Pi = alpha*betap
%          alpha  = an (nobs x nord) matrix
%          betap  = an (nord x nobs) matrix
%          betaor = an (nobs x nord) matrix orthogonal to betap
%         ferror  = a flag for errors
%---------------------------------------------------
```

83 exactmedfv

```
function [ff,beta,e,f,str,hb,Mb]=exactmedfv(beta,y,x,str,tol,Y,chb)
% PURPOSE: given a structure, it computes the functions in ff such that
% the expression ff'*ff is minimized in the Levenberg-Marquardt method.
% It uses the fast square root version of the Kalman filter.
%---------------------------------------------------
% USAGE: [ff,beta,e,f,str,hb,Mb]=exactmedfv(beta,y,x,str,tol,Y,chb)
% where:    beta   = the parameter vector
%           y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%           tol    = tolerance for not updating in the square CKMS rec.
%           Y      = an (nobs x (neqs x nbeta)) regression matrix
%           chb    = 1   compute hb and Mb
%                    0 do not compute hb and Mb
%---------------------------------------------------
% RETURNS: ff   = a vector containing the individual functions at the
%                 solution
%          beta = the parameter vector, possibly modified
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%         str   = the input structure str, possibly modified
%          hb   = the beta estimator
%          Mb   = the Mse of the beta estimator
%---------------------------------------------------
```

84 exactmedfvc

```
function [ff,beta,e,f,str,stx,recrs]=exactmedfvc(beta,y,x,str,Y,chb)
% PURPOSE: given a structure, it computes the functions in ff such that
% the expression ff'*ff is minimized with the Levenberg-Marquardt
% method. The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} + D*x_{t}  + a_{t}
%
%---------------------------------------------------
% USAGE: [ff,beta,e,f,str,stx,recrs]=exactmedfvc(beta,y,x,str,Y,chb)
% where:    beta   = the parameter vector
%           y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%           str    = a structure containing the model information
%           Y      = an (nobs x (neqs x nbeta)) regression matrix
%                      (neqs x nbeta) if it is time invariant
%           chb    = 1 compute hb and Mb in Kalman filter
%                      0 do not compute hb and Mb in Kalman filter
%                      2 compute recursive residuals updating regression
%                      p.
%                      3 compute recursive residuals with fixed
%                      regression p.
%---------------------------------------------------
% RETURNS: ff   = a vector containing the individual functions at the
%                 solution
%          beta = the parameter vector, possibly modified
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%          str  = the input structure str, possibly modified
%          stx  = a structure containing the following fields
%       .X,.Z,.G,.W,.T,.H,.ins,.i are the matrices and initial
%       conditions information corresponding to the state space model
%       for the endogenous part
%         .hb   = the beta estimator
%         .Mb   = the Mse of the beta estimator
%         .A    = the estimated augmented state vector at the end of
%                 filtering
%         .P    = the Mse of A at the end of filtering
%  .B,.D, .m1   = the initial state estimate and the matrices B and D
%                 for the state space model corresponding to the
%                 exogenous part
%        .kro   = the kronecker indices for the undecoupled model
%         recrs = standardized recursive residuals
%---------------------------------------------------
```

85 exactmedfvcd

```
function [ff,xv,e,f,str,stx,recrs]= ...
          exactmedfvcd(xv,y,xf,str,Y,chb,constant)
% PURPOSE: given a structure, it computes the functions in ff such that
% the expression ff'*ff is minimized with the Levenberg-Marquardt
% method. The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} + a_{t}
%
%---------------------------------------------------
% USAGE: [ff,xv,e,f,str,stx,recrs]=exactmedfvc(xv,y,x,str,Y,chb)
% where:    xv   = the parameter vector
%           y      = an (nobs x neqs) matrix of y-vectors
%           str    = a structure containing the model information
%           Y      = an (nobs x (neqs x nbeta)) regression matrix other
%                       than the mean
%                    (neqs x nbeta) if it is time invariant
%           chb    = 1 compute hb and Mb in Kalman filter
%                    0 do not compute hb and Mb in Kalman filter
%                    2 compute recursive residuals updating regression
%                    p.
%                    3 compute recursive residuals with fixed
%                     regression p.
%  constant: =1 a constant should be included in the model for the
%              differenced series
%             0 no constant in the model for the differenced series
%---------------------------------------------------
% RETURNS: ff   = a vector containing the individual functions at the
%                 solution
%          xv = the parameter vector, possibly modified
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%          str  = the input structure str, possibly modified
%          stx  = a structure containing the following fields
%       .X,.Z,.G,.W,.T,.H,.ins,.i are the matrices and initial
%       conditions information corresponding to the state space model
%       for the endogenous part
%         .hb   = the beta estimator
%         .Mb   = the Mse of the beta estimator
%         .A    = the estimated augmented state vector at the end of
%                 filtering
%         .P    = the Mse of A at the end of filtering
%        .kro   = the kronecker indices for the model
%         recrs = standardized recursive residuals
%---------------------------------------------------
```

86 fasttf

```
function [f,xv,e,g,M,yd1] = fasttf(xv,y,Y,parm,infm,xf)
%
%
%        this function computes model residuals using the fast Morf, Sihdu
%        and Kailath algorithm
%
%
%        INPUTS:
%       xv: an array containing model parameters
%        y: an array containing the input series
%        Y: a matrix containing regression variables
%     parm: a structure containing model information, where
%       .s:  seasonality
%       .S:  second seasonality
%       .p:  AR order
%      .ps: order of the AR of order s
%       .q:  order of the regular MA
%      .qs: order of the MA of order s (1 at most)
%      .qS: order of the MA of order S (1 at most)
%      .dr: order of regular differencing
%      .ds: order of differencing of order s
%      .dS: order of differencing of order S
%    .pvar:  array containing the indices of variable parameters
%    .pfix:  array containing the indices of fixed parameters
%  .ninput: number of inputs
%  .inputv: array containing the input variables
%   .delay: array with the delays of the input filters
%      .ma: array with the ma parameters of the input filters
%      .ar: array with the ar parameters of the input filters
%     .npr: number of forecasts
%     infm: a structure containing information on the estimation method,
%           where
%     .mvx: =1, exact max. likelihood; =0, unconditional mean squares
%     .chb: = 1  compute the beta estimate and its MSE
%             0  do not compute the beta estimate and its MSE
%     .inc: = 0, the initial states in the filter equations to obtain the
%                 filtered variables are equal to zero (not estimated)
%           = 1, the initial states in the filter equations are estimated
%       xf: an array containing fixed parameter values
%
%        OUTPUTS:
%        f: residual vector, whose sum of squares will be minimized
```

87 fdhess

```
function H = fdhess(f,x,varargin)
% PURPOSE: Computes finite difference Hessian
% -------------------------------------------------------
% Usage:  H = fdhess(func,x,varargin)
% Where: func = function name, fval = func(x,varargin)
%           x = vector of parameters (n x 1)
%    varargin = optional arguments passed to the function
% -------------------------------------------------------
% RETURNS:
%           H = finite differnce hessian
% -------------------------------------------------------

% Code from:
% COMPECON toolbox [www4.ncsu.edu/~pfackler]
% documentation modified to fit the format of the Ecoometrics Toolbox
% by James P. LeSage, Dept of Economics
% University of Toledo
% 2801 W. Bancroft St,
% Toledo, OH 43606
% jlesage@spatial-econometrics.com
```

88 fdis_cdf

```
function cdf = fdis_cdf(x,a,b)
% PURPOSE: returns cdf at x of the F(a,b) distribution
%---------------------------------------------------
% USAGE: cdf = fdis_cdf(x,a,b)
% where: x = a vector
%        a = numerator dof
%        b = denominator dof
%---------------------------------------------------
% RETURNS:
%   a vector of cdf at each element of x of the F(a,b) distribution
% --------------------------------------------------
% SEE ALSO: fdis_d, fdis_inv, fdis_rnd, fdis_pdf, fdis_prb
%---------------------------------------------------
```

89 fdjac2

```
function [fjac,g] = fdjac2(info,x,ff,varargin)
%
% This function computes a finite-difference jacobian
%
% Input arguments:
%   info structure containing function names and optimization options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 a transformation of the parameters corresponding to tr
%           variables is performed
%           =0 no transformation of parameters is performed
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian at gradient at the solution is
%           performed
%           =0 no evaluation of jacobian at gradient at the solution is
%           performed
%   .max:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
% ff:       the vector of functions evaluated at x
% x:        a vector containing the initial parameter values
% varargin: arguments to be passed to function f
%
```

```
% Output arguments:
% fjac     a matrix containing the jacobian
% g        a vector containing the (1/2)gradient
```

90 findurpir

```
function [nr,ferror]=findurpir(Pi,Th,hm1,can)
%
%
%   This function checks whether there are unit roots in Pi
%
% Input arguments:
%                 Pi: an m x m matrix
%                 Th: an m x m matrix
%                hm1: a number with which the absolute value of the
%                     roots is compared
%                can: a small number to handle cancellation
%  Output: nr= an integer, number of unit roots
%         ferror= a flag for errors
```

91 fipa

```
function r=fipa(fi)
%
% this function transforms the parameters of an autoregressive
% model 1+phi_1*z+...+phi_p*z^p into its partial autocorrelation
% coefficients
% input : fi, a 1 x p vector
% output: r, a 1 x p vector
```

92 fixvarmapqPQ

```
function [str,ferror] = fixvarmapqPQ(str)
% PURPOSE: given a structure containing information about a VARMA
% model, it fixes the parameters in phi, Phi, th, Th and Lh that
% correspond to those elements in phin, Phin, thn, Thn and Lhn that are
% not equal to NaN.
%---------------------------------------------------
% USAGE: str = fixvarmaxpqPQ(str)
% where:   str      = a structure created with function suvarmapqPQ
%---------------------------------------------------
```

```
% RETURNS: str = a structure containing model information
%----------------------------------------------------
```

93 fixvarmapqPQe

```
function [str,ferror] = fixvarmapqPQe(str)
% PURPOSE: given a structure containing information about a VARMA
% model, it fixes the parameters in phi, Phi, th, Th and Lh that
% correspond to those elements in phin, Phin, thn, Thn and Lhn that are
% not equal to NaN.
%----------------------------------------------------
% USAGE: str = fixvarmaxpqPQe(str)
% where:   str      = a structure created with function suvarmapqPQ
%----------------------------------------------------
% RETURNS: str = a structure containing model information
%----------------------------------------------------
```

94 fstlkhev

```
function [f,e,g,M]=fstlkhev(y,Y,Z,T,H,Sigma,chb)
%
%    this function computes the residuals using the CKMS recursions
%    corresponing to an ARIMA model, possibly with regression variables.
%    The series is assumed to be stationary. The state space model is
%
%    x_{t} = T x_{t-1} + H a_{t}
%    y_{t}   = Y \beta + Z x_{t},
%
%    where Var(x_{1}) = Sigma.
%
%        INPUTS:
%        y: an array containing the input series
%        Y: a matrix containing regression variables
%        Z: the Z matrix
%        T: the T matrix
%        H: the H matrix
%    Sigma: the Sigma matrix
%      chb: = 1, compute regression estimates, = 0, do not compute reg.
%           estimates.
%
%        OUTPUTS:
%        f: residual vector, whose sum of squares will be minimized
%        e: residual vector for inference
%        g: array containing the regression estimates
%        M: matrix containing the mse of the regression estimates
```

95 gacf

```
function [gammcf,gln] = gacf(A,X)
%
% auxiliary function called in function gammp (gamma density function)
```

96 gammln

```
function g=gammln(x)
%
% auxiliary function called in functions gacf and gser
```

97 gammp

```
function y=gammp(A,X)
%
% computes the probability density function of the gamma distribution
```

98 genfixseaspat

```
function Y = genfixseaspat(modescr,n)
%*****************************************************************
% This function creates regression variables corresponding to fixed
% seasonal patterns of the form
%
%  s_t = a*cos(w*t) + b*sin(w*t),
%
% where w=2*pi*k/n
%
%    INPUTS:
%  modescr : structure with the following fields:
%             .seas : number of seasonal patterns
%            .seasp : cell array containing the pairs [per_j,m_j]
%                     for the seasonal patterns, where per_j is the
%                     period and m_j is the number of harmonics in the
%                     j-th seasonal pattern
%        n : desired length for the regression variables
%----------------------------------------------------------------------
%
%   OUTPUTS:
%   Y : matrix with regression variables
%
```

99 genleap

```
function Y = genleap(Iy,Im,N,Mq)

% this function generates the leap year variable.  It works until 2100.
%
% this function generates the leap year variable.  It works until 2100.
%
% input variables       Iy      : the initial year
%                       Im      : the initial period
%                       N       : the length of the desired vector
%                       Mq      : the series frequency (=12 for monthly,
%                                                      =4 for
%                                                      quarterly)
%
% output variables      Y       : N x 1 vector containing the leap year
%                                 variable%
```

100 gensersm

```
function B = (phi,alpha,th,stda,ctr,Ns,N,l,seed)
%
% This function obtains Ns simulated series of length N that follow the
% model defined by phi, alpha, th, stda and ctr.
%
% input arguments:
% phi: AR polynomial
% alpha: nonstationary polynomial (differencing operator)
% th: MA polynomial
% stda: standard deviation of the innovations in percentage of the
%       levels.
% ctr: = 0 include neither a constant nor a time trend in the model
%      = 1 include a constant in the model
%      = 2 include a constant plus a time trend in the model
% Ns: number of simulations
% N: series length
% l: number of initial observations discarded in the simulated series
%
% output arguments:
% B: an Ns_N x 1 vector containing the stacked simulated series
```

101 genycor

```
function ycor=genycor(y,Y,ny,g)
%
% this function computes the series corrected by regression effects in a
% regression model of the form
%
% y = Y*g +e,
%
% where ny is the series length.
```

102 ggbpsinbut

```
function ggbpsinbut(D,omp1,omp2,oms2,d,alph,lambda)
%
% This function plots the squared gain function corresponding to a
% band-pass filter based on the sine Butterworth filter.
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and
%               D2. It can be empty.
%     ompp1   : a number, design frequency Omegap1 divided by pi. It
%               can be  empty
%     omp2    : a number, design frequency Omegap2 divided by pi. It
%               can be empty
%     oms2    : a number, design frequency Omegas2 divided by pi. It
%               can be empty
%     d       : a number, the exponent in Alpha(z) and num(z). Required
%     alph    : alph parameter in 1 - 2*alph*z + z^2. Required
%     lambda  : a number, the square root of the signal to noise ratio
%              (sigma^2_n/sigma^2_b) in the But. tangent filter.
%              Required
% Note: The parameters d, alph and lambda define the filter. If D, omp1,
%       omp2 and oms2 are entered by the user, the program will draw the
%       toleration lines.
```

103 ggbptanbut

```
function ggbptanbut(D,omp1,omp2,oms2,d,alph,lambda)
%
% This function plots the gain function corresponding to a
% band-pass filter based on the tangent Butterworth filter.
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and
%               D2. It can be empty.
%     ompp1   : a number, design frequency Omegap1 divided by pi. It
%              can be empty
%     omp2    : a number, design frequency Omegap2 divided by pi. It
%               can be empty
%     oms2    : a number, design frequency Omegas2 divided by pi. It
%               can be empty
%     d       : a number, the exponent in Alpha(z) and num(z). Required
%     alph    : alph parameter in 1 - 2*alph*z + z^2. Required
%     lambda  : a number, the square root of the signal to noise ratio
%              (sigma^2_n/sigma^2_b) in the But. tangent filter.
%              Required
% Note: The parameters d, alph and lambda define the filter. If D, omp1,
%       omp2 and oms2 are entered by the user, the program will draw the
%       toleration lines.
```

104 ggsintanbut

```
function ggsintanbut(D,Thetap,Thetas,d,thc)
%
% This function plots the gain function corresponding to a sine or
% tangent Butterworth filter.
% Input parameters:
%     D       : a (1 x 2) array containing the design tolerances D1 and
%               D2. It can be empty.
%     Thetap  : a number, design frequency Thetap divided by pi. It can
%               be empty
%     Thetas  : a number, design frequency Thetas divided by pi. It can
%               be empty
%     d       : a number, the exponent in Alpha(z) and num(z). Required
%     thc     : a number, the frequency for gain .5, divided by pi.
%               Required
% Note: The parameters d, and thc define the filter. If D, Thetap and
%       Thetas are entered by the user, the program will draw the
%       toleration lines.
```

105 glags

```
function   X=glags(x,lags)
%
% this function generates a matrix containing lags of x
%
% Input arguments:
% x: an (n) x (1) vector series
% lags: the number of lags to be generated
%
% Output argument:
% X: an (n-lags) x (lags) matrix containing the lagged variables
% Note that lags observations are lost
```

106 glcd

```
function [G,U,Dr,Nr,ierrglcd] = glcd(D,N,ivarmax)
% This function computes the Greatest Common Left Divisor G of a pair
% of polynomial matrices D and N of dimensions (n,n) and (n,m)
% respectively
% U is a unimodular matrix such that [D  N] U = [G 0] and G is lower
% triangular
% Moreover U = Vl -Nr  and if (D, N) was a left matrix fraction
% description of the  Ul  Dr  transfer function H then (Dr, Nr) is a
% right coprime matrix fraction description of H. If we compute inv(U),
% then  inv(U) = Dl  Nl and (Nl, Dl) is a left coprime matrix fraction
% description of H -Ur  Vr
```

107 gser

```
function [gamser,gln]=gser(A,X)
%
% auxiliary function called in function gammp
```

108 hanris

```
function  [x,laphi,x2]=hanris(yc,s,S,p,ps,q,qs,qS,ols,a)
%
```

```
% this function applies the Hannan-Rissanen method to obtain
% ARMA estimates
%
% Input arguments:
% yd   : a vector containing the series
% s    : seasonality
% S    : second seasonality
% p    : degree of AR polynomial
% ps   : degree of AR seasonal polynomial
% q    : degree of MA polynomial
% qs   : degree of MA seasonal polynomial
% qS   : degree of MA second seasonal polynomial
% ols  : = 1, perform OLS, = 0, use the Durbin Levinson algorithm
% a    : an integer, the degree of the AR approximation in the first
%        step of the Hanna-Rissanen method.
%
% Output arguments:
% x       : second or third step estimates (if process has an MA part
%           and the second step model is invertible)
% laphi   : a vector containing the estimates of the AR approximation in
%           first step of the Hanna-Rissanen method
% x2      : second step estimates
```

109 hist2

```
function [bin,cutpnt,otlrt0,otlr] = hist2(Y,med)
%
%-----------------------------------------------------------------------
%   Calculates the histogram of the Nobs long data vector, y.
%-----------------------------------------------------------------------
% c med     d  Local median of the y's
% c bin     i  Local vector of counts of observations between cut points
% c              i.e. bins
% c cutpnt  d  Local vector of cut points where observations counted in
% c              bin(i) are cutpnt(i-1) < y <= cutpnt(i)
% c              nobs-nefobs
% c otlr    i  Local notlr long list of the values of residuals
% c              greater than 3.25
% c otlrt0  i  Local notlr long list of residuals greater than 3.25
% c              standard deviations from the median
```

110 housref

```
function [Q,R,Indx,ierror] = housref(A,Q)
%
%---------------------------------------------------
% USAGE: [Q,R,Indx,ierror] = housref(A,Q)
% where:    A = an n x m matrix
%           Q = an n x n identity matrix
% Matrix A is reduced to row echelon form by means of Housholder
% transformations
%---------------------------------------------------
% RETURNS:
%           R = the upper triangular matrix in row echelon form such
%               that A=Q*[R;0]
%           Q = a unitary matrix such that Q'*A = [R;0]
%        Indx = an index containing the l.i. and the l.d. columns in R
%        Indx(i)=0, i-th column is l.i.
%               =1, i-th column is l.d.
%        ierror =1, dimension mismatch in A and Q
%               =2,  Q is not the identity matrix on input
%               =0, there are no errors on input
%---------------------------------------------------
% If matrix Q is given on input, then the orthogonal Q matrix such that
% Q'*A = [R;0] is computed and given on output. If not, Q=[] on output.
```

111 ical

```
function obs = ical(year,period,cstructure)
% PURPOSE: finds observation # associated with a year,period
%          given a cal() structure
% ---------------------------------------------------
% USAGE: obs = ical(year,period,c_str)
% where:  year   = year (4-digit)
%       period   = period ( <= frequency)
%         c_str  = a structure returned by cal() function
% ---------------------------------------------------
% RETURNS: obs = # of observation associated with year,period
% e.g., cstr = cal(1982,1,12)
%        obs = ical(1986,1,cstr) would return 48
% e.g., cstr = cal(1982,1,12)
%        obs = ical(1982,1,cstr) would return 1
% ----------------------------------------------------
% SEE ALSO: cal() a function to set up a time-series calendar
%           that associates observation #'s with dates
%           tsdate() that returns a string for the date associated
%           with observation #
% written by:
% James P. LeSage, Dept of Economics
% University of Toledo
% 2801 W. Bancroft St,
% Toledo, OH 43606
% jpl@jpl.econ.utoledo.edu
```

112 imparm

```
function [s,S,dr,ds,dS,p,ps,q,qs,qS,ny,nreg,pfix,pvar] = imparm(parm)
%
% this function obtains the paramters in structure parm
```

113 incossm

```
function [ins,i,ferror,P,V,XX]=incossm(T,H,ndelta)
%
%
%       This function obtains the initial conditions corresponding to
%       the model
%
%       y_t = X*beta + Z*alpha_t + G*epsilon_t
%       alpha_{t+1}= W*beta + T*alpha_t + H*epsilon_t
%
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse). It is
%       assumed that W_0=[] and, therefore, cw0 below is zero.
%
%       Input parameters:
%       T    : an (nalpha x nalpha) matrix
%       H    : an (nalpha x nepsilon) matrix
%     ndelta : a positive integer, the number of diffuse components in
%              alpha_t (= number of eigenvalues of unit module in T)
```

```
%
%        Output parameters:
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1],  where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%      ferror : flag for errors
%        P    : first output matrix of SortSchur function
%        V    : solution of the Lyapunov equation corresponding to the
%               stationary part, V = Fs*V*Fs' + Hs*Hs'
%       XX    : solution of the continuous time Lyapunov equation
%               Fn*XX - XX*Fs = A
```

114 incovma

```
function [A,Sigma,Xi]=incovma(phi,alpha,th)
%
%
%  This function computes the initial covariance matrix, Sigma,
%  and the A matrix for the initial conditions of the Kalman filter.
%  The initial state vector is
%
%        x_{d+1} = A*\delta + \Xi*c,
%
%  where Var(c) = Sigma. See Gomez and Maravall (1994), "Estimation,
%  Prediction and Interpolation for Nonstationary Series with the
%  Kalman Filter", Journal of the American Statistical Association,
%  89, 611-624. The filter is initialized at time t = d+1, where d is
%  the differencing degree, and the first d observations are stacked
%  to form the \delta vector.
```

115 inest

```
function  x=inest(yd,beta,s,S,p,ps,q,qs,qS,ols,a)
%
% this function computes initial ARMA estimates using
% the Hannan-Rissanen method
%
```

```
% Input arguments:
% yd   : an (n x m) matrix containing the series, yd(:,1), and an (n x
%        m-1) matrix of regression variables if m > 1.
% beta : an m-1 vector containing the OLS estimators if m > 1, empty if
%        m = 1
% s    : seasonality
% S    : second seasonality
% p    : degree of AR polynomial
% ps   : degree of AR seasonal polynomial
% q    : degree of MA polynomial
% qs   : degree of MA seasonal polynomial
% qS   : degree of MA second seasonal polynomial
% ols  : = 1, perform OLS, = 0, use the Durbin Levinson algorithm in
%        the HR method
% a    : an integer, the degree of the AR approximation in the first
%        step of the Hanna-Rissanen method.
%
% Output arguments:
% x    : a vector containing the ARMA parameter estimates
```

116 infcr

```
% PURPOSE: to determine optimal lag length using information criteria
%---------------------------------------------------
% USAGE:  lagsopt = infcr(y,maxlag,minlag,crt,prt,x)
% where:    y    = an (nobs x nvar) matrix of y-vectors
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           crt = the information criterion, 'aic' or 'bic'
%                  (default = 'aic')
%           prt = flag for printing
%                   0 = no, 1 = yes
%                   (default = 0)
%           x    = optional matrix of variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: lagsopt, the optimum number of lags
%---------------------------------------------------
```

117 inv2

```
function str = inv2(str)
```

```
% PURPOSE: given a structure passed after executing the second step of
% HR method such that the model is not invertible, this function
% inverts the model using the DARE.
%---------------------------------------------------
% USAGE: str = inv2(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

118 inv2r

```
function str = inv2r(str)
% PURPOSE: given a structure passed after executing the second step of
% HR method such that the model is not invertible, this function
% inverts the model.
%---------------------------------------------------
% USAGE: str = inv2r(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

119 inv3

```
function str = inv3(str)
% PURPOSE: given a structure passed after executing the third step of
% HR method such that the model is not invertible, this function
% inverts the model using the DARE.
%---------------------------------------------------
% USAGE: str = inv3(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

120 inv3r

```
function str = inv3r(str)
% PURPOSE: given a structure passed after executing the third step of
% HR method such that the model is not invertible, this function
% inverts the model.
%---------------------------------------------------
% USAGE: str = inv3r(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

121 invmodel

```
function [str,ierror]=invmodel(str)
% PURPOSE: given a structure with a noninvertible model, it obtains the
% invertible model with the same covariance generating function than
% the original model. The model is assumed to be in echelon form. The
% invertible model is also in echelon form. However, if there are
% additional constraints in the original model, the invertible model
% does not impose those constraints. It only imposes the constraints of
% the echelon form.
%---------------------------------------------------
% USAGE: str=invmodel(str)
% where:    str    = a structure containing the model information
%---------------------------------------------------
% RETURNS: str  = the structure with the invertible model
%---------------------------------------------------
```

122 invroots

```
function z=invroots(x,p,ps,q,qs,qS)
%*********************************************************************
% This function enforces that all roots of the polynomials of a
% multiplicative ARMA model are outside of the unit circle. This is
% achieved by first transforming the coefficients of each polynomial as
% if they were AR coefficients into partial correlation coefficients.
% Then, if there are parcor coefficients greater than one in absolute
% value, these are transformed into parcor coefficients that lie in
% (-1,1). Finally, the new parcor coefficients are transformed into AR
% coefficients.
%
%  INPUTS:
%      x : coefficients of the polynomials of a multiplicative ARMA
%          model p, ps, q, qs, qS : integers specifying where the
%          coefficients of the ARMA model are in x.
%      More specifically,
%      p : first p are AR coefficients
%     ps : starting with the (p+1)th coefficient, the next ps are AR c.
%      q : starting with the (p+1+ps+1)th coefficient, the next q are
%           MA c.
%     qs : starting with the (p+1+ps+1+q+1)th coefficient,
%          the next qs are MA c.
%     qS : starting with the (p+1+ps+1+q+1+qs+1)th coefficient,
%          the next qS are MA c.
%
% OUTPUTS:
%      z : coefficients such that all roots are outside of the unit
%          circle
```

123 isOctave

```
%%
%% Return: true if the environment is Octave.
%%
function retval = isOctave
```

124 jnorm

```
function jnr = jnorm(x,p,q)
%  given an n-column vector x and a signature matrix J=diag(I_p,-I_q),
%  this function calculates the J-norm of x, that is, the quantity
%  sqrt(x'*J*x), assuming that x'*J*x > 0.
%
%---------------------------------------------------
% USAGE: jnr = jnorm(x,p,q)
% where:    x = an n-column vector
%           p,q= integers such that J = diag(I_p,-I-q) is a signature
%           matrix
%---------------------------------------------------
% RETURNS: the J-norm of x
%---------------------------------------------------
```

125 jprod

```
function jpr = jprod(x,y,p,q)
%  given two n-column vectors, x and y, and a signature matrix
%  J=diag(I_p,-I_q), this function calculates the product x'*J*y.
%
%---------------------------------------------------
% USAGE: jpr = jprod(x,y,p,q)
% where:    x, y = n-column vectors
%           p,q= integers such that J = diag(I_p,-I-q) is a signature
%           matrix
%---------------------------------------------------
% RETURNS: the J product of x and y
%---------------------------------------------------
```

126 jqrt

```
function [Q,R] = jqrt(A,p,q)
%
%---------------------------------------------------
% USAGE: [Q,R] = jqrt(A,p,q)
% where:    A = an n x m matrix with n >=m
%           p,q = integers such that J=diag(I_p,-I_q) is a signature
%                 matrix
% It is assumed that A'*J*A = R'*J*R, where R is an upper triangular
% matrix. Then, there exists a J-unitary matrix Q such that A=Q*[R;0]
% To construct the matrix Q, J-unitary Housholder transformations are
% used
%---------------------------------------------------
% RETURNS:
%           R = the upper triangular matrix such that A=Q*[R;0]
%           Q = a factored form of a J-unitary matrix such that
%               QJQ'=Q'JQ=J and Q'*A = [R;0]
%           If the product Q'*B is desired for some matrix B, call
%           function qtb of this library (QtB=qtb(A,B,p,q)).
%---------------------------------------------------
%        A is an m by n array. on input A contains the matrix for
%          which the qr factorization is to be computed. on output
%          the strict upper trapezoidal part of A contains the strict
%          upper trapezoidal part of R, and the lower trapezoidal
%          part of a contains a factored form of Q.
```

127 kAI

```
function [C,ierror] = kAI(A)
%
```

```
% This function computes the Kronecker product of the matrix A by I
```

128 kIA

```
function [C,ierror] = kIA(A)
%
% This function computes the Kronecker product of the matrix I by A
```

129 kIv

```
function [C,ierror] = kIv(v)
%
% This function computes the Kronecker product of the matrix I by the
% vector v
```

130 lagaena

```
function [nlag,aenames]=lagaena(parm)
%
% function to create names for ARIMA estimation printing
```

131 lagaenar

```
function aenames=lagaenar(parm)
%
% this function generates an string array containing the names for the
% roots of all the polynomials of a transfer function model. These
% include the ARMA polynomials and the rational input filters.
```

132 lbs

```
function [qstat,pval,df,sea] = lbs(ne,p,r,nr)
%
% this function obtains the Q-statistics and their p-values for a
% sequence of integers p
%
% input arguments:
%             ne: length of the variable
%              p: a sequence of integers, i.e. p=1:lag
%               r: the covariance sequence
%              nr: number of parameters
% output arguments:
%           qstat: an array containing the Q-statistics
%            pval: an array containing the P-values
%              df: an array containing the degrees of freedom
%             sea: standard errors
```

133 leapid

```
function oparm=leapid(y,Y,infm,parm,ser,ols,a,fid,fmarqdt)
%
% this function identifies the leap year period using BIC
```

134 lkevarmapqPQ

```
function [F,xv,xf,e,f,g,M,A,P] = lkevarmapqPQ(xv,y,Y,xf,str,chb)
% PURPOSE: given a structure containing information about a VARMA
% model, it evaluates the likelihood of that model after putting it
% into state space form.
%---------------------------------------------------
% USAGE: [F,xv,xf,e,f,g,M,A,P] = lkevarmapqPQ(xv,y,Y,xf,str,chb)
% where:
%          xv       = a vector containing the parameters to be estimated
%          y        = an (n x neqs) matrix containing the data
%          Y        = an (n x (neqs x nbeta)) matrix containing
%                     regression matrix
%          xf       = a vector containing the fixed parameters
%          str      = a structure containing the initial model
%                     information
%          chb      =1, compute regression estimate and covariance
%                       matrix
%                   =0, do not compute
%---------------------------------------------------
%---------------------------------------------------
% RETURNS: F    = a vector containing the individual functions at the
%                 solution
%         xv    = a vector containing the estimated parameters
%         xf    = a vector containing the fixed parameters
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%          g    = a vector containing the regression estimates
%          M    = a matrix containing the mse of g
%          A    = the estimated augmented state vector
%                 at the end of filtering
%          P    = the mse of A at the end of filtering
%---------------------------------------------------
```

135 lkevarmapqPQd

```
function [F,xv,xf,e,f,g,M,A,P] = lkevarmapqPQd(xv,y,Y,xf,str,chb,constant)
% PURPOSE: given a structure containing information about a VARMA
% model, it evaluates the likelihood of that model after putting it
% into state space form.
%---------------------------------------------------
% USAGE: [F,xv,xf,e,f,g,M] = lkevarmapqPQd(xv,y,Y,xf,str,chb,constant)
% where:   y        = an (n x neqs) matrix containing the data
%          Y        = an (n x (neqs x nbeta)) matrix containing
%                     regression matrix
%          xv       = a vector containing the parameters to be estimated
%          xf       = a vector containing the fixed parameters
%          xi       = an index vector, if xi(i)=1, the i-th parameter
%                     is to be estimated, =0, not
%          str      = a structure containing the initial model
%                     information
%          chb      =1, compute regression estimate and covariance
%                    matrix
%                   =0, do not compute
%     constant      =1 a constant should be included in the model for
%                    the differenced series
%                    0 no constant in the model for the differenced
%                    series
%---------------------------------------------------
%---------------------------------------------------
% RETURNS: F    = a vector containing the individual functions at the
%                 solution
%         xv    = a vector containing the estimated parameters
%         xf    = a vector containing the fixed parameters
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%          g    = a vector containing the regression estimates
%          M    = a matrix containing the mse of g
%---------------------------------------------------
```

136 lkevarmapqPQe

```
function [F,xv,xf,e,f,g,M,A,P] = lkevarmapqPQe(xv,y,Y,xf,str,chb,constant)
% PURPOSE: given a structure containing information about a
% cointegrated VARMA model, it evaluates the likelihood of that model
% after putting it into state space form.
%---------------------------------------------------
% USAGE: [F,xv,xf,e,f,g,M] = lkevarmapqPQe(xv,y,Y,xf,str,chb,constant)
% where:   y        = an (n x neqs) matrix containing the data
%          Y        = an (n x (neqs x nbeta)) matrix containing
%                     regression matrix
%          xv       = a vector containing the parameters to be estimated
%          xf       = a vector containing the fixed parameters
%          xi       = an index vector, if xi(i)=1, the i-th parameter
%                     is to be estimated, =0, not
%          str      = a structure containing the initial model
%                     information
%          chb      =1, compute regression estimate and covariance
%                       matrix
%                   =0, do not compute
%     constant      =1 a constant should be included in the model for
%                      the differenced series
%                    0 no constant in the model for the differenced
%                      series
%---------------------------------------------------
%---------------------------------------------------
% RETURNS: F    = a vector containing the individual functions at the
%                 solution
%         xv    = a vector containing the estimated parameters
%         xf    = a vector containing the fixed parameters
%          e    = a vector containing the standardized residuals
%          f    = a scalar containing the determinantal term
%          g    = a vector containing the regression estimates
%          M    = a matrix containing the mse of g
%--------------------------------------------------
```

137 lkhev

```
function [f,e,g,M,AA,Sigma]=lkhev(y,Y,Z,T,H,A,Sigma,Xi,d)
%
%    this function computes the residuals and the GLS estimator of a
%    regression model with ARIMA errors using the two-stage Kalman
%    filter. The series can be nonstationary. The state space model is
%
%    x_{t} = T x_{t-1} + H a_{t}
%    y_{t}   = Y \beta + Z x_{t},
%
%    where the initial state vector is
%
%        x_{d+1} = A*\delta + \Xi*c,
%
%    and Var(c) = Sigma. See Gomez and Maravall (1994), "Estimation,
%    Prediction and Interpolation for Nonstationary Series with the
%    Kalman Filter", Journal of the American Statistical
%    Association, 89, 611-624. The filter is initialized at time t
%    = d+1, where d is
%    the differencing degree, and the first d observations are stacked
%    to form the \delta vector.
%
%        INPUTS:
%        y: an array containing the input series
%        Y: a matrix containing regression variables
%        Z: the Z matrix
%        T: the T matrix
%        H: the H matrix
%    Sigma: the Sigma matrix
%        A: the A matrix
%       Xi: the Xi matrix
%        d: an integer, the degree of the differencing operator
%
%        OUTPUTS:
%        f: residual vector, whose sum of squares will be minimized
%        e: residual vector for inference
%        g: array containing the regression estimates
%        M: matrix containing the mse of the regression estimates
%       AA: the estimated augmented state vector at the end of filtering
%    Sigma: the Mse of A at the end of filtering
```

138   lm1KF

```
function L=lm1KF(x,nd,s,S,p,ps,q,qs,qS)
%
% this function computes the L matrix for outlier detection
% using the fast Kalman filter algorithm. L is the inverse of the
% Cholesky factor of the covariance matrix of the data.
%     INPUTS:
%           x:  an array containing the ARIMA parameter values
%          nd:  an integer, the length of the differenced series
%           s:  seasonality
%           S:  second seasonality
%           p:  AR order
%          ps:  order of the AR of order s
%           q:  order of the regular MA
%          qs:  order of the MA of order s (1 at most)
%          qS:  order of the MA of order S (1 at most)
%  OUTPUTS:
%      L     : a lower triangular matrix
```

139 logF

```
function Y = logF(xx,f,varargin)
%
%
% This function evaluates the log of F'*F
```

140 lomonth

```
function Y = lomonth(Iy,Im,N,Mq)
%
% this function generates the length of month variable
% It works until 2100.
%
% input variables       Iy      : the initial year
%                       Im      : the initial period
%                       N       : the length of the desired vector
%                       Mq      : the series frequency (=12 for monthly,
%                                                        =4 for quarterly)
%
% output variables      Y       : N x 1 array containing the trading day
%                                 variables%
```

141 lratiocr

```
% PURPOSE: performs likelihood ratio test for var model  to determine
% optimal lag length. All models are estimated  using the same sample
% size: nobs - maxlag.
%---------------------------------------------------
% USAGE:  [lagsopt,initres] = lratiocr(y,maxlag,minlag,prt,x)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           prt    = flag for printing
%                    0 = no, 1 = yes (default = 0)
%           x      = optional matrix of variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: lagsopt = the optimum number of lags
%          initres = an (maxlag x neqs) matrix of initial residuals
%                    corresponding to the estimated VARXs of order
%                    1,2,...,maxlag.
%---------------------------------------------------
```

142 lratiocrax

```
unction [lagsopt,initres] = lratiocrax(y,maxlag,minlag,incr,prt,a,x)
% PURPOSE: performs sequential likelihood ratio tests in varmax(p,p,p)
%          models to determine optimal p, starting from
%          varmax(minlag,minlag,minlag) and proceeding up to
%          varmax(maxlag,maxlag,maxlag). All models are estimated using
%          the same sample size: nobs - maxlag.To estimjate the models,
%          it requires as input the estimated innovations.
%---------------------------------------------------
% USAGE:  lrratio(y,maxlag,minlag,incr,prt,a,x,seas)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           incr = the increment for the LR test
%           prt = flag for printing
%                    0 = no, 1 = yes
%                    (default = 0)
%           x    = matrix of input variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%           a    = matrix of estimated innovations (nobs x neqs)
%---------------------------------------------------
% RETURNS: lagsopt, the optimum number of lags
%---------------------------------------------------
```

143 lratiocrx

```
function [lagsopt,initres] = lratiocrx(y,maxlag,minlag,prt,x)
% PURPOSE: performs likelihood ratio test for varx model  to determine
% optimal lag length. All models are estimated  using the same sample
% size: nobs - maxlag.
%---------------------------------------------------
% USAGE:  [lagsopt,initres] = lratiocrx(y,maxlag,minlag,prt,x)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           prt    = flag for printing
%                    0 = no, 1 = yes (default = 0)
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: lagsopt = the optimum number of lags
%          initres = an (maxlag x neqs) matrix of initial residuals
%                    corresponding to the estimated VARXs of order
%                    1,2,..., maxlag.
%---------------------------------------------------
```

144 lratioppt

```
function [lagsopt,a,ferror] = lratioppt(y,x,seas,maxlag,minlag,prt)
% PURPOSE: performs sequential likelihood ratio tests in varmax(p,p,p)
%          models to determine optimal p, starting from p=minlag and
%          proceeding up to p=maxlag. All models are estimated
%          using the same sample size: nobs - maxlag.
%---------------------------------------------------
% USAGE:  [lagsopt,ferror] = lratioppt(y,x,seas,maxlag,minlag,prt)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           x    = matrix of input variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%         seas   = seasonality
%           maxlag = the maximum lag length. If empty on entry, it is
%                    calculated by the program as the order of the VARX
%                    approximation.
%           minlag = the minimum lag length
%           prt = flag for printing
%                    0 = no, 1 = yes
%                    (default = 0)
```

```
%---------------------------------------------------
% RETURNS: lagsopt, the optimum number of lags
%                      a = residuals of the VARX approximation (nobs x
%                      neqs)
%---------------------------------------------------
```

145 lratiopqr1

```
function [lagsopt,ferror] = lratiopqr1(y,l,a,x,maxlag,minlag,prt)
% PURPOSE: performs sequential likelihood ratio tests in the l-th equation
%          of a varmax(p,q,r) model to determine optimal p, q and r,
%          starting from varmax(minlag,minlag,minlag) and proceeding
%          up to varmax(maxlag,maxlag,maxlag). All models are
%          estimated  using the same sample size: nobs - maxlag.
%---------------------------------------------------
% USAGE:  [lagsopt,ferror] = lratiopqr1(y,l,a,x,maxlag,minlag,prt)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           l    = an integer to specify the l-th variable of y
%           a    = an (nobs x neqs) matrix of residuals estimated with
%                  a long VARX model
%           x    = matrix of input variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           prt = flag for printing
%                   0 = no, 1 = yes
%                   (default = 0)
%---------------------------------------------------
% RETURNS: lagsopt, the optimum number of lags
%---------------------------------------------------
```

146 lratiopqr

```
function [lagsopt,ferror] = lratiopqr(y,x,seas,maxlag,minlag,prt)
% PURPOSE: performs sequential likelihood ratio tests in varmax(p,q,r)
%          models to determine optimal p, q and r, starting from
%          varmax(minlag,minlag,minlag) and proceeding up to
%          varmax(maxlag,maxlag,maxlag). All models are estimated
%          using the same sample size: nobs - maxlag.
%---------------------------------------------------
% USAGE:  [lagsopt,ferror] = lratiopqr(y,x,seas,maxlag,minlag,prt)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           x    = matrix of input variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%         seas   = seasonality
%           maxlag = the maximum lag length. If empty on entry, it is
%                    calculated by the program as the order of the VARX
%                    approximation.
%           minlag = the minimum lag length
%           prt = flag for printing
%                    0 = no, 1 = yes
%                    (default = 0)
%---------------------------------------------------
% RETURNS: lagsopt, the optimum number of lags
%---------------------------------------------------
```

147 ltflag

```
function xlag = ltflag(x,n)
% this function generates a matrix of n+1 lags from a matrix or vector
% to use in ARMAX models. The first lag is the zero lag.
%---------------------------------------------------
% USAGE:     xlag = ltflag(x,nlag)
% where: x = a matrix of dimension nobs x nvar
%     nlag = number of lags to use in the ltf method
%---------------------------------------------------
% RETURNS:
%        xlag = a matrix of lags (nobs-nlag) x ((nlag+1)nvar)
%        x(t), x(t-1), x(t-2), ... x(t-nlag)
```

148 m2mor

```
function   [Morp,ferror]=m2mor(M)
%
% Given an (s x r+1) matrix M, this function obtains an (s-r x s)
% matrix Morp such that Morp*M = 0. The last column of M is assumed to
% be an index Idx, such that for the i-th row of M, Idx(i) = 0 means
% that this row is linearly independent.
%
% Inputs  :    M: an (s x r+1) matrix
%  Output : Morp: an (s-r x s) matrix such that Morp*M = 0.
```

149 macgf

```
function [c,ierror]=macgf(phi,th,Sigma,nc)
%
% This function computes the autocovariance function of a VARMA process
% phi(B)z_t=th(B)a_t
% The parameter nc is the number of desired autocovariances plus one,
% because the variance is included: c(0),c(1),...,c(nc-1)
```

150 marqdt

```
function [x,fjac,ff,g,iter,conf] = marqdt(info,x,varargin)
%
% This function minimizes a non-linear sum of squares function using
% Levenberg-Marquard's method.
% Input arguments:
%   info structure containing function names and optimization options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 x is passed from marqdt to f but not passed from f to
%           marqdt
%           =0 x is passed from marqdt to f and passed from f to marqdt
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian and gradient at the solution is
%           performed
%           =0 no evaluation of jacobian and gradient at the solution
%           is performed
% .maxit:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
% x:        a vector containing the initial parameter values
% varargin: arguments to be passed to function f
%
% Output arguments:
% x        a vector of (untransformed) parameters containing the
%          solution
% fjac     a matrix containing the jacobian at the solution
% ff       a vector containing the individual functions at the solution
% g        a vector containing the (1/2)gradient at the solution
% iter     a scalar whose value is the number of iterations
```

151 matechelon

```
function [str,ferror] = matechelon(kro,s,m)
% PURPOSE: obtains the matrix structure of a VARMAX model in echelon
% form, both in polynomial and state space form. The state space echelon
% form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = H*alpha_{t} + D*x_{t}  + a_{t}
%
% The VARMAX echelon form is
%  phi(B)*y_{t} = gamma(B)*x_{t} + theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%---------------------------------------------------------------------
% USAGE:  str = matechelon(kro,s,m)
% where:  kro = a (1 x s) vector containing the Kronecker indices
%           s = number of outputs
%           m = number of inputs
%---------------------------------------------------------------------
% RETURNS: a structure with the following fields
%         s: number of outputs
%         m: number of inputs
%       kro: a (1 x s) vector containing the Kronecker indices
%       phi: an (s x s x maxkro) array
%     theta: an (s x s x maxkro) array
%     gamma: an (s x m x maxkro) array
%     nparm: number of parameters
%      npar: an (s x s) array
%         F: an (n x n) matrix
%         H: an (s x n) matrix
%         K: an (n x s) matrix
%         B: an (n x m) matrix
%         D: an (s x m) matrix
% where maxkro = max(Kronecker indices), n=McMillan degree =
% sum(Kronecker indices), npar is an array of indices necessary to
% define the Kronecker indices, phi,theta and gamma are the VARMAX
% matrices in echelon form and F, H, K, B and D are the VARMAX matrices
% in state space echelon form. The parameters in the echelon form are
% indicated with NaN.
%---------------------------------------------------------------------
```

152 mautcov

```
function str=mautcov(y,lag,ic,nr)
%
%      This function computes the autocovariance matrices of y(t) up to
%      lag lag.
% ic = 1: compute the autocorrelation matrices
%      0: do not compute the autocorrelation matrices
% nr = number of parameters for Hosking's portmanteau statistic
%
%      returns: a structure str containing
%               c0: covariance at lag 0
%               cv: three dimensional array containing the
%                   autocovariance matrices up to lag lag
%                r: the autocorrelation matrices up to lag lag
%              sgn: matrices containing the significance of the
%                   autocorrelations according to the 2/sqrt(n) limits
%             sgnt: matrix containing all sgn matrices together
%              r0: the autocorrelation matrix for lag zero
%  In addition, if nr is input,
%             qstat: the Q statistics up to lag
%             pval : the p-values of the Q statistics
```

153 mclyapunov

```
function [X,ferror] = mclyapunov(A,B,C)
%
%
%        This function computes the solution to the continuous time
%        LYAPUNOV equation
%        AX - XB = C
%
% Input parameters:
% A       = a nxn matrix
% B       = a mxm matrix
% C       = a nxm matrix
% Output parameters:
% X       = the solution of the continuous time Lyapunov equation
%
% The method is as follows. First, compute the Schur decomposition of
% A and B (complex version). Then, solve recursively a system of linear
% equations.
% QaTaQa'X-XQbTbQb'=C;  TaQa'XQb - Qa'XQbTb = Qa'CQb;
% Z=Qa'XQb; TaZ - ZTb = Qa'CQb;  X=real(QaZQb');
```

154 mconestim

```
function  [xvf,str,ferror]=mconestim(y,x,str)
%
% This function estimates a VARMAX model in echelon form with some of
% its parameters possibly restricted to zero. Estimation is performed
% using the conditional method. It is assumed that initial values
% obtained with the three stages of the Hannan-Rissanen method are
% available in str.beta3. These initial estimates can be obtained using
% the functions MHANRIS, ESTVARMAXPQRPQR or ESTVARAMXKRO.
% The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = H*alpha_{t} + D*x_{t}  + a_{t}
%
% The VARMAX echelon form is

%  phi(B)*y_{t} = gamma(B)*x_{t} + theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%
%
% Inputs: y      = an (nobs x s) matrix of y-vectors
%         x      = matrix of input variables (nobs x m)
%                     (NOTE: constant vector automatically included)
%       str: a structure containing a preliminary model estimation
%            obtained with functions mhanris, estvaramxpqrPQR or
%             estvarmaxkro.
%  Output: xvf: estimated parameters
%          str: a structure containing the estimated VARMAX model.
%       ferror: flag for errors
% The fields of structure str on output are the same as those on input
% plus the following
%       residcon: an [(nobs-nlag) x s] matrix of standardized residuals
%                 after estimation, where nlag = max(kro)
%      sigmarcon: estimated covariance matrix of innovations
%        phiscon: an (s x s x nlag) array containing the estimated AR
%                 parameters
%      thetascon: an (s x s x nlag) array containing the estimated MA
%                 parameters
%      gammascon: an (s x m x nlag) array containing the estimated
%                 exogenous parameters
%       phitvcon: an (s x s x nlag) array containing the p-values of the
%                 estimated AR parameters
%     thetatvcon: an (s x s x nlag) array containing the p-values of the
%                 estimated MA parameters
%     gammatvcon: an (s x m x nlag) array containing the p-values of the
%                 estimated exogenous parameters
%         muscon: an (s x 1) vector containing the estimated constant
%        mutvcon: an (s x 1) vector containing the p-values of the
%                 constant
%       phistcon: same as phiscon but with coefficient matrices
%                 premultiplied by phis(:,:,1)^{-1} (VARMAX model not in
%                 echelon form)
%     thetastcon: same as thetascon but with coefficient matrices
%                 premultiplied by phis(:,:,1)^{-1} (VARMAX model not in
%                 echelon form)
%     gammastcon: same as gammascon but with coefficient matrices
%                 premultiplied by phis(:,:,1)^{-1} (VARMAX model not in
%                 echelon form)
%          Fscon: an (n x n) matrix containing the estimated F, where n
%                 is the McMillan degree = sum of the Kronecker indices
%          Kscon: an (n x s) matrix containing the estimated K
%          Bscon: an (n x m) matrix containing the estimated B
%          Dscon: an (s x m) matrix containing the estimated D
%          Hscon: an (s x n) matrix containing the estimated H
%      ferror   = flag for errors
```

155 mcrcregr

```
function   [D,nr,yd,DA,ferror]=mcrcregr(y,x)
%
% This function applies the CRC criterion to the multivariate y series
% to obtain the number of unit roots. This criterion is the
% generalization to the multivariate case of the one described for
% univariate time series in the paper "A Strongly Consistent Criterion
% to Decide Between I(1) and I(0) Processes Based on Different
% Convergence Rates" by V\'{\i}ctor G\'{o}mez, (2013), Communications
% in Statistics - Simulation and Computation, 42, pp. 1848-1864.
% Maximum regular differencing considered is one.
%
```

```
% Inputs: y: matrix containing the output series
%         x: matrix containing the input series
%  Output: D: an (ny x ny x seas+1) matrix 'differencing' matrix
%             polynomial
%         nr: an integer, number of unit roots
%         yd= matrix containing the 'differenced' series
%         DA= matrix of the form [DAr Indxr], where DAr is the regular
%             parameterization of the differencing polynomial, and
%             Indxr is an index vector to identify the l.i. rows of DAr.
%         ferror= a flag for errors
```

156 mctrbf

```
function [abar,bbar,cbar,t,k] = mctrbf(a, b, c, tol)
%CTRBF  Controllability staircase form.
%
%   [ABAR,BBAR,CBAR,T,K] = MCTRBF(A,B,C) returns a decomposition
%   into the controllable/uncontrollable subspaces.
%
%   If the controllability matrix, Co=CTRB(A,B), has rank r <= n
%   = SIZE(A,1), then there is a similarity transformation T such
%   that
%
%      Abar = T * A * T' ,  Bbar = T * B ,  Cbar = C * T'
%
%   and the transformed system has the form
%
%              | Anc    0 |           | 0 |
%      Abar =  ----------  ,  Bbar =  ---  ,  Cbar = [Cnc| Cc].
%              | A21   Ac |           |Bc |
%                                                -1          -1
%   where (Ac,Bc) is controllable, and Cc(sI-Ac)Bc = C(sI-A)B.
%
%   The last output K is a vector of length n containing the
%   number of controllable states identified at each iteration
%   of the algorithm.  The number of controllable states is SUM(K).
```

157 mdfestim1r

```
function   [D,DA,yd,ferror]=mdfestim1r(y,x,prt,nr)
%
% This function performs VARMAX(1,1) estimation with rank imposed and
% returns 'differencing' polynomial and 'differenced' series.
%
% Inputs: y: matrix containing the output series
%         x: matrix containing the input series
%         prt      = 1 print results of the VARX, VARMAX(p,p,p) tests
%         nr: an integer, number of regular unit roots
%  Output: D: an (ny x ny x 2) 'differencing' matrix polynomial
%         yd= matrix containing the 'differenced' series
%         DA= matrix of the form [DAr Indxr], where DAr is the
%             parameterization of the differencing polynomial, and
%             Indxr is an index vector to identify the l.i. rows of DAr.
%         ferror= a flag for errors
```

158 mdifpol

```
function [Dr,Fd,ferror]=mdifpol(r,t,Pi1,Pid1)
%
%
%   This function obtains the differencing polynomial matrix
%   correspondig to the matrix polynomial pi(z) = eye(s) + pi_1*z +
%   pi_2*z^2. Cases I(1)  and I(2) are considered.
%
% Input arguments:
%         r  : an integer, the rank of Pi1
%         t  : an integer, the rank of alphaor'*Pid(1)*betaor (Johansen)
%         Pi1: an s x s polynomial matrix, equal to -pi(1)
%        Pid1: an s x s polynomial matrix, equal to dpi(1)
% Output arguments:
%         Dr: a matrix polynomial, equal to the differencing matrix
%             polynomial
%         Fd: a matrix containing information for the parametrization of
%             the differencing matrix polynomial
%         ferror: a flag for errors
```

159 mecf2mid

```
function   [phi,D,DA,ferror]=mecf2mid(Lambda,alpha,betap)
%
% Given the error correction form
%
%       Phi(z) = Lambda(z)*(I-z*I) - Pi*z
%
```

```
% corresponding to a polynomial matrix Phi(z) = I + Phi_1*z + Phi_1*z +
% ... + Phi_{p}*z^{p}, where Pi = -Phi(1) = alpha*betap, this function
% obtains the 'differencing' polynomial matrix D(z) = I + D_1*z, were
%
%              D_1 = -betaor*pinv(betaor'*betaor)*betaor',
%
% and betaor is the orthogonal complement of beta = betap', and the
% polynomial matrix phi(z) = I +  phi_1*z + phi_1*z + ...
% + phi_{p-1}*z^{p-1} such that
%
%     Phi(z) =  phi(z)*D(z).
%
% On output, DA = [betaor Idx], where Idx is an index indicating the
% rows of betaor that are linearly independent (=0) and those that are
% linearly dependent (=1).
%
%  Inputst: Lambda : a polynomial matrix of degree p-1
%            alpha : an (s x r) matrix such that Pi = -Phi(1) =
%                    alpha*betap
%            betap : an (r x s) matrix
%  Output :    phi : a polynomial matrix of degree p-1
%                D : a polynomial matrix of degree 1
%               DA : an (s x r+1) matrix such that DAf=[betaor Idx]
```

160 mexactestim

```
function  [xvf,str,ferror]=mexactestim(y,x,str,Y)
%
% This function estimates a VARMAX model in echelon form with some of
% its parameters possibly restricted to zero using the exact maximum
% likelihood method with a fast square root filter. It is assumed that
% initial values obtained with the three stages of the Hannan-Rissanen
% or the conditional method are available in str.beta3. These initial
% estimates can be obtained using the functions MHANRIS,
% ESTVARMAXPQRPQR or ESTVARAMXKRO for the Hannan-Rissanen method or
% function MCONESTIM for the conditional method. The state space
% echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} + D*x_{t}  + a_{t}
%
% The VARMAX echelon form is
```

```
%
%  phi(B)*y_{t} = gamma(B)*x_{t} + theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%
% Inputs: y      = an (nobs x s) matrix of y-vectors
%         x      = matrix of input variables (nobs x m)
%       str: a structure containing a preliminary model estimation
%            obtained with functions mhanris, estvaramxpqrPQR or
%            estvarmaxkro.
%         Y: (nobs*s x nbeta) matrix for regression variables. If empty,
%            the variables are centered and the mean is not estimated.
%  Output: xvf: estimated parameters
%          str: a structure containing the estimated VARMAX model.
%       ferror: flag for errors
% The fields of structure str on output are the same as those on input
% plus the following
%         sigma2c: concentrated parameter estimate
%      sigmarexct: estimated covariance matrix of innovations
%                  regression paramters
%               e: stack of white noise residuals
%        phisexct: an (s x s x nlag) array containing the estimated AR
%                  parameters
%      thetasexct: an (s x s x nlag) array containing the estimated MA
%                  parameters
%      gammasexct: an (s x m x nlag) array containing the estimated
%                  exogenous parameters
%       phitvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated AR parameters
%     thetatvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated MA parameters
%     gammatvexct: an (s x m x nlag) array containing the p-values of
%                  the estimated exogenous parameters
%         musexct: an (s x 1) vector containing the estimated constant
%        mutvexct: an (s x 1) vector containing the p-values of the
%                  constant
%       phistexct: same as phiscon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not
%                  in echelon form)
%     thetastexct: same as thetascon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not
%                  in echelon form)
%     gammastexct: same as gammascon but with coefficient matrices
%                 premultiplied by phis(:,:,1)^{-1} (VARMAX model not %
%         Fsexct: an (n x n) matrix containing the estimated F, where
%                 n is the McMillan degree = sum of the Kronecker
%                 indices
%         Ksexct: an (n x s) matrix containing the estimated K
%         Bsexct: an (n x m) matrix containing the estimated B
%         Dsexct: an (s x m) matrix containing the estimated D
%         Hsexct: an (s x n) matrix containing the estimated H
%     ferror    = flag for errors
```

161 mexactestimc

```
function  [xvf,str,ferror]=mexactestimc(y,x,str,Y)
%
% This function estimates a VARMAX model in echelon form with some of
% its parameters possibly restricted to zero. Estimation is performed
% using the exact maximum likelihood method. It is assumed that initial
% values obtained with the three stages of the Hannan-Rissanen or the
% conditional method are available in str.beta3. These initial estimates
% can be obtained using the functions MHANRIS, ESTVARMAXPQRPQR or
% ESTVARAMXKRO for the Hannan-Rissanen method or function MCONESTIM for
% the conditional method. The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} + D*x_{t}  + a_{t}
%
% The parameters in beta are concentrated out of the likelihood.
%
% The VARMAX echelon form is

%  phi(B)*y_{t} = gamma(B)*x_{t} + theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%
% Inputs: y      = an (nobs x s) matrix of y-vectors
%         x      = matrix of input variables (nobs x m)
%       str: a structure containing a preliminary model estimation
%            obtained with functions mhanris, estvaramxpqrPQR or
%            estvarmaxkro.
%         Y: (nobs*s x nbeta) matrix for regression variables.
%            (s x nbeta) if it is time invariant;
%  Output: xvf: estimated parameters
%          str: a structure containing the estimated VARMAX model.
%       ferror: flag for errors
% The fields of structure str on output are the same as those on input
% plus the following
%         sigma2c: concentrated parameter estimate
%      sigmarexct: estimated covariance matrix of innovations
%                  regression paramters
%               e: stack of white noise residuals
%        phisexct: an (s x s x nlag) array containing the estimated AR
%                  parameters
%      thetasexct: an (s x s x nlag) array containing the estimated MA
%                  parameters
%      gammasexct: an (s x m x nlag) array containing the estimated
%                  exogenous parameters
%       phitvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated AR parameters
%     thetatvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated MA parameters
%     gammatvexct: an (s x m x nlag) array containing the p-values of
%                  the estimated exogenous parameters
%         musexct: an (s x 1) vector containing the estimated constant
%        mutvexct: an (s x 1) vector containing the p-values of the
%                  constant
%       phistexct: same as phiscon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not
%                  in echelon form)
%     thetastexct: same as thetascon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not %
%     gammastexct: same as gammascon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not %
%          Fsexct: an (n x n) matrix containing the estimated F, where
%                  n is the McMillan degree = sum of the Kronecker
%                  indices
%          Ksexct: an (n x s) matrix containing the estimated K
%          Bsexct: an (n x m) matrix containing the estimated B
%          Dsexct: an (s x m) matrix containing the estimated D
%          Hsexct: an (s x n) matrix containing the estimated H
%      ferror    = flag for errors
```

162 mexactestimcd

```
function  [xvf,str,ferror]=mexactestimcd(y,str,Y,constant)
```

```
%
% This function estimates a VARMA model in echelon form with some of its
% parameters possibly restricted to zero followed by a multivariate
% ‘‘differenced’’ series. Estimation is performed using the exact
% maximum likelihood method. It is assumed that initial values obtained
% with the three stages of the Hannan-Rissanen or the conditional
% method are available in str.beta3. These initial estimates can be
% obtained using the functions MHANRIS, ESTVARMAXPQRPQR or ESTVARAMXKRO
% for the Hannan-Rissanen method or function MCONESTIM for the
% conditional method. The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} +a_{t}
%
% The parameters in beta are concentrated out of the likelihood.
%
% The VARMA echelon form is

%  phi(B)*y_{t} = theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%
% Inputs: y       = an (nobs x s) matrix of y-vectors
%       str: a structure containing a preliminary model estimation
%            obtained with functions mhanris, estvaramxpqrPQR or
%            estvarmaxkro.
%         Y: (nobs*s x nbeta) matrix for regression variables other
%            than the mean. (s x nbeta) if it is time invariant;
%  constant: =1 a constant should be included in the model for the
%               differenced series
%             0 no constant in the model for the differenced series
%  Output: xvf: estimated parameters
%          str: a structure containing the estimated VARMAX model.
%       ferror: flag for errors
% The fields of structure str on output are the same as those on input
% plus the following
%         sigma2c: concentrated parameter estimate
%      sigmarexct: estimated covariance matrix of innovations
%                  regression paramters
%               e: stack of white noise residuals
%        phisexct: an (s x s x nlag) array containing the estimated AR
%                  parameters
%      thetasexct: an (s x s x nlag) array containing the estimated MA
%                  parameters
%      gammasexct: an (s x m x nlag) array containing the estimated
%                  exogenous parameters
%       phitvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated AR parameters
%     thetatvexct: an (s x s x nlag) array containing the p-values of
%                  the estimated MA parameters
%     gammatvexct: an (s x m x nlag) array containing the p-values of
%                  the estimated exogenous parameters
%         musexct: an (s x 1) vector containing the estimated constant
%        mutvexct: an (s x 1) vector containing the p-values of the
%                  constant
%       phistexct: same as phiscon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not
%                  in echelon form)
%     thetastexct: same as thetascon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not %
%     gammastexct: same as gammascon but with coefficient matrices
%                  premultiplied by phis(:,:,1)^{-1} (VARMAX model not %
%          Fsexct: an (n x n) matrix containing the estimated F, where
%                  n is the McMillan degree = sum of the Kronecker
%                  indices
%          Ksexct: an (n x s) matrix containing the estimated K
%          Bsexct: an (n x m) matrix containing the estimated B
%          Dsexct: an (s x m) matrix containing the estimated D
%          Hsexct: an (s x n) matrix containing the estimated H
%      ferror    = flag for errors
```

163 mhanris2

```
function str = mhanris2(y,res,x,str)
% PURPOSE: performs the second step of the multivariate Hannan-Rissanen
% method for VARMAX models with restrictions and returns the estimated
% parameters
%---------------------------------------------------
% USAGE: str = mhanris2(y,res,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           res    = an (nobs x neqs) matrix of residuals estimated
%                    with a long VARX model
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%           str    = a structure containing the structure of the VARMAX
```

```
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the estimated parameters
%---------------------------------------------------
```

164 mhanris3

```
function str = mhanris3(y,x,str)
% PURPOSE: performs the third step of the multivariate Hannan-Rissanen
% method for VARMAX models with restrictions and returns the estimated
% parameters
%---------------------------------------------------
% USAGE: str = mhanris3(y,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%           str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the estimated parameters
%---------------------------------------------------
```

165 mhanris21pqr

```
function str = mhanris21pqr(y,l,res,x,str)
% PURPOSE: performs the second step of the multivariate Hannan-Rissanen
% method for VARMAX models with restrictions and returns the estimated
% parameters
%---------------------------------------------------
% USAGE: str = mhanris2(y,res,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           l    = an integer to specify the l-th variable of y
%           res    = an (nobs x neqs) matrix of residuals estimated
%                    with a long VARX model
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%           str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the estimated parameters
%---------------------------------------------------
```

166 mhanris

```
function str = mhanris(y,x,seas,str,hr3,finv2,mstainv,nsig,tsig)
% PURPOSE: applies the Hannan-Rissanen method for VARMAX models in
% echelon form with restrictions to the series y with inputs x. It
% returns a structure containing the estimated model. The state space
% echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = H*alpha_{t} + D*x_{t}  + a_{t}
%
% The VARMAX echelon form is

%  phi(B)*y_{t} = gamma(B)*x_{t} + theta(B)a_{t},
%
% where B is the backshift operator, B*y_{t} = y_{t-1}.
%---------------------------------------------------
% USAGE: str = mhanris(y,x,seas,str,hr3,finv2,mstainv,nsig,tsig)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%       seas      = seasonality, =1 no seasonality, >1 seasonality
%        str      = a structure containing model information as given
%                    by function MATECHELON on output (model in echelon
%                    form)
%         hr3      = 1, perform only the first two stages of the HR
%                     method
%                    0, perform the three stages of the HR method, but
%                    only if the second stage model is invertible.
%       finv2     = 1, make model invertible after second stage of HR
%                    0, leave model as it is after second stage of HR
%     mstainv     = 1, use the DARE for enforcing stationarity or
%                    invertibility. This can only be used when there
%                    are no restrictions in the model.
%                 = 0, use multiplication by a small number.
%       nsig      = a (1 x 2) array. If nsig(i)=1, eliminate
%                    nonsignificant parameters after the i-th stage of
%                    the HR method, i=1,2. Default nsig=[0 0];
%       tsig      = a (1 x 2) array. If the t-value is less than
%                    tsig(i), the parameter is eliminated after the
%                    i-th stage of the HR method and the model is
%                    re-estimated, i=1,2. Default tsig=[.75 1.].
%---------------------------------------------------
```

```
% RETURNS: str = a structure containing the the following
%               fields
%         s: number of outputs (neqs)
%         m: number of inputs  (nx)
%       kro: a (1 x s) vector containing the Kronecker indices
%       phi: an (s x s x maxkro) array with NaNs as parameters
%     theta: an (s x s x maxkro) array with NaNs as parameters
%     gamma: an (s x m x maxkro) array with NaNs as parameters
%     nparm: number of parameters
%      npar: an (s x s) array to define the Kronecker indices
%         F: an (n x n) matrix with NaNs as parameters
%         H: an (s x n) matrix with NaNs as parameters
%         K: an (n x s) matrix with NaNs as parameters
%         B: an (n x m) matrix with NaNs as parameters
%         D: an (s x m) matrix with NaNs as parameters
%    residv: an (nobs x s) matrix containing the residuals obtained in
%            the first stage
%   sigmarv: an (s x s) covariance matrix of residv
%      vgam: a {[(2*nlag + 1)*s^2 + (nlag + 1)*s*m + neqs] x 1} vector
%            containing the stacks of phi (except the first matrix),
%            theta, gamma, and s NaNs to account for the mean, where
%            nlag =  max(kro).
%      bind: an [(nparm + s) x 1] index vector for the parameters in
%            vgam.
%      beta: an [(nparm + s) x 1] vector containing the parameters
%            estimated in the second stage
%        tv: an [(nparm + s) x 1] vector containing the t-values of beta
%     vgams: a vector like vgam but with the NaNs replaced with the
%            parameters estimated in the second stage.
%    vgamtv: a vector like vgam but with the NaNs replaced with the
%            t-values corresponding to the second stage.
%   noninv2: = 1, if model is noninvertible after the second stage
%            = 0, if model is invertible after the second stage
%    nonst2: = 1, if model is nonstationary after the second stage
%            = 0, if model is stationary after the second stage
%    resid2: an [(nobs-nlag) x s] matrix containing the residuals of the
%            second stage regression
%   sigmar2: covariance matrix of resid2
%    musers: mean corresponding to the constant estimated in the second
%            stage
%      phis: same as phi but with the NaNs replaced with the parameters
%            estimated in the second stage
%     phitv: same as phi but with the Nans replaced with the t-values
%            corresponding to the second stage
%    thetas: same as theta but with the NaNs replaced with the
%            parameters estimated in the second stage
%   thetatv: same as theta but with the Nans replaced with the t-values
%            corresponding to the second stage
%    gammas: same as gamma but with the NaNs replaced with the
%            parameters estimated in the second stage
%   gammatv: same as gamma but with the Nans replaced with the t-values
%            corresponding to the second stage
%        mu: an (s x 1) vector containing the constant estimated in the
%            second stage.
%      mutv: an (s x 1) vector containing the t-values of the constant
%            estimated in the second stage.
%     phist: same as phis but with coefficient matrices premultiplied by
%            phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   thetast: same as thetas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%   gammast: same as gammas but with coefficient matrices premultiplied
%            by phis(:,:,1)^{-1} (VARMAX model not in echelon form)
%        Fs: same as F but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Hs: same as H but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ks: same as K but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Bs: same as B but with the NaNs replaced with the parameters
%            estimated in the second stage
%        Ds: same as D but with the NaNs replaced with the parameters
%            estimated in the second stage
%   noninv3: = 1, if model is noninvertible after the third stage
%            = 0, if model is invertible after the third stage
%    nonst3: = 1, if model is nonstationary after the third stage
%            = 0, if model is stationary after the third stage
%   resid23: an (nobs x s) matrix of residuals obtained before the third
%            stage using the VARMAX difference equation estimated in the
%            second stage starting with zeros.
%  sigmar23: covariance matrix of resid23
%     beta3: same as beta but containing the parameters
%            estimated in the third stage
%       tv3: same as tv but containing the t-values corresponding to the
%            third stage
%    vgams3: same as vgams but with the parameters estimated in the
%            third  stage
%   vgamtv3: same as vgamtv but with the t-values corresponding to the
%            third stage
%      mus3: same as mu but with the constant estimated in the third
%            stage
%     phis3: same as phis but containing the parameters estimated in the
%            third stage
%   thetas3: same as thetas but containing the parameters estimated in
%            the third stage
%   gammas3: same as gammas but containing the parameters estimated in %            the t
%    phitv3: same as phitv but containing the t-values corresponding to %            the
%  thetatv3: same as thetatv but containing the t-values corresponding
%            to the third stage
%  gammatv3: same as gammatv but containing the t-values corresponding %            to th
%     mutv3: same as mutv but containing the t-values corresponding to %            the t
%    phist3: same as phist but containing the parameters estimated in %            the th
%  thetast3: same as thetast but containing the parameters estimated in %            the
%  gammast3: same as gammast but containing the parameters estimated in %            the
%       Fs3: same as Fs but containing the parameters estimated in the
%            third stage
%       Ks3: same as Ks but containing the parameters estimated in the
%            third stage
%       Bs3: same as Bs but containing the parameters estimated in the
%            third stage
%       Ds3: same as Ds but containing the parameters estimated in the
%            third stage
%       Hs3: same as Hs but containing the parameters estimated in the
%            third stage
%    resid3: an [(nobs-nlag) x s] matrix of residuals corresponding to %            the t
%   sigmar3: covariance matrix of resid3
%---------------------------------------------------
```

167 mid2mecf

```
function   [Pi,Lambda,alpha,betap,ferror]=mid2mecf(phi,D,DAf)
%
% Given a polynomial matrix of the form phi(z)*D(z), where phi(z) = I +
% phi_1*z + phi_1*z + ... + phi_{p}*z^{p} is an autoregressive
% polynomial matrix and D(z) = I + D_1*z is a 'differencing' polynomial
% matrix, this function obtains the error correction form such that
%
%      phi(z)*D(z) = Lambda(z)*(I-z*I) - Pi*z,
%
```

```
% where Lambda(z) = I + Lambda_1*z + ... + Lambda_{p-1}*z^{p-1} and Pi =
% -phi(1)*D(1). It is assumed that
%
%               D_1 = -betaor*pinv(betaor'*betaor)*betaor',
%
% where DAf = [betaor Idx] and Idx is an index indicating the rows of
% betaor that or linearly independent.
%
% Inputs  :    phi: a polynomial matrix of degree p
%                D: a polynomial matrix of degree 1
%              DAf: an (s x r+1) matrix such that DAf=[betaor Idx]
%  Output :    Pi : a matrix such that Pi = -phi(1)*D(1) = alpha*betap
%          Lambda : a polynomial matrix of degree p-1
%           alpha : an (s x r) matrix
%           betap : an (r x s) matrix
```

168 mifmin

```
function [y,fval,exitflag] = mifmin(r, den, x1, x2)
%
% minimization function called in candec
```

169 minfm

```
function infm = minfm(f,tr,mvx,tolf,maxit,nu0,jac,prt,chb,inc)
%*******************************************************************
% This function puts the function name and optimization options into
% structure infm
%
%   INPUTS:
%       f : a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%      tr > 0 : x is passed from marqdt to f but not passed from f to
%              marqdt
%         = 0 : x is passed from marqdt to f and passed from f to
%              marqdt
%     mvx :  =1 exact maximum likelihood
%            =0 unconditional least squares
%    tolf : parameter used for stopping
%   maxit : maximum number of iterations
%     nu0 : initial value of the nu parameter
%     jac = 1 : evaluation of jacobian and gradient at the solution is
%               performed
%         = 0 : no evaluation of jacobian and gradient at the solution
%               is performed
%     prt = 1 : printing of results
%         = 0 : no printing of results
%     chb = 1 : compute the beta estimate and its MSE
%           0 : do not compute the beta estimate and its MSE
%     inc = 0, the initial states in the filter equations to obtain the
%              filtered variables are equal to zero (not estimated)
%         = 1, the initial states in the filter equations are estimated
```

170 minft

```
function inft = minft(fid,fh,wd,nd,scale)
%*************************************************************************
% This function puts the printing options into structure inft
%
%          INPUTS:
%            .fid: the device on which the table will be written
%             .fh: flag for header and years
%             .wd: format width
%             .nd: number of decimal points
%          .scale: =1 scale data if necessary
%                  =0 do not scale data
%
%          OUPTUT:
%            inft: structure with printing options given by the inputs
```

171 miout

```
function iout = miout(omet,C,delta,schr,nrout,nind,tip,Yo,ornames)
%
% function to create a structure containing parameters about outlier
% detection
```

172 mlyapunov

```
function [P,ferror] = mlyapunov(F,Q,eigmod)
%
%
%        This function computes the solution to the LYAPUNOV equation
%        P=FPF' + Q
%
% Input parameters:
% F       = a nxn matrix
% Q       = a nxn symmetric matrix
% eigmod  = a number with which the modulus of the eigenvalues of F
%           will be compared
% Output parameters:
% P       = the solution of the Lyapunov equation
```

173 mnorm

```
function nr = mnorm(x)
%      given an n-vector x, this function calculates the square of the
%      euclidean norm of x.
%
%      the euclidean norm is computed by accumulating the sum of
%      squares in three different sums. the sums of squares for the
%      small and large components are scaled so that no overflows
%      occur. non-destructive underflows are permitted. underflows
%      and overflows do not occur in the computation of the unscaled
%      sum of squares for the intermediate components.
%      the definitions of small, intermediate and large components
%      depend on two constants, rdwarf and rgiant. the main
%      restrictions on these constants are that rdwarf^2 not
%      underflow and rgiant^2 not overflow. the constants
%      given here are suitable for every known computer.
%---------------------------------------------------
% USAGE: nr = mnorm(x)
% where:    x = an n-dimensional vector
%---------------------------------------------------
% RETURNS: the square of the euclidean norm of x
%---------------------------------------------------
```

174 mobsvf

```
function [abar,bbar,cbar,t,k] = mobsvf(a,b,c,tol)
%OBSVF  Observability staircase form.
%
%   [ABAR,BBAR,CBAR,T,K] = MOBSVF(A,B,C) returns a decomposition
%   into the observable/unobservable subspaces.
%
```

```
%   If the observability matrix, Ob=OBSV(A,C), has rank r <= n =
%   SIZE(A,1), then there is a similarity transformation T such that
%
%      Abar = T * A * T' ,  Bbar = T * B  ,  Cbar = C * T' .
%
%   and the transformed system has the form
%
%          | Ano   A12|           |Bno|
%   Abar =  ----------  ,  Bbar =  ---  ,  Cbar = [ 0 | Co].
%          |  0    Ao |           |Bo |
%
%                                                -1           -1
%   where (Ao,Bo) is controllable, and Co(sI-Ao) Bo = C(sI-A) B.
%
%   The last output K is a vector of length n containing the
%   number of observable states identified at each iteration
%   of the algorithm.  The number of observable states is SUM(K).
```

175 modelstruc

```
function models = modelstruc(xv,y,Y,pfix,pvar,xf,modescr,datei,npr)
%*********************************************************************
% This function creates a structure containing all the information
% needed for a structural model.
% This function allows for other values of freq besides 1, 4 and 12.
% Note that freq is a field of structure datei and that it is also the
% number of seasons. Therefore, it determines the seasonal component.
%
%    INPUTS:
%       xv : vector with parameters to be estimated
%           (see description below *)
%        y : data vector
%        Y : matrix for regression variables. It contains the stack
%            of the Y_t matrices
%     pfix : array with fixed parameter indices
%     pvar : array with variable parameter indices
%       xf : vector with fixed parameters (see description below *)
%  modescr : structure with the following fields:
%            (for the list of the codes of the components, see
%            description below **)
%            .trend : trend code
%            .slope : slope code
%             .seas : seasonal code
%            .cycle : cycle code
%              .xl1 : lower bound of the frequency interval in which the
%                     cycle is supposed to be defined
%              .xl2 : upper bound of the frequency interval in which the
%                     cycle is supposed to be defined
%               .ar : AR code
%            .irreg : irregular code
%            .conc : index for the parameter that is concentrated out
%                    (see description below ***)
%            .stord : array containing parameter indices
%                    (see description below ****)
%    datei : calendar structure (output of function cal)
%      npr : number of forecast
%-------------------------------------------------------------------
% * xv and xf are subvectors of the parameter vector x, where the
%   parameters,  except the one that is concentrated out, are put in
%   the following order:
%   1 - irregular standard deviation
%   2 - level standard deviation
%   3 - slope standard deviation
%   4 - seasonal standard deviation
%   5 - autoregressive standard deviation
%   6 - cycle standard deviation
%   7,8 - cycle parameters, rho and frequency
%   9,10,.. autoregressive parameters
%
% ** codes for the components:
%    trend = -1  constant
%             1  stochastic
%             2  Butterworth tangent
%    slope = -1  constant
%             1  stochastic
%    seas  = -1  fixed dummy seasonality
%             1  stochastic dummy seasonality
%             2  trigonometric seasonality
%             4  Butterworth tangent
%    cycle =  1  structural model cycle
%             2  Butterworth sine cycle
%    irreg =  1  stochastic
%      ar  =  k  autoregressive component of order k
%
% *** One of the standard deviations is concentrated out and,
% therefore, is not estimated. The field conout contains information
% about this standard deviation. The user can select this standard
% deviation or the program can do it automatically instead. The biggest
% variance should be selected.
%
% **** stord is an index such that its i-th element indicates to which
%     component (according to the ordering above) belongs the i-th
%     element of x.
%-----------------------------------------------------------------------
%
%   OUTPUTS:
%   models : the same structure as modescr (with .ar renamed to .arp)
%            plus the following fields:
%      matrices according to the model
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%      More specifically:
%            .X : an (n x nbeta) matrix containing the X_t matrices;
%                 a  (1 x nbeta) if it is time invariant;
%                 it can be []
%            .Z : an (n x nalpha) matrix containing the Z_t matrices;
%                 a  (1 x nalpha) matrix if it is time invariant
%            .G : an (n x nepsilon) matrix containing the G_t matrices;
%                 a  (1 x nepsilon) matrix if it is time invariant
%            .W : an (n*nalpha x nbeta) matrix containing the W_t
%                 matrices; an (nalpha x nbeta) matrix if it is time
%                 invariant; it can be []
%            .T : an (n*nalpha x nalpha) matrix containing the T_t
%                 matrices; an (nalpha x nalpha) matrix if it time
%                 invariant
%            .H : an (n*nalpha x nepsilon) matrix containing the H_t
%                 matrices; an (nalpha x nepsilon) if it is time
%                 invariant
%     .ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       .i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1 cca1],
%            where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%
%*********************************************************************
```

176 modelstrucmm

```
function models = modelstrucmm(xv,y,Y,pfix,pvar,xf,modescr,npr)
%**********************************************************************
% This function creates a structure containing all the information
% needed for a structural model with complex seasonal patterns. The
% seasonal component is of the form
%  s_t = \sum_{j=1}^{N} s_{t}^j, s_{t}^j =
% \sum_{i=1}^{m_j}s_{i,t}^j, where  n_j is the period of s_{t}^j, m_j
% is the  number of harmonics of s_{t}^j and
%
% [s_{i,t}^j    ]  [cos(2\pi i/n_j)  sin(2\pi i/n_j)]    [j_{i,t}   ]
% [s_{i,t}^{* j}]= [-sin(2\pi i/n_j) cos(2\pi i/n_j)]  + [j^*_{i,t} ].
%
%    INPUTS:
%       xv : vector with parameters to be estimated
%           (see description below *)
%        y : data vector
%        Y : matrix for regression variables. It contains the stack
%            of the Y_t matrices
%     pfix : array with fixed parameter indices
%     pvar : array with variable parameter indices
%       xf : vector with fixed parameters (see description below *)
%  modescr : structure with the following fields:
%            (for the list of the codes of the components, see
%            description below **)
%            .trend : trend code
%            .slope : slope code
%            .seas  : a cell array whose elements are 1 x 2 dimensional
%                    arrays defining the seasonal patterns. The two
%                    numbers in each array, [per_j,m_j], are the period
%                    and the number of harmonics.
%            .cycle : cycle code
%              .xl1 : lower bound of the frequency interval in which the
%                     cycle is supposed to be defined
%              .xl2 : upper bound of the frequency interval in which the
%                     cycle is supposed to be defined
%               .ar : AR code
%            .irreg : irregular code
%            .conc : index for the parameter that is concentrated out
%                    (see description below ***)
%            .stord : array containing parameter indices
%                    (see description below ****)
%      npr : number of forecast
%-------------------------------------------------------------------
% * xv and xf are subvectors of the parameter vector x, where the
%   parameters, except the one that is concentrated out, are put in the
%   following order:
%        1 - irregular standard deviation
%        2 - level standard deviation
%        3 - slope standard deviation
% 4,...3+N - ith-seasonal standard deviation, where N = number of
%            seasonal patterns
%      4+N - autoregressive standard deviation
%      5+N - cycle standard deviation
%  6+N,7+N - cycle parameters, rho and frequency
% 8+N,9+N,.. autoregressive parameters
%
% ** codes for the components:
%    trend = -1  constant
%             1  stochastic
%             2  Butterworth tangent
%    slope =  1  stochastic
%    cycle =  1  structural model cycle
%             2  Butterworth sine cycle
%    irreg =  1  stochastic
%      ar  =  k  autoregressive component of order k
%
% *** One of the standard deviations is concentrated out and,
%     therefore, is not estimated. The field conout contains
%     information about this standard deviation. The user can select
%     this standard deviation or the program can do it automatically
%     instead. The biggest variance should be selected.
%
% **** stord is an index such that its i-th element indicates to which
%     component (according to the ordering above) belongs the i-th
%     element of x.
%-----------------------------------------------------------------------
%
%   OUTPUTS:
%   models : the same structure as modescr (with .ar renamed to .arp)
%            plus the following fields:
%      matrices according to the model
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%      More specifically:
%            .X : an (n x nbeta) matrix containing the X_t matrices;
%                 a  (1 x nbeta) if it is time invariant;
%                 it can be []
%            .Z : an (n x nalpha) matrix containing the Z_t matrices;
%                 a  (1 x nalpha) matrix if it is time invariant
%            .G : an (n x nepsilon) matrix containing the G_t matrices;
%                 a  (1 x nepsilon) matrix if it is time invariant
%            .W : an (n*nalpha x nbeta) matrix containing the W_t
%                 matrices; an (nalpha x nbeta) matrix if it is time
%                 invariant; it can be []
%            .T : an (n*nalpha x nalpha) matrix containing the T_t
%                 matrices; an (nalpha x nalpha) matrix if it time
%                 invariant
%            .H : an (n*nalpha x nepsilon) matrix containing the H_t
%                 matrices; an (nalpha x nepsilon) if it is time
%                 invariant
%     .ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       .i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1 cca1],
%            where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%
%**********************************************************************
```

177 mparm

```
function parm= mparm(s,S,dr,ds,dS,p,ps,q,qs,qS,ny,nreg,...
      pfix,pvar,lam,flagm,trad,leap,east,dur,ninput,nmiss)
%
% function to create a structure containing ARIMA parameters
```

178 mpbf

```
function x=mpbf(a,b);
%multiplies pol. in B (a0+a1*B+...)by pol.in F=B^{-1} (b0+b1*F+...)
%in output:x(1) is the coeff. of F^(max)
```

179 mprint

```
function mprint(y,info)
% PURPOSE: print an (nobs x nvar) matrix in formatted form
%---------------------------------------------------
% USAGE:     mprint(y,info)
% where: y         = (nobs x nvar) matrix (or vector) to be printed
%        info      = a structure containing printing options
%        info.begr = beginning row to print,    (default = 1)
%        info.endr = ending row to print,       (default = nobs)
%        info.begc = beginning column to print, (default = 1
%        info.endc = ending column to print,    (default = nvar)
%        info.cnames = an (nvar x 1) string vector of names for columns
%                      (optional) e.g. info.cnames =
%                      strvcat('col1','col2');
%                      (default = no column headings)
%        info.rnames = an (nobs+1 x 1) string vector of names for rows
%                      (optional)
%                      e.g. info.rnames = strvcat('Rows','row1','row2');
%                      (default = no row labels)
%        info.fmt    = a format string, e.g., '%12.6f' or '%12d'
%                      (default = %10.4f)
%                      or an (nvar x 1) string containing formats
%                      e.g.,
%                      info.fmt=strvcat('%12.6f','%12.2f','%12d'); for
%                      nvar = 3
%        info.fid    = file-id for printing results to a file
%                      (defaults to the MATLAB command window)
%                      e.g. fid = fopen('file.out','w');
%        info.rflag  = 1 for row #'s printed, 0 for no row #'s (default
%                    = 0)
%        info.width  = # of columns before wrapping occurs (default =
%                      80)
%---------------------------------------------------
% e.g.   in.cnames = strvcat('col1','col2');
%        in.rnames = strvcat('rowlabel','row1','row2');
%        mprint(y,in), prints entire matrix, column and row headings
%        in2.endc = 3; in2.cnames = strvcat('col1','col2','col3');
%    or: mprint(y,in2), prints 3 columns of the matrix, just column
%        headings
%    or: mprint(y), prints entire matrix, no column headings or row
%        labels
% NOTES: - defaults are used for info-elements not specified
%        - default wrapping occurs at 80 columns, which varies
%          depending on the format you use, e.g. %10.2f will wrap
%          after 8 columns
%---------------------------------------------------
% SEE ALSO: tsprint, mprint_d, lprint
%---------------------------------------------------

% written by:
% James P. LeSage, Dept of Economics
% University of Toledo
% 2801 W. Bancroft St,
% Toledo, OH 43606
% jpl@jpl.econ.utoledo.edu
```

180 mprintar

```
function mprintar(ar,info,tit,strt)
% PURPOSE: print an (np,mp,kp) array in formatted form
%---------------------------------------------------
% USAGE:     mprint(ar,info)
% where: ar   = (np,mp,kp) array to be printed
%        tit         = a character string (tilte) for each (np,mp)
%                      matrix
%        strt       = an integer to start the counting of the (np,mp)
%                        matrices
%        info      = a structure containing printing options
%        info.begr = beginning row to print,    (default = 1)
%        info.endr = ending row to print,       (default = np)
%        info.begc = beginning column to print, (default = 1
%        info.endc = ending column to print,    (default =
%                   mp*kp)
%        info.cnames = an (mp*kpr x 1) string vector of names for
%                     columns (optional) e.g. info.cnames =
%                     strvcat('col1','col2');
%                     (default = no column headings)
%        info.rnames = an (np+1 x 1) string vector of names for rows
%                     (optional) e.g. info.rnames =
%                     strvcat('Rows','row1','row2');
%                     (default = no row labels)
%        info.fmt    = a format string, e.g., '%12.6f' or '%12d'
%                     (default = %10.4f) or an (mp*kp x 1) string
%                      containing formats
%        info.fid    = file-id for printing results to a file
%                     (defaults to the MATLAB command window)
%                     e.g. fid = fopen('file.out','w');
%        info.rflag  = 1 for row #'s printed, 0 for no row #'s (default
%                   = 0)
%        info.width  = # of columns before wrapping occurs (default =
%                     80)
%---------------------------------------------------
```

181 mprintr

```
function mprintr(result,fid)
% PURPOSE: print a two column table with the estimates and their
% t-values contained in structure result
%---------------------------------------------------
% USAGE:     mprint(x,info)
% where: fid       = file identifier for output (default = 1)
%        result    = a structure containing estimation results with the
%                   following fields:
%           .xvf : estimated parameters
%           .xf : vector of fixed parameters
%      .sigma2c : concentrated parameter estimate
%       .Sigmar : estimated exact covariance matrix of residuals
%            .tv : t-values of the estimated varma parameters
%     .residexct : matrix containing recursive residuals, only if Y is
%                  empty
%             .e : vector of standardized residuals at the end of
%                  estimation (Q'_2*y)
%            .ff : vector of nonlinear functions whose sum of squares is
%                  minimized at the end of estimation
%            .h  : vector of estimated regression estimates
%            .H  : matrix of mse of h
%            .A  : estimated state vector, x_{t|t-1}, obtained with the
%                  Kalman filter at the end of the sample
%            .P  : Mse of A
%            .tvr: vector of t-values for h
%        .ferror : flag for errors
%---------------------------------------------------
```

182 mshape

```
function SigBar = mshape(Tm)
%This function computes the shape of a matrix
```

183 mulFA

```
function [C, ierror] = mulFA(F,A,kro)
%
% This function computes the product of the matrices F and A, where F is
% assumed to be in echelon form (x_{t+1} = F*x_{t} + K*a_{t}).
```

184 mulHA

```
function [C, ierror] = mulHA(H,A,kro)
%
% This function computes the product of the polynomial matrices H and A
```

185 mulhkp

```
function [HKp]=mulhkp(H,K)
%
% this function multiplies matrix H by matrix K'
% assuming H*K' is symmetric
```

186 mulmols

```
function [beta,M,e]=mulmols(y,Y)
%
% Given the multivariate linear regression model
%
%  y'_t = Y'_t*beta +  epsilon'_t,       t=1,2,...,n,
%
% or, more compactly,
%
%  y = Y*B + E,
%
% this function computes the multiple OLS estimator, its covariance
% matrix and the residuals. The covariance matrix is not multiplied by
% Sigmar.
%---------------------------------------------------
% USAGE: [beta,M,e]=mulmols(y,Y)
% where:    y      = an (n x m) matrix of y-vectors
%           Y      = matrix of input variables (n x nY)
%---------------------------------------------------
% RETURNS: beta    = an (nY x m) matrix of regression coefficients
%             M    = an (nY x nY) matrix containing (Y'*Y)^{-1}
%             e    = an ((n-nY) x m) matrix containing the residuals
%---------------------------------------------------
```

187 mulols

```
function beta=mulols(y,Y)
%*********************************************************************
% This function computes the multiple OLS estimator
%
%   INPUTS:
%       y : (n x p) data matrix, where
%            p is the number of time series and n is the length of them
%       Y : (n x nreg) matrix with regression variables, where
%            nreg is the number of regression variables
%
%   OUTPUT:
%    beta : (nreg x p) vector of regression coefficients
```

188 multval

```
function [beta,tv,sigmar,covvecbeta,corvecbeta]=multval(y,Y)
%
% This function computes the multiple OLS estimator and the t-values of
% the multivariate linear regression model
%
%  y'_t = Y'_t*beta +  epsilon'_t,       t=1,2,...,n.
%
% Matrices y and Y contain the stack of y'_t and Y'_t, respectively.
%---------------------------------------------------
% USAGE: [beta,tv,sigmar,covvecbeta,corvecbeta]=multval(y,Y)
% where:    y      = an (n x m) matrix of y-vectors
%           Y      = matrix of input variables (n x nY)
%---------------------------------------------------
% RETURNS: beta    = an (nY x m) matrix of regression coefficients
%            tv    = an (nY x m) matrix containing the t-values
%        sigmar    = an (m x m) matrix containing the residual
%                    covariance matrix
%      covvecbeta  = an (nY*m x nY*m) matrix containing the covariance
%                    matrix of vec(beta)
%      corvecbeta  = an (nY*m x nY*m) matrix containing the correlation
%                    matrix of vec(beta)
%---------------------------------------------------
```

189 nberrecplot

```
function nberrecplot(recdates,y,color)
%*********************************************************************
%    Function nberrecplot plots areas specifying recession dates
%
%       INPUTS:
%      REQUIRED
%   recdates  : (mrec x 4) matrix with the peak and trough dates;
%                mrec is the number of recessions;
%               Columns:
%               1. year of each peak
%               2. month of year (from 1.) of each peak
%               3. year of each trough
%               4. month of year (from 3.) of each trough
%          y  : vector with at least two elements;
%               y can be a series or a vector with two
%               elements, in each case the minimum and the maximum
%               value will be used for specifying the height of the
%               rectangular area;
%
%     OPTIONAL(the order does not matter)
%      color  : color specification, e.g.
%               color = 'red'
%               color = [0.6,0.5,0.9];
%               default is [0.8,0.8,0.8] (gray)
%*************************************************************************
```

190 nse2

```
function [ct2,str] = nse2(y,residv,x,tsig2,str)
% PURPOSE:
% Eliminates nonsignificant parameter after second step of HR method.
% The Kronecker indices should be preserved. This means
% that in each row of the VARMAX model the maximum degree is the
% Kronecker index. For example, if s=1, p or q is equal to the
% Kronecker index. If s=2, in the first row, p_1 or q_1 is equal to
% k_1, in the second row, p_2 or q_2 is equal to k_2, etc.
% Put nsig2=1 and choose tsig2 for insignificant t-value
%---------------------------------------------------
% USAGE: [ct2,str] = nse2(y,residv,x,tsig2,str)
% where:    str    = a structure containing the structure of the VARMAX
% model
%---------------------------------------------------
% RETURNS:
%         ct2 = the number of parameters eliminated
%          str = a structure containing the inverted model
%---------------------------------------------------
```

191 nse3

```
function [ct3,str] = nse3(y,x,tsig3,invert2,str)
% PURPOSE:
% Eliminates nonsignificant parameter after third step of HR method.
% The Kronecker indices should be preserved. This means
% that in each row of the VARMAX model the maximum degree is the
% Kronecker index. For example, if s=1, p or q is equal to the
% Kronecker index. If s=2, in the first row, p_1 or q_1 is equal to
% k_1, in the second row, p_2 or q_2 is equal to k_2, etc.
% Put nsig3=1 and choose tsig3 for insignificant t-value
%---------------------------------------------------
% USAGE: [ct3,str] = nse2(y,residv,x,tsig2,str)
% where:    str    = a structure containing the structure of the VARMAX
% model
%---------------------------------------------------
% RETURNS:
%        ct3 = the number of parameters eliminated
%         str = a structure containing the inverted model
%---------------------------------------------------
```

192 nselimhr2

```
function [mp, minc, mint] = nselimhr2(y,x,str)
% PURPOSE: eliminates nonsignificant parameters in the second stage of
% the Hannan-Rissanen method for VARMAX models with restrictions
%---------------------------------------------------
% USAGE:  [mp, minc, mint] = nselimhr2(y,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%                x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%               str    = a structure containing the structure of the
%                        VARMAX   model
%---------------------------------------------------
% RETURNS:
%           minc  = a 1 x 3 array containing the index in the array of
%                   the eliminated parameter
%             minc = 'ph', 'th' or 'ga', referring to AR, MA or X part
%                    to  which the parameter belongs
%             mint =  the t-value of the eliminated parameter
%---------------------------------------------------
```

193 nselimhr3

```
function [mp, minc, mint] = nselimhr3(y,x,str)
% PURPOSE: eliminates nonsignificant parameters using the
% Hannan-Rissanen method for VARMAX models with restrictions
%---------------------------------------------------
% USAGE:  [mp, minc, mint] = nselimhr3(y,x,str)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%                x      = matrix of input variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%               str    = a structure containing the structure of the
%                        VARMAX   model
%----------------------------------------------------
% RETURNS:
%           minc  = a 1 x 3 array containing the index in the array of
%                     the eliminated parameter
%             minc = 'ph', 'th' or 'ga', referring to AR, MA or X part
%                     to which the parameter belongs
%             mint =  the t-value of the eliminated parameter
%----------------------------------------------------
```

194 nullref

```
function [K,ierror] = nullref(R,Indx)
%
%----------------------------------------------------
% USAGE: [K,ierror] = nullref(R,Indx)
% where:    R = an n x m matrix in column echelon form obtained after
%               applying housref on A'. Thus, if [Q,r,Indx,ierror] =
%               housref(A'), then R=r'.
%         Ind = an index containing the l.i. rows (0) and the l.d.
%               rows (1) of R.
%----------------------------------------------------
% RETURNS:
%           K = a matrix in reversed row echelon form containing a
%               basis of the left null-space of R. Therefore, K*R=0;
%        ierror =1, dimension mismatch in R and Indx
%               =0, there are no errors on input
%----------------------------------------------------
```

195 OLSres

```
function [olsres] = OLSres(out)
%
%
% This function obtains the OLS residuals after having used function
% arimaestos, arimaestni or arimaestwi
%
% input arguments:
% out: a structure, the output of function arimaestos, arimaestni or
% arinamestwi
%
```

```
% output arguments:
% res: a vector containing the OLS residuals
```

196 OLSrres

```
function [recrs,recr,srecr] = OLSrres(out)
%
%
% This function obtains the OLS residuals after having used function
% arimaestos, arimaestni or arimaestwi
%
% input arguments:
% out: a structure, the output of function arimaestos, arimaestni or
% arinamestwi
%
% output arguments:
%       recrs: standardized recursive residuals
%        recr: recursive residuals
%       srecr: covariance matrices of recursive residuals
```

197 outlr

```
function  [nrout,nind,tip,Yo]= outlr(y,Y,parm,iout,infm,npr,...
x0,sp1,sp2,fmarqdt,ols,aa)
%
% this function performs automatic outlier detection
% three types of outlier are considered:
% AO: defined by a one at t=T and zeros elsewhere
% TC: defined by a one at t=T followed by delta^i at t=T+i, i=1,2,...
%     Usually, delta=.7.
% LS: defined by ones from t=T to the end.
%
% Input arguments:
% y: vector containing the data
% Y: matrix containing regression variables
% parm: astructure containing model information, where
% .s:  seasonality
% .S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
% iout:   a structure containing information for outlier detection,
%         where
% .C:     critical value for outlier detection
% .delta: the value for delta in TC outliers
% .mthd: method to compute ARMA parameter estimates (0 Hannan Rissanen,
%                                                     1 Max. Lik.)
% .schr: =0 outliers of type AO and TC are considered (default)
%        =1 outliers of type AO, TC and LS are considered
%   infm     : structure containing function names and optimization
%                options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 x is passed from marqdt to f but not passed from f to
%           marqdt
%           =0 x is passed from marqdt to f and passed from f to marqdt
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian and gradient at the solution is
%              performed
%           =0 no evaluation of jacobian and gradient at the solution is
%              performed
% .maxit:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
%   .chb:   = 1  compute the beta estimate and its MSE
%             0  do not compute the beta estimate and its MSE
%   .inc:   = 0, the initial states in the filter equations to obtain
%                the filtered variables are equal to zero (not
%                estimated)
%           = 1, the initial states in the filter equations are
%                estimated
% x0: initial parameter vector
% npr: number of forecasts
% Note that AO and LS can be obtained by setting delta=1 and schr=2
% sp1,sp2: the outliers are searched in the time span (sp1,sp2)
% fmarqdt: a parameter for the estimation method
%          = 1 Levenberg-Marquardt method
%          = 0 Lsqnonlin (Matlab)
%
% Output arguments:
% nrout: number of outliers detected
% nind : index numbers of detected outliers
% tip  : array containing the type of the detectec outliers
% Yo: new desing matrix containing the detected outliers in the last
%     columns. That is, Yo=[Y X], where X contains the outlier
%     variables.
```

198 pafi

```
function fi=pafi(r)
%
% this function transforms the partial autocorrelation coefficients
% of an autoregressive model 1+phi_1*z+...+phi_p*z^p into the
% model parameters
% input : r, a 1 x p vector
% output: fi, a 1 x p vector
```

199 param2armaxe

```
function str = param2armaxe(str)
% PURPOSE: given a vector of Hannan-Rissanen estimates, it computes the
% VARMAX echelon form
%---------------------------------------------------
% USAGE: str = param2armaxe(str)
% where:    str    = a structure containing the vector of second step
%                    estimates
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the matrices of the VARMAX echelon form
%---------------------------------------------------
```

200 param2mdp

```
function [yd,Dr,Ds,ferror]=param2mdp(y,DA,nr,ns,seas)
%
%
% This function obtains the series
%         yd_t = D(B)*y_t,
% where D(z)=Dr(z)*Ds(z) is a polynomial matrix compatible
% with y_t and B is the backshift operator, By_t = y_{t-1}. The
% polynomial matrix D is given in condensed form in matrix DA.
%
% Input arguments:
%          y: an m x s matrix
%        DA= matrix of the form [DAr Indxr DAs Indxs], where DAr and
%            DAs are the parameterizations of the regular and seasonal
%            differencing matrix polynomials, and Indxr and Indxs are
%            two index vectors to identify the l.i. rows of DAr and
%            DAs.
%        nr= number of regular unit roots
%        ns= number of seasonal unit roots
%      seas: seasonality
% Output arguments:
%                 yd: the series D(B)*y_t
%                 Dr: regular differencing matrix polynomial
%                 Ds: seasonal differencing matrix polynomial
%             ferror: a flag for erros
```

201 param2sse

```
function str = param2sse(str)
% PURPOSE: given a vector of Hannan-Rissanen estimates, it computes the
% state space echelon form
%---------------------------------------------------
% USAGE: str = param2sse(str)
% where:    str    = a structure containing the vector of second step
%                    estimates
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                the matrices of the VARMAX echelon form
%---------------------------------------------------
```

202 parambeta

```
function   [DA,ferror]=parambeta(beta)
%
% Given an (s x r) matrix of rank r, this function parameterizes beta by
% finding r linearly independent rows. It returns an (s x r+1) matrix of
% the form [beta Idx], where beta is parameterized and Idx is an index
% such that Idx(i) = 0 if the i-th row is linearly independent and
% Idx(i) = 1 if the i-th row is linearly dependent.
%
% Inputs  :    betap : an (s x r) matrix
%  Output :     DA   : an (s x r+1) matrix such that DA=[beta Idx]
```

203 parar

```
function y=parar(x,p,ps,q,qs,qS)
%*********************************************************************
% Given the polynomials of a multiplicative ARMA model such that their
% coefficients have been transformed into parcor coefficients, this
% function converts parcor coefficients back into AR coefficients
%
%  INPUTS:
%      x : coefficients of the polynomials of a multiplicative ARMA
%          model p, ps, q, qs, qS : integers specifying where the
%          coefficients of the ARMA model are in x.
%      More specifically,
%      p : first p are AR coefficients
%     ps : starting with the (p+1)th coefficient, the next ps are AR c.
%      q : starting with the (p+1+ps+1)th coefficient, the next q are
%          MA c.
%     qs : starting with the (p+1+ps+1+q+1)th coefficient,
%          the next qs are MA c.
%     qS : starting with the (p+1+ps+1+q+1+qs+1)th coefficient,
%          the next qS are MA c.
%
% OUTPUTS:
%      y : coefficients of all the polynomials of the
%          ARMA model
```

204 parzen

```
function [w, m] = parzen(n, width)
%
%        This function computes the weights for the
%        Parzen window
%
%     INPUTS:
%         n : lentgh of the series
%     width : window width factor (1/3 by default)
%             if width <= 0, it is set to 1/3
```

```
%
%    OUTPUTS:
%         w : weights of the Parzen window
%         m : window lag size;
```

205 pecheform

```
function [phie,thetae,kro,ierror] = pecheform(phi,theta,kro)
%
% This function computes the echelon form corresponding to a transfer
% function Psi(z)=phi^{-1}(z)*theta(z) and, possibly, the Kronecker
% indices.  It is assumed that phi(z) is square and that phi(0) is
% nonsingular. Polynomial matrix theta(z) can be nonsquare and,
% therefore, theta(0) is not assumed to be the identity matrix.
%---------------------------------------------------
% USAGE: [phie,thetae,kro,ierror] = pecheform(phi,theta,kro)
% where:    phi   = a k x k polynomial matrix with phi(0) nonsingular
%           theta = a k x m polynomial matrix
%           kro   = a 1 x k vector containing the Kronecker indices
%---------------------------------------------------
% RETURNS:
%           phie  = the AR echelon polynomial matrix
%          thetae = the MA echelon polynomial matrix
%           kro   = a 1 x k vector containing the Kronecker indices
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
% If kro is not input, the function uses functions housref and nullref
% on the augmented Sylvester matrices constructed with phi and theta to
% compute the Kronecker indices. If kro is input, a system of linear
% equations based on an appropriate augmented Sylvester matrix is
% solved.
```

206 periodg

```
function [f, frq] = periodg(x, win, width, wina)
%        This function computes the (smoothed) periodogram
%
%     INPUTS:
%------------
%         x : series
%         win : window function used for smoothing the peiodogram
%               0, no window is applied (nonsmoothed periodogram).
%               1, the Blackman-Tukey window with parameter wina
%               2, the Parzen window (default)
%               3, the Tukey-Hanning window (same as Blackman-Tukey with
%                  wina= 0.25)
%               if win < 0, it is set to 2
%       width : window width factor (1/3 by default)
%               if width <= 0, it is set to 1/3
%        wina : "a" parameter for Blackman-Tukey window (0.23 by default)
%               if wina <= 0, it is set to 0.23
%    OUTPUTS:
%      f    : (smoothed) periodogram
%      frq  : array containing the frequencies
```

207 permat

```
function [B] = permat(A, n, m);
% This function permutes the rows of a matrix A (that has nm rows) as
% if we premultiplied A by Knm, the permutation matrix of parameters n
% and m
%
%permutation matrix: P*vec(A) = vec(A')
%
```

208 pfctsusm

```
function pfctsusm(out)
%*********************************************************************
% This function plots forecasts
%
%      INPUT:
%       out : structure with the following fields:
%      .yor : original time series
%        .y : time series used in computation of forecasts
%       .ny : length of y
%      .pry : forecasts of y
%     .spry : standard errors of pry
%     .opry : forecasts of y in the original scale (if lam = 0)
%    .ospry : standard errors of opry
%      .npr : number of forecasts
%       .cw : critical value of the standard normal distribution used in
%             computation of confidence bounds
%    .tname : name of the time series
%        .s : frequency of the data
%      .lam = 0 : compute logs of yor
%           = 1 : do not compute logs of yor
```

209 pleft2rightcmfd

```
function [phir,thetar,kro,ierror] = pleft2rightcmfd(phi,theta,np,kro)
%
% This function computes a right coprime MFD given a left MFD. That is,
% given phi^{-1}(z)*theta(z), a right coprime MFD  is computed such
% that thetar(z)*phir^{-1}(z) = phi^{-1}(z)*theta(z).
% It is assumed that phi(z) is square and that phi(0) is nonsingular.
% Polynomial matrix theta(z) can be nonsquare and, therefore, theta(0)
% is not assumed to be the identity matrix.
%---------------------------------------------------
% USAGE: [phir,thetar,kro,ierror] = pleft2rightcmfd(phi,theta,np,kro)
% where:    phi   = a k x k polynomial matrix with phi(0) nonsingular
%           theta = a k x m polynomial matrix
%           kro   = a 1 x k vector containing the Kronecker indices
%           np   = un upper bound for the Kronecker indices
%---------------------------------------------------
% RETURNS:
%           phir  = the AR echelon polynomial matrix
%          thetar = the MA echelon polynomial matrix
%           kro   = a 1 x k vector containing the Kronecker indices
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
% If kro is not input, the function uses functions housref and nullref
% on the augmented Sylvester matrices constructed with phi and theta to
% compute the Kronecker indices. If kro is input, a system of linear
% equations based on an appropriate augmented Sylvester matrix is
% solved.
```

210 plotres

```
function plotres(y,Y,g,yor,datei,cw,fname,gflag,nrout,Youtg,nreg,Yrg,infr,s,lam)
%*****************************************************************
% This function plots original series, residuals, outliers, regression
% variables, residual histogram, and correlograms of residuals and
% squared residuals.
```

```
%
%     INPUTS:
%         y : data vector
%         Y : matrix with regression variables
%         g : vector with regression coeffients
%       yor : original time series
%     datei : calendar structure
%        cw : critical value of the standard normal distribution to
%             compute confidence bounds
%     fname : series label appearing in the legend
%     gflag = 1 : pause between figures
%             0 : no pause
%     nrout = 0 : do not produce graph of outlier effects
%           > 0 : produce graph for each outlier effect
%     Youtg : matrix with outlier effects
%      nreg = 0 : do not produce graph for the effects of regression
%                 variables other than outliers
%           > 0 : produce graph for the effects of each regression
%                 variable other than outlier
%       Yrg : matrix with effects of regression variables other than
%             outliers
%      infr : residual structure (output of rescomp)
%         s : frequency of the data
%       lam = 0 : compute logs of the original series
%           = 1 : do not compute logs
```

211 plotspcd

```
function plotspcd(outa)
%
% function to plot the spectra of the canonical decomposition of an
% ARIMA model previously identified with function arimaestos.
%
% phi(B)*phi_s(B^s)*(delta*delta_s*y_t -mu) =
% th(B)*th_s(B^s)*a_t
```

212 pmatmul

```
function [C, ierror] = pmatmul(A,B)
%
% This function computes the product of the polynomial matrices A and B
%
```

213 pmattrans

```
function At = pmattrans (A)
%
% This function computes the transpose of a polynomial matrix A
%----------------------------------------------------
% USAGE: At = ptransmat (A)
% where:    A = a polynomial matrix not necessarily square
%----------------------------------------------------
% RETURNS:
%           At = the transpose of A
%----------------------------------------------------
```

214 pmattrian

```
function [T,U,ierrpmatri] = pmattrian(A, full, iHerm)
% This function computes T, a lower triangular form of a full column
% rank polinomial matrix A. U is a unimodular matrix such that AU = T
% and rk = rank(A) can be used on output to check if A really had full
% rank. If it had not, on output T and U are empty. See Henrion and
% Sebek (1999) Reliable Numerical Methods for Polynomial Matrix
% Triangularization, IEEE Trans. Aut. Control 44-3 pp. 497-508
% Author Felix Aparicio-Perez, Instituto Nacional de Estadistica, Spain
```

215 pmmulbf

```
function X=pmmulbf(A,B);
%  multiplies matrix pol. in z (A0+A1*z+...)by matrix pol.in z^{-1}
% (B0+B1*z^{-1}+...)'
% in output:X(1) is the coeff. of z^{-1}(max)
```

216 pmspectfac

```
function [Omega,Theta,ierror,iter,normdif]=pmspectfac(Lp,niter,lim)
%
% This function computes the spectral factorization
%   Lp(z,z^{-1})=Theta(z)*Omega*Theta'(z^{-1}),
% where Theta(0)=I.
%This is achieved by solving the similar problem
%   Lp(z,z^{-1})=phi(z)*phi'(z^{-1}),
% where phi is a polynomial matrix,
%   phi(z)=phi_0+phi_1*z+ ... + phi_p*z^p,
% such that phi_0 is a lower triangular matrix, Lp is a symmetric
% Laurent polynomial matrix of the form
%   Lp(z,z^{-1})= L'_pz^{-p}+.....+L'_1z^{-1}+L_0+L_1z+...L_pz^p,
% and L_0 is a symmetric, positive definite, matrix.
% The solution is found using Newton's method, iterating in the
% symmetric polynomial matrix equation
%    X(z)phi'(z^{-1}) + phi(z)X'(z^{-1}) = 2*Lp(z,z^{-1}),
% where p=degree(phi(z))= degree(X(z)), and the starting value for phi
% is phi_0=C*C', where C is a lower triangular matrix such that
% C*C'=L_0, and phi_i=0, i=1,...,p.
%
% Input:  Lp  = [n,n,p+1] matrix containing L_0+L_1z+...L_pz^p
%       niter = maximum number of iterations (default 10)
%        lim  = coefficient error limit for convergence
%              max(|phi(z)*phi'(z^{-1})-Lp|) < lim (default 1e-6)
% Output: Theta = [n,n,p] matrix containing the solution Theta(z)
%                 without Theta(0)=I
%         Omega = n x n symmetric, positive definite
%        ierror = 0,1  a flag for errors in dimensions
%          iter = number of needed iterations
%       normdif = norm of the difference upon convergence
```

217 poldiv

```
function y=poldiv(b,a,n)
%psi(B)=psi0+psi1*B+...=(b0+b1*B+...)/(a0+a1*B+...)
%dimension of psi=n+1
```

218 postmulW

```
function [C,ierror] = postmulW(A,k)
%
% This function computes the product of the matrix A by W
%
```

219 pr2ecf

```
function [Lambda,alpha,betap,th,Th,L,ferror] = pr2ecf(xv,xf,DA,str)
% PURPOSE: given a structure containing information about a VARMA model
% in error correction form, it obtains the model
```

```
%-----------------------------------------------------
% USAGE: [Lambda,alpha,betap,th,Th,L,ferror] = pr2ecf(xv,xf,DA,str)
% where:
%          xv       = a vector containing the parameters to be estimated
%          xf       = a vector containing the fixed parameters
%                     be estimated, =0, not
%          DA       = the matrix containing the parameterization of the
%                     differencing polynomial
%          str      = a structure containing the initial model
%                     information
%-----------------------------------------------------
%-----------------------------------------------------
% RETURNS:
%          Lambda   = a polynomial matrix of degree p-1, where p is the
%                     degree of the overall AR matrix polynomial
%          alpha    = an (s x r) matrix, where s is series dimension
%                     and r is the cointegration rank
%          betap    = an (r x s) matrix
%          th       = the regular MA matrix polynomial
%          Phi      = the seasonal AR matrix polynomial
%          Th       = the seasonal MA matrix polynomial
%          L        = the Cholesky factor of the innovations covariance
%                     matrix
%          ferror   = flag for errors
%-----------------------------------------------------
```

220 pr2usm

```
function [X,Z,G,W,T,H,ins,i,ferror] = pr2usm(xx,xf,str)
% PURPOSE: given a structure containing information about a univariate
% structural model, it passes the parameters to the state space form
%-----------------------------------------------------
% USAGE: [X,Z,G,W,T,H,ins,i,ferror] = pr2usm(xx,xf,str)
% where:
%          xx       = a vector containing the estimated parameters
%          xf       = a vector containing the fixed parameters
%          str      = a structure containing the initial model
%                     information
%-----------------------------------------------------
%-----------------------------------------------------
% RETURNS: updated matrices and initial conditions of the state space
%          form of a univariate structural model:
```

```
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%          More specifically:
%          X : an (n x nbeta) matrix containing the X_t matrices;
%              a  (1 x nbeta) if it is time invariant;
%              it can be []
%          Z : an (n x nalpha) matrix containing the Z_t matrices;
%              a  (1 x nalpha) matrix if it is time invariant
%          G : an (n x nepsilon) matrix containing the G_t matrices;
%              a  (1 x nepsilon) matrix if it is time invariant
%          W : an (n*nalpha x nbeta) matrix containing the W_t matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%          T : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%          H : an (n*nalpha x nepsilon) matrix containing the H_t %              matrices
%              an (nalpha x nepsilon) if it is time invariant
%        ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%              initial
%              state information, according to array i below
%          i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%              cc   = nalpha if c is not missing (0 if c missing)
%              cw0  = number of columns in W_0 (0 if W_0 missing)
%              ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%              cca1 = number of columns in A_1 (0 if A_1 missing)
%     ferror : flag for errors
%---------------------------------------------------
```

221 pr2usmm

```
function [X,Z,G,W,T,H,ins,ii,ferror] = pr2usmm(xx,xf,str)
```

```
% PURPOSE: given a structure containing information about a univariate
% structural model, it passes the parameters to the state space form
%---------------------------------------------------
% USAGE: [X,Z,G,W,T,H,ins,ii,ferror] = pr2usm(xx,xf,str)
% where:
%         xx       = a vector containing the estimated parameters
%         xf       = a vector containing the fixed parameters
%         str      = a structure containing the initial model
%                    information
%---------------------------------------------------
%---------------------------------------------------
% RETURNS: updated matrices and initial conditions of the state space
%          form of a univariate structural model:
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%          More specifically:
%          X : an (n x nbeta) matrix containing the X_t matrices;
%              a  (1 x nbeta) if it is time invariant;
%              it can be []
%          Z : an (n x nalpha) matrix containing the Z_t matrices;
%              a  (1 x nalpha) matrix if it is time invariant
%          G : an (n x nepsilon) matrix containing the G_t matrices;
%              a  (1 x nepsilon) matrix if it is time invariant
%          W : an (n*nalpha x nbeta) matrix containing the W_t matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%          T : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%          H : an (n*nalpha x nepsilon) matrix containing the H_t %              matrice
%              an (nalpha x nepsilon) if it is time invariant
%        ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%              initial
%              state information, according to array i below
%         i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%             cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%    ferror : flag for errors
%---------------------------------------------------
```

222 pr2varmapqPQ

```
function [phi,th,Phi,Th,L,ferror] = pr2varmapqPQ(xv,xf,str)
% PURPOSE: given a structure containing information about a VARMA
% model, it passes the parameters to the state space form.
%---------------------------------------------------
% USAGE: [phi,th,Phi,Th,L,ferror] = pr2varmapqPQ(xv,xf,str)
% where:
%         xv       = a vector containing the parameters to be estimated
%         xf       = a vector containing the fixed parameters
%                    be estimated, =0, not
%         str      = a structure containing the initial model
%                    information
%---------------------------------------------------
%---------------------------------------------------
% RETURNS:
%         phi      = the regular AR matrix polynomial
%         th       = the regular MA matrix polynomial
%         Phi      = the seasonal AR matrix polynomial
%         Th       = the seasonal MA matrix polynomial
%         L        = the Cholesky factor of the innovations covariance
%                    matrix
%         ferror   = flag for errors
%---------------------------------------------------
```

223 pr2varmapqPQd

```
function [yd,xvv,xff,DA,Dr,Ds,ferror] = pr2varmapqPQd(y,xv,xf,str)
% PURPOSE: given a structure containing information about a VARMA model
% with unit roots, it passes the parameters to the state space form.
%---------------------------------------------------
% USAGE: [yd,xvv,xff,DA,Dr,Ds,ferror] = pr2varmapqPQd(y,xv,xf,str)
% where:   y       = an (n x neqs) matrix containing the data
%          xv       = a vector containing the parameters to be estimated
%          xf       = a vector containing the fixed parameters
%                     be estimated, =0, not
%          str      = a structure containing the initial model
%                     information
%----------------------------------------------------
%----------------------------------------------------
% where:   yd     = the differenced series
%          xvv    = the variable paramaters for VARMA model
%          xff    = the fixed paramaters for VARMA model
%          DA     = the matrix containing the parameterization of the
%                   differencing polynomial
%          Dr     = the regular 'differencing' polynomial
%          Ds     = the seasonal 'differencing' polynomial (not used)
%          ferror   = flag for errors
%----------------------------------------------------
```

224 predt

```
function [pr,spr]=predt(n,npr,str,Y,Z,T,H,AA,Sigma,g,M)
%
% this function computes npr forecasts of an ARIMA model and their mse
% using the Kalman filter. The state space model is
%
%    x_{t} = T x_{t-1} + H a_{t}
%    y_{t} = Y_{t} \beta + Z x_{t},
%
%    where he initial state vector is
%
%        x_{d+1} = A*\delta + \Xi*c,
%
% See Gomez and Maravall (1994), "Estimation,
% Prediction and Interpolation for Nonstationary Series with the
% Kalman Filter", Journal of the American Statistical Association,
% 89, 611-624. The filter is initialized at time t = d+1, where d is
% the differencing degree, and the first d observations are stacked
% to form the \delta vector.
%
% Input arguments:
% n  : the series length
% npr: the number of forecasts
% str: estimated standard deviation of the residuals
% Y: matrix containing regression variables
% Z: the Z matrix
% T: the T matrix
% H: the H matrix
% g: array containing the regression estimates
% M: matrix containing the mse of the regression estimates
% AA: the estimated augmented state vector at the end of filtering
% Sigma: the Mse of A at the end of filtering
%
% Output arguments:
% pr : array containing the forecasts
% spr: array containing the ms
```

225 preres

```
function [Z,T,H,A,Sigma] = preres(x,s,pr,ps,qr,qs)
%
%
%  This function computes the system matrices corresponding to a
%  stationary ARMA model. The state space model is
%
%    x_{t} = T x_{t-1} + H a_{t}
%    y_{t} = Z x_{t},
%
%    where Var(x_{1}) = Sigma. Given an ARIMA model,the initial state
%    vector is
%
%        x_{d+1} = A*\delta + \Xi*c,
%
% and Var(c) = Sigma. See Gomez and Maravall (1994), "Estimation,
% Prediction and Interpolation for Nonstationary Series with the
% Kalman Filter", Journal of the American Statistical Association,
% 89, 611-624. The filter is initialized at time t = d+1, where d is
% the differencing degree, and the first d observations are stacked
% to form the \delta vector. In this case, because d = 0, x_1 = c;

%
%        INPUTS:
%        x: array containing model parameters
%        s:  number of seasons
%       pr:  AR order
%       ps: order of the AR of order s
%       qr:  order of the regular MA
%       qs: order of the MA of order s
%
%        OUTPUTS:
%        Z: the Z matrix
%        T: the T matrix
%        H: the H matrix
%        A: the empty matrix because the series is stationary
%    Sigma: the Sigma matrix
```

226 pright2leftcmfd

```
function [phie,thetae,kro,ierror] = pright2leftcmfd(phir,thetar,np,kro)
%
% This function computes a left coprime MFD given a right MFD. That is,
% given thetar(z)*phir^{-1}(z), a left coprime MFD  is computed such
% that phie^{-1}(z)*thetae(z) = thetar(z)*phir^{-1}(z).
% It is assumed that phir(z) is square and that phir(0) is nonsingular.
% Polynomial matrix thetar(z) can be nonsquare and, therefore,
% thetar(0) is not assumed to be the identity matrix.
%---------------------------------------------------
% USAGE: [phie,thetae,kro,ierror] = pright2leftcmfd(phir,thetar,np,kro)
% where:    phir   = a k x k polynomial matrix with phir(0) nonsingular
%           thetar = a k x m polynomial matrix
%           np     = un upper bound for the Kronecker indices
%           kro    = a 1 x k vector containing the Kronecker indices
%---------------------------------------------------
% RETURNS:
%           phie  = the AR echelon polynomial matrix
%          thetae = the MA echelon polynomial matrix
%           kro   = a 1 x k vector containing the Kronecker indices
%        ierror =1, dimension mismatch in phir and thetar
%               =0, there are no errors on input
%---------------------------------------------------
% If kro is not input, the function uses functions housref and nullref
% on the augmented Sylvester matrices constructed with phir and thetar
% to compute the Kronecker indices. If kro is input, a system of linear
% equations based on an appropriate augmented Sylvester matrix is
% solved.
```

227 printres

```
function printres(fid,infr)
%*************************************************************************
% This function prints test results based on residuals
%
%   INPUTS:
%     fid : file identifier, needed for writing
%    infr : residuals structure (output of rescomp)
```

228 printusmer

```
function  printusmer(fid,datei,tname,yor,y,ny,lam,modescr,result,nreg,nbeta)
%*************************************************************************
% This function prints the estimation results of a univariate structural
% model
%
%    INPUTS:
%      fid : file identifier, needed for writing the output into text
%            file
%    datei : calendar structure
%    tname : name of the series (string variable)
%      yor : original time series
%        y : time series used in the estimation etc.
%       ny : length of y
%      lam = 0 : compute logs of y
%          = 1 : do not compute logs
%  modescr : structure containing model information (output of suusm)
%   result : structure with the estimation results (output of usmestim)
%     nreg : number of regression variables in the observation equation;
%    nbeta : number of regression coefficients in the state space model;
%            number of columns of the matrices X and W
%
% Note: nreg corresponds to the number of nonzero columns of X and
%       nbeta to the number of the columns of X and W, where
%       X is an (n x nbeta) matrix containing the X_t matrices;
%         an(1 x nbeta) matrix if it is time invariant; it can be []
%       W is an (n*nalpha x nbeta) matrix containing the W_t matrices;
%         an (nalpha x nbeta) matrix if it is time invariant; it can be
%         []
%       and
%       X_t and W_t are matrices of the state space model:
```

```
%
%           y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%           alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%           where epsilon_t is (0,sigma^2I),
%
%           with initial state
%
%           alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%           where c is (0,Omega) and delta is (0,kI) (diffuse)
```

229 printusmerm

```
function  printusmerm(fid,datei,tname,yor,y,ny,lam,modescr,result,nreg,nbeta)
%*********************************************************************
% This function prints the estimation results of a univariate structural
% model with complex seasonal patterns
%
%    INPUTS:
%      fid : file identifier, needed for writing the output into text
%            file
%    datei : calendar structure
%    tname : name of the series (string variable)
%      yor : original time series
%        y : time series used in the estimation etc.
%       ny : length of y
%      lam = 0 : compute logs of y
%          = 1 : do not compute logs
%  modescr : structure containing model information (output of suusm)
%   result : structure with the estimation results (output of usmestim)
%     nreg : number of regression variables in the observation equation;
%    nbeta : number of regression coefficients in the state space model;
%            number of columns of the matrices X and W
%
% Note: nreg corresponds to the number of nonzero columns of X and
%       nbeta to the number of the columns of X and W, where
%       X is an (n x nbeta) matrix containing the X_t matrices;
%         an(1 x nbeta) matrix if it is time invariant; it can be []
%       W is an (n*nalpha x nbeta) matrix containing the W_t matrices;
%         an (nalpha x nbeta) matrix if it is time invariant; it can be
%         []
%       and
%       X_t and W_t are matrices of the state space model:
%
%           y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%           alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%           where epsilon_t is (0,sigma^2I),
%
%           with initial state
%
%           alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%           where c is (0,Omega) and delta is (0,kI) (diffuse)
```

230 prmod1x

```
function prmod1x(fid,s,p,dr,q,ps,ds,qs,S,dS,qS,lam,flagm)
%
% this function prints in the file fid the ARIMA model specification of
% the form (p,dr,q) (ps,ds,qs)_s (0,dS,qS)_S, adding information as to
% whether logs have been taken and whether a mean has been included.
```

231 prmod2x

```
function prmod2x(fid,s,p,dr,q,ps,ds,qs,S,dS,qS,lam,flagm)
%
% this function prints in the file fid the ARIMA model specification of
% the form (p,dr,q) (ps,ds,qs)_s (0,dS,qS)_S, adding information as to
% whether logs have been taken and whether a mean has been included.
```

232 prmod11x

```
function prmod11x(fid,s,p,dr,q,ps,ds,qs,S,dS,qS,lam,flagm)
%
% this function prints in the file fid the ARIMA model specification of
% the form (p,dr,q) (ps,ds,qs)_s (0,dS,qS)_S, adding information as to
% whether logs have been taken and whether a mean has been included.
```

233 prsummry

```
function prsummry(ii,ny,nreg0,fid,fname,parm,iout)
%
% this function prints the automatic model identification summary for a
% list of ARIMA or transfer function models.
%
%     INPUTS:
%     ii     : an integer, corresponding to the series currently
%              handled.
%     ny     : the series length
%     nreg0  : the number of original regression variables
%     fid    : the number of the output file
%     fname  : a string containing the series name
% parm: astructure containing model infomation, where
% .s:  seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .lam: = 0, logs are taken, = 1, no logs
%.flagm: = 0, no mean, = 1, mean in the model
%.trad : the estimated number of TD variables
%.leap : = 0, no leap year effect, = 1, leap year effect in the model
%.east : = 0, no Easter effect, = 1, Easter effect in the model
%.dur  : duration of the Easter effect
% iout:   a structure containing information for outlier detection,
%         where
% .C:     critical value for outlier detection
% .delta: the value for delta in TC outliers
% .mthd: method to compute ARMA parameter estimates (0 Hannan Rissanen,
%        1 Max. Lik.)
% .schr: =0 outliers of type AO and TC are considered (default)
%        =1 outliers of type AO, TC and LS are considered
```

234 prtransfer

```
function [K, ierror] = prtransfer ( A, B, n)
%
% This function computes the right transfer function
% K(z)=B(z)*A^{-1}(z) up to the (n-1)-th term, that is, K(0),...,K(n-1)
% It is assumed that A(z) is square and that A(0) is nonsingular.
% Polynomial matrix B(z) can be nonsquare and, therefore, B(0) is not
% assumed to be the identity matrix.
%---------------------------------------------------
% USAGE: [K, ierror] = prtransfer ( A, B, n)
% where:    A = a square polynomial matrix with A(0) nonsingular
%           B = a polynomial matrix not necessarily square
%---------------------------------------------------
% RETURNS:
%           K = the first n weights of K(z)=B(z)*A^{-1}(z)
%        ierror =1, dimension mismatch in A and B
%               =0, there are no errors on input
%---------------------------------------------------
```

235 prtser

```
function prtser(fid,fname,yor,y,ny,datei,inft,lam)
%*********************************************************************
% This function prints time series into text file
%
%    INPUTS:
%      fid : file identifier, needed for writing the output into text
%            file
%    fname : name of the series
%      yor : original time series
%        y : time series used in the estimation etc.
%       ny : number of observations
%    datei : calendar structure
%     inft : structure containing printing options
%      .fh :  flag for header and years
%      .wd : format width
%      .nd : number of decimal points
%   .scale = 1 : scale data if necessary
%          = 0 : do not scale data
%      lam = 0 : compute logs of y
%          = 1 : do not compute logs
```

236 ptransfer

```
function [K, ierror] = ptransfer ( A, B, n)
%
```

```
% This function computes the transfer function K(z)=A^{-1}(z)*B(z) up to
% the (n-1)-th term, that is, K(0),...,K(n-1)
% It is assumed that A(z) is square and that A(0) is nonsingular.
% Polynomial matrix B(z) can be nonsquare and, therefore, B(0) is not
% assumed to be the identity matrix.
%---------------------------------------------------
% USAGE: [K, ierror] = ptransfer ( A, B, n)
% where:    A = a square polynomial matrix with A(0) nonsingular
%           B = a polynomial matrix not necessarily square
%---------------------------------------------------
% RETURNS:
%           K = the first n weights of K(z)
%        ierror =1, dimension mismatch in A and B
%               =0, there are no errors on input
%---------------------------------------------------
```

237 pu2ma

```
function [th, sigma2, ierrpu2ma] = pu2ma(p)
% This function obtains the moving average part corresponding
% to a finite covariance generating function that has been
% transformed into a polynomial in the variable U=z + z^(-1)
```

238 qarmax2ss1

```
function [H,F,G,J,ierror] = qarmax2ss1(phi,theta)
%
%---------------------------------------------------
% This function puts the armax model into Akaike's state space form
%   x(t+1) = F*x(t) + G*u(t)
%    y(t)  = H*x(t) + J*u(t),
% where
%            [0 I 0  ... ....  0]         [ Psi_1    ]
%            [0 0 I  ... ....  0]         [ Psi_2    ]
%     F =    [ ...   ...  ...   ],   G = [ ...      ],
%            [0 0 0  ... ....  I]         [ Psi_{r-1}]
%            [-bphi_r ... -bphi_1]        [ Psi_r    ]
%     H =    [I 0 0  ...  ... 0],   J =  Psi_0,
% bphi_i = phi_0^{-1}*phi_i and phi^{1}(z)*theta(z) = Psi_0 + Psi_1*z
%  + Psi_2*z^2+ ..., and r = max{degree(phip), degree(theta)}.

% USAGE: [H,F,G,J,ierror] = qarmax2ss(phi,theta)
% where:    phi   = a matrix polynomial with phi(0) nonsingular
%           theta = a matrix polynomial not necessarily square
%---------------------------------------------------
% RETURNS:
%           H = a k x n matrix
%           F = an n x n matrix
%           G = a n x m matrix
%           J = a k x m matrix
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
```

239 qarmax2ss2

```
function [H,F,G,J,ierror] = qarmax2ss2(phi,theta)
%
%---------------------------------------------------
% USAGE: [H,F,G,J,ierror] = qarmax2ss(phi,theta)
% where:    phi   = a matrix polynomial with phi(0) nonsingular
%           theta = a matrix polynomial not necessarily square
% This function puts the armax model into a state space form
%   x(t+1) = F*x(t) + G*u(t)
%    y(t)  = H*x(t) + J*u(t),
% where
%       [-bphi_1     I 0  ... .... 0]      [theta_1-phi_1*Psi0        ]
%       [-bphi_2     0 I  ... .... 0]      [theta_2-phi_2*Psi0        ]
% F =   [  ...            ...  ... ], G = [   ...                     ],
%       [-bphi_{r-1} 0 0  ...  ... I]      [theta_{r-1}-phi_{r-1}*Psi0]
%       [-bphi_r     0 0  ...  ... 0]      [theta_r-phi_r*Psi0        ]
% H =    [phi_0^{-1} 0 0  ...  ... 0],  J =  Psi_0,
% bphi_i = phi_i*phi_0^{-1} and phi^{-1}(z)*theta(z) = Psi_0 + Psi_1*z
%          + Psi_2*z^2+ ...
%---------------------------------------------------
% RETURNS:
%           H = a k x n matrix
%           F = an n x n matrix
%           G = a n x m matrix
%           J = a k x m matrix
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
```

240 qarmax2ss12

```
function [H,F,G,ierror] = qarmax2ss12(phi,theta)
%
%---------------------------------------------------
% USAGE: [H,F,G,ierror] = qarmax2ss(phi,theta)
% where:    phi   = a matrix polynomial with phi(0) nonsingular
%           theta = a matrix polynomial not necessarily square
% This function puts the armax model into Akaike's state space form
%   x(t+1) = F*x(t) + G*u(t)
%    y(t)  = H*x(t),
% where
%            [0 I 0  ... ....   0]        [ Psi_0    ]
%            [0 0 I  ... ....   0]        [ Psi_1    ]
%     F =    [ ...   ...  ...    ],   G = [ ...      ],
%            [0 0 0  ... ....   I]        [ Psi_{r-2}]
%            [-bphi_r ... -bphi_1]        [ Psi_{r-1}]
%     H =    [I 0 0  ...  ... 0],
% bphi_i = phi_0^{-1}*phi_i, phi^{1}(z)*theta(z) = Psi_0 + Psi_1*z
% + Psi_2*z^2+ ..., and r = max{degree(phip), degree(theta)+1}.
%---------------------------------------------------
% RETURNS:
%           H = a k x n matrix
%           F = an n x n matrix
%           G = a n x m matrix
%           J = a k x m matrix
%        ierror =1, dimension mismatch in phi and theta
%               =0, there are no errors on input
%---------------------------------------------------
```

241 qtb

```
function QtB=qtb(A,B,p,q)
% given the output matrix A of function jqrt containing the
% J-unitary Housholder transformations and a matrix B, this
% function computes the product of Q'*B
%
%---------------------------------------------------
% USAGE: QtB=qtb(A,B,p,q)
% where:    A = an m x n matrix, m >= n
%           B = mb x nb matrix
%         p,q = integers such that J = diag(I_p,-I-q) is a signature
%               matrix with n=p+q
%---------------------------------------------------
% RETURNS: the product Q'*B
%---------------------------------------------------
```

242 rbols

```
function res=rbols(y,Y)
%*********************************************************************
% This function computes the OLS residuals
%
%   INPUTS:
%       y : data vector
%       Y : matrix with regression variables
%
%   OUTPUT:
%     res : residuals vector
```

243 rescomp

```
function infr = rescomp(e,lag,nr,Ss,conp,sconp,Ff,ndrs,nreg)
%*********************************************************************
%        This function generates a structure
%        containing some information on the
%        residuals
%
%    INPUTS:
%        e : residuals vector
%      lag : integer specifying lag up to which autocorrelations are to
%            be computed
%       nr : number of parameters to be estimated
%       Ss : residual sum of squares
%     conp : prediction error variance
%    sconp : square root of conp
%       Ff : the product F'*F, where F is the vector of nonlinear
%            functions whose sum of squares is minimized at the end of
%            estimation
%     ndrs : length of the series minus the number of nonstationary
%            components
%     nreg : number of regression variables
%
%   OUTPUT :
```

```
%     infr : residuals structure containing the following fields:
%       .e : residuals vector
%      .ne : length of e
%      .ve : residuals variance
%    .stde : residuals standard deviation
%    .conp : residual sum of squares
%   .sconp : square root of conp
%  .orders : vector with integers specifying at which lags
%            autocorrelationsare to be computed; values between 1 and
%            lag
%       .r : autocorrelations
%      .pc : partial autorrelations
%   .qstat : Q-statistics based on residuals
%    .pval : p-values of the Q-statistics based on residuals
%      .df : degrees of freedom for the Q-statictics based on residuals
%     .sea : standard errors of autocorrelations
%     .sep : standard error of partial correlations;
%            value equal to 1/sqrt(ne)
%      .no : bins of the residuals histogram
%      .xo : vector of cut points where observations counted in bin(i)
%            are cutpnt(i-1) < y <= cutpnt(i)
%    .hot0 : list of residuals greater than 3.25 standard deviations
%            from the median
%      .ho : list of the values of residuals greater than 3.25
%      .me : mean of e
%    .rstd : standard deviation of the mean of e
%   .rtval : t-value of the mean of e
%    .maxe : maximum value of e
%     .mde : median value of e
%    .mine : minimum value of e
%    .skew : skewness
%    .kurt : kurtosis
%     .bst : Bowman-Shenton normality statistic
%     .pnt : p-value of the Bowman-Shenton statistic
%     .tsk : p-value of skewness
%     .tkr : p-value of kurtosis
%      .dw : Durbin-Watson statistic
%     .tdw : t-value of the Durbin-Watson statistic
%    .ptdw : p-value of the Durbin-Watson statistic
%      .n0 : number of residuals lower than the median
%      .n1 : number of residuals higher than the median
%      .nr : number of runs on residuals
%    .Tval : t-value of the number of runs on residuals
%      .rs : autocorrelations of squared residuals
%     .pcs : partial correlations of squared residuals
%  .qstats : Q-statistics based on squared residuals
%   .pvals : p-values of the Q-statistics based on squared residuals
%     .dfs : degrees of freedom for the Q-statistics based on squared
%            residuals
%    .seas : standard errors associated with squared residuals
%       .h : closest integer to ne/3 needed in the computation of H;
%            degrees of freedom of the F-distribution
%       .H : heteroskedasticity statistic
%      .pH : p-value of the heteroskedasticity statistic
%     .aic : Akaike information criterion
%     .bic : Bayes information criterion
%    .mser : mean squared error of residuals
%   .stder : standard error of residuals
```

244 residual2x

```
function [F,e,g,M,A,P,matsis] = residual2x(x,y,Y,s,S,dr,ds,dS,pr,ps,qr,qs,qS)
%
%
%        This function evaluates the residuals for
%        the nonlinear minimization of the sum of
%        squares of an ARIMA model with two possible seasonalities
%
%        INPUTS:
%        x: an array containing model parameters
%        y: an array containing the input series
%        Y: a matrix containing regression variables
%        s:  seasonality
%        S:  second seasonality
%        p:  AR order
%       ps: order of the AR of order s
%        q:  order of the regular MA
%       qs: order of the MA of order s (1 at most)
%       qS: order of the MA of order S (1 at most)
%       dr: order of regular differencing
%       ds: order of differencing of order s
%       dS: order of differencing of order S
%
%        OUTPUTS:
%        F: residual vector, whose sum of squares will be minimized
%        e: residual vector for inference
%        g: array containing the regression estimates
%        M: matrix containing the mse of the regression estimates
%        A: the estimated augmented state vector at the end of filtering
%        P: the Mse of A at the end of filtering
%   matsis: a structure containing the system matrices
```

245 residual3

```
function [F,e] = residual3(x,y,Y,s,pr,ps,qr,qs)
%
%
%        This function evaluates the residuals for
%        the nonlinear minimization of the sum of
%        squares of an ARMA model using the CKMS recursions
%
% Input arguments:
% x: array containing model parameters
% y: vector containing the data
% Y: matrix containing regression variables
% s:  number of seasons
% pr:  AR order
% ps: order of the AR of order s
% qr:  order of the regular MA
% qs: order of the MA of order s
%
% Output arguments:
%  F: residual vector, whose sum of squares will be minimized
%  e: residual vector for inference
```

246 restrcmodel

```
function str = restrcmodel(s,m,seas,ordersr,orderss)
% PURPOSE: given the regular and seasonal orders of a VARMAX(p,q,r)
%          (P,Q,R)_seas model, this function creates a structure
%          containing the polynomial and state space forms of the model
%          with NaNs for the parameters to be estimated and zero
%          otherwise.
%---------------------------------------------------
% USAGE:  str = restrcmodel(s,m,seas,ordersr,orderss)
% where:    s       = an integer, the dimension of the output y_t
%           m       = an integer, the dimension of the input  x_t
%        seas       = seasonality
%        ordersr    = a 1 x 3 array containing the regular VARMAX orders
%        orderss    = a 1 x 3 array containing the seasonal VARMAX
%                      orders
%---------------------------------------------------
% RETURNS: str, a structure containing model information. The fields of
%          str are the same than those in the structure returned by
%          function matechelon, but with the zero restrictions of the
%          VARMAX model imposed. The VARMAX model is assumed to follow
%          an VARMAX model in echelon form with all Kronecker indices
%          equal to max(p,q,r) + seas*max(P,Q,R) and with the zero
%          restrictions imposed.
%---------------------------------------------------
```

247 rootsarma

```
function z=rootsarma(x,parm)
%
% given an ARMA model, this function computes a matrix containing the
% roots of the AR and MA polynomials, their arguments and their periods.
%
% Input arguments:
% x    : array containing model parameters
% parm : a structure where
% .s:  seasonality
% .S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
% .ninput: number of inputs
% .delay: array with the delays of the input filters
% .ma: array with the ma parameters of the input filters
% .ar: array with the ar parameters of the input filters
%
% Output arguments:
```

```
% z: an array with four columns containing the real and imaginary parts
% of the roots, their arguments and their periods
```

248 rpplot

```
function rpplot(r,p,sea,sep,c,fname)
%*********************************************************************
% This function creates plots of sample autocorrelations and sample
% partial correlations
%
%    INPUTS:
%        r : autocorrelations
%        p : partial autocorrelations
%      sea : standard errors of autocorrelations
%      sep : standard error of partial correlations
%        c : critical value of the standard normal distribution
%    fname : label used in the legend
```

249 runcom

```
function [n0,n1,nr,Tval] = runcom(X,N,Xmed)
%*********************************************************************
%This function computes the elements of a test of randomness based on
% runs.
%
%   INPUTS:
%       X : data vector
%       N : length of X
%    Xmed : median of X
%
%  OUTPUTS:
%      n0 : number of values of X smaller than Xmed (-)
%      n1 : number of values of X grater or equal to Xmed (+)
%      nr : total number of runs, (-) and (+).
%    Tval : approximate t-value of nr
```

250 sacspacdif

```
function [c0s,cvs,rs,fis,pcs] = sacspacdif(y,tname,dr,ds,freq,lag,cw,fplot)
%*********************************************************************
%  This function outputs cv0s, cvs, rs, fis and pcs described below.
%  In addition, it optionally plots the series before and after
%  differencing, as well as the sample autocorrelations and the sample
%  partial correlations of the differenced series.
%
%    INPUTS :
%          y : series
%      tname : name of the series
%         dr : number of regular differences
%         ds : number of seasonal differences
%       freq : frequency of the data
%        lag : number of lags up to which cvs,rs,fis,pcs of the
%              differenced y are computed
%         cw : critical value of the standard normal distribution
%      fplot : =1 plot series (default), =0 do not plot series
%
%   OUTPUTS :
%        c0s : sample variance of differenced y
%        cvs : sample autocovariances of differenced y
%         rs : sample autocorrelations of differenced y
%        fis : an (1 x lag) vector containing the AR(lag) polynomial
%        pcs : sample partial correlations of differenced y
```

251 sarimac

```
function x=sarimac(p,ps,q,qs,phir,phis,thr,ths)
%***********************************************************************
% Auxiliary function called in arimasimul_d.m to set the ARIMA coefficients
%
%  INPUTS:
%           p:  AR order
%          ps:  order of the AR of order s
%           q:  order of the regular MA
%          qs:  order of the MA of order s (1 at most)
%     phir   : an array containing the regular AR polynomial
%     phis   : an array containing the seasonal AR polynomial
%     thr    : an array containing the regular MA polynomial
%     thr    : an array containing the seasonal MA polynomial
%
%  OUTPUTS:
%           x:  an array containing the ARIMA parameter values
```

252 scakff

```
function [KKP,PT,hd,Md,initf,recrs,recr,srecr]=scakff(y,X,Z,G,W,T,H,ins,i)
%
%
%        This function applies the augmented Kalman filter and smoother
%        to the series y corresponding to the model
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time
%               invariant; it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t
%               matrices;
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t
%               matrices;
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1],  where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%
%
%        Output parameters:
%        KKP : an (n x nalpha) matrix containing the estimated x_{t|t}
%        PT  : an (n*nalpha x nalpha) matrix containing the
%              Mse of x_{t|t}
%        hd  : the beta estimate
%        Md  : the Mse of hd
%       initf: flag to indicate when regression parameters are
%              identified
%       recrs: standardized recursive residuals
%        recr: recursive residuals
%       srecr: covariance matrices of recursive residuals
```

253 scakfff

```
function [KKP,PT,recrs,recr,srecr,t1,A1,P1,KG]=scakfff(y,X,Z,G,W,T,H,ins,i,g)
%
%
%        This function applies the augmented Kalman filter and smoother
%        to the series y corresponding to the model
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component. The
%        regression parameter vector, g, is considered fixed.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time
%               invariant; it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t
%               matrices;
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t
%               matrices;
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1],  where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%        g  : the beta vector, considered fixed
%
%        Output parameters:
%        KKP : an (n x nalpha) matrix containing the estimated x_{t|t}
%        PT  : an (n*nalpha x nalpha) matrix containing the
%              Mse of x_{t|t}
%       recrs: standardized recursive residuals
%        recr: recursive residuals
%       srecr: covariance matrices of recursive residuals
%          t1: initial time of the collapsed filter
%          A1: x_{t1|t1-1} initial state for the collapsed filter
%          P1: Mse(x_{t1|t1-1})
%          KG: stack of the Kalman gain vectors for the collapsed filter
```

254 scakfffsqrt

```
function [KKP,PT,recrs,recr,srecr,t1,A1,LP1,KG]=scakfffsqrt(y,X,Z,G,W,T,H,...
%                                                   ins,i,g,icollps)
```

```
%
%        This function applies the square root version of the two stage
%        Kalman filter to the series y corresponding to the model to
%        obtain the filtered estimator of the state vector, its Mse,
%        and recursive residuals. The regression coefficient, g, is
%        considered fixed.
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time
%               invariant; it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t
%               matrices;
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t
%               matrices;
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1],  where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%        g  : the beta vector, considered fixed
%   icollps= an integer, corresponding to the observation number in
%            which a collapse takes place
%
%        Output parameters:
%        KKP : an (n x nalpha) matrix containing the estimated x_{t|t}
%        PT  : an (n*nalpha x nalpha) matrix containing the
%              Mse of x_{t|t}
%       recrs: standardized recursive residuals
%        recr: recursive residuals
%       srecr: covariance matrices of recursive residuals
%          t1: initial time of the collapsed filter
%          A1: x_{t1|t1-1} initial state for the collapsed filter
%         LP1: Square root of Mse(x_{t1|t1-1})
%          KG: stack of the Kalman gain vectors for the collapsed filter
```

255 scakffsqrt

```
function [KKP,PT,hd,Md,initf,recrs,recr,srecr]=scakffsqrt(y,X,Z,G,W,T,H,...
%                                                ins,i,icollps)
%
%
%        This function applies the square root version of the two stage
%        Kalman filter to the series y corresponding to the model to
%        obtain the filtered estimator of the state vector, its Mse,
%        and recursive residuals.
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
```

```
%       Input parameters:
%       y:     an (n x p) matrix of observations;
%       X    : an (n*p x nbeta) matrix containing the X_t matrices;
%              a  (p x nbeta) if it is time invariant;
%              it can be []
%       Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%              a  (p x nalpha) matrix if it is time invariant
%       G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%              a  (p x nepsilon) matrix if it is time invariant
%       W    : an (n*nalpha x nbeta) matrix containing the W_t
%              matrices;
%              an (nalpha x nbeta) matrix if it is time
%              invariant; it can be []
%       T    : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%       H    : an (n*nalpha x nepsilon) matrix containing the H_t
%              matrices;
%              an (nalpha x nepsilon) if it is time invariant
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%            initial state information, according to array i below
%       i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1],  where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%   icollps= an integer, corresponding to the observation number in
%           which a  collapse takes place
%
%       Output parameters:
%       KKP : an (n x nalpha) matrix containing the estimated x_{t|t}
%       PT  : an (n*nalpha x nalpha) matrix containing the
%             Mse of x_{t|t}
%       hd  : the beta estimate
%       Md  : the Mse of hd
%      initf: flag to indicate when regression parameters are identified
%      recrs: standardized recursive residuals
%       recr: recursive residuals
%      srecr: covariance matrices of recursive residuals
```

256 scakfle2

```
function [e,f,hb,Mb,A,P,qyy,R,olsres]=scakfle2(y,X,Z,G,W,T,H,ins,i,chb)
%
%
%       This function applies the two stage Kalman filter to the series y
%       for prediction and likelihood evaluation corresponding to the
%       model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%       alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%       collapse is applied to get rid of the diffuse component.
%
%       Input parameters:
%       y:     an (n x p) matrix of observations;
%       X    : an (n*p x nbeta) matrix containing the X_t matrices;
%              a  (p x nbeta) if it is time invariant;
%              it can be []
%       Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%              a  (p x nalpha) matrix if it is time invariant
%       G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%              a  (p x nepsilon) matrix if it is time invariant
%       W    : an (n*nalpha x nbeta) matrix containing the W_t matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%       T    : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%       H    : an (n*nalpha x nepsilon) matrix containing the H_t
%              matrices;
%              an (nalpha x nepsilon) if it is time invariant
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%       chb= 1 compute hb and Mb
%            0 do not compute hb and Mb
%
%       Output parameters:
%       e    : residual vector (Q'_2*y)
%       f    : factor by which the residuals are to be multiplied
%              for minimization of the nonlinear sum of squares
%       hb   : the beta estimator
%       Mb   : the Mse of the beta estimator
%        A   : the estimated augmented state vector at the end of
%              filtering
%        P   : the Mse of A at the end of filtering
%      qyy   : Q'_1*y in the QR decomposition to estimate beta (chb=1)
%        R   : R in the QR decomposition to estimate beta      (chb=1)
%   olsres   : vector of OLS residuals                         (chb=1)
```

257 scakflepc

```
function [e,f,hb,Mb,A,P,qyy,R]=scakflepc(y,X,Z,G,W,T,H,ins,i,chb)
%
%
%       This function applies the two stage Kalman filter to the series
%       y for prediction and profile likelihood evaluation
%       corresponding to the model
%
%       y_t = X_t*beta + Z*alpha_t + G*epsilon_t
%       alpha_{t+1}= W_t*beta + T*alpha_t + H*epsilon_t,
%
%       where epsilon_t is (0,I),
%
%       with initial state
%
%       alpha_1 = delta
%
%       that is considered fixed and unknown. It is assumed that the
%       state space model is in innovations form and P_t = 0 for all t.
%       That is, G=Sigma^{1/2} and H=K*Sigma^{1/2}, where Sigma =
%       Sigma^{1/2}*Sigma^{1/2 '} is the Cholesky decomposition of the
%       covariance matrix of the innovations. Therefore, the recursions
%       are simplified and only the innovations and the state
%       estimators need to be updated. No collapsing takes place. No
%       missing values are allowed.
%
%       Input parameters:
%       y:     an (n x p) matrix of observations;
%       X    : a  (p x nbeta) if it is time invariant;
%              it can be []
%       Z    : a  (p x nalpha) matrix if it is time invariant
%       G    : a  (p x nepsilon) matrix if it is time invariant
%       W    : an (n*nalpha x nbeta) matrix containing the W_t matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%       T    : an (nalpha x nalpha) matrix if it time invariant
%       H    : an (nalpha x nepsilon) if it is time invariant
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1],  where
%            cc   = 0
%            cw0  = 0
%            ca1  = 0
%            cca1 = nalpha
%       chb= 1 compute hb and Mb
%            0 do not compute hb and Mb
%
%       Output parameters:
%       e   : residual vector (Q'_2*y)
%       f   : factor by which the residuals are to be multiplied
%             for minimization of the nonlinear sum of squares
%       hb  : the beta estimator
%       Mb  : the Mse of the beta estimator
%        A  : the estimated augmented state vector at the end of
%             filtering
%        P  : the Mse of A at the end of filtering
%      qyy  : Q'_1*y in the QR decomposition to estimate beta
%        R  : R in the QR decomposition to estimate beta
```

258 scakflesqrt

```
function [e,f,hb,Mb,A,LP,qyy,R]=scakflesqrt(y,X,Z,G,W,T,H,ins,i,chb,icollps)
```

```
%
%
%       This function applies the square root version of the two stage
%       Kalman filter to the series y for prediction and likelihood
%       evaluation corresponding to the model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%       alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%       collapse is applied to get rid of the diffuse component.
%
%       Input parameters:
%       y:     an (n x p) matrix of observations;
%       X    : an (n*p x nbeta) matrix containing the X_t matrices;
%              a  (p x nbeta) if it is time invariant;
%              it can be []
%       Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%              a  (p x nalpha) matrix if it is time invariant
%       G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%              a  (p x nepsilon) matrix if it is time invariant
%       W    : an (n*nalpha x nbeta) matrix containing the W_t matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%       T    : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%       H    : an (n*nalpha x nepsilon) matrix containing the H_t
%              matrices;
%              an (nalpha x nepsilon) if it is time invariant
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%       chb= 1 compute hb and Mb
%            0 do not compute hb and Mb
%   icollps= an integer, corresponding to the observation number in
%            which a collapse takes place
%
%       Output parameters:
%       e   : residual vector (Q'_2*y)
%       f   : factor by which the residuals are to be multiplied
%             for minimization of the nonlinear sum of squares
%       hb  : the beta estimator
%       Mb  : the Mse of the beta estimator
%        A  : the estimated augmented state vector at the end of
%             filtering
%        P  : the Mse of A at the end of filtering
%      qyy  : Q'_1*y in the QR decomposition to estimate beta
%        R  : R in the QR decomposition to estimate beta
```

259 scakfs

```
function [KKP,PT,hd,Md]=scakfs(y,X,Z,G,W,T,H,ins,i)
%
%
%        This function applies the augmented Kalman filter and smoother
%        to the series y corresponding to the model
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%                a  (p x nbeta) if it is time invariant;
%              it can be []
%       Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%              a  (p x nalpha) matrix if it is time invariant
%       G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%              a  (p x nepsilon) matrix if it is time invariant
%       W    : an (n*nalpha x nbeta) matrix containing the W_t
%              matrices;
%              an (nalpha x nbeta) matrix if it is time invariant;
%              it can be []
%       T    : an (n*nalpha x nalpha) matrix containing the T_t
%              matrices;
%              an (nalpha x nalpha) matrix if it time invariant
%       H    : an (n*nalpha x nepsilon) matrix containing the H_t
%              matrices;
%              an (nalpha x nepsilon) if it is time invariant
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%            initial state information, according to array i below
%       i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%
%
%       Output parameters:
%       KKP : an (n x nalpha) matrix containing the estimated x_{t|n}
%       PT  : an (n*nalpha x nalpha) matrix containing the
%             Mse of x_{t|n}
%       hd  : the (delta',beta')' estimate
%       Md  : the Mse of hd
```

260 scakfssqrt

```
function [KKP,PT,hd,Md]=scakfssqrt(y,X,Z,G,W,T,H,ins,i,icollps)
%
%
%       This function applies the square root version of the augmented
%       Kalman filter and smoother to the series y corresponding to
%       the model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time invariant;
%               it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t
%               matrices;
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t
%               matrices;
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%   icollps= an integer, corresponding to the observation number in
%            which a collapse takes place
%
%
%        Output parameters:
```

```
%        KKP : an (n x nalpha) matrix containing the estimated x_{t|n}
%        PT  : an (n*nalpha x nalpha) matrix containing the
%              Mse of x_{t|n}
%        hd  : the (delta',beta')' estimate
%        Md  : the Mse of hd
```

261 seasdm

```
function Y=seasdm(N,datei)
%
%
%       This function generates a matrix of seasonal dummies
```

262 seasdmom

```
function Y=seasdmom(N,datei)
%
%
%       This function generates a matrix of seasonal dummies
%       orothogonal to the mean
```

263 SEATSres

```
function [res,str] = SEATSres(out)
%
%
% This function obtains the residuals given by program SEATS by Gomez
% and Maravall (see Gomez, V., and Maravall, A., 2001. Programs TRAMO
% and SEATS, Instructions for the User (Beta Version: June 1997)
% (Working Paper No. 97001). Direccion General De Presupuestos,
% Ministry of Finance, Madrid, Spain.)
%
% input arguments:
% out: a structure, the output of function arimaestos
%
% output arguments:
% res: a vector containing the SEATS residuals
% str: a structure containing the pure MA model used in Burman (1980)
```

264 shank

```
function [b,a,err] = shank(nu,db,da)
%
% Given the impulse response function nu(z) = nu_0 + nu_1*z + nu_2*z^2 +
% ...., this function obtains the rational approximation nu(z ) ~=
% b(z)/a(z), where b(z) = b_0 + b_1*z + ... + b(nb)*z^nb and a(z) = 1
% + a_1*z + ... + a(na)*z^na. The coefficients of b(z) and a(z) are
% computed using Shank's method (Discrete Random signal processing, p.
% 558)
%
% Input arguments:
% nu: the impulse response function
% db: the degree of b(z)
% da: the degree of a(z)
%
% Output arguments:
% b: a (db+1) array containing the b(z) coefficients in ascending order
% a: a (da+1) array containing the a(z) coefficients in ascending order
```

265 sinfelo

```
function  [meind,metip]=sinfelo(eind,etip,nind,ntip,imin)
%
% this function stores information on the eliminated outlier
%
% Input arguments:
% eind: array containing the time index of the previously eliminated
%       outliers
% etip: array containing the type of the previously eliminated outliers
% Output arguments:
% mind: array incorporating the new time index
% mtip: array incorporating the new typer
```

266 skewkur

```
function [Skew,Kurt,Sk,Ss]=  skewkur(e,me,ve,nr,ne)
%*****************************************************************
% This function calculates components of the skewness and kurtosis test
% statistics
%
%    INPUTS:
```

```
%         e : residual vector
%        me : mean of e
%        ve : variance of e
%        nr : number of regression variables
%        ne : length of e
%
%    OUTPUTS:
%      Skew : skewness
%      Kurt : kurtosis
%        Sk : constant to obtain kurtosis test statistic: (Kurt/Sk)^2
%        Ss : constant to obtain skewness test statistic: (Skew/Ss)^2
```

267 smfest

```
function [F,xv] = smfest(xv,y,s,pfix,pvar,xf,chb,models)
%*********************************************************************
% This function transforms model parameters so that their values lie in
% the feasable region and evaluates the residuals for the nonlinear
% minimization of the sum of squares of a canonical structural model
%
%   INPUTS:
%      xv : vector with parameters to be estimated
%       y : data vector
%       s : frequency of the data
%    pfix : array with fixed parameter indices
%    pvar : array with variable parameter indices
%      xf : vector with fixed parameters
%     chb = 1 : compute beta estimator hb and MSE of hb
%         = 0 : do not compute hb and MSE of hb
%  models : structure containing model information
%
%  OUTPUTS:
%       F : residual vector multiplied with factor f given by scakfle2;
%           used in minimization of nonlinear sum of squares
%      xv : vector with parameters to be estimated that has been
%           tranformed if necessary
```

268 smfestm

```
function [F,xv] = smfestm(xv,y,pfix,pvar,xf,chb,models)
%*********************************************************************
% This function transforms model parameters so that their values lie in
% the feasable region and evaluates the residuals for the nonlinear
% minimization of the sum of squares of a canonical structural model
%
%   INPUTS:
%      xv : vector with parameters to be estimated
%       y : data vector
%       s : frequency of the data
%    pfix : array with fixed parameter indices
%    pvar : array with variable parameter indices
%      xf : vector with fixed parameters
%     chb = 1 : compute beta estimator hb and MSE of hb
%         = 0 : do not compute hb and MSE of hb
%  models : structure containing model information
%
%  OUTPUTS:
%       F : residual vector multiplied with factor f given by scakfle2;
%           used in minimization of nonlinear sum of squares
%      xv : vector with parameters to be estimated that has been
%           tranformed if necessary
```

269 smfun

```
function [F,e,g,M,Pevf,A,P,olsres] = smfun(xx,y,s,pfix,pvar,xf,chb,models)
%*********************************************************************
%  This function evaluates the residuals for the nonlinear minimization
%  of the sum of squares of a canonical structural model
%
%   INPUTS:
%      xx : vector with parameters to be estimated
%       y : data vector
%       s : frequency of the data
%    pfix : array with fixed parameter indices
%    pvar : array with variable parameter indices
%      xf : vector with fixed parameters
%     chb = 1 : compute beta estimator hb and MSE of hb
%         = 0 : do not compute hb and MSE of hb
%  models : structure containing model information
%
%  OUTPUTS:
%       F : residual vector multiplied with factor f given by scakfle2;
%           used in minimization of nonlinear sum of squares
%       e : residual vector
%       g : the beta estimator
%       M : the MSE of the beta estimator
%    Pevf : prediction error variance
%       A : the estimated augmented state vector at the end of
%           filtering
%       P  : the MSE of A at the end of filtering
```

270 smfunm

```
function [F,e,g,M,Pevf,A,P] = smfunm(xx,y,pfix,pvar,xf,chb,models)
%*****************************************************************
%  This function evaluates the residuals for the nonlinear minimization
%  of the sum of squares of a canonical structural model
%
%   INPUTS:
%      xx : vector with parameters to be estimated
%       y : data vector
%    pfix : array with fixed parameter indices
%    pvar : array with variable parameter indices
%      xf : vector with fixed parameters
%     chb = 1 : compute beta estimator hb and MSE of hb
%         = 0 : do not compute hb and MSE of hb
%  models : structure containing model information
%
%  OUTPUTS:
%       F : residual vector multiplied with factor f given by scakfle2;
%           used in minimization of nonlinear sum of squares
%       e : residual vector
%       g : the beta estimator
%       M : the MSE of the beta estimator
%    Pevf : prediction error variance
%       A  : the estimated augmented state vector at the end of
%            filtering
%       P  : the MSE of A at the end of filtering
```

271 smoothgen

```
function [KKP,PT,hd,Md]=smoothgen(y,X,Z,G,W,T,H,ins,i,mucd,U,C,D)
%
%
%        This function applies the augmented Kalman filter and smoother
%        to the series y corresponding to the model
```

```
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        It is desired to smooth the vector
%
%         Y_t = U_t*\beta + C_t*alpha_t + D_t*epsilon_t
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time invariant;
%               it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t
%               matrices;
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t
%               matrices;
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial
%             state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%               cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%       mucd: an integer, the dimension of Y_t
%        U  : an (n*mucd x nbeta) matrix containing the U_t matrices;
%             an (mucd x nbeta) if it is time invariant
%             it can be []
%        C  : an (n*mucd x nalpha) matrix containing the C_t matrices;
%             an (mucd x nalpha) if it is time invariant
%        D  : an (n*mucd x nepsilon) matrix containing the D_t matrices;
%             an (mucd x nepsilon) if it is time invariant
%
%        Output parameters:
%        KKP : an (n x mucd) matrix containing the estimated Y_{t|n}
%        PT  : an (mucd*n x mucd) matrix containing the Mse of Y_{t|n}
%        hd  : the (delta',beta')' estimate
%        Md  : the Mse of hd
```

272 SMTres

```
unction [smtres] = SMTres(out)
%
%
% This function obtains the OLS residuals after having used function
% arimaestos, arimaestni or arimaestwi
%
% input arguments:
% out: a structure, the output of function arimaestos, arimaestni or
% arinamestwi
%
% output arguments:
% res: a vector containing the OLS residuals
```

273 sn2u

```
function [p,ierrsn2u] = sn2u(c)
% This function transforms a polynomial in the variables S(n)=z^n +
% z^(-n) into a polynomial in the variable U=z + z^(-1).
% The recursions
%   S(n) = U*S(n-1) - S(n-2),    n>=2
%   S(0)=2,  S(1)=U,
% are used.
%
```

```
% c on input contains the coefficients in the order 1,2,...,n,
%  c= c(1) + c(2)*S(1) + ... + c(n-1)*S(n-2) + c(n)*S(n-1)
% p on output contains the coefficients in reversed order n, n-1, ...
% ,1, p(U) = p(1)*U^(n-1) + p(2)*U^(n-2) + ... + p(n-1)*U + p(n)
```

274 SortSchur

```
function [Q,R,ap] = SortSchur(Q,R,z);
% this function sorts the eigenvalues of the Schur decompostion
```

275 specgraph

```
function  specgraph(th,phi,var)
%*****************************************************************
% PURPOSE: This function graphs the spectrum of the ARIMA model
%
%          phi(B)y_t = th(B)*a_t
%
%--------------------------------------------------
% USAGE: specgraph(th,phi,var)
%
% Inputs: phi : a polynomial containing the AR part
%         th  : a polynomial containing the MA part
%        var  : a positive number, variance of the series model
%               innovations
```

276 spectralan

```
function spr = spectralan(y, per, win, corlag, graph, vnames, width, wina)
%
% Spectral analysis
%
% This programa computes the spectrum, coherence, phase delay, gain and
% cross correlations between a reference cycle and other cycles. If only
% one series is input, the periodogram and the autocorrelations are
% computed.
%
%      INPUTS :
%-----------------
%          y : (ly x ny) matrix with the series;
%              if ny = 1, univariate spectral analysis and computation
%                of autocorrelations of y are performed,
%                if ny > 1, multivariate spectral analysis
%                and computation of cross-correlations are performed;
%                the program assumes that the first column contains
%                the reference series
%          per : frequency of the data (number of seasons)
%                if per < 0, it is set to 1
%          win : window function used for (cross-)periodogram smoothing
%                0, no window is applied (nonsmoothed periodogram).
%                1, the Blackman-Tukey window
%                2, the Parzen window (default)
%                3, the Tukey-Hanning window
%                if win < 0, it is set to 2
%       corlag : number of leads and lags at which the
%                auto-/cross-correlations are computed;
%                if corlag <= 0 or >= length(y), it is set to length(y)-1
%        graph : 0, do not produce graphs
%                1, produce graphs in the original scale
%                2, produce graphs in logarithms (default)
%                if graph < 0, it is set to 2
%       vnames : string cell array with names for the series; the program
%                assumes that their order coincides with the order in y;
%                default: refseries, series1, series2,...
%                if vnames <= 0, the program generates the names according
%                to the default
%       width : window width factor (1/3 by default)
%               if width <= 0, it is set to 1/3
%        wina : "a" parameter for Blackman-Tukey window (0.23 by default)
%               if wina <= 0, it is set to 0.23
%       OUTPUT :
%------------------
%       spr    : structure containing the following fields
%                always
%                .y      : input matrix y
%                .per    : input per
%                .names  : input names or names created by the program
%                .frq    : frequencies
%                depending on the size of y
%                fields for ny = 1:
%                .f      : (smoothed) periodogram of the series
%                .cr     : autocorrelations
%                fields for ny > 2,
%                .f      : matrix with columns containing the (smoothed)
%                             periodogram of the reference series and all the
%                             other series
%                .cr     : matrix containing the autocorrelations  of the
%                             reference series and the cross correlations
%                             between the reference series and all the other
%                             series
%                .co     : matrix with columns containing coherence
%                             between the ref. series and all the other series
%                .ga     : matrix with columns containing the gain
%                             between the ref. series and all the other series
%                .ph     : matrix with columns containing the phase delay
%                             between the ref. series and all the other series
%                .mc     : array containing the maximum coherence values
%                             between the reference series and all the other
%                             series
%                .phmc   : array containing the phase delays corresponding
%                             to the maximum coherence
%                .mcor   : array containing the maximum correlations
%                             between the reference series and all the other
%                             series
%                .imcor  : time indices corresponding to the maximum cross
%                             correlations
%                .mpa    : array containing the mean phase angle in
%                             radians between the reference series and all the
%                             other series
%                .mph    : array containing the mean phase delay between
%                             the reference series and all the other series
```

277 sqrt_ckms

```
function [e,E,rSigmat]=sqrt_ckms(y,Y,str,maxupdt,tol)
%
%
%        This function applies to the series (y,Y) the square root CKMS
%        recursions corresponding to the model in echelon form
%
%        y_t = Y_t*beta + H*x_t + K*a_t
%        x_{t+1}= F*x_t + a_t,
%
%        where a_t is (0,sigmar2),
%
%        with initial state
```

```
%
%        alpha_1= (0,Omega).
%
%        Input parameters:
%        y  : an n x p matrix of observations
%        Y  : an n x (p x nbeta) matrix containing the Y_t matrices; it
%             can be []
%        str: a structure containing the model parameters
%        maxupdt : an integer parameter equal to the maximum number of
%                  updates before passing to the steady state
%                  recursions. If maxupdt is empty, it is made equal to
%                  the number of observations.
%        tol: a real parameter that controls when to pass to the steady
%             state recursions. This happens when the difference
%             between the (1,1) element of the covariance square root
%             of the filter and the (1,1) element of the covariance
%             square root of the model innovations is less than or
%             equal to tol.
%
%        Output parameters:
%      (e,E)   : augmented residual vector
%    rSigmat : the square roots of the residual covariance matrices
```

278 ss2if

```
function [P,K,Sigma,U,iU] = ss2if(F, G, H, J, Q, S, R)
%
% This function transforms a general SS model into an innovations form
% solving the DARE. The state space form is
%
%     x_{t+1}  =  Fx_{t} + Gu_{t}
%     Y_{t}    =  Hx_{t} + Jv_t,
%
%   E[u_t]
%    [v_t][u'_s, v'_s]
%    =[Q  S]
%     [S' R]\delta_{ts}
%
%   P  = Solution of the DARE (discrete algebraic Riccati equation) =
%   MSE(\hat{x}_{T}).
%   K  = Kalman gain in the innovations model
%     \hat{x}_{t+1}  =  F\hat{x}_{t} + Ka_{t}
%     Y_{t}          =  H\hat{x}_{t} + a_{t},
%  Sigma = covariance matrix of the innovations = Var(a_{t})
%   U = Sigma^{1/2}
%   iU = Sigma^{-1/2}
```

279 ss2lcvarmaxf

```
function [phi,theta,ierror]=ss2lcvarmaxf(H,F,K,G)
%
% This function computes a left coprime VARMAX model corresponding to
% a system (H,F,K,G),
%   x(t+1) = F*x(t) + K*u(t)
%    y(t)  = H*x(t) + G*u(t),
% where H is a k x n vector, F is a square n x n matrix, K is an n x m
% matrix and G is a k x m. The system (H,F,K,G) is not necessarily
% minimal.
%
% Input:  H      = a k x n matrix
%         F      = an n x n matrix
%         K      = an n x m matrix
%         G      = a k x m matrix
% If G is not given, it is assumed that m=k and G = I_k.
% Output: phi    = a k x k x l matrix containing the AR matrix
%         polynomial
%         theta  = a k x m x l matrix containing the MA matrix %         polynomial
%                  l = max(n_i:i=1,...,k), where n_i are the Kronecker
%                  indices.
%      ierror    = 0,1  a flag for errors in dimensions
% The method uses the algorithm of Chen to pass from right to left MFD.
% It is based on the decomposition
%
%   Y(t) = [zH(I-zF)^{-1}K + G]u(t) = [S(z)R^{-1}(z)K + G]u(t)
%        = [Rb^{-1}(z)Sb(z)K + G]u(t),
%
% that implies
%
%  Rb(z)Y(t) = [Sb(z)K + Rb(z)G]u(t)
%
% or
%
%  phi(z)Y(t) = theta(z)u(t),
%
```

```
% where phi(z)=Rb(z) and theta(z)=Sb(z)K + Rb(z)G. This method is used
% in Kucera (1991), p. 226.
```

280 ssmpred

```
function [ypr,mypr,alpr,malpr]=ssmpred(n,p,A,P,X,Z,G,W,T,H,g,M)
%
%
%        This function computes n forecasts of the state vector and the
%        observations corresponding to the model
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%        where c is (0,Omega) and delta is (0,kI) (diffuse).
%
%        Input parameters:
%        n  : number of forecasts
%        p  : number of variables
%        A  : the estimated augmented state vector at the end of
%             filtering
%        P  : the Mse of A at the end of filtering
%        X  : an (n*p x nbeta) matrix containing the X_t matrices if it
%             is time-varying. A (p x nbeta) if it is time invariant.
%             It can be []
%        Z  : an (n*p x nalpha) matrix containing the Z_t matrices if it
%             is time-varying. A (p x nalpha) matrix if it is time
%             invariant
%        G  : an (n*p x nepsilon) matrix containing the G_t matrices if
%             it is time-varying. A (p x nepsilon) matrix if it is time
%             invariant
%        W  : an (n*nalpha x nbeta) matrix containing the W_t matrices
%             if is time-varying. An (nalpha x nbeta) matrix if it is
%             time invariant. It can be []
%        T  : an (n*nalpha x nalpha) matrix containing the T_t matrices
%             if it is time-varying. An (nalpha x nalpha) matrix if it
%             time invariant.
%        H  : an (n*nalpha x nepsilon) matrix containing the H_t
%             matrices if it is time-varying. An (nalpha x nepsilon) if
%             it is time invariant.
%        g   : the beta estimator
%        M   : the Mse of the beta estimator
%
%        Output parameters:
%        ypr : a (p x n) matrix containing the forecasts of the
%              observations
%        mypr: a (p x p x n) array containing the Mse of forecasts
%        alpr: an (nalpha x n) matrix containing the forecasts of the
%              state vector
%       malpr: an (nalpha x nalpha x n) array containing the Mse of alpr
```

281 ssmpredexg

```
function [ypr,mypr,alpr,malpr]=ssmpredexg(n,x,stx,sts)
%
%        This function computes n forecasts of the exogenous part of a
%        VARMAX model in echelon form.
%
% The state space echelon form is:
%
%  alpha_{t+1} = F*alpha_{t} + B*x_t{t} + K*a_{t}
%      y_{t}   = Y_{t}*beta + H*alpha_{t} + D*x_{t}  + a_{t}
%
% Thus,
%
%      y_{t} = Y_{t}*beta + V_{t} + U_{t},
%
% where V_{t} is the exogenous part,
%
%      V_{t} = [H*(zI - F)_{t}^{-1}B*x_{t} + D*x_{t}] + H*F^{t-1}*m1
%
%---------------------------------------------------
% USAGE: [ypr,mypr,alpr,malpr]=ssmpredexg(n,x,stx,sts)
%
%        Input parameters:
%        n  : number of forecasts
%        x  : a (nobs x mx) matrix containing the input variables
%      stx  : a structure with fields .T, .Z, .B and .D such that
%             T=F, Z=H, and B and D are matrices of the VARMAX model
%      stx  = a structure. If it is empty, the inputs are nonstochastic and
%             their forecasts are included in x. If it is not empty, it
%             has fields .T, .Z, .H and .G corresponding to an VARMA
%             model followed by the inputs.
%
%        Output parameters:
%        ypr : a (p x n) matrix containing the forecasts of the
%              observations
%        mypr: a (p x p x n) array containing the Mse of forecasts
%        alpr: an (nalpha x n) matrix containing the forecasts of the
%              state vector
%       malpr: an (nalpha x nalpha x n) array containing the Mse of alpr
```

282 ssmspectfac

```
function [Theta,Omega]=ssmspectfac(Gamma)
%
%        This function computes the spectral factorization
%        of the covariance generating function
%        G(z) = Gamma(0)+\sum_{i=1}^q Gamma(i)z^{i} + \sum_{i=1}^q Gamma'(i)z^{-i}.
%        That is, it finds a matrix polynomial \Theta(z) = I+\Theta_1z+ ... +\Theta_qz^{
%        such that G(z) = \Theta(z)\Omega\Theta'(z^{-1}). This is achieved by solving
%        the Riccati equation for the Kalman filter based on covariance
%        data only
%
%        Sigma = FSigmaF' - (FSigmaH' + G)(HSigmaH' + Gamma(0))^{-1}(FSigmaH' + G)'
%
%        This equation is the resulta of replacing P with -Sigma in the
%        usual DARE equation
%
%        P = FPF' + GQG' - (FPH' + GS)(R + HPH')^{1}(FPH' + GS)',
%
%        given that in this case Pi = P  + Sigma = 0. See Gomez (2016, Sec.
%        5.6)
%
% Input parameters:
% Gamma   = an nxnxm matrix containing the covariances
%           Gamma(0),...,Gamma(q)
% Output parameters:
% K       = the moving average coefficients
% Omega   = the covariance matrix of the innovations
```

283 sta2

```
function str = sta2(str)
% PURPOSE: given a structure passed after executing the second step of
% HR method such that the model is not stationary, this function makes
% it stationary.
%---------------------------------------------------
% USAGE: str = sta2(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

284 sta2r

```
function str = sta2r(str)
% PURPOSE: given a structure passed after executing the second step of
% HR method such that the model is not stationary, this function makes
% it stationary.
%---------------------------------------------------
% USAGE: str = sta2r(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

285 sta3

```
function [str,ierror] = sta3(str)
% PURPOSE: given a structure passed after executing the third step of
% HR method such that the model is not stationary, this function makes
% it stationary.
%---------------------------------------------------
% USAGE: str = sta3(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

286 sta3r

```
function str = sta3r(str)
% PURPOSE: given a structure passed after executing the third step of
% HR method such that the model is not stationary, this function makes
% it stationary.
%---------------------------------------------------
% USAGE: str = sta3r(str)
% where:    str    = a structure containing the structure of the VARMAX
%---------------------------------------------------
% RETURNS: str = a structure containing the inverted model
%---------------------------------------------------
```

287 stair

```
function [a1,b1,k,u,C,v,ierror]=stair(a,b,tol,adj)
%
% [A1,B1,K,U,C,V,IERROR]=STAIR(A,B,TOL,ADJ) performs the staircase reduction of
% the pair (A,B). The transformed pair (A1,B1) has the typical staircase form
% (here with 4 "stairs"):
%
%                                     | X : * * * * * | } K(1)
%                                     |   : X * * * * | } K(2)
%         (B1:A1):=(U'*B*V:U'*A*U) =  |   :   X * * * | } K(3)
%                                     |   :     X * * | } K(4)
%                                     |   :         Z | }
%
% where each "stair" X has full row rank K(i) and has a lower
% triangular form (left adjusted) or an upper triangular form (right
% adjusted). In case adj is not given on input, the matrix V to adjust
% the X is not computed and the "stairs" X have just full row rank. The
% square matrix Z (if present) contains the uncontrollable modes of
% (A,B). The parameter tol is used to calculate the rank of the
% triangular matrices given by the
% qr algorithm.
%
% tol = [], the tolerence is computed by the program
%
%
% adj = 'l', lower triangular form (left adjusted)
%       'r', upper triangular form (right adjusted)
%
```

```
% This function is taken from VanDoren (2003) and modified by me on
% May 2008.
```

288 stamodel

```
function [str,ierror]=stamodel(str)
% PURPOSE: given a structure with a nonstationary model, it obtains a
% stationary model.
% The model is assumed to be in echelon form. The stationary model is
% also in echelon form, but it imposes less constraints. The
% constraints are equal to those of the MA part.
%---------------------------------------------------
% USAGE: str=stamodel(str)
% where:    str    = a structure containing the model information
%---------------------------------------------------
% RETURNS: str  = the structure with the invertible model
%---------------------------------------------------
```

289 sucdm

```
function [X,Z,G,W,T,H,ins,ii,strc,ferror] = sucdm(comp,y,Y,stra,npr)
%*********************************************************************
% PURPOSE: this function sets up a state space model given a structure
% containing information about the components of a canonical
% decompostion model. It returns a structure containing the model
% information. The canonical decomposition model is
%
%  y_t = Y_t*beta + p_t + s_t + i_t,
%
% where Y_t is a vector of regression variables, p_t is the canonical
% trend-cycle, s_t is the canonical seassonal and i_t is the canonical
% irregular component.
%---------------------------------------------------
% USAGE: [X,Z,G,W,T,H,ins,ii,strc,ferror] = sucdm(comp,y,Y,stra,npr)
% Inputs:
%           comp   = a structure with the following fields:
%          .ptnum  = a polynomial containing the trend-cycle numerator
%          .ptden  = a polynomial containing the trend-cycle denominator
%          .ptnur  = number of unit roots in ptden
%          .ptvar  = a positive number containing the variance of the
%                    trend-cycle innovations
%          .stnum  = a polynomial containing the seasonal numerator
%           .stden  = a polynomial containing the seasonal denominator
%           .stnur  = number of unit roots in stden
%           .stvar  = a positive number containing the variance of the
%                     seasonal innovations
%           .rt     = a polynomial containing the transitory component
%           .rtvar  = a positive number containing the variance of the
%                     transitory component innovations.
%           .itvar  = a positive number containing the variance of the
%                     irregular component
%           .sigmaa = a positive number containing the variance of the
%                     series model innovations.
%            y      = an (n x 1) matrix containing the data
%            Y      = an (n x nbeta) matrix containing the regression
%                     variables
%            stra   = a structure, given as output by function
%                     suvarmapqPQ
%            npr    = number of forecasts
%
% Outputs:
%       X,Z,G,W,T,H,ins,ii are the matrices and initial conditions
%       information corresponding to the state space model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%       alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse).
%       More specifically,
%       X    : a  (p x nbeta) matrix
%              it can be []
%       W    : an empty matrix
%       Z    : a  (p x nalpha) matrix
%       G    : a  (p x nepsilon) matrix
%       T    : an (nalpha x nalpha) matrix
%       H    : an (nalpha x nepsilon) matrix
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       ii   : a  1 x 4 array containing 4 integers, ii=[cc cw0 ca1
%              cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%
% Other outputs:
%       strc     = a structure with the following fields:
%       .stra
%       .X,.Z,.G,.W,.T,.H,.ins and .ii
%       .comp
```

290 sucdmpbp

```
function [X,Z,G,W,T,H,ins,ii,strc,ferror] = sucdmpbp(comp,compf,y,Y,...
                                                     stra,npr)
%*********************************************************************
% PURPOSE: this function sets up a state space model given a structure
% containing information about the components of a canonical
% decompostion model, where the trend-cycle is further decomposed into
% a smooth trend and a cycle by means of the application of a band-pass
% filter. See "The Use of Butterworth Filters for Trend and Cycle
% Estimation in Economic Time Series", G\'{o}mez, V. (2001), Journal of
% Business and Economic Statistics, 19, 365-373. It returns a structure
% containing the model information. The canonical decomposition model is
%
%  y_t = Y_t*beta + p_t + s_t + i_t,
%
% where Y_t is a vector of regression variables, p_t is the canonical
% trend-cycle, s_t is the canonical seassonal and i_t is the canonical
% irregular component. The trend-cycle is further decomposed as
%
% p_t = ps_t + c_t,
%
% where the models of ps_t and c_t are given by the parameters of p_t
% and the parameters of the filter.
%---------------------------------------------------
% USAGE: [X,Z,G,W,T,H,ins,ii,strc,ferror] =
%                          sucdmpbp(comp,compf,y,Y,stra,npr)
% Inputs:
%              comp  = a structure with the following fields:
%            .ptnum  = a polynomial containing the trend-cycle numerator
%           .ptden  = a polynomial containing the trend-cycle
%                     denominator
%           .ptnur  = number of unit roots in ptden
%           .ptvar  = a positive number containing the variance of the
%                     trend-cycle innovations
%           .stnum  = a polynomial containing the seasonal numerator
%           .stden  = a polynomial containing the seasonal denominator
%           .stnur  = number of unit roots in stden
%           .stvar  = a positive number containing the variance of the
%                     seasonal innovations
%           .rt     = a polynomial containing the transitory component
%           .rtvar  = a positive number containing the variance of the
%                    transitory component innovations.
%           .itvar  = a positive number containing the variance of the
%                     irregular component
%           .sigmaa = a positive number containing the variance of the
%                     series model innovations.
%           .phi    = a polynomial containing the regular phi
%
%          compf    = a structure with the following fields
%          .num     = a polynomial containing the filter numerator
%          .den     = a polynomial containing the filter denominator
%          .Alpha   = a polynomial containing the filter Alpha
%          .sa      = a number, filter sa
%          .Di      = a positive integer, filter Di
%          .Thetac  = a number, the frequency, divided by pi, of gain
%                     .5 in the But. sine/tangent filter.
%          .Lambda  = a positive number, filter Lambda
%
%           y       = an (n x 1) matrix containing the data
%           Y       = an (n x nbeta) matrix containing the regression
%                     variables
%           stra    = a structure, given as output by function
%                     suvarmapqPQ
%           npr     = number of forecasts
%
% Outputs:
%      X,Z,G,W,T,H,ins,ii are the matrices and initial conditions
%      information corresponding to the state space model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%       alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
```

```
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse).
%       More specifically,
%       X    : a  (p x nbeta) matrix
%              it can be []
%       W    : an empty matrix
%       Z    : a  (p x nalpha) matrix
%       G    : a  (p x nepsilon) matrix
%       T    : an (nalpha x nalpha) matrix
%       H    : an (nalpha x nepsilon) matrix
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       ii   : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%             cca1], where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%
%  Other outputs:
%        strc     = a structure with the following fields:
%        .stra
%        .X,.Z,.G,.W,.T,.H,.ins and .ii
%        .comp
%        .compf
```

291 sucdmpbst

```
function [X,Z,G,W,T,H,ins,ii,strc,ferror] = sucdmpbst(comp,compf,y,Y,...
stra,npr)
%*********************************************************************
% PURPOSE: this function sets up a state space model given a structure
% containing information about the components of a canonical
% decompostion model, where the trend-cycle is further decomposed into
% a smooth trend and a cycle by means of the application of a low-pass
% filter of the Butterworth sine or tangent type. See "The Use of
% Butterworth Filters for Trend and Cycle Estimation in Economic Time
% Series", G\'{o}mez, V. (2001), Journal of Business and Economic
% Statistics, 19, 365-373. It returns a structure containing the model
% information. The canonical decomposition model is
%
%  y_t = Y_t*beta + p_t + s_t + i_t,
%
% where Y_t is a vector of regression variables, p_t is the canonical
% trend-cycle, s_t is the canonical seassonal and i_t is the canonical
% irregular component. The trend-cycle is further decomposed as
%
% p_t = ps_t + c_t,
%
% where the models of ps_t and c_t are given by the parameters of p_t and
% the parameters of the filter.
%---------------------------------------------------
% USAGE: [X,Z,G,W,T,H,ins,ii,strc,ferror] =
%                                 sucdmpbst(comp,compf,y,Y,stra,npr)
% Inputs:
%            comp  = a structure with the following fields:
%          .ptnum  = a polynomial containing the trend-cycle numerator
%          .ptden  = a polynomial containing the trend-cycle
%                    denominator
%          .ptnur  = number of unit roots in ptden
%          .ptvar  = a positive number containing the variance of the
%                    trend-cycle innovations
%          .stnum  = a polynomial containing the seasonal numerator
%          .stden  = a polynomial containing the seasonal denominator
%          .stnur  = number of unit roots in stden
%          .stvar  = a positive number containing the variance of the
%                    seasonal innovations
%          .rt     = a polynomial containing the transitory component
%          .rtvar  = a positive number containing the variance of the
%                   transitory component innovations.
%          .itvar  = a positive number containing the variance of the
%                    irregular component
%          .sigmaa = a positive number containing the variance of the
%                    series model innovations.
%          .phi    = a polynomial containing the regular phi
%
%         compf    = a structure with the following fields
%         .num     = a polynomial containing the filter numerator
%         .den     = a polynomial containing the filter denominator
%         .Alpha   = a polynomial containing the filter Alpha
%          .sa      = a number, filter sa
%          .Di      = a positive integer, filter Di
%          .Thetac  = a number, the frequency, divided by pi, of gain
%                     .5 in the But. sine/tangent filter.
%          .Lambda  = a positive number, filter Lambda
%
%           y       = an (n x 1) matrix containing the data
%           Y       = an (n x nbeta) matrix containing the regression
%                     variables
%           stra    = a structure, given as output by function
%                     suvarmapqPQ
%           npr     = number of forecasts
%
% Outputs:
%      X,Z,G,W,T,H,ins,ii are the matrices and initial conditions
%      information corresponding to the state space model
%
%       y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%       alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t,
%
%       where epsilon_t is (0,sigma^2I),
%
%       with initial state
%
%       alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%       where c is (0,Omega) and delta is (0,kI) (diffuse).
%
% More specifically,
%       X    : a  (p x nbeta) matrix
%              it can be []
%       W    : an empty matrix
%       Z    : a  (p x nalpha) matrix
%       G    : a  (p x nepsilon) matrix
%       T    : an (nalpha x nalpha) matrix
%       H    : an (nalpha x nepsilon) matrix
%       ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       ii   : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%             cca1], where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%
% Other outputs:
%        strc     = a structure with the following fields:
%        .stra
%        .X,.Z,.G,.W,.T,.H,.ins and .ii
%        .comp
%        .compf
```

292 sumpol

```
% This function sums two polynomials A and B, possibly of different
% degrees. The polynomials are assumed to be in reversed order, that is
% A(x) = A(1)*x^(p-1) + A(2)*x^(p-2) + ... + A(p-1)*x + A(p)
```

293 susmspbp

```
function [np,X,Z,G,W,T,H,ins,ii]=susmspbp(compbp,x,pfix,pvar,xf,...
%                                        stordt,conc,n,r,l)
%*********************************************************************
% Function to set up the state space model corresponding to the
% structural model
%
%      y_t = mu_t + epsilon_t,
%
% where
%
%      mu_{t+1} = mu_{t} + beta_{t} + zeta_{t}
%    beta_{t+1} = beta_{t} + eta_{t},
%
% plus a band-pass filter, applied as described in "The Use of
% Butterworth Filters for Trend and Cycle Estimation in Economic Time
% Series", G\'{o}mez, V. (2001), Journal of Business and Economic
% Statistics, 19, 365-373. This model is used in the paper "Monthly US
% Business Cycle Indicators: A new Multivatiate Approach Based on a
% Band-pass Filter", Marczak and G\'{o}mez, Empirical Economics, 52,
% 1379-1408.
%
%  INPUTS:

%         x : array with estimated parameters
%      pfix : array with fixed parameter indices
%      pvar : array with variable parameter indices
%        xf : array with fixed parameters
%    stordt : array with indices for standard deviations
%      conc : index for the standard deviation to be concentrated
%             out
%         n : number of variables in the data matrix y
%         r : number of slope factors
%         l : numbef of quarterly series
%
%    OUTPUTS:
%        np : number of rows of matrix Tp (see below)
%        the following arguments are for generated the state space
%        model
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time invariant;
%               it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t %               matrice
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t %               matric
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        ii   : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
```

294 suusm

```
function [str,ferror] = suusm(comp,y,Y,npr)
%*******************************************************************
% PURPOSE: this function sets up a univariate structural model given a
% structure containing information about the components. It returns a
% structure containing the model information. The model is
%
%  y_t = Y_t*beta + p_t + s_t + u_t + v_t + e_t,
%
% where Y_t is a vector of regression variables, p_t is the trend, s_t
% is the seassonal, u_t is the cyclical, v_t is the AR, and e_t is the
% irregular component.  This function allows for other values of freq
% besides 1, 4 and 12.
% Note that freq is a field of structures comp and datei. It is also the
% number of seasons. Therefore, it determines the seasonal component.
%---------------------------------------------------
% USAGE: [str,ferror] = suusm(comp,y,Y,npr)
% where structure comp has the following fields:
%         .level  = a 1 x 3 dimensional array such that level(1) is a
%                   code (see below *), level(2) is the standard error
%                   of the level and level(3) = NaN means the standard
%                   error is to be estimated, =0 it is fixed
%         .slope  = a 1 x 3 dimensional array such that slope(1) is a
%                   code (see below *), slope(2) is the standard error
%                   of the slope and slope(3) = NaN means the standard
%                   error is to be estimated, =0 it is fixed
%         .seas   = a 1 x 3 dimensional array such that seas(1) is a
%                   code (see below *), seas(2) is the standard error
%                   of the seasonal and seas(3) = NaN means the
%                   standard error is to be estimated, =0 it is fixed
%         .cycle  = a 1 x 3 dimensional array such that cycle(1) is a
%                   code (see below *), cycle(2) is the standard error
%                   of the cycle and cycle(3) = NaN means the standard
%                   error is to be estimated, =0 it is fixed
%         .cyclep = (only if field .cycle is present) a 2 x 2 array
%                   containing the first row the two cycle parameters
%                   (rho and freqc) and the second row a NaN or zero
%                   for each cycle parameter. Each NaN means that the
%                   correspondig cycle parameter is to be estimated
%                   and each zero means that it is fixed
%         .cycleb = (only if field .cycle is present) a 1 x 2 array
%                   such that cycleb(1) and cycleb(2) contain the end
%                   points of the frequency interval in which the
%                   cycle is supposed to be defined.
%         .ar     = a 1 x 3 dimensional array such that ar(1) is a
%                   code (see below *), ar(2) is the standard error of
%                   the ar component and ar(3) = NaN means the standard
%                    error is to be estimated, =0 it is fixed
%          .arp    = (only if field .ar is present) a 2 x k array, where
%                    k is the order of the autoregressive, containing
%                    the first row the autoregressive parameters and the
%                    second row a NaN or zero for each autoregressive
%                    parameter. Each NaN means that the correspondig
%                    autoregressive parameter is to be estimated and
%                    each zero means that it is fixed
%          .irreg  = a 1 x 3 dimensional array such that irreg(1) is a
%                    code (see below *), irreg(2) is the standard error
%                    of the irregular and irreg(3) = NaN means the
%                    standard error is to be estimated, =0 it is fixed
%          .conout = 'level' if the standard error of the level is
%                    concentrated out
%                    'slope' if the standard error of the slope is
%                    concentrated out
%                    'seas' if the standard error of the seasonal
%                    is concentrated out
%                    'cycle' if the standard error of the cycle is
%                    concentrated out
%                    'ar' if the standard error of the ar component is
%                    concentrated out
%                    'irreg' if the standard error of the irregular is
%                    concentrated out
%                    If .conout is not input, the program will determine
%                    the biggest variance.
%          .freq   = number of observations per year
%          .datei  = calendar structure (output of function cal)
%         .sqrtfil = 0, use ordinary two-stage Kalman filter for
%                    estimation
%                    1, use square root version (specially for long
%                    series)
%          y       = data vector
%          Y       = matrix for regression variables. It contains the
%                    stack of the Y_t matrices.
%          npr     = number of forecasts
%-----------------------------------------------------------------------
% * codes for the components:
%   level = -1  constant
%            1  stochastic
%            2  Butterworth tangent
%   slope = -1  constant
%            1  stochastic
%   seas  = -1  fixed dummy seasonality
%            1  stochastic dummy seasonality
%            2  trigonometric seasonality
%            4  Butterworth tangent
%   cycle =  1  structural model cycle
%            2  Butterworth sine cycle
%   irreg =  1  stochastic
%   ar    =  k  autoregressive component of order k
%
%-----------------------------------------------
% RETURNS: str = a structure containing the following fields
%
%       matrices according to the model
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%      More specifically:
%            .X : an (n x nbeta) matrix containing the X_t matrices;
%                 a  (1 x nbeta) if it is time invariant;
%                 it can be []
%            .Z : an (n x nalpha) matrix containing the Z_t matrices;
%                 a  (1 x nalpha) matrix if it is time invariant
%            .G : an (n x nepsilon) matrix containing the G_t matrices;
%                 a  (1 x nepsilon) matrix if it is time invariant
%            .W : an (n*nalpha x nbeta) matrix containing the W_t
%                 matrices;
%                 an (nalpha x nbeta) matrix if it is time invariant;
%                 it can be []
%            .T : an (n*nalpha x nalpha) matrix containing the T_t %                 matr
%                 an (nalpha x nalpha) matrix if it time invariant
%            .H : an (n*nalpha x nepsilon) matrix containing the H_t %                 ma
%                 an (nalpha x nepsilon) if it is time invariant
%     .ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%            state information, according to array i below
%       .i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1 cca1],
%            where
%            cc   = nalpha if c is not missing (0 if c missing)
%            cw0  = number of columns in W_0 (0 if W_0 missing)
%            ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%            cca1 = number of columns in A_1 (0 if A_1 missing)
%   .trend : trend code
%   .slope : slope code
%    .seas : seasonal code
%   .cycle : cycle code
%     .xl1 : lower bound of the frequency interval in which the
%            cycle is supposed to be defined
%     .xl2 : upper bound of the frequency interval in which the
%            cycle is supposed to be defined
%     .arp : AR code
%   .irreg : irregular code
%    .conc : index for the parameter that is concentrated out
%           (see description below *)
%       .x : vector with all parameters (see description below **)
%      .xv : vector with parameters to be estimated
%      .xf : vector with fixed parameters
%    .pvar : array with variable parameter indices
%    .pfix : array with fixed parameter indices
%   .stord : array containing parameter indices (see description below
%            ***)
%    .freq : frequency of the data
%   .datei : calendar structure
%    .comp : structure comp (input to suusm)
%
%-------------------------------------------------
% * One of the standard deviations is concentrated out and, therefore,
%   is not estimated. The field conout contains information about this
%   standard deviation. The user can select this standard deviation or
%   the program can do it automatically instead. The biggest variance
%   should be selected.
%
% ** we put in x the parameters of the model, except the one that is
%    concentrated out, in the order:
%    1 - irregular standard deviation
%    2 - level standard deviation
%    3 - slope standard deviation
%    4 - seasonal standard deviation
%    5 - autoregressive standard deviation
%    6 - cycle standard deviation
%    7,8 - cycle parameters, rho and frequency
%    9,10,.. autoregressive parameters
%
% *** stord is an index such that its i-th element indicates to which
%     component (according to the ordering above) belongs the i-th
%     element of x.
%*********************************************************************
```

295 suusmm

```
function [str,ferror] = suusmm(comp,y,Y,npr)
%*********************************************************************
% PURPOSE: this function sets up a univariate structural model with
% complex seasonal patterns given a structure containing information
% about the components. It returns a structure containing the model
% information. The model is
%
%  y_t = Y_t*beta + p_t + s_t + u_t + v_t + e_t,
%
% where Y_t is a vector of regression variables, p_t is the trend, s_t
% is the seassonal, u_t is the cyclical, v_t is the AR, and e_t is the
% irregular component.  This function allows for several patterns of
% seasonality. That is, s_t = \sum_{j=1}^{N} s_{t}^j, s_{t}^j =
% \sum_{i=1}^{m_j}s_{i,t}^j, n_j is the period of s_{t}^j, m_j is the
% number of harmonics of s_{t}^j and
%
% [s_{i,t}^j    ]  [cos(2\pi i/n_j)  sin(2\pi i/n_j)]    [j_{i,t}   ]
% [s_{i,t}^{* j}]= [-sin(2\pi i/n_j) cos(2\pi i/n_j)]  + [j^*_{i,t} ].

%---------------------------------------------------
% USAGE: [str,ferror] = suusmm(comp,y,Y,npr)
% where structure comp has the following fields:
%          .level  = a 1 x 3 dimensional array such that level(1) is a
%                    code (see below *), level(2) is the standard error
%                    of the level and level(3) = NaN means the standard
%                    error is to be estimated, =0 it is fixed
%          .slope  = a 1 x 3 dimensional array such that slope(1) is a
%                    code (see below *), slope(2) is the standard error
%                    of the slope and slope(3) = NaN means the standard
%                    error is to be estimated, =0 it is fixed
%          .seasp  = a cell array whose elements are 1 x 4 dimensional
%                    arrays defining the seasonal patterns. The first
%                    pair in each array, [per_j,m_j], are the period
%                    and the number of harmonics. The third element in
%                    the array is the standard error of that seasonal
%                    component and the fourth element in the array =
%                    NaN means the standard error is to be estimated,
%                    =0 it is fixed
%          .cycle  = a 1 x 3 dimensional array such that cycle(1) is a
%                    code (see below *), cycle(2) is the standard error
%                    of the cycle and cycle(3) = NaN means the standard
%                    error is to be estimated, =0 it is fixed
%          .cyclep = (only if field .cycle is present) a 2 x 2 array
%                    containing the first row the two cycle parameters
%                    (rho and freqc) and the second row a NaN or zero
%                    for each cycle parameter. Each NaN means that the
%                    correspondig cycle parameter is to be estimated
%                    and each zero means that it is fixed
%          .cycleb = (only if field .cycle is present) a 1 x 2 array
%                    such that cycleb(1) and cycleb(2) contain the end
%                    points of the frequency interval in which the
%                    cycle is supposed to be defined.
%          .ar     = a 1 x 3 dimensional array such that ar(1) is a
%                    code (see below *), ar(2) is the standard error of
%                    the ar component and ar(3) = NaN means the standard
%                    error is to be estimated, =0 it is fixed
%          .arp    = (only if field .ar is present) a 2 x k array, where
%                    k is the order of the autoregressive, containing
%                    the first row the autoregressive parameters and the
%                    second row a NaN or zero for each autoregressive
%                    parameter. Each NaN means that the correspondig
%                    autoregressive parameter is to be estimated and
%                    each zero means that it is fixed
%          .irreg  = a 1 x 3 dimensional array such that irreg(1) is a
%                    code (see below *), irreg(2) is the standard error
%                    of the irregular and irreg(3) = NaN means the
%                    standard error is to be estimated, =0 it is fixed
%          .conout = 'level' if the standard error of the level is
%                    concentrated out
%                    'slope' if the standard error of the slope is
%                    concentrated out
%                    'seasp' if the standard error of the seasonal
%                    is concentrated out
%                    'cycle' if the standard error of the cycle is
%                    concentrated out
%                    'ar' if the standard error of the ar component is
%                    concentrated out
%                    'irreg' if the standard error of the irregular is
%                    concentrated out
%                    If .conout is not input, the program will determine
%                    the biggest variance.
%         .sqrtfil = 0, use ordinary two-stage Kalman filter for
%                    estimation
%                    1, use square root version (specially for long
%                    series)
%          y       = data vector
%          Y       = matrix for regression variables. It contains the
%                    stack of the Y_t matrices.
%          npr     = number of forecasts
%-----------------------------------------------------------------------
% * codes for the components:
%   level = -1  constant
%            1  stochastic
%            2  Butterworth tangent
%   slope =  1  stochastic
%   cycle =  1  structural model cycle
%            2  Butterworth sine cycle
%   irreg =  1  stochastic
%   ar    =  k  autoregressive component of order k
%
%-----------------------------------------------
% RETURNS: str = a structure containing the following fields
%
%       matrices according to the model
%
%          y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%      More specifically:
%            .X : an (n x nbeta) matrix containing the X_t matrices;
%                 a  (1 x nbeta) if it is time invariant;
%                 it can be []
%             .Z : an (n x nalpha) matrix containing the Z_t matrices;
%                 a  (1 x nalpha) matrix if it is time invariant
%             .G : an (n x nepsilon) matrix containing the G_t matrices;
%                 a  (1 x nepsilon) matrix if it is time invariant
%             .W : an (n*nalpha x nbeta) matrix containing the W_t
%                 matrices;
%                 an (nalpha x nbeta) matrix if it is time invariant;
%                 it can be []
%             .T : an (n*nalpha x nalpha) matrix containing the T_t %                 matr
%                 an (nalpha x nalpha) matrix if it time invariant
%             .H : an (n*nalpha x nepsilon) matrix containing the H_t %                ma
%                 an (nalpha x nepsilon) if it is time invariant
%       .ins : an nalpha x (cc+cw0+ca1+cca1) matrix containing the initial
%             state information, according to array i below
%         .i : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1 cca1],
%             where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%     .trend : trend code
%     .slope : slope code
%      .seas : seasonal code
%     .cycle : cycle code
%       .xl1 : lower bound of the frequency interval in which the
%             cycle is supposed to be defined
%       .xl2 : upper bound of the frequency interval in which the
%             cycle is supposed to be defined
%       .arp : AR code
%     .irreg : irregular code
%      .conc : index for the parameter that is concentrated out
%            (see description below *)
%         .x : vector with all parameters (see description below **)
%        .xv : vector with parameters to be estimated
%        .xf : vector with fixed parameters
%      .pvar : array with variable parameter indices
%      .pfix : array with fixed parameter indices
%     .stord : array containing parameter indices (see description below
%             ***)
%      .freq : frequency of the data
%     .datei : calendar structure
%      .comp : structure comp (input to suusm)
```

```
%
%-------------------------------------------------
% * One of the standard deviations is concentrated out and, therefore,
%   is not estimated. The field conout contains information about this
%   standard deviation. The user can select this standard deviation or
%   the program can do it automatically instead. The biggest variance
%    should be selected.
%
% ** we put in x the parameters of the model, except the one that is
%    concentrated out, in the order:
%        1 - irregular standard deviation
%        2 - level standard deviation
%        3 - slope standard deviation
% 4,...3+N - ith-seasonal standard deviation, where N = number of
%            seasonal patterns
%      4+N - autoregressive standard deviation
%      5+N - cycle standard deviation
%  6+N,7+N - cycle parameters, rho and frequency
% 8+N,9+N,.. autoregressive parameters
%
% *** stord is an index such that its i-th element indicates to which
%     component (according to the ordering above) belongs the i-th
%      element of x.
%*********************************************************************
```

296 suvarmapqPQ

```
function [str,ferror] = suvarmapqPQ(phi,th,Phi,Th,Sigma,freq)
%********************************************************************
% PURPOSE: sets up a VARMA model given the matrix polynomials and the
% frequency (number of observations per year). It returns a structure
% containing the model information
%-------------------------------------------------
% USAGE: [str,ferror] = suvarmaxpqPQ(phi,th,Phi,Th,Sigma,freq)
% where:   phi      = the regular AR matrix polynomial
%          th       = the regular MA matrix polynomial
%          Phi      = the seasonal AR matrix polynomial
%          Th       = the seasonal MA matrix polynomial
%          Sigma    = the innovations covariance matrix
%          freq     = number of observations per year
%-------------------------------------------------
% RETURNS: str = a structure containing the following fields
%            .s: dimension of Y_t
%         .freq: number of observations per year
%          .phi: phi(z) matrix polynomial
%           .th: theta(z) matrix polynomial
%          .Phi: Phi(z^freq) matrix polynomial
%           .Th: Theta(z^freq) matrix polynomial
%        .Sigma: innovations covariance matrix, where the element
%               (1,1) has been concentrated out of the likelihood
%           .Lh: vech of the Cholesky factor of Sigma
%         .phin: phi(z) matrix polynomial, where each parameter to
%                estimate is replaced with NaN and each fixed parameter
%                is replaced with zero
%         .Phin: Phi(z^freq) matrix polynomial, where each parameter to %
%                is replaced with zero
%          .thn: Theta(z^freq) matrix polynomial, where each parameter %               e
%                is replaced with zero
%          .Thn: phi(z) matrix polynomial, where each parameter to %               esti
%                is replaced with zero
%          .Lhn: Lh vector, where each parameter to estimate is
%                replaced with NaN and each fixed parameter is replaced
%                with zero
%            .x: parameter vector, including fixed and variable
%                parameters
%        .nparm: number of parameters
%           .xi: array of ones and zeros, where each one indicates a
%                parameter in x that is to be estimated and each zero
%                indicates a fixed parameter
%           .xv: array of parameters to estimate
%           .xf: array of fixed parameters
%        .phirs: product of phi(z) and Phi(z^freq)
%         .thrs: product of theta(z) and Theta(z^freq)
%
%      Z,T,H,G,X,W matrices of the state space model
%
%                 Y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
```

```
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%
%      More specifically:
%            .Z: a (s x nalpha) matrix
%            .T: a (nalpha x nalpha) matrix
%            .H: a (nalpha x nepsilon) matrix
%            .G: a (s x nepsilon) matrix
%            .X: []
%            .W: []
%
%     Note: user can subsequently incorporate regression variables into
%           the state space model given by suvarmapqPQ by an appropriate
%           specification of matrix X
%---------------------------------------------------
```

297 suvarmapqPQe

```
function [str,ferror] = suvarmapqPQe(Lambda,alpha,betap,th,...
                                     Th,Sigma,freq)
%*******************************************************************
% PURPOSE: sets up a VARMA model given the matrix polynomials and the
% frequency (number of observations per year). It returns a structure
% containing the model information
%---------------------------------------------------
% USAGE: [str,ferror] = suvarmaxpqPQ(phi,th,Phi,Th,Sigma,freq)
% where:   Lambda   = the regular AR matrix polynomial in ecf
%          alpha    = a matrix such that Pi = alpha*betap
%          betap    = a matrix such that Pi = alpha*betap
%          th       = the regular MA matrix polynomial
%          Th       = the seasonal MA matrix polynomial
%          Sigma    = the innovations covariance matrix
%          freq     = number of observations per year
%---------------------------------------------------
% RETURNS: str = a structure containing the following fields
%            .s: dimension of Y_t
%         .freq: number of observations per year
%          .phi: phi(z) matrix polynomial
%           .th: theta(z) matrix polynomial
%          .Phi: Phi(z^freq) matrix polynomial
%           .Th: Theta(z^freq) matrix polynomial
%        .Sigma: innovations covariance matrix, where the element
%               (1,1) has been concentrated out of the likelihood
%           .Lh: vech of the Cholesky factor of Sigma
%         .phin: phi(z) matrix polynomial, where each parameter to %             esti
%                is replaced with zero
%         .Phin: Phi(z^freq) matrix polynomial, where each parameter to %
%                is replaced with zero
%          .thn: Theta(z^freq) matrix polynomial, where each parameter %               e
%                is replaced with zero
%          .Thn: phi(z) matrix polynomial, where each parameter to %             esti
%                is replaced with zero
%          .Lhn: Lh vector, where each parameter to estimate is
%                replaced with NaN and each fixed parameter is replaced
%                with zero
%            .x: parameter vector, including fixed and variable
%                parameters
%        .nparm: number of parameters
%           .xi: array of ones and zeros, where each one indicates a
%                parameter in x that is to be estimated and each zero
%                indicates a fixed parameter
%           .xv: array of parameters to estimate
%           .xf: array of fixed parameters
%        .phirs: product of phi(z) and Phi(z^freq)
%         .thrs: product of theta(z) and Theta(z^freq)
%         .nr  : number of unit roots in the model
%         .ns  : number of seasonal unit roots in the model (not used)
%         .xd  : parameter vector for betaor (the unit root part),
%                including fixed and variable parameters
%       .nparmd: number of parameters for the unit root part
%         .xid : array of ones and zeros, as in xi, for the unit root
%                part
%         .xvd : array of parameters to estimate for the unit root part
%         .xfd : array of fixed parameters for the unit root part
%
%      Z,T,H,G,X,W matrices of the state space model
%
%                 Y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%          alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%          where epsilon_t is (0,sigma^2I),
%
%          with initial state
%
%          alpha_1= c + W_0*beta + a_1 + A_1*delta
%
```

```
%          where c is (0,Omega) and delta is (0,kI) (diffuse)
%
%      More specifically:
%            .Z: a (s x nalpha) matrix
%            .T: a (nalpha x nalpha) matrix
%            .H: a (nalpha x nepsilon) matrix
%            .G: a (s x nepsilon) matrix
%            .X: []
%            .W: []
%
%     Note: user can subsequently incorporate regression variables into
%           the state space model given by suvarmapqPQe by an
%           appropriate specification of matrix X
%----------------------------------------------------
```

298 sylvesterf

```
function [Rd,fil,col] = sylvesterf(A, da, n, m, du);
% Given A, a polynomial matrix with n rows, m columns and degree da,
% this function computes its row-permuted Sylvester Matrix Rd. The
% permutation is done using the kronecker square commutation matrix of
% dimension n times da+du+1 Comm(n, da+du+1) (Actually this matrix is
% not computed and, equivalently but with fewer operations, the row
% permutation is done by the permat routine)
```

299 sympmeq

```
function [X,ierror]=sympmeq(phi,Lp)
%
% This function solves the symmetric polynomial matrix equation
%    X(z)phi'(z^{-1}) + phi(z)X'(z^{-1}) = Lp(z,z^{-1}),
% where Lp is a symmetric Laurent polynomial matrix of the form
%    Lp(z,z^{-1})= L'_pz^{-p}+.....+L'_1z^{-1}+L_0+L_1z+...L_pz^p,
% L_0 is a symmetric matrix, and p=degree(phi(z))= degree(X(z)).
%
% Input: phi = [n,n,p+1] matrix containing the phi polynomial matrix
%        Lp  = [n,n,p+1] matrix containing L_0+L_1z+...L_pz^p
% Output: X  = [n,n,p+1] matrix containing the solution X(z)
%        ierror = 0,1  a flag for errors in dimensions
```

300 tabla

```
function tabla(y,datei,info)
%
% This function produces a table for a time series
%
% Input arguments:
%              y: the time series
%          datei: a structure containing the initial date
%           info: a structure containing printing options,
%  where
%           .fid: the device on which the table will be written
%          .fh: flag for header and years
%            .wd: format width
%            .nd: number of decimal points
%          .scale: =1 scale data if necessary
%                  =0 do not scale data
%
%    Note: this routine can be used to print data without a header and
%          years in several columns using datei for the number of
%          columns.
% Example: datei=cal(1900,1,6) and info.fh=0 will print only the data
%          with six columns. The year 1900 is immaterial here.
```

301 tasa

```
function yt=tasa(y,s)
%*********************************************************************
%  This function computes growth rate of order s
%
%   INPUTS:
%       y : data vector
%       s : integer, order of the growth rate of y
%
%   OUTPUT:
%      yt : growth rate of y of order s
```

302 tfeasy

```
function [out,ser] = tfeasy(y,x,freq,varargin)
%*********************************************************************
%                        EASY TRANSFER FUNCTION MODELING
```

```
%
%                              USAGE :
% out = tfeasy(y,x,freq,'option1',optionvalue1,'option2',optionvalue2,...)
%
%       INPUTS :
%------------------
%      REQUIRED
%            y : (ly x 1) array containing the series;
%            x : (ly x ni) array containing the inputs
%         freq : data frequency (number of observations per year)
%------------------
%       OPTIONS
%  '[bg_year bg_per]': (1 x 2) array containing the initial year and the
%                       initial period. Default [2000 1]
%               'lam': data transformation (logs), = 0 logs, =1 no logs,
%                      default -1 (test for logs)
%          '[p dr q]': (1 x 3) array containing the regular orders
%                      default: [0 1 1]
%        '[ps ds qs]': (1 x 3) array containing the first seasonal orders
%                      default: [0 1 1]
%                 'S': second seasonality. Default 0
%           '[dS qS]': (1 x 2) array containing the second seasonal orders
%                      default: [1 1]
%             'flagm': flag for mean, =1 mean, =0, no mean, default 0
%                      It has not effect with automatic model
%                      identification
%              'pfix': index array for fixed parameters
%              'vfix': array for fixed parameter values
%            'fixdif': flag for fixing the differencing degrees, =1
%                      degrees are fixed, = 0 not fixed, default 0
%            'autmid': flag for automatic model identification, = 1,
%                      perform automatic model identification, = 0, no
%                      automatic model identification, default 1
%                 'Y': array for regression variables, default []
%          'rnamesrg': string matrix for names of regression variables,
%                      default []
%           'nlestim': flag for nonlinear estimation, = 1, nl estimation,
%                      = 0, no nl estimation, default 1
%               'mvx': flag for nl method, = 1, exact maximum likelihood,
%                      = 0, unconditional least squares, default 1
%               'npr': number of forecasts, default 0
%            'olsres': flag for OLS residuals, = 1, OLS residuals are used,
%                      = 0, uncorrelated residuals (transformation of OLS
%                       residuals) are used, default 0
%                 'pr': flag for printing in an external file, = 1, printing
%                        = 0, no printing, default 1
%                'gft': flag for graphics, = 1, plot series, = 0, no plots
%                        = 2, plots are saved but not displayed, = 3, plots
%                        are both saved and displayed, default 0
%                'out': out = 1 perform outlier detection
%                       = 0 do not perform outlier de
%               'omet': omet = 1 use exact ML for model estimation
%                       = 0 use Hannan-Rissanen
%                  'C': critical value for outlier detection
%                       if negative, it is computed depending on the
%                       sample size
%                 'C0': critical value for outlier detection used in the log
%                       test and automatic model identification, default
%                       C0=2.6 + log(log(ny)) (ny = series length)
%               'schr': = 0 outliers of type AO and TC are considered, =1
%                       outliers of type AO, TC and LS are considered,
%                       default 1
%                'sp1': (sp1,sp2) span for outlier detection, default sp1 =1
%                       default sp2=ny, where ny = series length
%                'sp2':
%               'trad': = 0 no trading day effect, = 1 TD effect, = -1, test
%                       for TD effect, default 0
%            'tradval': possible number of TD variables (0 is also a value),
%                       default [1 6]
%              'leapy': = 0, no leap year effect, = 1 LP effect, = -1, test
%                       for LP effect, default 0
%              'easte': = 0 no Easter effect, = 1 Easter effect, = -1, test
%                       for Easter effect, default 0
%             'durval': possible days previous to Easter (0 is also a value)
%                       default [4 6]
%              'sname': character array containing the series name
%                       default series1
%            'rnamesi': flag for names of the input variables, = 1, names are
%                       given by the user, =0, names are given by the program,
%                       default 0
%           'rnamesiv': character array containing the names for the input
%                       variables, default []
%           'prelivar': flag for preliminary VAR analysis, = 1, perform VAR
%                       analysis, = 0, no VAR analysis, default 0
%              'delay': array containing the filter delays, if tfident=0 and
%                       prelivar=0
%                'ma': array containing the filter ma degrees, if tfident=0
%                      and prelivar=0
%                'ar': array containing the filter ar degrees, if tfident=0
%                      and prelivar=0
%               'inc': = 0, the initial states in the filter equations to obtain
%                      the filtered variables are equal to zero (not estimated)
%                      = 1, the initial states in the filter equations are
%                      estimated
%          'modinput': structure containing the input models in subfields
%                      phi, theta and sigma2 if subfield mod = 1; default
%                      mod 0. The input model is used to compute the mse,
%                      not the input forecasts. It should contain the
%                      nonstationary part.
%           'modpred': structure containing the input forecasts in subfield
%                      pred. If npr > 0, the user should provide the input
%                      forecasts as an (npr x 1) array for each input,
%                      whether there is a model for the input or not
%           'tfident': flag for automatic TF identification, default 0
%            'backwd': flag for backward elimination in transfer function
%                      identification, default 0
%                'Cb': critical value for backwar elimination in transfer
%                      function identification, default 2.
%            'nlagtf': number of lags for automatic model identification. If
%                      negative, the program will compute the number of lags.
%                      default, -1
%           'maxndtf': maximum degree for numerator in transfer function
%                      identification
%           'maxddtf': maximum degree for denominator in transfer function
%                      identification%
%
%       OUTPUT : a structure, the output of function arimaestni
%------------------
%
%     Examples:
%
%   [out,ser]=tfeasy(y,x,freq,'tfident',1,'out',1)
%   out=arimaeasy(y,x,freq,'[p dr q]',[0 1 1],'leapy',-1,'tfident',1)
```

303 tfivparm

```
function parm= tfivparm(Y,parm)
%
```

```
% this function adds to the structure parm the fields ninput and inputv
%
% Input arguments:
% Y    : a matrix containing the input variables
% parm : a structure where
%  s:  seasonality
%  S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
%
% Output arguments:
% parm: the input structure with the added fields
% .ninput: number of inputs
% .inputv: array containing the input variables
```

304 trade

```
function Y = trade(Iy,Im,N,Itrad,Ntrad,Yh,Mq)
% this function generates the trading day variables and the leap year
% variable.  It works until 2100.
%
% input variables       Iy      : the initial year
%                       Im      : the initial period
%                       N       : the length of the desired vector
%                       Itrad   : the number of trading day variables
%                                 =1 one trading day variable (sundays
%                                 and saturdays are the holidays)
%                                 =2 like 1, but the leap year variable
%                                 is added
%                                 =6 six trading day variables are
%                                 considered
%                                 =7 like 6, but the leap year variable
%                                 is added
%                       Ntrad   : a flag for additional holydays (=1
%                                yes, =0 no)
%                       Yh      : the variable where the additional
%                                holidays are stored
%                       Mq      : the series frequency (=12 for
%                                 monthly, =4 for quarterly)
%
% output variables      Y       : N x Itrad array containing the
%                                trading day variables
```

305 tradid

```
function oparm=tradid(y,Y,infm,parm,ser,ols,a,tradval,fid,fmarqdt)
%
% this function automatically estimates the number of TD variables for
% an ARIMA model with TD correction
%
% Input arguments:
% y: vector containing the data
% Y: matrix containing regression variables
%   infm     : structure containing function names and optimization
%              options
%   .f  :   a function to evaluate the vector ff of individual functions
%           such that ff'*ff is minimized
%   .tr :   >0 x is passed from marqdt to f but not passed from f to
%           marqdt
%           =0 x is passed from marqdt to f and passed from f to marqdt
%   .tol:   a parameter used for stopping
%   .jac:   =1 evaluation of jacobian and gradient at the solution is
%           performed
%           =0 no evaluation of jacobian and gradient at the solution
%           is performed
% .maxit:   maximum number of iterations
%   .nu0:   initial value of the nu parameter
%   .prt:   =1 printing of results
%           =0 no printing of results
% parm: astructure containing model information, where
% .s:  seasonality
% .S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
% ser   : a structure containing the series parameters (the ones
%         specified by the user in the spec file and the default ones)
% ols   : = 1, perform OLS, = 0, use the Durbin Levinson algorithm in
%         the HR method
% a     : an integer, the degree of the AR approximation in the first
%         step  of the Hanna-Rissanen method.
% tradval: an integer array containing the possible numbers of TD
%           variables (0 is also a value)
% fid    : the number of the external output file
% fmarqdt: a parameter for the estimation method
%           = 1 Levenberg-Marquardt method
%           = 0 Lsqnonlin (Matlab)
```

306 trtout

```
function trtout(fid,iout,ser)
%
% This function prints the results of outlier detection
```

307 tskfsribf

```
function [e,f,hd,Md,A,P,ne]=tskfsribf(y,X,Z,G,W,T,H,ins,i,chb)
%
%
%        This function applies the tskf-sribf to the series y for
%        prediction and likelihood evaluation corresponding to the model
%
%        y_t = X_t*beta + Z_t*alpha_t + G_t*epsilon_t
%        alpha_{t+1}= W_t*beta + T_t*alpha_t + H_t*epsilon_t
%
%        where epsilon_t is (0,sigma^2I),
%
%        with initial state
%
%        alpha_1= c + W_0*beta + a_1 + A_1*delta
%
```

```
%        where c is (0,Omega) and delta is (0,kI) (diffuse). A single
%        collapse is applied to get rid of the diffuse component.
%
%        Input parameters:
%        y:     an (n x p) matrix of observations;
%        X    : an (n*p x nbeta) matrix containing the X_t matrices;
%               a  (p x nbeta) if it is time invariant;
%               it can be []
%        Z    : an (n*p x nalpha) matrix containing the Z_t matrices;
%               a  (p x nalpha) matrix if it is time invariant
%        G    : an (n*p x nepsilon) matrix containing the G_t matrices;
%               a  (p x nepsilon) matrix if it is time invariant
%        W    : an (n*nalpha x nbeta) matrix containing the W_t
%               matrices;
%               an (nalpha x nbeta) matrix if it is time invariant;
%               it can be []
%        T    : an (n*nalpha x nalpha) matrix containing the T_t %               matrice
%               an (nalpha x nalpha) matrix if it time invariant
%        H    : an (n*nalpha x nepsilon) matrix containing the H_t %               matric
%               an (nalpha x nepsilon) if it is time invariant
%        ins: an nalpha x (cc+cw0+ca1+cca1) matrix containing the
%             initial state information, according to array i below
%        i    : a  1 x 4 array containing 4 integers, i=[cc cw0 ca1
%              cca1], where
%             cc   = nalpha if c is not missing (0 if c missing)
%             cw0  = number of columns in W_0 (0 if W_0 missing)
%             ca1  = 1 if a_1 is not missing (0 if a_1 missing)
%             cca1 = number of columns in A_1 (0 if A_1 missing)
%       chb= 1 compute hb and Mb
%            0 do not compute hb and Mb
%
%        Output parameters:
%        e  : vector containing the stack of the standardized residuals
%        f  : factor by which the residuals are to be multiplied
%              for minimization of the nonlinear sum of squares
%        hd : the beta estimate
%        Md : the Mse of hd
%        A  : the estimated augmented state vector at the end of
%             filtering
%        P  : the Mse of A at the end of filtering
%        ne : vector containing the stack of the observation numbers
%             corresponding to the standardized residuals
```

308 tsplot

```
function tsplot(y,cstruc,varargin)
% PURPOSE: time-series plot with dates and labels
%---------------------------------------------------
% USAGE:     tsplot(y,cstruc,begp,endp,vnames,ydigit)
%        or: tsplot(y,cal_struc,vnames), which plots the entire series
%            with the default date format
%        or: tsplot(y,cal_struc,vnames,ydigit), which plots the entire
%            series with the date format ydigit
%        or: tsplot(y,cal_struc), entire series with no variable names
%        or: tsplot(y,cal_struc,[],ydigit), entire series with the date
%            format ydigit but no variable names
%
% where:  y      = matrix (or vector) of series to be plotted
%         cstruc = a structure returned by cal()
%         begp   = the beginning observation to plot (optional)
%         endp   = the ending observation to plot (optional)
%        vnames  = a string matrix of names for a legend (optional)
%                 e.g. vnames = ['y    ',
%                               'x1   ',  NOTE: fixed width
%                               'x2   ',        like all MATLAB
%                               'cterm'];       strings
%        ydigit  = a string specifying the date format (optional)
%                 e.g. ydigit = 'yyyy'
%                      ydigit = 'mmmyyyy'
%---------------------------------------------------
% e.g.   cstr = cal(1970,1,12);
%        tsplot(y,cstr); would plot all data
%
%    or: tsplot(y,cstr,ical(1981,1,cstr),ical(1981,12,cstr)),
%         which would plot data for 1981 only
%---------------------------------------------------
% Original version of tsplot written by:
% James P. LeSage, Dept of Economics
% University of Toledo
% 2801 W. Bancroft St,
% Toledo, OH 43606
% jpl@jpl.econ.utoledo.edu
%
% Modified by Martyna Marczak, 20.09.2012
% Department of Economics (520G)
% University of Hohenheim
% Schloss, Museumsfluegel
% 70593 Stuttgart, Germany
% Phone: + 49 711 459 23823
% E-mail: marczak@uni-hohenheim.de
```

309 tukhan

```
function [w, m] = tukhan(n, width)
%
%        This function computes the weights for the
%        Tukey-Hanning window
%
%     INPUTS:
%         n : lentgh of the series
%     width : window width factor (1/3 by default)
%             if width <= 0, it is set to 1/3
%
%    OUTPUTS:
%         w : weights of the Tukey-Hanning window
%         m : window lag size
```

310 updatef

```
function [f,fc]=updatef(ff,ffc)
%
% function to update the vector whose nonlinear sum of squares is
% minimized. It is stored in the form f=(f^(1/n))*(2^(fc/n) ) to avoid
% underflow and overflow.
```

311 updbic

```
function  [bicm,oparm]=updbic(yd,beta,s,S,p,ps,q,qs,qS,ols,a,bicm,oparm)
%
% this function updates the BIC criterion of an ARMA model after
% checking for stationarity and invertibility
%
% Input arguments:
% yd   : an (n x m) matrix containing the series, yd(:,1), and an (n x
%        m-1) matrix of regression variables if m > 1.
% beta : an m-1 vector containing the OLS estimators if m > 1, empty if
%        m = 1
% s    : seasonality
% S    : second seasonality
% p    : degree of AR polynomial
% ps   : degree of AR seasonal polynomial
% q    : degree of MA polynomial
% qs   : degree of MA seasonal polynomial
% qS   : degree of MA second seasonal polynomial
% ols  : = 1, perform OLS, = 0, use the Durbin Levinson algorithm
% a    : an integer, the degree of the AR approximation in the first
%        step of the Hanna-Rissanen method.
% bicm : the previous bic criterion
% oparm: a structure where
% .s:  seasonality
% .S:  second seasonality
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
```

312 usmeasy

```
function [out,ser] = usmeasy(y,freq,varargin)
%*********************************************************************
%                        EASY STRUCTURAL MODELING
%
%                             USAGE :
% out = usmeasy(y,freq,'option1',optionvalue1,'option2',optionvalue2,...)
%
%       INPUTS :
%------------------
%      REQUIRED
%            y : (ly x 1) array containing the series;
%         freq : data frequency (number of observations per year)
%------------------
%       OPTIONS
%  '[bg_year bg_per]': (1 x 2) array containing the initial year and the
%                    initial period. Default [2000 1]
%               'lam': data transformation (logs), = 0 logs, =1 no logs,
%                      default -1 (test for logs)
%                 'Y': (n x nY) array for regression variables, where n is
%                      the series length plus the number of forecasts and
%                      nY is the number of regression variables, default []
%             Ycomp  : a cell array, containing the assignment of each
%                      regression variable to a component. Possible values
%                      are 'level','slope','seas','cycle', 'ar' and 'irreg'
%          'rnamesrg': string matrix for names of regression variables,
%                      default []
%                 'W': (n*nalpha x nbeta) array for the transition equation
%                      of the state space model, where n is the series
%                      length plus the number of forecasts, nalpha is the
%                      state vector length and nbeta is the number of
%                      intervention effects to be modeled this way, default
%                      []
%             'level': (1 x 3) array to specify the level
%             'slope': (1 x 3) array to specify the slope
%             'cycle': (1 x 3) array to specify the cycle
%            'cyclep': (2 x 2) array to specify the rho and alpha
%                      parameters of the cycle
%            'cycleb': (1 x 2) array to specify the cyclical interval
%              'seas': (1 x 3) array to specify the seasonal component
%                'ar': (1 x 3) array to specify the autoregressive
%                      component
%               'arp': (2 x p) array to specify the autoregressive
%                       parameters
%            'conout':'level' if the standard error of the level is
%                      concentrated out
%                      'slope' if the standard error of the slope is
%                      concentrated out
%                      'seas' if the standard error of the seasonal
%                      is concentrated out
%                      'cycle' if the standard error of the cycle is
%                      concentrated out
%                      'ar' if the standard error of the ar component is
%                      concentrated out
%                      'irreg' if the standard error of the irregular is
%                      concentrated out
%                      If .conout is not input, the program will determine
%                      the biggest variance.
%           'sqrtfil': =1 use the square root Kalman filter, =0 do not use
%                       it, default 0
%            'nlestim': flag for nonlinear estimation, = 1, nl estimation,
%                       = 0, no nl estimation, default 1
%                'npr': number of forecasts, default 0
%             'olsres': flag for OLS residuals, = 1, OLS residuals are used,
%                       = 0, uncorrelated residuals (transformation of OLS
%                       residuals) are used, default 0
%                 'pr': flag for printing in an external file, = 1, printing
%                        = 0, no printing, default 1
%                'gft': flag for graphics, = 1, plot series, = 0, no plots
%                        = 2, plots are saved but not displayed, = 3, plots
%                        are both saved and displayed, default 0
%              'sname': character array containing the series name
%                       default series1
%-----------------------------------------------------------------------
% * codes for the components:
%   level = -1  constant
%            1  stochastic
%            2  Butterworth tangent
%   slope = -1  constant
%            1  stochastic
%   seas  = -1  fixed dummy seasonality
%            1  stochastic dummy seasonality
%            2  trigonometric seasonality
%            4  Butterworth tangent
%   cycle =  1  structural model cycle
%            2  Butterworth sine cycle
%   irreg =  1  stochastic
%   ar    =  k  autoregressive component of order k
%
%----------------------------------------------------
%
%
%       OUTPUT : a structure, the output of function arimaestni
%------------------
%
%     Example:
%
%   [out,ser]=usmeasy(y,freq,'pr',1,'gft',1,'sname','myusmseries',...
%                      'level',[1 0.1 NaN],'slope',[1 0.1 NaN],'seas',...
%                      [2 0.1 NaN],'irreg',[1 0.1 NaN]);
```

313 usmestim

```
function  [result,str] = usmestim(y,str)
%
% This function estimates a univariate structural model using the exact
% maximum likelihood method.
%
%
% Inputs:
%      y: matrix containing the output series
%    str: a structure containing the initial model information. It
%         should be input as well as output because the concentrated
%         parameter can change. The concentrated parameter should be the
%         greatest variance. The program performs a preliminary
%         estimation to check it.
%  Outputs:
%       result: a structure with the following fields
%        .xvf : estimated parameters
%        .xf : vector of fixed parameters
%    .sigma2c: the estimated standard error of the parameter that has
%              been concentrated out
%       .Pevf: prediction error variance
%      .SPevf: square root of Pevf
%       .tv  : t-values of the estimated parameters
%        .e  : vector of standardized residuals at the end of estimation
%              (Q'_2*y)
%        .Ss : residual sum of squares (e'*e)
%        .F  : vector of nonlinear functions whose sum of squares is
%              minimized at the end of estimation
%        .Ff : the product F'*F
%        .h  : vector of estimated regression estimates
%        .M  : matrix of mse of h
%        .A  : estimated state vector, x_{t|t-1}, obtained with the
%              Kalman filter at the end of the sample
%        .P  : Mse of A
%       .tvr : t-values of the estimated regression parameters
%       .ser : standard errors of the estimated regression parameters
%    .ferror : flag for errors
%        str : the same structure as input. It should be present because
%              the concentrated parameter may change
```

314 usmestimm

```
function  [result,str] = usmestimm(y,str)
%
% This function estimates a univariate structural model with complex
% seasonal patterns using the exact maximum likelihood method.
%
%
% Inputs:
%      y: matrix containing the output series
%    str: a structure containing the initial model information. It
%         should be input as well as output because the concentrated
%         parameter can change. The concentrated parameter should be the
%         greatest variance. The program performs a preliminary
%         estimation to check it.
%  Outputs:
%       result: a structure with the following fields
%        .xvf : estimated parameters
%        .xf : vector of fixed parameters
%    .sigma2c: the estimated standard error of the parameter that has
%              been concentrated out
%       .Pevf: prediction error variance
%      .SPevf: square root of Pevf
%       .tv  : t-values of the estimated parameters
%        .e  : vector of standardized residuals at the end of estimation
%              (Q'_2*y)
%        .Ss : residual sum of squares (e'*e)
%        .F  : vector of nonlinear functions whose sum of squares is
%              minimized at the end of estimation
%        .Ff : the product F'*F
%        .h  : vector of estimated regression estimates
%        .M  : matrix of mse of h
%        .A  : estimated state vector, x_{t|t-1}, obtained with the
%              Kalman filter at the end of the sample
%        .P  : Mse of A
%       .tvr : t-values of the estimated regression parameters
%       .ser : standard errors of the estimated regression parameters
%    .ferror : flag for errors
%        str : the same structure as input. It should be present because
%              the concentrated parameter may change
```

315 usmestni

```
function outa = usmestni(dbname,ser)
%
% function to identify, estimate and forecast an ARIMA model for one
% series. The series may have up to two seasonalities. The ARIMA model
% is of the form:
%
% phi(B)*phi_s(B^s)*phi_S(B^S)*(delta*delta_s*delta_S*y_t -mu) =
% th(B)*th_s(B^s)*th_S(B^S)*a_t
%
% In the subdirectory spec, there is a specification file where all the
% options for the ARIMA model are defined and returned in the structue
% ser. The name of this file is given in function arimaestos and passed
% to this function.
% These options include, log transformation criteria, automatic
% identification of ARMA model and differencing operators, automatic
% specification of trading day, Easter effect and leap year (for
% quarterly and monthly series only), outlier search and forecasting,
% among other things.
% No automatic model identification is performed for the second
% seasonality (S). This part must be entered by the user. Automatic
% model identification is performed for the regular and the first
% seasonal part. Output is written in an external file in the
% subdirectory results.
%
%     INPUTS:
%   dbname : name of the series
%   ser    : a structure, containing the instructions for this function
%   fidr   : an integer, corresponding to the output file
%   ii     : an integer, corresponding to the series currently handled.
%
%  OUTPUTS:
%  outa  : a structure containing model information for the input
%          with fields:
%   title: a string with the name of series
%  nziyip: a 1 x 3 array with number of obs., initial year, initial per.
%    freq: number of seasons
%    orig: original series
%   model: structre with model information. It contains the following
%          fields
%      lam: flag for logarithmic transformation, = 0, take logs, = 1,
%             do not take logs
%          X: X matrix in the state space form
%          Z: Z matrix in the state space form
%          G: G matrix in the state space form
%          W: W matrix in the state space form
%          T: T matrix in the state space form
%          H: H matrix in the state space form
%        ins: ins matrix for the initial conditions
%          i: i array for the initial conditions
%     resinf: structure containing residual information
%      sconp: residual standard error
%    StochCc: matrix containing the stochastic components
%   StochSCc: matrix containing the mse of the stochastic components
%   oStochCc: matrix containing the stochastic components in the
%             original scale
% oStochSCc: matrix containing the mse of the stochastic components in
%             the original scale
%         Cc: matrix containing the components including deterministic
%             effects
%        SCc: matrix containing the mse of Cc
%        oCc: matrix containing the Cc in the original scale
%       oSCc: matrix containing the mse of the oCc
%        npr: number of forecasts
%         Xp: matrix containing the forecasts of X
%         Wp: matrix containing the forecasts of W
%        pry: forecasts
%       spry: mse of the forecasts
%       alpr: matrix containing the forecasts of the state vector
%      malpr: three dimensional array containing each of the covariance
%             matrices of alpr
%      salpr: matrix containing the mse of alpr
%       opry: forecasts in the original scale
%      ospry: mse of the forecasts in the original scale
%      oalpr: matrix containing the alpr in the original scale
%     osalpr: matrix containing the mse of oalpr
%        ser: the input structure
%     result: a structure containing estimation results. It has
%             the following fields:
%        xvf: array containing the estimated parameters
%         xf: array containing the fixed parameters
%          e: array containing the residuals
%         Ss: residual sum of squares
%         Ff: the product F'*F
%    sigma2c: standard error of the parameter concentrated out of the
%            likelihood
%      Pevf: prediction error variance
%     SPevf: square root of Pevf
%        tv: t-values of the estimated parameters
%        se: standard errors of the estimated parameters
%        .F: vector of nonlinear functions whose sum of squares is
%            minimized at the end of estimation
%         h: vector of regression parameters
%         M: mse of h
%         A: Augmented state vector at the end of filtering
%         P: mse matrix of A at the end of filtering
%    olsres: OLS residuals
%       tvr: t-values of the regression parameters
%       ser: standard errors of the regression parameters
%    ferror: flag for errors
```

316 usmestos

```
function outa = usmestos(fname,fmeta)
%
% Function for automatic identification, estimation and forecasting of
% ARIMA or transfer function models for one or several series
%
%     INPUTS:
%  fname : If fmeta = 0 or absent, a string such that fname.m is a
%          matlab function in the spec subdirectory that returns the
%          structure ser. In this structure, instruction for this
%          function are given. If fmeta = 1, a string such that
%          fname.txt contains a list of names of matlab functions in
%          the spec subdirectory that will be treated sequentially.
%  fmeta : = 0, fname.m is a matlab function in the spec subdirectory
%          that returns the structure ser; = 1, fname.txt is a file
%          that contains a list of matlab functions in the spec
%          subdirectory that will be treated sequentially. If not
%          input, the program sets by default fmeta = 0,
%
%  OUTPUTS:
%  outa  : a structure containing model information for the input
%          with fields:
%   title: a string with the name of series
%  nziyip: a 1 x 3 array with number of obs., initial year, initial per.
%    freq: number of seasons
%    orig: original series
%   model: structre with model information. It contains the following
%          fields
%      lam: flag for logarithmic transformation, = 0, take logs, = 1,
%           do not take logs
%        X: X matrix in the state space form
%        Z: Z matrix in the state space form
%        G: G matrix in the state space form
%        W: W matrix in the state space form
%        T: T matrix in the state space form
%        H: H matrix in the state space form
%      ins: ins matrix for the initial conditions
%        i: i array for the initial conditions
%    resinf: structure containing residual information
%     sconp: residual standard error
%   StochCc: matrix containing the stochastic components
%  StochSCc: matrix containing the mse of the stochastic components
%  oStochCc: matrix containing the stochastic components in the
%           original scale
% oStochSCc: matrix containing the mse of the stochastic components in
%           the original scale
%       Cc: matrix containing the components including deterministic
%           effects
%      SCc: matrix containing the mse of Cc
%      oCc: matrix containing the Cc in the original scale
%     oSCc: matrix containing the mse of the oCc
%      npr: number of forecasts
%       Xp: matrix containing the forecasts of X
%       Wp: matrix containing the forecasts of W
%      pry: forecasts
%     spry: mse of the forecasts
%     alpr: matrix containing the forecasts of the state vector
%    malpr: three dimensional array containing each of the covariance
%           matrices of alpr
%    salpr: matrix containing the mse of alpr
%     opry: forecasts in the original scale
%    ospry: mse of the forecasts in the original scale
%    oalpr: matrix containing the alpr in the original scale
%   osalpr: matrix containing the mse of oalpr
%      ser: the input structure
%   result: a structure containing estimation results. It has
%           the following fields:
%      xvf: array containing the estimated parameters
%        xf: array containing the fixed parameters
%         e: array containing the residuals
%        Ss: residual sum of squares
%        Ff: the product F'*F
%   sigma2c: standard error of the parameter concentrated out of the
%            likelihood
%      Pevf: prediction error variance
%     SPevf: square root of Pevf
%        tv: t-values of the estimated parameters
%        se: standard errors of the estimated parameters
%        .F: vector of nonlinear functions whose sum of squares is
%            minimized at the end of estimation
%         h: vector of regression parameters
%         M: mse of h
%         A: Augmented state vector at the end of filtering
%         P: mse matrix of A at the end of filtering
%    olsres: OLS residuals
%       tvr: t-values of the regression parameters
%       ser: standard errors of the regression parameters
%    ferror: flag for errors
```

317 var_est

```
function res = var_est(y,nlag,test,x)
% PURPOSE: performs vector autogressive estimation
%          and returns a structure
%---------------------------------------------------
% USAGE: res = var_est(y,nlag,test,x)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           nlag = the lag length
%           test = a logical variable to perform additional tests
%           x    = optional matrix of variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: a structure containing the following fields
%    .resid   = residuals
%    .phi     = VAR matrix polynomials. Signs are those of Box-Jenkins
%               (I-phi_1*z-phi_2*z^2 - ...-phi_p*z^p)
%    .phitv   = matrix polynomials containing the t-values and
%               corresponding to the VAR matrix polynomials
%    .const   = vector containing the estimated constant
%    .consttv = vector containing the t-values of the estimated
%               constant
%    .betavar  = matrix containing the estimated regression
%                coefficients, beta
%    .tvvar      = t-values of beta
%    .sigmar     = covariance matrix of residuals
%    .covvecbeta = covariance matrix of vec(beta)
%    .corvecbeta = correlation matrix of vec(beta)
%    .dusigmar   = determinant of the maximum likelihood estimator of
%                  the covariance matrix of residuals
%    .llkhd      = log-likelihood
%    .aic        = aic
%    .bic        = bic
%    .ssr(j)     = Sum-of-squares residuals for each equation. Only
%                  if test = 1
%    .Rsqr(j)    = R^2 for each equation. Only if test = 1
%    .SEeq(j)    = Standard error of regression  for each equation.
%                  Only if test = 1
%    .Fstat(j)   = F-statistic for each equation. Only if test = 1
%    .llkhdeq(j) = log-likelihood for each equation. Only if test = 1
%    .aiceq(j)   = aic for each equation. Only if test = 1
%    .biceq(j)   = bic for each equation. Only if test = 1
%    .ftest(i,j) = Granger F-tests, only if test = 1
%    .fprob(i,j) = Granger marginal probabilities, only if test = 1
%---------------------------------------------------
```

318 var_res

```
function   resid = var_res(y,nlag,x)
% PURPOSE: performs vector autogressive estimation
%          and returns only residuals
%---------------------------------------------------
% USAGE: resid = var_res(y,nlag,x)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           nlag = the lag length
%           x    = optional matrix of variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: a matrix of residuals (nobs x neqs)
%---------------------------------------------------
```

319 varident

```
function [lg,initres] = varident(y,maxlag,minlag,prt,x)
% PURPOSE: performs likelihood ratio test, bic,
%                   and aic for var model to determine
%                   optimal lag length
%---------------------------------------------------
% USAGE:  [lagsopt,initres] = varident(y,maxlag,minlag,prt,x)
% where:    y      = an (nobs x neqs) matrix of y-vectors
%           maxlag = the maximum lag length
%           minlag = the minimum lag length
%           prt    = flag for printing
%                    0 = no, 1 = yes (default = 0)
%           x      = optional matrix of variables (nobs x nx)
%                   (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: lagsopt = the optimum number of lags
%          initres = an (maxlag x neqs) matrix of initial residuals
%                    corresponding to the estimated VARXs of order
%                    1,2,..., maxlag.
%---------------------------------------------------
```

320 varimass

```
function z=varimass(l,N,H,F,K,A,Sigma,Xi,stda,seed)
%
%        This function generates a VARMA model
%        It uses the state space representation
%        stda: standard deviation of the innovations in percentage of
%        the levels (default: 1
```

321 varmafil

```
function [z,rx1] = varmafil(u,F,H,B,D,kro,inc)
% This function filters the series u_t  using the filter H(z) =
% omega(z)^(-1)*delta(z), where omega(z)=omega_0 + omega_1*z +
% omega_2*z^2 + ....+omega_q*z^q and delta(z) = I + delta_1*z + ... +
% delta_p*z^p.
%
% Input arguments:
% u: the input series
% F: a matrix
% H: a matrix
% B: a matrix
% D: a matrix
% kro: a vector containing the Kronecker indices of the filter H(z)
% such that
%
%  x_{t+1}} = F*x_{t} + B*u_{t}
%  z_{t}    = H*x_{t} + D*u_{t}
%
% inc: = 1 initial conditions, x_{1}, for the filter are estimated
%      = 0 initial conditions equal to zero
%
% Output arguments:
% z  : the output series
% rx1: a matrix containing the design matrix to estimate x_1
```

322 varmafilp

```
function [z,sz] = varmafilp(u,delta,omega,phi,th,Phi,Th,Sigma,freq)
% This function filters the series u_t  using the filter H(z) =
% omega(z)^(-1)*delta(z), where omega(z)=omega_0 + omega_1*z + omega_2*z^2
% + ....+omega_q*z^q and delta(z) = I + delta_1*z + ... + delta_p*z^p. The
% series u can follow a VARMApqPQ model or not. If it follows no model, it
% is filtered using zeros as starting values. The series followed by u is
% multiplicative seasonal of the form
%
%   phi(B)Phi(B)u_t = th(B)Th(B)a_t
%
% If there are unit roots, they are assumed to be in phi, Phi or delta
%
% Input arguments:
%     u: the input series
% omega: the filter denominator, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
% delta: the filter numerator, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
%   phi: the regular AR part, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
%   Phi: the seasonal AR part, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
%    th: the regular MA part, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
%    Th: the seasonal MA part, as a MATLAB polynomial in the scalar case,
%        and as a matrix polynomial in the vector case
% Sigma: the covariance matrix of the input innovations
%  freq: the number of seasons
%
% Output arguments:
% z  : the filtered series
% sz : the mse of z
```

323 varmapqPQ2ssm

```
function [phirs,thrs,H,F,G,J,ferror] = varmapqPQ2ssm(phi,th,Phi,Th,L,str)
% PURPOSE: given the matrix polynomials of a VARMA(p,q)(P,Q)_s model and
% the Cholesky factor of the innovations covariance, this function puts
% the model into the state space form
%   x(t+1) = F*x(t) + G*u(t)
%    y(t)  = H*x(t) + J*u(t),
% where
%     [-bphi_1     I 0  ... .... 0]        [ theta_1-phi_1*Psi0       ]
%     [-bphi_2     0 I  ... .... 0]        [ theta_2-phi_2*Psi0       ]
% F = [  ...            ...  ...  ],   G = [      ...                 ],
%     [-bphi_{r-1} 0 0  ...   ... I]        [ theta_{r-1}-phi_{r-1}*Psi0]
%     [-bphi_r     0 0  ...   ... 0]        [ theta_r-phi_r*Psi0        ]
% H =    [phi_0^{-1} 0 0  ...   ... 0],  J =  Psi_0,
% bphi_i = phi_i*phi_0^{-1} and phi^{-1}(z)*theta(z) = Psi_0 + Psi_1*z
% + Psi_2*z^2+ ...
%
%and computes the AR and MA matrix polynomials
%---------------------------------------------------
% USAGE: [phirs,thrs,H,F,G,J,ferror] =
%              varmapqPQ2ssm(phi,th,Phi,Th,L,str)
% where:   phi      = the regular AR matrix polynomial
%          th       = the regular MA matrix polynomial
%          Phi      = the seasonal AR matrix polynomial
%          Th       = the seasonal MA matrix polynomial
%          L        = the Cholesky factor of the innovations covariance
%                     matrix
%          str      = a structure containing model information
%---------------------------------------------------
%---------------------------------------------------
% RETURNS: phirs = the overall AR matrix polynomial
%          thrs  = the overall MA matrix polynomial
%          H     = a matrix of the state space form
%          F     = a matrix of the state space form
%          G     = a matrix of the state space form
%          J     = a matrix of the state space form
%      ferror    = flag for errors
%---------------------------------------------------
```

324 varmapqPQestim

```
function  [result,ferror] = varmapqPQestim(y,str,Y)
%
% This function estimates a VARMA(p,q)(P,Q)_s model using the exact
% maximum likelihood method.
%
%
% Inputs: y: matrix containing the input series
%        Y: an (n*neqs x nbeta) matrix containing the regression matrix
%      str: a structure containing the initial model information given
%           by function suvarmapqPQ.m
%  Output: .xvf : estimated parameters
%          .xf : vector of fixed parameters
%     .sigma2c : concentrated parameter estimate
%      .Sigmar : estimated exact covariance matrix of residuals
%          .tv : t-values of the estimated varma parameters
%    .residexct : matrix containing recursive residuals, only if Y is
%                 empty
%           .e : vector of standardized residuals at the end of
%                estimation (Q'_2*y)
%          .ff : vector of nonlinear functions whose sum of squares is
%                minimized at the end of estimation
%          .h  : vector of estimated regression estimates
%          .H  : matrix of mse of h
%          .A  : estimated state vector, x_{t|t-1}, obtained with the
%                Kalman filter at the end of the sample
%          .P  : Mse of A
%          .tvr: vector of t-values for h
%      .ferror : flag for errors
```

325 varmapqPQestimd

```
function  [result,ferror] = varmapqPQestimd(y,str,Y,constant)
%
```

```
% This function estimates a VARMA model with unit roots parameterized in
% terms of the model for the ‘‘differenced’’ series using the exact
% maximum likelihood method.
%
%
% Inputs: y: matrix containing the input series
%         Y: an (n*neqs x nbeta) matrix containing the regression matrix
%       str: a structure containing the initial model information
%  constant: =1 a constant should be included in the model for the
%               differenced series
%             0 no constant in the model for the differenced series
%  Output: .xvf : estimated parameters
%           .xf : vector of fixed parameters
%      .sigma2c : concentrated parameter estimate
%       .Sigmar : estimated exact covariance matrix of residuals
%           .tv : t-values of the estimated varma parameters
%    .residexct : matrix containing recursive residuals, only if Y is
%                 empty
%            .e : vector of standardized residuals at the end of
%                 estimation (Q'_2*y)
%           .ff : vector of nonlinear functions whose sum of squares is
%                 minimized at the end of estimation
%           .h  : vector of estimated regression estimates
%           .H  : matrix of mse of h
%           .A  : estimated state vector, x_{t|t-1}, obtained with the
%                 Kalman filter at the end of the sample
%           .P  : Mse of A
%           .tvr: vector of t-values for h
%       .ferror : flag for errors
```

326 varmapqPQestime

```
function  [result,ferror] = varmapqPQestime(y,str,Y,constant)
%
% This function estimates a VARMA model with unit roots parameterized in
% terms of the model in error correction form using the exact
% maximum likelihood method.
%
%
% Inputs: y: matrix containing the input series
%         Y: an (n*neqs x nbeta) matrix containing the regression matrix
%       str: a structure containing the initial model information
%  constant: =1 a constant should be included in the model for the
%              differenced series
%            0 no constant in the model for the differenced series
%  Output: a structure, result, with the following fields
%           .xvf : estimated parameters
%           .xf  : vector of fixed parameters
%      .sigma2c  : concentrated parameter estimate
%       .Sigmar  : estimated exact covariance matrix of residuals
%           .tv  : t-values of the estimated varma parameters
%    .residexct  : matrix containing recursive residuals, only if Y is
%                  empty
%            .e  : vector of standardized residuals at the end of
%                  estimation (Q'_2*y)
%           .ff  : vector of nonlinear functions whose sum of squares is
%                  minimized at the end of estimation
%           .h   : vector of estimated regression estimates
%           .H   : matrix of mse of h
%           .A   : estimated state vector, x_{t|t-1}, obtained with the
%                  Kalman filter at the end of the sample
%           .P   : Mse of A
%           .tvr: vector of t-values for h
%       .ferror : flag for errors
```

327 varmasim

```
function [z,ferror]=varmasim(l,N,phi,th,stda,seed)
%
%
%       This function generates a VARMA model
%       It uses Akaike's state space representation
%
%       Input parameters:
%       l: number of observations discarded at the beginning of the
%          series
%       N: number of observations of the simulated series
%       phi: AR matrix polynomial
%       th:  MA matrix polynomial
%       stda:  covariance matrix of the innovations  (default: I)
%       seed: a number to start random normal generation
%
%       Output parameters:
%       z: the simulated series
%        ferror: a flag for errors
```

328 varmaxgenid

```
function [order,kro] = varmaxgenid(y,x,seas,maxorder,hr3,ct,prt)
% PURPOSE: identifies a VARMAX model for the series y with inputs x.
% According to the sign of maxorder, it identifies:
% a) if maxorder > 0, a VARMAX model using the generic neighborhood and
%    ct = 'AIC', 'BIC' or 'HQ'.
% b) if maxorder = 0, a VARMAX(p,p,p) model using sequential LR tests.
% c) if maxorder < 0, a VARMAX(p,p,p) model using ct = 'AIC', 'BIC' or
%    'HQ'.
%---------------------------------------------------
% USAGE: [order,kro] = varmaxgenid(y,x,seas,maxorder,hr3,ct,prt)
% where:    y   = an (nobs x neqs) matrix of y-vectors
%           x   = matrix of input variables (nobs x nx)
%                (NOTE: constant vector automatically included)
%        seas   = seasonality
%    maxorder   = >0, maximum order to estimate the McMillan degree
%                 using the generic neighborhood and AIC, BIC, HQ.
%               =  0, estimate the order of the optimum VARMAX(p,p,p)
%                 model by LR tests with maximum order obtained by
%                 the
%                 program. In this case, input ct is ignored (it can be
%                 set to empty for example).
%                 <0, -maxorder is the maximum order to estimate the
%                 McMillan degree using equal K.i. and AIC, BIC, HQ.
%         hr3  = 1 perform only the first two stages of the HR method
%                 0 perform the three stages of the HR method
%          ct   = 'AIC','BIC', 'HQ'
%         prt   = 1 print results of the VARX, VARMAX(p,p,p) tests
%                 and different criteria
%---------------------------------------------------
% RETURNS: order = the system order, that is, the McMillan degree
%            kro = the Kronecker indices
%---------------------------------------------------
```

329 varmaxscmidn

```
function [order,kro,scm] = varmaxscmidn(y,x,seas,maxorder,hr3,prt)
% PURPOSE: identifies a VARMAX model based on scalar component models
% for the series y with inputs x. The series is assumed to follow an
% invertible but possibly nonstationary model.
% The scalar components are identified after a VARMAX(p,q,r) has been
% previously identified and estimated. The estimated  max(p,q,r) is an
% estimate of the maximum Kronecker index.
% If maxorder is empty, the maximum order for the VARMAX(p,q,r) model is
% set equal to the order of a VARX approximation.
% If maxorder is positive, a VARMAX(p,q,r) model is estimated using
% maxorder as maximum for p, q and r.
% The procedure identifies the s.c._i, i, i=1,2,...,s, equation by
% equation. En each equation, first, the past innovations are replaced
% with the estimated innovations of the VARMAX(p,q,r) model of the
% first step. Then, a sequence of LR tests allows for the estimation of
% the s.c. of that equation.
%---------------------------------------------------
% USAGE: [order,kro,scm] = varmaxscmidn(y,x,seas,maxorder,hr3,prt)
% where:    y   = an (nobs x neqs) matrix of y-vectors
%           x   = matrix of input variables (nobs x nx)
%                (NOTE: constant vector automatically included)
%        seas   = seasonality
%    maxorder   = empty, use the order of a VARX approximation as
%                 maximum order for the VARMAX(p,q,r) model to be
%                 identified.
%                 >0, this maximum order is used as the maximum order
%                 for the VARMAX(p,q,r) model to be identified.
%         hr3   = 1 perform only the first two stages of the HR method
%                 0 perform the three stages of the HR method
%         prt   = 1 print results of the VARX and VARMAX(p,p,p) tests
%---------------------------------------------------
% RETURNS: order = the system order, that is, the McMillan degree
%            kro = the Kronecker indices
%            scm = an array containg the scalar component models
%---------------------------------------------------
```

330 varx_est

```
function res = varx_est(y,nlag,x,test,xx)
% PURPOSE: performs vector autogressive with exogenous inputs (VARX)
%          estimation and returns a structure
%---------------------------------------------------
% USAGE: res = varx_est(y,nlag,x,test,xx)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           nlag = the lag length
%          x    = matrix of input variables (nobs x nx)
%                (NOTE: constant vector automatically included)
%          test = a logical variable to perform additional tests
%          xx   = optional matrix of variables (nobs x nxx)
%---------------------------------------------------
% RETURNS: a structure containing the following fields
%     .resid  = residuals
%     .phi    = VARX matrix polynomials corresponding to the outputs.
%              Signs are those of Box-Jenkins  (I-phi_1*z-phi_2*z^2 -
%              ...-phi_p*z^p)
%     .phitv    = matrix polynomials containing the t-values and
%                corresponding to the output VARX matrix polynomials
%     .phix      = VARX matrix polynomials corresponding to the inputs.
%                 Signs are those of Box-Jenkins  (phix_0 -phix_1*z-
%                          -phix_2*z^2 -...-phix_p*z^p)
%     .phixtv   = matrix polynomials containing the t-values and
%                corresponding to the input VARX matrix polynomials
%     .const   = vector containing the estimated constant
%     .consttv = vector containing the t-values of the estimated
%               constant
%     .betava  = matrix containing the estimated regression
%               coefficients, beta
%     .tvvar     = t-values of beta
%     .sigmar   = covariance matrix of residuals
%     .covvecbeta = covariance matrix of vec(beta)
%     .corvecbeta = correlation matrix of vec(beta)
%     .dusigmar   = determinant of the maximum likelihood estimator of
%                  the covariance matrix of residuals
%     .llkhd      = log-likelihood
%     .aic        = aic
%     .bic        = bic
%     .ssr(j)     = Sum-of-squares residuals for each equation. Only
%                  if test = 1
%     .Rsqr(j)    = R^2 for each equation. Only if test = 1
%     .SEeq(j)    = Standard error of regression  for each equation.
%                  Only if test = 1
%     .Fstat(j)   = F-statistic for each equation. Only if test = 1
%     .llkhdeq(j) = log-likelihood for each equation. Only if test = 1
%     .aiceq(j)   = aic for each equation. Only if test = 1
%     .biceq(j)   = bic for each equation. Only if test = 1
%     .ftest(i,j) = Granger F-tests, only if test = 1
%     .fprob(i,j) = Granger marginal probabilities, only if test = 1
%---------------------------------------------------
```

331 varx_res

```
function resid = varx_res(y,nlag,x)
% PURPOSE: performs vector autogressive with exogenous inputs (VARX)
% estimation and returns only residuals
%---------------------------------------------------
% USAGE: resid = varx_res(y,nlag,x)
% where:    y    = an (nobs x neqs) matrix of y-vectors
%           nlag = the lag length
%           x    = matrix of input variables (nobs x nx)
%                 (NOTE: constant vector automatically included)
%---------------------------------------------------
% RETURNS: a matrix of residuals (nobs x neqs)
%---------------------------------------------------
```

332 vec

```
unction v = vec(x)
% PURPOSE: creates a column vector by stacking columns of x
%----------------------------------------------------------
% USAGE:  v = vec(x)
% where:  x = an input matrix
%---------------------------------------------------------
% RETURNS:
%         v = output vector containing stacked columns of x
%----------------------------------------------------------
% Written by KH (Kurt.Hornik@tuwien.ac.at) on 1995/05/08
% Copyright Dept of Probability Theory and Statistics TU Wien

% Modified by J.P. LeSage
```

333 vech

```
function v = vech(x)
% PURPOSE: creates a column vector by stacking columns of x
%          on and below the diagonal
%----------------------------------------------------------
% USAGE:  v = vech(x)
% where:  x = an input matrix
%---------------------------------------------------------
% RETURNS:
%         v = output vector containing stacked columns of x
```

```
%-----------------------------------------------------------

% Written by Mike Cliff, UNC Finance  mcliff@unc.edu
% CREATED: 12/08/98
```

334 vecparwr

```
function str = vecparwr(str)
% PURPOSE: given a structure of a VARMAX model in echelon form, forms a
% vector with the nonrestricted parameters and restricted parameters
%---------------------------------------------------
% USAGE: str = vecparwr(str)
% where:    str    = a structure containing the structure of the VARMAX
%                    model in echelon form
%---------------------------------------------------
% RETURNS: str = a structure containing the previous structure plus
%                a vector with the the nonrestricted parameters and
%                restricted parameters
%---------------------------------------------------
```

335 vincovma

```
function [A,Sigma,Xi]=vincovma(F,K,stda)
%
%
%        This function computes the elements of the initial state vector
%        for the diffuse Kalman filter in ARIMA models
```

336 xmparm

```
function [xx,pvar,pfix,parm]= xmparm(ninput,x0,xm,xf,pvar,pfix,...
%                                    nr,nlagtf,g,tford,parm,nreg)
%
% this function automatically identifies the input filters in a transfer
% function model using the LTF method. See Liu, L. M., and Hanssens, D.
% M. (1982), "Identification of Multiple{Input Transfer Function
% Models", Communications in Statistics, Theory and Methods, 11,
% 297-314.
%
% Input arguments:
% x0    : array containing the initial model parameters
% xm    : array containing the ARMA variable parameters
% xf    : array containing the ARMA fixed parameters
% pvar  : array containing the indices of ARMA variable parameters
% pfix  : array containing the indices of ARMA fixed parameters
% nr    : the number of ARMA variable parameters
% nlagtf: the number of lags for the polynomial approximations to the
%         rational input filter expansions (see LTF method)
% g     : array containing the estimated weigths of the polynomial
%         approximations to the rational input filter expansions (see
%         LTF method)
% tford : (ninput x 3) array containing for each input variable the
%         delay, the degree of the ma part, and the degree of the ar
%          part
% parm : a structure where
%  s:  seasonality
%  S:  second seasonality
% .p:  AR order
% .ps: order of the AR of order s
% .q:  order of the regular MA
% .qs: order of the MA of order s (1 at most)
% .qS: order of the MA of order S (1 at most)
% .dr: order of regular differencing
% .ds: order of differencing of order s
% .dS: order of differencing of order S
% nreg: number of regression variables
%
% Output arguments:
% xx  : an array containing the transfer function model parameters
% pvar: an array containing the transfer function model variable parameters
% pfix: an array containing the transfer function model fixed parameters
% parm: a structure containing the transfer function model information.
%       In addition to the input fields, it contains
% .pvar:  array containing the indices of variable parameters
% .pfix:  array containing the indices of fixed parameters
% .ninput: number of inputs
% .delay: array with the delays of the input filters
% .ma: array with the ma parameters of the input filters
% .ar: array with the ar parameters of the input filters
```