Contents

CODE

IC.m

```
function [AIC, HQIC, BIC] = IC(var, T, K)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'IC' computes various information criteria

% Inputs:
% — llf, log marginal likelihood
% — T, sample size
% — K, number of regressors

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S   = var.Sigma_ols; iS  = inv(S);
```

```
E   = var.e_ols;
N   = size(S,1);
llf = - (T * N / 2) * (1 + log(2 * pi)) - T / 2 * log(det(S));
llf = llf - 1 /2 * trace( iS * E' * E);

AIC = - 2 * llf / T + 2 * K / T;
% SIC = - 2 * llf / T + K * log(T) / T;
HQIC = - 2 * llf / T + 2 * K * log(log(T)) / T;
BIC = - 2 * llf / T + K * log(T) / T;
```

OrderingIndexes.m

```
function [sindex] = OrderingIndexes(varordering,varnames,newstrng)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'OrderingIndexes' find the specified order of variables

% input:
%- varnames    = the list (names) of all the variables in the database
%- varordering = the names and the order of the variables in the VAR

% output:
%- sindex = the index in the varnames that correspond to the var ordering
%- index_XX = the index that the variable XXX has in the VAR

% Filippo Ferroni,
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin<3
    newstrng='';
end

[~,sindex] = ismember(varordering,varnames);
for ii = 1 : size(varordering,2)
    check=0;
    for jjj = 1: size(varnames,2)
         if strcmp(deblank(varnames{jjj}),deblank(varordering{ii})) == 1,
             check = 1;
         end
    end
    if check == 1
        assignin('base', [newstrng 'index_' varordering{ii} ], ii);
    else
        warning(['I did not find ' varordering{ii} ' in varnames'])
```

```
    end
end
```

YXB_.m

```
function [YYact,XXact] = YXB_(YY,lags,constant_timetrend)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'YXB_' organizes the data in the form
% of Y = XB+E
% NO dummy observations

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin<3
    constant = 1;
    timetrend = 0;
else
    constant  = constant_timetrend(1);
    timetrend = constant_timetrend(2);
end

nlags_   = lags;                  % number of lags   */
T0       = lags;                  % size of pre-sample */

nv      = size(YY,2);             %* number of variables */
nobs    = size(YY,1)-T0;          %* number of observations */

% Actual observations

YYact = YY(T0+1:T0+nobs,:);
XXact = zeros(nobs,nv*nlags_);
i = 1;

while (i <= nlags_)
    XXact(:,(i-1)*nv+1:i*nv) = YY(T0-(i-1):T0+nobs-i,:);
    i = i+1;
end

if constant
    % last column of XXact = constant
    XXact = [XXact ones(nobs,1)];
end
```

```
if timetrend
    % last column of XXact = constant
    XXact = [XXact (1:nobs)'];
end
```

blp_ml.m

```
% function [log_dnsty] = blp_ml(shrinkage,hh,prior,olsreg_,F,G,Fo,
    positions_nylags,position_constant)
function [log_dnsty] = blp_ml(shrinkage,hh,prior,olsreg_,F,G,Fo,
    positions_nylags,position_constant)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'blp_ml' computes the marginal likelihood for the NMIW LP

% Inputs:
% — hyperpara, shrinkage hyperpara over which maximize the marginal
% likelihood

% Output: marginal data density

% Filippo Ferroni, 3/21/2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*****************************************************
% SETTINGS
%*****************************************************
ny   = size(G,2);
lags = size(G,2)/ny;
%*****************************************************
% Conjugate Prior: MN—IW for LP
%*****************************************************
[posterior1,prior1] = p2p(hh,shrinkage,prior,olsreg_,F,G,Fo,positions_nylags,
    position_constant);

%*****************************************************************
%* Compute the log marginal data density for the VAR model with MNIW
%*****************************************************************
var.y = olsreg_.Y;
var.X = olsreg_.X(:,[positions_nylags,position_constant]);
prior1.Sigma.df     = prior1.df;
prior1.Sigma.scale  = prior1.S;
prior1.Phi.mean     = prior1.BetaMean;
prior1.Phi.cov      = prior1.BetaVar;
```

```
35
36 log_dnsty      = mniw_log_dnsty(prior1,posterior1,var);
```

blp_opt_hyperpara.m

```
1  function minus_log_dnsty = blp_opt_hyperpara(hyperpara,hh,prior,olsreg_,F,G,Fo
       ,positions_nylags,position_constant)
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  % 'blp_opt_hyperpara' computes the marginal likelihood over the
5  % hyperparameters of for the LP
6
7  % Inputs:
8  % - hyperpara, prior shrinkage
9  % likelihood
10
11 % Output: marginal data density
12
13 % Filippo Ferroni, 3/21/2020
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15
16 hyperpara       = exp(hyperpara);
17 log_dnsty       = blp_ml(hyperpara,hh,prior,olsreg_,F,G,Fo,positions_nylags,
       position_constant);
18 minus_log_dnsty = -log_dnsty;
```

bvar.m

```
1  function [BVAR] = bvar(y,lags,options)
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  % 'bvar' generates draws from the paramters of VAR model with error
6  % distributed as a multivariate normal
7
8  % Core Inputs:
9  % - y, data columns variables
10 % - lags, lag order of the VAR
11
12 % Additonal Inputs collected options:
13 % - options are not mandatory. if nothing is specified then a Jeffrey prior is
14 %   assumed
15 % - options.K, number of draws from the posterior distribution
```

```
% — options.hor, horizon to compute the impulse response
% — options.fhor, horizon to compute the out—of—sample forecast
% — options.priors, a string with he priors for the autoregressive paramters and
%   for the scaling matrix.
% (...) see below

% Output: Draws from the conditional distribution of Phi, Sigma and
% Omega, impulse response with the cholesky decompotion and long run
% restricions, forecast and marginal likelihood.

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
% Revised, 27/02/2020
% Revised, 27/04/2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2
    error('the BVAR toolbox needs at least two inputs: data and number of lags');
end
if lags < 1
    error('lags cannot be zero or negative');
end
% number of observable variables
ny                  = size(y, 2);

%****************************************************
%* DEFAULT SETTINGS
%****************************************************
% Control random number generator
if isOctave == 0
    isMatlab = 1;
    rng('default');
    rng(999);
else
    isMatlab = 0;
    % pkg load optim
    randn('state',999);
    rand('state',999);
end


% Default Settings (they can all be changed in 'options' see below)
```

```
61 K                    = 5000;          % number of draws from the posterior
62 hor                  = 24;            % horizon for the IRF
63 fhor                 = 12;            % horizon for the forecasts
64 firstobs             = lags+1;        % first observation
65 presample            = 0;             % using a presample for setting the hyper-
       parameter of the Minnesota prior
66 noconstant           = 0;             % when 0, includes a constatn in the VAR
67 timetrend            = 0;             % when 1, includes a time trend in the VAR
68 minn_prior_tau       = 3;             % Minnesota prior Hyper-Param: Overall
       Tightness
69 minn_prior_decay     = 0.5;           % Minnesota prior Hyper-Param: Tighness on
        lags>1
70 minn_prior_lambda    = 5;             % Minnesota prior Hyper-Param: Sum-of-
       Coefficient
71 minn_prior_mu        = 2;             % Minnesota prior Hyper-Param: Co-
       Persistence
72 minn_prior_omega     = 2;             % Minnesota prior Hyper-Param: Shocks
       Variance
73 long_run_irf         = 0;             % when 0, it does not compute long run IRF
74 irf_1STD              = 1;             % when 1, IRF are computed as 1SD increase
       . Else, IRF are compued as unitary increase in the shock
75 cfrcst_yes           = 0;             % no conditional forecast unless defined
       in options
76
77 signs_irf            = 0;
78 narrative_signs_irf  = 0;
79 zeros_signs_irf      = 0;
80 proxy_irf            = 0;
81 noprint              = 0;
82 nexogenous           = 0;
83 exogenous            = [];
84
85 % for mixed frequecy / irregurerly sampled data.
86 % Interpolate the missing values of each times series.
87 if any(any(isnan(y)))==1
88     warning('Activating the Mixed Frequency BVAR')
89
90     mixed_freq_on = 1;
91     index_nan_var = find(sum(isnan(y),1) ~= 0);
92     yoriginal     = y;
93     % interporlate the variables that have nans
94     T = 1:1:length(y);
95     for kk  = 1 : length(index_nan_var)
96         v                       = y(isfinite(y(:,index_nan_var(kk))),
               index_nan_var(kk));
97         x                       = find(isfinite(y(:,index_nan_var(kk))));
98         if isMatlab == 1
```

```
99              y(:,index_nan_var(kk))  = interp1(x,v,T','spline');
100         else
101             y(:,index_nan_var(kk))  = spline(x,v,T');
102         end
103         yinterpol               = y;
104     end
105 else
106     mixed_freq_on = 0;
107 end
108 if mixed_freq_on == 1 && nargin < 3
109     warning(['You did not specified the aggregation of the mixed freq. '...
110         'I will treat them as stocks.']);
111     index = zeros(size(y,2),1);
112 end
113
114 % Priors declaration: default Jeffrey prior
115 dummy    = 0;
116 flat     = 1;
117 priors   = priors_( );
118
119 % declaring the names for the observable variables
120 for v = 1 : ny
121     eval(['varnames{'   num2str(v) '} = ''Var' num2str(v) ''';'])
122 end
123
124
125 %*********************************************************
126 %* CUSTOMIZED SETTINGS
127 %*********************************************************
128 if nargin > 2
129     if isfield(options,'vnames')==1
130         varnames = options.vnames;
131     end
132     %=========================================================================
133     % Inference options
134     %=========================================================================
135     if isfield(options,'K')==1
136         K = options.K;
137     end
138     if isfield(options,'firstobs')==1
139         firstobs = options.firstobs;
140         if firstobs < lags + 1
141             error('firstobs need to be larger than lags +1')
142         end
143     end
144     if isfield(options,'presample')==1
145         presample = options.presample;
```

```
146     end
147     if isfield(options,'noconstant')==1
148         noconstant = options.nocostant;
149     end
150     if isfield(options,'timetrend')==1
151         timetrend  = options.timetrend;
152         noconstant = 0;
153     end
154     %=========================================================================
155     % Exogenous Variables options
156     %=========================================================================
157     if isfield(options,'exogenous')==1
158         exogenous = options.exogenous;
159         nexogenous = size(exogenous,2);
160         if size(exogenous,1) ~= size(y,1)+ fhor && size(exogenous,1) ~= size(y
                ,1)
161             error('Size Mismatch between endogenous and exogenos variables;
                    exo must be either T or T+fhor');
162         end
163     end
164     %=========================================================================
165     % Minnesota prior options
166     %=========================================================================
167     if (isfield(options,'priors')==1 && strcmp(options.priors.name,'Minnesota'
            )==1) || (isfield(options,'priors')==1 && strcmp(options.priors.name,'
            minnesota')==1) || ...
168        (isfield(options,'prior')==1 && strcmp(options.prior.name,'Minnesota')
               ==1) || (isfield(options,'prior')==1 && strcmp(options.prior.name,'
               minnesota')==1)
169         % MINNESOTA PRIOR
170         dummy = 1;
171         flat  = 0;
172         timetrend = 0;
173         priors.name= 'Minnesota';
174     end
175     if isfield(options,'minn_prior_tau')==1 || isfield(options,'bvar_prior_tau
            ')==1
176         dummy = 1;
177         flat  = 0;
178         priors.name= 'Minnesota';
179         % MINNESOTA PRIOR: tightness
180         if isfield(options,'bvar_prior_mu')==1
181             minn_prior_tau = options.bvar_prior_tau;
182         else
183             minn_prior_tau = options.minn_prior_tau;
184         end
185     end
```

```
186     if isfield(options,'minn_prior_decay')==1 || isfield(options,'
            bvar_prior_decay')==1
187         dummy = 1;
188         flat  = 0;
189         priors.name= 'Minnesota';
190         % MINNESOTA PRIOR: decay
191         if isfield(options,'bvar_prior_mu')==1
192             minn_prior_decay = options.bvar_prior_decay;
193         else
194             minn_prior_decay = options.minn_prior_decay;
195         end
196     end
197     if isfield(options,'minn_prior_lambda')==1 || isfield(options,'
            bvar_prior_lambda')==1
198         dummy = 1;
199         flat  = 0;
200         priors.name= 'Minnesota';
201         % MINNESOTA PRIOR: sum-of-coeff
202         if isfield(options,'bvar_prior_mu')==1
203             minn_prior_lambda = options.bvar_prior_lambda;
204         else
205             minn_prior_lambda = options.minn_prior_lambda;
206         end
207     end
208     if isfield(options,'minn_prior_mu')==1 || isfield(options,'bvar_prior_mu')
            ==1
209         dummy = 1;
210         flat  = 0;
211         priors.name= 'Minnesota';
212         % MINNESOTA PRIOR: co-persistence
213         if isfield(options,'bvar_prior_mu')==1
214             minn_prior_mu = options.bvar_prior_mu;
215         else
216             minn_prior_mu = options.minn_prior_mu;
217         end
218     end
219     if isfield(options,'minn_prior_omega')==1 || isfield(options,'
            bvar_prior_omega')==1
220         dummy = 1;
221         flat  = 0;
222         priors.name= 'Minnesota';
223         % MINNESOTA PRIOR: variance
224         if isfield(options,'bvar_prior_omega')==1
225             minn_prior_omega = options.bvar_prior_omega;
226         else
227             minn_prior_omega = options.minn_prior_omega;
228         end
```

```
229     end
230     if isfield(options,'max_minn_hyper')==1 && options.max_minn_hyper ==1 &&
            mixed_freq_on ==0
231         % maximize the hyper parameters of the minnesota prior
232         hyperpara(1) = minn_prior_tau;
233         hyperpara(2) = minn_prior_decay;
234         hyperpara(3) = minn_prior_lambda;
235         hyperpara(4) = minn_prior_mu;
236         hyperpara(5) = minn_prior_omega;
237
238         dummy = 1;
239         flat  = 0;
240         priors.name= 'Minnesota';
241         try
242             [postmode,lm,~] = bvar_max_hyper(hyperpara,y,lags,options);
243
244             minn_prior_tau      = postmode(1);
245             minn_prior_decay    = postmode(2);
246             minn_prior_lambda   = postmode(3);
247             minn_prior_mu       = postmode(4);
248             minn_prior_omega    = postmode(5);
249             disp('                          ')
250             disp('Maximization Successful: I will use the the mode values.')
251
252         catch
253             warning('Maximization NOT Successful')
254             disp('Using hyper parameter default values')
255         end
256     end
257     %=========================================================================
258     % Conjugate/Hierachical prior options
259     %=========================================================================
260     if (isfield(options,'priors')==1 && strcmp(options.priors.name,'Conjugate'
            )==1) || (isfield(options,'priors')==1 && strcmp(options.priors.name,'
            conjugate')==1) || ...
261            (isfield(options,'prior')==1 && strcmp(options.prior.name,'Conjugate')
                ==1) || (isfield(options,'prior')==1 && strcmp(options.prior.name,'
                conjugate')==1)
262
263         if isfield(options,'prior')==1
264             options.priors = options.prior;
265         end
266         if dummy == 1
267             warning('You have set both the Conjugate and Minnesota (perhaps
                    via options.minn_prior_XX)');
268             warning('I will consider the Conjugate prior only');
269         end
```

```
270         dummy = 2;
271         %  warning('The Conjugate prior is still under construction ... ');
272         flat  = 0;
273         priors.name= 'Conjugate';
274         % Priors for the AR parameters
275         if isfield(options.priors,'Phi') == 1
276             % mean
277             if isfield(options.priors.Phi,'mean') == 1
278                 prior.Phi.mean  = options.priors.Phi.mean;
279                 if max(size(prior.Phi.mean) ~= [ny*lags+(1-noconstant)+
                        timetrend+nexogenous   ny]) ~= 0
280                     error('Size mismatch')
281                 end
282             else
283                 warning(['You did not provide a prior mean for the AR coeff. '
                        ...
284                     'Assume zeros everywhere.'])
285                 prior.Phi.mean  = zeros(ny*lags+(1-noconstant)+timetrend+
                        nexogenous , ny);
286             end
287             % variance
288             if isfield(options.priors.Phi,'cov') == 1
289                 prior.Phi.cov   = options.priors.Phi.cov;
290 %                 if length(prior.Phi.cov) ~= (ny*lags+(1-noconstant) +
    timetrend ) * ny
291                 if length(prior.Phi.cov) ~= (ny*lags+(1-noconstant) +timetrend
                        +nexogenous ) || size(prior.Phi.cov,1 )~=size(prior.Phi.
                        cov,2)
292                     error('Size mismatch: Covariance Phi should be square, e.g
                            . size(Phi.mean,1)x size(Phi.mean,1)x')
293                 end
294             else
295                 warning(['You did not provide a Covariance for the AR coeff. '
                        ...
296                     'Assume 10 times Identity Matrix.'])
297 %                 prior.Phi.cov  = 10 * eye((ny*lags+(1-noconstant) +
    timetrend) * ny);
298                 prior.Phi.cov  = 10 * eye((ny*lags+(1-noconstant) + timetrend+
                        nexogenous));
299             end
300         else
301             warning(['You did not provide prior mean and covariance for the AR
                    coeff ' ...
302                 'Assume zeros everywhere with covariance 10 times Identity
                        Matrix.'])
303 %             prior.Phi.cov   = 10 * eye((ny*lags+(1-noconstant) +timetrend )
    * ny);
```

```
304             prior.Phi.cov    = 10 * eye((ny*lags+(1-noconstant) + timetrend+
                    nexogenous ));
305             prior.Phi.mean   = zeros(ny*lags+(1-noconstant) + timetrend+
                    nexogenous ,  ny);
306         end
307         % Priors for the Residual Covariance
308         if isfield(options.priors,'Sigma') == 1
309             % scale
310             if isfield(options.priors.Sigma,'scale') == 1
311                 prior.Sigma.scale = options.priors.Sigma.scale;
312                 if size(prior.Sigma.scale) ~= [ny ny]
313                     error('Size mismatch')
314                 end
315             else
316                 warning(['You did not provide a prior mean for the Residual
                        Covariance. ' ...
317                     'Assume identity matrix.'])
318                 prior.Sigma.scale = eye(ny);
319             end
320             % degrees of freedom
321             if isfield(options.priors.Sigma,'df') == 1
322                 prior.Sigma.df = options.priors.Sigma.df;
323                 if length(prior.Sigma.df) ~= 1
324                     error('Size mismatch')
325                 end
326                 if prior.Sigma.df/2 <= ny-1
327                     error('Too few degrees of freedom - Increase prior df')
328                 end
329             else
330                 warning(['You did not provide the degrees of freedom for the
                        Residual Covariance. ' ...
331                     'Assume ny+1.'])
332                 prior.Sigma.df = ny + nexogenous + timetrend + 1;
333                 while prior.Sigma.df/2 <= ny-1 % too few df
334                     prior.Sigma.df =  prior.Sigma.df +1;
335                 end
336             end
337         else
338             warning(['You did not provide prior mean and variance for the
                    Residual Covariance. ' ...
339                 'Assume an identity matrix matrix with N+1 degrees of freedom.
                        '])
340             prior.Sigma.scale = eye(ny);
341             prior.Sigma.df    = ny + nexogenous + timetrend + 1;
342             while prior.Sigma.df/2 <= ny-1 % too few df
343                 prior.Sigma.df =  prior.Sigma.df +1;
344             end
```

```
345         end
346     end
347     %=========================================================================
348     % IRF options
349     %=========================================================================
350     if isfield(options,'hor') ==1
351         hor = options.hor;
352     end
353     if isfield(options,'long_run_irf')==1
354         % Activating Long run IRF
355         long_run_irf = options.long_run_irf;
356     end
357     if isfield(options,'signs')==1
358         % Activating IRF with sign restrictions (mulitple horizons allowed)
359         signs_irf       = 1;
360         signs           = options.signs;
361         if iscellstr(signs) == 0
362             error(['options.signs should be a cell array. Each cell must
                    contain a string with the format'...
363                 '\n''y(a,b,c)<0'' or ''y(a,b,c)>0'' where a, b and c are
                        integers.',...
364                 '\na = index of the variable',...
365                 '\nb = horizon',...
366                 '\nc = index of the shock'],class(zeros))
367         end
368     end
369     if isfield(options,'narrative')==1
370         if signs_irf  == 0
371             warning('You did not provide any sign restrictions.')
372             signs{1} = 'isempty(y(1,1,1))==0';
373         end
374         narrative_signs_irf = 1;
375         narrative           = options.narrative ;
376     end
377     if isfield(options,'zeros_signs')==1
378         if signs_irf  == 1
379             signs_irf       = 0;  % disactivating signs
380         end
381         if narrative_signs_irf  == 1
382             narrative_signs_irf       = 0;  % disactivating narrative
383         end
384         % Activating IRF with zeros and sign restrictions (mulitple horizons
                NOT allowed)
385         zeros_signs_irf  = 1;
386         zeros_signs      = options.zeros_signs;
387         if iscellstr(zeros_signs) == 0
388             error(['options.zeros_signs should be a cell array.'...
```

```
389                 '\nEach cell must contain a string with the following format'
                        ...
390                 '\nFor sign restrictions ''y(a,b)=1'' or ''y(a,b)=-1'',',...
391                 '\nFor short run zero restriction ''ys(a,b)=0'',',...
392                 '\nFor long run restriction ''yr(a,1,b)=0'' where a and b are
                        integers.',...
393                 '\na = index of the variable',...
394                 '\nb = index of the shock'],class(zeros_signs))
395             end
396             [f,sr] = sign2matrix(zeros_signs,ny);
397             if isfield(options,'var_pos')==0
398                 var_pos = ones(1,ny);
399             else
400                 var_pos = options.var_pos;
401             end
402         end
403         if isfield(options,'proxy')==1
404             % Activating IRF with provy
405             proxy_irf   = 1;
406             in.proxies  = options.proxy;
407             in.vars     = y;
408             in.p        = lags;
409             if isfield(options,'proxy_end') == 1
410                 in.T_m_end  = options.proxy_end;
411             else
412                 in.T_m_end  = 0;  %if the times series of the instrument ends when
                        VAR data ends
413             end
414             in.irhor    = hor;
415             if isnumeric(in.proxies) == 0
416                 error(['options.proxy should be a numeric array (nans or inf not
                            allowed)'],class(in.proxies))
417             end
418         end
419         if isfield(options,'irf_1STD')==1
420             % Activating of unitary IRF, i.e. a unitary increase in the shocks
421             % (instead of 1 STD)
422             irf_1STD = options.irf_1STD;
423         end
424         %=========================================================================
425         % (Un)Conditional Forecasts options
426         %=========================================================================
427         if isfield(options,'fhor')==1
428             fhor = options.fhor;
429             if fhor < 1
430                 error('Forecast horizon must be positive')
431             end
```

```
432     end
433     if isfield(options,'endo_index')==1
434         % Forecast conditional on the path of an endogenous var
435         cfrcst_yes      = 1;
436         if isfield(options,'endo_path')== 0
437             error('You need to provide the p[ath for the endogenou variable')
438         end
439         % rows forecasts, column variables
440         endo_path       = options.endo_path;
441         endo_path_index = options.endo_index;
442         if length(endo_path_index) ~= size(endo_path)
443             error(['Mismatch beween the number of endogenous paths and the
                    number of conditioned variables'...
444                 '\nE.g. the #of conditioned variables must coincide with the
                        #of column in ''options.endo_path'''],class(
                        endo_path_index));
445         end
446         if isfield(options,'exo_index')==1
447             % Forecast conditional on the path of an endo var using only a
448             % subset of shocks. notice that the # of endo and # exo must
                    coincide
449             cfrcst_yes  = 2;
450             exo_index   = options.exo_index;
451             Omega       = eye(ny);
452             %               if isfield(options,'Omegaf')==1
453             %                   Omegaf = options.Omegaf;
454             %               end
455             if length(exo_index) ~= length(endo_path_index)
456                 error('the #of conditioned endogenous and #exogenous shocks
                        used must coincide');
457             end
458         end
459
460     end
461     %=========================================================================
462     % Missing Values options
463     %=========================================================================
464     if mixed_freq_on == 1 && isfield(options,'mixed_freq_index')==1
465         index = options.mixed_freq_index;
466         if length(index) ~= size(y,2)
467             error(['You have to specify as many index as observables.'...'
468                 '\nIf no missing values for var j index(j)=0'...
469                 '\nIf missing values for var j, and var j is a stock index(j)
                        =0'...
470                 '\nIf missing values for var j, and var j is a real flow index
                        (j)=2'],class(index))
471         end
```

```
472     elseif mixed_freq_on == 1 && isfield(options,'mf_varindex')== 1
473         index = zeros(ny,1);
474         index(options.mf_varindex) = 2;
475     elseif (mixed_freq_on == 1 && isfield(options,'mixed_freq_index')== 0) ||
            (mixed_freq_on == 1 && isfield(options,'mf_varindex')== 0)
476         warning(['You did not specified the aggregation of the mixed freq.'...
477             '\nI will treat them as stocks.']);
478         index = zeros(size(y,2),1);
479     end
480     if isfield(options,'noprint')==1
481         noprint = options.noprint;
482     end
483 end
484
485 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
486 %% Consistency Checks
487 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
488 nobs = size(y,1)-firstobs+1;
489 if (firstobs+ nobs-1)> size(y,1)
490     fprintf('Incorrect or missing specification of the number of observations.
            nobs can be at most %4u\n',size(y,1)-firstobs+1);
491     error('Inconsistent number of observations.')
492 end
493 if firstobs + presample + lags >= nobs
494     error('presample too large')
495 end
496 if firstobs + presample  <= lags
497     error('firstobs+presample should be > # lags (for initializating the VAR)'
            )
498 end
499 if dummy == 1 && nexogenous > 0
500     warning('I will not use exogenous variables with Minnesota');
501     nexogenous = 0;
502 end
503 if mixed_freq_on == 1 && nexogenous > 0
504     warning('I will not use exogenous variables with missing observations');
505     nexogenous = 0;
506 end
507 if cfrcst_yes ~= 0 && nexogenous > 0
508     warning('I will not use exogenous variables with conditional forecasts');
509     nexogenous = 0;
510 end
511 if size(exogenous,1) == size(y,1) && nexogenous > 0
512     warning('For forecast purposes, I will assume that exo are zero out-of
            sample.')
513     fprintf('To change this, include the exogenous forecasts in options.
            exogenous.\n')
```

```
514     exogenous = [exogenous; zeros(fhor,nexogenous)];
515 end
516
517 %*******************************************************
518 %* Priors and Posterior Distributions
519 %*******************************************************
520
521 idx = firstobs+presample-lags:firstobs+nobs-1;
522 nx  = 1;
523 if noconstant
524     nx = 0;
525 end
526
527 % organize data as  yy = XX B + E
528 [yy,XX] = YXB_(y(idx, :),lags,[nx timetrend]);
529
530 ydum    = [];
531 xdum    = [];
532 pbreaks = 0;
533 lambda  = 0;
534 mu      = 0;
535
536 ydata   = y(idx, :);
537 T       = size(ydata, 1);
538 if T-lags < lags*ny + nx %+ flat*(ny+1)
539     error('Less observations than regressors: increase the # of obs or
          decrease the # of lags.')
540 end
541 xdata   = ones(T,nx);
542 if timetrend == 1
543     % xdata = [xdata [1:T]'];
544     xdata = [xdata [1-lags : T-lags]'];
545 end
546 if nexogenous > 0
547     xdata = [xdata exogenous(idx,:)];
548     XX    = [XX exogenous(idx(1)+lags : idx(end),:)];
549 end
550
551 % OLS estimate [NO DUMMY]:
552 varols  = rfvar3(ydata, lags, xdata, [T; T], 0, 0);
553
554
555 % specify the prior
556 if dummy == 1
557     % MINNESOTA PRIOR:
558     mnprior.tight         = minn_prior_tau;
559     mnprior.decay         = minn_prior_decay;
```

```
560     % Use only initializations lags for the variance prior
561     % vprior.sig             = std(y(firstobs+presample-lags : firstobs+
            presample,:))';
562     vprior.sig              = std(y(firstobs-lags : firstobs+presample-1,:))';
563     vprior.w                = minn_prior_omega;
564     lambda                  = minn_prior_lambda;
565     mu                      = minn_prior_mu;
566     [ydum, xdum, pbreaks] = varprior(ny, nx, lags, mnprior, vprior);
567     % Prior density
568     Tp = presample + lags;
569     if nx
570         xdata = xdata(1:Tp, :);
571     else
572         xdata = [];
573     end
574 %     varp              = rfvar3([ydata(1:Tp, :); ydum], lags, [xdata; xdum], [
        Tp; Tp + pbreaks], lambda, mu);
575     varp            = rfvar3([y(firstobs-lags : firstobs+presample-1, :); ydum
            ], lags, [xdata; xdum], [Tp; Tp + pbreaks], lambda, mu);
576     Tup             = size(varp.u, 1);
577     prior.df        = Tup - ny*lags - nx - flat*(ny+1);
578     prior.S         = varp.u' * varp.u;
579     prior.XXi       = varp.xxi;
580     prior.PhiHat    = varp.B;
581     prior.YYdum     = varp.y;
582     prior.XXdum     = varp.X;
583     if prior.df < ny
584         error('Too few degrees of freedom in the Inverse-Wishart part of prior
                distribution. You should increase training sample size.')
585     end
586     prior.minn_prior_tau    = minn_prior_tau;
587     prior.minn_prior_decay  = minn_prior_decay;
588     prior.minn_prior_lambda = minn_prior_lambda;
589     prior.minn_prior_mu     = minn_prior_mu;
590     prior.minn_prior_omega  = minn_prior_omega;
591 elseif dummy == 0
592     % JEFFREY OR UNIFORMATIVE PRIOR:
593     % [priors] = jeffrey(y,lags);
594     prior.name  = 'Jeffrey';
595
596 end
597
598 % specify the posterior (the varols agin on actual+dummy)
599 [posterior] = posterior_(y);
600
601
602 %****************************************************
```

```
%* Compute the log marginal data density for the VAR model
%******************************************************

posterior_int = matrictint(posterior.S, posterior.df, posterior.XXi);
if dummy == 1 % only for minnesota dummy
    prior_int = matrictint(prior.S, prior.df, prior.XXi);
    lik_nobs  = posterior.df - prior.df;
    log_dnsty = posterior_int - prior_int - 0.5*ny*lik_nobs*log(2*pi);

elseif dummy == 0 % jeffrey prior
    lik_nobs  = posterior.df;
    log_dnsty = posterior_int - 0.5*ny*lik_nobs*log(2*pi);

elseif dummy == 2 % conjugate MN-IW prior
    log_dnsty = mniw_log_dnsty(prior,posterior,varols);

end


%************************************************
%* Generating draws form the Posterior Distribution
%************************************************

% Preallocation of memory
% Matrices for collecting draws from Posterior Density
% Last dimension corresponds to a specific draw
Phi_draws     = zeros(ny*lags+nx+timetrend + nexogenous, ny, K);   %
    Autoregressive Parameters
Sigma_draws   = zeros(ny,ny,K);                       % Shocks Covariance
ir_draws      = zeros(ny,hor,ny,K);                   % variable, horizon, shock
     and draws - Cholesky IRF
irlr_draws    = zeros(ny,hor,ny,K);                   % variable, horizon, shock
     and draws - Long Run IRF
Qlr_draws     = zeros(ny,ny,K);                       % long run impact matrix
e_draws       = zeros(size(yy,1), ny,K);                    % residuals
yhatfut_no_shocks         = NaN(fhor, ny, K);   % forecasts with shocks
yhatfut_with_shocks       = NaN(fhor, ny, K);   % forecast without the shocks
yhatfut_cfrcst            = NaN(fhor, ny, K);   % forecast conditional on
    endogenous path
logL                      = NaN(K,1);

if signs_irf == 1
    irsign_draws = ir_draws;
    Omega_draws    = Sigma_draws;
end
if narrative_signs_irf == 1
    irnarrsign_draws = ir_draws;
```

```
646     Omegan_draws     = Sigma_draws;
647 end
648 if zeros_signs_irf == 1
649     irzerosign_draws   = ir_draws;
650     Omegaz_draws       = Sigma_draws;
651 end
652 if proxy_irf == 1
653     irproxy_draws = ir_draws;
654 end
655
656 % Settings for the forecasts
657 forecast_data.xdata       = ones(fhor, nx);
658 if timetrend
659     forecast_data.xdata = [forecast_data.xdata (T-lags+1 : T-lags+fhor)'];
660 end
661 if nexogenous>0
662     forecast_data.xdata = [forecast_data.xdata exogenous(T-lags+1 : T-lags+
        fhor,:)];
663 end
664 forecast_data.initval     = ydata(end-lags+1:end, :);
665
666 try
667     S_inv_upper_chol    = chol(inv(posterior.S));
668 catch
669     warning('POSTERIOR MEAN of SIGMA IS ILL-BEHAVED (NON-POSITIVE DEFINITE)')
670     try
671         warning('I will try with the LDL decomposition')
672         iS               = inv(posterior.S);
673         [L, D, P]        = ldl(iS);
674         S_inv_upper_chol = sqrt(D) * L' * P';%chol()
675     catch
676         warning('I will add +1e-05 to the diagonal')
677         S_inv_upper_chol    = chol(inv(posterior.S + 1e-05*eye(ny)));
678     end
679 end
680
681
682 XXi_lower_chol      = chol(posterior.XXi)';
683
684 nk                  = ny*lags+nx+timetrend + nexogenous;
685
686 % Declaration of the companion matrix
687 Companion_matrix = diag(ones(ny*(lags-1),1),-ny);
688
689 waitbar_yes = 0;
690 if K > 99
691     waitbar_yes = 1;
```

```
692     wb = waitbar(0, 'Generating draws from the Posterior Distribution');
693 end
694
695 for  d =  1 : K
696
697     %=========================================================================
698     % Inferece: Drawing from the posterior distribution
699     % Step 1: draw from the Covariance
700     Sigma = rand_inverse_wishart(ny, posterior.df, S_inv_upper_chol);
701
702     % Step 2: given the Covariance Matrix, draw from the AR parameters
703     Sigma_lower_chol = chol(Sigma)';
704     Phi1 = randn(nk * ny, 1);
705     Phi2 = kron(Sigma_lower_chol , XXi_lower_chol) * Phi1;
706     Phi3 = reshape(Phi2, nk, ny);
707     Phi  = Phi3 + posterior.PhiHat;
708
709
710     % store the draws
711     Phi_draws(:,:,d)   = Phi;
712     Sigma_draws(:,:,d) = Sigma;
713     errors             = yy - XX * Phi;
714     e_draws(:,:,d)     = errors;
715
716
717     %=========================================================================
718     % IRF
719     % Compute the impulse response funcitons
720     % with cholesky
721     if irf_1STD == 1
722         % one STD increase
723         ir_draws(:,:,:,d)      = iresponse(Phi,Sigma,hor,eye(ny));
724     else
725         % one percent increase
726         ir_draws(:,:,:,d)      = iresponse(Phi,Sigma,hor,eye(ny),0);
727     end
728     % with long run restrictions
729     if long_run_irf == 1
730         [irlr,Qlr]             = iresponse_longrun(Phi,Sigma,hor,lags);
731         irlr_draws(:,:,:,d)    = irlr;
732         Qlr_draws(:,:,d)       = Qlr;
733     end
734     % with sign restrictions
735     if signs_irf == 1
736         [irsign,Omega]         = iresponse_sign(Phi,Sigma,hor,signs);
737         irsign_draws(:,:,:,d)  = irsign;
738         Omega_draws(:,:,d)     = Omega;
```

```
739     end
740     % with narrative and sign restrictions
741     if narrative_signs_irf == 1
742         [irnarrsign,Omega]          = iresponse_sign_narrative(errors,Phi,Sigma
                ,hor,signs,narrative);
743         irnarrsign_draws(:,:,:,d)  = irnarrsign;
744         Omegan_draws(:,:,d)         = Omega;
745     end
746     % with zeros and sign restrictions
747     if zeros_signs_irf == 1          %= iresponse_zeros_signs( Phi,Sigma,bvar1.
            hor,lags,var_pos,f,sr);
748         [irzerosign,Omega]           = iresponse_zeros_signs(Phi,Sigma,hor,lags
                ,var_pos,f,sr);
749         irzerosign_draws(:,:,:,d)    = irzerosign;
750         Omegaz_draws(:,:,d)          = Omega;
751     end
752     % with proxy
753     if proxy_irf == 1
754         in.res                   = e_draws(:,:,d);
755         in.Phi                   = Phi_draws(:,:,d)  ;
756         in.Sigma                 = Sigma;
757         tmp_                     = iresponse_proxy(in);
758         irproxy_draws(:,:,1,d)   = tmp_.irs';
759         clear tmp_
760     end
761
762     %=========================================================================
763     % Forecasts
764     % compute the out of sample forecast (unconditional)
765     [frcst_no_shock,frcsts_with_shocks] = forecasts(forecast_data,Phi,Sigma,
            fhor,lags);
766     yhatfut_no_shocks(:,:,d)            = frcst_no_shock;
767     yhatfut_with_shocks(:,:,d)          = frcsts_with_shocks;
768
769     if cfrcst_yes == 1
770         % Forecast conditional on the path of an endo var using all shocks
771         [sims_with_endopath,EPS(:,:,d)] = ...
772             cforecasts(endo_path,endo_path_index,forecast_data,Phi,Sigma);
773         yhatfut_cfrcst(:,:,d) = sims_with_endopath;
774
775     elseif cfrcst_yes == 2
776         % Forecast conditional on the path of an endo var using only a
777         % subset of shocks. notice that the # of endo and # exo must coincide
778         % Omega is the structural orthonormal matrix
779         [sims_with_endopath,EPS(:,:,d)] = ...
780             cforecasts2(endo_path,endo_path_index,exo_index,forecast_data,Phi,
                    Sigma,Omega);
```

```
781             yhatfut_cfrcst(:,:,d) = sims_with_endopath;
782         end
783
784
785         %=========================================================================
786         % Mixed Frequency
787         if mixed_freq_on
788             % Checking the eigenvalues of the companion matrix (on or inside the
789             % unit circle). Needed for the initial of KF
790             Companion_matrix(1:ny,:) = Phi(1:ny*lags,:)';
791             test = (abs(eig(Companion_matrix)));
792             if any(test>1.0000000000001)
793                 KFoptions.initialCond = 1;
794             end
795             KFoptions.index   = index;
796             KFoptions.noprint = noprint;
797             % Forward Kalman Filter and Smoother
798             [KFout]           = kfilternan(Phi,Sigma,yoriginal,KFoptions);
799             yfill(:,:,d)      = KFout.smoothSt_plus_ss(:,KFout.index_var);
800             yfilt(:,:,d)      = KFout.filteredSt_plus_ss(:,KFout.index_var);
801             % recompute the posterior with smoothed data
802             [posterior1]      = posterior_(yfill(:,:,d));
803             S_inv_upper_chol  = chol(inv(posterior1.S));
804             XXi_lower_chol    = chol(posterior1.XXi)';
805             posterior.PhiHat = posterior1.PhiHat;
806             forecast_data.initval = yfill(end-lags+1:end, :, d);
807             logL(d) = KFout.logL;
808         end
809         if waitbar_yes, waitbar(d/K, wb); end
810     end
811     if waitbar_yes, close(wb); end
812
813
814     %******************************************************
815     %* Storing the resutls
816     %******************************************************
817
818     % classical inference: OLS estimator
819     BVAR.Phi_ols     = varols.B;
820     BVAR.e_ols       = varols.u;
821     BVAR.Sigma_ols   = 1/(nobs-nk)*varols.u'*varols.u;
822     [BVAR.InfoCrit.AIC, BVAR.InfoCrit.HQIC, BVAR.InfoCrit.BIC] = IC(BVAR, nobs, nk
        );
823     % the model with the lowest IC is preferred
824
825     % OLS irf
826     BVAR.ir_ols       = iresponse(BVAR.Phi_ols,BVAR.Sigma_ols,hor,eye(ny));
```

```
827 if long_run_irf == 1
828     [irlr,Qlr]              = iresponse_longrun(BVAR.Phi_ols,BVAR.Sigma_ols,
            hor,lags);
829     BVAR.irlr_ols           = irlr;
830     BVAR.Qlr_ols(:,:)       = Qlr;
831 end
832
833 % test the normality of the ols VAR residuals (matlab stat toolbox needed)
834 if  exist('kstest') ==2
835     for gg = 1 : ny
836         [H,Pv] = kstest(BVAR.e_ols(:,gg)/BVAR.Sigma_ols(gg,gg));
837         BVAR.HP(gg,:) = [H,Pv];
838     end
839 else
840     BVAR.HP = [];
841 end
842
843 % bayesian inference:
844 % the last dimension of these objects corresponds to a draw from the posterior
845
846 % inference and IRFs
847 BVAR.Phi_draws    = Phi_draws;          % draws from the autoregressive part
848 BVAR.Sigma_draws  = Sigma_draws;        % draws from the covarance matrix
849 BVAR.alpha_draws  = Phi_draws;          % Older name: draws from the
        autoregressive part
850 BVAR.sigma_draws  = Sigma_draws;        % Older name: draws from the covarance
         matrix
851
852 BVAR.ir_draws     = ir_draws;           % draws from the IRF with cholesky
853 BVAR.irlr_draws   = irlr_draws;         % draws from the IRF with Long Run
854 BVAR.Qlr_draws    = Qlr_draws;          % Long Run Rotation matrix
855 BVAR.lags         = lags;               % lags
856 BVAR.N            = ny;                 % number of variables
857 BVAR.e_draws      = e_draws;            % residuals
858 BVAR.e            = e_draws;            % backward compatible with earlier
        versions
859
860 BVAR.posterior    = posterior;
861 BVAR.prior        = prior;             % priors used
862 BVAR.logmlike     = log_dnsty;
863 BVAR.X            = var.X;              % regressors (including dummy if
        Minnesota)
864 BVAR.y            = var.y;              % dependent  (including dummy if
        Minnesota)
865 BVAR.XX           = XX;                 % regressors (no dummy)
866 BVAR.yy           = yy;                 % dependent  (no dummy)
867
```

```
% prediction
BVAR.fhor          = fhor;                % forecast horizon
BVAR.hor           = hor;                 % IRF horizon
BVAR.forecasts.no_shocks       = yhatfut_no_shocks;          % trajectories of
    forecasts without shocks
BVAR.forecasts.with_shocks     = yhatfut_with_shocks;        % trajectories of
    forecasts with shocks
BVAR.forecasts.conditional     = [];             % trajectories of conditional
    forecasts
BVAR.forecasts.EPScond         = [];             % shocks of conditional
    forecasts
if cfrcst_yes ~= 0
    BVAR.forecasts.conditional     = yhatfut_cfrcst;             % trajectories
        of forecasts
    BVAR.forecasts.EPScond         = EPS;                        % shocks of
        forecasts
end
BVAR.forecast_data             = forecast_data;

%
BVAR.varnames      = varnames;
BVAR.ndraws        = K;

if signs_irf == 1 && narrative_signs_irf == 0
    BVAR.irsign_draws = irsign_draws;
    BVAR.Omegas       = Omega_draws(:,:,d);
else
    BVAR.irsign_draws = [];
    BVAR.Omegas       = [];
end
if narrative_signs_irf == 1
    BVAR.irnarrsign_draws = irnarrsign_draws;
    BVAR.Omegan           = Omegan_draws(:,:,d);
else
    BVAR.irnarrsign_draws = [];
    BVAR.Omegan           = [];
end
if zeros_signs_irf == 1
    BVAR.irzerosign_draws   = irzerosign_draws;
    BVAR.Omegaz             = Omegaz_draws(:,:,d);
else
    BVAR.irzerosign_draws = [];
    BVAR.Omegaz           = [];
end
if proxy_irf == 1
    BVAR.irproxy_draws = irproxy_draws;
else
```

```
    BVAR.irproxy_draws= [];
end

% missing observations
BVAR.data         = y;                        % raw data
if mixed_freq_on
    BVAR.yfill = yfill;
    BVAR.yfilt = yfilt;
    BVAR.yinterpol = yinterpol;
    BVAR.logL  = logL;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% end of bvar.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*********************************************************
%*********************************************************
    function [posterior] = posterior_(y)

        % This part is needed in the case of missing values. The posterior
        % needs to be revaluated given a new value of the Kalman smoothed
        % observables
        %========================================
        ydata = y(idx, :);
        %========================================
        T     =  size(ydata, 1);
        xdata = ones(T,nx);
        if timetrend ==1
            xdata = [xdata [1-lags:T-lags]'];
            % xdata = [xdata [1:T]'];
        end
        if nexogenous >0
            xdata = [xdata exogenous(idx,:)];
        end
        % posterior density
        var = rfvar3([ydata; ydum], lags, [xdata; xdum], [T; T+pbreaks],
            lambda, mu);
        Tu = size(var.u, 1);

        if dummy ==  1
            %*********************************************************
            % Minnesota Prior
            %*********************************************************
            % Prior density
            Tp = presample + lags;
            if nx
```

```
955                 xdata = xdata(1:Tp, :);
956             else
957                 xdata = [];
958             end
959             % varp              = rfvar3([ydata(1:Tp, :); ydum], lags, [xdata;
                    xdum], [Tp; Tp + pbreaks], lambda, mu);
960             varp              = rfvar3([y(firstobs-lags : firstobs+presample-1,
                    :); ydum], lags, [xdata; xdum], [Tp; Tp + pbreaks], lambda, mu
                    );
961             Tup               = size(varp.u, 1);
962             prior.df          = Tup - ny*lags - nx - flat*(ny+1);
963             prior.S           = varp.u' * varp.u;
964             prior.XXi         = varp.xxi;
965             prior.PhiHat      = varp.B;
966             priors.YYdum      = varp.y;
967             priors.XXdum      = varp.X;
968             if prior.df < ny
969                 error('Too few degrees of freedom in the Inverse-Wishart part
                        of prior distribution. You should increase training sample
                        size.')
970             end
971             posterior.df      = Tu - ny*lags - nx - flat*(ny+1);
972             posterior.S       = var.u' * var.u;
973             posterior.XXi     = var.xxi;
974             posterior.PhiHat = var.B;
975
976         elseif dummy == 2
977             %-------------------------------------------------------
978             % Conjugate Prior
979             %-------------------------------------------------------
980             Ai                = inv(prior.Phi.cov);
981             posterior.df      = Tu - ny*lags - nx - nexogenous - prior.Sigma.df;
982             posterior.XXi     = inv(var.X'*var.X + Ai);
983             posterior.PhiHat = posterior.XXi * (var.X' * var.y + Ai * prior.
                    Phi.mean);
984             %posterior.S       = var.u' * var.u + prior.Sigma.scale ;
985             %posterior.S       = var.u' * var.u + prior.Sigma.scale  + (var.B -
                    prior.Phi.mean)' * Ai * (var.B - prior.Phi.mean);
986             %posterior.S       = var.u' * var.u + prior.Sigma.scale  + ...
987             %     (posterior.PhiHat - prior.Phi.mean)' * Ai * (var.B - prior.
                    Phi.mean);
988             % check
989             posterior.S = var.u' * var.u + prior.Sigma.scale + ...
990                 prior.Phi.mean' * Ai * prior.Phi.mean + ...
991                 var.B' * (var.X'*var.X) * var.B ...
992                 - posterior.PhiHat' * (var.X'*var.X + Ai) * posterior.PhiHat;
993
```

```
        else
            %*****************************************************
            % Flat Jeffrey Prior
            %*****************************************************
            posterior.df    = Tu - ny*lags - nx + flat*(ny+1) - nexogenous;
            posterior.S     = var.u' * var.u;
            posterior.XXi   = var.xxi;
            posterior.PhiHat = var.B;
        end

    end
%*****************************************************
%*****************************************************
    function [priors] = priors_( )

        priors.name  = 'N/A';

    end
%*****************************************************
%*****************************************************

end
```

bvar_max_hyper.m

```
function [postmode,log_dnsty,HH] = bvar_max_hyper(hyperpara,y,lags,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'bvar_max_hyper' maximizes the marginal likelihood over the
% hyperparameters of for the Minnesota prior

% Inputs:
% - hyperpara, Minnesota hyperpara over which maximize the marginal
% likelihood
% - y, data columns variables
% - lags, lag order of the VAR
% - options, see below for details

% Output: mode, Hessian and marginal likelihood at the mode

% Filippo Ferroni, 6/1/2017
% Revised, 3/21/2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

max_compute = 3;
```

```
21 lb          = -1e10*ones(length(hyperpara),1);
22 ub          = 1e10*ones(length(hyperpara),1);
23 objective_function = 'bvar_opt_hyperpara';
24 dummy_       = 1;
25 x0          = log(hyperpara);
26 index_fixed = [];
27 index_est   = 1:5;
28 hyperpara_fidex =  [];
29
30 if nargin > 3
31     if isfield(options,'index_est') ==1
32         clear x0 index_fixed index_est hyperpara_fidex;
33         index_est       = options.index_est;
34         index_fixed     = setdiff(1:5,index_est);
35         x0              = log(hyperpara(index_est));
36         hyperpara_fidex = log(hyperpara(index_fixed));
37     end
38     if isfield(options,'max_compute') ==1
39         max_compute    = options.max_compute;
40     end
41     if isfield(options,'lb') ==1
42         lb    = options.lb;
43         if length(lb) ~=  length(x0)
44             error('Mismatch between the size of lower bounds and the param
                vector');
45         end
46     end
47     if isfield(options,'ub') ==1
48         ub    = options.ub;
49         if length(ub) ~=  length(x0)
50             error('Mismatch between the size of upper bounds and the param
                vector');
51         end
52     end
53     if isfield(options,'objective_function') ==1
54         objective_function = options.objective_function;
55         dummy_ =0;
56     end
57 end
58
59
60 options.index_est       = index_est;
61 options.index_fixed     = index_fixed;
62 options.hyperpara_fidex =  hyperpara_fidex;
63
64
65 switch max_compute
```

```
66      case 1 % unconstraint
67          % Set default optimization options for fminunc.
68          optim_options = optimset('display','iter','MaxFunEvals',100000,'TolFun
                ',1e-8,'TolX',1e-6);
69          [xh,fh,~,~,~,H] = ...
70              fminunc(objective_function,x0,optim_options,y,lags,options);
71       %=====================================================================
72       case 2 % constraint
73          % Set default optimization options for fmincon.
74          optim_options = optimset('display','iter', 'LargeScale','off', '
                MaxFunEvals',100000, 'TolFun',1e-8, 'TolX',1e-6);
75          [xh,fh,~,~,~,~,H] = ...
76              fmincon(objective_function,x0,[],[],[],[],lb,ub,[],optim_options,y
                    ,lags,options);
77       %=====================================================================
78      case 3 % sims
79          crit = 10e-5;
80          nit  = 10e-4;
81          [fh, xh, ~, H, ~, ~, ~] = csminwel(objective_function,x0,.1*eye(length
                (x0)),[],crit,nit,y,lags,options);
82       %=====================================================================
83      case 7 % Matlab's simplex (Optimization toolbox needed).
84  %          optim_options = optimset('display','iter','MaxFunEvals',30000,'
    MaxIter',10000,'TolFun',1e-3,'TolX',1e-3,'OutputFcn',@outsavefun);
85          optim_options = optimset('display','iter','MaxFunEvals',30000,'MaxIter
                ',10000,'TolFun',1e-3,'TolX',1e-3);
86          [xh,fh,~,~] = fminsearch(objective_function,x0,optim_options,y,lags,
                options);
87          H = zeros(length(x0));
88  end
89
90  if dummy_ % Print the output for Minnesoty prior with dummy
91      postmode    = zeros(1,5);
92      postmode(options.index_est)   = exp(xh);
93      postmode(options.index_fixed) = exp(options.hyperpara_fidex);
94
95      % processing the output of the maximization
96      log_dnsty   = -fh;
97      JJ          = jacob_bvar(xh);
98      HH          =  JJ * H * JJ';
99
100     disp('=====================================================================');
101     disp('  ');
102     disp('** Initial Hyperpara Values and Log Density **');
103     disp('  ');
104     rownam = {'tau','decay','lambda','mu','omega','log density'};
105     minus_log_dnsty_0     = bvar_opt_hyperpara(x0,y,lags,options);
```

```
    x                = [hyperpara'; -minus_log_dnsty_0];
    for jj =1: length(x)
        X = sprintf('%s = %0.5g',rownam{jj},x(jj));
        disp(X)
    end
    disp('  ');
    disp('** Posterior Mode: (Minimization of -Log Density)  **');
    disp('  ');
    x      = [postmode'; log_dnsty];
    for jj =1: length(x)
        X = sprintf('%s = %0.5g',rownam{jj},x(jj));
        disp(X)
    end
else % other maximization e.g. conjugate MNIW
    postmode    = exp(xh);
    % processing the output of the maximization
    log_dnsty   = -fh;
    JJ          = jacob_bvar(xh);
    HH          =  JJ * H * JJ';
end


end
```

bvar_ml.m

```
function log_dnsty = bvar_ml(hyperpara,y,lags,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'bvar_ml' computes the marginal likelihood for the minnesota prior

% Inputs:
% - hyperpara, Minnesota hyperpara over which maximize the marginal
% likelihood
% - y, data columns variables
% - lags, lag order of the VAR
% - options, see below for details

% Output: marginal data density


% Filippo Ferroni, 6/1/2017
% Revised, 3/21/2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
20 %******************************************************
21 % SETTINGS
22 %******************************************************
23
24
25 % randn('state',999);
26 % rand('state',999);
27 %
28 ny                  = size(y, 2);
29 first_obs           = lags+1;
30 presample           = 0;
31 noconstant          = 0;
32
33 bvar_prior_tau      = hyperpara(1);%3;
34 bvar_prior_decay    = hyperpara(2);%0.5;
35 bvar_prior_lambda   = hyperpara(3);%5;
36 bvar_prior_mu       = hyperpara(4);%2;
37 bvar_prior_omega    = hyperpara(5);%1;
38 unit_root_          = ones(ny,1);
39 flat                = 0;
40
41
42 if nargin > 3
43 %     fields = fieldnames(options);
44 %     nf     = size(fieldnames(options),1);
45 %     for i = 1 : nf
46 %         if strcmp(fields{i},'bvar_prior_train')
47 %             bvar_prior_train = options.bvar_prior_train;
48 %         elseif strcmp(fields{i},'first_obs')
49 %             first_obs = options.first_obs;
50 %         elseif strcmp(fields{i},'presample')
51 %             presample = options.presample;
52 %         elseif strcmp(fields{i},'train')
53 %             train = options.train;
54 %         elseif strcmp(fields{i},'noconstant')
55 %             noconstant = options.nocostant;
56 %         end
57 %     end
58     if isfield(options,'first_obs')==1
59         first_obs = options.first_obs;
60     end
61     if isfield(options,'presample')==1
62         presample = options.presample;
63     end
64     if isfield(options,'noconstant')==1
65         noconstant = options.nocostant;
66         % MINNESOTA PRIOR
```

```
67      end
68      if isfield(options,'unit_root_')==1
69          unit_root_ = options.unit_root_;
70          %  MINNESOTA PRIOR: unit root assumption only for a subset of var
71      end
72
73  end
74
75  nobs = size(y,1)-first_obs+1;
76
77  if (first_obs+ nobs-1)> size(y,1)
78      fprintf('Incorrect or missing specification of the number of observations.
            nobs can be at most %4u\n',size(y,1)-first_obs+1);
79      error('Inconsistent number of observations.')
80  end
81
82  % Parameters for prior
83  if first_obs + presample <= lags
84      error('first_obs+presample should be > lags (for initializing the VAR)')
85  end
86
87  if first_obs + presample  <= lags
88      error('first_obs+presample should be > nlags (for initializating the VAR)'
            )
89  end
90
91  idx = first_obs+presample-lags:first_obs+nobs-1;
92
93
94  if noconstant
95          nx = 0;
96  else
97          nx = 1;
98  end
99
100
101 mnprior.tight = bvar_prior_tau;
102 mnprior.decay = bvar_prior_decay;
103 mnprior.unit_root_ = unit_root_;
104 % Use only initializations lags for the variance prior
105 vprior.sig = std(y(first_obs+presample-lags : first_obs+presample,:))';
106 vprior.w = bvar_prior_omega;
107 lambda = bvar_prior_lambda;
108 mu     = bvar_prior_mu;
109 [ydum, xdum, pbreaks] = varprior(ny, nx, lags, mnprior, vprior);
110
111 ydata = y(idx, :);
```

```
112 T     =  size(ydata, 1);
113 xdata = ones(T,nx);
114 % posterior density
115 var = rfvar3([ydata; ydum], lags, [xdata; xdum], [T; T+pbreaks], lambda, mu);
116 Tu = size(var.u, 1);
117
118 % Prior density
119 Tp = presample + lags;
120 if nx
121     xdata = xdata(1:Tp, :);
122 else
123     xdata = [];
124 end
125 varp = rfvar3([ydata(1:Tp, :); ydum], lags, [xdata; xdum], [Tp; Tp + pbreaks],
        lambda, mu);
126 Tup = size(varp.u, 1);
127
128 prior.df        = Tup - ny*lags - nx - flat*(ny+1);
129 prior.S         = varp.u' * varp.u;
130 prior.XXi       = varp.xxi;
131 prior.PhiHat    = varp.B;
132
133 priors.YYdum = varp.y;
134 priors.XXdum = varp.X;
135
136 if prior.df < ny
137     error('Too few degrees of freedom in the Inverse-Wishart part of prior
          distribution. You should increase training sample size.')
138 end
139 posterior.df     = Tu - ny*lags - nx - flat*(ny+1);
140 posterior.S      = var.u' * var.u;
141 posterior.XXi    = var.xxi;
142 posterior.PhiHat = var.B;
143
144
145 %******************************************************
146 %* Compute the log marginal data density for the VAR model
147 %******************************************************
148
149 posterior_int   = matrictint(posterior.S, posterior.df, posterior.XXi);
150 prior_int       = matrictint(prior.S, prior.df, prior.XXi);
151 lik_nobs        = posterior.df - prior.df;
152 log_dnsty       = posterior_int - prior_int - 0.5*ny*lik_nobs*log(2*pi);
153 % minus_log_dnsty = -log_dnsty;
154 end
```

bvar_opt_hyperpara.m

```
function minus_log_dnsty = bvar_opt_hyperpara(hyperpara,y,lags,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'bvar_opt_hyperpara' computes the marginal likelihood over the
% hyperparameters of for the Minnesota prior
% Bridge function between 'bvar_ml' and the minimization routine
% 'bvar_max_hyper'

% Inputs:
%- hyperpara, Minnesota hyperpara over which maximize the marginal
% likelihood
%- y, data columns variables
%- lags, lag order of the VAR
%- options, see below for details

% Output: marginal data density

% Filippo Ferroni, 6/1/2017
% Revised, 3/21/2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin <4
    hyperpara = exp(hyperpara);
    log_dnsty = bvar_ml(hyperpara,y,lags);
else
    hyperpara(options.index_est)   = exp(hyperpara);
    if isempty(options.index_fixed) == 0
        hyperpara(options.index_fixed) = exp(options.hyperpara_fidex);
    end
    log_dnsty                      = bvar_ml(hyperpara,y,lags,options);
end

minus_log_dnsty = -log_dnsty;
```

cforecasts.m

```
function [sims_with_endopath,EPSn] = cforecasts(endo_path,endo_index,
    forecast_data,Phi,Sigma)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'cforecasts' performs the forecast conditional on a path for one or more
% endogenous variables using all the shocks of the VAR
% references: Waggoner and Zha (1999) and Jarocinski (2010)
```

```
% Inputs:
% - endo_path, path for the endogenous variables
% - endo_index, order of the endogenous variables with a contional path
% - forecast_data, last data
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks

% Output:
% - sims_with_endopath
% 1st dimension:   horizon
% 2nd dimension:   variable
% - EPSn, shocks generating the path

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fhor,Ncondvar] = size(endo_path);
Nvar            = size(Sigma,2);

if size(Ncondvar,2) ~= length(endo_index)
    error('something went wrong: the number of path and the number of vars are not consistent.')
end
if size(endo_index,1) > size(endo_index,2)
    error('the index needs to be a row vector');
end
Nres = size(endo_index,2)*fhor;

[F,~,~,const,~,lags]     = var2ss(Phi,Sigma);

% endo_path_deviation = endo_path - repmat(const',1,fhor);

% no shocks forecasts
[sims_no_shock,~]    = forecasts(forecast_data,Phi,Sigma,fhor,lags);

% restrictions
% err                = endo_path_deviation - sims_no_shock(:,endo_path_index);
% err = zeros(Nvar*fhor);
err = endo_path - sims_no_shock(:,endo_index);
err = reshape(err',size(endo_index,1)*fhor,1);

% 1 standard deviation increase (if unit = eye(N))
```

```
[C] = chol(Sigma,'lower');

R    = zeros(Nvar * fhor);
tmp0 = [];
for ff = 1 :fhor
    tmp  = F^(ff-1);
    tmp0 = [tmp(1:Nvar,1:Nvar) * C tmp0];
    R((ff-1)*Nvar + 1 : (ff)*Nvar, 1: size(tmp0,2) ) = tmp0;
    index(ff,:) = (ff-1)*Nvar + endo_index;
end
index = reshape(index',Nres,1);
% index= [4  10];
Rtilde = R(index,:);

[U,D,V] = svd(Rtilde);
V1 = V(:,1:Nres);
V2 = V(:,Nres+1:end);

eps = V1*inv(D(1:Nres,1:Nres))*U'*err + V2 * randn(size(R,1)-Nres,1);

% orthogonal schoks
EPSn = reshape(eps,Nvar,fhor);
% reduced form shocks
EPS  = C*EPSn;

[~,sims_with_endopath] = forecasts(forecast_data,Phi,Sigma,fhor,lags,EPS');
```

cforecasts2.m

```
function [sims_with_endopath,EPSn] = cforecasts2(endo_path,endo_index,
    exo_index,forecast_data,Phi,Sigma,Omega,EPSi,epslags_)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'cforecasts' performs the forecast conditional on a path for one or more
% endogenous variables using some of teh STRUCTURAL shocks of the VAR
% references: Waggoner and Zha (1999), Maih (2010), Dynare reference manual
% Inputs:
% - endo_path, path for the endogenous variables
% - endo_index, order of the endogenous variables with a contional path
% - exo_index, order of the structural VAR shocks to use
% - forecast_data, last data
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - Omega, Rotation for identification

```

```
% Output:
%— sims_with_endopath
% 1st dimension:   horizon
% 2nd dimension:   variable
%— EPSn, shocks generating the path

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fhor,Ncondvar] = size(endo_path);
Nvar            = size(Sigma,2);

if nargin < 7
    Omega = eye(Nvar);
end
if nargin < 8
    % drawing the shocks
    % 1 standard deviation increase (if unit = eye(N))
    EPSi                = randn(fhor,Nvar);
    EPSi(:,exo_index)   = zeros(fhor,length(exo_index));
end
if nargin < 9
    epslags_ = 0;
end

if size(Ncondvar,2) ~= length(endo_index)
    error('something went wrong: the number of path and the number of vars are
        not consistent.')
end
if size(endo_index,1) > size(endo_index,2)
    error('the index needs to be a row vector');
end
if size(endo_index,2) ~= size(exo_index,2)
    error('the number of conditioning shocks needs to coincide with the number
        of conditioning endo');
end

[A,B,C,const,Sigma,lags]   = var2ss(Phi,Sigma);
sims_with_endopath = nan(fhor,Nvar);
EPSn               = nan(fhor,Nvar);

% % unconditional forecasts without zero
% [sims_no_shock,~]    = forecasts(forecast_data,Phi,Sigma,fhor,lags);
%
```

```
61 %% restrictions
62 %% err                    = endo_path_deviation - sims_no_shock(:,
       endo_path_index);
63 %% err = zeros(Nvar*fhor);
64 % err = (endo_path - sims_no_shock(:,endo_index))';
65 %% err = reshape(err',size(endo_index,1)*fhor,1);
66
67 [C]                  = chol(Sigma,'lower');
68 R                    = C*Omega;
69 EPS                  = (R*EPSi')';
70
71 if epslags_ == 1
72     tmp = A*B*R;
73     RR  = tmp(1:Nvar,1:Nvar);
74
75 elseif epslags_ == 2
76     tmp = A^2*B*R;
77     RR  = tmp(1:Nvar,1:Nvar);
78
79 elseif epslags_ == 3
80     tmp = A^3*B*R;
81     RR  = tmp(1:Nvar,1:Nvar);
82 else
83     RR  = R;
84 end
85
86 % With shocks but exo_index
87 lags_data = forecast_data.initval;
88 e         = zeros(length(exo_index),fhor);
89
90 for t = 1 : fhor
91     X = [ reshape(flipdim(lags_data, 1)', 1, Nvar*lags) forecast_data.xdata(t,
           :) ];
92     % unconditional forecasts with all shocks but 'exo_index'
93     shock  = EPS(t,:);
94     y = X * Phi + shock;
95     e(:,t) = inv(RR(endo_index,exo_index))*(endo_path(t,:) - y(endo_index));
96     y = y + (RR(:,exo_index)*e(:,t))';
97     lags_data(1:end-1,:) = lags_data(2:end, :);
98     lags_data(end,:) = y;
99     sims_with_endopath(t, :) = y;
100 end
101
102 EPSn  = EPSi;
103 EPSn(:,exo_index) = e';
```

checkrestrictions.m

```
function d = checkrestrictions(restriction,y,v)

% Check the restrictions

if nargin<3
    v = [];
end

d=0;
count   = 0;
for ii = 1 : size(restriction,2)
    tmp = eval(restriction{ii});
    count = count + min(tmp);
end
if isempty(count)==1
    error('There is a nan in the narrative restrictions.');
end
if count == size(restriction,2) % if all signs are verified stop
    d=1;
end
```

colorspace.m

```
function varargout = colorspace(Conversion,varargin)
%COLORSPACE  Transform a color image between color representations.
%   B = COLORSPACE(S,A) transforms the color representation of image A
%   where S is a string specifying the conversion.  The input array A
%   should be a real full double array of size Mx3 or MxNx3.  The output B
%   is the same size as A.
%
%   S tells the source and destination color spaces, S = 'dest<-src', or
%   alternatively, S = 'src->dest'.  Supported color spaces are
%
%      'RGB'               sRGB IEC 61966-2-1
%      'YCbCr'             Luma + Chroma ("digitized" version of Y'PbPr)
%      'JPEG-YCbCr'        Luma + Chroma space used in JFIF JPEG
%      'YDbDr'             SECAM Y'DbDr Luma + Chroma
%      'YPbPr'             Luma (ITU-R BT.601) + Chroma
%      'YUV'               NTSC PAL Y'UV Luma + Chroma
%      'YIQ'               NTSC Y'IQ Luma + Chroma
%      'HSV' or 'HSB'      Hue Saturation Value/Brightness
%      'HSL' or 'HLS'      Hue Saturation Luminance
%      'HSI'               Hue Saturation Intensity
```

```
21 %       'XYZ'                CIE 1931 XYZ
22 %       'Lab'                CIE 1976 L*a*b* (CIELAB)
23 %       'Luv'                CIE L*u*v* (CIELUV)
24 %       'LCH'                CIE L*C*H* (CIELCH)
25 %       'CAT02 LMS'          CIE CAT02 LMS
26 %
27 %    All conversions assume 2 degree observer and D65 illuminant.
28 %
29 %    Color space names are case insensitive and spaces are ignored.  When
30 %    sRGB is the source or destination, it can be omitted.  For example
31 %    'yuv<-' is short for 'yuv<-rgb'.
32 %
33 %    For sRGB, the values should be scaled between 0 and 1.  Beware that
34 %    transformations generally do not constrain colors to be "in gamut."
35 %    Particularly, transforming from another space to sRGB may obtain
36 %    R'G'B' values outside of the [0,1] range.  So the result should be
37 %    clamped to [0,1] before displaying:
38 %       image(min(max(B,0),1));  % Clamp B to [0,1] and display
39 %
40 %    sRGB (Red Green Blue) is the (ITU-R BT.709 gamma-corrected) standard
41 %    red-green-blue representation of colors used in digital imaging.  The
42 %    components should be scaled between 0 and 1.  The space can be
43 %    visualized geometrically as a cube.
44 %
45 %    Y'PbPr, Y'CbCr, Y'DbDr, Y'UV, and Y'IQ are related to sRGB by linear
46 %    transformations.  These spaces separate a color into a grayscale
47 %    luminance component Y and two chroma components.  The valid ranges of
48 %    the components depends on the space.
49 %
50 %    HSV (Hue Saturation Value) is related to sRGB by
51 %       H = hexagonal hue angle    (0 <= H < 360),
52 %       S = C/V                    (0 <= S <= 1),
53 %       V = max(R',G',B')          (0 <= V <= 1),
54 %    where C = max(R',G',B') - min(R',G',B').  The hue angle H is computed on
55 %    a hexagon.  The space is geometrically a hexagonal cone.
56 %
57 %    HSL (Hue Saturation Lightness) is related to sRGB by
58 %       H = hexagonal hue angle                 (0 <= H < 360),
59 %       S = C/(1 - |2L-1|)                      (0 <= S <= 1),
60 %       L = (max(R',G',B') + min(R',G',B'))/2   (0 <= L <= 1),
61 %    where H and C are the same as in HSV.  Geometrically, the space is a
62 %    double hexagonal cone.
63 %
64 %    HSI (Hue Saturation Intensity) is related to sRGB by
65 %       H = polar hue angle          (0 <= H < 360),
66 %       S = 1 - min(R',G',B')/I      (0 <= S <= 1),
67 %       I = (R'+G'+B')/3             (0 <= I <= 1).
```

```
68 %  Unlike HSV and HSL, the hue angle H is computed on a circle rather than
69 %  a hexagon.
70 %
71 %  CIE XYZ is related to sRGB by inverse gamma correction followed by a
72 %  linear transform.  Other CIE color spaces are defined relative to XYZ.
73 %
74 %  CIE L*a*b*, L*u*v*, and L*C*H* are nonlinear functions of XYZ.  The L*
75 %  component is designed to match closely with human perception of
76 %  lightness.  The other two components describe the chroma.
77 %
78 %  CIE CAT02 LMS is the linear transformation of XYZ using the MCAT02
79 %  chromatic adaptation matrix.  The space is designed to model the
80 %  response of the three types of cones in the human eye, where L, M, S,
81 %  correspond respectively to red ("long"), green ("medium"), and blue
82 %  ("short").
83
84 % Pascal Getreuer 2005--2010
85
86
87 %%% Input parsing %%%
88 if nargin < 2, error('Not enough input arguments.'); end
89 [SrcSpace,DestSpace] = parse(Conversion);
90
91 if nargin == 2
92    Image = varargin{1};
93 elseif nargin >= 3
94    Image = cat(3,varargin{:});
95 else
96    error('Invalid number of input arguments.');
97 end
98
99 FlipDims = (size(Image,3) == 1);
100
101 if FlipDims, Image = permute(Image,[1,3,2]); end
102 if ~isa(Image,'double'), Image = double(Image)/255; end
103 if size(Image,3) ~= 3, error('Invalid input size.'); end
104
105 SrcT = gettransform(SrcSpace);
106 DestT = gettransform(DestSpace);
107
108 if ~ischar(SrcT) && ~ischar(DestT)
109    % Both source and destination transforms are affine, so they
110    % can be composed into one affine operation
111    T = [DestT(:,1:3)*SrcT(:,1:3),DestT(:,1:3)*SrcT(:,4)+DestT(:,4)];
112    Temp = zeros(size(Image));
113    Temp(:,:,1) = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) + T(7)*Image(:,:,3) + T
        (10);
```

```
114     Temp(:,:,2) = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) + T(8)*Image(:,:,3) + T
            (11);
115     Temp(:,:,3) = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) + T(9)*Image(:,:,3) + T
            (12);
116     Image = Temp;
117  elseif ~ischar(DestT)
118     Image = rgb(Image,SrcSpace);
119     Temp = zeros(size(Image));
120     Temp(:,:,1) = DestT(1)*Image(:,:,1) + DestT(4)*Image(:,:,2) + DestT(7)*
            Image(:,:,3) + DestT(10);
121     Temp(:,:,2) = DestT(2)*Image(:,:,1) + DestT(5)*Image(:,:,2) + DestT(8)*
            Image(:,:,3) + DestT(11);
122     Temp(:,:,3) = DestT(3)*Image(:,:,1) + DestT(6)*Image(:,:,2) + DestT(9)*
            Image(:,:,3) + DestT(12);
123     Image = Temp;
124  else
125     Image = feval(DestT,Image,SrcSpace);
126  end
127
128  %%% Output format %%%
129  if nargout > 1
130     varargout = {Image(:,:,1),Image(:,:,2),Image(:,:,3)};
131  else
132     if FlipDims, Image = permute(Image,[1,3,2]); end
133     varargout = {Image};
134  end
135
136  return;
137
138
139  function [SrcSpace,DestSpace] = parse(Str)
140  % Parse conversion argument
141
142  if ischar(Str)
143     Str = lower(strrep(strrep(Str,'-',''),'=',''));
144     k = find(Str == '>');
145
146     if length(k) == 1          % Interpret the form 'src->dest'
147        SrcSpace = Str(1:k-1);
148        DestSpace = Str(k+1:end);
149     else
150        k = find(Str == '<');
151
152        if length(k) == 1       % Interpret the form 'dest<-src'
153           DestSpace = Str(1:k-1);
154           SrcSpace = Str(k+1:end);
155        else
```

```
156           error(['Invalid conversion, ''',Str,'''.']);
157        end
158     end
159
160     SrcSpace = alias(SrcSpace);
161     DestSpace = alias(DestSpace);
162  else
163     SrcSpace = 1;              % No source pre-transform
164     DestSpace = Conversion;
165     if any(size(Conversion) ~= 3), error('Transformation matrix must be 3x3.');
            end
166  end
167  return;
168
169
170  function Space = alias(Space)
171  Space = strrep(strrep(Space,'cie',''),' ','');
172
173  if isempty(Space)
174     Space = 'rgb';
175  end
176
177  switch Space
178  case {'ycbcr','ycc'}
179     Space = 'ycbcr';
180  case {'hsv','hsb'}
181     Space = 'hsv';
182  case {'hsl','hsi','hls'}
183     Space = 'hsl';
184  case {'rgb','yuv','yiq','ydbdr','ycbcr','jpegycbcr','xyz','lab','luv','lch'}
185     return;
186  end
187  return;
188
189
190  function T = gettransform(Space)
191  % Get a colorspace transform: either a matrix describing an affine transform,
192  % or a string referring to a conversion subroutine
193  switch Space
194  case 'ypbpr'
195     T =
           [0.299,0.587,0.114,0;-0.1687367,-0.331264,0.5,0;0.5,-0.418688,-0.081312,0];

196  case 'yuv'
197     % sRGB to NTSC/PAL YUV
198     % Wikipedia: http://en.wikipedia.org/wiki/YUV
199     T = [0.299,0.587,0.114,0;-0.147,-0.289,0.436,0;0.615,-0.515,-0.100,0];
```

```
200 case 'ydbdr'
201     % sRGB to SECAM YDbDr
202     % Wikipedia: http://en.wikipedia.org/wiki/YDbDr
203     T = [0.299,0.587,0.114,0;-0.450,-0.883,1.333,0;-1.333,1.116,0.217,0];
204 case 'yiq'
205     % sRGB in [0,1] to NTSC YIQ in
            [0,1];[-0.595716,0.595716];[-0.522591,0.522591];
206     % Wikipedia: http://en.wikipedia.org/wiki/YIQ
207     T =
            [0.299,0.587,0.114,0;0.595716,-0.274453,-0.321263,0;0.211456,-0.522591,0.311135,0];

208 case 'ycbcr'
209     % sRGB (range [0,1]) to ITU-R BRT.601 (CCIR 601) Y'CbCr
210     % Wikipedia: http://en.wikipedia.org/wiki/YCbCr
211     % Poynton, Equation 3, scaling of R'G'B to Y'PbPr conversion
212     T =
            [65.481,128.553,24.966,16;-37.797,-74.203,112.0,128;112.0,-93.786,-18.214,128];

213 case 'jpegycbcr'
214     % Wikipedia: http://en.wikipedia.org/wiki/YCbCr
215     T =
            [0.299,0.587,0.114,0;-0.168736,-0.331264,0.5,0.5;0.5,-0.418688,-0.081312,0.5]*255;

216 case {'rgb','xyz','hsv','hsl','lab','luv','lch','cat02lms'}
217     T = Space;
218 otherwise
219     error(['Unknown color space, ''',Space,'''.']);
220 end
221 return;
222
223
224 function Image = rgb(Image,SrcSpace)
225 % Convert to sRGB from 'SrcSpace'
226 switch SrcSpace
227 case 'rgb'
228     return;
229 case 'hsv'
230     % Convert HSV to sRGB
231     Image = huetorgb((1 - Image(:,:,2)).*Image(:,:,3),Image(:,:,3),Image(:,:,1)
            );
232 case 'hsl'
233     % Convert HSL to sRGB
234     L = Image(:,:,3);
235     Delta = Image(:,:,2).*min(L,1-L);
236     Image = huetorgb(L-Delta,L+Delta,Image(:,:,1));
237 case {'xyz','lab','luv','lch','cat02lms'}
238     % Convert to CIE XYZ
```

```
239     Image = xyz(Image,SrcSpace);
240     % Convert XYZ to RGB
241     T = [3.2406, -1.5372, -0.4986; -0.9689, 1.8758, 0.0415; 0.0557, -0.2040,
            1.057];
242     R = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) + T(7)*Image(:,:,3);  % R
243     G = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) + T(8)*Image(:,:,3);  % G
244     B = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) + T(9)*Image(:,:,3);  % B
245     % Desaturate and rescale to constrain resulting RGB values to [0,1]
246     AddWhite = -min(min(min(R,G),B),0);
247     R = R + AddWhite;
248     G = G + AddWhite;
249     B = B + AddWhite;
250     % Apply gamma correction to convert linear RGB to sRGB
251     Image(:,:,1) = gammacorrection(R);  % R'
252     Image(:,:,2) = gammacorrection(G);  % G'
253     Image(:,:,3) = gammacorrection(B);  % B'
254 otherwise  % Conversion is through an affine transform
255     T = gettransform(SrcSpace);
256     temp = inv(T(:,1:3));
257     T = [temp,-temp*T(:,4)];
258     R = T(1)*Image(:,:,1) + T(4)*Image(:,:,2) + T(7)*Image(:,:,3) + T(10);
259     G = T(2)*Image(:,:,1) + T(5)*Image(:,:,2) + T(8)*Image(:,:,3) + T(11);
260     B = T(3)*Image(:,:,1) + T(6)*Image(:,:,2) + T(9)*Image(:,:,3) + T(12);
261     Image(:,:,1) = R;
262     Image(:,:,2) = G;
263     Image(:,:,3) = B;
264 end
265
266 % Clip to [0,1]
267 Image = min(max(Image,0),1);
268 return;
269
270
271 function Image = xyz(Image,SrcSpace)
272 % Convert to CIE XYZ from 'SrcSpace'
273 WhitePoint = [0.950456,1,1.088754];
274
275 switch SrcSpace
276 case 'xyz'
277     return;
278 case 'luv'
279     % Convert CIE L*uv to XYZ
280     WhitePointU = (4*WhitePoint(1))./(WhitePoint(1) + 15*WhitePoint(2) + 3*
            WhitePoint(3));
281     WhitePointV = (9*WhitePoint(2))./(WhitePoint(1) + 15*WhitePoint(2) + 3*
            WhitePoint(3));
282     L = Image(:,:,1);
```

```
283     Y = (L + 16)/116;
284     Y = invf(Y)*WhitePoint(2);
285     U = Image(:,:,2)./(13*L + 1e-6*(L==0)) + WhitePointU;
286     V = Image(:,:,3)./(13*L + 1e-6*(L==0)) + WhitePointV;
287     Image(:,:,1) = -(9*Y.*U)./((U-4).*V - U.*V);                    % X
288     Image(:,:,2) = Y;                                               % Y
289     Image(:,:,3) = (9*Y - (15*V.*Y) - (V.*Image(:,:,1))./(3*V);     % Z
290 case {'lab','lch'}
291     Image = lab(Image,SrcSpace);
292     % Convert CIE L*ab to XYZ
293     fY = (Image(:,:,1) + 16)/116;
294     fX = fY + Image(:,:,2)/500;
295     fZ = fY - Image(:,:,3)/200;
296     Image(:,:,1) = WhitePoint(1)*invf(fX);  % X
297     Image(:,:,2) = WhitePoint(2)*invf(fY);  % Y
298     Image(:,:,3) = WhitePoint(3)*invf(fZ);  % Z
299 case 'cat02lms'
300     % Convert CAT02 LMS to XYZ
301     T = inv([0.7328, 0.4296, -0.1624;-0.7036, 1.6975, 0.0061; 0.0030, 0.0136,
            0.9834]);
302     L = Image(:,:,1);
303     M = Image(:,:,2);
304     S = Image(:,:,3);
305     Image(:,:,1) = T(1)*L + T(4)*M + T(7)*S;  % X
306     Image(:,:,2) = T(2)*L + T(5)*M + T(8)*S;  % Y
307     Image(:,:,3) = T(3)*L + T(6)*M + T(9)*S;  % Z
308 otherwise   % Convert from some gamma-corrected space
309     % Convert to sRGB
310     Image = rgb(Image,SrcSpace);
311     % Undo gamma correction
312     R = invgammacorrection(Image(:,:,1));
313     G = invgammacorrection(Image(:,:,2));
314     B = invgammacorrection(Image(:,:,3));
315     % Convert RGB to XYZ
316     T = inv([3.2406, -1.5372, -0.4986; -0.9689, 1.8758, 0.0415; 0.0557,
            -0.2040, 1.057]);
317     Image(:,:,1) = T(1)*R + T(4)*G + T(7)*B;  % X
318     Image(:,:,2) = T(2)*R + T(5)*G + T(8)*B;  % Y
319     Image(:,:,3) = T(3)*R + T(6)*G + T(9)*B;  % Z
320 end
321 return;
322
323
324 function Image = hsv(Image,SrcSpace)
325 % Convert to HSV
326 Image = rgb(Image,SrcSpace);
327 V = max(Image,[],3);
```

```
328 S = (V - min(Image,[],3))./(V + (V == 0));
329 Image(:,:,1) = rgbtohue(Image);
330 Image(:,:,2) = S;
331 Image(:,:,3) = V;
332 return;
333
334
335 function Image = hsl(Image,SrcSpace)
336 % Convert to HSL
337 switch SrcSpace
338 case 'hsv'
339    % Convert HSV to HSL
340    MaxVal = Image(:,:,3);
341    MinVal = (1 - Image(:,:,2)).*MaxVal;
342    L = 0.5*(MaxVal + MinVal);
343    temp = min(L,1-L);
344    Image(:,:,2) = 0.5*(MaxVal - MinVal)./(temp + (temp == 0));
345    Image(:,:,3) = L;
346 otherwise
347    Image = rgb(Image,SrcSpace);  % Convert to sRGB
348    % Convert sRGB to HSL
349    MinVal = min(Image,[],3);
350    MaxVal = max(Image,[],3);
351    L = 0.5*(MaxVal + MinVal);
352    temp = min(L,1-L);
353    S = 0.5*(MaxVal - MinVal)./(temp + (temp == 0));
354    Image(:,:,1) = rgbtohue(Image);
355    Image(:,:,2) = S;
356    Image(:,:,3) = L;
357 end
358 return;
359
360
361 function Image = lab(Image,SrcSpace)
362 % Convert to CIE L*a*b* (CIELAB)
363 WhitePoint = [0.950456,1,1.088754];
364
365 switch SrcSpace
366 case 'lab'
367    return;
368 case 'lch'
369    % Convert CIE L*CH to CIE L*ab
370    C = Image(:,:,2);
371    Image(:,:,2) = cos(Image(:,:,3)*pi/180).*C;  % a*
372    Image(:,:,3) = sin(Image(:,:,3)*pi/180).*C;  % b*
373 otherwise
374    Image = xyz(Image,SrcSpace);  % Convert to XYZ
```

```
375     % Convert XYZ to CIE L*a*b*
376     X = Image(:,:,1)/WhitePoint(1);
377     Y = Image(:,:,2)/WhitePoint(2);
378     Z = Image(:,:,3)/WhitePoint(3);
379     fX = f(X);
380     fY = f(Y);
381     fZ = f(Z);
382     Image(:,:,1) = 116*fY - 16;     % L*
383     Image(:,:,2) = 500*(fX - fY);  % a*
384     Image(:,:,3) = 200*(fY - fZ);  % b*
385 end
386 return;
387
388
389 function Image = luv(Image,SrcSpace)
390 % Convert to CIE L*u*v* (CIELUV)
391 WhitePoint = [0.950456,1,1.088754];
392 WhitePointU = (4*WhitePoint(1))./(WhitePoint(1) + 15*WhitePoint(2) + 3*
        WhitePoint(3));
393 WhitePointV = (9*WhitePoint(2))./(WhitePoint(1) + 15*WhitePoint(2) + 3*
        WhitePoint(3));
394
395 Image = xyz(Image,SrcSpace); % Convert to XYZ
396 Denom = Image(:,:,1) + 15*Image(:,:,2) + 3*Image(:,:,3);
397 U = (4*Image(:,:,1))./(Denom + (Denom == 0));
398 V = (9*Image(:,:,2))./(Denom + (Denom == 0));
399 Y = Image(:,:,2)/WhitePoint(2);
400 L = 116*f(Y) - 16;
401 Image(:,:,1) = L;                          % L*
402 Image(:,:,2) = 13*L.*(U - WhitePointU);  % u*
403 Image(:,:,3) = 13*L.*(V - WhitePointV);  % v*
404 return;
405
406
407 function Image = lch(Image,SrcSpace)
408 % Convert to CIE L*ch
409 Image = lab(Image,SrcSpace);  % Convert to CIE L*ab
410 H = atan2(Image(:,:,3),Image(:,:,2));
411 H = H*180/pi + 360*(H < 0);
412 Image(:,:,2) = sqrt(Image(:,:,2).^2 + Image(:,:,3).^2);  % C
413 Image(:,:,3) = H;                                          % H
414 return;
415
416
417 function Image = cat02lms(Image,SrcSpace)
418 % Convert to CAT02 LMS
419 Image = xyz(Image,SrcSpace);
```

```
420 T = [0.7328, 0.4296, -0.1624;-0.7036, 1.6975, 0.0061; 0.0030, 0.0136, 0.9834];
421 X = Image(:,:,1);
422 Y = Image(:,:,2);
423 Z = Image(:,:,3);
424 Image(:,:,1) = T(1)*X + T(4)*Y + T(7)*Z;  % L
425 Image(:,:,2) = T(2)*X + T(5)*Y + T(8)*Z;  % M
426 Image(:,:,3) = T(3)*X + T(6)*Y + T(9)*Z;  % S
427 return;
428
429
430 function Image = huetorgb(m0,m2,H)
431 % Convert HSV or HSL hue to RGB
432 N = size(H);
433 H = min(max(H(:),0),360)/60;
434 m0 = m0(:);
435 m2 = m2(:);
436 F = H - round(H/2)*2;
437 M = [m0, m0 + (m2-m0).*abs(F), m2];
438 Num = length(m0);
439 j = [2 1 0;1 2 0;0 2 1;0 1 2;1 0 2;2 0 1;2 1 0]*Num;
440 k = floor(H) + 1;
441 Image = reshape([M(j(k,1)+(1:Num).'),M(j(k,2)+(1:Num).'),M(j(k,3)+(1:Num).')
      ],[N,3]);
442 return;
443
444
445 function H = rgbtohue(Image)
446 % Convert RGB to HSV or HSL hue
447 [M,i] = sort(Image,3);
448 i = i(:,:,3);
449 Delta = M(:,:,3) - M(:,:,1);
450 Delta = Delta + (Delta == 0);
451 R = Image(:,:,1);
452 G = Image(:,:,2);
453 B = Image(:,:,3);
454 H = zeros(size(R));
455 k = (i == 1);
456 H(k) = (G(k) - B(k))./Delta(k);
457 k = (i == 2);
458 H(k) = 2 + (B(k) - R(k))./Delta(k);
459 k = (i == 3);
460 H(k) = 4 + (R(k) - G(k))./Delta(k);
461 H = 60*H + 360*(H < 0);
462 H(Delta == 0) = nan;
463 return;
464
465
```

```
function Rp = gammacorrection(R)
Rp = zeros(size(R));
i = (R <= 0.0031306684425005883);
Rp(i) = 12.92*R(i);
Rp(~i) = real(1.055*R(~i).^0.416666666666666667 - 0.055);
return;


function R = invgammacorrection(Rp)
R = zeros(size(Rp));
i = (Rp <= 0.0404482362771076);
R(i) = Rp(i)/12.92;
R(~i) = real(((Rp(~i) + 0.055)/1.055).^2.4);
return;


function fY = f(Y)
fY = real(Y.^(1/3));
i = (Y < 0.008856);
fY(i) = Y(i)*(841/108) + (4/29);
return;


function Y = invf(fY)
Y = fY.^3;
i = (Y < 0.008856);
Y(i) = (fY(i) - 4/29)*(108/841);
return;
```

demean.m

```
function [x]=demean(y)
T=size(y,1);
my=repmat(nanmean(y),T,1);
x=(y-my);
```

directmethods.m

```
function [dm] = directmethods(y,lags,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'directmethods' generates local projeciton IRF and direct forecasts
```

```
% Core Inputs:
%- y, data columns variables
%- lags, lag order of the VAR

% Additonal Inputs collected options: see below

% Filippo Ferroni, 27/02/2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2
    error('the BVAR toolbox needs at least two inputs: data and number of lags');
end
if lags < 1
    error('lags cannot be zero or negative');
end
% number of observable variables
[T,ny]       = size(y);

%*****************************************************
%* DEFAULT SETTINGS
%*****************************************************
% Control random number generator
if isOctave == 0
    isMatlab = 1;
    rng('default');
    rng(999);
else
    isMatlab = 0;
    % pkg load optim
     pkg load statistics
    randn('state',999);
    rand('state',999);
end

hor          = 24;
conf_sig     = 0.9;
controls_    = 0;
robust_se_   = 1; % by default robust SE
proxy_       = 0;
noconstant   = 0;
ns           = 0;
Q            = eye(ny);
dummy        = 0;
K            = 5000;
```

```
52 max_prior_tau_ = 0;
53 lb              = -1e10;
54 ub              = 1e10;
55
56
57 %******************************************************
58 %* CUSTOMIZED SETTINGS
59 %******************************************************
60 if nargin>2
61     %=========================================================================
62     % Various options
63     %=========================================================================
64     if isfield(options,'hor') ==1 % horizon of IRF and Forecasts
65         hor = options.hor;
66     end
67     if isfield(options,'conf_sig') ==1 % CI of OLS estimation of LP and DF
68         conf_sig = options.conf_sig;
69     end
70     if isfield(options,'Q') == 1 % orthonormal matrix
71         Q = options.Q;
72     end
73     if isfield(options,'K') == 1 % # of draws for the BLP
74         K = options.K;
75     end
76     %=========================================================================
77     % options: adding controls
78     %=========================================================================
79     if isfield(options,'controls') ==1
80         controls_  = 1;
81         controls   = options.controls;
82         if T~=size(controls,1)
83             error('Control variables and observables must have the same time
                  length')
84         end
85     end
86     %=========================================================================
87     % options: adding proxy variable for identification
88     %=========================================================================
89     if isfield(options,'proxy') ==1
90         proxy_  = 1;
91         proxy   = options.proxy;
92         ns      = size(proxy,2);
93         if T~=size(proxy,1)
94             error('Shocks proxies and observables must have the same time
                  length')
95         end
96     end
```

```
97      %=========================================================================
98      % options: Compuing Robust Standard Errors
99      %=========================================================================
100     if isfield(options,'robust_se_') == 1
101         robust_se_ = options.robust_se_;
102         % robust_se_ = 0     unadjusted SE
103         % robust_se_ = 1     NW Robust SE:  Hamilton (1994), Ch 10 pag 282, eq
                (10.5.20)
104         % robust_se_ = 5     Matlab HAC function: need Matlab Econ Toolbox
105     end
106     %=========================================================================
107     % Conjugate/Hierachical prior options
108     %=========================================================================
109     if (isfield(options,'priors')==1 && strcmp(options.priors.name,'Conjugate'
            )==1) || (isfield(options,'priors')==1 && strcmp(options.priors.name,'
            conjugate')==1) || ...
110             (isfield(options,'prior')==1 && strcmp(options.prior.name,'
                    Conjugate')==1) || (isfield(options,'prior')==1 && strcmp(
                    options.prior.name,'conjugate')==1)
111
112         dummy = 2;
113         prior.name= 'Conjugate';
114         % Priors for the AR parameters
115         if isfield(options.priors,'Phi') == 1
116             % mean
117             if isfield(options.priors.Phi,'mean') == 1
118                 prior.Phi.mean  = options.priors.Phi.mean;
119                 if size(prior.Phi.mean) ~= [ny*lags+(1-noconstant)    ny]
120                     error('Size mismatch')
121                 end
122             else
123                 warning(['You did not provide a prior mean for the AR coeff. '
                            ...
124                     'Assume zeros everywhere.'])
125                 prior.Phi.mean  = zeros(ny*lags+(1-noconstant) , ny);
126             end
127             % variance
128             if isfield(options.priors.Phi,'cov') == 1
129                 prior.Phi.cov   = options.priors.Phi.cov;
130                 if length(prior.Phi.cov) ~= (ny*lags+(1-noconstant) ) || size(
                        prior.Phi.cov,1 )~=size(prior.Phi.cov,2)
131                     error('Size mismatch: Covariance Phi should be square, e.g
                            . size(Phi.mean,1)x size(Phi.mean,1)x')
132                 end
133             else
134                 warning(['You did not provide a Covariance for the AR coeff. '
                            ...
```

```
135                     'Assume 10 times Identity Matrix.'])
136                 prior.Phi.cov  = 10 * eye((ny*lags+(1-noconstant) ));
137             end
138         else
139             warning(['You did not provide prior mean and covariance for the AR
                    coeff ' ...
140                 'Assume zeros everywhere with covariance 10 times Identity
                        Matrix.'])
141             %             prior.Phi.cov    = 10 * eye((ny*lags+(1-noconstant) +
                    timetrend ) * ny);
142             prior.Phi.cov    = 10 * eye((ny*lags+(1-noconstant)  ));
143             prior.Phi.mean   = zeros(ny*lags+(1-noconstant) ,  ny);
144         end
145         % Priors for the Residual Covariance
146         if isfield(options.priors,'Sigma') == 1
147             % scale
148             if isfield(options.priors.Sigma,'scale') == 1
149                 prior.Sigma.scale = options.priors.Sigma.scale;
150                 if size(prior.Sigma.scale) ~= [ny ny]
151                     error('Size mismatch')
152                 end
153             else
154                 warning(['You did not provide a prior mean for the Residual
                        Covariance. ' ...
155                     'Assume identity matrix.'])
156                 prior.Sigma.scale = eye(ny);
157             end
158             % degrees of freedom
159             if isfield(options.priors.Sigma,'df') == 1
160                 prior.Sigma.df = options.priors.Sigma.df;
161                 if length(prior.Sigma.df) ~= 1
162                     error('Size mismatch')
163                 end
164                 if prior.Sigma.df < ny
165                     error('Too few degrees of freedom - prior on variance
                            residuals')
166                 end
167             else
168                 warning(['You did not provide the degrees of freedom for the
                        Residual Covariance. ' ...
169                     'Assume ny+1.'])
170                 prior.Sigma.df = ny + 1;
171             end
172         else
173             warning(['You did not provide prior mean and variance for the
                    Residual Covariance. ' ...
174                 'Assume an identity matrix matrix with N+1 degrees of freedom.
```

```
                    '])
175             prior.Sigma.scale = eye(ny);
176             prior.Sigma.df    = ny + 1;
177         end
178         if isfield(options.priors,'tau') == 1 && isnumeric(options.priors.tau)
                ==  1
179             prior.tau = options.priors.tau;
180             if length(options.priors.tau) ~= hor
181                 error('tau must be a vector of size hor')
182             end
183         elseif isfield(options.priors,'max_tau') == 1 && options.priors.
                max_tau ==1
184             max_prior_tau_ = 1;
185             max_compute    = 3;
186             prior.tau = ones(hor,1);
187         else
188             prior.tau = ones(hor,1);
189         end
190     end
191     % options for the maximization
192     if isfield(options,'max_compute') == 1
193         max_compute    = options.max_compute;
194     end
195     if isfield(options,'ub') == 1
196         ub    = options.ub;
197         if length(ub) ~=  length(prior.tau)
198             error('Mismatch between the size of upper bounds and the param
                    vector');
199         end
200     end
201     if isfield(options,'lb') == 1
202         lb    = options.lb;
203         if length(lb) ~=  length(prior.tau)
204             error('Mismatch between the size of lower bounds and the param
                    vector');
205         end
206     end
207 end
208
209 % Confidence Interval Points and Index for confidence serts
210 alpha  = 1 - conf_sig;
211 talpha = abs(tinv(alpha/2,T-ny-1));
212 sort_idx   = round((0.5 + [-conf_sig, conf_sig, 0]/2) * K);
213
214 %************************************************
215 % Construct the RHS matrix
216 %************************************************
```

```
217
218 X1 = lagX(y,1:lags);
219 positions_nylags = 1  : size(X1,2);
220 nylags           = length(positions_nylags);
221
222 % forecast launching point
223 fdata_initval            = lagX(y(end-lags+1:end, :),0:lags-1);
224 fdata_initval(1:end-1,:) = [];
225 % add controls if any
226 if controls_ == 1
227     a  = size(X1,2) + 1;
228     X1 = [X1 lagX(controls,1:lags)];
229     b  = size(X1,2);
230     position_controls = a:b;
231     % forecast launching point
232     lastvalc = lagX(controls(end-lags+1:end, :),0:lags-1);
233     fdata_initval  = [fdata_initval, controls(end,:)];
234
235 else
236     position_controls = [];
237 end
238 % add proxy shocks if any
239 if proxy_ == 1
240     a     = size(X1,2) + 1;
241     X1    = [X1 proxy];
242     b     = size(X1,2);
243     position_proxy = a:b ;
244     % forecast launching point (assume shocks are zero)
245     fdata_initval  = [fdata_initval,  zeros(1,ns)];
246 else
247     position_proxy = [];
248 end
249 X_ = X1;
250 if noconstant == 0
251     X_ = [X1 ones(T,1)];
252     position_constant = size(X_,2);
253     % forecast launching point
254     fdata_initval  = [fdata_initval, 1];
255 else
256     position_constant = [];
257 end
258 nx           = 1 - noconstant;
259 % other checks
260 if (lags + hor) >= size(X_,1)
261     error('More parameters than observations: consider reducing ''hor'' or ''
        lags''')
262 end
```

```
263
264
265 %************************************************
266 % pre allocation
267 %************************************************
268 Omegaproxy  = eye(ny);
269 Omega       = eye(ny);
270 ir_lp       = nan(ny,hor+1,ny,3);  % variable, horizon, shock and mean upper
        lower
271 irproxy_lp  = nan(ny,hor+1,1,3);
272 forecasts   = nan(hor,ny,3);
273 if dummy == 2
274     ir_blp                  = nan(ny,hor+1,ny,K);
275     irproxy_blp             = nan(ny,hor+1,1,K);
276     bforecasts_no_shocks    = nan(hor,ny,K);
277     bforecasts_with_shocks  = nan(hor,ny,K);
278     log_dnsty               = nan(hor,1);
279 end
280
281 %************************************************
282 %* Computing the LP and DF
283 %************************************************
284 wb = waitbar(0, 'Direct_Methods');
285 for hh = 0 : hor % iteration over horizon
286     ytmp = lagX(y, -hh);
287     % Reduced Form estimations
288     if robust_se_ ~= 0 % Robust SE
289         %  options.robust_se_ = robust_se_;
290         options.L     = lags + hh + 1;
291         olsreg_(hh+1) = ols_reg(ytmp, X_, options);
292     else
293         olsreg_(hh+1) = ols_reg(ytmp, X_);
294     end
295     % Proxy IV identifications
296     if proxy_
297         irproxy_lp(:, hh+1, :, 2) = olsreg_(hh+1).beta(position_proxy, :)'; %
            mean
298         irproxy_lp(:, hh+1, :, 3) = irproxy_lp(:, hh+1, :, 2) + talpha *
            olsreg_(hh+1).se(position_proxy, :)'; % upper
299         irproxy_lp(:, hh+1, :, 1) = irproxy_lp(:, hh+1, :, 2) - talpha *
            olsreg_(hh+1).se(position_proxy, :)'; % lower
300         if hh == 0
301             Omegaproxy(:,1) = olsreg_(hh+1).beta(position_proxy, :)';
302             Omegasproxy(:,1,:) =  repmat(Omegaproxy(:,1), 1, 1, K) + ...
303                 olsreg_(hh+1).se(position_proxy, :)'.* randn(ny,1,K);
304         end
305     end
```

```
306     % Choleski/Rotated identification
307     if hh == 0
308         Omega = chol(olsreg_(hh+1).Serror,'Lower') * Q;
309         % generate uncertainty on S (flat prior)
310         Sbar = olsreg_(hh+1).error'*olsreg_(hh+1).error;
311         df   = olsreg_(hh+1).N - olsreg_(hh+1).K;
312         [~, Omegas] = generateOmegas(Sbar,df,K,Q);
313         Omegasort = sort(Omegas,3);
314
315         ir_lp(:, hh+1, :, 2) = Omega; % mean
316         ir_lp(:, hh+1, :, 3) = Omegasort(:,:,sort_idx(2)); % UPPER
317         ir_lp(:, hh+1, :, 1) = Omegasort(:,:,sort_idx(1)); % LOWER
318     else
319         [ir] =  iresponse(olsreg_(hh).beta(positions_nylags, :), eye(ny) , 2,
                Omega);
320         ir_lp(:, hh+1, :, 2) = ir(:, 2, :);  % mean
321         [ir3] =  iresponse(olsreg_(hh).beta(positions_nylags, :) + talpha *
                olsreg_(hh).se(positions_nylags, :), eye(ny) , 2, Omega);
322         ir_lp(:, hh+1, :, 3) = ir3(:, 2, :); % upper
323         [ir1] =  iresponse(olsreg_(hh).beta(positions_nylags, :) - talpha *
                olsreg_(hh).se(positions_nylags, :), eye(ny) , 2, Omega);
324         ir_lp(:, hh+1, :, 1) = ir1(:, 2, :); % lower
325     end
326
327     % forecast part
328     forecasts(hh+1, :, 2) = (fdata_initval * olsreg_(hh+1).beta);
329     forecasts(hh+1, :, 3) = forecasts(hh+1, :, 2) + talpha * diag(olsreg_(hh
            +1).Serror)' ;
330     forecasts(hh+1, :, 1) = forecasts(hh+1, :, 2) - talpha * diag(olsreg_(hh
            +1).Serror)' ;
331
332     %=========================================================================
333     % Baysian DM
334     if dummy == 2 % activating Bayesian Direct methods.
335         if hh == 0
336             % cholesky IRF on impact
337             ir_blp(:, hh+1, :, :)  = Omegas;
338             % proxy IRF on impact
339             if proxy_
340                 irproxy_blp(:, hh+1, :, :) = Omegasproxy;
341             end
342             % one-step ahead forecast
343             bforecasts_no_shocks(hh+1, :, :)    = repmat(forecasts(hh+1, :, 2)
                    ,1,1,K);
344             fnoise = Omega * randn(ny,K);
345             bforecasts_with_shocks(hh+1, :, :) = bforecasts_no_shocks(hh+1, :,
                     :) + reshape(fnoise,1,ny,K);
```

```
346
347             % computing the companion matrix
348             F       = [prior.Phi.mean(1 : ny * lags, :)'; eye(ny*(lags-1), ny*
                    lags)];
349             % constant
350             Fo        = [prior.Phi.mean(end, :)'; zeros(ny * (lags-1), 1)];
351             % Shocks Companion
352             G       = eye(ny * lags, ny);
353
354         else % hh > 0
355             %******************************************************
356             % Conjugate Prior: MN-IW
357             %******************************************************
358             if max_prior_tau_ == 1 % Maximize the shrinkage on the VAR
                    coefficients
359                 try
360                     disp(['**********************************************'])
361                     disp(['**********************************************'])
362                     disp(['Optimization_at_horizon_' num2str(hh)])
363                     x0   = log(prior.tau(hh));
364                     switch max_compute
365                         case 1
366                             optim_options = optimset('display','iter','
                                MaxFunEvals',100000,'TolFun',1e-8,'TolX',1e-6)
                                ;
367                             [xh,fh,~,~,~,~] = ...
368                                 fminunc('blp_opt_hyperpara',x0,optim_options
                                    ,...
369                                 hh,prior,olsreg_(hh),F,G,Fo,positions_nylags,
                                    position_constant);
370                             %
                                ================================================================

371                         case 2 % constraint
372                             % Set default optimization options for fmincon.
373                             optim_options = optimset('display','iter', '
                                LargeScale','off', 'MaxFunEvals',100000, '
                                TolFun',1e-8, 'TolX',1e-6);
374                             [xh,fh,~,~,~,~,~] = ...
375                                 fmincon('blp_opt_hyperpara',x0,[],[],[],[],lb(
                                    hh),ub(hh),[],optim_options,y,lags,options
                                    );
376                             %
                                ================================================================

377                         case 3 % Sims
378                             crit = 10e-5;
```

```
379                         nit   = 10e-4;
380                         [fh, xh, ~, ~, ~, ~, ~] = ...
381                             csminwel('blp_opt_hyperpara',x0,.1*eye(length(
                                    x0)),[],crit,nit,hh,prior,olsreg_(hh),F,G,
                                    Fo,positions_nylags,position_constant);
382                         %
                                ================================================================

383                     case 7 % Matlab's simplex (Optimization toolbox needed
                            ).
384                         optim_options = optimset('display','iter','
                                MaxFunEvals',30000,'MaxIter',10000,'TolFun',1e
                                -3,'TolX',1e-3);
385                         [xh,fh,~,~] = fminsearch('blp_opt_hyperpara',x0,
                                optim_options,y,lags,options);
386                 end
387                 fprintf('%s = %0.5g\n','Hyper-parameter Mode ',exp(xh))
388                 fprintf('%s = %0.5g\n','Marginal Likelihood ',-fh)
389             catch
390                 warning('Maximization NOT Successful')
391                 disp('Using hyper parameter default values')
392                 xh = x0;
393             end
394             prior.tau(hh) = exp(xh);
395             disp(['**********************************************'])
396         end
397         % constructing the posterior moments given the (optimal) shrinkage
398         [posterior_, ~]    = p2p(hh,prior.tau(hh),prior,olsreg_(hh),F,G,Fo,
                positions_nylags,position_constant);
399         % computing the marginal likelihood
400         log_dnsty(hh)      = blp_ml(prior.tau(hh),hh,prior,olsreg_(hh),F,G,
                Fo,positions_nylags,position_constant);
401         % Second moments
402         S_inv_upper_chol   = chol(inv(posterior_.S));
403         XXi_lower_chol     = chol(posterior_.XXi)';
404         % number of regressors
405         nk = nylags + nx;
406         %************************************************
407         %* Generating draws form the Posterior Distribution
408         %************************************************
409         for  d =  1 : K %  Gibbs Sampler
410             %
                    ========================================================================

411             % Inferece: Drawing from the posterior distribution
412             % Step 1: draw from the Covariance
413             Sigma = rand_inverse_wishart(ny, posterior_.df,
```

```
                    S_inv_upper_chol);
414
415                 % Step 2: given the Covariance Matrix, draw from the AR
                        parameters
416                 Sigma_lower_chol = chol(Sigma)';
417                 Phi1 = randn(nk * ny, 1);
418                 Phi2 = kron(Sigma_lower_chol , XXi_lower_chol) * Phi1;
419                 Phi3 = reshape(Phi2, nk, ny);
420                 Phi  = Phi3 + posterior_.PhiHat;
421
422                 % Step 3: compute IRF
423                 blp  =  iresponse(Phi, eye(ny) , 2, Omega);
424                 ir_blp(:, hh+1, :, d) = blp(:, 2, :);  % mean
425                 if proxy_
426                     blpproxy  =  iresponse(Phi, eye(ny) , 2, Omegaproxy);
427                     irproxy_blp(:, hh+1, :, d) = blpproxy(:,2,1);
428                 end
429                 % Step 4: compute Forecasts
430                 bforecasts_no_shocks(hh+1, :, d) = (fdata_initval(1, [
                        positions_nylags position_constant]) * Phi);
431                 bforecasts_with_shocks(hh+1, :, d) = (fdata_initval(1, [
                        positions_nylags position_constant]) * Phi) + (
                        Sigma_lower_chol * randn(ny,1))';
432             end
433         end
434     end
435     waitbar(hh/hor, wb);
436 end
437 close(wb);
438
439 %*********************************************************
440 %* Storing the resutls
441 %*********************************************************
442 % dm.olsreg_      = olsreg_;
443 dm.forecasts   = forecasts;
444 dm.ir_lp       = ir_lp;
445 dm.irproxy_lp  = irproxy_lp;
446 if dummy == 2
447     dm.ir_blp                 = ir_blp;
448     dm.irproxy_blp            = irproxy_blp;
449     dm.bforecasts.no_shocks   = bforecasts_no_shocks;         % trajectories
            of forecasts without shocks
450     dm.bforecasts.with_shocks = bforecasts_with_shocks;       % trajectories
            of forecasts with shocks
451     dm.logmlike               = log_dnsty;
452     dm.prior                  = prior;
453 else
```

```
454     dm.ir_blp                  = [];
455     dm.irproxy_blp             = [];
456     dm.bforecasts.no_shocks    = [];
457     dm.bforecasts.with_shocks  = [];
458     dm.logmlike                = [];
459     dm.prior                   = [];
460 end
461
462
463
464 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
465 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
466 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
467 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
468 function [S,Omegas] = generateOmegas(Sbar,df,K,Q)
469 % generate uncertainty on S
470 ny      = size(Sbar,1);
471 S       = nan(ny,ny,K);
472 Omegas  = nan(ny,ny,K);
473
474 S_inv_upper_chol = chol(inv(Sbar));
475
476 for  d = 1:K
477     S(:,:,d)      = rand_inverse_wishart(ny, df, S_inv_upper_chol);
478     Omegas(:,:,d) = chol(S(:,:,d),'Lower') * Q;
479 end
```

distinguishable_colors.m

```
1  function colors = distinguishable_colors(n_colors,bg,func)
2  % DISTINGUISHABLE_COLORS: pick colors that are maximally perceptually distinct
3  %
4  % When plotting a set of lines, you may want to distinguish them by color.
5  % By default, Matlab chooses a small set of colors and cycles among them,
6  % and so if you have more than a few lines there will be confusion about
7  % which line is which. To fix this problem, one would want to be able to
8  % pick a much larger set of distinct colors, where the number of colors
9  % equals or exceeds the number of lines you want to plot. Because our
10 % ability to distinguish among colors has limits, one should choose these
11 % colors to be "maximally perceptually distinguishable."
12 %
13 % This function generates a set of colors which are distinguishable
14 % by reference to the "Lab" color space, which more closely matches
15 % human color perception than RGB. Given an initial large list of possible
16 % colors, it iteratively chooses the entry in the list that is farthest (in
```

```
17 % Lab space) from all previously-chosen entries. While this "greedy"
18 % algorithm does not yield a global maximum, it is simple and efficient.
19 % Moreover, the sequence of colors is consistent no matter how many you
20 % request, which facilitates the users' ability to learn the color order
21 % and avoids major changes in the appearance of plots when adding or
22 % removing lines.
23 %
24 % Syntax:
25 %    colors = distinguishable_colors(n_colors)
26 % Specify the number of colors you want as a scalar, n_colors. This will
27 % generate an n_colors-by-3 matrix, each row representing an RGB
28 % color triple. If you don't precisely know how many you will need in
29 % advance, there is no harm (other than execution time) in specifying
30 % slightly more than you think you will need.
31 %
32 %    colors = distinguishable_colors(n_colors,bg)
33 % This syntax allows you to specify the background color, to make sure that
34 % your colors are also distinguishable from the background. Default value
35 % is white. bg may be specified as an RGB triple or as one of the standard
36 % "ColorSpec" strings. You can even specify multiple colors:
37 %      bg = {'w','k'}
38 % or
39 %      bg = [1 1 1; 0 0 0]
40 % will only produce colors that are distinguishable from both white and
41 % black.
42 %
43 %    colors = distinguishable_colors(n_colors,bg,rgb2labfunc)
44 % By default, distinguishable_colors uses the image processing toolbox's
45 % color conversion functions makecform and applycform. Alternatively, you
46 % can supply your own color conversion function.
47 %
48 % Example:
49 %    c = distinguishable_colors(25);
50 %    figure
51 %    image(reshape(c,[1 size(c)]))
52 %
53 % Example using the file exchange's 'colorspace':
54 %    func = @(x) colorspace('RGB->Lab',x);
55 %    c = distinguishable_colors(25,'w',func);
56
57 % Copyright 2010-2011 by Timothy E. Holy
58
59 % Parse the inputs
60 if (nargin < 2)
61     bg = [1 1 1];  % default white background
62 else
63     if iscell(bg)
```

```
64          % User specified a list of colors as a cell aray
65          bgc = bg;
66          for i = 1:length(bgc)
67              bgc{i} = parsecolor(bgc{i});
68          end
69          bg = cat(1,bgc{:});
70      else
71          % User specified a numeric array of colors (n-by-3)
72          bg = parsecolor(bg);
73      end
74  end
75
76  % Generate a sizable number of RGB triples. This represents our space of
77  % possible choices. By starting in RGB space, we ensure that all of the
78  % colors can be generated by the monitor.
79  n_grid = 30;  % number of grid divisions along each axis in RGB space
80  x = linspace(0,1,n_grid);
81  [R,G,B] = ndgrid(x,x,x);
82  rgb = [R(:) G(:) B(:)];
83  if (n_colors > size(rgb,1)/3)
84      error('You can''t readily distinguish that many colors');
85  end
86
87  % Convert to Lab color space, which more closely represents human
88  % perception
89  if (nargin > 2)
90      lab = func(rgb);
91      bglab = func(bg);
92  else
93      C = makecform('srgb2lab');
94      lab = applycform(rgb,C);
95      bglab = applycform(bg,C);
96  end
97
98  % If the user specified multiple background colors, compute distances
99  % from the candidate colors to the background colors
100 mindist2 = inf(size(rgb,1),1);
101 for i = 1:size(bglab,1)-1
102     dX = bsxfun(@minus,lab,bglab(i,:)); % displacement all colors from bg
103     dist2 = sum(dX.^2,2);  % square distance
104     mindist2 = min(dist2,mindist2);  % dist2 to closest previously-chosen
            color
105 end
106
107 % Iteratively pick the color that maximizes the distance to the nearest
108 % already-picked color
109 colors = zeros(n_colors,3);
```

```
110 lastlab = bglab(end,:);   % initialize by making the "previous" color equal to
        background
111 for i = 1:n_colors
112     dX = bsxfun(@minus,lab,lastlab); % displacement of last from all colors on
            list
113     dist2 = sum(dX.^2,2);  % square distance
114     mindist2 = min(dist2,mindist2);  % dist2 to closest previously-chosen
            color
115     [~,index] = max(mindist2);  % find the entry farthest from all previously-
            chosen colors
116     colors(i,:) = rgb(index,:);  % save for output
117     lastlab = lab(index,:);  % prepare for next iteration
118 end
119 end
120
121 function c = parsecolor(s)
122 if ischar(s)
123     c = colorstr2rgb(s);
124 elseif isnumeric(s) && size(s,2) == 3
125     c = s;
126 else
127     error('MATLAB:InvalidColorSpec','Color specification cannot be parsed.');
128 end
129 end
130
131 function c = colorstr2rgb(c)
132 % Convert a color string to an RGB value.
133 % This is cribbed from Matlab's whitebg function.
134 % Why don't they make this a stand-alone function?
135 rgbspec = [1 0 0;0 1 0;0 0 1;1 1 1;0 1 1;1 0 1;1 1 0;0 0 0];
136 cspec = 'rgbwcmyk';
137 k = find(cspec==c(1));
138 if isempty(k)
139     error('MATLAB:InvalidColorString','Unknown color string.');
140 end
141 if k~=3 || length(c)==1,
142     c = rgbspec(k,:);
143 elseif length(c)>2,
144     if strcmpi(c(1:3),'bla')
145         c = [0 0 0];
146     elseif strcmpi(c(1:3),'blu')
147         c = [0 0 1];
148     else
149         error('MATLAB:UnknownColorString', 'Unknown color string.');
150     end
151 end
152 end
```

fetchData.m

```
1
2  %% Fetch Data
3  %Fetch data sets up a function so that Haver Data can be imported directly
4  %to Matlab
5
6  %Inputs:
7  % Database - refers to Haver database such as USECON, USNA, or CAPSTOCK; USNA
8  % most commonly used to keep consistency with Jeff's previous chain
       aggregation code
9  % and other BEA numbers.
10 % seriesName - used for the acutal names coming from Haver
11 % startDate and endDate are previously chosen for the range of data you'd like
        to pull
12
13 function [data,timev]=fetchData(database,seriesName,startDate,endDate,
       frequency)
14          D=fetch(database,seriesName,startDate,endDate,frequency);
15          data=D(:,2);
16          timev =D(:,1);
17 end
```

fevd.m

```
1  function FEVD = fevd(hor,Phi,Sigma,Omega)
2
3  % computes the forecast error variance decomposition
4
5  N           = size(Sigma,1);
6  [m , k]     = size(Phi);
7  lags        = (m-1)/k;
8
9  if nargin < 4
10     Omega = eye(N);
11 end
12
13
14 % companion
15 F        = [Phi(1 : N * lags, :)'; eye(N*(lags-1), N*lags)];
16 G        = eye(N * lags, N);
17 %C        = [Phi(end, :)'; zeros(N*(lags-1),1)];
18
19 A      = chol(Sigma,'lower');
20 Kappa = G * A * Omega ;
```

```
21
22 tmp_=0;
23 for hh  = 1 : hor
24     tmp_ = tmp_ + F^(hh-1) * Kappa * Kappa' * F^(hh-1)';
25 end
26 out_.all_var_  =  diag(tmp_(1:N,1:N));
27
28 for sho =1 : N
29     tmp_1 = 0;
30     Ind               = zeros(N);
31     Ind(sho,sho) = 1;
32     for hh  = 1 : hor
33         tmp_1 = tmp_1 + F^(hh-1) * Kappa * Ind * Kappa' * F^(hh-1)';
34
35     end
36     out_.var_(:,sho) = diag(tmp_1(1:N,1:N));
37 end
38
39 if max(max(abs( sum(out_.var_,2) - out_.all_var_))) > 1e-10
40     error('Something went wrong')
41 end
42
43 for indx_sho = 1 : N
44     FEVD(:,indx_sho) = out_.var_(:,indx_sho)./out_.all_var_*100;
45
46 end
```

findP.m

```
1  % Copyright Andrew Binning 2013
2  % Please feel free to use and modify this code as you see if fit. If you
3  % use this code in any academic work, please cite
4  % Andrew Binning, 2013.
5  % "Underidentified SVAR models: A framework for combining short and long-run
       restrictions with sign-restrictions,"
6  % Working Paper 2013/14, Norges Bank.
7  function P = findP(C,B,Q,p,k,index)
8  %=========================================================================
9  % finds the rotation matrix that satisfies the short and long run
10 % restrictions.  Based on Juan F. Rubio-Ramirez & Daniel F. Waggoner & Tao Zha
       , 2010.
11 % "Structural Vector Autoregressions: Theory of Identification and
12 % Algorithms for Inference," Review of Economic Studies, Oxford University
13 % Press, vol. 77(2), pages 665-696.
14 %
```

```
% inputs:
% C = initial short run impact matrix, usually from a cholesky
% decomposition of the forecast error variance
% B = Matrix of coefficients (including intercept estimates)
% Q = A cell containing the linear restrictions for each columnn
% p = number of lags
% k = number of dependent variables
% index = original column ordering in the matrix of restrictions
%
% outputs:
% P = orthogonal rotation matrix
%===========================================================================

L0 = C;

beta_temp = B(2:end,:)';

beta = zeros(k,k);

for ii = 1:p

    beta = beta + beta_temp(:,(1:k)+(ii-1)*k);

end

Linf = (eye(k)-beta)\C;

F = [L0;Linf];

P = zeros(k,k);

for ii = 1:k
    if ii == 1
        Qtilde = Q{ii}*F;
    else
        Qtilde = [Q{ii}*F;P'];
    end
    [QQ,RR] = qr(Qtilde');
    P_temp = QQ(:,end);
    P(:,ii) = P_temp;

end

P = P(:,index);
```

findQs.m

```
% Copyright Andrew Binning 2013
% Please feel free to use and modify this code as you see if fit. If you
% use this code in any academic work, please cite
% Andrew Binning, 2013.
% "Underidentified SVAR models: A framework for combining short and long-run
    restrictions with sign-restrictions,"
% Working Paper 2013/14, Norges Bank.
function [Q,index,flag] = findQs(k,f)
%=========================================================================
% finds the Q matrices that describe the linear restrictions on the shock
% impact matrix. Based on Juan F. Rubio-Ramirez & Daniel F. Waggoner & Tao Zha
    , 2010.
% "Structural Vector Autoregressions: Theory of Identification and
% Algorithms for Inference," Review of Economic Studies, Oxford University
% Press, vol. 77(2), pages 665-696.
%
% inputs:
% k = number of dependent variables
% f = matrix of short and long run restrictions
%
% outputs:
% Q = a cell that contains the linear restrictions for each equation
% index = the original column order of the matrix of restrictions
% flag = indicates whether the model is over, under or exactly identified
%=========================================================================
E = eye(k);

Q_init = cell(k,2);

for ii = 1:k

    Q_init{ii,1} = double(diag(f*E(:,ii)==0));
    Q_init{ii,2} = rank(Q_init{ii,1});

end

for ii = 1:k

    temp = Q_init{ii,1};
    Q_init{ii,1} = temp(logical(sum(temp,2)),:);

end

[new,ord] = sort([Q_init{:,2}],2,'descend');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check identification

if any(new - (k - (1:k)) > 0)  % over-identified
    flag = 1;
elseif all(new - (k - (1:k)) == 0) % exactly identified
    flag = 0;
elseif any(new - (k - (1:k)) < 0) % under-identified
    flag = -1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

index = nan(k,1);

for ii = 1:k
    index(ord(ii)) = ii;
end

Q = cell(k,1);

for ii = 1:k

    Q{ii} = Q_init{ord(ii),1};

end
```

forecasts.m

```
function [frcst_no_shock,frcst_with_shocks]=forecasts(forecast_data,Phi,Sigma,
    fhor,lags,EPS)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'forecasts' computes out-of-sample forecasts

% Inputs:
% - forecast_data, last data
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - fhor, horizon of the out-of-sample forecasts
% - lags
% - EPS, a particular shock realization

% Output:
```

```
% — frcst_no_shock
% 1st dimension:   horizon
% 2nd dimension:   variable

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ny = size(Sigma,1);
if nargin < 6
    shock_given = 0;
else
    shock_given = 1;
end

Sigma_lower_chol = chol(Sigma)';

% preallocating memory
frcst_no_shock     = nan(fhor,ny);
frcst_with_shocks  = nan(fhor,ny);

lags_data = forecast_data.initval;
for t = 1 : fhor
    X = [ reshape(flip(lags_data, 1)', 1, ny*lags) forecast_data.xdata(t, :)
        ];
    y = X * Phi;
    lags_data(1:end-1,:) = lags_data(2:end, :);
    lags_data(end,:) = y;
    frcst_no_shock(t, :) = y;
end

% With shocks
lags_data = forecast_data.initval;
for t = 1 : fhor
    X = [ reshape(flip(lags_data, 1)', 1, ny*lags) forecast_data.xdata(t, :)
        ];
    shock = (Sigma_lower_chol * randn(ny, 1))';
    if shock_given == 1
        shock  = EPS(t,:);
    end
    y = X * Phi + shock;
    lags_data(1:end-1,:) = lags_data(2:end, :);
    lags_data(end,:) = y;
    frcst_with_shocks(t, :) = y;
end
```

generateDraw.m

```
% Copyright Andrew Binning 2013
% Please feel free to use and modify this code as you see if fit. If you
% use this code in any academic work, please cite
% Andrew Binning, 2013.
% "Underidentified SVAR models: A framework for combining short and long-run
    restrictions with sign-restrictions,"
% Working Paper 2013/14, Norges Bank.
function C = generateDraw(C,k)
%=========================================================================
% Generates a draw that is consistent with the shock variance/covariance
% matrix. Based on Juan F. Rubio-Ramirez & Daniel F. Waggoner & Tao Zha, 2010.
% "Structural Vector Autoregressions: Theory of Identification and
% Algorithms for Inference," Review of Economic Studies, Oxford University
    Press, vol. 77(2), pages 665-696.
%
% inputs:
% C = initial impact matrix, usually from the cholesky decomposition of the
% forecast error variance decomposition
% k = number of dependent variables
%
% outputs:
% C = new draw of the short run impact matrix
%=========================================================================

newmatrix = randn(k,k);

[Q,R] = qr(newmatrix);

for ii = 1:k
    if R(ii,ii)<0
        Q(:,ii) = -Q(:,ii);
    end
end

C = C*Q;
```

generateQ.m

```
function Omega = generateQ(m)

% generate an orthonormal matrix.

G     = randn(m);
```

```
[Q,R] = qr(G);
% normalize to positive entry in the diagonal
In    = diag(sign(diag(R)));
Omega = Q  * In;
```

histdecomp.m

```
function [histdec,ierror] = histdecomp(bvar_,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'histdecomp' computes decomposition of the observable variables in
% terms of structural VAR shocks and initial condition

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% tolerance
tol =  10^(-7);
% Retrieve the initial condition
yo      = bvar_.XX(1,1:end-1)'; % remove the constant from X
% use posterior means
u       = mean(bvar_.e_draws,3);        % reduced form errors
alpha   = mean(bvar_.Phi_draws,3);      % AR
Sigma   = mean(bvar_.Sigma_draws,3);    % Sigma
% retrieve VAR settings
N            = bvar_.N;
lags         = bvar_.lags;
% no rotation
Omega = eye(N);
% data
data = bvar_.data;


if nargin > 1
    if isfield(options,'tol')==1
        tol = options.tol;
    end
    if isfield(options,'yo')==1 && options.yo == 0 %no initial condition
        yo    = zeros(N*lags,1);
    end
    if isfield(options,'Omega')==1 % use a specific rotation
         Omega = options.Omega;
```

```
39      end
40      if isfield(options,'draw')==1 % use a specific draw for the reduced form
            param
41          draw = options.draw;
42          u      = bvar_.e_draws(:,:,draw);            % reduced form errors
43          alpha  = bvar_.Phi_draws(:,:,draw);          % AR
44          Sigma  = bvar_.Sigma_draws(:,:,draw);        % Sigma
45      end
46      if isfield(options,'median')==1 && options.median == 1 % use median
            instead of mean
47          u      = median(bvar_.e_draws,3);        % reduced form errors
48          alpha  = median(bvar_.Phi_draws,3);      % AR
49          Sigma  = median(bvar_.Sigma_draws,3);    % Sigma
50      end
51  end
52
53  A      = chol(Sigma,'lower');
54  ierror = zeros(length(u),N);
55
56  % (1) e = A * Omega * eta
57  % e = n * 1          %reduced
58  % eta = n * 1        % structural
59  % A = n * n
60  % Omega = n * n
61  % e' =  eta' * A' * Omega';
62  % (2) E  =  ETA  * A' * Omega';
63  % where
64  % H = T * n
65  % E = T * n
66
67  % structural innovations
68  ierror = u * inv( Omega' * A'); %#ok<MINV>
69
70  % companion form
71  F       = [alpha(1 : N * lags, :)'; eye(N*(lags-1), N*lags)];
72  G       = eye(N * lags, N);
73  C       = [alpha(end, :)'; zeros(N*(lags-1),1)];
74  ystar   = zeros(length(u),N*lags,N);
75
76  % Deterministic Part
77  for t = 1 : length(u)
78      Aa =  0 ;
79      for tau = 1 : t
80          Aa = Aa + F^(tau-1)*C;
81      end
82  %     B(t,:) = (Aa + F^(t) * yo)';
83      B(t,:) = (Aa + F^(t) * yo)';
```

```
84 %      initial condition
85      Bb(t,:) = (F^(t) * yo)';
86  end
87
88  Kappa = G * A * Omega ;
89  % Stochastic part
90  for shock = 1 : N
91      Ind               = zeros(N);
92      Ind(shock,shock) = 1;
93      for t = 1 : length(u)
94          D  = 0;
95          for tau = 1 : t
96              D     = D + F^(t-tau) *  Kappa * Ind * ierror(tau,:)';
97          end
98          E(t,:,shock) = D';
99      end
100 end
101
102 % check
103 W       = B + sum(E,3);
104 yhat    = W(:, 1 : N);
105 tmp = max(max(abs(data(lags+1:end,:) - yhat)));
106 if  tmp> tol,
107     histdec = [];
108     warning(['Maximium Discrepancy ' num2str(tmp)])
109     return;
110 end
111
112 histdec             = E(:,1:N,:);
113 histdec(:,:,N+1)    = B(:,1:N);
114
115 end
```

histdecomposition.m

```
1  function [histdec,ierror] = histdecomposition(error,alpha,Sigma,data,Omega,yo,
       tol)
2
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  % 'histdecomposition' computes decomposition of the observable variables in
5  % terms of structural VAR shocks and initial condition
6
7  % Filippo Ferroni, 6/1/2015
8  % Revised, 2/15/2017
9  % Revised, 3/21/2018
```

```
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if nargin < 7
    tol =  10^(-7);
end

N      = size(Sigma,1);
[m , k]      = size(alpha);
lags         = (m-1)/k;

if nargin < 5
    Omega = eye(N);
    yo    = zeros(N*lags,1);
end
if nargin < 6
    yo    = zeros(N*lags,1);
end
%

A       = chol(Sigma,'lower');
ierror = zeros(length(error),N);

% (1) e = A * Omega * eta
% e = n * 1          %reduced
% eta = n * 1        % structural
% A = n * n
% Omega = n * n
% e' =  eta' * A' * Omega';
% (2) E  =  ETA  * A' * Omega';
% where
% H = T * n
% E = T * n



ierror = error * inv( Omega' * A'); %#ok<MINV>


% companion
F        = [alpha(1 : N * lags, :)'; eye(N*(lags-1), N*lags)];
G        = eye(N * lags, N);
C        = [alpha(end, :)'; zeros(N*(lags-1),1)];
ystar    = zeros(length(error),N*lags,N);

% Deterministic Part
```

```
for t = 1 : length(error)
    Aa =  0 ;
    for tau = 1 : t
        Aa = Aa + F^(tau-1)*C;
    end
%     B(t,:) = (Aa + F^(t) * yo)';
    B(t,:) = (Aa + F^(t) * yo)';
%     initial condition
    Bb(t,:) = (F^(t) * yo)';
end

Kappa = G * A * Omega ;
% Stochastic part
for shock = 1 : N
    Ind              = zeros(N);
    Ind(shock,shock) = 1;
    for t = 1 : length(error)
        D  = 0;
        for tau = 1 : t
            D     = D + F^(t-tau) *  Kappa * Ind * ierror(tau,:)';
        end
        E(t,:,shock) = D';
    end
end

% check
W       = B + sum(E,3);
yhat    = W(:, 1 : N);
tmp = max(max(abs(data(lags+1:end,:) - yhat)));
if  tmp> tol,
    histdec = [];
    warning(['Maximium Discrepancy ' num2str(tmp)])
    return;
end

histdec             = E(:,1:N,:);
histdec(:,:,N+1)    = B(:,1:N);

end
```

iresponse.m

```
function [ir]=iresponse(alpha,Sigma,hor,Omega,unit)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% 'iresponse' computes the impulse response functions

% Inputs:
%— alpha, AR parameters of the VAR
%— Sigma, Covariance matrix of the reduced form VAR shocks
%— hor, horizon of the IRF
%— Omega, a particular orthonormal rotation
%— unit, 1 shock STD or 1 percent increase

% Output:
%— ir contains the IRF
% 1st dimension:   variable
% 2st dimension:   horizon
% 3st dimension:   shock

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N            = size(Sigma,1);
ir           = zeros(N,hor,N); % variables, horizon, shock
[m , n]      = size(alpha);
lags         = floor((m—1)/n);
[Q]          = chol(Sigma,'lower');

% units
if nargin < 5
    unit = eye(N);
else
    unit = inv(diag(diag(Q)));
end

% 1 standard deviation increase (if unit = eye(N))
% 100 basis point increase (else)
Q = Q*unit;

% companion form
F        = [alpha(1 : N * lags, :)'; eye(N*(lags—1), N*lags)];
G        = eye(N * lags, N);
Fk       = eye(N * lags);
% compute IRFs
for k=1:hor
    PHI          = Fk * G * Q * Omega;
    ir(1:N,k,:) = G' * PHI;
    Fk           = F * Fk;
```

```
end
```

iresponse_longrun.m

```
function [ir,Q]=iresponse_longrun(alpha,Sigma,hor,lags)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'iresponse_longrun' computes the impulse response functions to a long run
    shock
% ordered first

% Inputs:
% - alpha, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - hor, horizon of the IRF
% - unit, 1 shock STD or 1 percent increase

% Output:
% - ir contains the IRF
% 1st dimension:   variable
% 2st dimension:   horizon
% 3st dimension:   shock

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N             = size(Sigma,1);
ir           = zeros(N,hor,N); % variables, horizon, shock

% companion
F       = [alpha(1 : N * lags, :)'; eye(N*(lags-1), N*lags)];
G       = eye(N * lags, N);
Inp     = eye(size(F));
% long run sum of VMA
C1              = Inp - F;
C1              = C1 \ Inp; % = inv(C1)*Inp;
C1              = C1(1 : N, 1 : N);
PSI1            = chol(C1 * Sigma * C1')';
% IMPACT MATRIX
Q               = (C1 \ eye(size(C1))) * PSI1;
% normalize to positive entry in the element Q(1,1)
Q(1,1)     = Q(1,1) * sign(Q(1,1));
```

```
Fk          = eye(N * lags);

for k=1:hor
    PHI          = Fk * G * Q;
    ir(1:N,k,:) = G'* PHI;
    Fk           = F * Fk;
end
```

iresponse_proxy.m

```
function in = iresponse_proxy(in)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'iresponse_proxy' computes the impulse response functions to shock
% identified via instrumental variables
% Codes are written based on the Mertens Ravn (2013, AER) source codes,
% modified to
%1) handle different length of VAR and factors
%2) Multiple instruments to explain the same shock

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Y       = in.vars(in.p+1:end,:);
[T,n]    = size(Y);
[T_m,~] = size(in.proxies);

% number of proxies
k  = 1;
%Assuming proxies start at least p periods later
instrument  = in.proxies(1:end,:);

% Identification
%%%%%%%%%%%%%%%%

% covariance of the reduced form shocks
in.Sigma_m = in.Sigma;

% Instrument on VAR residuals
Phib = [ones(T_m,1) instrument]\in.res(T-T_m-in.T_m_end+1:T-in.T_m_end,:);

% Fitted values of the identified shocks
```

```
% here is where thes prior on beta should enter , instead of Phib(:,1)
% m = beta e1 + v
uhat1           =   [ones(T_m,1) instrument]*Phib(:,1);

% regress the fitted values on the other reduced form shocks
b21ib11_TSLS    =   [ones(T_m,1) uhat1]\in.res(T-T_m-in.T_m_end+1:T-in.T_m_end
    ,2:end);
b21ib11_TSLS    =   b21ib11_TSLS(2:end,:)';
b21ib11         =   b21ib11_TSLS;

% Identification of b11 and b12 from the covariance matrix of the VAR
Sig11   = in.Sigma_m(1:k,1:k);
Sig21   = in.Sigma_m(k+1:n,1:k);
Sig22   = in.Sigma_m(k+1:n,k+1:n);
ZZp     = b21ib11*Sig11*b21ib11'-(Sig21*b21ib11'+b21ib11*Sig21')+Sig22;
b12b12p = (Sig21- b21ib11*Sig11)'*(ZZp\(Sig21- b21ib11*Sig11));
b11b11p = Sig11-b12b12p;
b11     = sqrt(b11b11p);
in.b1   = [b11; b21ib11*b11];
in.Phib = Phib;

% Impulse Responses
%%%%%%%%%%%%%%%%%%%%%%
% initial shock: eps(1,1)=1
irs(in.p+1,:) = in.b1(:,1);

for jj=2:in.irhor%+max(max(VAR.term_spreads_matur),max(VAR.real_rates_init+VAR
    .real_rates_matur-1))
    lvars = (irs(in.p+jj-1:-1:jj,:))';
    irs(in.p+jj,:) = lvars(:)'*in.Phi(1:in.p * n,:);
end

in.irs   = irs(in.p+1:in.p+in.irhor,:);
in.uhat1 = uhat1;
```

iresponse_sign.m

```
function [ir,Omeg] = iresponse_sign(Phi,Sigma,hor,signrestriction,cont)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'iresponse_sign' computes the impulse response functions using sign
% restrictions on the endogenous variables
% Reference: Rubio-Ramirez, J. F., Waggoner, D. F. and Zha, T.: 2010,
% Structural Vector Autoregresions: Theory of Identification and Algorithms
% for Inference, Review of Economic Studies 77(2), 665696.
```

```

% Inputs:
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - hor, horizon of the IRF
% - unit, 1 shock STD or 1 percent increase
% - signrestriction, cell array containing the restrictions. Sign
% restriction can be activated in the toolbox by setting the options
% options.signs{1} = 'y(a,b,c)>0';
% where a, b and c are integer. The syntax means that shock c has a
% positive impact on the a-th variable at horizon b.

% Output:
% - ir contains the IRF
% 1st dimension:   variable
% 2st dimension:   horizon
% 3st dimension:   shock

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
% Revised, 4/11/2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[m,n]    = size(Sigma);
ir       = nan(n,hor,n);
Omeg     = nan(n);
d        = 0;
tol      = 0;
favar    = 1;

if nargin < 5
    cont  = eye(n);
    favar = 0;
end


while d==0 && tol < 30000
    % generate a random orthonormal matrix
    Omega = generateQ(m);
    % compute IRF
    y = iresponse(Phi,Sigma,hor,Omega);
    % uncompress the factors (if favar)
    if favar == 1
        for jj  = 1 : size(y,3)
            yy(:,:,jj) = cont * y(:,:,jj);
```

```
        end
        clear y; y = yy;
    end
    % check restrictions
    d = checkrestrictions(signrestriction,y);
    if d==1
        Omeg  = Omega;
        ir    = y;
    else
        tol = tol + 1;
    end
end

if d==0
    warning('I␣could␣not␣find␣a␣rotation␣satisfying␣the␣restrictions.')
end

end
```

iresponse_sign_narrative.m

```
function [ir,Omeg] = iresponse_sign_narrative(errors,Phi,Sigma,hor,
    signrestriction,narrative,cont)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'iresponse_sign_narrative' computes the impulse response functions using
    sign
% restrictions on the endogenous variables
% Reference:

% Inputs:
% - e, VAR reduced form errors (T x n)
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - hor, horizon of the IRF
% - unit, 1 shock STD or 1 percent increase
% - signrestriction, cell array containing the restrictions. Sign
% restriction can be activated in the toolbox by setting the options
% options.signs{1} = 'y(a,b,c)>0';
% where a, b and c are integer. The syntax means that shock c has a
% positive impact on the a-th variable at horizon b.
% - narrative, cell array containing the restrictions. Sign
% restriction can be activated in the toolbox by setting the options
% options.narrative{1} = 'v(a,b)>0';
% where m can be a vector of scalar and n is a scalar. The syntax means
```

```
% that shock n has to be positive .

% Output:
%— ir contains the IRF
% 1st dimension:    variable
% 2st dimension:    horizon
% 3st dimension:    shock

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


[m,n]   = size(Sigma);
ir      = nan(n,hor,n);
Omeg    = nan(n);
d       = 0;
d0      = 0;
tol     = 0;
favar   = 1;
if nargin < 7
    cont  = eye(n);
    favar = 0;
end


A       = chol(Sigma,'lower');
v       = zeros(size(errors));

while d==0 && tol < 30000
    % generate a random orthonormal matrix
    Omega = generateQ(m);
    % compute IRF
    y = iresponse(Phi,Sigma,hor,Omega);
    % uncompress the factors (if favar)
    if favar == 1
        for jj  = 1 : size(y,3)
            yy(:,:,jj) = cont * y(:,:,jj);
        end
        clear y; y = yy;
    end
    % check sign restrictions
    d0 = checkrestrictions(signrestriction,y);
    if d0 ==1
        v = errors / ( Omega' * A');  % structural innovations
```

```
        % check narrative restrictions
        d = checkrestrictions(narrative,y,v);
    end
    if d==1 % stop
        Omeg  = Omega;
        ir    = y;
    else
        tol = tol + 1;
    end
end

if d==0
    warning('I could not find a rotation satisfying the restrictions.')
end
end
```

iresponse_zeros_signs.m

```
function [ir,Omeg] = iresponse_zeros_signs(Phi,Sigma,hor,lag,var_pos,f,sr,
    draws,toler)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'iresponse_zeros_sign' computes the impulse response functions using zero
% and sign restrictions on the endogenous variables
% References:
% Arias, J. E., Rubio-Ram?rez, J. F. and Waggoner, D. F.: 2018, Inference
% Based on SVARs Identified with Sign and Zero Restrictions: Theory and
% Applications, Econometrica 86, 685720.
% Binning, A.: 2013, Underidentified SVAR models: A framework for combining
% short and long-run restrictions with sign-restrictions, Working Paper
% 2013/14, Norges Bank.

% Inputs:
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - hor, horizon of the IRF
% - unit, 1 shock STD or 1 percent increase
% - (var_pos,f,sr) inputs for the zero and sign restrictions see bvar.m and
% tutorial_.m

% Output:
% - ir contains the IRF
% 1st dimension:   variable
% 2st dimension:   horizon
% 3st dimension:   shock
```

```

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin< 8
    draws = 1; % Number of draws
end
if nargin< 9
    toler = 10000; % Number of rotation attempt
end

p  = lag;
k  = size(Sigma,1);
C1 = chol(Sigma,'lower');

ir      = nan(k,hor,k);
Omeg    = nan(k);

T = hor; % length of impulse response function.

shocks = eye(k);

[Q,index,flag] = findQs(k,f);

if flag == 1
    error('Rank condition not satisfied, the model is overidentified');
end

shock_pos = logical(shocks); % position of shock

% var_pos = [1,1,4,1,1];
% var_pos = [1,2,2,3,3];        % position of corresponding variable to shock
% % eg monetary policy shock should result in a positive increase in interest
% % rates, aggregate demand shock should result in a positive increase in gdp
% % etc.

R = zeros(k,T,length(shocks),draws); % Contains impulse resonse functions
% 1st dimension = variable
% 2nd dimension = time
% 3th dimension = shock
% 4rd dimension = draw

counter = 1;

```

```
74  B       = [Phi(end,:); Phi(1:end-1,:)];
75  Btilde = B(2:end,:)';
76  % Btilde = Phi(1:end-1,:)';
77  alpha = [Btilde;eye(k*(p-1)),zeros(k*(p-1),k)]; % Build companion form matrix
78  if draws > 10
79      wb = waitbar(0, 'Generating Rotations');
80  end
81
82  tj = 0;
83  while counter < draws+1
84
85      tj = tj +1;
86
87      C = generateDraw(C1,k);
88
89      P = findP(C,B,Q,p,k,index);
90
91      W = C*P;
92
93      for jj = 1:length(shocks)
94
95          if W(var_pos(jj),jj) < 0
96              shock = -shocks(:,jj);
97          else
98              shock = shocks(:,jj);
99          end
100
101         V = zeros(k*p,T);
102
103         V(1:k,1) = W*shock;
104
105         chk = W*shock;
106         sr_index = ~isnan(sr(:,jj));
107         tmp = sign(chk(sr_index)) - sr(sr_index,jj);
108
109         if any(tmp~=0)
110             jj = 0;
111             break
112         end
113
114         for ii = 2:T
115             V(:,ii) = alpha*V(:,ii-1);
116         end
117
118         R(:,:,jj,counter) = V(1:k,:);
119
120     end
```

```
121
122     if jj == length(shocks)
123         counter = counter + 1;
124     end
125
126     if tj > toler
127         warning('I could not find a rotation')
128         return;
129     end
130     if draws > 10, waitbar(counter/draws, wb); end
131 end
132
133
134 if draws > 10, close(wb); end
135
136 Omeg = W;
137 ir    = R;
```

isOctave.m

```
1  function r = isOctave ()
2    persistent x;
3    if (isempty (x))
4      x = exist ('OCTAVE_VERSION', 'builtin');
5    end
6    r = x;
7  end
```

jacob_bvar.m

```
1  function J=jacob_bvar(param)
2
3  % Filippo Ferroni, 6/1/2015
4  % Revised, 2/15/2017
5  % Revised, 3/21/2018
6
7  J=zeros(length(param),1);
8  for jj = 1 : length(param)
9      J(jj)= bound0prime(param(jj));
10 end
11 J=diag(J);
12
13 %
```

```
% function y = bound01prime(x);
% y = exp(x)/(1+exp(x))^2;
%
function y = bound0prime(x);
y = exp(x);
```

kf_dk.m

```
function    [shatnew,signew,lh,yhat,fin,kgpart,yforc]=kf_dk(y,H,shat,sig,G,M)

% ========================================================================
% KF_DK
% function [shatnew,signew,lh,yhat,fin,kgain,yforc]=kf_dk(y,H,shat,sig,G,M)
%
% This is Chris Sims's KF with a couple of added outputs
% 1) The (Partial) Kalman Gain and F^(-1) matrices obtained using the
%      Generalized Inverse are part of the output
% s(t) = G*s(t-1) + R*n(t)   V( n(t) ) = Q
% then    M=R*Chol(Q)'=R( CQ')
%         M*M'=R*( CQ'*CQ )*R'
%
% See KF_MOD. for a related filter
% NOTE: KGPART is NOT the appropriate Kalman Gain
% KG=G*KGPART such that
% KGPART=(P(t|t-1)*(H')*(F^-1)
% Use this version when the G matix is time varying and adjust to the
% timing in DK which have a different timing in the state equation
% ========================================================================
% Revised, 2/15/2017
% Revised, 3/21/2018

lh=zeros(1,2);
omega=G*sig*G'+M*M';
[uo doo vo]=svd(omega);
[u d v]=svd(H*uo*sqrt(doo));
first0=min(find(diag(d)<1e-12));
if isempty(first0),first0=min(size(H))+1;end
u=u(:,1:first0-1);
v=v(:,1:first0-1);
d=diag(d);d=diag(d(1:first0-1));
spred = G*shat;
fac=vo*sqrt(doo);
yforc = H*spred;
yhat=y-yforc;
fhalf=(v/d)*u';
```

```
fin=fhalf'*fhalf;
ferr=fhalf*yhat;
lh(1)=-.5*ferr'*ferr;
lh(2)=-sum(log(diag(d)));
kgpart=fac*fhalf;
% Check
%comparemat(kgpart,omega*H'*(inv(H*omega*H')) );
shatnew=fac*ferr+spred;
signew=fac*(eye(size(v,1))-v*v')*fac';
lh=sum(lh);
```

kfilternan.m

```
function [outputkf] = kfilternan(Phi,Sigma,y,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'kfilternan' runs the Kalman filter and smoother with missing values
% References: Durbin and Koopman (2003)

% Inputs:
% - Phi, AR parameters of the VAR
% - Sigma, Covariance matrix of the reduced form VAR shocks
% - y, data
% - options, various options
% TVP parameters allowed (3rd dimension of Phi,Sigma and options.tauVec
% controls the time of the variation)

% outputkf : (see below)

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% kfilternan computes the forward Kalman filter with NaN

[T,var]          = size(y);
data             = y';
tauVec           = ones(T,1);
initialCond      = 0;
adjustment       = 0;
index            = zeros(var,1);
noprint          = 0;

```

```
33 if nargin > 3
34     if isfield(options,'tauVec') == 1
35         tauVec=optinos.tauVec;
36     end
37     if isfield(options,'initialCond') ==1
38         initialCond = options.initialCond;
39     end
40     if isfield(options,'initialCond')==1 && options.initialCond==2
41         initialCond = options.initialCond;
42         pZero   = options.pZero;
43         aZero   = options.aZero;
44     end
45     if isfield(options,'adjustment')==1
46         adjustment = options.adjustment;
47     end
48     if isfield(options,'index')==1
49         index = options.index;
50     end
51     if isfield(options,'noprint')==1
52         noprint = options.noprint;
53     end
54 end
55
56 if length(tauVec)~=T
57     warning('Tauvec is Longer than T');
58     quer('c');
59 end
60
61 %=============================================================================
62 % 1.0 obtain the steady state representation of the VAR
63 for jj  = 1 : size(Phi,3)
64     [A(:,:,jj),B(:,:,jj),C(:,:,jj),const(:,jj),Sigma(:,:,jj),~,index_var]=
          var2ss(Phi,Sigma,index);
65 end
66
67 %=============================================================================
68 % 1.1. Dimensions and storage
69 ns      = size(B,1);
70 vt      = zeros(var,T);
71 finvt   = zeros(var,var,T);
72 kpartg  = zeros(ns,var,T);
73 logLnc  = zeros(T,1);
74 yfor    = zeros(size(C,1),T);
75 sfor    = zeros(size(A,1),T);
76 % Matrices with one additional entrdataStru.data (initialization)
77 % to recover observables
78 stt        = zeros(ns,T+1);
```

```
79  ptt         = zeros(ns,ns,T+1);
80  W           = eye(var);
81  Zdim        = zeros(T,1);
82  mat_obspos = zeros(T,var);
83
84  % 1.2 Initialization
85  if initialCond==0
86      stt(:,1) = zeros(ns,1);
87      P0 = lyapunov_symm(A(:,:,tauVec(1)),...
88          B(:,:,tauVec(1))*(Sigma(:,:,tauVec(1))')...
89          *Sigma(:,:,tauVec(1))*(B(:,:,tauVec(1))'));
90      ptt(:,:,1)=P0;
91  elseif initialCond==1 %non stationary data
92      P0          = 10*eye(size(A,1));
93      stt(:,1)    = zeros(size(A,1),1);
94      ptt(:,:,1) = P0;
95  elseif initialCond==2
96      P0          = pZero;
97      stt(:,1)    = aZero;
98      ptt(:,:,1) = P0;
99  end
100
101 nbreak = 0;
102 time   = 0;
103 state  = stt(:,1);
104
105 %=========================================================================
106 % 1.3 Start Forward Filter using KF_DK
107 for ii=1:T
108
109     % Handling of missing observations
110     ytt      = data(:,ii);
111
112     % Determine W and position of the NAN
113     ind          = ~isnan(ytt);
114     rowt         = find(~isnan(ytt));
115     ytt          = ytt(ind);
116     Zdim(ii)     = length(ytt);
117     Ztt          = W((ind==1),:)*C(:,:,tauVec(ii));
118     mat_obspos(ii,1:Zdim(ii)) = rowt;
119     dimt         =( 1:Zdim(ii) );
120     % Demeaning is done here
121     ytt = ytt - Ztt*const(:,tauVec(ii));
122
123     % Forecast Part
124     % sfor is the state at time t conditional on info at time t-1, s(t|t-1)
125     sfor(:,ii)       = A(:,:,tauVec(ii))*state;
```

```
126     yfor(:,ii)      = C(:,:,tauVec(ii))*(A(:,:,tauVec(ii))*state + const(:,
            tauVec(ii)));
127
128     %     % computing the 1, 2,3,4 step ahead forecast
129     %     yfrsct(ii,dimt,1) = yfor(dimt,ii);
130     %     yfrsct(ii,dimt,2) = Ztt*(G(:,:,tauVec(ii))^2*stt(:,ii) + C(:,tauVec(
            ii)));
131     %     yfrsct(ii,dimt,3) = Ztt*(G(:,:,tauVec(ii))^3*stt(:,ii) + C(:,tauVec(
            ii)));
132     %     yfrsct(ii,dimt,4) = Ztt*(G(:,:,tauVec(ii))^4*stt(:,ii) + C(:,tauVec(
            ii)));
133
134     [stt(:,ii+1),ptt(:,:,ii+1),logLnc(ii),vt(dimt,ii),finvt(dimt,dimt,ii),...
135         kpartg(:,dimt,ii),]=feval(@kf_dk,ytt,Ztt,...%Z(:,:,tauVec(ii)),...
136         state,ptt(:,:,ii),A(:,:,tauVec(ii)),...
137         B(:,:,tauVec(ii))*(Sigma(:,:,tauVec(ii))'));
138
139     % if there is break
140     if ii < T
141         if tauVec(ii+1) - tauVec(ii) > 0
142             nbreak              = nbreak +1;
143             time(nbreak)        = ii+1;
144             state = stt(:,ii+1) + adjustment(:,nbreak);
145         else
146             state = stt(:,ii+1);
147         end
148     end
149
150 end
151
152 %=============================================================================
153 % 2. Likelihood with Integration Constant
154 outputkf.logLncFull     = logLnc;
155 %logLnc                  = logLnc(dataStru.trainVec(1):dataStru.trainVec(2));
156 logLnc                  = logLnc(1:end);
157 logL                    = -0.5*sum(Zdim)*log(2*pi)+sum(logLnc);
158
159 %=============================================================================
160 % 3. Truncate filters and obtain initial observations
161 % outSt.yferr=vt';
162 outputkf.yferr = (data- yfor)';
163 yfor        = yfor';
164 stt         = stt(:,2:end);
165 ptt         = ptt(:,:,2:end);
166
167 % % add the 1,2,3,4 ste ahead forecasts in the output
168 % for hf = 1
```

```
%       tmp   = (dataStru.data(:,1+hf:end) - yfrsct(1:end-hf,:,hf) ) *   ...
%           (dataStru.data(:,1+hf:end) - yfrsct(1:end-hf,:,hf) )';
%       outSt.frscterror(:,hf) =
% end

%=========================================================================
% 4. Disturbance smoother with TV matrices
% Obtain the Innovations using a disturbance smoother
etamat      = zeros(var,T);
smooth_st   = zeros(ns,T);
rmat        = zeros(ns,T);

%=========================================================================
% 4.1 Initialize RSTAR & start at t=Nobs
rstar       = zeros(ns,1);
[rstar,etamat(:,end)]   = ...
    smoothdis(rstar,...
    (Sigma(:,:,tauVec(end))')*Sigma(:,:,tauVec(end)),B(:,:,tauVec(end))',...
    Ztt', finvt(dimt,dimt,end),zeros(ns),vt(dimt,end));
smooth_st(:,end)        = stt(:,end)+ptt(:,:,end)*rstar;
rmat(:,end)             = rstar;
Nmat = zeros(ns,ns,T);
%=========================================================================
% 4.2 Begin Backward recursion
for ii=(T-1):-1:1

    % Varying dimension in the backward recurions
    dimt=( 1:Zdim(ii) );
    % [Ztt]'= [Wtt*Z]' = = Z'*Wtt'
    Ztt = (C(:,:,tauVec(ii))')*( W( mat_obspos(ii,1:Zdim(ii)),:)');

    [rstar,etamat(:,ii)]=smoothdis(rstar,...
        (Sigma(:,:,tauVec(ii))')*Sigma(:,:,tauVec(ii)),...
        B(:,:,tauVec(ii))',Ztt,finvt(dimt,dimt,ii),...
        ((A(:,:,tauVec(ii+1))-A(:,:,tauVec(ii+1))*...
        kpartg(:,dimt,ii)*Ztt')'),vt(dimt,ii));

    smooth_st(:,ii)=stt(:,ii)+ptt(:,:,ii)*rstar;

    rmat(:,ii)=rstar;

end
%=========================================================================
% 4.3 smoothed initial condition
if initialCond~=2
    a0 = P0*A(:,:,tauVec(1))'*rstar;   % Note: this is only correct in the
        case where a0 = zeros(ns,1)
```

```
215 else
216     iGG = pinv(A(:,:,tauVec(1)));
217     a0  = iGG*( smooth_st(:,1) - B(:,:,tauVec(1))* etamat(:,1));
218 end
219
220 etamat      = (etamat)';
221 smooth_st   = (smooth_st)';
222 stt         = stt';
223
224 %=============================================================================
225 % % 5. Check Smoother
226 % % Check that Smooth States are identical if using disturbance smoother
227 % % (above) vs. state smoother (below) and if can also recover the observables
228 tol=1e-5;
229
230 Ydem = data - repmat((C(:,:,tauVec(1))*const(:,tauVec(1))),1,size(data,2));
231 if max(tauVec) > 1 % only one break
232     indx = min(find(tauVec==2));
233     tmp1      = data(:,1:indx-1) - repmat((C(:,:,tauVec(1))*const(:,tauVec(1))
        ),1,indx-1);
234     tmp2      = data(:,indx:end) - repmat((C(:,:,tauVec(2))*const(:,tauVec(2))
        ),1,size(data,2)-indx+1);
235     Ydem = [tmp1 tmp2];
236 end
237
238 maxdifYf     = max(max(abs(stt(:,index_var) - Ydem')));
239 if noprint,
240 else
241     disp(['Max Discrepancy Filtered vs. Actual Data: ' num2str(maxdifYf)]);
242 end
243
244 maxdifYs     = max(max(abs(smooth_st(:,index_var) - Ydem')));
245 if noprint,
246 else
247     disp(['Max Discrepancy Smooth vs. Actual Data: ' num2str(maxdifYs)]);
248 end% maxdifY=comparemat(dataStru.data, Ynanfill');
249
250 if maxdifYf > tol || maxdifYs > tol
251     warning('Smoother and Filter discrepancy exceeds tolerance')
252 end
253
254
255 %=============================================================================
256 % 6. Store results
257
258 outputkf.index_var          = index_var;
259 outputkf.logL               = logL;
```

```
260 outputkf.filteredSt          = stt;
261 outputkf.CovSt               = ptt;
262 outputkf.smoothSt            = smooth_st;
263 outputkf.innovations         = etamat;
264 outputkf.forecastObs         = yfor;
265 outputkf.aZero               = a0;
266 outputkf.adjustment          = adjustment;
267 outputkf.nbreak              = nbreak;
268 outputkf.time                = time;
269 outputkf.smoothSt_plus_ss    = zeros(size(smooth_st));
270 outputkf.filteredSt_plus_ss = zeros(size(smooth_st));
271 if time>0
272     outputkf.smoothSt_plus_ss(1:time-1,:) = smooth_st(1:time-1,:) + repmat(
        const(:,1)',time-1,1);
273     outputkf.smoothSt_plus_ss(time:end,:) = smooth_st(time:end,:) + repmat(
        const(:,2)',T-time+1,1);
274     outputkf.filteredSt_plus_ss(1:time-1,:) = stt(1:time-1,:) + repmat(const
        (:,1)',time-1,1);
275     outputkf.filteredSt_plus_ss(time:end,:) = stt(time:end,:) + repmat(const
        (:,2)',T-time+1,1);
276
277 else
278     outputkf.smoothSt_plus_ss = smooth_st + repmat(const(:,end)',size(
        smooth_st,1),1);
279     outputkf.filteredSt_plus_ss = stt + repmat(const(:,end)',size(smooth_st,1)
        ,1);
280 end
281
282 Ydem(find(isnan(Ydem))) = 0;
283 tmpf = outputkf.yferr';
284 tmpf(find(isnan(tmpf))) = 0;
285 outputkf.r2 = 1 - diag(tmpf * tmpf') ./ diag(Ydem*Ydem');
286 %=========================================================================
287 % % 7. Simulating the state vector
288 etatilde     = zeros(var,T);
289
290 for tt =1 : T
291     Qt = (Sigma(:,:,tauVec(tt))')*Sigma(:,:,tauVec(tt));
292     Ct = Qt - Qt*B(:,:,tauVec(tt))'*Nmat(:,:,tt)*B(:,:,tauVec(tt))*Qt;
293     [a,b,~] = svd(Ct);
294     iS      = a* sqrt(b);
295     dt = iS*randn(var,1);
296     etatilde(:,tt) = dt + Qt*B(:,:,tauVec(tt))'*rmat(:,tt);
297 end
298 if nbreak == 0
299     [~,smooth_sim]=kfilterRegSplitSimulation(a0,etatilde);
300 else
```

```
301     options.nbreak       = nbreak;
302     options.time         = time;
303     options.adjustment   = adjustment;
304     [~,smooth_sim] = kfilterRegSplitSimulation(a0,etatilde,options);
305 end
306 if time>0
307     outputkf.smoothSt_sim_plus_ss(1:time-1,:) = smooth_sim(1:time-1,:) +
            repmat(const(:,1)',time-1,1);
308     outputkf.smoothSt_sim_plus_ss(time:end,:) = smooth_sim(time:end,:) +
            repmat(const(:,2)',T-time+1,1);
309 else
310     outputkf.smoothSt_sim_plus_ss = smooth_sim + repmat(const(:,end)',size(
            smooth_sim,1),1);
311 end
312
313
314 %% Subroutine kfilterRegSplitSimulation allows to simulate the model
315 % Inputs
316 % sInitial: [ns 1] Initial State
317 % innovMat: [nx T] matrix of innovations
318 % By being a sub-routine it has access to all variables defined above
319 % Be-careful not to use an index (ii,jj) used above or to repeat variable
320 % names
321     function [ySim,sSim]=kfilterRegSplitSimulation(sInitial,innovMat,options)
322         if nargin < 3
323             nbreak = 0;
324             adjustment =0;
325             time =0;
326         else
327             if isfield(options,'time')
328                 time = options.time;
329             else
330                 error('time of the break is missing')
331             end
332             if isfield(options,'adjustment')
333                 adjustment = options.adjustment;
334             else
335                 error('State adjustment is missing')
336             end
337         end
338
339         if ~isequal(size(innovMat),[var T])
340             error('Input innovMat must be [nx T]')
341         end
342         sSim=zeros(ns,T);
343         ySim=zeros(size(data));
344         sSim(:,1)=A(:,:,tauVec(1))*sInitial + ...
```

```
345             B(:,:,tauVec(1))*innovMat(:,1);
346         ySim(:,1)=C(:,:,tauVec(1))*sSim(:,1); %+C(:,tauVec(1)));
347         for kk=2:T;
348             if any(kk == time)
349                 % sSim(:,kk-1) = sSim(:,kk-1) ;
350                 sSim(:,kk-1) = sSim(:,kk-1) + adjustment(:,find(kk == time));
351             end
352             sSim(:,kk)=A(:,:,tauVec(kk))*sSim(:,kk-1)+...
353                 B(:,:,tauVec(kk))*innovMat(:,kk);
354             ySim(:,kk)=C(:,:,tauVec(kk))*sSim(:,kk); %+C(:,tauVec(kk)));
355         end
356         ySim=ySim';
357         sSim=sSim';
358     end
359
360     function [rzero,what]=smoothdis(rone,Q,Rtr,Ztr,Fnvr,Ltr,v)
361         % function [rzero,what]=smoothdis(rone,Q,Rtr,Ztr,Fnvr,Ltr,v)
362         %
363         % Disturbance smoother recursion for a model with no error in the obs.
364         % equation using the formulas in Durbin and Koopman Ch. 4.2
365         % rzero=Z'*F^(-1)*v + L'*rone
366         % what =Q*R'*rone
367         %
368         % Inputs
369         % Rtr = R'
370         % Ztr = Z'
371         % Ltr = L'
372         % Finvr = F^(-1)
373         %
374         % Revised, 2/15/2017
375         % Revised, 3/21/2018
376
377         rzero=Ztr*Fnvr*v + Ltr*rone ;
378         what=Q*Rtr*rzero;
379     end
380
381
382 %% End of File
383 end
```

lagX.m

```
1  function XLag = lagX(X,lags)
2  %lagX Create matrix of lagged time series
3  %
```

```
if size(X,2) > size(X,1)
   X = X'; % Ensure a column vector
end

missingValue = NaN;  % Assign default missing value
Lags = length(lags); % Number of lags to apply to each time series
[T,ny] = size(X);
XLag = missingValue(ones(T,ny*Lags)); % Preallocate

for c = 1:Lags

    L       = lags(c);
    columns = (ny*(c-1)+1):c*ny; % Columns to fill, this lag

    if L > 0 % Time delays
       XLag((L + 1):end,columns) = X(1:(end - L), :);

    elseif L < 0 % Time leads
       XLag(1:(end + L),columns) = X((1 - L):end, :);

    else % No shifts
       XLag(:,columns) = X;

    end

end
```

lag_crit_var.m

```
function lag_crit_var(y,maxlag)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'lag_crit_var' prints various info crit (not allowed NaN)

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 9/11/2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

crt_ = nan(4,maxlag);

opt.K =1;
for ilag = 1 : maxlag
    tmp_ = bvar(y,ilag,opt);
```

```
17      crt_(1,ilag) =  tmp_.InfoCrit.AIC;
18      crt_(2,ilag) =  tmp_.InfoCrit.SIC;
19      crt_(3,ilag) =  tmp_.InfoCrit.HQIC;
20      crt_(4,ilag) =  tmp_.InfoCrit.BIC;
21  end
22  rownam = {'AIC','SIC','HQIC','BIC'};
23  for ilag = 1 : maxlag
24      disp('============================================')
25      X = sprintf('%s = %0.0g','Number of lags',ilag);
26      disp(X)
27      for jj =1: size(crt_,1)
28          X = sprintf('%s = %0.5g',rownam{jj},crt_(jj,ilag));
29          disp(X)
30      end
31  end
```

lyapunov_symm.m

```
1  % Copyright (C) 2006-2017 Dynare Team
2  % solves x-a*x*a'=b for b (and then x) symmetrical
3  function [x,info]=lyapunov_symm(a,b)
4    info = 0;
5    n = size(b,1);
6    if n == 1
7      x=b/(1-a*a);
8      return
9    end
10   x=zeros(n,n);
11   [u,t]=schur(a);
12   b=u'*b*u;
13   for i=n:-1:2
14     if t(i,i-1) == 0
15       if i == n
16         c = zeros(n,1);
17       else
18         c = t(1:i,:)*(x(:,i+1:end)*t(i,i+1:end)')+...
19             t(i,i)*t(1:i,i+1:end)*x(i+1:end,i);
20       end
21       q = eye(i)-t(1:i,1:i)*t(i,i);
22       x(1:i,i) = q\(b(1:i,i)+c);
23       x(i,1:i-1) = x(1:i-1,i)';
24     else
25       if i == n
26         c = zeros(n,1);
27         c1 = zeros(n,1);
```

```
      else
        c = t(1:i,:)*(x(:,i+1:end)*t(i,i+1:end)')+...
            t(i,i)*t(1:i,i+1:end)*x(i+1:end,i)+...
            t(i,i-1)*t(1:i,i+1:end)*x(i+1:end,i-1);
        c1 = t(1:i,:)*(x(:,i+1:end)*t(i-1,i+1:end)')+...
             t(i-1,i-1)*t(1:i,i+1:end)*x(i+1:end,i-1)+...
             t(i-1,i)*t(1:i,i+1:end)*x(i+1:end,i);
      end
      q = [eye(i)-t(1:i,1:i)*t(i,i) -t(1:i,1:i)*t(i,i-1);...
           -t(1:i,1:i)*t(i-1,i) eye(i)-t(1:i,1:i)*t(i-1,i-1)];
      z =  q\[b(1:i,i)+c;b(1:i,i-1)+c1];
      x(1:i,i) = z(1:i);
      x(1:i,i-1) = z(i+1:end);
      x(i,1:i-1)=x(1:i-1,i)';
      x(i-1,1:i-2)=x(1:i-2,i-1)';
      i = i - 1;
    end
  end
  if i == 2
    c = t(1,:)*(x(:,2:end)*t(1,2:end)')+t(1,1)*t(1,2:end)*x(2:end,1);
    x(1,1)=(b(1,1)+c)/(1-t(1,1)*t(1,1));
  end
  x=u*x*u';
```

matrictint.m

```
function w = matrictint(S, df, XXi)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computes the log of the integral of the kernel of the PDF of a
% normal-inverse-Wishart distribution.
%
% S:    parameter of inverse-Wishart distribution
% df:   number of degrees of freedom of inverse-Wishart distribution
% XXi:  first component of VCV matrix of matrix-normal distribution
%
% Computes the integral over (Phi, Sigma) of:
%
% det(Sigma)^(-k/2)*exp(-0.5*Tr((Phi-PhiHat)'*(XXi)^(-1)*(Phi-PhiHat)*Sigma^(-1)))*
% det(Sigma)^((df+ny+1)/2)*exp(-0.5*Tr(Sigma^(-1)*S))
%
% (where k is the dimension of XXi and ny is the dimension of S and
% Sigma)

```

```
% Original file downloaded from:
% http://sims.princeton.edu/yftp/VARtools/matlab/matrictint.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k=size(XXi,1);
ny=size(S,1);
[cx,p]=chol(XXi);
[cs,q]=chol(S);

if any(diag(cx)<100*eps)
    error('singular XXi')
end
if any(diag(cs<100*eps))
    error('singular S')
end

% Matrix-normal component
w1 = 0.5*k*ny*log(2*pi)+ny*sum(log(diag(cx)));

% Inverse-Wishart component
w2 = -df*sum(log(diag(cs))) + 0.5*df*ny*log(2) + ny*(ny-1)*0.25*log(pi) + ggammaln(ny, df);

w = w1 + w2;

function lgg = ggammaln(m, df)
if df <= (m-1)
    error('Too few df in ggammaln: increase the # of obs or decrease the # of lags.')
else
    garg = 0.5*(df+(0:-1:1-m));
    lgg = sum(gammaln(garg));
end
```

max_fevd.m

```
function Qbar  = max_fevd(i, h, j, Phi, Sigma, Kappa)
% finds the rotation where shock j maximizes the fevd of vriable i at horizon h
% see also fevd.m

if nargin < 6
    Kappa = 1000;
end

N    = size(Sigma,1);
```

```
crit = nan(Kappa,1);
Q    = nan(N,N,Kappa);
for k = 1 : Kappa
    Q(:,:,k) = generateQ(N);                  % generate an orthonormal matrix
    FEVD     = fevd(h,Phi,Sigma,Q(:,:,k));  % compute the FEVD
    % Calculate the contribution of shock j, to variable i forecast error volatility, at horizon h
    crit(k,1) = FEVD(i, h, j);
end
% Pick the maximum
[~,index] = max(crit);
Qbar      = Q(:,:,index);

end
```

mniw_log_dnsty.m

```
function  log_dnsty = mniw_log_dnsty(prior,posterior,var)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'logmlike' computes the marginal likelihood for the NM-IW
% Inputs:
% Output: marginal data density
% Filippo Ferroni, 3/21/2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nobs,ny]  = size(var.y);
nk         = size(var.X,2);

Fv    = chol(prior.Phi.cov)';
Fo    = chol(inv(prior.Sigma.scale))';
iV    = inv(prior.Phi.cov);

var.e = var.y - var.X*posterior.PhiHat;

log_dnsty = - nobs*ny/2 * log(pi);
log_dnsty = log_dnsty - nobs / 2 *log(det(prior.Sigma.scale));
log_dnsty = log_dnsty - ny/2 * log(det( eye(nk) + Fv'*var.X'*var.X*Fv ));
% % Giannone, Lenza Primiceri (2015) Appendix  A13-A14
FF    = eye(ny) ...
    + Fo'* ((posterior.PhiHat - prior.Phi.mean)'* iV * (posterior.PhiHat - prior.Phi.mean) ...
    + var.e'* var.e) * Fo;
log_dnsty = log_dnsty - (nobs + prior.Sigma.df)/2 * log(det(  FF ));
log_dnsty = log_dnsty + ggammaln(ny,(nobs + prior.Sigma.df)/2) ;
```

```
log_dnsty = log_dnsty - ggammaln(ny,prior.Sigma.df/2);


function lgg = ggammaln(m, df)
if df <= (m-1)
    error('too few df in ggammaln; increase the number of observations or reduce the number of lags')
else
    garg = 0.5*(df+(0:-1:1-m));
    lgg = sum(gammaln(garg));
end

% posterior.S = var.u' * var.u + prior.Sigma.scale + ...
%     prior.Phi.mean' * Ai * prior.Phi.mean + ...
%     var.B' * (var.X'*var.X) * var.B ...
%     - posterior.PhiHat' * (var.X'*var.X + Ai) * posterior.PhiHat;
% FF =  posterior.S;
% log_dnsty1 = - nobs*ny/2 * log(pi);
% log_dnsty1 = log_dnsty1 + prior.Sigma.df / 2 *log(det(prior.Sigma.scale));
% log_dnsty1 = log_dnsty1 - ny/2 * log(det(prior.Phi.cov));
% log_dnsty1 = log_dnsty1 - ny/2 * log(det( var.X'*var.X + iV )); % X'X + inv(V)
% %log_dnsty1 = log_dnsty1 - ny/2 * log(det( eye(nk) + Fv'*var.X'*var.X*Fv ));
% % % Giannone, Lenza Primiceri (2015) Appendix A13-A14
% FF    = prior.Sigma.scale ...
%     + (posterior.PhiHat - prior.Phi.mean)'* iV * (posterior.PhiHat - prior.Phi.mean) ...
%     + var.e'* var.e;
% % FF    = eye(ny) ...
% %     + Fo'* ((posterior.PhiHat - prior.Phi.mean)'* iV * (posterior.PhiHat - prior.Phi.mean) ...
% %     + var.e'* var.e) * Fo;
% log_dnsty1 = log_dnsty1 - (nobs + prior.Sigma.df)/2 * log(det(  FF ));
% log_dnsty1 = log_dnsty1 + ggammaln(ny,(nobs + prior.Sigma.df)/2) - ggammaln(ny,prior.Sigma.df/2);
```

ols_reg.m

```
function [output] = ols_reg(Y,X,options)

% this function computes the ols coefficients of Y on X
% Y TxN
% X Txk (interepct at the end)
% Y = XB + E

```

```
8  nindex       = find(sum(isnan([Y,X]),2)==0);
9  index        = find(sum(isnan([Y,X]),2)>0);
10 Y(index,:)   = [];
11 X(index,:)   = [];
12 [N,K]        = size(X);
13 robust_se_   = 0;
14 L            = round(N/4);
15
16 if nargin > 2
17     if isfield(options,'robust_se_') == 1
18         robust_se_ = options.robust_se_;
19     end
20     if isfield(options,'L') == 1
21         L = options.L;
22     end
23 end
24
25
26 ny = size(Y,2);
27
28 XX       = (X'*X);
29 iXX      = XX\eye(K);
30 Bols     = X\Y;                    % Bols    = (X'*X)\(X'*Y);
31 err      = Y - X*Bols;
32 se       = nan(K,ny);
33 Sols     = zeros(K*ny);
34
35 if robust_se_==2 % NW Robust SE
36     %-------------------------------------------------------------------------%
37     Serror     = (err'*err);
38     nwWeights  = (L+1-(1:L))./(L+1);
39     for j= 1 : L
40         G      = (err(j+1 : N, :)'*err(1 : N-j , :));
41         Serror = Serror + nwWeights(j) * (G + G');
42     end
43     Serror = Serror/(N-K);
44     Sols   = kron(diag(diag(Serror)),iXX);
45     se     = reshape(sqrt(diag(Sols)), K, ny);
46
47 elseif robust_se_ == 1 % Hamilton (1994), Ch 10 pag 282, eq (10.5.20)
48     %-------------------------------------------------------------------------%
49     for vv = 1: ny % equation by equation
50         u= err(:,vv);
51         errs=X.*u;
52         V0 = [errs'*errs] / N ; %regular weighting matrix
53         for ind_i = (1:L)
54             S = errs(1:N-ind_i,:)'*errs(1+ind_i:N,:) / N;
```

```
55              V0 = V0 + (1 - ind_i/(L+1))*(S + S');
56          end
57          %      D     =    inv((X'*X)/N);
58          %      varb = 1/N*D*V1*D;
59          Solsj =  N * iXX * V0 * iXX;
60          Sols((vv-1)*K+1 : vv*K, (vv-1)*K+1 : K*vv ) = Solsj;
61      end
62      Serror = (err'*err)/(N-K); % not sure this is the correct Coavariance of
            the shocks.
63      se     = reshape(sqrt(diag(Sols)), K, ny);
64
65
66  elseif robust_se_ == 5 % Matlab HAC function
67      %-------------------------------------------------------------------------%
68      if  exist('hac') ==2
69          % find constant
70          index = find(sum(diff(X,2),1)~=0);
71          indexC = find(sum(diff(X,2),1)==0);
72          for vv = 1: ny
73  %               [EstCoeffCov0,se0,~] = hac(X(:,index),Y(:,ny),'display','off','
        type','HC');
74              [EstCoeffCov0,se0,~] = hac(X(:,index),Y(:,ny),'display','off','
                    bandwidth','AR1OLS'); % remove the intercept
75              se(index,vv)  = se0(2:end);
76              se(indexC,vv) = se0(1); %intercept
77              Sols((vv-1)*K+1 : vv*K, (vv-1)*K+1 : K*vv ) = EstCoeffCov0; %
                    order is not correct
78          end
79          Serror  = 1/(N-K)*(err'*err);
80  %          output.TtestRobust  = coeff./se;
81  %          output.pvalueRobust = tpdf(output.TtestRobust,N-K);
82      else
83          error('Matlab_Econ_Toolbox_missing')
84      end
85  else
86      Serror  = 1/(N-K)*(err'*err);
87      Sols    = kron(diag(diag(Serror)),iXX);
88      se      = reshape(sqrt(diag(Sols)), K, ny);
89  end
90
91  Ttest  = Bols./ reshape(sqrt(diag(Sols)), K, ny);
92  ESS    = diag((X*Bols - mean(Y))'*(X*Bols - mean(Y)));
93  RSS    = diag(err' * err);
94  TSS    = diag((Y - mean(Y))' * (Y - mean(Y)));
95  R2     = ones(length(ESS),1) - RSS ./ TSS;
96  adjR2  = ones(length(ESS),1) - (ones(length(ESS),1) - R2)*(N-1)/(N-K);
97  Ftest  = ESS/(K-1) ./ diag(Serror);
```

```
98
99  for v = 1 : ny
100     output.logl(v,1)    = -N/2*log(2*pi*Serror(v,v)) - RSS(v,1)/(2*Serror(v,v))
            ;
101     [output.AIC(v,1), output.SIC(v,1), output.HQIC(v,1)] = IC(output.logl(v,1)
            , N, K);
102 end
103
104 output.beta   = Bols;                    % OLS estimator
105 output.error  = err;                    % (TxN) matrix of Residuals
106 output.Serror = Serror;                 % Covariance matrix of Residuals
107 output.Sols   = Sols;                   % Covariance matrix of Bols
108 output.Ttest  = Ttest;                  % t-statistics
109 output.pvalue = tpdf(Ttest,N-K);        % p-value
110 output.Ftest  = Ftest;                  % F-test
111 output.R2     = R2;                     % R2
112 output.adjR2  = adjR2;                  % Adjusted R2
113 output.yfit   = X*Bols;                 % Fitted Values
114 output.N      = N;                      % # of observation used
115 output.K      = K;                      % # of regressors
116 output.nindex = nindex;                 % index of missing observation
117 output.index  = index;                  % index of observation used
118 output.se     = se;
119 output.XX     = XX;
120 output.X      = X;
121 output.Y      = Y;
122
123
124 % if nargin > 2
125 %     [EstCov,se,coeff] = hac(X(:,1:end-1),Y,'display','off'); % remove the
        intercept
126 %     output.serob         = se(2:end);
127 %     output.serob(end+1) = se(1); %intercept
128 %     output.TtestRobust   = coeff./se;
129 %     output.pvalueRobust = tpdf(output.TtestRobust,N-K);
130 % end
```

p2p.m

```
1  function [posterior,prior] = p2p(hh, shrinkage,prior0,olsreg,F,G,Fo,
       positions_nylags,position_constant)
2  %
3  %******************************************************
4  % Conjugate Prior: MN-IW for Direct Methods
5  %******************************************************
```

```
6
7  % settings
8  if isempty(position_constant) == 0
9      nx = 1;
10 else
11     nx = 0;
12 end
13
14 ny       = size(G,2);
15 nylags   = size(F,1);
16 iIminusF = inv(eye(nylags) - F);
17
18 % constructing the prior mean
19 Fhh      = F^hh;
20 Fohh     = (iIminusF * (eye(nylags)-Fhh)) * Fo;
21 if hh < 40
22     priorSigmaScale  = zeros(ny);
23     for j = 0 : hh
24         priorSigmaScale = priorSigmaScale + G' * F^(hh-j)* G * prior0.Sigma.
               scale * G' * F^(hh-j)' * G;
25     end
26 else
27     % solves x-a*x*a'=b for b (and then x) symmetrical
28     % function [x, info]=lyapunov_symm(a, b)
29     [priorSigmaScale0,~] = lyapunov_symm(F,G*prior0.Sigma.scale*G' - F^(hh+1)*
           G * prior0.Sigma.scale * G' * F^(hh+1)');
30     % max(max(abs(priorSigmaScale-G'*priorSigmaScale0*G)))
31     priorSigmaScale      = G'*priorSigmaScale0*G;
32 end
33
34 prior.BetaMean   = [Fhh(1:ny,:)'; Fohh(1 : ny,1)'];
35 prior.BetaVar    = prior0.Phi.cov * 1/ shrinkage;
36
37 prior.df  = prior0.Sigma.df;            % usually number of regressors minus
       the 2
38 prior.XXi = inv( prior.BetaVar );       % V^{-1}
39 prior.S   = priorSigmaScale;            % Sigma0
40
41 % retrieve the OLS
42 B_   = olsreg.beta([positions_nylags position_constant], :);
43 XX_  = olsreg.X(:,[positions_nylags position_constant])' * olsreg.X(:,[
       positions_nylags position_constant]);
44 E_   = olsreg.error;
45 XXp_ = XX_ + prior.XXi;
46
47 % construct the posterior
48 posterior.df     = olsreg.N - nylags - nx + prior0.Sigma.df;
```

```
posterior.XXi    = inv( XXp_ );
posterior.PhiHat = posterior.XXi * (XX_ * B_ + prior.XXi * prior.BetaMean);
posterior.S      =  ...
    E_'* E_ + priorSigmaScale + prior.BetaMean' * prior.XXi * prior.BetaMean +
        ...
    B_' * XX_ * B_ - posterior.PhiHat' * XXp_ * posterior.PhiHat;
% FF    = prior.Sigma.scale + (posterior1.PhiHat - prior1.BetaMean)'* prior1.
    XXi * (posterior1.PhiHat - prior1.BetaMean) ...
%     + posterior1.E_'* posterior1.E_;

posterior.B_   = B_ ;
posterior.XX_  = XX_;       %olsreg.X(:,[positions_nylags position_constant])'
     * olsreg.X(:,[positions_nylags position_constant]);
posterior.E_   = E_;        %olsreg.error';
posterior.XXp_ = XXp_;      %XX_ + prior.XXi;
posterior.U_   = olsreg.Y - olsreg.X(:,[positions_nylags position_constant]) *
     posterior.PhiHat;% posterior errors;
```

pc_T.m

```
% principal components with normalization F'F/T=I
% X is observed
% r is the true number of true factors
% F is T by r matrix of true factors
% Lambda N by r is the true loading matrix
% C=F*Lambda' T by N is the true common component
% chat is the estimated common component

function [ehat,fhat,lambda,ss,Scale]=pc_T(yy,nfac,DEMEAN)

Scale = ones(size(yy,2),1);

if DEMEAN == 2
    [y,Scale]=standard(yy);
elseif DEMEAN ==1
    y=demean(yy);
else
    y=yy;
end


[bigt,bign]=size(y);
yy=y*y';
[Fhat0,eigval,Fhat1]=svd(yy);
fhat=Fhat0(:,1:nfac)*sqrt(bigt);
```

```
lambda=y'*fhat/bigt;

%chi2=fhat*lambda';
%diag(lambda'*lambda)
%diag(fhat'*fhat)                % this should equal the largest eigenvalues
%sum(diag(eigval(1:nfac,1:nfac)))/sum(diag(eigval))
%mean(var(chi2))                 % this should equal decomposition of variance

ehat=y-fhat*lambda';

ve2=sum(ehat'.*ehat')'/bign;
ss=diag(eigval);
```

plot_all_irfs_.m

```
function plot_all_irfs_(irfs,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

% input : irfs
% 1st dimension: variable
% 2nd dimension: horizon
% 3rd dimension: shock
% 4th dimension: draws
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nvar    = size(irfs,1);
hor     = size(irfs,2);
nshocks = size(irfs,3);
ndraws  = size(irfs,4);
nplots  = [nvar nshocks];
savefig_yes = 0;
conf_sig    = 0.68;
normz       = 1;
add_irfs_yes = 0;
normz_yes   = 0;
add_multiple_bands_yes = 0;

if nargin <2
    disp('You did not provided names for shocks nor variables.')
    disp('I call them Var 1, Var 2, .... and Shck 1, ....')
    for v = 1 : nvar
```

```
31          eval(['varnames{'   num2str(v) '} = ''Var  ' num2str(v) ''';'])
32      end
33      for v = 1 : nshocks
34          eval(['shocksnames{' num2str(v) '} = ''Shck ' num2str(v) ''';'])
35      end
36  else
37      if isfield(options,'varnames') ==1
38          varnames = options.varnames;
39      end
40      if isfield(options,'shocksnames') == 1
41          shocksnames = options.shocksnames;
42          if length(shocksnames) ~= nshocks
43              error('There is a mismatch between the number of shocks and the
                    names')
44          end
45      else
46          for v = 1 : nshocks
47              eval(['shocksnames{' num2str(v) '} = ''Shck ' varnames{v} ''';'])
48          end
49      end
50      if isfield(options,'normz') ==1 && options.normz==1,
51          normz_yes = 1;
52      end
53      if isfield(options,'conf_sig') ==1;
54          conf_sig = options.conf_sig;
55      end
56      if isfield(options,'nplots') ==1;
57          nplots = options.nplots;
58      end
59      if isfield(options,'saveas_strng') ==1;
60          savefig_yes = 1;
61          % setting the names of the figure to save
62          fnam_suffix = [ 'irfs_' options.saveas_strng ];
63          fnam_dir    = '.';
64      end
65      if isfield(options,'saveas_dir') ==1;
66          % setting the folder where to save the figure
67          fnam_dir = options.saveas_dir;
68          if exist(fnam_dir,'dir') == 0
69              mkdir(fnam_dir)
70          end
71      end
72      if isfield(options,'add_irfs') ==1
73          % setting the folder where to save the figure
74          add_irfs = options.add_irfs;
75          add_irfs_yes = 1;
76      end
```

```
77      if isfield(options,'conf_sig_2') ==1
78          add_multiple_bands_yes = 1;
79          sort_idx_2    = round((0.5 + [-options.conf_sig_2, options.conf_sig_2,
                0]/2) * ndraws);
80      end
81
82  end
83
84
85
86  % nfigs   = ceil(length(varnames)/( nplots(1)*nplots(2)) );
87  % nplots = repmat(nplots,nfigs,1);
88  % for j=1:size(nplots,1),
89  %       nbofplots(j)=nplots(j,1)*nplots(j,2);
90  % end
91  %
92  % ntotplots = sum(nbofplots);
93  % if ntotplots<length(varnames),
94  %       nfigplus = ceil((length(pplotvar)-ntotplots)/nbofplots(end));
95  %       lastrow=nplots(end,:);
96  %       lastrow=repmat(lastrow,nfigplus,1);
97  %       nplots = [nplots;lastrow];
98  %       nbofplots = [nbofplots repmat(nbofplots(end),1,nfigplus)];
99  % end
100 %
101 % conf_sig = 0.68;
102 % sort_idx = round((0.5 + [-conf_sig, conf_sig, 0]/2) * options.K);
103 %
104 % sims_shock_down_conf = normz * sims_shock_sort(:, :, :,  sort_idx(1));
105 % sims_shock_up_conf   = normz * sims_shock_sort(:, :, :,  sort_idx(2));
106 % sims_shock_median    = normz * sims_shock_sort(:, :, :,  sort_idx(3));
107
108 if ndraws > 1
109     sort_idx   = round((0.5 + [-conf_sig, conf_sig, 0]/2) * ndraws);
110     irf_sort   = sort(irfs,4);
111     if normz_yes == 1
112         % normalize the IRF relative to a 100 bpt increase in the first
113         % variable, first horizon, first shock
114         normz = 1/ irf_sort(1, 1, 1,  sort_idx(3) );
115     end
116     if sort_idx(1) == 0,
117         sort_idx(1) = 1;
118         warning('IRF_Bands_not_reliable._You_have_too_few_draws.')
119     end
120     irf_Median = normz * squeeze(irf_sort(:, :, :,  sort_idx(3) ));
121     irf_low    = normz * squeeze(irf_sort(:, :, :,  sort_idx(1) ));
122     irf_up     = normz * squeeze(irf_sort(:, :, :,  sort_idx(2) ));
```

```
123     if  add_multiple_bands_yes == 1
124         irf_low_low  = normz * squeeze(irf_sort(:, :, :,  sort_idx_2(1) ));
125         irf_up_up    = normz * squeeze(irf_sort(:, :, :,  sort_idx_2(2) ));
126     end
127
128 else
129     irf_Median = normz * irfs;
130     irf_low    = normz * irfs;
131     irf_up     = normz * irfs;
132 end
133
134 jplot = 0;
135 % jfig  = 0;
136 figure('name',['All_IRFs'] );
137 for sho = 1 : nshocks
138     for var= 1: size(varnames,2)
139
140
141         jplot=jplot+1;
142         subplot(nplots(1),nplots(2),jplot)
143
144         if add_multiple_bands_yes == 1
145             h = area([irf_low_low(var,:,sho)',...
146                 irf_low(var,:,sho)' - irf_low_low(var,:,sho)',...
147                 irf_up(var,:,sho)' - irf_low(var,:,sho)',...
148                 irf_up_up(var,:,sho)' - irf_up(var,:,sho)']);%,'FaceColor
                        ',[.85 .85 .85]);
149             set(h(4),'FaceColor',[.95 .95 .95])
150             set(h(3),'FaceColor',[.85 .85 .85])
151             set(h(2),'FaceColor',[.95 .95 .95])
152             set(h(1),'FaceColor',[1 1 1])
153             set(h,'linestyle','none')
154             hold on
155
156         else
157             h = area([irf_low(var,:,sho)',...
158                 irf_up(var,:,sho)' - irf_low(var,:,sho)']);%,'FaceColor ',[.85
                        .85 .85]);
159             set(h(2),'FaceColor',[.85 .85 .85])
160             set(h(1),'FaceColor',[1 1 1])
161             set(h,'linestyle','none')
162             hold on;
163         end
164
165         plot(irf_Median(var,:,sho),'k');
166         if add_irfs_yes == 1
167             plot(add_irfs(var,:,sho),'b','LineWidth',2);
```

```
        end
%           if add_multiple_bands_yes == 1
%               plot(irf_up_up(var,:,sho),'k:','LineWidth',1.2);
%               plot(irf_low_low(var,:,sho),'k:','LineWidth',1.2);
%           end
        hold on;
        plot(zeros(1,hor),'k')
        hold on;
        hold on
        axis tight
        if jplot <= nvar
            title(varnames{var})
        end
        if jplot == nvar*(sho-1) + 1
            ylabel(shocksnames{sho})
        end
        set(gcf,'position' ,[50 50 800 650])
    end
end
if savefig_yes == 1,
    STR_RECAP = [ fnam_dir '/' fnam_suffix ];
    saveas(gcf,STR_RECAP,'fig');
    saveas(gcf,STR_RECAP,'eps');
    savefigure_pdf([STR_RECAP '.pdf']);
end
```

plot_frcst_.m

```
function plot_frcst_(frcsts,y,time,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 8/08/2019

% input : frcsts
% 1st dimension: horizon
% 2nd dimension: variable
% 3rdth dimension: draws
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if length(time) ~= length(y)
    error('Mismatch between time and in sample data (y)');
end
```

```
18 if size(time,2)>size(time,1)
19     time=time';
20 end
21
22 hor      = size(frcsts,1);
23 nvar     = size(frcsts,2);
24 ndraws   = size(frcsts,3);
25 nplots   = [ceil(sqrt(nvar)) ceil(sqrt(nvar))];
26 savefig_yes = 0;
27 conf_sig    = 0.68;
28 add_frcst_yes = 0;
29 add_multiple_bands_yes = 0;
30 fnam_dir    = '.';
31 fnam_suffix = 'frcsts';
32 trasf_yes   =  0;
33
34 % retreive the frequency
35 integerTest = ~mod(time,1);
36 indx        = find(integerTest==1);
37 frq         = indx(2)-indx(1);
38
39 if frq == 12 % monthly
40        timefor = time(end) + 1/12 : 1/12 : time(end) + hor/12;
41        time = [time; timefor'];
42 elseif frq == 4 % quarterly
43        timefor = time(end) + 1/4 : 1/4 : time(end) + hor/4;
44        time = [time; timefor'];
45 elseif frq == 1 % annual
46        timefor = time(end) + 1 : 1 : time(end) + hor;
47        time = [time; timefor'];
48 elseif frq == 48 % weekly
49        timefor = time(end) + 1/48 : 1/48 : time(end) + hor/48;
50        time = [time; timefor'];
51 else
52     error('Frequency not defined.')
53 end
54 %
55 % if strmatch(freq,'m') == 1 %#ok<*MATCH2>
56 %         timefor = time(end) + 1/12 : 1/12 : time(end) + hor/12;
57 %         time = [time; timefor'];
58 % elseif strmatch(freq,'q') == 1
59 %         timefor = time(end) + 1/4 : 1/4 : time(end) + hor/4;
60 %         time = [time; timefor'];
61 % elseif strmatch(freq,'a') == 1
62 %         timefor = time(end) + 1 : 1 : time(end) + hor;
63 %         time = [time; timefor'];
64 % else
```

```
%     error('You need to provide a frequency: ''m'', ''q'' or ''a''.')
% end
time_start = 1;
time_end = length(time);

if nargin < 4
    disp('You did not provided names for variables.')
    disp('I call them Var 1, Var 2, ...')
    for v = 1 : nvar
        eval(['varnames{'   num2str(v) '} = ''Var ' num2str(v) ''';'])
    end
else
    if isfield(options,'time_start') ==1
        time_start = find(options.time_start==time);
        if isempty(time_start) ==1
            error('''time_start'' is not included in ''T''.')
        end
    end
    if isfield(options,'order_transform') ==1
        trasf_yes       = 1;
        order_trasform  = options.order_transform;
        if length(order_trasform) ~= nvar
            error('Mismatch between the ''order_transform'' size and # of
                variables.')
        end
    end
    if isfield(options,'varnames') ==1
        varnames = options.varnames;
        if length(varnames) ~= nvar
            error('Mismatch between the # varnames and # of variables to plot'
                )
        end
    else
        disp('You did not provided names for the endogenous variables.')
        disp('I call them Var 1, Var 2, ...')
        for v = 1 : nvar
            eval(['varnames{'   num2str(v) '} = ''Var ' num2str(v) ''';'])
        end
    end
    if isfield(options,'nplots') ==1
        nplots = options.nplots;
    end
    if isfield(options,'saveas_strng') ==1
        savefig_yes = 1;
        % setting the names of the figure to save
        fnam_suffix = [ fnam_suffix options.saveas_strng ];
    end
```

```
110     if isfield(options,'saveas_dir') == 1
111         savefig_yes = 1;
112         % setting the folder where to save the figure
113         fnam_dir = options.saveas_dir;
114         if exist(fnam_dir,'dir') == 0
115             mkdir(fnam_dir)
116         end
117     end
118     if isfield(options,'add_frcst') ==1
119         add_frcst = options.add_frcst;
120         if size(add_frcst) ~= [length(time), nvar]
121             error('The ''add_frcst'' dimensions must be in sample + output of
                    sample length and the # of variables to plot')
122         end
123         add_frcst_yes = 1;
124     end
125     if isfield(options,'conf_sig') ==1
126         conf_sig = options.conf_sig;
127     end
128     if isfield(options,'conf_sig_2') ==1
129         if options.conf_sig_2 < conf_sig
130             error('Additional confidence bands should be larger than ''options
                    .conf_sig''.')
131         end
132         add_multiple_bands_yes = 1;
133         sort_idx_2   = round((0.5 + [-options.conf_sig_2, options.conf_sig_2,
                0]/2) * ndraws);
134     end
135 end
136
137 nfigs  = ceil(length(varnames)/( nplots(1)*nplots(2)) );
138 nplots = repmat(nplots,nfigs,1);
139 for j=1:size(nplots,1),
140     nbofplots(j)=nplots(j,1)*nplots(j,2);
141 end
142
143 ntotplots = sum(nbofplots);
144 if ntotplots<length(varnames),
145     nfigplus = ceil((length(pplotvar)-ntotplots)/nbofplots(end));
146     lastrow=nplots(end,:);
147     lastrow=repmat(lastrow,nfigplus,1);
148     nplots = [nplots;lastrow];
149     nbofplots = [nbofplots repmat(nbofplots(end),1,nfigplus)];
150 end
151
152 if ndraws > 1
153     frcsts_ = nan(length(time),nvar,ndraws);
```

```
154     for kk = 1 : ndraws
155         frcsts_(:,:,kk) = [y; frcsts(:,:,kk)];
156     end
157     frcsts = frcsts_;
158
159     sort_idx   = round((0.5 + [-conf_sig, conf_sig, 0]/2) * ndraws);
160
161     if trasf_yes ==1
162         frcsts_ = nan(size(frcsts));
163         for var = 1 : nvar
164             if order_trasform(var) == 1 % period over period
165                 frcsts_(2:end,var,:) = (frcsts(2:end,var,:) - frcsts(1:end-1,
                        var,:)) ;
166
167             elseif order_trasform(var) == 100 % percentage period over period
168                 frcsts_(2:end,var,:) = 100*(frcsts(2:end,var,:) - frcsts(1:end
                        -1,var,:));
169
170             elseif order_trasform(var) == 12 % percentage 12 periord over 12
                    period (year over year % change f  or monthly data)
171                 frcsts_(13:end,var,:) = 100*(frcsts(13:end,var,:) - frcsts(1:
                        end-12,var,:));
172
173             elseif order_trasform(var) == 4 % percentage 4 periord over 4
                    period (year over year % change for quarterly data)
174                 frcsts_(5:end,var,:) = 100*(frcsts(5:end,var,:) - frcsts(1:end
                        -4,var,:));
175
176             elseif order_trasform(var) == 400
177                 frcsts_(2:end,var,:) = 400*(frcsts(2:end,var,:) - frcsts(1:end
                        -1,var,:));
178
179             elseif order_trasform(var) == 1200
180                 frcsts_(2:end,var,:) = 1200*(frcsts(2:end,var,:) - frcsts(1:
                        end-1,var,:));
181
182             else
183                 frcsts_(:,var,:) = frcsts(:,var,:);
184             end
185         end
186         frcsts = frcsts_;
187     end
188
189     frcsts_sort    = sort(frcsts,3);
190     if sort_idx(1) == 0
191         sort_idx(1) = 1;
192         warning('Bands␣not␣reliable.␣You␣have␣too␣few␣draws.')
```

```
193     end
194     irf_Median = squeeze(frcsts_sort(:, :, sort_idx(3) ));
195     irf_low    = squeeze(frcsts_sort(:, :, sort_idx(1) ));
196     irf_up     = squeeze(frcsts_sort(:, :, sort_idx(2) ));
197     if  add_multiple_bands_yes == 1
198         irf_low_low  = squeeze(frcsts_sort(:, :, sort_idx_2(1) ));
199         irf_up_up    = squeeze(frcsts_sort(:, :, sort_idx_2(2) ));
200     end
201
202 else
203     irf_Median = [y; frcsts];
204
205     if trasf_yes ==1
206         irf_Median_ = nan(size(irf_Median));
207         for var = 1 : nvar
208             if order_trasform(var) == 1
209                 irf_Median_(2:end,var) = diff(irf_Median(:,var)) ;
210
211             elseif order_trasform(var) == 12 % percentage 12 periord over 12
                    period (year over year % change f  or monthly data)
212                 irf_Median_(13:end,var) = 100*(irf_Median(13:end,var) -
                        irf_Median(1:end-12,var));
213
214             elseif order_trasform(var) == 4 % percentage 4 periord over 4
                    period (year over year % change for quarterly data)
215                 irf_Median_(5:end,var) = 100*(irf_Median(5:end,var) -
                        irf_Median(1:end-4,var));
216
217             elseif order_trasform(var) == 100
218                 irf_Median_(2:end,var) = 100*diff(irf_Median(:,var));
219
220             elseif order_trasform(var) == 400
221                 irf_Median_(2:end,var) = 400*diff(irf_Median(:,var));
222
223             elseif order_trasform(var) == 1200
224                 irf_Median_(2:end,var) = 1200*diff(irf_Median(:,var));
225
226             else
227                 irf_Median_(:,var) = irf_Median(:,var);
228             end
229         end
230         irf_Median = irf_Median_;
231     end
232     irf_low    = irf_Median;
233     irf_up     = irf_Median;
234
235
```

```
236 end
237
238 if trasf_yes ==1 && add_frcst_yes ==1
239     add_frcst_ = nan(size(add_frcst));
240     for var = 1 : nvar
241         if order_trasform(var) == 1
242             add_frcst_(2:end,var) = diff(add_frcst(:,var)) ;
243
244         elseif order_trasform(var) == 12 % percentage 12 periord over 12
                period (year over year % change for monthly data)
245             add_frcst_(13:end,var) = 100*(add_frcst(13:end,var) - add_frcst(1:
                end-12,var));
246
247 %           elseif order_trasform(var) == 4 % percentage 4 periord over 4 period
     (year over year % change for quarterly data)
248 %               add_frcst_(4:end,var) = 100*(add_frcst(4:end,var) - add_frcst(1:
    end-3,var));
249
250         elseif order_trasform(var) == 4 % percentage 4 periord over 4 period (
                m over m % change for monthly data or YoY for Q data)
251             add_frcst_(5:end,var) = 100*(add_frcst(5:end,var) - add_frcst(1:
                end-4,var));
252
253         elseif order_trasform(var) == 100
254             add_frcst_(2:end,var) = 100*diff(add_frcst(:,var));
255
256         elseif order_trasform(var) == 400
257             add_frcst_(2:end,var) = 400*diff(add_frcst(:,var));
258
259         elseif order_trasform(var) == 1200
260             add_frcst_(2:end,var) = 1200*diff(add_frcst(:,var));
261
262         else
263             add_frcst_(:,var) = add_frcst(:,var);
264         end
265     end
266     add_frcst = add_frcst_;
267 end
268
269
270 jplot = 0;
271 jfig  = 0;
272
273 index_time = time_start:time_end;
274
275 for var= 1: nvar
276
```

```
277     if jplot==0,
278         figure('name',['Forecasts'] );
279         jfig=jfig+1;
280     end
281
282     jplot=jplot+1;
283     subplot(nplots(1),nplots(2),jplot)
284
285
286     if add_multiple_bands_yes == 1
287
288         h = area([time(index_time)],[irf_low_low(index_time,var),...
289             irf_low(index_time,var) - irf_low_low(index_time,var),...
290             irf_up(index_time,var) - irf_low(index_time,var),...
291             irf_up_up(index_time,var) - irf_up(index_time,var)]);%,'FaceColor
                ',[.85 .85 .85]);
292         set(h(4),'FaceColor',[.95 .95 .95])
293         set(h(3),'FaceColor',[.85 .85 .85])
294         set(h(2),'FaceColor',[.95 .95 .95])
295         set(h(1),'FaceColor',[1 1 1])
296         set(h,'linestyle','none')
297         hold on
298         min_=min(irf_low_low(index_time,var));
299         max_=max(irf_up_up(index_time,var));
300     else
301         h = area([time(index_time)],[irf_low(index_time,var),...
302             irf_up(index_time,var) - irf_low(index_time,var)]);%,'FaceColor
                ',[.85 .85 .85]);
303         set(h(2),'FaceColor',[.85 .85 .85])
304         set(h(1),'FaceColor',[1 1 1])
305         set(h,'linestyle','none')
306         min_=min(irf_low(index_time,var));
307         max_=max(irf_up(index_time,var));
308     end
309     hold on;
310     plot(time(index_time),irf_Median(index_time,var),'k');
311     hold on;
312     plot(time(index_time),zeros(length(index_time),1),'k');
313     hold on;
314     axis tight
315     ylim([min_,max_]);
316     if add_frcst_yes == 1
317         plot(time(index_time),add_frcst(index_time,var),'b','LineWidth',2);
318         ylim([min([min_,min(add_frcst(index_time,var))]) , max([max_,max(
              add_frcst(index_time,var))])  ] );
319     end
320     title(varnames{var})
```

```
    set(gcf,'position' ,[50 50 800 650])
    if jplot==nbofplots(jfig) || var==length(varnames)
        %legend(legenda);
        if savefig_yes == 1
            STR_RECAP = [ fnam_dir '/' fnam_suffix  '_' int2str(jfig)];
            saveas(gcf,STR_RECAP,'fig');
            saveas(gcf,STR_RECAP,'eps');
            % savefigure_pdf(STR_RECAP);
            savefigure_pdf([STR_RECAP '.pdf']);
        end
        jplot=0;
    end
end
```

plot_irfs_.m

```
function plot_irfs_(irfs,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
% Revised, 8/08/2019

% input : irfs
% 1st dimension: variable
% 2nd dimension: horizon
% 3rd dimension: shock
% 4th dimension: draws
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nvar    = size(irfs,1);
hor     = size(irfs,2);
nshocks = size(irfs,3);
ndraws  = size(irfs,4);
nplots  = [ceil(sqrt(nvar)) ceil(sqrt(nvar))];
savefig_yes = 0;
conf_sig    = 0.68;
normz       = 1;
add_irfs_yes = 0;
normz_yes   = 0;
add_multiple_bands_yes = 0;
fnam_dir    = '.';
fnam_suffix = 'irfs_';
fontsize    = 12;
```

```
30
31 if nargin <2
32     disp('You did not provided names for shocks nor variables.')
33     disp('I call them Var 1, Var 2, ... and Shck 1, ...')
34     for v = 1 : nvar
35         eval(['varnames{'   num2str(v) '} = ''Var  ' num2str(v) ''';'])
36     end
37     for v = 1 : nshocks
38         eval(['shocksnames{' num2str(v) '} = ''Shck ' num2str(v) ''';'])
39     end
40 else
41     if isfield(options,'varnames') ==1
42         varnames = options.varnames;
43         if length(varnames) ~= nvar
44             error('There is a mismatch between the number of var and the names
                   ')
45         end
46     else
47         disp('You did not provided names for the endogenous variables.')
48         disp('I call them Var 1, Var 2, ...')
49         for v = 1 : nvar
50             eval(['varnames{'   num2str(v) '} = ''Var  ' num2str(v) ''';'])
51         end
52     end
53     if isfield(options,'shocksnames') == 1
54         shocksnames = options.shocksnames;
55         if length(shocksnames) ~= nshocks
56             error('There is a mismatch between the number of shocks and the
                   names')
57         end
58     else
59         disp('You did not provided names for the shocks.')
60         disp('I call them Shck 1,  2, ...')
61         for v = 1 : nshocks
62             eval(['shocksnames{' num2str(v) '} = ''Shck ' num2str(v) ''';'])
63         end
64     end
65     if isfield(options,'normz') ==1 && options.normz==1,
66         normz_yes = 1;
67     end
68     if isfield(options,'conf_sig') ==1;
69         conf_sig = options.conf_sig;
70     end
71     if isfield(options,'nplots') ==1;
72         nplots = options.nplots;
73     end
74     if isfield(options,'saveas_strng') ==1;
```

```
75          savefig_yes = 1;
76          % setting the names of the figure to save
77          fnam_suffix = [ fnam_suffix options.saveas_strng ];
78      end
79      if isfield(options,'saveas_dir') ==1;
80          savefig_yes = 1;
81          % setting the folder where to save the figure
82          fnam_dir = options.saveas_dir;
83          if exist(fnam_dir,'dir') == 0
84              mkdir(fnam_dir)
85          end
86      end
87      if isfield(options,'add_irfs') ==1
88          % setting the folder where to save the figure
89          add_irfs = options.add_irfs;
90          add_irfs_yes = 1;
91      end
92      if isfield(options,'conf_sig_2') ==1
93          add_multiple_bands_yes = 1;
94          sort_idx_2   = round((0.5 + [-options.conf_sig_2, options.conf_sig_2,
                0]/2) * ndraws);
95      end
96      if isfield(options,'fontsize') ==1
97          % title font size
98          fontsize = options.fontsize;
99      end
100
101 end
102
103
104
105 nfigs  = ceil(length(varnames)/( nplots(1)*nplots(2)) );
106 nplots = repmat(nplots,nfigs,1);
107 for j=1:size(nplots,1),
108     nbofplots(j)=nplots(j,1)*nplots(j,2);
109 end
110
111 ntotplots = sum(nbofplots);
112 if ntotplots<length(varnames),
113     nfigplus = ceil((length(pplotvar)-ntotplots)/nbofplots(end));
114     lastrow=nplots(end,:);
115     lastrow=repmat(lastrow,nfigplus,1);
116     nplots = [nplots;lastrow];
117     nbofplots = [nbofplots repmat(nbofplots(end),1,nfigplus)];
118 end
119
120 % conf_sig = 0.68;
```

```
% sort_idx = round((0.5 + [-conf_sig, conf_sig, 0]/2) * options.K);
%
% sims_shock_down_conf = normz * sims_shock_sort(:, :, :,  sort_idx(1));
% sims_shock_up_conf   = normz * sims_shock_sort(:, :, :,  sort_idx(2));
% sims_shock_median    = normz * sims_shock_sort(:, :, :,  sort_idx(3));

if ndraws > 1
    sort_idx   = round((0.5 + [-conf_sig, conf_sig, 0]/2) * ndraws);
    irf_sort   = sort(irfs,4);
    if normz_yes == 1
        % normalize the IRF relative to a 100 bpt increase in the first
        % variable, first horizon, first shock
        normz = 1/ irf_sort(1, 1, 1,  sort_idx(3) );
    end
    if sort_idx(1) == 0,
        sort_idx(1) = 1;
    end
    irf_Median = normz * squeeze(irf_sort(:, :, :,  sort_idx(3) ));
    irf_low    = normz * squeeze(irf_sort(:, :, :,  sort_idx(1) ));
    irf_up     = normz * squeeze(irf_sort(:, :, :,  sort_idx(2) ));
    if  add_multiple_bands_yes == 1
        irf_low_low  = normz * squeeze(irf_sort(:, :, :,  sort_idx_2(1) ));
        irf_up_up    = normz * squeeze(irf_sort(:, :, :,  sort_idx_2(2) ));
    end

else
    irf_Median = normz * irfs;
    irf_low    = normz * irfs;
    irf_up     = normz * irfs;
end

jplot = 0;
jfig  = 0;
for sho = 1 : nshocks
    %     figm = figure('Name',['IRFs of ' deblank(varnames{sho})] );
    for var= 1: nvar %size(varnames,2)

        if jplot==0,
            figure('name',['IRFs of ' shocksnames{sho}] );
            jfig=jfig+1;
        end

        jplot=jplot+1;
        subplot(nplots(1),nplots(2),jplot)

        if add_multiple_bands_yes == 1

```

```
168             h = area([irf_low_low(var,:,sho)',...
169                 irf_low(var,:,sho)' - irf_low_low(var,:,sho)',...
170                 irf_up(var,:,sho)' - irf_low(var,:,sho)',...
171                 irf_up_up(var,:,sho)' - irf_up(var,:,sho)']);%,'FaceColor
                        ',[.85 .85 .85]);
172             set(h(4),'FaceColor',[.95 .95 .95])
173             set(h(3),'FaceColor',[.85 .85 .85])
174             set(h(2),'FaceColor',[.95 .95 .95])
175             set(h(1),'FaceColor',[1 1 1])
176             set(h,'linestyle','none')
177             hold on
178 
179         else
180             h = area([irf_low(var,:,sho)',...
181                 irf_up(var,:,sho)' - irf_low(var,:,sho)']);%,'FaceColor ',[.85
                        .85 .85]);
182             set(h(2),'FaceColor',[.85 .85 .85])
183             set(h(1),'FaceColor',[1 1 1])
184             set(h,'linestyle','none')
185         end
186         %           hold on
187         %           plot(irf.cholesky.up(var,:,sho),'b:');
188         %           hold on;
189         %           plot(irf.cholesky.low(var,:,sho),'b:');
190         hold on;
191         plot(irf_Median(var,:,sho),'k');
192         hold on;
193         if add_irfs_yes == 1
194             for hh = 1: size(add_irfs,4)
195                 plot(add_irfs(var,:,sho,hh),'b','LineWidth',2);
196             end
197         end
198         hold on;
199         plot(zeros(1,hor),'k')
200         hold on;
201         hold on
202         axis tight
203         title(varnames{var},'FontSize',fontsize)
204         set(gcf,'position' ,[50 50 800 650])
205         if jplot==nbofplots(jfig) || var==length(varnames)
206             %legend(legenda);
207             if savefig_yes == 1
208                 STR_RECAP = [ fnam_dir '/' fnam_suffix '_' shocksnames{sho} '_
                        ' int2str(jfig)];
209                 saveas(gcf,STR_RECAP,'fig');
210                 saveas(gcf,STR_RECAP,'eps');
211                 %savefigure_pdf(STR_RECAP);
```

```
                    savefigure_pdf([STR_RECAP '.pdf']);
                end
                jplot=0;
            end
        end
        jfig = 0;
    end
```

plot_sdcmp_.m

```
function  plot_sdcmp_(input,BVAR,options)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nshocks = BVAR.N;
for v = 1 : nshocks
    eval(['namesshock{' num2str(v) '} = ''Shck' num2str(v) ''';'])
end
for v = 1 : BVAR.N
    eval(['pplotvar{'  num2str(v) '} = ''Var' num2str(v) ''';'])
end
for v = 1 : BVAR.N
    eval(['ex_names_{'  num2str(v) '} = {''Shck' num2str(v) '''};'])
end
ex_names_ = ex_names_';
for v = 1 : BVAR.N
    eval(['leg{'  num2str(v) '} = ''Shck' num2str(v) ''';'])
end
leg{end+1} = 'initial condition';

TT   = 1:1:size(input,1);
Tlim   = [TT(1) TT(end)];

[~,positions] = ismember(pplotvar,BVAR.varnames);
dcmp_group_yes = 0;
dmcp_type      = 'stacked';
tags           = [];
colors_decomp_yes = 0;
savefig_yes = 0;
% initial_state_dcmp = 0;
```

```
36 % addplot_=0;
37 % addplot0_=0;
38
39 if nargin > 2
40     if isfield(options,'plotvar_') ==1
41         clear plotvar_
42         pplotvar             = options.plotvar_;
43     end
44     if isfield(options,'snames_') ==1
45         clear ex_names_
46         ex_names_             = options.snames_;
47     end
48     if isfield(options,'stag_') ==1
49         clear leg
50         leg             = options.stag_;
51     end
52     if isfield(options,'snames_') ==1 && isfield(options,'stag_') == 0
53         error('You need to provide also ''stag_''')
54     end
55 %     if isfield(options,'snames_') == 0 && isfield(options,'stag_') == 1
56 %         error('YOu need to provide also ''snames_''')
57 %     end
58     if size(ex_names_,1)+1 ~= length(leg)
59         error('Mismatch between shock aggregations and shocks names')
60     end
61     if isfield(options,'time') ==1
62         TT = options.time;
63         if isfield(options,'Tlim') ==1;
64             Tlim=options.Tlim;
65             if Tlim(1) < TT(1)
66                 warning('You have set a intitial date that starts earlier than
                        the first obs');
67                 warning('I change it with the frist obs');
68                 Tlim(1) = TT(1);
69             end
70             if Tlim(2) > TT(end)
71                 warning('You have set a final date that exceeds the forecast
                        horizon');
72                 warning('I change it with the endo of forecast');
73                 Tlim(2) = TT(end);
74             end
75         end
76
77     end
78
79     if isfield(options,'save_strng') ==1
80         tags = options.save_strng;
```

```
81      end
82      if isfield(options,'dcmp_grouped') ==1 && options.dcmp_grouped ==1
83          dcmp_group_yes = 1;
84          dmcp_type      = 'grouped';
85      end
86      if isfield(options,'plotvarnames') ==1
87          pplotvarname = options.plotvarnames;
88          if length(pplotvarname)~=length(pplotvar)
89              error('The number of plot titles (pplotvarname) and of plot
                    variables needs to coincide')
90          end
91      end
92      if isfield(options,'colors_decomp') ==1 && options.colors_decomp==1
93          colors_decomp_yes =1;
94      end
95      if isfield(options,'colors_decomp') ==1 && options.colors_decomp==2
96          colors_decomp_yes =2;
97      end
98      if isfield(options,'colors_decomp') ==1 && options.colors_decomp==3
99          colors_decomp_yes =3;
100     end
101     if isfield(options,'saveas_dir') ==1
102         savefig_yes = 1;
103         %   setting the folder where to save the figure
104         fnam_dir = options.saveas_dir;
105         if exist(fnam_dir,'dir') == 0
106             mkdir(fnam_dir)
107         end
108     end
109
110     %      if isfield(options,'addplots_yes') ==1  && options.addplots_yes==1,
111     %          tags = [ tags '_memo'];
112     %          if isfield(input,'frcsts') ==1
113     %              addplot_=1;
114     %              frcsts = input.frcsts.states.Mean(:,positions);
115     %              s_s = input.frcsts.states.steady(positions);
116     %          end
117     %          if isfield(input,'frcsts0') ==1
118     %              addplot0_=1;
119     %              frcsts0 = input.frcsts0.states.Mean(:,positions);
120     %
121     %          end
122     %      end
123 end
124
125 [~,positions] = ismember(pplotvar,BVAR.varnames);
126 pplotvarname  = pplotvar;
```

```
127
128 deco  = input;
129
130 % setting the names of the figure to save
131 fnam_suffix = [tags '_shcks_dcmp'];
132
133 ngroups0 = size(ex_names_,1);
134 % ngroups  = ngroups0+1+no_initial_effect;
135 if colors_decomp_yes == 0
136     func = @(x) colorspace('RGB->Lab',x);
137     MAP = distinguishable_colors(ngroups0+1,'w',func);
138     % MAP = CreateColorMap(ngroups0+1);
139     MAP(end,:) = [0.7 0.7 0.7];
140 elseif colors_decomp_yes == 1
141     MAP = zeros(4,3);
142     MAP(end,:) = [0.7 0.7 0.7]; % gray
143     MAP(end-1,:) = [0.2 0.5 0.99];       % blue
144     MAP(end-2,:) =  [1 1 0];         % yellow
145     MAP(end-3,:) = [0.8 0 0.8] ;             % purple
146     %      MAP(end-4,:) = [0  0.7  0];           % green
147     %      MAP(end-5,:) = [1  0  0];             % red
148     %      MAP(end-6,:) = [0  0  1];             % blue
149     %      MAP(end-7,:) = [.5  .5  0];           % light green
150 elseif colors_decomp_yes == 2
151     MAP = zeros(10,3);
152     MAP(end,:) = [0.7 0.7 0.7]; % gray
153     MAP(end-1,:) = [0.2 0.5 0.99];        % blue
154     MAP(end-2,:) = [0.8 0 0.8] ;         % purple
155     MAP(end-3,:) = [1 1 0] ;             % yellow
156     MAP(end-4,:) = [0 0.7 0];           % green
157     MAP(end-5,:) = [1 0 0];              % red
158     MAP(end-6,:) = [0 0 1];              % blue
159     MAP(end-7,:) = [.5 .5 0];           % light green
160     MAP(end-8,:) = [0 0.25 0]; %brown
161     MAP(end-9,:) = [0 0.9 0.9]; % violet
162 elseif colors_decomp_yes == 3
163     MAP = zeros(11,3);
164     MAP(end,:) = [0.7 0.7 0.7]; % gray
165     MAP(end-1,:) = [0.2 0.5 0.99];        % blue
166     MAP(end-2,:) = [0.8 0 0.8] ;         % purple
167     MAP(end-3,:) = [1 1 0] ;             % yellow
168     MAP(end-4,:) = [0 0.7 0];           % green
169     MAP(end-5,:) = [1 0 0];              % red
170     MAP(end-6,:) = [0 0 1];              % blue
171     MAP(end-7,:) = [.5 .5 0];           % light green
172     MAP(end-8,:) = [0 0.25 0]; %brown
173     MAP(end-9,:) = [0 0.9 0.9]; % violet
```

```
174     MAP(end-10,:) = [0.5 0.1 0.1]; % violet
175 end
176
177 st = find(Tlim(1)==TT);
178 en = find(Tlim(2)==TT);
179 if savefig_yes == 1
180     fidTxt = fopen([fnam_dir '\legenda_' fnam_suffix '_plots.txt'],'w');
181     fprintf(fidTxt,['LEGENDA ' tags ' SHOCK DECOMPOSITION PLOTS\n']);
182     fprintf(fidTxt,['\n']);
183 end
184
185 for j = 1 : size(pplotvar,2)
186     clear sdec sdec_tot,
187     indx = positions(j);
188     sdec0 = squeeze(deco(:,indx,:));
189     for i=1:ngroups0
190         clear index,
191         for ii=1:size(ex_names_{i},2)
192             indbuf = strmatch(ex_names_{i}{ii},namesshock,'exact');
193             if ~isempty(indbuf)
194                 index(ii) = indbuf;
195             elseif ~isempty(ex_names_{i}{ii})
196                 error(['Shock name ',ex_names_{i}{ii}, ' not found.' ]);
197             end
198         end
199         sdec(:,i)=sum(sdec0(:,index),2);
200         sdec0(:,index)=0;
201     end
202
203     h= figure('Name',['Shocks Decomposition for ' pplotvarname{j}]);
204     sdec_tot=[sdec,  sum(sdec0,2)];
205     if dcmp_group_yes == 0
206         ind_pos = (sdec_tot>0);
207         ind_neg = (sdec_tot<0);
208         temp_neg  = cumsum(sdec_tot.*ind_neg ,2).*ind_neg;
209         temp_pos  = cumsum(sdec_tot.*ind_pos ,2).*ind_pos;
210         temp      = temp_neg + temp_pos;
211         for kk = size(temp,2) : -1 : 1
212             hold on
213             bbar = bar(TT(st:en),temp(st:en,kk),dmcp_type,'EdgeColor',[0 0 0])
                    ;
214             set(bbar, 'FaceColor', MAP(kk,:))
215             shading faceted; hold on;
216         end
217         leg0 = leg(end:-1:1);
218
219     else
```

```
220         temp       = sdec_tot;
221         bbar = bar(TT(st:en),temp(st:en,:),dmcp_type,'EdgeColor',[0 0 0]);
                colormap(MAP);
222         %          hleg=legend(leg(1:1:end),'interpreter','none','location','
                Best');
223         %          shading faceted;
224         %
225     end
226     %      set(hleg,'position',[0.5  0.15  0.4  0.2],'units','normalized')
227     hold on
228     axis tight
229     fillips = sum(sdec_tot,2);
230     hold on, h1=plot(TT(st:en),fillips(st:en),'k-d');
231     set(h1,'MarkerFaceColor', 'k')
232
233     %      if addplot_ ==1
234     %          hold on, h1=plot(TT(st:en),frcsts(st:en,j),'k-*','LineWidth',2);
235     %          hold on, h1=plot(TT(st:en),s_s(j)*ones(length(TT(st:en)),1),'k
            -.','LineWidth',2);
236     %          leg0{end+1} = 'Current Forecast';
237     %          leg0{end+1} = 'steady state';
238     %      end
239     %
240     %      if addplot0_ ==1
241     %          hold on, h1=plot(TT(st:en),frcsts0(st:en,j),'r-o','LineWidth',2)
            ;
242     %          leg0{end+1} = 'Previous Forecast';
243     %      end
244
245     set(gcf,'position' ,[50 50 800 650])
246     hleg=legend(leg0,'interpreter','none','location','Best');
247     shading faceted;
248
249     %      define_.timestart + (define_.nobs-1)/4
250     %plot the last in-+sample obs
251     %      hold on; plot([TT(define_.nobs) TT(define_.nobs)],[low up],'color
            ',[1 0 0],'LineWidth',2)
252     %      h=vline(define_.timestart + (define_.nobs-1)/4, 'k', 'Last Obs');
253     %      set(h,'Linewidth',2)
254
255     title(pplotvarname{j})
256     %      set(gca,'Xtick',TT(st:6:en))
257     %      tmp_str= sample2date(TT(st:6:en));
258     set(gca,'Xtick',TT(st:6:en))
259     %      tmp_str= sample2date(TT(st:2:en));
260     %      set(gca,'Xticklabel',tmp_str)
261     %      tmp_str= sample2date(TT);
```

```
262     %     STR_RECAP = [ 'model_' fnam_suffix '_' tmp_str{st} '_' tmp_str{end}
            '_' int2str(j)];
263
264     if savefig_yes == 1
265         STR_RECAP = [ fnam_dir '/svar_' fnam_suffix '_' int2str(j)];
266         saveas(gcf,STR_RECAP,'fig');
267         saveas(gcf,STR_RECAP,'pdf');
268
269         fprintf(fidTxt,['The␣figure␣sdcmp_' fnam_suffix '_' int2str(j) '␣
                contains␣the␣following␣variable:\n' ]);
270         tmp = strrep(char(pplotvarname{j} ),'\','␣');
271         fprintf(fidTxt,[tmp ';␣']);
272
273         fprintf(fidTxt,['\n' ]);
274         fprintf(fidTxt,['\n']);
275         close all
276     end
277
278 end
279
280 if savefig_yes == 1
281     fclose(fidTxt);
282 end
283
284 end
```

quer.m

```
1  function quer(str);
2  % function quer(str);
3  % Query user
4  % if str=='c' Continue or not
5  % if str=='g' close Graphs or not
6  str=upper(str);
7  switch str
8  case 'C'
9
10     button=questdlg('Continue?␣Y/N',...
11         'Continue␣Program',...
12         'Yes','No','Yes');
13     switch button
14     case 'No'
15         error('Program␣execution␣stopped');
16     case 'Yes'
17         fprintf('%-30s␣\n','␣␣');
```

```
18      end
19  case 'G'
20      button=questdlg('Close All Graphs Y/N?','Graphs','No','Yes','Yes');
21      switch button
22      case 'No'
23          fprintf('%-30%s \n','  ');
24      case 'Yes'
25          close all;
26      end
27  end
```

rand_inverse_wishart.m

```
1   function G = rand_inverse_wishart(m, v, H_inv_upper_chol)
2
3   % function G = rand_inverse_wishart(m, v, H_inv_upper_chol)
4   % rand_inverse_wishart  Pseudo random matrices drawn from an
5   % inverse Wishart distribution
6   % G = rand_inverse_wishart(m, v, H_inv_upper_chol)
7   % Returns an m-by-m matrix drawn from an inverse-Wishart distribution.
8   %
9   % INPUTS:
10  %     m:              dimension of G and H_inv_upper_chol.
11  %     v:              degrees of freedom, greater or equal than m.
12  %     H_inv_chol:     upper cholesky decomposition of the inverse of the
13  %                     matrix parameter.
14  %                     The upper cholesky of the inverse is requested here
15  %                     in order to avoid to recompute it at every random draw.
16  %                     H_inv_upper_chol = chol(inv(H))
17  % OUTPUTS:
18  %     G:              G ~ IW(m, v, H) where H = inv(H_inv_upper_chol'*
        H_inv_upper_chol)
19  %                     or, equivalently, using the correspondence between Wishart
        and
20  %                     inverse-Wishart: inv(G) ~ W(m, v, S) where
21  %                     S = H_inv_upper_chol'*H_inv_upper_chol = inv(H)
22  %
23  % SPECIAL REQUIREMENT
24  %     none
25  %
26
27  % Copyright (C) 2003-2009 Dynare Team
28  %
29  % This file is part of Dynare.
30  %
```

```
% Dynare is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% Dynare is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with Dynare.  If not, see <http://www.gnu.org/licenses/>.

X = randn(v, m) * H_inv_upper_chol;


% At this point, X'*X is Wishart distributed
% G = inv(X'*X);

% Rather compute inv(X'*X) using the SVD
[U,S,V] = svd(X, 0);
SSi = 1 ./ (diag(S) .^ 2);
G = (V .* repmat(SSi', m, 1)) * V';
```

reorderVAR.m

```
function [Phip,Sigmap] = reorderVAR(Phi,Sigma,reordering)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

% Permute the autoregressive matrix and the variance coveriance of the
% shocks with the new ordering of the variables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N            = size(Sigma,1);

if length(reordering) ~= N,
    error('Not enough permutations: the vector with new ordering of variables must have size N.')
end

Phip    = Phi(:,reordering);
```

```
Sigmap  = nan(N);

for jj  = 1 : length(reordering)
    for hh = 1 : length(reordering)

        tmp = Sigma(reordering(jj),reordering(hh));
        Sigmap(jj,hh) = tmp;
    end

end

end
```

rescaleFAVAR.m

```
function [ReScale] = rescaleFAVAR(STD,Lambda,n_1,order_pc)

if nargin < 4
    % principal component ordered first
    order_pc = 1;
end

n_w  = size(Lambda,2);
n_2  = size(Lambda,1);

ReScale= NaN;

if size(STD,2)>size(STD,1)
    STD =STD';
end


switch order_pc
    case 1 % factor first

        Scale_  = repmat([STD; ones(n_1,1)], 1, n_w+n_1);
        Lambda_ = [Lambda zeros(n_2 , n_1); zeros(n_1,n_w+n_1)];
        Lambda_(n_2+1 : n_2+n_1 ,  n_w + 1 : n_w+n_1) = eye(n_1);
        ReScale = Scale_ .* Lambda_ ;

    case 2 % factor second

        Scale_  = repmat([ones(n_1,1); STD], 1, n_w +n_1);
        Lambda_ = [zeros(n_1,n_1+n_w); zeros(n_2 , n_1) Lambda];
        Lambda_(1:n_1, 1:n_1) = eye(n_1);
```

```
31          ReScale = Scale_ .* Lambda_ ;
32
33  end
```

rfvar3.m

```
1   function var=rfvar3(ydata,lags,xdata,breaks,lambda,mu)
2   %function var=rfvar3(ydata,lags,xdata,breaks,lambda,mu)
3   % This algorithm goes for accuracy without worrying about memory requirements.
4   % ydata:    dependent variable data matrix
5   % xdata:    exogenous variable data matrix
6   % lags:     number of lags
7   % breaks:   rows in ydata and xdata after which there is a break.  This allows
        for
8   %           discontinuities in the data (e.g. war years) and for the
        possibility of
9   %           adding dummy observations to implement a prior.  This must be a
        column vector.
10  %           Note that a single dummy observation becomes lags+1 rows of the
        data matrix,
11  %           with a break separating it from the rest of the data.  The function
         treats the
12  %           first lags observations at the top and after each "break" in ydata
        and xdata as
13  %           initial conditions.
14  % lambda:   weight on "co-persistence" prior dummy observations.  This
        expresses
15  %           belief that when data on *all* y's are stable at their initial
        levels, they will
16  %           tend to persist at that level.  lambda=5 is a reasonable first try.
          With lambda<0,
17  %           constant term is not included in the dummy observation, so that
        stationary models
18  %           with means equal to initial ybar do not fit the prior mean.  With
        lambda>0, the prior
19  %           implies that large constants are unlikely if unit roots are present
        .
20  % mu:       weight on "own persistence" prior dummy observation.  Expresses
        belief
21  %           that when y_i has been stable at its initial level, it will tend to
         persist
22  %           at that level, regardless of the values of other variables.  There
        is
23  %           one of these for each variable.  A reasonable first guess is mu=2.
24  %       The program assumes that the first lags rows of ydata and xdata are
```

```
    real data, not dummies.
% Dummy observations should go at the end, if any. If pre-sample x's are not available,
% repeating the initial xdata(lags+1,:) row or copying xdata(lags+1:2*lags,:) into
% xdata(1:lags,:) are reasonable subsititutes. These values are used in forming the
% persistence priors.

% Original file downloaded from:
% http://sims.princeton.edu/yftp/VARtools/matlab/rfvar3.m

[T,nvar] = size(ydata);
nox = isempty(xdata);
if ~nox
    [T2,nx] = size(xdata);
else
    T2 = T;
    nx = 0;
    xdata = zeros(T2,0);
end
% note that x must be same length as y, even though first part of x will not be used.
% This is so that the lags parameter can be changed without reshaping the xdata matrix.
if T2 ~= T, error('Mismatch of x and y data lengths'),end
if nargin < 4
    nbreaks = 0;
    breaks = [];
else
    nbreaks = length(breaks);
end
breaks = [0;breaks;T];
smpl = [];
for nb = 1:nbreaks+1
    smpl = [smpl;[breaks(nb)+lags+1:breaks(nb+1)]'];
end
Tsmpl = size(smpl,1);
X = zeros(Tsmpl,nvar,lags);
for is = 1:length(smpl)
    X(is,:,:) = ydata(smpl(is)-(1:lags),:)';
end
X = [X(:,:) xdata(smpl,:)];
y = ydata(smpl,:);
% Everything now set up with input data for y=Xb+e

% Add persistence dummies
```

```
66  if lambda ~= 0 || mu > 0
67      ybar = mean(ydata(1:lags,:),1);
68      if ~nox
69          xbar = mean(xdata(1:lags,:),1);
70      else
71          xbar = [];
72      end
73      if lambda ~= 0
74          if lambda>0
75              xdum = lambda*[repmat(ybar,1,lags) xbar];
76          else
77              lambda = -lambda;
78              xdum = lambda*[repmat(ybar,1,lags) zeros(size(xbar))];
79          end
80          ydum = zeros(1,nvar);
81          ydum(1,:) = lambda*ybar;
82          y = [y;ydum];
83          X = [X;xdum];
84      end
85      if mu>0
86          xdum = [repmat(diag(ybar),1,lags) zeros(nvar,nx)]*mu;
87          ydum = mu*diag(ybar);
88          X = [X;xdum];
89          y = [y;ydum];
90      end
91  end
92
93  % Compute OLS regression and residuals
94  [vl,d,vr] = svd(X,0);
95  di = 1./diag(d);
96  B = (vr.*repmat(di',nvar*lags+nx,1))*vl'*y;
97  u = y-X*B;
98  xxi = vr.*repmat(di',nvar*lags+nx,1);
99  xxi = xxi*xxi';
100
101 var.B = B;
102 var.u = u;
103 var.xxi = xxi;
104 var.y   = y;
105 var.X   = X;
```

savefigure_pdf.m

```
1  function [ figformat ] = savefigure_pdf( name,varargin )
2
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

% SAVEFIGURE_PDF: this function saves figures in .pdf format
% name: Name of the figure, example: 'figure1'
% type: Format of the figure, example: 'fig','espc'
% If no input is provided, figure saved twice with fig and espc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set(gcf,'Units','Inches');
pos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3),
    pos(4)])
print(gcf,name,'-dpdf','-r0');
figformat=1;
end
```

shade.m

```
function []=shade(start,finish,colorstr,up,low);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

% function []=shade(start,finish,colorstr);
%
%  start and finish are Nx1 vectors of starting and ending years.
%  The function shades between the start and finish pairs using colorstr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ~exist('colorstr'); colorstr='y'; end;  % default is yellow
curax=axis;
y=[curax(3)+low up*curax(4) up*curax(4) curax(3)+low];
hold on;
for i=1:length(start);
  x=[start(i) start(i) finish(i) finish(i)];
   fill(x,y,colorstr);
end;

% Now, prevent the shading from covering up the lines in the plot.
h = findobj(gca,'Type','line');
```

```
set(h,'EraseMode','xor');

h = findobj(gca,'Type','patch');
set(h,'EdgeColor','none');

% This last one makes the tick marks visible
set(gca, 'Layer', 'top')
```

sign2matrix.m

```
function [f,sr] = sign2matrix(signs,ny)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

yr  = ones(ny);
ys  = ones(ny);
y   = nan(ny);
for ell = 1 : length(signs)
    eval(signs{ell})
end

f(1:ny , 1:ny)       = ys;
f(1+ny:2*ny , 1:ny)  = yr;
sr                   = y;
```

standard.m

```
function [x,Scale]=standard(y)
T=size(y,1);
my=repmat(nanmean(y),T,1);
sy=repmat(nanstd(y),T,1);
x=(y-my)./sy;
x(find(isnan(x)))=randn;
Scale = nanstd(y)';
%x=(y-kron(mean(y),ones(rows(y),1)))./kron(std(y),ones(rows(y),1));
```

var2ss.m

```
function [A,B,C,const,Sigma,lags,index_var]=var2ss(Phi,Sigma,index)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% var2ss coverts the VAR into a state space system of this form
% x(t) = A x(t-1) + B Sigma' u(t) ~ N(0,I)
% y(t) = C*(cons + x(t-1))
% where A is the companion form of the lag struture

% Filippo Ferroni, 6/1/2015
% Revised, 2/15/2017
% Revised, 3/21/2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 3
    %% all stocks
    index = zeros(size(Sigma,1),1);
end

Sigma = chol(Sigma);

N           = size(Sigma,1);
[m , n]     = size(Phi);
lags        = (m-1)/n;

% companion form
A       = [Phi(1 : N * lags, :)'; eye(N*(lags-1), N*lags)];
B       = eye(N * lags, N);
C       = eye(N, N * lags);

IminusAlags = eye(N);
for ell = 1 : lags
    IminusAlags = IminusAlags - Phi(N * (ell -1) + 1 : N * ell, :)';
end
iIminusAlags  = inv(IminusAlags);

const    = [iIminusAlags*Phi(end, :)'; zeros(N*(lags-1), 1)];

% index =0 % stock:        xq(t) = xm(t)
% index =1 % TBA
% index =2 % deflator/real flow:    xq(t) = 1/3( xm(t) +  xm(t-1) +  xm(t-2))

index_var = 1 : N;

for vv = 1 : N
```

```
45      if index(vv) == 1
46  %         C(vv,vv : N : N * 3) = 1;
47      elseif index(vv) == 2 %
48          if lags < 2
49              error('When you specify the flow/deflation aggregation, you need
                    at least 2 lags')
50          end
51          A                       = [A zeros(size(A,1),1)];
52          A                       = [A; zeros(1,size(A,2))];
53          A(end,vv : N : N * 2)   = 1/3;
54          Ao                      = eye(length(A));
55          Ao(end,vv)              = -1/3;
56          iAo                     = inv(Ao);
57          A                       = iAo * A;
58
59          B = [B; zeros(1,size(B,2))];
60          B = iAo * B;
61
62          C  = [C zeros(N,1)];
63          C(vv,vv)  = 0;
64          C(vv,end) = 1;
65
66          const(size(A,1)) = const(vv);
67
68          index_var(vv) = size(A,1);
69
70      elseif index(vv) == 4 %
71          if lags < 3
72              error('When you specify the flow/deflation aggregation (weekly-
                    monthly or quarterly-annual), you need at least 3 lags')
73          end
74          A                       = [A zeros(size(A,1),1)];
75          A                       = [A; zeros(1,size(A,2))];
76          A(end,vv : N : N * 3)   = 1/4;
77          Ao                      = eye(length(A));
78          Ao(end,vv)              = -1/4;
79          iAo                     = inv(Ao);
80          A                       = iAo * A;
81
82          B = [B; zeros(1,size(B,2))];
83          B = iAo * B;
84
85          C  = [C zeros(N,1)];
86          C(vv,vv)  = 0;
87          C(vv,end) = 1;
88
89          const(size(A,1)) = const(vv);
```

```
90
91          index_var(vv) = size(A,1);
92
93      end
94
95  end
```

varprior.m

```
1   function [ydum,xdum,breaks]=varprior(nv,nx,lags,mnprior,vprior)
2   %function [ydum,xdum,breaks]=varprior(nv,nx,lags,mnprior,vprior)
3   % ydum, xdum:    dummy observation data that implement the prior
4   % breaks:        vector of points in the dummy data after which new dummy obs's
        start
5   %                   Set breaks=T+[0;breaks], ydata=[ydata;ydum], xdum=[xdata;
        xdum], where
6   %                   actual data matrix has T rows, in preparing input for
        rfvar3
7   % nv,nx,lags: VAR dimensions
8   % mnprior.tight: Overall tightness of Minnesota prior
9   % mnprior.decay: Standard deviations of lags shrink as lag^(-decay)
10  % vprior.sig:    Vector of prior modes for diagonal elements of r.f. covariance
        matrix
11  % vprior.w:      Weight on prior on vcv.  1 corresponds to "one dummy
        observation" weight
12  %                   Should be an integer, and will be rounded if not.  vprior.
        sig is needed
13  %                   to scale the Minnesota prior, even if the prior on sigma
        is not used itself.
14  %                   Set vprior.w=0 to achieve this.
15  % Note:          The original Minnesota prior treats own lags asymmetrically,
        and therefore
16  %                   cannot be implemented entirely with dummy observations.
        It is also usually
17  %                   taken to include the sum-of-coefficients and co-
        persistence components
18  %                   that are implemented directly in rfvar3.m.  The diagonal
        prior on v, combined
19  %                   with sum-of-coefficients and co-persistence components and
        with the unit own-first-lag
20  %                   prior mean generates larger prior variances for own than
        for cross-effects even in
21  %                   this formulation, but here there is no way to shrink
        toward a set of unconstrained
22  %                   univariate AR's.
```

```
23
24 % Original file downloaded from:
25 % http://sims.princeton.edu/yftp/VARtools/matlab/varprior.m
26
27 if ~isempty(mnprior)
28     xdum = zeros(lags+1,nx,lags,nv);
29     ydum = zeros(lags+1,nv,lags,nv);
30     for il = 1:lags
31         ydum(il+1,:,il,:) = il^mnprior.decay*diag(vprior.sig);
32     end
33     ydum(1,:,1,:) = diag(vprior.sig);
34     ydum = mnprior.tight*reshape(ydum,[lags+1,nv,lags*nv]);
35     ydum = flipdim(ydum,1);
36     xdum = mnprior.tight*reshape(xdum,[lags+1,nx,lags*nv]);
37     xdum = flipdim(xdum,1);
38     breaks = (lags+1)*[1:(nv*lags)]';
39     lbreak = breaks(end);
40 else
41     ydum = [];
42     xdum = [];
43     breaks = [];
44     lbreak = 0;
45 end
46 % elle = 0;
47 % for j1 = 1 : nv
48 %     for j2 = 1 : lags
49 %         elle = 1 + elle;
50 %         ydum(:,:,elle) = ydum(:,:,elle) * mnprior.unit_root_(j1);
51 %     end
52 % end
53 if ~isempty(vprior) && vprior.w>0
54     ydum2 = zeros(lags+1,nv,nv);
55     xdum2 = zeros(lags+1,nx,nv);
56     ydum2(end,:,:) = diag(vprior.sig);
57     for i = 1:vprior.w
58         ydum = cat(3,ydum,ydum2);
59         xdum = cat(3,xdum,xdum2);
60         breaks = [breaks;(lags+1)*[1:nv]'+lbreak];
61         lbreak = breaks(end);
62     end
63 end
64 dimy = size(ydum);
65 ydum = reshape(permute(ydum,[1 3 2]),dimy(1)*dimy(3),nv);
66 xdum = reshape(permute(xdum,[1 3 2]),dimy(1)*dimy(3),nx);
67 breaks = breaks(1:(end-1));
```

cmintools/bfgsi.m

```
function H = bfgsi(H0,dg,dx)
% H = bfgsi(H0,dg,dx)
% dg is previous change in gradient; dx is previous change in x;
% 6/8/93 version that updates inverse hessian instead of hessian
% itself.
% Copyright by Christopher Sims 1996.  This material may be freely
% reproduced and modified.
if size(dg,2)>1
   dg=dg';
end
if size(dx,2)>1
   dx=dx';
end
Hdg = H0*dg;
dgdx = dg'*dx;
if (abs(dgdx) >1e-12)
   H = H0 + (1+(dg'*Hdg)/dgdx)*(dx*dx')/dgdx - (dx*Hdg'+Hdg*dx')/dgdx;
else
   disp('bfgs update failed.')
   disp(['|dg| = ' num2str(sqrt(dg'*dg)) '  |dx| = ' num2str(sqrt(dx'*dx))]);
   disp(['dg''*dx = ' num2str(dgdx)])
   disp(['|H*dg| = ' num2str(Hdg'*Hdg)])
   H=H0;
end
save H.dat H
```

cmintools/csminit.m

```
function [fhat,xhat,fcount,retcode] = csminit(fcn,x0,f0,g0,badg,H0,varargin)
% [fhat,xhat,fcount,retcode] = csminit(fcn,x0,f0,g0,badg,H0,...
%                                      P1,P2,P3,P4,P5,P6,P7,P8)
% retcodes: 0, normal step.  5, largest step still improves too fast.
% 4,2 back and forth adjustment of stepsize didn't finish.  3, smallest
% stepsize still improves too slow.  6, no improvement found.  1, zero
% gradient.
%----------------------
% Modified 7/22/96 to omit variable-length P list, for efficiency and
    compilation.
% Places where the number of P's need to be altered or the code could be
    returned to
% its old form are marked with ARGLIST comments.
%
% Fixed 7/17/93 to use inverse-hessian instead of hessian itself in bfgs
```

```
14 % update.
15 %
16 % Fixed 7/19/93 to flip eigenvalues of H to get better performance when
17 % it's not psd.
18 %
19 %tailstr = ')';
20 %for i=nargin-6:-1:1
21 %    tailstr=[ ',P' num2str(i)  tailstr];
22 %end
23 %ANGLE = .03;
24 ANGLE = .005;
25 %THETA = .03;
26 THETA = .3; %(0<THETA<.5) THETA near .5 makes long line searches, possibly
       fewer iterations.
27 FCHANGE = 1000;
28 MINLAMB = 1e-9;
29 % fixed 7/15/94
30 % MINDX = .0001;
31 % MINDX = 1e-6;
32 MINDFAC = .01;
33 fcount=0;
34 lambda=1;
35 xhat=x0;
36 f=f0;
37 fhat=f0;
38 g = g0;
39 gnorm = norm(g);
40 %
41 if (gnorm < 1.e-12) & ~badg % put ~badg 8/4/94
42    retcode =1;
43    dxnorm=0;
44    % gradient convergence
45 else
46    % with badg true, we don't try to match rate of improvement to directional
47    % derivative.  We're satisfied just to get some improvement in f.
48    %
49    %if(badg)
50    %    dx = -g*FCHANGE/(gnorm*gnorm);
51    %  dxnorm = norm(dx);
52    %  if dxnorm > 1e12
53    %      disp('Bad, small gradient problem.')
54    %      dx = dx*FCHANGE/dxnorm;
55    %   end
56    %else
57    % Gauss-Newton step;
58    %---------- Start of 7/19/93 mod ---------------
59    %[v d] = eig(H0);
```

```
60      %toc
61      %d=max(1e-10,abs(diag(d)));
62      %d=abs(diag(d));
63      %dx = -(v.*(ones(size(v,1),1)*d'))*(v'*g);
64  %       toc
65      dx = -H0*g;
66  %       toc
67      dxnorm = norm(dx);
68      if dxnorm > 1e12
69         disp('Near-singular H problem.')
70         dx = dx*FCHANGE/dxnorm;
71      end
72      dfhat = dx'*g0;
73      %end
74      %
75      %
76      if ~badg
77         % test for alignment of dx with gradient and fix if necessary
78         a = -dfhat/(gnorm*dxnorm);
79         if a<ANGLE
80            dx = dx - (ANGLE*dxnorm/gnorm+dfhat/(gnorm*gnorm))*g;
81            % suggested alternate code: ------------------------
82            dx = dx*dxnorm/norm(dx)     % This keeps scale invariant to the angle
                  correction
83            %-------------------------------------------------------
84            dfhat = dx'*g;
85            % dxnorm = norm(dx);  % this line unnecessary with modification that
                  keeps scale invariant
86            disp(sprintf('Correct for low angle: %g',a))
87         end
88      end
89      disp(sprintf('Predicted improvement: %18.9f',-dfhat/2))
90      %
91      % Have OK dx, now adjust length of step (lambda) until min and
92      % max improvement rate criteria are met.
93      done=0;
94      factor=3;
95      shrink=1;
96      lambdaMin=0;
97      lambdaMax=inf;
98      lambdaPeak=0;
99      fPeak=f0;
100     lambdahat=0;
101     while ~done
102        if size(x0,2)>1
103           dxtest=x0+dx'*lambda;
104        else
```

```
105            dxtest=x0+dx*lambda;
106         end
107         % home
108         f = feval(fcn,dxtest,varargin{:});
109         %ARGLIST
110         %f = feval(fcn,dxtest,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13);
111         % f = feval(fcn,x0+dx*lambda,P1,P2,P3,P4,P5,P6,P7,P8);
112         disp(sprintf('lambda = %10.5g; f = %20.7f',lambda,f ))
113         %debug
114         %disp(sprintf('Improvement too great? f0-f: %g, criterion: %g',f0-f,-(1-
                THETA)*dfhat*lambda))
115         if f<fhat
116            fhat=f;
117            xhat=dxtest;
118            lambdahat = lambda;
119         end
120         fcount=fcount+1;
121         shrinkSignal = (~badg & (f0-f < max([-THETA*dfhat*lambda 0]))) | (badg &
                (f0-f) < 0) ;
122         growSignal = ~badg & ( (lambda > 0)  &  (f0-f > -(1-THETA)*dfhat*lambda)
                );
123         if  shrinkSignal  &   ( (lambda>lambdaPeak) | (lambda<0) )
124            if (lambda>0) & ((~shrink) | (lambda/factor <= lambdaPeak))
125               shrink=1;
126               factor=factor^.6;
127               while lambda/factor <= lambdaPeak
128                  factor=factor^.6;
129               end
130               %if (abs(lambda)*(factor-1)*dxnorm < MINDX) | (abs(lambda)*(factor
                     -1) < MINLAMB)
131               if abs(factor-1)<MINDFAC
132                  if abs(lambda)<4
133                     retcode=2;
134                  else
135                     retcode=7;
136                  end
137                  done=1;
138               end
139            end
140            if (lambda<lambdaMax) & (lambda>lambdaPeak)
141               lambdaMax=lambda;
142            end
143            lambda=lambda/factor;
144            if abs(lambda) < MINLAMB
145               if (lambda > 0) & (f0 <= fhat)
146                  % try going against gradient, which may be inaccurate
147                  lambda = -lambda*factor^6
```

```
148             else
149                 if lambda < 0
150                     retcode = 6;
151                 else
152                     retcode = 3;
153                 end
154                 done = 1;
155             end
156         end
157     elseif  (growSignal & lambda>0) |  (shrinkSignal & ((lambda <=
            lambdaPeak) & (lambda>0)))
158         if shrink
159             shrink=0;
160             factor = factor^.6;
161             %if  ( abs(lambda)*(factor-1)*dxnorm< MINDX ) | ( abs(lambda)*(
                    factor-1)< MINLAMB)
162             if abs(factor-1)<MINDFAC
163                 if abs(lambda)<4
164                     retcode=4;
165                 else
166                     retcode=7;
167                 end
168                 done=1;
169             end
170         end
171         if ( f<fPeak ) & (lambda>0)
172             fPeak=f;
173             lambdaPeak=lambda;
174             if lambdaMax<=lambdaPeak
175                 lambdaMax=lambdaPeak*factor*factor;
176             end
177         end
178         lambda=lambda*factor;
179         if abs(lambda) > 1e20;
180             retcode = 5;
181             done =1;
182         end
183     else
184         done=1;
185         if factor < 1.2
186             retcode=7;
187         else
188             retcode=0;
189         end
190     end
191   end
192 end
```

```
193 disp(sprintf('Norm of dx %10.5g', dxnorm))
```

cmintools/csminwel.m

```
1  function [fh,xh,gh,H,itct,fcount,retcodeh] = csminwel(fcn,x0,H0,grad,crit,nit,
       varargin)
2  %[fhat,xhat,ghat,Hhat,itct,fcount,retcodehat] = csminwel(fcn,x0,H0,grad,crit,
       nit,varargin)
3  % fcn:    string naming the objective function to be minimized
4  % x0:     initial value of the parameter vector
5  % H0:     initial value for the inverse Hessian.  Must be positive definite.
6  % grad:   Either a string naming a function that calculates the gradient, or
       the null matrix.
7  %         If it's null, the program calculates a numerical gradient.  In this
       case fcn must
8  %         be written so that it can take a matrix argument and produce a row
       vector of values.
9  % crit:   Convergence criterion.  Iteration will cease when it proves
       impossible to improve the
10 %         function value by more than crit.
11 % nit:    Maximum number of iterations.
12 % varargin: A list of optional length of additional parameters that get handed
        off to fcn each
13 %         time it is called.
14 %         Note that if the program ends abnormally, it is possible to retrieve
       the current x,
15 %         f, and H from the files g1.mat and H.mat that are written at each
       iteration and at each
16 %         hessian update, respectively.  (When the routine hits certain kinds
       of difficulty, it
17 %         write g2.mat and g3.mat as well.  If all were written at about the
       same time, any of them
18 %         may be a decent starting point.  One can also start from the one with
        best function value.)
19 [nx,no]=size(x0);
20 nx=max(nx,no);
21 Verbose=1;
22 NumGrad= isempty(grad);
23 done=0;
24 itct=0;
25 fcount=0;
26 snit=100;
27 %tailstr = ')';
28 %stailstr = [];
29 % Lines below make the number of Pi's optional.  This is inefficient, though,
```

```
    and precludes
% use of the matlab compiler.  Without them, we use feval and the number of Pi's must be
% changed with the editor for each application.  Places where this is required are marked
% with ARGLIST comments
%for i=nargin-6:-1:1
%    tailstr=[ ',P' num2str(i)  tailstr];
%    stailstr=[' P' num2str(i) stailstr];
%end
f0 = feval(fcn,x0,varargin{:});
%ARGLIST
%f0 = feval(fcn,x0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13);
% disp('first fcn in csminwel.m----------') % Jinill on 9/5/95
if f0 > 1e50, disp('Bad initial parameter.'), return, end
if NumGrad
   if length(grad)==0
      [g badg] = numgrad(fcn,x0, varargin{:});
      %ARGLIST
      %[g badg] = numgrad(fcn,x0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13);
   else
      badg=any(find(grad==0));
      g=grad;
   end
   %numgrad(fcn,x0,P1,P2,P3,P4);
else
   [g badg] = feval(grad,x0,varargin{:});
   %ARGLIST
   %[g badg] = feval(grad,x0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13);
end
retcode3=101;
x=x0;
f=f0;
H=H0;
cliff=0;
while ~done
   g1=[]; g2=[]; g3=[];
   %addition fj. 7/6/94 for control
   disp('----------')
   disp('----------')
   %disp('f and x at the beginning of new iteration')
   disp(sprintf('f at the beginning of new iteration, %20.10f',f))
   %----------Comment out this line if the x vector is long----------
      disp([sprintf('x = ') sprintf('%15.8g %15.8g %15.8g %15.8g\n',x)]);
   %------------------------
   itct=itct+1;
   [f1 x1 fc retcode1] = csminit(fcn,x,f,g,badg,H,varargin{:});
```

```
74      %ARGLIST
75      %[f1 x1 fc retcode1] = csminit(fcn,x,f,g,badg,H,P1,P2,P3,P4,P5,P6,P7,...
76      %            P8,P9,P10,P11,P12,P13);
77      % itct=itct+1;
78      fcount = fcount+fc;
79      % erased on 8/4/94
80      % if (retcode == 1) | (abs(f1-f) < crit)
81      %    done=1;
82      % end
83      % if itct > nit
84      %    done = 1;
85      %    retcode = -retcode;
86      % end
87      if retcode1 ~= 1
88         if retcode1==2 | retcode1==4
89            wall1=1; badg1=1;
90         else
91            if NumGrad
92               [g1 badg1] = numgrad(fcn, x1,varargin{:});
93               %ARGLIST
94               %[g1 badg1] = numgrad(fcn, x1,P1,P2,P3,P4,P5,P6,P7,P8,P9,...
95               %                  P10,P11,P12,P13);
96            else
97               [g1 badg1] = feval(grad,x1,varargin{:});
98               %ARGLIST
99               %[g1 badg1] = feval(grad, x1,P1,P2,P3,P4,P5,P6,P7,P8,P9,...
100              %                  P10,P11,P12,P13);
101           end
102           wall1=badg1;
103           % g1
104           save g1 g1 x1 f1 varargin;
105           %ARGLIST
106           %save g1 g1 x1 f1 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13;
107        end
108        if wall1 & (length(H) > 1)%
109           % Bad gradient or back and forth on step length.  Possibly at
110           % cliff edge.  Try perturbing search direction if problem not 1D
111           %
112           %fcliff=fh; xcliff=xh;
113           Hcliff=H+diag(diag(H).*rand(nx,1));
114           disp('Cliff.  Perturbing search direction.')
115           [f2 x2 fc retcode2] = csminit(fcn,x,f,g,badg,Hcliff,varargin{:});
116           %ARGLIST
117           %[f2 x2 fc retcode2] = csminit(fcn,x,f,g,badg,Hcliff,P1,P2,P3,P4,...
118           %     P5,P6,P7,P8,P9,P10,P11,P12,P13);
119           fcount = fcount+fc; % put by Jinill
120           if  f2 < f
```

```
121             if retcode2==2 | retcode2==4
122                   wall2=1; badg2=1;
123             else
124                if NumGrad
125                   [g2 badg2] = numgrad(fcn, x2,varargin{:});
126                   %ARGLIST
127                   %[g2 badg2] = numgrad(fcn, x2,P1,P2,P3,P4,P5,P6,P7,P8,...
128                   %       P9,P10,P11,P12,P13);
129                else
130                   [g2 badg2] = feval(grad,x2,varargin{:});
131                   %ARGLIST
132                   %[g2 badg2] = feval(grad,x2,P1,P2,P3,P4,P5,P6,P7,P8,...
133                   %       P9,P10,P11,P12,P13);
134                end
135                wall2=badg2;
136                % g2
137                badg2
138                save g2 g2 x2 f2 varargin
139                %ARGLIST
140                %save g2 g2 x2 f2 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13;
141             end
142             if wall2
143                disp('Cliff again.  Try traversing')
144                if norm(x2-x1) < 1e-13
145                   f3=f; x3=x; badg3=1;retcode3=101;
146                else
147                   gcliff=((f2-f1)/((norm(x2-x1))^2))*(x2-x1);
148                   if(size(x0,2)>1), gcliff=gcliff', end
149                   [f3 x3 fc retcode3] = csminit(fcn,x,f,gcliff,0,eye(nx),
                         varargin{:});
150                   %ARGLIST
151                   %[f3 x3 fc retcode3] = csminit(fcn,x,f,gcliff,0,eye(nx),P1,
                         P2,P3,...
152                   %          P4,P5,P6,P7,P8,...
153                   %       P9,P10,P11,P12,P13);
154                   fcount = fcount+fc; % put by Jinill
155                   if retcode3==2 | retcode3==4
156                      wall3=1; badg3=1;
157                   else
158                      if NumGrad
159                         [g3 badg3] = numgrad(fcn, x3,varargin{:});
160                         %ARGLIST
161                         %[g3 badg3] = numgrad(fcn, x3,P1,P2,P3,P4,P5,P6,P7,P8
                               ,...
162                         %                              P9,P10,P11,P12,P13);
163                      else
164                         [g3 badg3] = feval(grad,x3,varargin{:});
```

```
165                         %ARGLIST
166                         %[g3 badg3] = feval(grad,x3,P1,P2,P3,P4,P5,P6,P7,P8
                                ,...
167                         %                              P9,P10,P11,P12,P13);
168                      end
169                      wall3=badg3;
170                      % g3
171                      badg3
172                      save g3 g3 x3 f3 varargin;
173                      %ARGLIST
174                      %save g3 g3 x3 f3 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12
                             P13;
175                   end
176                end
177             else
178                f3=f; x3=x; badg3=1; retcode3=101;
179             end
180          else
181             f3=f; x3=x; badg3=1;retcode3=101;
182          end
183       else
184          % normal iteration, no walls, or else 1D, or else we're finished here
                .
185          f2=f; f3=f; badg2=1; badg3=1; retcode2=101; retcode3=101;
186       end
187    else
188       f2=f;f3=f;f1=f;retcode2=retcode1;retcode3=retcode1;
189    end
190    %how to pick gh and xh
191    if f3 < f - crit & badg3==0
192       ih=3
193       fh=f3;xh=x3;gh=g3;badgh=badg3;retcodeh=retcode3;
194    elseif f2 < f - crit & badg2==0
195       ih=2
196       fh=f2;xh=x2;gh=g2;badgh=badg2;retcodeh=retcode2;
197    elseif f1 < f - crit & badg1==0
198       ih=1
199       fh=f1;xh=x1;gh=g1;badgh=badg1;retcodeh=retcode1;
200    else
201       [fh,ih] = min([f1,f2,f3]);
202       disp(sprintf('ih = %d',ih))
203       %eval(['xh=x' num2str(ih) ';'])
204       switch ih
205          case 1
206             xh=x1;
207          case 2
208             xh=x2;
```

```
209             case 3
210                xh=x3;
211          end %case
212          %eval(['gh=g' num2str(ih) ';'])
213          %eval(['retcodeh=retcode' num2str(ih) ';'])
214          retcodei=[retcode1,retcode2,retcode3];
215          retcodeh=retcodei(ih);
216          if exist('gh')
217             nogh=isempty(gh);
218          else
219             nogh=1;
220          end
221          if nogh
222             if NumGrad
223                [gh badgh] = numgrad(fcn,xh,varargin{:});
224             else
225                [gh badgh] = feval(grad, xh,varargin{:});
226             end
227          end
228          badgh=1;
229       end
230       %end of picking
231       %ih
232       %fh
233       %xh
234       %gh
235       %badgh
236       stuck = (abs(fh-f) < crit);
237       if (~badg)&(~badgh)&(~stuck)
238          H = bfgsi(H,gh-g,xh-x);
239       end
240       if Verbose
241          disp('-----')
242          disp(sprintf('Improvement on iteration %d = %18.9f',itct,f-fh))
243       end
244       % if Verbose
245          if itct > nit
246             disp('iteration count termination')
247             done = 1;
248          elseif stuck
249             disp('improvement < crit termination')
250             done = 1;
251          end
252          rc=retcodeh;
253          if rc == 1
254             disp('zero gradient')
255          elseif rc == 6
```

```
256           disp('smallest step still improving too slow, reversed gradient')
257        elseif rc == 5
258           disp('largest step still improving too fast')
259        elseif (rc == 4) | (rc==2)
260           disp('back and forth on step length never finished')
261        elseif rc == 3
262           disp('smallest step still improving too slow')
263        elseif rc == 7
264           disp('warning: possible inaccuracy in H matrix')
265        end
266     % end
267     f=fh;
268     x=xh;
269     g=gh;
270     badg=badgh;
271  end
272  % what about making an m-file of 10 lines including numgrad.m
273  % since it appears three times in csminwel.m
```

cmintools/csolve.m

```
1  function [x,rc] = csolve(FUN,x,gradfun,crit,itmax,varargin)
2  %function [x,rc] = csolve(FUN,x,gradfun,crit,itmax,varargin)
3  % FUN should be written so that any parametric arguments are packed in to x,
4  % and so that if presented with a matrix x, it produces a return value of
5  % same dimension of x.  The number of rows in x and FUN(x) are always the
6  % same.  The number of columns is the number of different input arguments
7  % at which FUN is to be evaluated.
8  %
9  % gradfun:  string naming the function called to evaluate the gradient matrix.
       If this
10 %           is null (i.e. just "[]"), a numerical gradient is used instead.
11 % crit:     if the sum of absolute values that FUN returns is less than this,
12 %           the equation is solved.
13 % itmax:    the solver stops when this number of iterations is reached, with
       rc=4
14 % varargin: in this position the user can place any number of additional
       arguments, all
15 %           of which are passed on to FUN and gradfun (when it is non-empty)
       as a list of
16 %           arguments following x.
17 % rc:       0 means normal solution, 1 and 3 mean no solution despite
       extremely fine adjustments
18 %           in step length (very likely a numerical problem, or a
       discontinuity). 4 means itmax
```

```
19 %            termination.
20 %---------- delta ----------
21 % differencing interval for numerical gradient
22 delta = 1e-6;
23 %---------------------------------------
24 %---------- alpha ----------
25 % tolerance on rate of descent
26 alpha=1e-3;
27 %---------------------------------------
28 %---------- verbose ----------
29 verbose=1;% if this is set to zero, all screen output is suppressed
30 %---------------------------------------
31 %---------- analyticg ----------
32 analyticg=1-isempty(gradfun); %if the grad argument is [], numerical
       derivatives are used.
33 %---------------------------------------
34 nv=length(x);
35 tvec=delta*eye(nv);
36 done=0;
37 if isempty(varargin)
38    f0=feval(FUN,x);
39 else
40    f0=feval(FUN,x,varargin{:});
41 end
42 af0=sum(abs(f0));
43 af00=af0;
44 itct=0;
45 while ~done
46    if itct>3 & af00-af0<crit*max(1,af0) & rem(itct,2)==1
47       randomize=1;
48    else
49       if ~analyticg
50          if isempty(varargin)
51             grad = (feval(FUN,x*ones(1,nv)+tvec)-f0*ones(1,nv))/delta;
52          else
53             grad = (feval(FUN,x*ones(1,nv)+tvec,varargin{:})-f0*ones(1,nv))/
                   delta;
54          end
55       else % use analytic gradient
56          grad=feval(gradfun,x,varargin{:});
57       end
58       if isreal(grad)
59          if rcond(grad)<1e-12
60             grad=grad+tvec;
61          end
62          dx0=-grad\f0;
63          randomize=0;
```

```
64          else
65              if(verbose),disp('gradient imaginary'),end
66              randomize=1;
67          end
68      end
69      if randomize
70          if(verbose),fprintf(1,'\n Random Search'),end
71          dx0=norm(x)./randn(size(x));
72      end
73      lambda=1;
74      lambdamin=1;
75      fmin=f0;
76      xmin=x;
77      afmin=af0;
78      dxSize=norm(dx0);
79      factor=.6;
80      shrink=1;
81      subDone=0;
82      while ~subDone
83          dx=lambda*dx0;
84          f=feval(FUN,x+dx,varargin{:});
85          af=sum(abs(f));
86          if af<afmin
87              afmin=af;
88              fmin=f;
89              lambdamin=lambda;
90              xmin=x+dx;
91          end
92          if ((lambda >0) & (af0-af < alpha*lambda*af0)) | ((lambda<0) & (af0-af <
                0) )
93              if ~shrink
94                  factor=factor^.6;
95                  shrink=1;
96              end
97              if abs(lambda*(1-factor))*dxSize > .1*delta;
98                  lambda = factor*lambda;
99              elseif (lambda > 0) & (factor==.6) %i.e., we've only been shrinking
100                 lambda=-.3;
101             else %
102                 subDone=1;
103                 if lambda > 0
104                     if factor==.6
105                         rc = 2;
106                     else
107                         rc = 1;
108                     end
109                 else
```

```
110                 rc=3;
111              end
112           end
113        elseif (lambda >0) & (af-af0 > (1-alpha)*lambda*af0)
114           if shrink
115              factor=factor^.6;
116              shrink=0;
117           end
118           lambda=lambda/factor;
119        else % good value found
120           subDone=1;
121           rc=0;
122        end
123     end % while ~subDone
124     itct=itct+1;
125     if(verbose)
126        fprintf(1,'\nitct %d, af %g, lambda %g, rc %g',itct,afmin,lambdamin,rc)
127        fprintf(1,'\n    x  %10g %10g %10g %10g',xmin);
128        fprintf(1,'\n    f  %10g %10g %10g %10g',fmin);
129     end
130     x=xmin;
131     f0=fmin;
132     af00=af0;
133     af0=afmin;
134     if itct >= itmax
135        done=1;
136        rc=4;
137     elseif af0<crit;
138        done=1;
139        rc=0;
140     end
141  end
```

cmintools/initialize_mh.m

```
1  % initialize the MH by minimizing the - logposterior kernel
2  T = length(y);
3  vSwitch = ones(2, 1);
4  % % The parameters might be bounded, but the minimization routine will not
5  % % care, thus we need to make sure that the routine can go anywhere and
6  % % that we are still able to stay within the bounds.
7  % x0 = boundsINV(guess);
8  x0 = guess;
9  % posterior maximization
10 [fh, xh, gh, H, itct, fcount, retcodeh] = csminwel('logpostkernel',x0,.01*eye(
```

```
    length(x0)),[],10e-5,1000,centered,y, vSwitch, iPriorScale);
% processing the output of the maximization
postmode = xh;
JJ       = jacob(xh);
HH       = JJ * H * JJ';
```

cmintools/numgrad.m

```
function [g, badg] = numgrad(fcn,x,varargin)
% function [g badg] = numgrad(fcn,x,varargin)
%
%delta = 1e-6;
delta=1e-6;
n=length(x);
tvec=delta*eye(n);
g=zeros(n,1);
%--------------------old way to deal with variable # of P's----------------
%tailstr = ')';
%stailstr = [];
%for i=nargin-2:-1:1
%    tailstr=[ ',P' num2str(i)  tailstr];
%    stailstr=[' P' num2str(i) stailstr];
%end
%f0 = eval([fcn '(x' tailstr]); % Is there a way not to do this?
%-------------------------------------------------------------------------^yes
f0 = feval(fcn,x,varargin{:});
% disp(' first fcn in numgrad.m ----------------------')
%home
% disp('numgrad.m is working. -----') % Jiinil on 9/5/95
% sizex=size(x),sizetvec=size(tvec),x,     % Jinill on 9/6/95
badg=0;
for i=1:n
   scale=1; % originally 1
   % i,tveci=tvec(:,i)% ,plus=x+scale*tvec(:,i) % Jinill Kim on 9/6/95
   if size(x,1)>size(x,2)
      tvecv=tvec(i,:);
   else
      tvecv=tvec(:,i);
   end
   g0 = (feval(fcn,x+scale*tvecv', varargin{:}) - f0) ...
         /(scale*delta);
   % disp(' fcn in the i=1:n loop of numgrad.m ----------------------')% Jinill
       9/6/95
   % disp('           and i is ')                % Jinill
   % i                          % Jinill
```

```
37      % fprintf('Gradient w.r.t. %3d: %10g\n',i,g0) %see below Jinill 9/6/95
38  %------------------------- special code to essentially quit here
39      % absg0=abs(g0) % Jinill on 9/6/95
40      if abs(g0)< 1e15
41          g(i)=g0;
42          % disp('good gradient') % Jinill Kim
43      else
44          disp('bad gradient ----------------------------') % Jinill Kim
45          % fprintf('Gradient w.r.t. %3d: %10g\n',i,g0) %see above
46          g(i)=0;
47          badg=1;
48          % return
49          % can return here to save time if the gradient will never be
50          % used when badg returns as true.
51      end
52  end
53  %-----------------------------------------------------------------
54  %       if g0 > 0
55  %           sided=2;
56  %           g1 = -(eval([fcn '(x-scale*tvec(:,i)''' tailstr]) - f0) ...
57  %               /(scale*delta);
58  %           if g1<0
59  %               scale = scale/10;
60  %           else
61  %               break
62  %           end
63  %       else
64  %           sided=1;
65  %           break
66  %       end
67  %   end
68  %   if sided==1
69  %       g(i)=g0;
70  %   else
71  %       if (g0<1e20)
72  %           if (g1>-1e20)
73  %               g(i)=(g0+g1)/2;
74  %           else
75  %               g(i)=0;
76  %               badg=1;
77  %               disp( ['Banging against wall, parameter ' int2str(i)] );
78  %           end
79  %       else
80  %           if g1>-1e20
81  %               if g1<0
82  %                   g(i)=0;
83  %                   badg=1;
```

```
84 %                 disp( ['Banging against wall, parameter ' int2str(i)] );
85 %             else
86 %                 g(i)=g1;
87 %             end
88 %         else
89 %             g(i)=0;
90 %             badg=1;
91 %             disp(['Valley around parameter ' int2str(i)])
92 %         end
93 %     end
94 %  end
95 %end
96 %save g.dat g x f0
97 %eval(['save g g x f0 ' stailstr]);
```