Simulating the Hybrid Version of the New Keynesian Model for a Small Open Economy with MATLAB®
(response to a temporary cost-push shock)

Contents

Introduction

The purpose of the following problem set is said to be a *step-by-step instruction* to write a MATLAB-file (`*.m`-file) containing the required commands to simulate the hybrid version of the NKM for a small open economy (SME) in response to a temporary cost-push shock ($\rho_a = 0.2$ and $\rho_a = 0.8$) numerically. It has a specific structure:

- Every section begins with more or less detailed instructions concerning the current task
- followed by a grey-shaded box/area which represents lines of missing code.
- Below you can find lines of the corresponding MATLAB-output[1] which would occur if you would have run the code above. Therefore, the output serves as a direct hint for the solution and should be used as guidance.

The general exercise is to replicate the given MATLAB-output using the instructions, MATLAB's `help`- and/or `doc`-command, the mathworks documentation center (http://www.mathworks.de/de/help/) and, of course, the *manuscript*.
So, please write an `*.m`-file containing the missing MATLAB-Code. You can either use the prepared `*.m`-file (from the folder "`X:\Kursmaterial`") or create a new one by opening a new MATLAB-script-file and save it perhaps as `NKMsoe_model.m`.
Note that, except for parameters and exogenously given variables, it is not sufficient just to adopt the given lines of output since, in general, MATLAB-output shows evaluated numerical values, i.e. the result of a certain calculation which depends on these previously defined parameters/variables.
The objective of developed MATLAB-Code is to be applicable to various parameter sets and, therefore, to various variations of the system matrices. Thus, the emphasis of today's lecture is also on automatization of calculation processes.

[1] MATLAB-output can be numerical values and also plots.

Step 1 - Parameterization

Once one has derived the hybrid IS equation, the hybrid PC, the UIP condition and the Taylor Rule (TR), one must introduce auxiliary state variables to obtain the state space representation of the dynamic system (3.2.16) (state equations in their explicit form)

$$A\begin{pmatrix} w_{t+1} \\ E_t v_{t+1} \end{pmatrix} = B\begin{pmatrix} w_t \\ v_t \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \varepsilon_{t+1}$$

In order to compute the solution time paths for $\varphi^\nu = 0$ and $\varphi^\nu = 0.8$ in a single run, we specify the entire code subject to the values of the autocorrelation parameter (`AR_par` represents φ^ν in the code) using a `for`-loop which repeats the entire computation for every given value of φ^ν.

```
AR_par_vector = [0 0.8];    % vector of values for the autocorrelation parameter
format = ['k-o '; 'k--d']; % see "help plot" for further information
figure(2); clf;            % see "help clf" for further information

for AR_par_pos = 1:length(AR_par_vector) % beginning of the initial for-loop

phi    = 0.5;
sig    = 2;
gam_f  = 0.5;
gam_b  = 0.5;
gam_x  = 0.2;
del_pi = 1.5;
del_x  = 0.5;
c      = 0.2;
lambda = 0.05;
AR_par = AR_par_vector(AR_par_pos);
```

Step 2 - Specify the System Matrices A and B

```
% System is: A * [w; v](+1) = B * [w; v] + [1;0;0;0;0;0;0]*eps
  fprintf('System matrices for AR_par = %g:\n', AR_par);
  A11 = eye(4);
  A12 = zeros(4,3);
  A21 = zeros(3,4);
  A22 = [1-phi 0 1/sig; 0 -1 1; 0 0 gam_f];
  A   = [ [A11 A12] ; [A21 A22] ]

  B11 = [AR_par 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0];
  B12 = [0 0 0; eye(3)];
  B21 = [0 -phi 0 0; 0 0 0 0; -1 0 -lambda -gam_b];
  B22 = [1+del_x/sig c del_pi/sig; del_x -1 del_pi; -gam_x lambda 1-lambda];
  B   = [ [B11 B12] ; [B21 B22] ]
```

(See: `help fprintf`)

```
System matrices for AR_par = 0:

  A =
    1.0000         0         0         0         0         0         0
         0    1.0000         0         0         0         0         0
         0         0    1.0000         0         0         0         0
         0         0         0    1.0000         0         0         0
         0         0         0         0    0.5000         0    0.5000
         0         0         0         0         0   -1.0000    1.0000
         0         0         0         0         0         0    0.5000

  B =
         0         0         0         0         0         0         0
         0         0         0         0    1.0000         0         0
         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0    1.0000
         0   -0.5000         0         0    1.2500    0.2000    0.7500
         0         0         0         0    0.5000   -1.0000    1.5000
   -1.0000         0   -0.0500   -0.5000   -0.2000    0.0500    0.9500

  System matrices for AR_par = 0.8:

  A =
    1.0000         0         0         0         0         0         0
         0    1.0000         0         0         0         0         0
         0         0    1.0000         0         0         0         0
         0         0         0    1.0000         0         0         0
         0         0         0         0    0.5000         0    0.5000
         0         0         0         0         0   -1.0000    1.0000
         0         0         0         0         0         0    0.5000

  B =
    0.8000         0         0         0         0         0         0
         0         0         0         0    1.0000         0         0
         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0    1.0000
         0   -0.5000         0         0    1.2500    0.2000    0.7500
         0         0         0         0    0.5000   -1.0000    1.5000
   -1.0000         0   -0.0500   -0.5000   -0.2000    0.0500    0.9500
```

Step 3 - Application of the QZ Factorization

Generally, one can premultiply the original system with the inverse of the system matrix on the LHS (here A) in order to obtain the following form

$$A\begin{pmatrix} w_{t+1} \\ E_t v_{t+1} \end{pmatrix} = B\begin{pmatrix} w_t \\ v_t \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \varepsilon_{t+1} \quad \Longrightarrow \quad \begin{pmatrix} w_{t+1} \\ E_t v_{t+1} \end{pmatrix} = C\begin{pmatrix} w_t \\ v_t \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \varepsilon_{t+1} \quad \text{with } C = A^{-1}B.$$

Then the Schur decomposition can be applied to solve the *standard* eigenvalue problem

$$Cv = \lambda v.$$

Unfortunately, there are cases in which the system matrix A is singular (i.e. not invertible). In such cases, the Schur (as well as the Jordan) decomposition cannot be applied. A solution to this problem is usually the application of

the QZ Factorization[2] which is often also called *generalized Schur decomposition* because it solves the *generalized* eigenvalue problem

$$Bv = \lambda Av$$

where we have to decompose both system matrices instead of only one on the RHS. As a result, the algorithm provides a solution in **any** case. Now A and B are decomposed into

$$A = \overline{Q}' S \overline{Z}'$$
$$B = \overline{Q}' T \overline{Z}'$$

which leads to the following representation of the dynamic equation system in matrix notation

$$\overline{Q}' S \overline{Z}' \begin{pmatrix} w_{t+1} \\ E_t v_{t+1} \end{pmatrix} = \overline{Q}' T \overline{Z}' \begin{pmatrix} w_t \\ v_t \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \varepsilon_{t+1}$$

After some rearrangement of terms, one is able to build the ratios of the diagonal elements of T to the corresponding diagonal elements of S which provides the generalized eigenvalues

$$\lambda_i = \frac{T_{ii}}{S_{ii}}$$

which solve the already mentioned generalized eigenvalue problem.

As a consequence, the difference between the Schur decomposition and its generalized version is the order of the applied mathematical operations. Now the decomposition of the system matrices is prior to the formation of the ratio instead of doing it the other way round.

Now please compute the QZ factorization (see: `help qz`) for the system matrices A and B, i.e. compute the upper triangular matrices S and T as well as the corresponding unitary matrices Q and Z (the specification of the complex flag within the `qz`-command is not necessary since its complex version is already the default value of the `qz`-function).

```
30 fprintf('Generalized Schur decomposition for AR_par = %g:\n', AR_par);
31 [S,T,Q,Z] = qz(A,B)
32 if sum(sum(abs(Q*A*Z-S))) < 0.0001 && sum(sum(abs(Q*B*Z-T))) < 0.0001
33    fprintf('==> Successful application of generalized Schur decomposition for AR_par = %g\r\n', AR_par);
34 else
35     error('Error in generalized Schur decomposition')
36 end
```

```
Generalized Schur decomposition for AR_par = 0:

S =
   0.8395 - 0.4119i   0.0425 - 0.0446i  -0.1422 + 0.0096i   0.0257 - 0.2321i  -0.2148 + 0.1203i   0.0698 - 0.0651i         0
         0            0.8192 + 0.4019i   0.0008 - 0.0910i  -0.3597 - 0.0065i   0.1664 - 0.1270i  -0.0947 + 0.0387i         0
         0                  0            1.2505            -0.3950            -0.0268            -0.4523                  0
         0                  0                  0            0.4268             0.3441            -0.0836                  0
         0                  0                  0                 0             0.4965            -0.0340                  0
         0                  0                  0                 0                  0            -1.1055                  0
         0                  0                  0                 0                  0                  0             1.0000

T =
   0.4852   -0.1594 - 0.0577i  -0.3920 + 0.4080i   0.1446 + 0.4225i   0.7607 + 0.0784i   0.4147 - 0.1675i  -0.0044 + 0.1032i
         0    0.4735            0.5978 - 0.2130i   0.6739 + 0.1357i   0.1981 + 0.5000i  -0.2201 + 0.2516i   0.1606 + 0.0074i
         0         0            1.5783            -0.4638             1.0092            -1.3085             0.3118
         0         0                 0             0.6958             0.0438            -0.1865             0.7965
         0         0                 0                  0             1.0308             0.1518            -0.4815
         0         0                 0                  0                  0                  0            -0.0000
         0         0                 0                  0                  0                  0                  0
```

[2] A factorization is the decomposition of an object (for example, a number, a polynomial, or here a matrix) into a product of other objects, or factors, which when multiplied together give the original.

```
Q =
         0   -0.7072 + 0.0965i  -0.3042 - 0.3750i   0.1904 - 0.3496i  -0.1493 + 0.0001i   0.2362 - 0.1012i   0.0044 - 0.1032i
         0    0.0802 - 0.4479i  -0.6156 - 0.2342i  -0.5268 + 0.0883i  -0.0147 - 0.0966i  -0.1345 + 0.1427i  -0.1606 - 0.0074i
         0   -0.3446             0.3279            -0.1464            -0.3702            -0.7197            -0.3118
         0    0.2424             0.0102             0.1804            -0.3587             0.3815            -0.7965
         0    0.2247            -0.0573            -0.0474            -0.8381             0.0985             0.4815
         0    0.2357            -0.4714             0.7071             0.0000            -0.4714             0.0000
    1.0000                  0                   0                   0                   0                   0                   0

Z =
         0                   0                   0                   0                   0                   0              1.0000
   -0.6334 + 0.2103i  -0.1488 + 0.4266i  -0.2886             0.1731             0.4379            -0.2132                   0
   -0.1009 + 0.4401i  -0.5946 - 0.0261i   0.4705             0.1769            -0.0861             0.4264                   0
    0.3038 + 0.2151i  -0.3723 - 0.2777i  -0.2219             0.4097            -0.1394            -0.6396                   0
   -0.3431 - 0.0468i   0.1451 + 0.2683i  -0.0839             0.2602            -0.8440            -0.0000                   0
   -0.1476 + 0.1819i  -0.2213 + 0.0686i   0.1656            -0.7940            -0.2276            -0.4264                   0
    0.0924 + 0.1696i  -0.2596 - 0.1085i  -0.7821            -0.2698            -0.1315             0.4264                   0

==> Successful application of generalized Schur decomposition for AR_par = 0


Generalized Schur decomposition for AR_par = 0.8:

S =
    0.8395 - 0.4119i   0.0425 - 0.0446i  -0.1422 + 0.0096i   0.0257 - 0.2321i  -0.2148 + 0.1203i   0.0698 - 0.0651i         0
         0             0.8192 + 0.4019i   0.0008 - 0.0910i  -0.3597 - 0.0065i   0.1664 - 0.1270i  -0.0947 + 0.0387i         0
         0                  0             1.2505            -0.3950            -0.0268            -0.4523                   0
         0                  0                  0             0.4268             0.3441            -0.0836                   0
         0                  0                  0                  0             0.4965            -0.0340                   0
         0                  0                  0                  0                  0            -1.1055                   0
         0                  0                  0                  0                  0                  0              1.0000

T =
    0.4852   -0.1594 - 0.0577i  -0.3920 + 0.4080i   0.1446 + 0.4225i   0.7607 + 0.0784i   0.4147 - 0.1675i  -0.0044 + 0.1032i
         0    0.4735             0.5978 - 0.2130i   0.6739 + 0.1357i   0.1981 + 0.5000i  -0.2201 + 0.2516i   0.1606 + 0.0074i
         0         0             1.5783            -0.4638             1.0092            -1.3085             0.3118
         0         0                  0             0.6958             0.0438            -0.1865             0.7965
         0         0                  0                  0             1.0308             0.1518            -0.4815
         0         0                  0                  0                  0                  0            -0.0000
         0         0                  0                  0                  0                  0             0.8000

Q =
         0   -0.7072 + 0.0965i  -0.3042 - 0.3750i   0.1904 - 0.3496i  -0.1493 + 0.0001i   0.2362 - 0.1012i   0.0044 - 0.1032i
         0    0.0802 - 0.4479i  -0.6156 - 0.2342i  -0.5268 + 0.0883i  -0.0147 - 0.0966i  -0.1345 + 0.1427i  -0.1606 - 0.0074i
         0   -0.3446             0.3279            -0.1464            -0.3702            -0.7197            -0.3118
         0    0.2424             0.0102             0.1804            -0.3587             0.3815            -0.7965
         0    0.2247            -0.0573            -0.0474            -0.8381             0.0985             0.4815
         0    0.2357            -0.4714             0.7071             0.0000            -0.4714             0.0000
    1.0000    0                   0                   0                   0                   0                   0



Z =
         0                   0                   0                   0                   0                   0              1.0000
   -0.6334 + 0.2103i  -0.1488 + 0.4266i  -0.2886             0.1731             0.4379            -0.2132                   0
   -0.1009 + 0.4401i  -0.5946 - 0.0261i   0.4705             0.1769            -0.0861             0.4264                   0
    0.3038 + 0.2151i  -0.3723 - 0.2777i  -0.2219             0.4097            -0.1394            -0.6396                   0
   -0.3431 - 0.0468i   0.1451 + 0.2683i  -0.0839             0.2602            -0.8440            -0.0000                   0
   -0.1476 + 0.1819i  -0.2213 + 0.0686i   0.1656            -0.7940            -0.2276            -0.4264                   0
    0.0924 + 0.1696i  -0.2596 - 0.1085i  -0.7821            -0.2698            -0.1315             0.4264                   0

==> Successful application of generalized Schur decomposition for AR_par = 0.8
```

Step 4 - Rearrangement of QZ Factorizations

The generalized eigenvalues

$$\lambda_i = \frac{T_{ii}}{S_{ii}}$$

of the system, i.e. the ratios of the diagonal elements of T and S, appear also in random order (i.e. similarly to the standard Schur decomposition). Therefore, one has to rearrange the rows of T, S, Q and Z in such a way that the first (upper) subsystem contains all *stable* eigenvalues $\lambda_i < 1$ and the second (lower) subsystem contains all *unstable* eigenvalues $\lambda_i > 1$ (see: `help ordqz`). In contrast to the Schur decomposition, we can't see their initial position by throwing a glance at the main diagonal of a single matrix. At first, we have to compute the corresponding ratios (see: `help ./`) in order to generate an appropriate logical vector for the `ordqz`-command which is then used for the rearrangement. So, isolate and store the diagonal elements of T and S and compute their (element-by-element) ratios to obtain the EVs initial position.

```
37  fprintf('Initial position of gen. EV for AR_par = %g:\n', AR_par);
38  t_ii = diag(T);
39  s_ii = diag(S);
40  ratio_of_diag_elements = abs(t_ii ./ s_ii)'
41  initial_pos_of_EV = ratio_of_diag_elements>1
```

```
Initial position of gen. EV for AR_par = 0:

ratio_of_diag_elements = 0.5189   0.5189   1.2621   1.6301   2.0761     0     0

initial_pos_of_EV = 0     0     1     1     1     0     0


Initial position of gen. EV for AR_par = 0.8:

ratio_of_diag_elements = 0.5189   0.5189   1.2621   1.6301   2.0761     0     0.8000

initial_pos_of_EV = 0     0     1     1     1     0     0
```

What we observe is that they aren't in ascending order. Since it is our objective to develop MATLAB-code that exhibits a high degree of automatism, it would be very nice if we don't have to specify the logical vectors for all values of the autocorrelation parameter *manually*, as done so far. Hence, try to develop a piece of code which generates the logical vectors *automatically* using a `for`-loop. What you should search for is a process that assigns the indices in the following manner: the smaller the eigenvalue the higher its assigned index. Make use of the following code structure (try to understand what it does and use the revealed information):

```
x = [];
for t = 0:10
    x = [x t]
end
```

Hint: Compare each eigenvalue with the others to obtain its relative position within the entire set of eigenvalues. For instance, if you have a set of EV consisting of $i = 1, \ldots, 7$ elements, check for all elements how many are greater than element i [What is the MATLAB-output if you compare a vector to a value?]. For example, look at the first EV for $\varphi^{\nu} = 0$ (i.e. 0.5189), its relative position is determined by the comparison with the other EV. Since 3 EV are greater, the algorithm should assign the cluster index 3 to it. The general mechanism evaluates the relative position of EV at initial position i by the comparison with all other EV and assigns an index to it according to the amount of EV which are greater than EV i. Therefore, the smaller the EV i is, the more EV are greater than i and the higher is the cluster index assigned to i. The subsequent rearrangement orders the clusters in such a way that their indices appear in descending order, i.e. the EV appear in ascending order (see: `help lt` and `help sum`).

```
42  fprintf('Generate logical vector for AR_par = %g:\n', AR_par);
43  logic_vector = []; % create an empty-vector
44  for p = 1:length(ratio_of_diag_elements)
45          logic_vector = [logic_vector sum(ratio_of_diag_elements(p) < ratio_of_diag_elements)]
46  end
```

```
Generate logical vector for AR_par = 0:

logic_vector = 3
logic_vector = 3     3
logic_vector = 3     3     2
logic_vector = 3     3     2     1
logic_vector = 3     3     2     1     0
logic_vector = 3     3     2     1     0     5
logic_vector = 3     3     2     1     0     5     5
```

```
Generate logical vector for AR_par = 0.8:

logic_vector = 4
logic_vector = 4     4
logic_vector = 4     4     2
logic_vector = 4     4     2     1
logic_vector = 4     4     2     1     0
logic_vector = 4     4     2     1     0     6
logic_vector = 4     4     2     1     0     6     3
```

Once you have generated the logical vectors you can rearrange the rows of S, T, Q and Z (see: `help ordqz`) and show that the eigenvalues actually appear in ascending order.

```
47  fprintf('Rearrange eigenvalues in ascending order for AR_par = %g:\n', AR_par);
48  [S_re,T_re,Q_re,Z_re] = ordqz(S,T,Q,Z,logic_vector)
49  t_ii_re = diag(T_re);
50  s_ii_re = diag(S_re);
51  ratio_of_diag_elements_re = abs(t_ii_re ./ s_ii_re)'
52  rearranged_pos_of_EV = ratio_of_diag_elements_re>1
```

```
Rearrange eigenvalues in ascending order for AR_par = 0:

S_re =
  -0.9995 - 0.0328i   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i
         0            0.9946 + 0.1040i   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
         0                  0            0.8117 - 0.3983i  -0.0770 + 0.1932i   0.2757 - 0.0791i  -0.0218 - 0.1550i   0.2584 - 0.0320i
         0                  0                  0            0.5683 + 0.2788i   0.4610 + 0.1980i   0.4899 + 0.3069i   0.2943 + 0.1012i
         0                  0                  0                  0            1.3637            -0.1311            -0.2737
         0                  0                  0                  0                  0            0.5974            -0.1875
         0                  0                  0                  0                  0                  0            0.5362

T_re =
   0.0000            0.0000 - 0.0000i  -0.2712 - 0.0316i   0.2238 + 0.3111i  -0.1784 - 0.6415i  -0.1395 - 0.5016i   0.0679 + 0.2441i
         0            0.0000            -0.1033 + 0.1917i   0.2090 + 0.4971i   0.1846 + 0.5731i  -0.0928 - 0.2881i   0.0239 + 0.0743i
         0                  0            0.4692            0.4432 + 0.1253i   0.3138 - 0.2698i   0.1278 + 0.2128i  -0.8261 - 0.1792i
         0                  0                  0            0.3284            1.1114 + 0.5849i  -0.6005 - 0.4072i  -0.0223 + 0.2395i
         0                  0                  0                  0            1.7211            -0.1759            0.6590
         0                  0                  0                  0                  0            0.9738            0.2237
         0                  0                  0                  0                  0                  0            1.1132

Q_re =
         0            0.0000 - 0.0000i   0.2666 + 0.9587i  -0.0267 - 0.0959i  -0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i
  -0.1377 - 0.4274i  -0.0000 + 0.0000i   0.0273 + 0.0846i   0.2726 + 0.8463i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.1443 - 0.1504i  -0.8648 - 0.2093i   0.0072 - 0.0075i   0.0722 - 0.0752i  -0.1734 - 0.0677i   0.3202 - 0.0102i   0.0520 - 0.1178i
   0.5970 + 0.3216i   0.0472 + 0.2939i   0.0299 + 0.0161i   0.2985 + 0.1608i   0.0935 + 0.1104i   0.2688 + 0.0666i   0.4227 + 0.2431i
  -0.3876            0.1529            -0.0194            -0.1938            0.3535             0.8111             0.0746
   0.3070            0.2408             0.0153             0.1535           -0.3984             0.3785            -0.7225
   0.2249           -0.1958             0.0112             0.1125            0.8128            -0.1398            -0.4670

Z_re =
         0           -0.1814 + 0.4107i   0.1771 + 0.0646i   0.3887           -0.1380             0.6455             0.4195
         0 - 0.0000i   0.0000 - 0.0000i  -0.6187 + 0.5143i   0.1350 - 0.3371i   0.0666             0.2884            -0.3651
  -0.2979 + 0.9494i   0.0359 - 0.0813i   0.0089 + 0.0032i   0.0194 - 0.0000i  -0.0069             0.0323             0.0210
   0.0298 - 0.0949i   0.3591 - 0.8133i   0.0885 + 0.0323i   0.1944 + 0.0000i  -0.0690             0.3228             0.2098
  -0.0000 + 0.0000i  -0.0000 + 0.0000i  -0.4057 + 0.0982i  -0.3940 - 0.1121i   0.2726            -0.0956             0.7580
   0.0000 - 0.0000i   0.0000 - 0.0000i  -0.0858 + 0.2691i   0.4178 - 0.1368i  -0.5955            -0.5517             0.2608
  -0.0000 - 0.0000i   0.0000 - 0.0000i   0.1782 + 0.1499i   0.5624 - 0.0386i   0.7367            -0.2852             0.0617


ratio_of_diag_elements_re = 0.0000    0.0000    0.5189    0.5189    1.2621    1.6301    2.0761

rearranged_pos_of_EV = 0     0     0     0     1     1     1


Rearrange eigenvalues in ascending order for AR_par = 0.8:

S_re =
  -0.9995 - 0.0328i   0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i
         0            0.8252 - 0.4049i   0.0941 + 0.0525i  -0.1351 - 0.0056i   0.1464 + 0.2669i   0.1080 - 0.0822i   0.0391 + 0.2042i
         0                  0            0.7532 + 0.3696i   0.2907 + 0.0902i  -0.3467 + 0.0136i  -0.2218 - 0.1091i  -0.1088 + 0.0604i
         0                  0                  0            0.9102            0.9017             0.3227             0.1673
         0                  0                  0                  0            0.9098            -0.3229            -0.4768
         0                  0                  0                  0                  0            0.6691            -0.0649
         0                  0                  0                  0                  0                  0            0.5851
```

```
T_re =
   0.0000            -0.0455 + 0.2476i  -0.1465 - 0.2960i   0.0046 + 0.0164i  -0.1930 - 0.6941i  -0.1362 - 0.4898i   0.0598 + 0.2149i
        0             0.4769             0.0800 - 0.2977i   0.2892 + 0.6634i   0.3774 + 0.2043i  -0.2968 + 0.2145i   0.0961 - 0.6584i
        0                  0             0.4353            -0.7016 + 0.0850i  -0.8349 - 0.1715i   0.6109 + 0.2951i  -0.1261 - 0.3470i
        0                  0                  0             0.7282             1.4005            -0.3543             0.3014
        0                  0                  0                  0             1.1483            -0.3400             0.3530
        0                  0                  0                  0                  0             1.0907             0.4489
        0                  0                  0                  0                  0                  0             1.2147

Q_re =
        0             0.0000 - 0.0000i   0.2666 + 0.9587i  -0.0267 - 0.0959i  -0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 + 0.0000i
        0             0.1377 - 0.7952i   0.0359 + 0.0302i   0.3590 + 0.3017i   0.0509 - 0.1610i   0.0284 + 0.2892i   0.1097 + 0.0400i
        0            -0.1987 - 0.4340i  -0.0807 - 0.0117i  -0.8073 - 0.1173i  -0.0897 - 0.1036i  -0.0962 + 0.1040i  -0.2404 - 0.0582i
   0.4484             0.1122            -0.0269            -0.2689             0.2429             0.6631             0.4632
  -0.7244             0.1102            -0.0040            -0.0400             0.2900             0.5457            -0.2822
   0.3881             0.2465             0.0138             0.1382            -0.4469             0.3741            -0.6555
   0.3516            -0.1881             0.0108             0.1081             0.7809            -0.1343            -0.4487

Z_re =
        0                  0                  0             0.4082            -0.2547             0.6383             0.6009
        0 - 0.0000i   0.4356 + 0.6005i  -0.3388 + 0.3355i  -0.0089             0.0724             0.3372            -0.3215
  -0.2979 + 0.9494i   0.0174 - 0.0394i  -0.0602 - 0.0220i  -0.0540             0.0133             0.0224             0.0185
   0.0298 - 0.0949i   0.1741 - 0.3943i  -0.6018 - 0.2196i  -0.5402             0.1326             0.2245             0.1847
  -0.0000 + 0.0000i   0.0657 + 0.3793i   0.1612 + 0.2116i  -0.3035             0.4136            -0.2692             0.6674
   0.0000 - 0.0000i   0.2423 + 0.0953i  -0.3641 + 0.0535i   0.0834            -0.6700            -0.5368             0.2296
  -0.0000 - 0.0000i   0.1487 - 0.1549i  -0.3802 - 0.0861i   0.6629             0.5405            -0.2595             0.0544

ratio_of_diag_elements_re = 0.0000    0.5189    0.5189    0.8000    1.2621    1.6301    2.0761

rearranged_pos_of_EV = 0     0     0     0     1     1     1
```

Step 5 - Check whether the Blanchard-Kahn Condition is satisfied

To see at a glance if there are exactly as much unstable eigenvalues as non-predetermined variables we can check if the BKC is fulfilled. Note that we need exactly 3 unstable eigenvalues to avoid indeterminacy problems because the non-predetermined state vector v_t consists of 3 variables.

```
53 fprintf('Check Blanchard-Kahn for AR_par = %g:\n', AR_par);
54 Number_of_unstable_EV = sum(rearranged_pos_of_EV)
55 if sum(rearranged_pos_of_EV) == 3
56    fprintf('==> Blanchard-Kahn Condition for AR_par = %g is satisfied\r\n', AR_par);
57 else
58     error('==> Blanchard-Kahn Condition is not satisfied')
59 end
```

```
Check Blanchard-Kahn for AR_par = 0:

Number_of_unstable_EV = 3

==> Blanchard-Kahn Condition for AR_par = 0 is satisfied

Check Blanchard-Kahn for AR_par = 0.8:

Number_of_unstable_EV = 3

==> Blanchard-Kahn Condition for AR_par = 0.8 is satisfied
```

Step 6 - Computing the solution time paths for x, π, τ_t and ν_t

Since

$$\begin{pmatrix} w_t \\ v_t \end{pmatrix} = \begin{pmatrix} Z_{11} \\ Z_{21} \end{pmatrix} z_t$$

and

$$E_t z_{t+1} = S_{11}^{-1} T_{11} z_t$$

define the required matrices of the subsystems:

```
T_11 = T_re(1:4,1:4);
S_11 = S_re(1:4,1:4);
Z_11 = Z_re(1:4,1:4);
P = 30;
```

Also define matrices which will contain numerical solutions:

```
w_solution = NaN(4,P);
z_solution = NaN(4,P);
```

Now define the initial values of w_t and z_t. Since

$$w_0 = \begin{pmatrix} \nu_0 \\ x_{-1} \\ \tau_{-1} \\ \pi_{-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$z_0 = Z_{11}^{-1} w_0$$

set the first columns, i.e. the initial jumps $E_0 w_0$ and $E_0 z_0$:

```
w_solution(:,1) = [1; 0; 0; 0];
z_solution(:,1) = inv(Z_11) * w_solution(:,1);
```

Create a `for`-loop to compute solution time paths

```
    for t = 2:P
        z_solution(:,t) = inv(S_11) * T_11 * z_solution(:,t-1);
        w_solution(:,t) = Z_11 * z_solution(:,t);
    end
```

Step 7 - Plots

The solution time paths will be plotted till $t = 20$ periods and only the real parts of the numerical solutions are taken into account (what happens if you run the code without the latter commands?).

```
    t = 0:20;
    w_solution = real(w_solution);
```

Now we just have to specify how the plots have to look like (see: `help subplot` and `help plot`)

```
     subplot(2,2,1); hold on;
     plot(t, w_solution(2,t+2), format(AR_par_pos,:), 'LineWidth',1)
     xlabel('t'); ylabel('x'); title('output gap')
     hline = refline([0 0]);
     set(hline,'Color','r')
     box on

     subplot(2,2,2); hold on;
     plot(t, w_solution(4,t+2), format(AR_par_pos,:), 'LineWidth',1)
     xlabel('t'); ylabel('\pi'); title('inflation')
     hline = refline([0 0]);
     set(hline,'Color','r')
     box on

     subplot(2,2,3); hold on;
     plot(t, w_solution(3,t+2), format(AR_par_pos,:), 'LineWidth',1)
     xlabel('t'); ylabel('\tau'); title('terms of trade')
     hline = refline([0 0]);
     set(hline,'Color','r')
     box on


```

```
96      subplot(2,2,4); hold on;
97      plot(t, w_solution(1,t+1), format(AR_par_pos,:), 'LineWidth',1)
98      xlabel('t'); ylabel('\nu'); title('Shock process')
99      hline = refline([0 0]);
100     set(hline,'Color','r')
101     box on
102 end                              % end of the initial for-loop
```

Don't forget to close the initial `for`-loop!

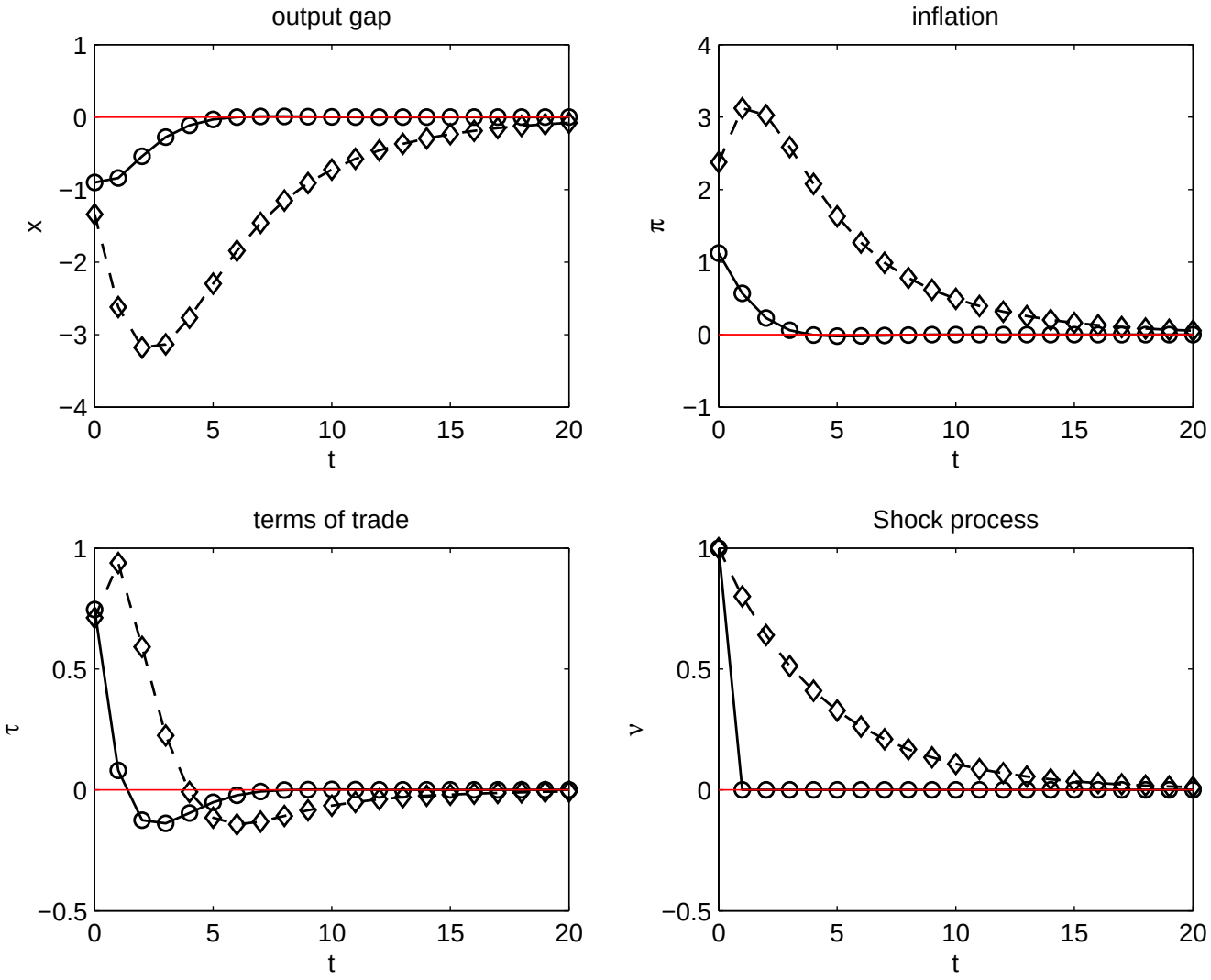

Figure 1: Adjustment time paths in response to a temporary cost-push shock shock with $\varphi^{\nu} = 0$ and $\varphi^{\nu} = 0.8$.