This is additional material for the paper

`Identification of DSGE models - The effect of second order approximation and pruning`

The paper establishes rank criteria for local identification given the pruned state-space representation in the fashion of Iskrev (2010) and Qu and Tkachenko (2012), also including higher-order moments, cumulants and polyspectra. It is shown that this may improve overall identification of a DSGE model via imposing additional restrictions on the moments and spectra.

In the Matlab code the user can choose in a graphical-user-interface between the models, the tests, which parameters to identify at which local point, analytical or numerical derivatives, and the order of approximation. Since all procedures are model independent, other models can be easily included and tested as long as they can be represented in the same framework.
How to run:

1. You will need Matlab's symbolic toolbox
2. Make sure to be in the main directory
3. Just run identification_run.m all options are asked via a GUI

As mentioned in the paper here is some additional material.

1 Magnus-Neudecker definition of Hessian

Define the steady state as $\overline{xy} := (\overline{x}', \overline{y}', \overline{x}', \overline{y}')'$, then the Jacobian $\mathcal{D}f(\overline{z})$ and Hessian $\mathcal{H}f(\overline{z})$ of f evaluated at the steady-state are defined as:

$$f(\overline{xy}) = \begin{pmatrix} f^1(\overline{xy}) \\ \vdots \\ f^n(\overline{xy}) \end{pmatrix}$$

$$\mathcal{D}f(\overline{xy}) := \begin{pmatrix} \frac{\partial f(\overline{xy})}{\partial x'_{t+1}} & \frac{\partial f(\overline{xy})}{\partial y'_{t+1}} & \frac{\partial f(\overline{xy})}{\partial x'_t} & \frac{\partial f(\overline{xy})}{\partial y'_t} \end{pmatrix} = \begin{pmatrix} \mathcal{D}f^1(\overline{xy}) \\ \vdots \\ \mathcal{D}f^n(\overline{xy}) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial f^1(\overline{xy})}{\partial x'_{t+1}} & \frac{\partial f^1(\overline{xy})}{\partial y'_{t+1}} & \frac{\partial f^1(\overline{xy})}{\partial x'_t} & \frac{\partial f^1(\overline{xy})}{\partial y'_t} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f^n(\overline{xy})}{\partial x'_{t+1}} & \frac{\partial f^n(\overline{xy})}{\partial y'_{t+1}} & \frac{\partial f^n(\overline{xy})}{\partial x'_t} & \frac{\partial f^n(\overline{xy})}{\partial y'_t} \end{pmatrix}$$

$$\mathcal{H}f(\overline{xy}) := \mathcal{D}vec((\mathcal{D}f(\overline{xy}))') = \begin{pmatrix} \mathcal{H}f^1(\overline{xy}) \\ \vdots \\ \mathcal{H}f^n(\overline{xy}) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial^2 f^1(\overline{xy})}{\partial x_{t+1}\partial x_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_{t+1}\partial y_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_{t+1}\partial x_t'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_{t+1}\partial y_t'} \\ \frac{\partial^2 f^1(\overline{xy})}{\partial y_{t+1}\partial x_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_{t+1}\partial y_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_{t+1}\partial x_t'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_{t+1}\partial y_t'} \\ \frac{\partial^2 f^1(\overline{xy})}{\partial x_t\partial x_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_t\partial y_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_t\partial x_t'} & \frac{\partial^2 f^1(\overline{xy})}{\partial x_t\partial y_t'} \\ \frac{\partial^2 f^1(\overline{xy})}{\partial y_t\partial x_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_t\partial y_{t+1}'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_t\partial x_t'} & \frac{\partial^2 f^1(\overline{xy})}{\partial y_t\partial y_t'} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f^n(\overline{xy})}{\partial x_{t+1}\partial x_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_{t+1}\partial y_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_{t+1}\partial x_t'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_{t+1}\partial y_t'} \\ \frac{\partial^2 f^n(\overline{xy})}{\partial y_{t+1}\partial x_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_{t+1}\partial y_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_{t+1}\partial x_t'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_{t+1}\partial y_t'} \\ \frac{\partial^2 f^n(\overline{xy})}{\partial x_t\partial x_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_t\partial y_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_t\partial x_t'} & \frac{\partial^2 f^n(\overline{xy})}{\partial x_t\partial y_t'} \\ \frac{\partial^2 f^n(\overline{xy})}{\partial y_t\partial x_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_t\partial y_{t+1}'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_t\partial x_t'} & \frac{\partial^2 f^n(\overline{xy})}{\partial y_t\partial y_t'} \end{pmatrix}.$$

f is of dimension $n \times 1$, the Jacobian $Df(\overline{z})$ of dimension $n \times (2n_x + 2n_y)$ and the Hessian $Hf(\overline{z})$ of dimension $n(2n_x + 2n_y) \times (2n_x + 2n_y)$.

2 Example for notation and index matrices

When separating matrices and especially Jacobians into states and shocks, we use index matrices to keep track of the corresponding positions of terms. For illustration, consider only the transition of states with $n_x = 2$ and $n_u = 1$. For $i, j = 1, 2$ denote $h^j_{x_i} := \frac{\partial h^j(\bar{x}_1, \bar{x}_2, 0)}{\partial x_{i,t-1}}$, $h^j_{x_i u} := \frac{\partial^2 h^j(\bar{x}_1, \bar{x}_2, 0)}{\partial x_{i,t-1}\partial u_t}$, where j corresponds to the j-th row of h_v. Similar notation applies for $h^j_u, h^j_{ux_i}, h^j_{x_i u}$ and h^j_{uu}. The solution matrices for states are given by

$$h_v = \begin{bmatrix} h^1_{x_1} & h^1_{x_2} & h^1_u \\ h^2_{x_1} & h^2_{x_2} & h^2_u \\ 0 & 0 & 0 \end{bmatrix}, \qquad h_{vv} = \begin{bmatrix} h^1_{x_1x_1} & h^1_{x_1x_2} & h^1_{x_1u} \\ h^1_{x_2x_1} & h^1_{x_2x_2} & h^1_{x_2u} \\ h^1_{ux_1} & h^1_{ux_2} & h^1_{uu} \\ h^2_{x_1x_1} & h^2_{x_1x_2} & h^2_{x_1u} \\ h^2_{x_2x_1} & h^2_{x_2x_2} & h^2_{x_2u} \\ h^2_{ux_1} & h^2_{ux_2} & h^2_{uu} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

In order to use notation of Andreasen et al (2014) we get rid of the zeros and reshape and permute these matrices to get

$$H_{xx} = \begin{bmatrix} h^1_{x_1x_1} & h^1_{x_2x_1} & h^1_{x_1x_2} & h^1_{x_2x_2} \\ h^2_{x_1x_1} & h^2_{x_2x_1} & h^2_{x_1x_2} & h^2_{x_2x_2} \end{bmatrix}$$

$$H_{xu} = \begin{bmatrix} h^1_{x_1u} & h^1_{x_2u} \\ h^2_{x_1u} & h^2_{x_2u} \end{bmatrix} H_{ux} = \begin{bmatrix} h^1_{ux_1} & h^1_{ux_2} \\ h^2_{ux_1} & h^2_{ux_2} \end{bmatrix} H_{uu} = \begin{bmatrix} h^1_{uu} \\ h^2_{uu} \end{bmatrix}$$

This can be accomplished by using the following matrices indicating the positions in h_{vv}:

$$idx_{H_{xx}} = \begin{bmatrix} 1 & 2 & 10 & 11 \\ 4 & 5 & 13 & 14 \end{bmatrix}, \quad idx_{H_{uu}} = \begin{bmatrix} 21 \\ 24 \end{bmatrix}$$
$$idx_{H_{xu}} = \begin{bmatrix} 19 & 20 \\ 22 & 23 \end{bmatrix}, \quad idx_{H_{ux}} = \begin{bmatrix} 3 & 12 \\ 6 & 15 \end{bmatrix}$$

That is, in order to compute e.g. H_{xx} we simply select the corresponding terms from h_{vv} using $idx_{H_{xx}}$. Since we now know the exact positions, we are further able to select the correct rows of $\mathtt{d}h_{vv}$ to compute $\mathtt{d}H_{xx}$.

In summary the quasi-Matlab-codes are:

```
ind.hv = reshape(1:nv^2,nv,nv);
ind.hx = ind.hv(1:nx,1:nx);
ind.hu = ind.hv(1:nx,(nx+1):end);
ind.hvv = reshape(1:nv^3,[nv^2 nv]);
ind.hvv_tensor = permute(reshape(ind.hvv,[nv nv nv]),[2 1 3]);
ind.hxx = ind.hvv_tensor(1:nx,1:nx,1:nx);
ind.huu = ind.hvv_tensor(1:nx,(nx+1):end,(nx+1):end);
ind.hux = ind.hvv_tensor(1:nx,(nx+1):end,1:nx);
ind.hxu = ind.hvv_tensor(1:nx,1:nx,(nx+1):end);
ind.Hxx = reshape(ind.hxx,nx,nx*nx);
ind.Hxu = reshape(ind.hxu,nx,nx*nu);
ind.Hux = reshape(ind.hux,nx,nu*nx);
ind.Huu = reshape(ind.huu,nx,nu*nu);
hx  = hv(ind.hx);   hu  = hv(ind.hu);
Hxx = hvv(ind.Hxx); Hxu = hvv(ind.Hxu);
Hux = hvv(ind.Hux); Huu = hvv(ind.Huu);
dhx  = dDhv(ind.hx,:);  dhu  = dhv(ind.hu,:);
dHxx = dhvv(ind.Hxx,:); dHxu = dhvv(ind.Hxu,:);
dHux = dhvv(ind.Hux,:); dHuu = dhvv(ind.Huu,:);
```

3 Deriving numerical derivatives

In order to derive the Jacobian of a function or matrix $F(\theta)$ at a point θ_0 with respect to θ, we use a two-sided finite difference method (also known as central differences). That is:
For each $j = 1, \ldots, n_\theta$

1. Select a step size h_j.
2. Solve the DSGE model twice using $\overline{\theta} = \theta_0 + e_j h_j$ and $\underline{\theta} = \theta_0 - e_j h_j$ with e_j a unit vector with the jth element equal to 1.
3. Compute
$$\mathtt{d}F^j := \frac{\partial vec(F(\theta_0))}{\partial \theta_j} \approx vec\left(\frac{F(\theta_0 + e_j h_j) - F(\theta_0 - e_j h_j)}{2h_j}\right)$$
4. Store $\mathtt{d}F^j$ as the j-th column of $\mathtt{d}F$.

4 Robustness check via nonidentification curve

As a robustness check for the Taylor rule coefficients, we compared the spectral density evaluated at θ_0 with the spectral densities evaluated at a hundred points from the nonidentification curve (fixing all parameters except the Taylor rule coefficients). Nonidentification curves are defined in Qu and Tkachenko (2012). If parameters are not identified, points on this curve yield the same spectral density at all frequencies apart from an approximation error; whereas if parameters are identified, the spectral densities differ. We found maximum relative and absolute deviations in the order 10^{-4} for the first 100 points away from θ_0, which is larger than the implied approximation error of 10^{-5} (step size used in the Euler method), and keep growing.

Results: Table 2: Deviations of spectra across frequencies (direction 1)

Maximum absolute deviations

Spectral density matrix element number

(1,1)	(2,1)	(3,1)	(2,2)	(3,2)	(3,3)
1.0e-04 *					
0.0004	0.0007	0.0001	0.0017	0.0010	0.0002
0.0042	0.0067	0.0007	0.0173	0.0098	0.0016
0.0084	0.0133	0.0014	0.0346	0.0196	0.0032
0.0126	0.0200	0.0020	0.0520	0.0294	0.0049
0.0169	0.0267	0.0027	0.0693	0.0392	0.0065
0.0211	0.0333	0.0034	0.0866	0.0490	0.0081
0.0253	0.0400	0.0041	0.1039	0.0589	0.0097
0.0295	0.0467	0.0048	0.1212	0.0687	0.0113
0.0337	0.0533	0.0054	0.1385	0.0785	0.0130
0.0380	0.0600	0.0061	0.1558	0.0883	0.0146

Maximum absolute deviations in relative form

Spectral density matrix element number

(1,1)	(2,1)	(3,1)	(2,2)	(3,2)	(3,3)
1.0e-04 *					
0.0026	0.0026	0.0009	0.0001	0.0001	0.0000
0.0257	0.0259	0.0089	0.0012	0.0012	0.0000
0.0514	0.0518	0.0177	0.0023	0.0025	0.0001
0.0771	0.0776	0.0266	0.0035	0.0037	0.0001
0.1029	0.1035	0.0354	0.0046	0.0049	0.0002
0.1286	0.1294	0.0443	0.0058	0.0061	0.0002
0.1543	0.1553	0.0531	0.0069	0.0074	0.0003
0.1800	0.1812	0.0619	0.0081	0.0086	0.0003
0.2057	0.2070	0.0708	0.0093	0.0098	0.0004
0.2314	0.2329	0.0796	0.0104	0.0110	0.0004

Maximum relative deviations

Spectral density matrix element number

(1,1)	(2,1)	(3,1)	(2,2)	(3,2)	(3,3)
1.0e-04 *					
0.0026	0.0031	0.0032	0.0010	0.0004	0.0005
0.0257	0.0312	0.0321	0.0098	0.0036	0.0046
0.0514	0.0624	0.0641	0.0195	0.0073	0.0092
0.0771	0.0935	0.0961	0.0293	0.0109	0.0138
0.1029	0.1247	0.1282	0.0391	0.0145	0.0184
0.1286	0.1559	0.1602	0.0488	0.0182	0.0230
0.1543	0.1871	0.1922	0.0586	0.0218	0.0275
0.1800	0.2183	0.2242	0.0684	0.0255	0.0321
0.2057	0.2495	0.2562	0.0782	0.0291	0.0367
0.2314	0.2806	0.2882	0.0879	0.0328	0.0412

We also used the points reported in Table 1 of Qu and Tkachenko (2012) and found maximum relative and absolute deviations in the order of 10^{+4}:

Results: Deviations of spectra across frequencies (direction 1)

Maximum absolute deviations

Spectral density matrix element number

(1,1)	(2,1)	(3,1)	(2,2)	(3,2)	(3,3)
1.0e+04 *					
0	0	0	0	0	0
9.3597	2.9310	1.3877	1.6769	0.9264	2.9818
9.3585	2.9660	1.3559	1.7159	0.9093	2.8800
9.3572	3.0012	1.3239	1.7566	0.8927	2.7803
9.3557	3.0366	1.2918	1.7991	0.8765	2.6825
9.3540	3.0720	1.2595	1.8434	0.8605	2.5868
9.3523	3.1076	1.2270	1.8896	0.8447	2.4932
9.3503	3.1433	1.1944	1.9378	0.8288	2.4017

```
    9.3483    3.1791    1.1616    1.9879    0.8129    2.3123
    9.3461    3.2151    1.1286    2.0402    0.7968    2.2250


          Maximum absolute deviations in relative form
           Spectral density matrix element number
  (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)
 1.0e+05 *

       0         0         0         0         0         0
    5.7076    1.7395    1.5635    0.1562    0.0860    0.0082
    5.7069    1.7579    1.5245    0.1589    0.0854    0.0079
    5.7060    1.7761    1.4855    0.1615    0.0848    0.0076
    5.7051    1.7941    1.4465    0.1641    0.0836    0.0074
    5.7041    1.8120    1.4089    0.1664    0.0824    0.0071
    5.7030    1.8297    1.3697    0.1685    0.0811    0.0069
    5.7019    1.8474    1.3306    0.1705    0.0796    0.0066
    5.7006    1.8647    1.2913    0.1722    0.0779    0.0064
    5.6993    1.8821    1.2534    0.1739    0.0762    0.0061


          Maximum relative deviations
           Spectral density matrix element number
  (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)
 1.0e+05 *

       0         0         0         0         0         0
    5.7076    1.9262    5.7157    0.4220    0.7507    0.5269
    5.7069    1.9486    5.5770    0.4310    0.7392    0.5144
    5.7060    1.9710    5.4379    0.4401    0.7271    0.5021
    5.7051    1.9934    5.2984    0.4493    0.7142    0.4901
    5.7041    2.0159    5.1584    0.4586    0.7007    0.4783
    5.7030    2.0383    5.0179    0.4681    0.6864    0.4668
    5.7019    2.0608    4.8770    0.4777    0.6715    0.4555
    5.7006    2.0833    4.7356    0.4874    0.6558    0.4444
    5.6993    2.1058    4.5938    0.4972    0.6395    0.4336
```

The Matlab code we used to compute these results is based on the code of Qu and Tkachenko (they also provide points on the nonidentification curve in a matrix).

```
load('OptionsRobustness'); % This loads the options set in identification_run.m for the An & Schorfheide model, in particular
addpath('./utils','./models','./models/AnSchorfheide','-begin');
theta0 = DSGE_Model.param.estim;      % Set local point as specified in the GUI
[Solut0,Deriv0] = EvaluateSparse(theta0,DSGE_Model,Settings.approx,'Analytical'); % Solve Model at theta0
Settings.speed = 'No Speed'; % Do analytical derivatives for innovations once to initialize script files
[G0,Omega0] = gmatrix(Solut0,Deriv0,DSGE_Model,Ident_Test,Settings); % Compute G matrix and spectrum across frequencies for t
Omega{1} = Omega0; % store into structre
Settings.speed = 'Speed'; % Do not compute analytical derivatives for innovations anymore, i.e. evaluate script files
index_par = ~DSGE_Model.param.fix; % Index for parameters of Taylor rule

%% 10 points on nonidentification curve reported in Qu and Tkachenko's paper table 1
thet_dir1 = repmat(theta0',10,1); % initialize points on nonidentification curve at theta0 (in particular all other parameter
load Ni_curves_data %load curve points given in Qu and Tkachenko's paper
thet_dir1(:,index_par)=theta_dir1([1445:1445:14450]'+1,[6 7 8 11]); %select ten equally spaced points along Direction 1
for j = 2:10
    thetaj = thet_dir1(j,:)';      % Set local point
    [Solutj,Derivj] = EvaluateSparse(thetaj,DSGE_Model,Settings.approx,'Analytical'); % Solve model at point from nonidentifi
    [Gj,Omegaj]=gmatrix(Solutj,Derivj,DSGE_Model,Ident_Test,Settings); % Compute G and spectrum at point from nonidentificati
    Omega{j} = Omegaj; % Store spectrum
end
maxdev_a1=zeros(10,9); %blanks
maxdev_r1=maxdev_a1;
maxdev_rr1=maxdev_a1;
maxdev_r2=maxdev_a1;
maxdev_a2=maxdev_a1;
maxdev_rr2=maxdev_a1;
windex=maxdev_a1;
for i=1:10
    ad=abs(Omega0-Omega{i});
```

```
    rd=ad./abs(Omega0);
    maxdev_a1(i,:)=max(ad); %maximum absolute deviations
    maxdev_r1(i,:)=max(rd); %maximum relative deviations
    for j=1:9
     windex(i,j)=max(find(ad(:,j)==max(ad(:,j))));
     maxdev_rr1(i,j) = rd(windex(i,j),j); % maximum absolute deviations in relative form
    end %measure 2 converts maximum abs deviations in maxdev_a1 and converts them into relative
end

format('short')
disp('Results: Deviations of spectra across frequencies (direction 1)');
disp('                  Maximum absolute deviations')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_a1(:,[1,2,3,5,6,9]))
disp('                                       ')
disp('             Maximum absolute deviations in relative form')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_rr1(:,[1,2,3,5,6,9]))
disp('                                       ')
disp('             Maximum relative deviations ')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_r1(:,[1,2,3,5,6,9]))


%% General procedure to get points on nonidentification curve given Euler
[V,D] = eig(G0); % V matrix of eigenvectors, D diagonal matrix with eigenvalues
[~,idxEV] = min(abs(real(diag(D)))); % find index for smallest eigenvalue
c_thet = real(V(:,idxEV)); % eigenvector for smallest eigenvalue
if c_thet(1) < 0; c_thet = -c_thet; end % restrict first element to be positive
h = 1e-5; % step size for Euler method
npoints = 100; % number of points considered
thetajold = theta0;
for j = 1:npoints
    thetaj = theta0; % initialize points on nonidentification curve at theta0 (in particular all other parameters do not char
    thetaj(find(index_par)) = thetajold(find(index_par)) + c_thet.*h;
    [Solutj,Derivj] = EvaluateSparse(thetaj,DSGE_Model,Settings.approx,'Analytical'); % Solve model at point from nonidentifi
    [Gj,Omegaj]=gmatrix(Solutj,Derivj,DSGE_Model,Ident_Test,Settings); % Compute G and spectrum at point from nonidentificati
    [V,D] = eig(Gj); % V matrix of eigenvectors, D diagonal matrix with eigenvalues
    [~,idxEV] = min(abs(real(diag(D)))); % find index for smallest eigenvalue
    c_thet = real(V(:,idxEV)); % eigenvector for smallest eigenvalue
    if c_thet(1) < 0; c_thet = -c_thet; end % restrict first element to be positive
    S = sprintf('robcheck/iter_%d',j);
    save(S,'thetaj','Omegaj');
    thetajold = thetaj;
end

clear Omega;
irun = 1;
for j = [1 10 20 30 40 50 60 70 80 90 100] % Select ten equally spaced points
    S = sprintf('robcheck/iter_%d',j);
    load(S)
    Omega{irun} = Omegaj;
    irun = irun+1;
end
maxdev_a1=zeros(10,9); %blanks
maxdev_r1=maxdev_a1;
maxdev_rr1=maxdev_a1;
maxdev_r2=maxdev_a1;
maxdev_a2=maxdev_a1;
maxdev_rr2=maxdev_a1;
windex=maxdev_a1;

for i=1:10
    ad=abs(Omega0-Omega{i});
    rd=ad./abs(Omega0);
```

```
    maxdev_a1(i,:)=max(ad); %maximum absolute deviations
    maxdev_r1(i,:)=max(rd); %maximum relative deviations
    for j=1:9
     windex(i,j)=max(find(ad(:,j)==max(ad(:,j))));
     maxdev_rr1(i,j) = rd(windex(i,j),j); % maximum absolute deviations in relative form
    end %measure 2 converts maximum abs deviations in maxdev_a1 and converts them into relative
end

format('short')
disp('Results: Table 2: Deviations of spectra across frequencies (direction 1)');
disp('                  Maximum absolute deviations')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_a1(:,[1,2,3,5,6,9]))
disp('                                      ')
disp('             Maximum absolute deviations in relative form')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_rr1(:,[1,2,3,5,6,9]))
disp('                                      ')
disp('             Maximum relative deviations ')
disp('              Spectral density matrix element number')
disp('    (1,1)      (2,1)     (3,1)     (2,2)     (3,2)     (3,3)')
disp(maxdev_r1(:,[1,2,3,5,6,9]))

function [G,Omega] = gmatrix(Solut,Deriv,DSGE_Model,Ident_Test,Settings)

%% Some auxiliary precomputations
gra = Solut.gra; Dgra_Dparam= Deriv.Dgra_Dparam;
hes = Solut.hes; Dhes_Dparam = Deriv.Dhes_Dparam;
Sigma = Solut.Sigma; DSigma_Dparam = Deriv.DSigma_Dparam;
etatilde = Solut.etatilde; Detatilde_Dparam = Deriv.Detatilde_Dparam;
sig = Solut.sig; Dsig_Dparam = Deriv.Dsig_Dparam;
SelectMat = Solut.SelectMat;
dfstudt = Solut.dfstudt; Ddfstudt_Dparam = Deriv.Ddfstudt_Dparam;
gv = Solut.gv; gvv = Solut.gvv; gSS = Solut.gSS;
hv = Solut.hv; hvv = Solut.hvv; hSS = Solut.hSS;
prun_A = Solut.prun_A; prun_B = Solut.prun_B; prun_C = Solut.prun_C; prun_D = Solut.prun_D;
prun_c = Solut.prun_c; prun_d = Solut.prun_d;
SolM = Solut.solM; SolN = Solut.solN; SolQ = Solut.solQ; SolR = Solut.solR;
SolS = Solut.solS; SolU = Solut.solU;

nv=DSGE_Model.numbers.nv; nx=DSGE_Model.numbers.nx; ny=DSGE_Model.numbers.ny; nu=DSGE_Model.numbers.nu;
nd=DSGE_Model.numbers.nd; nparam = size(Deriv.Dsig_Dparam,2);
n=nv+ny;% Number of variables

%% Separate gradient
f1 = gra(:,1:nv);
Df1_Dparam= Dgra_Dparam(1:n*nv,:);
f2 = gra(:,nv+1:n);
Df2_Dparam= Dgra_Dparam((n*nv+1):(n^2),:);
f3 = gra(:,(n+1):(n+nv));
Df3_Dparam= Dgra_Dparam((n^2+1):(n^2+n*nv),:);
f4 = gra(:,(n+nv+1):end);
Df4_Dparam= Dgra_Dparam((n^2+n*nv+1):end,:);

%% Compute etaT eta_etaT and derivatives
etatildeT = transpose(etatilde);
DetatildeT_Dparam = sparse(commutation(size(etatilde))*Detatilde_Dparam);
etatilde_etatildeT = etatilde*transpose(etatilde);
Detatilde_etatildeT_Dparam = DerivABCD(etatilde,Detatilde_Dparam,transpose(etatilde),DetatildeT_Dparam);
Solut.etatildeT = etatildeT;
Solut.etatilde_etatildeT = etatilde_etatildeT;
Deriv.DetatildeT_Dparam = DetatildeT_Dparam;
Deriv.Detatilde_etatildeT_Dparam = Detatilde_etatildeT_Dparam;

%% Construct Dhv_Dparam, DhvT_Dparam, Dgv_Dparam, DgvT_Dparam
F1=kron(transpose(hv),f2)+kron(speye(nv),f4);
```

```
F2=kron(speye(nv),f2*gv)+kron(speye(nv),f1);
F=-kron(transpose(hv)*transpose(gv),speye(n))*Df2_Dparam-kron(transpose(hv),speye(n))*Df1_Dparam -
kron(transpose(gv),speye(n))*Df4_Dparam-Df3_Dparam;
Dgvhv_Dparam=[F1 F2]\F;
Dgv_Dparam = Dgvhv_Dparam(1:ny*nv,:);
Dhv_Dparam = Dgvhv_Dparam(ny*nv+1:end, :);
%Get transposes
DhvT_Dparam=sparse(commutation(size(hv)))*Dhv_Dparam;
DgvT_Dparam=sparse(commutation(size(gv)))*Dgv_Dparam;
Deriv.Dhv_Dparam = Dhv_Dparam; Deriv.DhvT_Dparam = DhvT_Dparam;
Deriv.Dgv_Dparam = Dgv_Dparam; Deriv.DgvT_Dparam = DgvT_Dparam;

%% Construct Dgvv_Dparam, Dhvv_Dparam, DgvvT_Dparam, DhvvT_Dparam
    % Derivative of Q1=kron(hv',f2,hv')+kron(eye(nv),f4,eye(nv))
    DSolQ1_Dparam = DerivXkronY(transpose(hv),DhvT_Dparam,kron(f2,transpose(hv)),DerivXkronY(f2,Df2_Dparam,transpose(hv),DhvT
        + DerivXkronY(speye(nv),sparse(zeros(nv^2,nparam)),kron(f4,speye(nv)),DerivXkronY(f4,Df4_Dparam,speye(nv),sparse(zero
    % Derivative of Q2=kron(eye(nv),f1+f2*gv,eye(nv))
    Df2gv_Dparam = DerivABCD(f2,Df2_Dparam,gv,Dgv_Dparam);
    DSolQ2_Dparam = DerivXkronY(speye(nv),sparse(zeros(nv^2,nparam)),kron(f1+f2*gv,speye(nv)),DerivXkronY(f1+f2*gv,Df1_Dparam
    % Derivative of Q = [Q1 Q2]
    SolQinv = SolQ\speye(size(SolQ,1));
    DinvSolQ_Dparam=[]; % Using algorithm 1 of the paper
    for i=1:nparam
        dSolQ = [reshape(DSolQ1_Dparam(:,i),n*nv^2,nv^2*ny) reshape(DSolQ2_Dparam(:,i),n*nv^2,nv^3)];
        dinvSolQ = -SolQinv*dSolQ*SolQinv;
        DinvSolQ_Dparam = [DinvSolQ_Dparam dinvSolQ(:)];
    end

    % Derivative of R=kron(eye(n),M')*hes*M with M=(hv, gv*hv,eye(nv),gv)'
    DSolM_Dparam =[];  % Using algorithm 1 of the paper
    for i=1:nparam
        dSolM1 = reshape(Dhv_Dparam(:,i),size(hv));
        dSolM4 = reshape(Dgv_Dparam(:,i),size(gv));
        dSolM2 = gv*dSolM1 + dSolM4*hv;
        dSolM3 = zeros(nv);
        dSolM = [dSolM1; dSolM2; dSolM3; dSolM4];
        DSolM_Dparam = [DSolM_Dparam dSolM(:)];
    end
    DSolMT_Dparam = sparse(commutation(nv+ny+nv+ny,nv))*DSolM_Dparam;
    InKronSolMT = kron(speye(n),transpose(SolM));
    DInKronSolMT_Dparam = DerivXkronY(speye(n),sparse(zeros(n^2,nparam)),transpose(SolM),DSolMT_Dparam);
    DSolR_Dparam=DerivABCD(InKronSolMT,DInKronSolMT_Dparam,hes,Dhes_Dparam,SolM,DSolM_Dparam);

    % Derivative of [vec(gvv);vec(hvv)]=-Q^(-1)*vec(R)
    Dgvvhvv_Dparam = (-1)*DerivABCD(SolQinv,DinvSolQ_Dparam,vec(SolR),DSolR_Dparam);
    Dgvv_Dparam = Dgvvhvv_Dparam(1:numel(gvv),:);
    DgvvT_Dparam = sparse(commutation(size(gvv)))*Dgvv_Dparam;
    Dhvv_Dparam = Dgvvhvv_Dparam(numel(gvv)+1:end,:);
    DhvvT_Dparam = sparse(commutation(size(hvv)))*Dhvv_Dparam;

    %% Construct DgSS_Dparam, DhSS_Dparam
    % Derivative of inv(S)=inv([S1 S2])=inv([f1+f2*gv f2+f4])
    SolSinv = SolS\speye(size(SolS,1));
    DSolS1_Dparam = Df1_Dparam + Df2gv_Dparam;
    DSolS2_Dparam = Df2_Dparam + Df4_Dparam;
    DinvSolS_Dparam=[];  % Using algorithm 1 of the paper
    for i=1:nparam
        dSolS = [reshape(DSolS1_Dparam(:,i),n,nv) reshape(DSolS2_Dparam(:,i),n,ny)];
        dinvSolS = -SolSinv*dSolS*SolSinv;
        DinvSolS_Dparam = [DinvSolS_Dparam dinvSolS(:)];
    end

    % Derivative of U = f2*trm(U1) + trm(U2)
    % Derivative of U1 = kron(eye(ny),etatilde*etatilde')*gxx
    SolU1=kron(speye(ny),etatilde_etatildeT)*gvv;
    DSolU1_Dparam = DerivABCD(kron(speye(ny),etatilde_etatildeT),DerivXkronY(speye(ny),sparse(zeros(ny^2,nparam)),etatilde_et
    % Derivative of U2 = kron(eye(n),N')*H*N*etatilde_etatildeT with N=(eye(nx),gx,zeros(n,nx))'
    DSolN_Dparam = [];
```

```
    for i=1:nparam  % Using algorithm 1 of the paper
        dSolN = [sparse(zeros(nv));reshape(Dgv_Dparam(:,i),size(gv));sparse(zeros(n,nv))];
        DSolN_Dparam=[DSolN_Dparam dSolN(:)];
    end
    DSolNT_Dparam = sparse(commutation(2*n,nv))*DSolN_Dparam;
    SolU2=kron(speye(n),transpose(SolN))*hes*SolN*etatilde_etatildeT;
    DSolU2_Dparam = DerivABCD(kron(speye(n),transpose(SolN)),DerivXkronY(speye(n),sparse(zeros(n^2,nparam)),transpose(SolN),D

    % Derivative of U = f2*trm(U1) + trm(U2)= trm(kron(eye(ny),etatilde*etatilde')*gxx) + trm(kron(eye(n),N')*H*N*eta_etaT wi
    DSolU_Dparam=[];  % Using algorithm 1 of the paper
    for i=1:nparam
        df2 = reshape(Df2_Dparam(:,i),size(f2));
        dtrmSolU1= sparse(tracem(reshape(DSolU1_Dparam(:,i),size(SolU1))));
        dtrmSolU2= sparse(tracem(reshape(DSolU2_Dparam(:,i),size(SolU2))));
        DSolU_Dparam = [DSolU_Dparam (df2*sparse(tracem(SolU1))+f2*dtrmSolU1+dtrmSolU2)];
    end

    % Derivative of [hSS;gSS]==-inv(S)*U
    DhSS_gSS = -DerivABCD(SolSinv,DinvSolS_Dparam,SolU,DSolU_Dparam);
    DhSS_Dparam = DhSS_gSS(1:numel(hSS),:);
    DhSST_Dparam = sparse(commutation(size(hSS)))*DhSS_Dparam;
    DgSS_Dparam = DhSS_gSS(numel(hSS)+1:end,:);
    DgSST_Dparam = sparse(commutation(size(gSS)))*DgSS_Dparam;


%% Construct Dprun_A_Dparam, Dprun_B_Dparam, Dprun_C_Dparam, Dprun_D_Dparam, Dprun_c_Dparam, Dprun_d_Dparam
hx = hv(Solut.ind.hx); hu = hv(Solut.ind.hu); hss=hSS(1:nx);
Hxx= hvv(Solut.ind.Hxx); Hxu=hvv(Solut.ind.Hxu); Hux=hvv(Solut.ind.Hux); Huu=hvv(Solut.ind.Huu);
Dhx_Dparam = Dhv_Dparam(Solut.ind.hx,:); Dhu_Dparam = Dhv_Dparam(Solut.ind.hu,:);
DHxx_Dparam=Dhvv_Dparam(Solut.ind.Hxx,:);
DHxu_Dparam=Dhvv_Dparam(Solut.ind.Hxu,:);
DHux_Dparam=Dhvv_Dparam(Solut.ind.Hux,:);
DHuu_Dparam=Dhvv_Dparam(Solut.ind.Huu,:);
Dhss_Dparam=DhSS_Dparam(1:nx,:);
Dhu_kron_hu = DerivXkronY(hu,Dhu_Dparam,hu,Dhu_Dparam);
Dhu_kron_hx = DerivXkronY(hu,Dhu_Dparam,hx,Dhx_Dparam);
Dhx_kron_hu = DerivXkronY(hx,Dhx_Dparam,hu,Dhu_Dparam);
gx = gv(Solut.ind.gx); gu = gv(Solut.ind.gu);
Gxx=gvv(Solut.ind.Gxx); Gxu=gvv(Solut.ind.Gxu); Gux=gvv(Solut.ind.Gux); Guu=gvv(Solut.ind.Guu);
Dgx_Dparam = Dgv_Dparam(Solut.ind.gx,:); Dgu_Dparam = Dgv_Dparam(Solut.ind.gu,:);
DGxx_Dparam=Dgvv_Dparam(Solut.ind.Gxx,:);
DGxu_Dparam=Dgvv_Dparam(Solut.ind.Gxu,:);
DGux_Dparam=Dgvv_Dparam(Solut.ind.Gux,:);
DGuu_Dparam=Dgvv_Dparam(Solut.ind.Guu,:);

Dprun_A_Dparam = []; Dprun_B_Dparam = []; Dprun_C_Dparam = []; Dprun_D_Dparam = []; Dprun_c_Dparam = []; Dprun_d_Dparam = [];
for i=1:nparam %Using algorithm 1 of the paper
    dprun_A = [reshape(Dhx_Dparam(:,i),size(hx)), sparse(zeros(nx,nx)),sparse(zeros(nx,nx^2));
               sparse(zeros(nx,nx)), reshape(Dhx_Dparam(:,i),size(hx)), 0.5*reshape(DHxx_Dparam(:,i),size(Hxx));
               sparse(zeros(nx*nx,nx)),sparse(zeros(nx*nx,nx)),reshape(DerivXkronY(hx,Dhx_Dparam(:,i),hx,Dhx_Dparam(:,i)),[nx
    dprun_B = [reshape(Dhu_Dparam(:,i),size(hu)) sparse(zeros(nx,nu^2+nu*nx+nu*nx));...
              sparse(zeros(nx,nu)) 0.5*reshape(DHuu_Dparam(:,i),size(Huu)) 0.5*reshape(DHux_Dparam(:,i),size(Hux))  0.5*resha
              zeros(nx^2,nu) reshape(Dhu_kron_hu(:,i),[nx^2,nu^2]) reshape(Dhu_kron_hx(:,i),[nx^2,nu*nx]) reshape(Dhx_kron_hu
    dprun_C = [reshape(Dgx_Dparam(:,i),size(gx)), reshape(Dgx_Dparam(:,i),size(gx)),0.5*reshape(DGxx_Dparam(:,i),size(Gxx))];
    dprun_D = [reshape(Dgu_Dparam(:,i),size(gu)), 0.5*reshape(DGuu_Dparam(:,i),size(Guu)), 0.5*reshape(DGux_Dparam(:,i),size(
    dprun_c = [zeros(nx,1);
               reshape(0.5*(2*sig*hss*Dsig_Dparam(:,i) + sig^2*Dhss_Dparam(:,i)),size(hss)) + 0.5*(reshape(DHuu_Dparam(:,i),s
               reshape(Dhu_kron_hu(:,i),[nx^2,nu^2])*vec(Sigma)+kron(hu,hu)*DSigma_Dparam(:,i)];
    dprun_d = reshape(0.5*(2*sig*gSS*Dsig_Dparam(:,i) + sig^2*DgSS_Dparam(:,i)),size(gSS))+0.5*(reshape(DGuu_Dparam(:,i),size
    Dprun_A_Dparam = [Dprun_A_Dparam dprun_A(:)];
    Dprun_B_Dparam = [Dprun_B_Dparam dprun_B(:)];
    Dprun_C_Dparam = [Dprun_C_Dparam dprun_C(:)];
    Dprun_D_Dparam = [Dprun_D_Dparam dprun_D(:)];
    Dprun_c_Dparam = [Dprun_c_Dparam dprun_c(:)];
    Dprun_d_Dparam = [Dprun_d_Dparam dprun_d(:)];
end
Deriv.Dprun_A_Dparam = Dprun_A_Dparam; Deriv.Dprun_B_Dparam = Dprun_B_Dparam; Deriv.Dprun_C_Dparam = Dprun_C_Dparam; Deriv.Dp
Deriv.Dprun_c_Dparam = Dprun_c_Dparam; Deriv.Dprun_d_Dparam = Dprun_d_Dparam;
```

```
%% Analytical derivative of Expectation of observables (Ed)
[Ed,DEd_Dparam,E_xf_xf,DE_xf_xf_Dparam] = DerivExpectation(Solut,Deriv,DSGE_Model.numbers,'Analytical',Settings.approx);
Solut.Ed = Ed; Deriv.DEd_Dparam = DEd_Dparam;
Solut.E_xf_xf = E_xf_xf; Deriv.DE_xf_xf_Dparam = DE_xf_xf_Dparam;

%% Analytical Derivative of cumulants of Innovations xi_t=[u;kron(u,u)-vec(SIGU);kron(u,xf);kron(xf,u)]
[M2min,M3min,M4min,DM2min_Dparam,DM3min_Dparam,DM4min_Dparam] = ProdMom_inov(nu,nx,Sigma,DSigma_Dparam,E_xf_xf,DE_xf_xf_Dpara

%% Save or load duplication matrix from file depending on speed setting
filename = ['./models/', DSGE_Model.shortname,'/',DSGE_Model.shortname,'_spec',num2str(DSGE_Model.spec),'_approx',];
    GAMMA2min = M2min; DGAMMA2min_Dparam = DM2min_Dparam;
    if strcmp(Settings.speed,'No Speed')
        fprintf('Compute & save duplication matrix\n');
        DPxi = sparse(duplication(nu+nu*(nu+1)/2+nu*nx));
        try
            save([filename,num2str(Settings.approx),'_prodmom_auxiliary'],'DPxi','-append');
        catch
            save([filename,num2str(Settings.approx),'_prodmom_auxiliary'],'DPxi');
        end
    else
        fprintf('Load duplication matrix for second-order cumulant\n');
        load([filename,num2str(Settings.approx),'_prodmom_auxiliary'],'DPxi');
    end



%% Qu and Tkachenko's criteria
    % Construct G analytically
    fprintf('Compute Polyspectra for frequencies\n')
    % Create vector of Fourier frequencies for approximation of the integral
    N=str2double(Ident_Test.options{2}); % Subintervalls
    % Create Fourier frequencies
    w=2*pi*(-(N/2):1:(N/2))'/N;
    % Calculate zero-lag cumulants and its derivative
        % Auxiliary matrix that selects unique elements in xi_t
        Fxi = [speye(nu) spalloc(nu,nu*(nu+1)/2 + nu*nx,0);...
                spalloc(nu^2,nu,0) sparse(duplication(nu)) spalloc(nu^2,nu*nx,0);...
                spalloc(nu*nx,nu+nu*(nu+1)/2,0) speye(nu*nx);
                spalloc(nx*nu,nu+nu*(nu+1)/2,0) sparse(commutation(nx,nu))];

        GAMMA2 = reshape(DPxi*GAMMA2min,size(Fxi,2),size(Fxi,2)); %Note GAMMA2 is a matrix, not a vector

        for i=1:nparam
            dGAMMA2 = reshape(DPxi*DGAMMA2min_Dparam(:,i),size(Fxi,2),size(Fxi,2));
            DGAMMA2_Dparam(:,i) = dGAMMA2(:);
            clear dGAMMA2
        end
    GAMMA2full = Fxi*GAMMA2*transpose(Fxi);
    for j=1:nparam
        DGAMMA2full_Dparam(:,j) = vec(Fxi*reshape(DGAMMA2_Dparam(:,j),size(Fxi,2),size(Fxi,2))*transpose(Fxi));
    end
    G = zeros(nparam,nparam,length(w)); % Initialize objective function for power spectrum
    Omega = zeros(length(w),size(DSGE_Model.symbolic.SelectMat,1)^2); % Initialize objective function for power spectrum
    tic
    parfor l1=1:length(w); %loop computes analytical derivative of spectra
        z1=exp(-1i*w(l1)); % Use Fourier transform for lag operator
        [Hz1,DHz1_Dparam,DHz1cT_Dparam] = TransferFunction(z1,SelectMat,prun_A,prun_B,prun_C,prun_D,Dprun_A_Dparam,Dprun_B_Dp
        % Compute power spectrum
        Omega(l1,:) = (1/(2*pi))*transpose(vec(Hz1*GAMMA2full*Hz1'));
        % Compute derivative of power spectrum
        DOmega2_Dparam = (1/(2*pi))*DerivABCD(Hz1,DHz1_Dparam,GAMMA2full,DGAMMA2full_Dparam,Hz1',DHz1cT_Dparam);
        G(:,:,l1) = DOmega2_Dparam'*DOmega2_Dparam;
    end;
    toc
    G = 2*pi*sum(G,3)./length(w); % Normalize G2 Matrix
end

function [H,DH_Dparam,DHcT_Dparam] = TransferFunction(z,SelectMat,prun_A,prun_B,prun_C,prun_D,Dprun_A_Dparam,Dprun_B_Dparam,D
```

```
    % Compute H and DH_Dparam and its conjugate(!) transpose.
    zIminusA =  (z*speye(size(prun_A,1)) - prun_A);
    zIminusAinv = zIminusA\speye(size(prun_A,1));
    DzIminusA_Dparam = -Dprun_A_Dparam;
    DzIminusAinv_Dparam = kron(-(transpose(zIminusA)\speye(size(prun_A,1))),zIminusAinv)*DzIminusA_Dparam;
    H = SelectMat*(prun_D + prun_C*zIminusAinv*prun_B); % Transfer function
    DH_Dparam = kron(speye(size(prun_D,2)),SelectMat)*(Dprun_D_Dparam + DerivABCD(prun_C,Dprun_C_Dparam,zIminusAinv,DzIminusA
    DHcT_Dparam = commutation(size(H))*conj(DH_Dparam); % conjugate transpose!
end%transferfunction end
```