A User Guide for Matlab Code for an RBC Model Solution and Simulation

Ryo Kato*

Department of Economics, The Ohio State University

and Bank of Japan

Decemnber 10, 2002

Abstract

This note provides an easy and quick instruction for solution and simulation of a standard RBC model using Matlab.

The Matlab code introduced here is extremely compact and easy to hadle in the sense that it requires neither external functions/sub-procedures nor benchmark data. The solution method used in the code is standard undermined coefficient method (eigen decomposition method) based on log-linearized system.

The code is available at `http://economics.sbs.ohio-state.edu/kato/matlab/RBC1.m`

*E-mail: `kato.13@osu.edu` or `ryou.katou@boj.or.jp`

1 Overview

Make sure that your PC is installed with Symbolic Math tool-box in an appropriate folder. Usually, you find a folder named ‘‘`toolbox`’’ under your `MATLAB` folder. The code introduced in this note is only one sheet in length, but everything necessary to solve/simulate an RBC model is contained in the sheet. You do not have to take derivatives to derive the first order conditions here in the code. All those analytical calculations are processed automatically by your Matlab. Generally, what you have to do to run the code is only two things, namely, choosing parameter values and writing down all the first order conditions and resource constraints. Note that those conditions can be non-linear, since, as I mentioned, automatic log-linearization is a part of the code. The Matlab code is downloadable at `http://www.econ.ohio-state.edu/kato/matlab/RBC1.m` .

The code consists of five parts as follows,

1. Parameter proc
2. Steady State proc
3. Model proc
4. Linearization proc
5. Solution proc
6. Simulation proc

If you change the Model part, then you have to modify the Steady-state part accordingly. In most cases, you do not have to change the Linearization part and Solution part. There are two versions for the Simulation part, that is, (1) impulse response and (2) stochastic simulations to obtain second moment of simulated data.

This note is organized as follows. First, I illustrate a standard RBC model that will be solved in the code. Namely, I start with an example, since it is always easier (at least for me) to consider an example than general/abstract argument. Section 3 is the instruction for each part of a code. Section 4 provides some remarks.

2 The standard RBC model

Consider a standard RBC economy with a representative household whose preference is specified as follows,

$$\begin{aligned} \max_{c_t, l_t} \quad &: \quad E_0 \sum_{t=0}^{\infty t} \beta^t \left(\frac{c_t^{1-\sigma}}{1-\sigma} - l_t^{\lambda} \right) \\ s, t, \qquad a_{t+1} \quad &= \quad (1 + r_t) a_t - c_t + w_t l_t. \end{aligned}$$

where c_t and l_t denote consumption and labor supply. a_t, r_t and w_t stand for non-human asset, interest rate and wage rate for households. Assume that the representative firm has a Cob-Douglas technology in labor and capital. The recursive competitive equilibrium of the economy is defined as a set of c_t, l_t, n_t , k_{t+1}, w_t and r_t which satisfy the following efficiency conditions,

$$\begin{aligned} E_t\beta\left(1+r_{t+1}\right)\times c_{t+1}^{-\sigma} &= c_t^{-\sigma} && (\texttt{euler}) \\ \lambda l_t^{\lambda-1} - w_t c_t^{-\sigma} &= 0. && (\texttt{labor}) \end{aligned} \tag{1}$$

$$(1-\alpha)v_t\left(\frac{k_t}{n_t}\right)^{\alpha} = w_t \tag{2}$$

$$\alpha v_t\left(\frac{k_t}{n_t}\right)^{\alpha-1} - \delta = r_t \tag{3}$$

and resource constraints,

$$\begin{aligned} k_{t+1} - (1-\delta)\,k_t &= i_t \\ &= y_t - c_t && (\texttt{capital}) \\ l_t &= n_t \end{aligned}$$

where $y_t = v_t k_t^{\alpha} n_t^{1-\alpha}$. We can add one more exogenous stochastic process for productivity shock, v_t such as AR(1) as follows,

$$v_{t+1} = \phi v_t + (1-\phi)\,v^* + \varepsilon_t$$

where v^* denotes a normalized steady state level of productivity.

3 The Matlab code

3.1 Steady state proc

Let's start with the steady state part. The purpose of this part is to compute steady state values of endogenous variables. You have to solve a non-lnear multiple equation system numerically. Here is the steady state conditions for our example.

$$\beta(1+r^*) = 1$$

$$\lambda l^{*\lambda-1} - w^* c^{*-\sigma} = 0 \tag{4}$$

$$(1-\alpha)\,v^*\left(\frac{k^*}{l^*}\right) = w^* \tag{5}$$

$$\begin{aligned} \alpha v^*\left(\frac{k^*}{n^*}\right)^{\alpha-1} - \delta &= r^* \\ c^* + \delta k^* &= c^* + i^* \\ &= y^* \\ &= v^* k^{*\alpha} l^{*1-\alpha} \end{aligned}$$

You have to think for yourself in this part, if you change the model in this example. There can be many different orders for computation. Of course, you will obtain right answers, as long as you write down the steady state conditions correctly. But here in a Matlab code, I encourage you to pick up one that requires less computational burden. Note that Matlab works more quickly with step-by-step substitution, rather than solving simultaneous equations. So basically, reduce simultaneous equations. Here is my suggestion. I begin with r_*.

$$\begin{aligned} r^* &= \beta^{-1} - 1 \\ \frac{k^*}{l^*} &= \left(\frac{r^* + \delta}{\alpha}\right)^{\frac{1}{\alpha - 1}} \end{aligned}$$

denoted as

```
kls = (((1/beta)-1+delta)/alpha)^(1/(alpha-1));
```

I can ignore v^*, since it is normalized at one. Note that so far I only use step-by-step substitutions to obtain, r^*, w^* and k^*/l^* (=`kls`). For the rest of the part, a little device will save lots of computational burden. Divide resource constraint by l^*. Then we have c^*/l^* (=`cls`) as follows,

$$\begin{aligned} \frac{c^*}{l^*} &= \frac{y^*}{l^*} - \frac{i^*}{l^*} \\ &= \left(\frac{k^*}{l^*}\right)^{\alpha} - \delta \frac{k^*}{l^*}. \end{aligned}$$

This is denoted as,

```
cls = kls^alpha-delta*kls;
```

Then redefine $c^* = (c^*/l^*) \times l^*$ and eliminating w^* from eqn 4 and leaves,

$$\lambda l^{*\lambda - 1} \times c^{*\sigma} - (1 - \alpha) \left(\frac{k^*}{l^*}\right)^{\alpha} = 0.$$

Note that this equation contains only one unknown variable, l^*, since k^*/l^* (=`kls`) is already known and c^* is a function of l^*. To solve this non-linear equation numerically, we can use `solve` command in Matlab. You will find the following lines in the code,

```
ss1 = (lamda*ls^(lamda-1))*(cs^sigma) - (1-alpha)*(kls^alpha);
lss = solve(ss1,ls);
ls = double(lss);
 kstar  = kls*lstar;
 cstar  = clstar*lstar;
```

Now you find all the steady state values. There can be many other ways to compute them. This part depends on your way of thinking how to minimize the computational burden.

3.2 Model proc

First, make the list of all the variables that you want to compute by using `syms` command. In this simple RBC case, we have c_t, k_t, l_t, r_t, v_t, c_{t+1}, k_{t+1},l_{t+1},,r_{t+1} and v_{t+1} which can be written as,

```
syms ct kt lt rt vt ca ka la ra va ;
```

Remember that `a` stands for one-period ahead, such that `ca` denotes c_{t+1}, similarly, `ka` denotes k_{t+1}. This is just my notation. Then, write down all the efficiency conditions and resource constraints. There is only one rule here. Express all the equations with zero in the left-hand side. Further, you can give each equation a name and write those names in the LHS instead of zero.

```
wt  = vt*(1-alpha)*(kt/lt)^alpha;
ra  = va*alpha*(ka/la)^(1-alpha);

labor    = lamda*lt^(lamda-1)-wt*ct^(-sigma);
euler     = beta*(1+ra)*ca^(-sigma)-ct^(-sigma);
capital = ka - (1-delta)*kt-vt*(kt^alpha)*(lt^(1-alpha))+ct ;
tech       = va - phi*vt;
```

Before writing those conditions, I eliminate w_t and r_{t+1} using firm's optimal conditions. You do not necessarily have to do this. Nonetheless, this procedure will reduce computational burden. Note that the two efficiency conditions, `euler` and `labor` are expressed only by quantity variables, c_t, c_{t+1}, l_t, l_{t+1}, k_t and k_{t+1}, since the first and second lines substitute w_t and r_{t+1} out of them.

`optcon` compiles those efficiency conditions vertically in one matrix. This makes it possible to use `jacobian` command in next step.

3.3 Linearization proc

xx denotes endogenous variables vector to which you want to take derivative. When you pool them into the xx vector, let the $t+1$ variables come first, such that,

```
xx = [ca la ka va ct lt kt vt] ;
```

The next line `jopt` is the analytical expression of first order derivatives of all the equation pooled in `optcon`. `jacobian` command is extremely useful, since it gives you the Jacobian matrix in analytical form. Now, evaluating the Jacobian matrix at the steady state values will be the completion of linearization. This can be done by substituting the steady state values that we have already computed. You will see,

```
coef = eval(jopt);
```

which converts the Jacobian matrix in analytical form `jopt`, into numerical form `coef`. In the final part of the linearization step, you need to separate the `coef` matrix in two. The first four columns correspond to period $t+1$ variables and the rest of four columns to period t variables.

```
B = [ -coef(:,1:4) ].*TW ;
C = [ coef(:,5:8)   ].*TW ;
```

where `TW` denotes steady state values again, so that the coefficient matrix could be measured as percentage deviation from the steady state values. Here, we have the following linearized system.

$$\mathbf{C}\mathbf{y}_{\mathbf{t+1}} = \mathbf{B}\mathbf{y}_{\mathbf{t}}$$

where $\mathbf{y_t} = (c_t\, l_t\; k_t\, v_t)'$ and, $\mathbf{B}$ and $\mathbf{C}$ are 4×4 matrix with numerical elements. Finally, define $\mathbf{A}$ matrix as follows,

$$\begin{aligned} \mathbf{y_t} &= \mathbf{C^{-1}B}\mathbf{y_{t+1}} \\ &\equiv \mathbf{A}\mathbf{y_{t+1}}. \end{aligned} \tag{6}$$

3.4 Solution proc

In the Solution proc, our purpose is to decompose $\mathbf{A}$ matrix and eliminate all the unstable roots ($\theta_i < 1$)[1]. You can ignore the instruction of this proc, since you do not have to change this proc for any dynamic general equilibrium models.

First, eqn (6) can be expanded and eigen-decomposed as follows.

$$\begin{aligned} \begin{bmatrix} c_t \\ l_t \\ k_t \\ v_t \end{bmatrix} &= \begin{bmatrix} a_{11} & \cdots & a_{14} \\ \vdots & & \\ & & \\ a_{41} & & a_{44} \end{bmatrix} \begin{bmatrix} c_{t+1} \\ l_{t+1} \\ k_{t+1} \\ v_{t+1} \end{bmatrix} \\ &= \mathbf{Q^{-1}VQ}\mathbf{y_t} \\ \begin{bmatrix} \mathbf{Q_U} \\ \mathbf{Q_S} \end{bmatrix} \mathbf{y_t} &= \begin{bmatrix} \theta_1 & \cdots & 0 \\ \vdots & \theta_2 & \\ & & \theta_3 \\ 0 & & \theta_4 \end{bmatrix} \begin{bmatrix} \mathbf{Q_U} \\ \mathbf{Q_S} \end{bmatrix} \mathbf{y_{t+1}} \end{aligned} \tag{7}$$

[1] Do not get confused with the notation. Matlab computes the eigen matrix as the inverse of what we have in our mind. So make sure that $1/\theta_i$ is a root of the characteristic equation.

The following part of the code corresponds to the above calculation. Namely, (1) decomposition of A matrix, (2) Extracting stable roots ($\theta_i \geq 1$) and definition of the matrices $\mathbf{Q_U}$ (=UQ) and $\mathbf{Q_S}$ (=SQ).

```
[W V] = eig(A);
Q = inv(W);
W*V*Q;
theta = diag(V)
% Extract stable vectors
SQ = []; jw = 1;
for j = 1:length(theta)
    if abs(theta(j)) >1.000000001
    SQ(jw,:)  = Q(j,:);
    jw = jw+1;
    end
end
% Extract unstable vectors
UQ = []; jjw = 1;
for jj = 1:length(theta)
    if abs(theta(jj))<1.000001
    UQ(jjw,:)  = Q(jj,:);
    jjw = jjw+1;
    end
end
% Extract stable roots (lamda>1)
VLL = []; jjjw = 1;
for jjj = 1:length(theta)
    if abs(theta(jjj)) >1.000000001
VLL(jjjw,:)  = theta(jjj,:);
jjjw = jjjw+1;
    end
end

% Show Eigen Vectors on U-S Roots
UQ; % n x n+k
SQ; % k x n+k
```

By applying the Blanchard-Kahn theorem, we know that two out of four eigen values are greater than one. Let $\theta_3 > 1$ and $\theta_4 > 1$. Remember that they are *stable* roots. The number of rows of $\mathbf{Q_U}$ (=UQ) coincides with the number of unstable roots ($= n$) and similarly, number of rows of $\mathbf{Q_S}$ is equal to that of stable roots ($= k$).

```
k = min(size(SQ)); % # of predetermined vars
n = min(size(UQ)); % # of jump vars
```

And the following part,

```
PA = UQ(1:n,1:n); PB = UQ(1:n,n+1:n+k);
PC = SQ(1:k,1:n); PD = SQ(1:k,n+1:n+k);
```

implies,

$$\mathbf{Q}=\begin{bmatrix} \mathbf{Q_U} \\ \mathbf{Q_S} \end{bmatrix} = \begin{bmatrix} \mathbf{P_A} & \mathbf{P_B} \\ \mathbf{P_C} & \mathbf{P_D} \end{bmatrix}.$$

Then, the following relation is true to satisfy transversality condition.

$$\begin{aligned} \mathbf{Q_U y_t} &\equiv \begin{bmatrix} \mathbf{P_A} & \mathbf{P_B} \end{bmatrix} \mathbf{y_t} \\ &= 0 \end{aligned}$$

where $\mathbf{Q_U}$ is a 2×4 matrix. This can be rewritten as

$$\begin{aligned} \begin{bmatrix} c_t \\ l_t \end{bmatrix} &= \mathbf{P_A^{-1} P_B} \begin{bmatrix} k_t \\ v_t \end{bmatrix} \\ &\equiv \mathbf{P} \begin{bmatrix} k_t \\ v_t \end{bmatrix}. \end{aligned}$$

This is the rational expectation solution of the system. The corresponding part in the code is,

```
P = -inv(PA)*PB ;
```

Expanding eqn (7) and substituting the solution yields,

$$\begin{aligned} \mathbf{P_C} \begin{bmatrix} c_t \\ l_t \end{bmatrix} + \mathbf{P_D} \begin{bmatrix} k_t \\ v_t \end{bmatrix} &= \begin{bmatrix} \theta_3 & 0 \\ 0 & \theta_4 \end{bmatrix} \left\{ \mathbf{P_C} \begin{bmatrix} c_{t+1} \\ l_{t+1} \end{bmatrix} + \mathbf{P_D} \begin{bmatrix} k_{t+1} \\ v_{t+1} \end{bmatrix} \right\} \\ &= \mathbf{V_L^{-1}} \left(\mathbf{P_C P} + \mathbf{P_D} \right) \begin{bmatrix} k_{t+1} \\ v_{t+1} \end{bmatrix} \\ &\equiv \mathbf{V_L^{-1} P_E} \begin{bmatrix} k_{t+1} \\ v_{t+1} \end{bmatrix} \end{aligned}$$

Then finally, we have,

$$\begin{aligned} \begin{bmatrix} k_{t+1} \\ v_{t+1} \end{bmatrix} &= \mathbf{P_E^{-1} V_L P_E} \begin{bmatrix} k_t \\ v_t \end{bmatrix} \\ &\equiv \mathbf{A_A} \begin{bmatrix} k_t \\ v_t \end{bmatrix}. \end{aligned}$$

The ultimate purpose of this section is to derive $\mathbf{A_A}$ matrix shown above.

3.5 Simulation proc

Once you obtained the $\mathbf{A_A}$ matrix, impulse response function can be computed by step-by-step substitution as shown below,

```
Ss = S1;
S = zeros(t,k) ;
for i = 1:t
q = AA*Ss ;
    S(i,:)  = q';
Ss = S(i,:)';
    end
SY = [S1' ;S] ;
```

where `S1` and `SY` denote arbitrary initial value and the simulated path of state variable ($= (k_{t+i}\, v_{t+i})'$). Multiplying `P` matrix will give you the solution path of jump/control variables ($= (c_{t+i}\, l_{t+i})' =$`X`), namely,

```
X = (real(P)*SY')';
```

(Instruction for stochastic simulation is a work in progress.)

4 Remark

If you have any question, you can email me to `kato.13@osu.edu` or `ryou.katou@boj.or.jp`. Enjoy!