```
% DEMORUN  Run demonstrations of VAR and VARMA modelling
%
%   This file contains two kinds of demonstration of VAR and VARMA modelling
%   using the log-likelihood functions VAR_LL, VARMA_LLC and VARMA_LLM:
%
%   DEMOV  Demonstration of full matrix VAR(p) and VARMA(p,q) modelling with
%          simulated data, both without and with missing values.
%
%   DEMOD  Modelling of annual mean temperatures 1799-2006 at 4 Icelandic
%          meteorological stations using two constrained models, a diagonal VAR
%          model and a distributed lags VAR model. The data has 34% of the
%          values missing (though for speed a shorter period with less missing
%          data is modelled by default).
%
%   Issuing the command DEMORUN(OPTIMIZER) at the Matlab prompt runs the two
%   DEMOD models and four modellings with DEMOV:
%
%             complete data VAR(2) with n=400, r=3
%             missing value VAR(2) with n=200, r=3
%             complete data VARMA(1,1) with n=400, r=3
%             missing value VARMA(1,1) with n=400, r=2
%
%   Sample output from running "demorun('ucminf')" is shown in the file
%   demorunoutput.pdf which should accompany this file.
%
%   The modelling is done by maximizing the log-likelihood with the BFGS-method
%   as implemented in the function specified by OPTIMIZER, which should be one
%   of:
%        'fminunc' to use the Matlab optimization toolbox (see [1])
%        'ucminf'  to use ucminf of Hans Bruun Nielsen, (see [2])
%
%   Before running, one must ensure that the chosen optimizer is on Matlab's
%   search path. The function ucminf is available in a package called
%   immoptimox, see http://imm.dtu.dk/~hbn/immoptibox.
%
%   [1] Optimization Toolbox 3, User's Guide, The Mathworks, 2007, see
%       http://www.mathworks.com.
%
%   [2] HB Nielsen, UCMINF - An algorithm for unconstrained, nonlinear
%       optimization. Report IMM-REP-2000-19, Department of Mathematical
%       Modelling, Technical University of Denmark, 2000.
%
%   [3] K Jonasson and SE Ferrando 2006. Efficient likelihood evaluation for
%       VARMA processes with missing values. Report VHI-01-2006, Engineering
%       Research Institute, University of Iceland.

function demorun(optimizer)
  demov('var_ll',    'complete', optimizer)
  demov('var_ll',    'miss'    , optimizer)
  demov('varma_llc', 'complete', optimizer)
  demov('varma_llm', 'miss'    , optimizer)
  demod(1, optimizer)
  demod(2, optimizer)
end
```

```
% DEMOV  Demonstration of full-matrix model fitting for VAR or VARMA time series
%
%   DEMOV(LLFUN, CODE, OPTIMIZER) demonstrates how a VAR or VARMA time series
%     model may be constructed using the likelihood evaluation function LLFUN
%     and the BFGS-method as implemented in the function specified by OPTIMIZER
%     (see help text of DEMORUN). CODE should be 'complete' or 'miss' and
%     specifies whether to demonstrate missing values LLFUN should be one of
%     'var_ll', 'varma_llc' or 'varma_llm'.
%
%   STEPS THAT DEMOV INVOLVES:
%     1. Calling TESTCASE to draw random parameter matrices
%     3. Call VARMA_SIM to use parameter matrices to simulate a time series
%     4. If required, call MAKEMISSING to make the series contain missing values
%     6. Call VAR_START to find starting value for iteration
%     5. Call optimizer to determine parameter matrices that maximize likelihood

function demov(llfun, code, optimizer)

  fprintf(['\n\n\nDEMOV FOR ' upper(llfun) ' WITH CODE ''' upper(code) '''\n']);

  % DECODE PARAMETERS WHICH SELECT WHICH DEMO IS RUN AND GET PROBLEM DIMENSIONS
  switch code
    case 'complete', MISS = false;
    case 'miss',     MISS = true;
    otherwise        error('Unknown code')
  end
  switch llfun
    case 'var_ll',    p = 2; q = 0; r = 3; if MISS, n = 200; else n = 400; end
    case 'varma_llc', p = 1; q = 1; r = 2; n = 200; assert(~MISS)
    case 'varma_llm', p = 1; q = 1; r = 2; n = 200; assert(MISS)
    otherwise         error('Unknown llfun')
  end
  nA = r*r*p; % Count of A parameters
  nB = r*r*q; % Count of B parameters
  nmu = r;
  nSig = r*(r+1)/2;
  nPar = nA + nB + nSig + nmu;

  % OBTAIN PARAMETER MATRICES TO CREATE SIMULATED TIME SERIES
  randn('state',4); rand('state',5);
  [Ag, Bg, Sigg] = testcase(p,q,r); % "g" for "generating"
  mug = 0.1*(1:r)';

  % CONSTRUCT SIMULATED SERIES AND MAKE SOME VALUES MISSING
  X = varma_sim(Ag,Bg,Sigg,n,mug);
  if MISS
    X = makemissing(X, 'miss-5a');  % SEE [3] FOR EXPLANATION OF miss-xx
    miss = isnan(X);
  else
    mu = mean(X')';
    X = X - repmat(mu, 1, n);
    miss = false(r,n);
  end
  nMiss = sum(miss(:));
  nObs  = n*r - nMiss;
```

```
% DISPLAY SUMMARY
fprintf('\n%s\n', 'Dimensions and parameter counts:')
fprintf('r      = %4d        nA     = %3d\n', r, nA)
fprintf('n      = %4d        nB     = %3d\n', n, nB)
fprintf('p      = %4d        nSig   = %3d\n', p, nSig)
fprintf('q      = %4d        nmu    = %3d\n', q, nmu)
fprintf('nObs  = %4d        nTotal = %3d\n', nObs, nPar)
fprintf('nMiss = %4d                    \n', nMiss)

% FIND STARTING PARAMETERS
[A0, Sig0, mu0] = var_start(X, p);
B0 = zeros(r, r*q);
if ~MISS, mu0 = []; end

% EVALUATE LIKELIHOOD FOR GENERATING AND STARTING PARAMETERS
if MISS
  llgen   = varma_llm(X, Ag, Bg, Sigg, mug, miss);
  llstart = varma_llm(X, A0, B0, Sig0, mu0, miss);
else
  llgen   = varma_llc(X, Ag, Bg, Sigg);
  llstart = varma_llc(X, A0, B0, Sig0);
end
printmat(14,'Generating:', Ag, Bg, Sigg, mug);
printmat(14,'Starting:',   A0, B0, Sig0, mu0);
fprintf('\nLog-likelihood for generating parameters = %.2f\n', llgen);
fprintf('Log-likelihood at starting parameters    = %.2f\n\n', llstart);

% CARRY OUT THE OPTIMIZATION
theta0 = parmat2theta(A0, B0, Sig0, mu0);
opt = optim(optimizer);
fprintf(['Maximizing log-likelihood with "' optimizer '"... '])
fnValueCountPrint('init')

switch optimizer

  case 'fminunc'
    [theta,fval,flg,outp,g] = fminunc(@(th)loglik(th,X,p,q), theta0, opt);
    if flg<1 || flg > 2
      error(['Fminunc failure, exitflag = ' num2str(flg)])
    end
    normg = norm(g,inf);
    niter = outp.iterations;

  case 'ucminf'
    [theta, info] = ucminf(@loglik, theta0, opt, [], X, p, q);
    if info(6) < 1 || info(6) > 2
      error(['Ucminf failure, info(6) = ' num2str(info(6))]);
    end
    [fval, normg, niter] = deal(info(1), info(2), info(5));
  otherwise error('Illegal value of optimizer');
end
fmt1 = 'succeess\nnorm(g,inf)=%.1g, nit=%d, nf=%d\n';
fprintf(fmt1, normg, fnValueCountPrint('nfun'), niter);
[Ah, Bh, Sigh, muh] = theta2parmat(theta, p, q, r);
% "h" for "hat", used to label parameter values that maximize the likelihood
% CALCULATE AND DISPLAY OPTIMAL LIKELIHOOD
  llmax = varma_llm(X, Ah, Bh, Sigh, muh, miss);
  fprintf('\nMaximum log-likelihood = %.2f\n', llmax);
  if ~MISS, muh = mu; end
  printmat(14, 'Best fit', Ah, Bh, Sigh, muh);

end

% DEMOD  Demonstration of diagonal and distributed lags VAR modelling
%
%   DEMOJ(K, OPTIMIZER) for K=1 or K=2, fits one of two models that use the
%   Jacobian feature of VAR_LL. Both models use real-life data, namely yearly
%   mean tempartures at 4 Icelandic meteorological stations form 1799 to 2006.
%   There are a total of 280 values or 34% of the data missing. OPTIMIZER should
%   be 'fminunc' or 'ucminf' (see help text of DEMORUN). The two models are:
%
%   K=1: DIAGONAL VAR MODEL
%     The model is x(t) = A*x(t-1) + D1*x(t-2) + D2*x(t-3) + eps(t) where x(t)
%     and eps(t) are 3-dimensional vectors, eps(t) is N(mu,Sig) i.e. normally
%     distributed with mean mu and covariance Sig, A is a lower triangluar
%     matrix and D1 and D2 are diagonal:
%
%     x(t) = a11  0   0 *x(t-1) + d11 0  0 *x(t-2) + d21 0  0 *x(t-3) + eps(t)
%            a21 a22  0             0 d12 0            0 d22 0
%            a31 a32 a33            0  0 d13           0  0 d23
%
%     This model has a total of 21 parameters (including 6 in Sig and 3 in mu).
%
%   K=2: DISTRIBUTED LAGS VAR MODEL
%     The model is 4-dimensional x(t) = A*(x(t-1) + 0.5*x(t-2)) + eps(t)
%     (distributed lags). A is a general 4x4 matrix, and eps(t) is N(mu,Sig).
%     Thus there are a total of 16 + 10 + 4 = 30 parameters.

function demod(demoid, optimizer)

  % LOAD TERMPERATURE DATA
  f = fopen('temperature.dat');
  X = textscan(f, '%f %f %f %f %f', 'headerlines', 13);
  year = X{1};
  X = cell2mat(X(2:end));   % omit year column
  X = X(year > 1860, 2:4)'; % select subset of X to make demo faster
```

```
switch demoid

  case 1
    fprintf('\n\n\nLOWER AND DIAGONAL MODELLING OF METEOROLOGICAL DATA\n')

    % SELECT X-ROWS AND DEFINE JACOBIAN MATRIX
    J1 = eye(27);
    J = J1(:, [1:3,5:6,9,10,14,18,19,23,27]);

    % DETERMINE STARTING VALUES
    miss = isnan(X);
    [AD0, Sig0, mu0] = var_start(X, 3);
    AD0 = mat2cell(AD0, 3, [3,3,3]);
    A0 = tril(AD0{1});
    D10 = diag(diag(AD0{2}));
    D20 = diag(diag(AD0{3}));
    printmat(11,'Starting:', A0, D10, D20);
    printmat(11,''         , Sig0, mu0);
    llstart = var_ll(X, [A0, D10, D20], Sig0, mu0, miss);
    fprintf('\nLog-likelihood at starting parameters = %.2f\n\n', llstart);
    th0 = [vech(A0); diag(D10); diag(D20); vech(Sig0); mu0];

    % DEFINE LOG-LIKELIHOOD FUNCTION
    llfun = @(th) loglik1(th, X, J);

  case 2
    fprintf('\n\n\nDISTRIBUTED LAGS MODELLING OF METEOROLOGICAL DATA\n')

    % DEFINE JACOBIAN MATRIX
    r = size(X,1);
    I = eye(r^2);
    J = [I; I/2];

    % DETERMINE STARTING VALUES
    miss = isnan(X);
    [AD0, Sig0, mu0] = var_start(X, 2);
    A0 = (AD0(:,1:r) + AD0(:,r+1:2*r))*2/3;
    printmat(11,'Starting:', A0, Sig0, mu0);
    llstart = var_ll(X, [A0, A0/2], Sig0, mu0, miss);
    fprintf('\nLog-likelihood at starting parameters = %.2f\n\n', llstart);
    th0 = [vec(A0); vech(Sig0); mu0];

    % DEFINE LOG-LIKELIHOOD FUNCTION
    llfun = @(th) loglik2(th, X, J);
end

% CARRY OUT THE OPTIMIZATION
opt = optim(optimizer);
fprintf(['Maximizing log-likelihood with "' optimizer '"...\n'])
fnValueCountPrint('init')

switch optimizer

  case 'fminunc'
    [theta, fval, flg, outp, g] = fminunc(llfun, th0, opt);
    if flg<1 || flg > 2
      error(['Fminunc failure, exitflag = ' num2str(flg)])
    end
    normg = norm(g,inf);
    niter = outp.iterations;

  case 'ucminf'
    [theta, info] = ucminf(llfun, th0, opt, []);
    if info(6) < 1 || info(6) > 2
      error(['Ucminf failure, info(6) = ' num2str(info(6))]);
    end
    [fval, normg, niter] = deal(info(1), info(2), info(5));
end

% PRINT OUT RESULT
fmt1 = 'success\nnorm(g,inf)=%.1g, nit=%d, nf=%d\n';
fprintf(fmt1, normg, fnValueCountPrint('nfun'), niter);
switch demoid
  case 1
    [Ah, D1h, D2h, Sigh, muh] = theta2parmat1(theta);
    printmat(11, 'Max.loglik:', Ah, D1h, D2h);
    printmat(11, ''           , Sigh, muh);
  case 2
    [Ah, Sigh, muh] = theta2parmat2(theta, r);
    printmat(11, 'Max.loglik:', Ah, Sigh, muh);
end
fprintf('\nLog-likelihood at solution = %.2f\n\n', -fval);
end

function [f, g] = loglik(theta, X, p, q)  % LOG LIKELIHOOD FOR DEMOV
  % Evaluate -loglikelihood function and optionally its gradient choosing
  % var_ll, varma_llc or varma_llm using an appropriate call, depending on the
  % model being fitted, whether there are missing values and whether g is a
  % return variable.
  miss = isnan(X);
  MISS = any(miss(:));
  r = size(X,1);
  [A, B, Sig, mu] = theta2parmat(theta, p, q, r);
  if isempty(B)
    if ~MISS, [f, ok, g] = var_ll(X, A, Sig);
    else      [f, ok, g] = var_ll(X, A, Sig, mu, miss); end
  else
    if ~MISS, [f, ok, g] = varma_llc(X, A, B, Sig);
    else      [f, ok, g] = varma_llm(X, A, B, Sig, mu, miss); end
  end
  if ok, f = -f; else f = 1e20; end
  if ok, g = -g; else g = 1e20*ones(size(theta)); end
  fnValueCountPrint('fcall', f, g);
end
```

```
function [f, g] = loglik1(x, X, J)  % LOG LIKELIHOOD FOR DEMOD 1
  [A, D1, D2, Sig, mu] = theta2parmat1(x);
  miss = isnan(X);
  [f, ok, g] = var_ll(X, [A D1 D2], Sig, mu, miss, J);
  if ok, f = -f; else f = 1e20; end
  if ok, g = -g; else g = 1e20*ones(size(x)); end
  fnValueCountPrint('fcall', f, g);
end

function [f, g] = loglik2(x, X, J)  % LOG LIKELIHOOD FOR DEMOD 2
  r = size(X,1);
  [A, Sig, mu] = theta2parmat2(x, r);
  miss = isnan(X);
  [f, ok, g] = var_ll(X, [A A/2], Sig, mu, miss, J);
  if ok
    f = -f;
    g = -g;
  else
    f = 1e20;
    g = 1e20*ones(size(x));
  end
  fnValueCountPrint('fcall', f, g);
end

function [A, B, Sig, mu] = theta2parmat(theta, p, q, r)
  % EXTRACT PARAMETER MATRICES FROM A COMBINED PARAMETER VECTOR FOR DEMOV
  nA = r*r*p;
  nB = r*r*q;
  nSig = r*(r+1)/2;
  nmu = length(theta) - nA - nB - nSig;
  nPar = nA + nB + nSig + nmu;
  A =  reshape(theta(1       : nA),    r, r*p);
  B =  reshape(theta(nA+1    : nA+nB), r, r*q);
  Sig = makeSig(theta(nA+nB+1 : nA+nB+nSig));
  mu =  theta(end-nmu+1 : end);
end

function [A, D1, D2, Sig, mu] = theta2parmat1(x)
  % EXTRACT PARAMETER MATRICES FROM A COMBINED PAR VECTOR FOR DEMOD 1
  r = 3;
  A = [x(1:3) [0;x(4:5)] [0;0;x(6)]];
  D1 = diag(x(7:9));
  D2 = diag(x(10:12));
  Sig = makeSig(x(13:18));
  mu = x(19:21);
end

function [A, Sig, mu] = theta2parmat2(x, r)
  % EXTRACT PARAMETER MATRICES FROM A COMBINED PAR VECTOR FOR DEMOD 2
  rr = r*r;
  rs = r*(r+1)/2;
  A = reshape(x(1:rr), r, r);
  Sig = makeSig(x(rr+1:rr+rs));
  mu = x(end-r+1:end);
end

function theta = parmat2theta(A, B, Sig, mu)
  % COMBINE ENTRIES OF ALL PARAMETER MATRICES IN ONE COLUMN VECTOR FOR DEMOV
  theta = [vec(A); vec(B); vech(Sig); mu];
end

function n = fnValueCountPrint(operation, f, g)
  persistent nfun
  switch operation
    case 'init'
      nfun = 0;
      fprintf('\nnFun  -logLik  norm(g,inf)\n');
    case 'fcall'
      nfun = nfun + 1;
      if f>1e19, f=inf; g=inf; end
      fprintf('%3d %10.5f %10.4f\n', nfun, f, norm(g,inf));
    case 'nfun'
      n = nfun;
  end
end

function printmat(ntxt, txt, varargin)
  % PRINTMAT(N, TXT, A, B,...) prints A, B,... (which must all have same number
  % of rows) with format %6.3f and preceeded with txt and the variable names.
  % TXT is printed in a field if width ntxt.
  nlin = size(varargin{1}, 1);
  fprintf('\n');
  for i=1:nlin
    for j=1:length(varargin)
      varn = ['  ' inputname(j+2) ' ='];
      if j==1, varn = [sprintf('%-*s', ntxt, txt) varn]; end
      if i>1, varn(1:end) = ' '; end
      M = varargin{j};
      [m, n] = size(M);
      if isempty(M), continue, end
      fprintf(varn);
      for k = 1:n
        fprintf(' %6.3f', M(i,k));
      end
    end
    fprintf('\n');
  end
end
```

```
function [A, Sig, mu] = var_start(X, p, q)
  % FIND STARTING VALUES FOR VAR LIKELIHOOD MAXIMIZATION.
  %
  % [A,B,Sig,mu] = VAR_START(X, p) determines A = {A1 A2...Ap}, Sig and mu that
  % can be used as starting values for numerical maximization of a VAR
  % likelihood function. The rxn array X contains the observed time series with
  % NaN in missing value positions; p is the number of autoregressive terms.
  %
  % METHOD: The Ai-s are chosen to minimize the residual sum of squares (or
  % equivalently maximize the conditional likelihood). Sig is chosen as the data
  % covariance matrix of the residuals. If there are missing values, these are
  % first filled in with the average of the corresponding series (row in X).
  [r,n] = size(X);
  mu = nanmean(X,2);
  miss = isnan(X);
  for i = 1:r, X(i,isnan(X(i,:))) = mu(i); end  % fill in missing values
  X = X - repmat(mu, 1, n);
  [r, n] = size(X);
  x = X(:);
  N = p*r^2;
  xd = zeros(r*n, 1, N);
  A = zeros(r,r*p);
  [w, wd] = lambda_multiply(A, x, false(r, n), xd);
  F = zeros(r,r,p,r,r,p);
  b = zeros(r,r,p);
  G = zeros(r,r,p+1);
  for d=0:p
    G(:,:,d+1) = X(:,p+1-d:n-p)*X(:,p+1:n-p+d)';
  end
  for i=0:p
    for j=i:p
      d = j-i; ne = n-p+1;
      V = G(:,:,d+1) + X(:,p+1-j:p-d)*X(:,p+1-i:p)'+X(:,ne:n-j)*X(:,ne+d:n-i)';
      for l=1:r
        if i>0,  F(l,:,j,l,:,i) = V; end
        if i==0 && j>0, b(l,:,j) = V(:,l); end
      end
    end
  end
  F = reshape(F,N,N); b = reshape(b,N,1);
  SymPosDef = struct('SYM',true,'POSDEF',true);
  ok = false; del = 1e-10;
  while ~ok
    try
      a = linsolve(F', b, SymPosDef);
      ok = true;
    catch
      F = F + del*eye(N);
      del = del*10;
    end
  end
  A = reshape(a,r,r*p);
  w = lambda_multiply(A, X(:), false(r, n));
  W = reshape(w(r*p+1:end), r, n-p);
  Sig = cov(W(:,p+1:end)');
end

function opt = optim(optimizer)
  switch optimizer
    case 'fminunc'
      opt = optimset( ...
        'LargeScale', 'off',  ...
        'GradObj',    'on',   ...
        'TolX',       5e-7,   ...
        'Display',    'off', ...
        'TypicalX',   1e-3,   ...
        'TolFun',     1e-6);
    case 'ucminf'
      opt = [1e-3 ... % length of initial step
        ,    1e-4 ... % tolerance for norm(g,inf)
        ,    5e-7 ... % tolerance for dx
        ,    10000];  % max iteration count
    otherwise error('Unknown optimizer')
  end
end

function Sig = makeSig(s)
  % INVERSE OF VECH
  k = 1; r = floor(sqrt(length(s)*2));
  Sig = zeros(r,r);
  for i=1:r
    Sig(i:r,i) = s(k:k+r-i);
    k = k+r-i+1;
  end
  Sig = Sig + tril(Sig,-1)';
end

function v = vec(A)
  % CHANGE MATRIX TO COLUMN VECTOR
  v = A(:);
end

function v = vech(A)
  % CHANGE LOWER TRIANGLE TO COLUMN VECTOR
  if isempty(A), v=[];
  else
    [n,m,N] = size(A);
    assert(n==m && N==1);
    v = zeros(n*(n+1)/2, 1);
    m = 1;
    for i=1:n
      m1 = m + n-i;
      v(m:m1) = A(i:n, i);
      m = m1 + 1;
    end
  end
end
```