hw07: Schur decomposition and autonomous LRE model

Kenji Sato[*]

05 May, 2017

1 Overview

Purpose

Learn about

- canonical forms of matrices and
- simulation of LRE models without inputs

Prerequisite

Install the following R packages

- **QZ** for QZ decomposition

by executing

```
install.packages("QZ")
```

Instructions

In this assignment, you will

- clone the assignment repository and make a working branch (eg. `solution` branch);
- solve the problems in Section 4;
- write the solutions in `solution.Rmd` and knit it;
- commit `solution.Rmd` and the derived files; and

[*]Kobe University. Email: mail@kenjisato.jp

- open a Pull Request.

2 Theory

The dynamic analysis of models in economics involves separation of unstable subspace from stable subspace. The initial state must be chosen in the (weakly) stable subspace so that the unstable components continue to be zero.

Let's assume that the endogenous variables

$$x_t = \begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix} \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$$

have n_1 predetermined variables, x_t^1, and non-predetermined variables, x_t^2.

The simplest autonomous system is given as follows:

$$x_{t+1} = Ax_t \Leftrightarrow \begin{bmatrix} x_{t+1}^1 \\ x_{t+1}^2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix},$$

where the predetermined variables at time $t = 0$, x_0^1, are given as initial conditions.

The whole solution to the system can be simulated by repeating the following two steps.

- Compute the non-predetermined from the predetermined; $x_t^1 \mapsto x_t^2$
- Compute the predetermined of the next period; $(x_t^1, x_t^2) \mapsto x_{t+1}^1$

The common goal of many different solution methods is to obtain the following system equations:

$$x_t^2 = g(x_t^1) \tag{1}$$

$$x_{t+1}^1 = h(x_t^1, x_t^2) = h(x_t). \tag{2}$$

Code similar to the following will produce the simulation result. (`x0` is for x_0^1 and `t` is the simulation length. Since I didn't test it, copying-and-pasting may not work.)

```r
simulate <- function(g, h, x0, t) {

  n1 <- length(x0)     # Number of predetermined variables
  n2 <- length(g(x0))  # Number of non-predetermined variables

  pre <- 1:n1
  npr <- (n1 + 1):(n1 + n2)
  out <- matrix(0, t, n1 + n2)  # Zero matrix for simulation output

  out[1, pre] <- x0      # Initial Condition
  out[1, npr] <- g(x0)   # Eq. (2.1)

  for (i in 1:(t - 1)) {
    out[i + 1, pre] <- h(out[i, ])          # Eq. (2.2)
    out[i + 1, npr] <- g(out[i + 1, pre])  # Eq. (2.1)
  }
  out
}
```

The only remaining problem is to find g and h above.

2.1 Basic Idea: Block diagonalization

Let's see how canonical forms work for the autonomous system.

$$x_{t+1} = Ax_t \Leftrightarrow \begin{bmatrix} x_{t+1}^1 \\ x_{t+1}^2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix}.$$

Decompose this system, by using Jordan canonical form for example, into two components

$$\begin{bmatrix} y_{t+1}^s \\ y_{t+1}^u \end{bmatrix} = \begin{bmatrix} \Lambda_s & \\ & \Lambda_u \end{bmatrix} \begin{bmatrix} y_t^s \\ y_t^u \end{bmatrix}, \quad \mathrm{sp}(\Lambda_s) \subset \mathrm{cl}\,\mathbb{D},\ \mathrm{sp}(\Lambda_u) \subset (\mathrm{cl}\,\mathbb{D})^c$$

with

$$AV = V\Lambda$$
$$\Updownarrow$$
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} V_{1s} & V_{1u} \\ V_{2s} & V_{2u} \end{bmatrix} = \begin{bmatrix} V_{1s} & V_{1u} \\ V_{2s} & V_{2u} \end{bmatrix} \begin{bmatrix} \Lambda_s & \\ & \Lambda_u \end{bmatrix}.$$

$$\begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix} = \begin{bmatrix} V_{1s} & V_{1u} \\ V_{2s} & V_{2u} \end{bmatrix} \begin{bmatrix} y_t^s \\ y_t^u \end{bmatrix}$$

Since the stability condition is given by $y_0^u = \cdots = y_t^u = \cdots = 0$, we have

$$x_t^2 = V_{2s}y_t^s + V_{2u}y_t^u = V_{2s}y_t^s = V_{2s}V_{1s}^{-1}x_t^1,$$

assuming V_{1s} is square and non-singular.

We can perform the simulation with the following recursive formula:

$$\begin{aligned} x_t^2 &= V_{2s} V_{1s}^{-1} x_t^1 \\ x_{t+1}^1 &= A_{11} x_t^1 + A_{12} x_t^2 \end{aligned}$$

2.2 Shur Decomposition

You might have noticed that block diagonalization is not necessary for the above calculation. To be able to determine the initial vector on the stable subspace, splitting off the unstable subspace is sufficient.

For any square matrix A, there are a block triangular T and unitary Q such that

$$AQ = QT$$
$$\Updownarrow$$
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} Q_{1s} & Q_{1u} \\ Q_{2s} & Q_{2u} \end{bmatrix} = \begin{bmatrix} Q_{1s} & Q_{1u} \\ Q_{2s} & Q_{2u} \end{bmatrix} \begin{bmatrix} T_{ss} & T_{su} \\ & T_{uu} \end{bmatrix}.$$

The transformed dynamic system has the following form

$$\begin{bmatrix} y_{t+1}^s \\ y_{t+1}^u \end{bmatrix} = \begin{bmatrix} T_{ss} & T_{su} \\ & T_{uu} \end{bmatrix} \begin{bmatrix} y_t^s \\ y_t^u \end{bmatrix}$$

with

$$\begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix} = \begin{bmatrix} Q_{1s} & Q_{1u} \\ Q_{2s} & Q_{2u} \end{bmatrix} \begin{bmatrix} y_t^s \\ y_t^u \end{bmatrix}$$

If Q_{1s} is square and non-singular, we can perform the simulation with the following recursive formula:

$$x_t^2 := g(x_t^1) = Q_{2s} Q_{1s}^{-1} x_t^1 \tag{3}$$
$$x_{t+1}^1 := h(x_t) = A_{11} x_t^1 + A_{12} x_t^2 \tag{4}$$

3 Computation

```r
A <- matrix(c(
  0.7000000, 0.000000,  1.2000000,
  0.6363636, 1.909091,  0.1818182,
  0.0000000, -1.000000, 1.0000000
), byrow = TRUE, nrow = 3)
```

3.1 Schur decomposition using Matrix package

You can use **Matrix::Schur()** function to compute the Schur decomposition.

```
(sch <- Matrix::Schur(A))
```

```
## $Q
##            [,1]       [,2]      [,3]
## [1,]  0.8841517 -0.1788923 0.4315940
## [2,] -0.2909858  0.5118779 0.8082749
## [3,] -0.3655176 -0.8402253 0.4005226
##
## $T
##           [,1]       [,2]      [,3]
## [1,] 0.2039074 -0.7147422 0.2909828
## [2,] 0.0000000  1.7025918 1.1703045
## [3,] 0.0000000 -0.4563871 1.7025918
##
## $EValues
## [1] 0.2039074+0.00000i 1.7025918+0.73083i 1.7025918-0.73083i
```

```
all.equal(sch$Q %*% sch$T %*% Conj(t(sch$Q)), A)
```

```
## [1] TRUE
```

3.2 QZ package and reordering

Since Schur decomposition is a special case of QZ decomposition, **QZ** package has a function to compute the Schur decomposition.

```
QZ::qz(A)
```

```
## W:
## [1] 0.2039+0.000i 1.7026+0.731i 1.7026-0.731i
##
## T:
##        [,1]    [,2]  [,3]
## [1,] 0.2039 -0.7147 0.291
## [2,] 0.0000  1.7026 1.170
## [3,] 0.0000 -0.4564 1.703
##
## Q:
##         [,1]    [,2]   [,3]
## [1,]  0.8842 -0.1789 0.4316
## [2,] -0.2910  0.5119 0.8083
## [3,] -0.3655 -0.8402 0.4005
```

3.3 S3 and method dispatch

It seems like the function `Matrix::Schur()` is more accurate than `QZ::qx()`. Is that correct? Let's check it with

```
Matrix::Schur(A)$Q == QZ::qz(A)$Q
```

```
##      [,1] [,2] [,3]
## [1,] TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE
```

Notice that I didn't use `all.equal()` here; the two matrices are exactly identical. As a matter of fact, both `Matrix::Schur()` and `QZ::qz()` use `DGEES` subroutine of LAPACK linear algebra library, written in FORTRAN. (To be precise, `QZ::qz()` calls `DGEES` when one real matrix is passed as an argument. See the source code by typing `QZ::qz` in the console.)

Why did we see different results printed on the screen then? This is due to difference in printing functions, which are implicitly called when the return values are shown. As you can see with `class(QZ::qz(A))`, `QZ::qz()` returned an object of class "dgees". The `print()` function against this S3 object is delegated to `print.dgees()` function. In contrast, the return value of `Matrix::Schur()` is an ordinary list and hence `print()` used for it is slightly different from `print.dgees()`, i.e., `print.dgees()` truncates long floating point numbers more aggressively than default `print()`.

As you have seen, there are functions that behave differently depending on the type of its argument. Such functions are called **generics**. Examples include `print()`, `plot()`, `summary()` and `mean()`. These generics are used as common interface to more specialized functions, or **methods**. This mechanism, known as *method dispatch*, will help you write cleaner code in R. For more information on S3 object system and method dispatch (and more), read *Advanced R* http://adv-r.had.co.nz/OO-essentials.html

3.4 Reordering

The above matrix, A, was transformed into the ideal form in that all the (weakly) stable eigenvalues are collected at the upper-left of the triangular matrix T. This is not what you can always expect, however. Have a look at the following (counter)example.

```
B <- diag(c(1.5, 0.8, 0.5))
(Bsch <- Matrix::Schur(B))
```

```
## $Q
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
## $T
##      [,1] [,2] [,3]
## [1,]  1.5  0.0  0.0
## [2,]  0.0  0.8  0.0
## [3,]  0.0  0.0  0.5
##
## $EValues
## [1] 1.5 0.8 0.5
```

Since the algorithms to simulate LRE models are based on particular arrangement of eigenvalues, we must reorder the transformed matrices.

To this end, we use `QZ::qz.dtrsen()`.

```
QZ::qz.dtrsen(Bsch$T, Bsch$Q, abs(Bsch$EValues) <= 1)
```

```
## W:
## [1] 0.8 0.5 1.5
##
## T:
##      [,1] [,2] [,3]
## [1,]  0.8  0.0  0.0
## [2,]  0.0  0.5  0.0
## [3,]  0.0  0.0  1.5
##
## Q:
##      [,1] [,2] [,3]
## [1,]    0    0    1
## [2,]    1    0    0
## [3,]    0    1    0
```

Of course, you can use the results of `QZ::qz()`.

```
Bqz <- QZ::qz(B)
QZ::qz.dtrsen(Bqz$T, Bqz$Q, abs(Bqz$W) <= 1)
```

```
## W:
## [1] 0.8 0.5 1.5
##
## T:
##      [,1] [,2] [,3]
## [1,]  0.8  0.0  0.0
## [2,]  0.0  0.5  0.0
## [3,]  0.0  0.0  1.5
##
## Q:
##      [,1] [,2] [,3]
## [1,]    0    0    1
## [2,]    1    0    0
## [3,]    0    1    0
```

4 Problems

Problem 1

In the analysis in Section **??**, the assumptions that V_{1s} and Q_{1s} are square and non-singular are crucial. How can you check that V_{1s} and Q_{1s} are square? How is it related to the numbers of predetermined/non-predetermined variables and those of stable/unstable eigenvalues?

Problem 2

Read the documentation for `QZ::qz.dtrsen` (type `?QZ::qz.dtrsen` in the console pane) and compute the Schur decomposition of

```
C <- diag(c(1.5, 0.8, 3.9, 0.5, -0.2, 1.0, 5.9))
```

with T having all the eigenvalues greater than or equal to 1 in the upper-left block. That is, you should get the following T matrix (order within blocks does not matter).

```
T:
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  1.5  0.0    0  0.0  0.0  0.0  0.0
[2,]  0.0  3.9    0  0.0  0.0  0.0  0.0
[3,]  0.0  0.0    1  0.0  0.0  0.0  0.0
[4,]  0.0  0.0    0  5.9  0.0  0.0  0.0
[5,]  0.0  0.0    0  0.0  0.8  0.0  0.0
[6,]  0.0  0.0    0  0.0  0.0  0.5  0.0
[7,]  0.0  0.0    0  0.0  0.0  0.0 -0.2
```

NB: Take it as a mere exercise. You won't use this operation for the simulation.

Problem 3

Implement a function `lre_auto()` that satisfies the following specifications:

- Input
  - `A`: n-by-n matrix
  - `npr`: number of predetermined variables, `0 < npr < n`
- Output
  - g in Eq. (3)
  - h in Eq. (4)

Hint: To return multiple values, return a list.

Problem 4

Using simulation code similar to the one given in Section **??**, simulate $x_{t+1} = Ax_t$ with

```r
A <- matrix(c(
  0.7000000, 0.000000, 1.2000000,
  0.6363636, 1.909091, 0.1818182,
  0.0000000, -1.000000, 1.0000000
), byrow = TRUE, nrow = 3)
```

with the following initial condition

```r
x0 <- 0.1
```

Problem 5 (Optional but highly recommended)

Make a package to do the computation necessary for simulation. You might want to export functions

- to compute Schur decomposition with proper ordering (stable eigenvalues in the upper-left block).
- to compute g and h.
- to simulate the model.

You can design the API as you like.

In case you have no idea where to start, I would suggest the following:

```r
library("lre")
sol <- lre_auto(A, length(x0))
simulate(sol$g, sol$h, x0, t = 100)
```

Note

Since you already have one-month experience of using R and R markdown, I don't provide "fill-in-the-blank" solution sheet in `solution.Rmd`. You can freely add code chunks, reference to external R scripts. If your code depends on a package of your own creation, please provide an installation instruction so that I can install it on my computer.