Jordan vs. Schur Decomposition

In order to give an overview of decompositions of the system matrix A, the following table 1 shows the results for its defective version, whereas table 2 shows the results after the slightly change due to a changed parameterization. Both versions of system matrix A are decomposed each with the Jordan decomposition (manually as well with MATLAB's built-in command) and with the Schur decomposition. Compare the development of the decomposition solutions of the successively implemented decomposition algorithms. What do you observe?

Decomposition Method	EV Matrix (λ or T)	Auxiliary Matrix (H or Z)	Rank of Auxiliary Matrix
Jordan (manually)	$\Lambda = \begin{pmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$H = \begin{pmatrix} 0.8944 & -0.8944 & 0.8944 \\ -0.4472 & 0.4472 & -0.4472 \\ 0 & 0 & 0 \end{pmatrix}$	1
Jordan (built-in)	$\Lambda = \begin{pmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$H = \begin{pmatrix} 2 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	3
Schur (built-in)	$T = \begin{pmatrix} -2 & 5 & -2.2361 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$Z = \begin{pmatrix} 0.8944 & 0.4472 & 0 \\ -0.4472 & 0.8944 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	3

Table 1: Decomposition of the defective matrix $A = \begin{pmatrix} 0 & 4 & -2 \\ -1 & -4 & 1 \\ 0 & 0 & -2 \end{pmatrix}$.

Decomposition Method	EV Matrix (λ or T)	Auxiliary Matrix (H or Z)	Rank of Auxiliary Matrix
Jordan (manually)	$\Lambda = \begin{pmatrix} -3 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$H = \begin{pmatrix} -0.8165 & 0.8165 & -0.7071 \\ 0.4082 & -0.4082 & 0 \\ -0.4082 & -0.4082 & -0.7071 \end{pmatrix}$	3
Jordan (built-in)	$\Lambda = \begin{pmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$H = \begin{pmatrix} 2 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	3
Schur (built-in)	$T = \begin{pmatrix} -2 & 5 & -2.2361 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$	$Z = \begin{pmatrix} 0.8944 & 0.4472 & 0 \\ -0.4472 & 0.8944 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	3

Table 2: Decomposition of the slightly changed (yet non-defective) matrix $A = \begin{pmatrix} 0 & 4 & -2 \\ -1 & -4 & 1 \\ 0 & 1 & -2 \end{pmatrix}$

Task 1: Simulate the classical model with capital using the Schur decomposition instead of the Jordan decomposition. You can re-use large parts of the MATLAB-code from the first problem set.

Simulating the Hybrid Version of the New Keynesian Model 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 classical model with capital in response to a temporary technology shock ($\rho_a = 0.2$) 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 `rbc_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.

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

Step 1 - Parameterization

Set the parameters of the model as follows:

```
phi    = 0.5;
sig    = 1;
gam_f  = 0.5;
gam_b  = 0.5;
gam_x  = 0.2;
del_pi = 1.5;
del_x  = 0.5;
AR_par = 0.8;
```

Step 2 - Specification of the System Matrix A

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

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

Specify system matrix A.

```
% System is: [w; v](+1) = A * [w; v] + [1;0;0;0;0] * eps
  A11 = [AR_par 0 0; 0 0 0; 0 0 0];
  A12 = [0 0; 1 0; 0 1];
  A21 = [1/((1-phi)*sig*gam_f) -phi/(1-phi) gam_b/((1-phi)*sig*gam_f); ...
         -1/gam_f  0  -gam_b/gam_f];
  A22 = [1/(1-phi)*(1+del_x/sig+gam_x/(sig*gam_f))  1/(1-phi)*(del_pi/sig-1/(sig*gam_f)); ...
         -gam_x/gam_f  1/gam_f];
  A   = [  [A11 A12]  ;  [A21 A22]  ]
```

```
A =
    0.8000         0         0         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000
    4.0000   -1.0000    2.0000    3.8000   -1.0000
   -2.0000         0   -1.0000   -0.4000    2.0000
```

Step 3 - Application of the Schur Decomposition

In order to solve the state equations, the eigenvalues of A must be computed. Since the state equations are of the form

$$x_{t+1} = Ax_t,$$

the corresponding (standard) eigenvalue problem

$$Av = \lambda v$$

must be solved. This can be achieved by the application of the Schur decomposition (see: `help schur`) which transforms the system matrix A in such a way that its sought-after eigenvalues appear (even if somewhat randomly) on the principal diagonal of the upper triangular matrix T.

Please compute the Schur matrix T as well as the corresponding unitary matrix Z (use the complex flag).

```
disp('Schur Decomposition')
[Z, T] = schur(A, 'complex')
```

```
Schur Decomposition
Z =
   0.0000           -0.0000           -0.1926            0.3756 - 0.3486i    0.1594 - 0.8215i
  -0.2607            0.0439            0.8770            0.0070 - 0.3645i    0.1667 - 0.0154i
   0.0550           -0.5994           -0.2556           -0.1418 - 0.6141i    0.2808 + 0.3102i
  -0.9431           -0.2099           -0.1937            0.0124 + 0.1525i   -0.0697 - 0.0272i
   0.1991           -0.7712            0.3014            0.1073 + 0.4150i   -0.1898 - 0.2346i

T =
   3.6180            1.0383            3.3444           -1.1630 + 2.3718i   -1.0844 + 2.5438i
        0            1.3820           -0.5369            0.0028 - 1.4069i    0.6433 - 0.0062i
        0                 0            0.8000           -0.4180 + 1.1139i   -0.5093 + 0.9142i
        0                 0                 0            0.4000 + 0.2000i    0.3460
        0                 0                 0                 0              0.4000 - 0.2000i
```

Step 4 - Rearrangement of Schur Matrix T

Since the eigenvalues of the system appear randomly on the main diagonal of the Schur matrix T, rearrange its rows in such a way that the first (upper) subsystem contains all *stable* eigenvalues $\lambda_i < 1$ (and therefore determines the dynamics of the predetermined state vector w_t) and the second (lower) subsystem contains all *unstable* eigenvalues $\lambda_i > 1$ (and therefore determines the dynamics of the non-predetermined state vector v_t), i.e.

$$T = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}$$

$$T = \begin{pmatrix} |\lambda_1| < 1 & & & & \\ & |\lambda_2| < 1 & & T_{12} & \\ & & |\lambda_3| < 1 & & \\ & 0 & & |\lambda_4| > 1 & \\ & & & & |\lambda_5| > 1 \end{pmatrix}$$

(see: `help ordschur`). You will have to specify a corresponding logical vector of cluster indices appropriate to the appearance of the computed eigenvalues to set their desired order. A cluster is defined as sequence of eigenvalues which are already in *as*cending order[2]. All eigenvalues with the same cluster index form a distinct cluster[3]. `ordschur` provides the possibility to reorder multiple clusters at once. This logical vector will sort the specified clusters in *de*scending order which means that the cluster with the highest index appears in the upper left corner of T.

```
19  disp('Rearrange eigenvalues in ascending order along the principal diagonal')
20  [Z_re, T_re] = ordschur(Z,T,[1 2 3 4 4]) % [logical vector with cluster indices]
```

[2] Such a cluster can also consist of only a single element.
[3] In the case of an $n \times n$-matrix with n eigenvalues you can specify 1 to n clusters.

Additionally, it has to be checked whether there are any errors in the Schur decomposition. This is done by checking if

$$A = Z \cdot T \cdot \overline{Z}'$$

and

$$\overline{Z}' \cdot Z = I_{5\times 5}$$

holds.

```
21  if abs(sum(sum(Z_re*T_re*Z_re'-A))) > 0.0001 && sum(Z_re'*Z_re-eye(length(Z_re))) > 0.0001
22      error('Error in Schur decomposition')
23  end
```

```
Rearrange eigenvalues in ascending order along the principal diagonal

Z_re =
   0.0000 - 0.0000i   0.0000 - 0.0000i  -0.1043 + 0.4431i  -0.3665 + 0.1843i   0.2472 - 0.7506i
  -0.7813 + 0.2898i  -0.3556 + 0.3062i  -0.0016 + 0.0066i  -0.2302 + 0.1158i  -0.0430 + 0.1307i
   0.2722 + 0.2546i  -0.7090 - 0.1706i   0.1101 - 0.4678i   0.0533 - 0.0268i   0.0963 - 0.2923i
  -0.3705 - 0.0403i   0.2478 + 0.1772i   0.0589 - 0.2504i   0.6021 - 0.3028i   0.1558 - 0.4730i
   0.0580 + 0.1563i  -0.3901 - 0.0547i  -0.1613 + 0.6853i   0.4955 - 0.2492i  -0.0368 + 0.1117i

T_re =
   0.4000 + 0.2000i  -0.3768 - 0.1188i   0.1242 + 0.8124i  -0.7484 - 0.2468i  -0.3973 + 2.7650i
        0             0.4000 - 0.2000i  -0.1570 - 1.1001i  -1.0288 + 0.3432i  -0.5682 - 2.5881i
        0                  0             0.8000           -0.3368 - 0.4021i   2.8008 + 0.2440i
        0                  0                  0            1.3820             1.6308 - 1.6351i
        0                  0                  0                 0             3.6180
```

Now T contains the eigenvalues of A in order of increasing magnitude.

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 2 unstable eigenvalues to avoid indeterminacy problems because the non-predetermined state vector v_t consists of 2 variables.

```
24  disp('check Blanchard-Kahn')
25  eigenvalues = abs(diag(T_re))'
26  EV_A = eigenvalues > 1
27  if sum(eigenvalues > 1) == 2
28      disp('==> Blanchard-Kahn Condition is satisfied')
29  else
30      error('==> Blanchard-Kahn Condition is not satisfied')
31  end
```

```
check Blanchard-Kahn

eigenvalues =
    0.4472    0.4472    0.8000    1.3820    3.6180

EV_A =
     0     0     0     1     1

==> Blanchard-Kahn Condition is satisfied
```

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

Since

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

and

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

define the required matrices of the subsystems:

```
T_11 = T_re(1:3,1:3);
Z_11 = Z_re(1:3,1:3);
Z_21 = Z_re(4:5,1:3);
P = 30;
```

Also define matrices which will contain numerical solutions:

```
w_solution = NaN(3,P); % Define w as a 3-by-P matrix of NaNs (3 predet. variables)
z_solution = NaN(3,P); % (auxiliaries of 3 predet. variables)
v_solution = NaN(2,P); % Define v as a 2-by-P matrix (2 non-predet. variables)
i_solution = NaN(1,P); % Define i as a 1-by-P vector (nominal interest rate)
r_solution = NaN(1,P); % Define r as a 1-by-P vector (real interest rate)
```

These matrices/vectors now look like

$$\begin{aligned}
w_{solution} &= \begin{pmatrix} \mathrm{NaN}_{11} & \dots & \mathrm{NaN}_{1P} \\ \mathrm{NaN}_{21} & \dots & \mathrm{NaN}_{2P} \\ \mathrm{NaN}_{31} & \dots & \mathrm{NaN}_{3P} \end{pmatrix} = \begin{pmatrix} \nu_0 & \nu_1 & \dots & \nu_P \\ \tilde{x}_0 = x_{-1} & x_0 & \dots & \tilde{x}_{P-1} \\ \tilde{\pi}_0 = \pi_{-1} & \pi_0 & \dots & \tilde{\pi}_{P-1} \end{pmatrix} \\
z_{solution} &= \begin{pmatrix} \mathrm{NaN}_{11} & \dots & \mathrm{NaN}_{1P} \\ \mathrm{NaN}_{21} & \dots & \mathrm{NaN}_{2P} \\ \mathrm{NaN}_{31} & \dots & \mathrm{NaN}_{3P} \end{pmatrix} = \begin{pmatrix} Z_{11}^{-1}\nu_0 & Z_{11}^{-1}\nu_1 & \dots & Z_{11}^{-1}\nu_P \\ Z_{11}^{-1}\tilde{x}_0 = Z_{11}^{-1}x_{-1} & Z_{11}^{-1}x_0 & \dots & Z_{11}^{-1}\tilde{x}_{P-1} \\ Z_{11}^{-1}\tilde{\pi}_0 = Z_{11}^{-1}\pi_{-1} & Z_{11}^{-1}\pi_0 & \dots & Z_{11}^{-1}\tilde{\pi}_{P-1} \end{pmatrix} \\
v_{solution} &= \begin{pmatrix} \mathrm{NaN}_{11} & \dots & \mathrm{NaN}_{1P} \\ \mathrm{NaN}_{21} & \dots & \mathrm{NaN}_{2P} \end{pmatrix} = \begin{pmatrix} \mathrm{NaN}_{11} & x_1 & \dots & x_P \\ \mathrm{NaN}_{21} & \pi_1 & \dots & \pi_P \end{pmatrix} \\
i_{solution} &= \begin{pmatrix} \mathrm{NaN}_1 & \dots & \mathrm{NaN}_P \end{pmatrix} = \begin{pmatrix} \mathrm{NaN}_1 & i_1 & \dots & i_P \end{pmatrix} \\
r_{solution} &= \begin{pmatrix} \mathrm{NaN}_1 & \dots & \mathrm{NaN}_P \end{pmatrix} = \begin{pmatrix} \mathrm{NaN}_1 & r_1 & \dots & r_P \end{pmatrix}
\end{aligned}$$

Now define the initial values of w_t and z_t. Since

$$\begin{aligned}
w_0 &= \begin{pmatrix} \varepsilon_0 \\ x_{-1} \\ \pi_{-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\
z_0 &= Z_{11}^{-1} w_0
\end{aligned}$$

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

```
    w_solution(:,1) = [1; 0; 0];                    % Set the first column of w
    z_solution(:,1) = inv(Z_11) * w_solution(:,1); % Set the first column of z
```

Now they look like

$$\begin{aligned}
w_{solution} &= \begin{pmatrix} 1 & \mathrm{NaN}_{12} & \dots & \mathrm{NaN}_{1P} \\ 0 & \mathrm{NaN}_{22} & \dots & \mathrm{NaN}_{2P} \\ 0 & \mathrm{NaN}_{32} & \dots & \mathrm{NaN}_{3P} \end{pmatrix} \\
z_{solution} &= \begin{pmatrix} Z_{11}^{-1} \cdot 1 & \mathrm{NaN}_{12} & \dots & \mathrm{NaN}_{1P} \\ Z_{11}^{-1} \cdot 0 & \mathrm{NaN}_{22} & \dots & \mathrm{NaN}_{2P} \\ Z_{11}^{-1} \cdot 0 & \mathrm{NaN}_{32} & \dots & \mathrm{NaN}_{3P} \end{pmatrix}
\end{aligned}$$

The system dynamics are entirely described by the predetermined variables since v_t can be written as linear function of w_t.

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

```
    for t = 2:P
        z_solution(:,t) = T_11 * z_solution(:,t-1);
        w_solution(:,t) = Z_11 * z_solution(:,t);
        v_solution(:,t) = Z_21 * inv(Z_11) * w_solution(:,t);
        i_solution(:,t) = del_pi * w_solution(3,t) + del_x * w_solution(2,t); % TR
        r_solution(:,t) = i_solution(:,t) - v_solution(2,t);
    end
```

Note that since the first columns of the solution matrices are already reserved for the initial jumps of the variables the numerical solutions are computed exclusive of that first column, i.e. from column 2 to column P.

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);
    v_solution = real(v_solution);
    i_solution = real(i_solution);
    r_solution = real(r_solution);
```

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

```
    figure

    subplot(3,2,1); hold on;
    plot(t, w_solution(2,t+2), 'k-o', 'LineWidth',1)
    xlabel('t'); ylabel('x'); title('output gap')
    hline = refline([0 0]);
    set(hline,'Color','r')
    box on

    subplot(3,2,2); hold on;
    plot(t, w_solution(3,t+2), 'k-o', 'LineWidth',1)
    xlabel('t'); ylabel('\pi'); title('inflation')
    hline = refline([0 0]);
    set(hline,'Color','r')
    box on

    subplot(3,2,3); hold on;
    plot(t, i_solution(t+2), 'k-o', 'LineWidth',1)
    xlabel('t'); ylabel('i'); title('nominal interest rate')
    hline = refline([0 0]);
    set(hline,'Color','r')
    box on

    subplot(3,2,4); hold on;
    plot(t, r_solution(t+2), 'k-o', 'LineWidth',1)
    xlabel('t'); ylabel('r'); title('real interest rate')
    hline = refline([0 0]);
    set(hline,'Color','r')
    box on

    subplot(3,2,5); hold on;
```

```
86      plot(t, w_solution(1,t+1), 'k-o', 'LineWidth',1)
87      xlabel('t'); ylabel('\nu'); title('Shock process')
88      hline = refline([0 0]);
89      set(hline,'Color','r')
90      box on
```

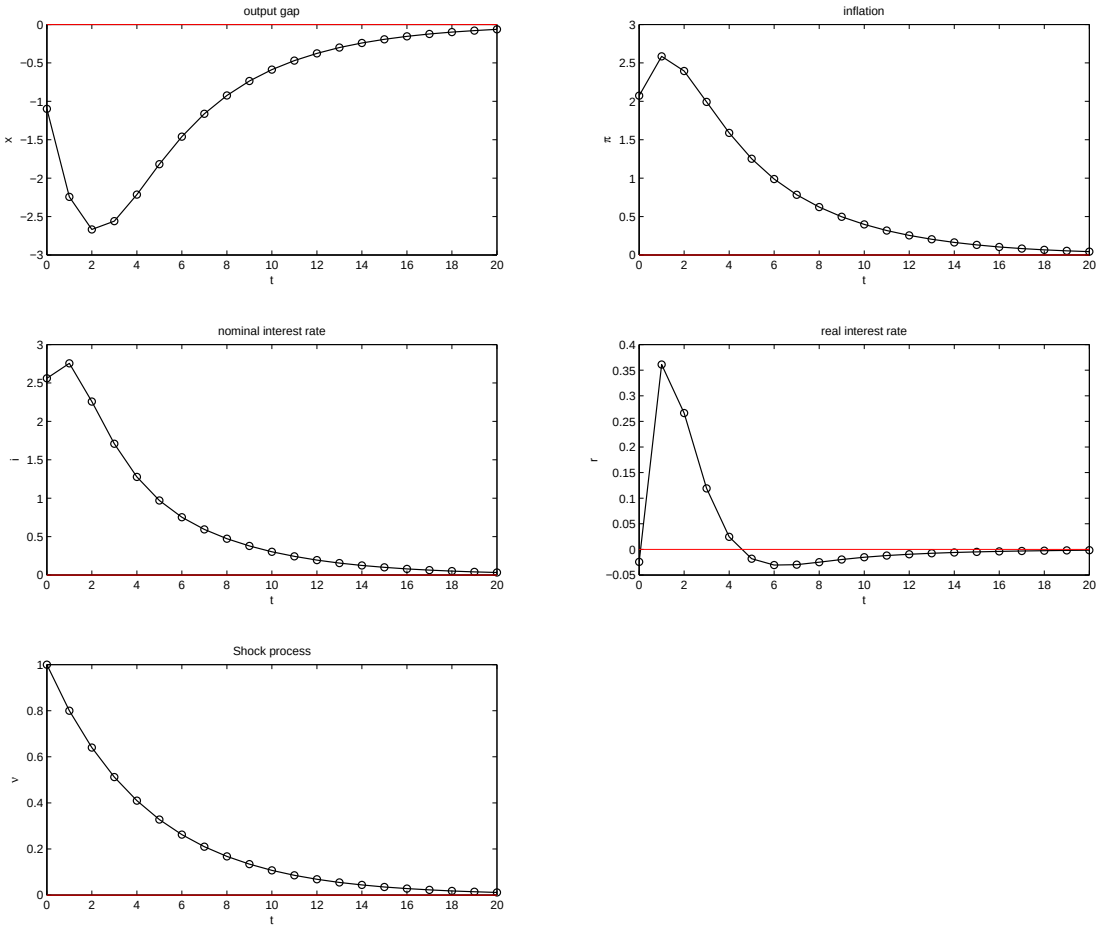

Figure 1: Adjustment time paths in response to a temporary cost-push shock shock with ($\varphi^{\nu} = 0.8$).