Linear Rational Expectations Models: Construction, Solution and Applications

Michael K. Johnston and Robert G. King

October 29, 2004
CONTENTS

4.3 Steps in specifying the model ........................................ 28
  4.3.1 Step 1: Variable List Specification .............................. 28
  4.3.2 Step 2A: Steady State Calculation ................................. 29
  4.3.3 Step 2B: Implications for model coefficients ................. 32
  4.3.4 Step 2C: Coefficient matrix specification ..................... 33
  4.3.5 Listing of lmrbc.m ................................................. 35
  4.3.6 Step 3: Driving Process Specification ......................... 37
  4.3.7 Step 4: Compute solution using RESOLKW .................... 38

5 Using the RE Solution ................................................. 41
  5.1 Impulse Responses .................................................. 41
  5.2 Simulations .......................................................... 45
    5.2.1 Artificial Shock Inputs ........................................... 45
    5.2.2 VAR Inputs ....................................................... 45
    5.2.3 Filtering Results ................................................ 45
  5.3 Population moments .................................................. 45
  5.4 Spectra .............................................................. 45
    5.4.1 ................................................................. 45
  5.5 Population moments of filtered series ......................... 45

6 Strategies for Larger Models ......................................... 47

7 Additional Examples ................................................... 49
  7.1 New IS-LM .......................................................... 49
  7.2 New Keynesian Taylor-Levin Staggering ......................... 49
  7.3 New Neo-Classical Synthesis (NNS) ................................. 49
  7.4 Open Economy NNS .................................................. 49
  7.5 State-Dependent Pricing ............................................ 49
  7.6 Optimal Monetary Policy ........................................... 49

Debugging and Error Messages ......................................... 51
  .1 Debugging .......................................................... 51
    .1.1 Checking for Rows of Zeros ..................................... 51
    .1.2 Checking $|Az - B| 
eq 0$ ........................................ 52
    .1.3 Checking Long-Run Multipliers ................................ 52
    .1.4 Checking the Roots ............................................. 53
CONTENTS

.1.5 Examining the Equations .................................. 54
.2 Error Messages .............................................. 55
  .2.1 Error messages from REDKW.m ................... 55
  .2.2 Error messages from DYNKW.m ................. 56
  .2.3 Errors from MDRKW.m ......................... 59

Mathematics 61
  .1 Jordan form (canonical form) ....................... 61

Bibliography 63
Introduction

Macroeconomic research makes substantial use of linear rational expectations models of the form

$$AE_{t}y_{t+1} = B y_{t} + \sum_{j=0}^{J} C_{j} E_{t}x_{t+j}$$

(1)

where $y$ is a vector of endogenous variables, $x$ is a vector of exogenous variables, and $A$, $B$, and $C(F)$ are matrices of coefficients. Some of the elements of $y_{t}$ are predetermined, in that they cannot respond to unexpected variations in $x_{t}$.

These models are usually solved under the assumption that there is a driving process for the exogenous variable which state variables as specified in $Q$:

$$x_{t} = Q \varsigma_{t}$$

(2)

$$\varsigma_{t} = \rho \varsigma_{t-1} + G \varepsilon_{t}$$

(3)

where $\rho$ is a matrix of coefficients for the autoregressive process and $G$ governs the response of the exogenous state variables to innovations $\varepsilon_{t}$.

Under conditions elaborated in the literature and below, there is a unique stable rational expectations solution that can be cast in the following state space form,
where $\Pi$ defines the relation between the rational expectations solution at a given point at time and the state variables of the same period. The vector of state variables $S_t$ contains both endogenous predetermined $k_t$ and exogenous $x_t$ variables. The second equation gives the dynamic behavior of the state variables as a function of their values in the previous period and current period innovations.

This document describes three aspects of such rational expectations models that are necessary for putting such linear rational expectations models to work:

Step 1: Formulation of the linear model (1)
Step 2: Formulation of the driving process (2,3)
Step 3: Use of the RE solution (4,5) in various applied contexts, including stochastic simulation; impulse response analysis; time and frequency domain moments and so forth.

The document also aids the reader in understanding two aspects of computation that are sometimes important for constructing and using rational expectations models. First, we provide a brief guide to the theory of such model solution as developed in Blanchard and Kahn [1980] and King and Watson [1998]. Second, we describes the solution of these models along lines described by King and Watson [2002], which is one method of moving from (1) and (2) to (3). Third, building on these two prior steps, we equip the reader to evaluate aspects of model construction that are useful in a range of activities, ranging from model debugging methods to analysis of unusual economic models.

To begin using these programs, you must unzip the file KWRE.zip to make use of the core programs and to explore introductory programs that are associated with the examples in this document. In the discussion that
follows, it is assumed that this directory is located on the "C" drive and is called "C:/KWRE/". To unzip the file, then you would place it in the "C" directory of your computer and then use Windows Explorer to extract the files, by going to "file" (alt-f) and then selecting "extract all".option, which will automatically create the following directories and subdirectories

Directory: **C:/KWRE**
Subdirectory containing main programs: **C:/KWRE/main**
Subdirectory containing utilities: **C:/KWRE/utilities**
Subdirectory containing first example: **C:/KWRE/muth2**
Subdirectory containing first study problem: **C:/KWRE/muth1**
Subdirectory containing first example: **C:/KWRE/rbc**
Chapter 1

A quick theoretical review

The linear rational expectations model (1) has been much studied in the literature, following the important work of Blanchard and Kahn [1980].¹ These authors studied the nonsingular case, where $A$ could be inverted to yield

$$E_t y_{t+1} = W y_t + \sum_{j=0}^{J} \Psi_j E_t x_{t+j}$$

(1.1)

with $W = A^{-1}B$ and $\Psi_j = A^{-1}C_j$.

Blanchard and Kahn [1980] established that there was a unique stable rational expectations solution to this model so long as two conditions were satisfied. First, there had to be as many stable eigenvalues of $W$ as there were predetermined variables, a result which is sometimes called the "Blanchard and Kahn counting rule" and has been widely used in applied research. Second, a particular matrix which linked the unstable "canonical variables" and nonpredetermined variables had to be of full rank. This "rank condition" can only be checked in the course of a detailed model solution and has therefore been less stressed.

King and Watson [1998] established that a unique stable rational expectations solution obtained for the general model (1) so long as the following conditions were fulfilled. First, there had to exist some scalar $z$ such that

$$|Az - B| \neq 0$$

as a necessary condition for any nontrivial solution to exist. Second, the Blanchard and Kahn counting rule, stated as equality between the number

¹For additional details in the current notation, see King and Watson [1998].
of stable roots and predetermined variables, must be satisfied for a unique stable rational expectations model. In this rule, a root is a solution to

\[ |A z - B| = 0 \]

or what is sometimes called a generalized eigenvalue of the matrices \( A \) and \( B \). Third, a generalized rank condition had to be satisfied.

King and Watson [2002] established that any solvable model of the form (1), including models with multiple solutions, also possessed a reduced dimension dynamic system of the form

\[
\begin{align*}
    f_t &= H d_t + \sum_{j=0}^{J_f} \Psi_{fj} E_t x_{t+j} \\
    E_t d_{t+1} &= W d_t + \sum_{j=0}^{J_d} \Psi_{dj} E_t x_{t+j}
\end{align*}
\]

They also described an algorithm for computing this "system reduction" and an additional algorithm for solving the resulting lower dimension dynamic model (1.3) using variants of the Blanchard-Kahn analysis that are improvements from the standpoint of numerical analysis.
Chapter 2

A Basic Muthian Model

To begin the discussion of solving and using linear rational expectations models, it is convenient to study an example in the style of Muth [1961]. We consider this model first from an analytical standpoint and we then consider it as an application of the methods described in this handbook.

2.1 Analytical approach

The basic Muthian model is convenient in that it is already linear, does not require steady state calculation, and it is simple enough to be solved easily by hand. In this model, expectations are important in that inventory investment today is based upon expectations about price in the future.

\[
\text{Supply : } q_t = \sigma p_t + k_t \\
\text{Demand : } q_t = -\alpha p_t + x_t + k_{t+1} \\
\text{Investment : } k_{t+1} = \phi[\beta E_t p_{t+1} - p_t]
\]

If we write this as a linear difference system, we have:

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & -\beta \phi & 1
\end{bmatrix}
\begin{bmatrix}
q_{t+1} \\
p_{t+1} \\
k_{t+1}
\end{bmatrix}
= \begin{bmatrix}
-1 & \sigma & 1 \\
1 & \alpha & 0 \\
0 & -\phi & 0
\end{bmatrix}
\begin{bmatrix}
q_t \\
p_t \\
k_t
\end{bmatrix}
+ \begin{bmatrix}
0 \\
0 \\
-1
\end{bmatrix}
x_t
\]
4

CHAPTER 2 A BASIC MUTHIAN MODEL

From the presence of a row of zeros, it is clear that the matrix $A$ is singular and cannot be inverted. This leaves the model in a form which cannot be solved by the method described in Blanchard and Kahn [1980].

2.1.1 System Reduction

However, it is possible to reduce the system analytically by hand. Equating supply and demand, we determine a restriction on price, $\sigma p_t + k_t = -\alpha p_t + x_t + k_{t+1}$. (We also know that we can always determine quantity from the supply curve once we have solve for $p_t$ and $k_t$.)

Eliminating $q_t$, we thus have a simplified system with two equations:

Price: $\sigma p_t + k_t = -\alpha p_t + x_t + k_{t+1}$

Investment: $k_{t+1} = \beta \phi E_t p_{t+1} - \phi p_t$

We can write this system as a linear difference system,

\[
\begin{bmatrix}
    -\beta \phi & 1 \\
    0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    p_{t+1} \\
    k_{t+1} \\
\end{bmatrix}
= \begin{bmatrix}
    -\phi & 0 \\
    \alpha + \sigma & 1 \\
\end{bmatrix}
\begin{bmatrix}
    p_t \\
    k_t \\
\end{bmatrix}
+ \begin{bmatrix}
    0 \\
    -1 \\
\end{bmatrix} x_t
\]

which has an nonsingular matrix $A$ so long neither of $\beta$ and $\phi$ is equal to zero. Since $A$ is nonsingular, we can solve for the reduced system by Inverting $A$ and premultiplying,

\[
\begin{bmatrix}
    p_{t+1} \\
    k_{t+1} \\
\end{bmatrix}
= \begin{bmatrix}
    \frac{1}{\beta \phi} & \frac{1}{\beta \phi} \\
    0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    -\phi & 0 \\
    \alpha + \sigma & 1 \\
\end{bmatrix}
\begin{bmatrix}
    p_t \\
    k_t \\
\end{bmatrix}
+ \begin{bmatrix}
    \frac{1}{\beta \phi} & \frac{1}{\beta \phi} \\
    0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    0 \\
    -1 \\
\end{bmatrix} x_t
\]

This small system thus provides an example of King and Watson’s [1998] suggestion that singular $A$ matrices “[…] arise from nonexpectational behavior relations or dynamic identities present in the singular linear difference system.” It also illustrates the idea of system reduction: equating supply and demand creates a smaller system in which $A$ is nonsingular.
2.1 ANALYTICAL APPROACH

2.1.2 Counting rule

As discussed above, Blanchard and Kahn [1980] specify a “counting rule” that is a necessary condition for a unique and stable solution. This rule is sometimes stated as "the number of unstable roots must be equal to the number of nonpredetermined variables" and sometimes as "the number of stable roots must be equal to the number of predetermined variables". Analytically, a root $z$ is considered stable iff $|z| < 1$ and is considered unstable iff $|z| > 1$. In the present system, there is one predetermined variable ($k_t$) and one nonpredetermined variable ($p_t$). To check the Blanchard and Kahn “counting rule” analytically, we examine the system eigenvalues. These may be written either as $|Iz - W| = 0$ or as $|Az - B| = 0$: the result is the same because the roots of the characteristic polynomials are the same (i.e., $|Az - B| = |AA^{-1}[Az - B]| = |A[Iz - W]| = |A||Iz - W| = 0$ which differ only by a constant term that does not affect the roots). It is sometimes convenient for us to calculate one expression or the other. Using either of these approaches, we determine that the roots are the solutions to

$$z^2 - \left( 1 + \frac{1}{\beta} + \frac{\alpha + \sigma}{\beta \phi} \right) z + \frac{1}{\beta} = 0$$

Hence, if we call the roots $\mu_1$ and $\mu_2$, then it follows that $(z - \mu_1)(z - \mu_2) = z^2 - \left( 1 + \frac{1}{\beta} + \frac{\alpha + \sigma}{\beta \phi} \right) z + \frac{1}{\beta} = 0$ implies that

$$\mu_1 \mu_2 = \frac{1}{\beta}$$

$$\mu_1 + \mu_2 = 1 + \frac{1}{\beta} + \frac{\alpha + \sigma}{\phi \beta}$$

Assume $\beta < 1$. For $\alpha + \sigma = 0$, the roots are 1 and $\frac{1}{\beta}$. For $\alpha + \sigma < 0$, both roots are greater than one and there is no stable solution. For $\alpha + \sigma > 0$, there are two unique roots, one greater than one and one less than one, and the Blanchard and Kahn "counting rule" is satisfied.¹

2.1.3 Solving the reduced system a la Blanchard-Kahn

The reduced system can be written in Jordan form (also called canonical form; see the appendix on Mathematics) in which a model transformation

¹Problem for interested reader: verify that the roots of the original three variable system are the same as those of the reduced system.
is based upon the eigenvalues and eigenvectors of the system. That is, let
\( P \equiv [P_1, ..., P_n] \) be an \( n \times n \) matrix of eigenvectors and let \( J \) be a matrix
containing the roots in Jordan form. Defining \( W \equiv PJP^{-1} \), and abstracting
from the particulars of our example,

\[
E_t y_{t+1} = Wy_t + \Psi_0 x_t + \Psi_1 E_t x_{t+1}
\]
\[
E_t P^{-1} y_{t+1} = P^{-1} PJP^{-1} y_t + P^{-1} [\Psi_0 x_t + \Psi_1 E_t x_{t+1}]
\]
\[
E_t y_{t+1}^* = J y_{t+1}^* + \Psi_0 x_t + \Psi_1 E_t x_{t+1}
\]

As the Jordan block matrix contains only diagonal entries ordered by
size, the system is effectively decoupled. That is, it can be can partition the
system into two sets of equations: one set governs stable dynamics and the
other governs unstable dynamics. More specifically, the partitioning implies
that there are "canonical variables" stable dynamics \((s_t)\) and one describing
variables with dynamics \((u_t)\):

\[
E_t u_{t+1} = J_u u_t + \Psi_{0u} x_t + \Psi_{1u} E_t x_{t+1}
\]
\[
E_t s_{t+1} = J_s s_t + C_{0s} x_t + C_{1s} E_t x_{t+1}
\]

where \( J_u \) and \( J_s \) are partitions from the Jordan matrix, one containing stable
roots and the other containing unstable roots.

The unstable equations of this model can be solved by unwinding forward
(Sargent’s procedure):

\[
J_u^{-1} E_t u_{t+1} = [\Psi_{0u} x_t + \Psi_{1u} E_t x_{t+1}]
\]

\[
= - \sum_{i=0}^{\infty} (J_u^{-1})^{i+1} E_t [\Psi_{0u} x_{t+i} + \Psi_{1u} E_t x_{t+i+1}]
\]

The nonpredetermined and predetermined variables in which we are truly
interested are related to the solution derived for the stable and unstable
partitions above by:

\[
\begin{bmatrix} V_{u\lambda} & V_{uk} \\ V_{sk} & V_{sk} \end{bmatrix} \begin{bmatrix} \lambda_t \\ k_t \end{bmatrix} = \begin{bmatrix} v_t \\ s_t \end{bmatrix}
\]
2.1 ANALYTICAL APPROACH

Hence, the solutions for the predetermined and nonpredetermined variables, respectively, are:

\[ k_{t+1} = w_{kk}k_t + w_{k\lambda}\lambda_t + \Psi_{0k}x_t + \Psi_{1k}E_t x_{t+1} \]
\[ \lambda_t = V_{u\lambda}^{-1} [u_t - V_{uk}k_t] \]

Note that this solution requires the Blanchard-Kahn "rank condition", which is that \( V_{u\lambda} \) is of full rank.

2.1.4 Imposing the driving process.

The discussion has proceeded to this point without imposing any assumptions about the driving process. However, let's now assume that it takes the form

\[ x_t = q\varsigma_t \]
\[ \varsigma_t = \rho\varsigma_{t-1} + g\varepsilon_t \]

which is a version of (2.3) but will lower case letters to remind us that all variables are scalars. Hence, given this system

\[ E_t x_{t+j} = q\rho^j \varsigma_t. \]

Using this in the solutions above, we can write

\[ u_t = -q \sum_{i=0}^{\infty} (J_u^{-1})^{i+1} E_t \left[ \Psi_{0u}\rho^i + \Psi_{1u}\rho^{i+1} \right] \varsigma_t = \theta_{uc}\varsigma_t \]

then

\[ \lambda_t = V_{u\lambda}^{-1} [u_t - V_{uk}k_t] = V_{u\lambda}^{-1} \theta_{uc}\varsigma_t - V_{u\lambda}^{-1}V_{uk}k_t = \pi_{\lambda\varsigma}\varsigma_t + \pi_{\lambda k}k_t \]
\[ k_{t+1} = w_{kk}k_t + w_{k\lambda}\lambda_t + \Psi_{0k}\varsigma_t + \Psi_{1k}q\varsigma_t \]
\[ = w_{kk}k_t + w_{k\lambda}(\pi_{\lambda\varsigma}\varsigma_t + \pi_{\lambda k}k_t) + \Psi_{0k}\varsigma_t + \Psi_{1k}q\varsigma_t \]
\[ = \pi_{\kappa\varsigma}\varsigma_t + \pi_{kk}k_t \]

which completes the derivations necessary to place the system in state space form.
2.2 Numerical analysis

The solution of the singular model considered above can be obtained using the method of King and Watson [2002]. In what follows, code shown is taken from files in the c:\KWRE\muth2 directory. To run the programs, you need to make this your active directory. You also need to run the SETPATH.m command at the beginning of the session, which modifies that path structure so that MATLAB knows where to look for the solution code and utilities. Hence, if you have chosen a location other than C:KWRE as a location to unzip the files, setpath.m must be edited. In the discussion below, we indicate files that are from the standard KWRE code with upper case letters and highlight that they are programs by using the typewriter font. We indicate files that are user generated or user modified by the bold bold typewriter font. These identification rules were just followed with the programs that control the path. It is assumed that the reader is sufficiently familiar with MATLAB to follow the discussion without comments. The companion document xxxx.pdf provides an overview of key MATLAB commands.

```matlab
1  % setpath.m
2  
3  path(path,'C:\KWRE\utilities')
4  path(path,'C:\KWRE\main')
```

2.2.1 Step 1: Variable List Specification

One starts by specifying a list of variables for the model, as in vlistmuth2.m. The variables $p$, $q$, and $k$ can be given to MATLAB as follows. The elements that the user needs to supply are highlighted in bold.

```matlab
% ********************************* %
% List of Names of Elements of Y(t) %
% ********************************* %

ylist=['q  ',
       'p  ',
       'p  ',
       'k   '];

% ********************************* %
% List of Names of Elements of x(t) %
% ********************************* %
xlist=['x '];
% ********************************* %
% Make ids for these variables
% ********************************* %
MAKEID(ylist,xlist,'imuth2')

The last line calls the MAKEID utility to create IDs for the variables
and to store them in the file imuth2.mat.

### 2.2.2 Step 2: Coefficient Matrix Specification

The equations for the singular system are

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & -\beta & 1
\end{bmatrix}
E_t
\begin{bmatrix}
q_{t+1} \\
p_{t+1} \\
k_{t+1}
\end{bmatrix}
=\begin{bmatrix}
1 -\sigma & -1 \\
1 & \alpha & 0 \\
0 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
q_t \\
p_t \\
k_t
\end{bmatrix}
+ \begin{bmatrix}
0 \\
-1 \\
0
\end{bmatrix}
x_t
\]

The program `sysmuth2.m` creates the coefficient matrices: a complete listing of this program is given later, but we begin by highlighting the user inputs that are necessary. The system file `sysmuth2.m` also executes the creation of the variable lists in `sysmuth2.m` and loads the resulting `imuth2.mat`.

**User Input 1: Specifying variable lists and ids**

% ********************************* %
% LISTS OF VARIABLES
% AND ASSOCIATED INDICES
% **************************************
vlstmuth2
load imuth2

User Input 2: Specifying lead length and predetermined variables

It is necessary to specify the number of leads of the exogenous variables \( (x_t) \) and the location of the predetermined variables. Rather than specifying the location of the predetermined variables explicitly, we make use of the IDs created by the MAKEID utility. To refer to capital \( (k_t) \), the only predetermined variable in the system, we simply enter \textit{ik}:

\[
\text{nlead}=1;
\]
\[
\text{lpd}=[\text{ik}];
\]

User Input 3: Specifying model parameters

Every model has parameters. The values of the various Muth model parameters are specified as:

\[
% \text{ Model parameters}
\]
\[
\text{sigma}=.1;
\]
\[
\text{alpha}=.2;
\]
\[
\text{phi}=4;
\]
\[
\text{b}=.99;
\]

Note that we deviate from the notation of the model by renaming the parameter \( \beta \) as \textit{b} in the text. We do this because "\texttt{beta.m}" is a built-in MATLAB function. Throughout our discussion, we identify such functions without capitals or bolding.
User Input 4: Specifying equations

System equations are specified as follows. The first command for each equation passes the existing equations list, the equation number, and a name for the equation to be created to the ADDEQN utility; which returns the new list of equations and the updated equation number. The remaining code is straightforward. Note that equation IDs are used again in place of explicitly stating the location of a variable.

```matlab
% EQN1
[elist,eqn]=ADDEQN('supply',elist,eqn);
B(eqn,iq)=-1;
B(eqn,ip)=sigma;
B(eqn,ik)=1;
% EQN2
[elist,eqn]=ADDEQN('demand',elist,eqn);
B(eqn,iq)=1;
B(eqn,ip)=alpha;
A(eqn,ik)=1;
CF0(eqn,ix)=-1;
% EQN3
[elist,eqn]=ADDEQN('inventory',elist,eqn);
A(eqn,ik)=1;
A(eqn,ip)=-b*theta;
B(eqn,ip)=-theta;
```

2.2.3 Step 3: Driving Process Specification

A driving process for the exogenous state variables must be specified in the driver file crashdrv.m. The driver file simply contains the coefficient matrices for the forcing process equations $x_t = Q\delta_t$ and $\delta_t = \rho\delta_{t-1} + \xi_t$:

```matlab
% drvmuth2.m
rho=.9;
x=.1;
ndelta=1;
```
nshock=1;
Q=1;
RHO=rho;
Gbar=1;

2.2.4 Step 4: Start RESOLKW

The last step is to pull together the elements of the previous steps and to pass them to the King and Watson code for a solution.\textsuperscript{2}

To solve the system, deriving the coefficient matrices for the state space solution, call RESOLKW.m:

\[
[M,PI,G,ylist,xlist,lpd,elist]=
\text{resolkw}('sysmuth2','drvmuth2',track,comp);
\]

Note that the last two input arguments – track and comp – are optional. \textbf{Track} sets the desired level of command window output regarding solution algorithm progress:

- \texttt{track} = 0 \hspace{1cm} \text{no tracking}
- \texttt{track} = 1 \hspace{1cm} \text{minimal tracking (includes reports on eigenvalues)}
- \texttt{track} = 10 \hspace{1cm} \text{extensive tracking}

\textbf{Comp} specifies the specific solution algorithm to be used:

- \texttt{comp} = [0 0] \hspace{1cm} \text{eigenvalue decomposition method}
- \texttt{comp} = [0 1] \hspace{1cm} \text{non-balanced eigenvalue decomposition method}
- \texttt{comp} = [1 0] \hspace{1cm} \text{Schur decomposition}

If you wish to resolve the system using the default level of tracking and the default solution algorithm, call RESOLKW.m without the last two arguments:

\textsuperscript{2}The beginning of the start or master file \texttt{mastermuth2.m} it is one place to specify the paths to the \texttt{kwre\main} directory and the \texttt{kwre\utilities} directory.
\[
[M,PI,G,ylist,xlist,lpd,elist]=
\text{resolkw('sysmuth2','drvmuth2')};
\]

MATLAB will produce the coefficient matrices for the correct state-space solution when the code is run. In our example, the output given for \texttt{track=1} is:

\begin{verbatim}
Reduced dimension dynamic system analysis (DYNKW.m):
reduced system dimension = 2
Root counting using a critical value of bcrit =0.99999
unstable eigenvalues = 1
nonpredetermined variables = 1
Information on eigenvalues of reduced system
abs(mu) bcrit*abs(mu)
1.3215 1.3215
0.7644 0.7644
\end{verbatim}

Notice that this information matches what we worked out by hand: there is a two variable reduced dimension system, which contains two eigenvalues (one stable and one unstable, in terms of modulus). Note also that the stability definition adopted by the program treats a root as unstable only if \( \text{bcrit} \times \text{abs} (\mu) > 1 \), where \( \text{bcrit} \) has a default value of 0.99999. This default value can be changed by editing RESOLKW.m

The state space coefficient matrices can be found in the workspace window. These are
CHAPTER 2 A BASIC MUTHIAN MODEL

\[
\begin{bmatrix}
q_t \\
p_t \\
k_t \\
x_t
\end{bmatrix} =
\begin{bmatrix}
\pi_{qk} & \pi_{qs} \\
\pi_{pk} & \pi_{ps}
\end{bmatrix}
\begin{bmatrix}
k_t \\
\varsigma_t
\end{bmatrix}
\]

\[
\begin{bmatrix}
k_t \\
\varsigma_t
\end{bmatrix} =
\begin{bmatrix}
m_{kk} & m_{k\varsigma} \\
0 & \rho
\end{bmatrix}
\begin{bmatrix}
k_{t-1} \\
\varsigma_{t-1}
\end{bmatrix}
+ \begin{bmatrix} 0 \\ q \end{bmatrix} \varepsilon_t
\]

Note that the matrices listed below satisfy several general restrictions suggested by the equations above

\[
\Pi = \begin{bmatrix} 0.9215 & 0.2463 \\ -0.7855 & 2.4626 \\ 1.0000 & 0.0000 \\ 0.0000 & 1.0000 \end{bmatrix}
\]

\[
M = \begin{bmatrix} 0.7644 & -0.2612 \\ 0.0000 & 0.9000 \end{bmatrix}
\]
2.3 Listing of sysmuth2.m

A complete listing of sysmuth2.m is provided next. Note that each of the "user inputs" discussed above is offset in the program list below. The remainder of the code is standard across various applications and program comments explain the general nature of component.

```matlab
% sysmuth2.m

% ***************************************************************
% This is an example of a system file which creates a
% a dynamic model in
% %
% A*y(t+1/t) = B*y(t) + C(F)*x(t/t),
% %
% where C(F) = C0 + C1*F + C2*F^2 + ... + CN*F^N.
% In this expression, F
% the forward shift operator; i.e., F*x(k/t) = x(k+1/t).
% 
% The program produces the matrices
% A
% B
% C = [C0, C1,..., CN]
% 
% It also creates some key vectors
% lpd = Array that contains locations of predetermined
% variables in the y vector.
% 
% ylist = Array of names of the elements of y (strings)
% xlist = Array of names of the elements of x (strings)
% ny = number of y variables
% nx = number of x variables
% nlead: Number of leads of x(t+j/t) -- (i.e., n in the above
% equation).

% ***************************************************************

% LISTS OF VARIABLES
```
% AND ASSOCIATED INDICES
% ********************
% USER INPUT 1: Create lists and load ids
vlstmuth2
load imuth2

% USER INPUT 2: Specify Location of predetetermined variables
lpd=ik];

% USER INPUT 3: Specify Length of leads in C matrices
nlead = 0;

% ********************
% system dimension
% ********************
ny=ROWS(ylist);
nx=ROWS(xlist);

% ********************
% DEFINING THE MATRICES A, B AND C 
% ********************

% Initialize Matrices:
A=zeros(ny,ny);
B=zeros(ny,ny);
for i=0:nlead;
eval(['CF' num2str(i) ' =zeros(ny,nx);']);
end
2.3 LISTING OF SYSMUTH2.M

69 CF=zeros(ny,nx);

70 % initialize equation list and equation number
71 elist=[];
72 eqn=0;
73 % USER INPUT 4: Specify model parameters
74    sigma=.1;
75    alpha=.2;
76    phi=4;
77    b=.99;
78 % USER INPUT 5: specify model equations
79 % EQN1
80    [elist,eqn]=ADDEQN(’supply’,elist,eqn);
81    B(eqn,iq)=-1;
82    B(eqn,ip)=sigma;
83    B(eqn,ik)=1;
84 % EQN2
85    [elist,eqn]=ADDEQN(’demand’,elist,eqn);
86    B(eqn,iq)=1;
87    B(eqn,ip)=alpha;
88    A(eqn,ik)=1;
89    CF0(eqn,ix)=-1;
90 % EQN3
91    [elist,eqn]=ADDEQN(’inventory’,elist,eqn);
92    A(eqn,ik)=1;
A(eqn,ip) = -b*phi;
B(eqn,ip) = -phi;

% ********************
% COMBINE Ci matrices
% ********************
CF = [];
for i = 0:nlead;
    loc = eval(['CF' num2str(i)]);
    CF = [CF loc];
end
Chapter 3

Casting Models in Standard Form

Solving models using the methods described here requires that the model be specified in the first-order form (1). There exist many models which are not naturally in first-order form but which may be placed in first-order form with a few tricks.

3.1 Another Muth Example

Muth [1961] begins his study of rational expectations models with a simple example, which we now discuss. Production must be determined one period in advance, which introduces expectations into the model equations.

\[
\begin{align*}
\text{Supply} & : q_t = \sigma E_{t-1} p_t \\
\text{Demand} & : q_t = -\alpha p_t + x_t
\end{align*}
\]

By equating supply and demand, we find a difference equation very similar to the example that Blanchard and Kahn [1980] refer to as being “zeroth-order.”

\[
\alpha p_t = -\sigma E_{t-1} p_t + \phi x_t
\]
CHAPTER 3 CASTING MODELS IN STANDARD FORM

This model possesses a natural solution for values of $\alpha + \sigma \neq 0$:

$$
\alpha E_{t-1}p_t = -\sigma E_{t-1}p_t + \phi E_{t-1}x_t \\
E_{t-1}p_t = \frac{\phi}{\alpha + \sigma} E_{t-1}x_t \\
p_t = \frac{1}{\alpha} \left( \frac{-\sigma \phi}{\alpha + \sigma} E_{t-1}x_t + \phi x_t \right)
$$

While this model is not immediately in first-order form, there exists a convenient transformation which yields a singular linear rational expectations model easily solvable by the methods discussed here. Define $w_t \equiv E_{t-1}p_t$ and include this definition as an equation in a first-order difference system

$$
\begin{bmatrix}
0 & 0 \\
1 & -1
\end{bmatrix}
\begin{bmatrix}
w_{t+1} \\
p_{t+1}
\end{bmatrix}
= 
\begin{bmatrix}
\sigma & \alpha \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
w_t \\
p_t
\end{bmatrix}
+ 
\begin{bmatrix}
-1 \\
0
\end{bmatrix}
x_t
$$

in which the first equation is the original and uses date $t$ variables and the second equation is the new variable definition and uses date $t+1$ variables. Naturally, both of the coefficient matrices $A$ and $B$ are singular. While the requirement for solvability that there must be a $z$ s.t. $|Az - B| \neq 0$ is not always intuitive, it is in this case. There is a $z$ such that $|Az - B| = -z (\alpha + \sigma)$ is not equal to zero so long as $\alpha + \sigma \neq 0$, exactly the requirement imposed by the analytical solution. Note also that there is a zero a zero root for this specification, so that there is an important sense in which it really is a "zero-th order" difference equation.

**Study problem:** Follow the steps above to make this model operational, creating your own code or modifying the code in the "muth 2" directory. The answer to this problem is provided in the "muth 1" directory.

### 3.2 More tricks

#### 3.2.1 Lagged expectations

The procedure discussed above can be directly extended to a vector system. Suppose that the model is

$$
AE_{t}y_{t+1} = By_t + \Gamma E_{t-1}y_t + Cx_t
$$

```
### 3.2 MORE TRICKS

Then a system in standard form can be created as

\[
\begin{bmatrix}
I & 0 \\
I & -I
\end{bmatrix} E_t \begin{bmatrix}
y_{t+1} \\
E_t y_{t+1}
\end{bmatrix} = \begin{bmatrix}
B & \Gamma \\
0 & 0
\end{bmatrix} \begin{bmatrix}
y_t \\
E_{t-1} y_t
\end{bmatrix} + \begin{bmatrix}
C \\
0
\end{bmatrix} x_t
\]

#### 3.2.2 Longer leads

Suppose that we had an RE model of the form

\[
A_2 E_t y_{t+2} + A_1 E_t y_{t+1} = B_0 y_t + B_1 y_{t-1} + B_2 y_{t-2} + C x_t
\]

Then we could write it as

\[
\begin{bmatrix}
I & 0 & 0 & 0 \\
0 & I & 0 & 0 \\
0 & 0 & I & 0 \\
0 & 0 & A_1 & A_2
\end{bmatrix} E_t \begin{bmatrix}
y_{t-1} \\
y_t \\
y_{t+1} \\
y_{t+2}
\end{bmatrix} = \begin{bmatrix}
0 & I & 0 & 0 \\
0 & 0 & I & 0 \\
0 & 0 & 0 & I \\
B_2 & B_2 & B_0 & 0
\end{bmatrix} \begin{bmatrix}
y_{t-2} \\
y_{t-1} \\
y_t \\
y_{t+1}
\end{bmatrix} + \begin{bmatrix}
0 \\
0 \\
0 \\
C
\end{bmatrix} x_t
\]

#### 3.2.3 Recursive Equilibria

The "big trick" is simply to model in a recursive manner. For example, suppose that there is a farmer who decides at \( t \) to plant a certain acreage, \( q_{t+1} \), that will yield output at date \( t+1 \). The date \( t \) farmer’s optimality condition might take the linear form

\[
q_{t+1} = \sigma E_t p_{t+1}
\]
The price is determined at $t$ by

$$q_t = -\alpha p_t + x_t$$

Hence, this system would naturally be written as

$$\begin{bmatrix} 0 & 0 \\ 1 & -\sigma \end{bmatrix} E_t \begin{bmatrix} q_{t+1} \\ p_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \alpha \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q_t \\ p_t \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} x_t$$

We will see a further application of this approach in the next section.
Chapter 4

RBC Basics

The previous chapters considered models which are already linear and do not require steady-state calculation. We now consider a basic RBC model with endogenous labor supply with which we will depart from these simplicities. A variety of means exist for (1) calculating linear and loglinear approximations and (2) for finding the steady state. Code for this example can be found in the `\kwre\rbc` directory.

4.1 Maximization

Dynamic programming and recursive equilibrium analysis provide convenient means of producing a set of model equations which take the general form

$$E_t F(y_{t+1}, y_t, x_{t+1}, x_t) = 0$$

which, upon linearization, produce a system of the form (1) for a large class of Markovian driving processes.

For the basic RBC model studied in this chapter, the Bellman equation is:

$$v(k_t, s_t) = \max \{ u(c_t, l_t) + \beta E_t v(k_{t+1}, s_{t+1}) \mid (k_t, s_t) \}$$

where the functional forms are

$$u(c_t, l_t) = \ln c_t - \frac{\chi}{1 + \eta} (1 - l_t)^{1+\eta}$$

$$a_t f(k_t, n_t) = a_t k_t^{1-\alpha} n_t^\alpha$$
The maximization involves the selection of \([c_t, n_t, l_t, i_t, k_{t+1}]\) and the restrictions are

\[
\begin{align*}
1 & = n_t + l_t \\
c_t + i_t + g_t & = a f (k_t, n_t) \\
k_{t+1} & = (1 - \delta) k_t + i_t
\end{align*}
\]

In dynamic programming, we are interested in finding optimal policies which maximize the objective subject to the constraints, which are given above. If we attach a multiplier to each constraint, then we can form the Lagrangian,

\[
L(k_t, \varsigma_t, c_t, l_t, n_t, k_{t+1}, \lambda_t, \omega_t) = u(c_t, l_t) + \beta E v(k_{t+1}, \varsigma_{t+1}) | (k_t, \varsigma_t) + p_t [a_t f (k_t, n_t) - c_t - i_t - g_t] + \lambda_t [1 - \delta] k_t + i_t - k_{t+1} + \omega_t [1 - n_t - l_t]
\]

We minimize the Lagrangian with respect to the multipliers \((\lambda_t, \omega_t)\) and maximize with respect to the choices \((c_t, l_t, i_t, n_t, k_{t+1})\).

### 4.1.1 First order conditions

For all functions \(f\), define \(f_x \equiv \frac{\partial f}{\partial x}\). In this notation, implied first order conditions include:

\[
\begin{align*}
c_t & : 0 = u_c (c_t, l_t) - p_t \\
l_t & : 0 = u_l (c_t, l_t) - \omega_t \\
i_t & : 0 = -p_t + \lambda_t \\
n_t & : 0 = \lambda_t a_t f_n (k_t, n_t) - \omega_t \\
k_{t+1} & : 0 = -\lambda_t + \beta E v_k (k_{t+1}, \varsigma_{t+1}) | (k_t, \varsigma_t) + p_t [a_t f (k_t, n_t) - c_t - i_t - g_t] + \lambda_t (1 - \delta) k_t + i_t - k_{t+1} + \omega_t [1 - n_t - l_t] \\
p_t & : 0 = a_t f (k_t, n_t) - c_t - i_t - g_t \\
\lambda_t & : 0 = 1 - n_t - l_t + \omega_t [1 - n_t - l_t] \\
\omega_t & : 0 = -v_k (k_t, \varsigma_t) + p_t a_t f_k (k_t, n_t) + \lambda_t (1 - \delta)
\end{align*}
\]
4.1 MAXIMIZATION

4.1.2 Driving process

Notice that we have been able to pose the dynamic programming problem and to state the FOCS without making any assumptions about the driving process, other than that productivity and government purchases are functions of a Markov process ($\varsigma_t$).

In our example, we assume that the exogenous variables are governed by a simple loglinear specification,

\[
\begin{pmatrix}
\ln a_t \\
\ln g_t
\end{pmatrix}_{xt} =
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}_{Q}
\begin{pmatrix}
\ln a_t \\
\ln g_t
\end{pmatrix}_{\varsigma_t}
\]

\[
\begin{pmatrix}
\ln a_t \\
\ln g_t
\end{pmatrix}_{\varsigma_t} =
\begin{pmatrix}
\rho_a & 0 \\
0 & \rho_g
\end{pmatrix}_{\rho}
\begin{pmatrix}
\ln a_{t-1} \\
\ln g_{t-1}
\end{pmatrix}_{\varsigma_{t-1}} +
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}_{Q}\varepsilon_t
\]

which is of the general form (2,3) and also is Markovian.
4.2 Linear Approximation (Analytical)

Define $m(k_t, \varsigma_t) = v_k(k_t, \varsigma_t)$ for convenience. Total differentiation provides the appropriate linearly approximated equations:

\[
\begin{align*}
c_t : 0 &= u_{cc}(c, l) dc_t + u_{cd}(c, l) dl_t - dp_t \\
l_t : 0 &= u_{lt}(c, l) dl_t + u_{lc}(c, l) dc_t - d\omega_t \\
i_t : 0 &= -dp_t + d\lambda_t \\
n_t : 0 &= -d\omega_t + pa \left[ f_{nn}(k, n) dn_t + f_{nk}(k, n) dk_t \right] + af_n(k, n) dp_t + pf_n(k, n) da_t \\
k_{t+1} : 0 &= -d\lambda_t + \beta E_t [dm_{t+1}] \\
p_t : 0 &= a \left[ f_k(k, n) dk_t + f_n(k, n) dn_t \right] + f(k, n) da_t \\
-dc_t - di_t - dg_t \\
\omega_t : 0 &= -(1-\delta)dk_t + di_t - dk_{t+1} \\
Et : 0 &= -dm_t + pa \left[ f_{kk}(k, n) dk_t + f_{kn}(k, n) dn_t \right] + af_k(k, n) dp_t + pf_k(k, n) da_t + (1-\delta)d\lambda_t \\
\end{align*}
\]

We can write these equations in matrix form as (note that $dl_t = -dn_t$ and for simplicity write $f_x(\cdot) = f_x$). Note that these equations are written in an unusual manner, with an additional material to help keep track of which equation we are looking at and which variable we are looking at. For example, the first order condition for labor is the row in the "B" matrix that begins with $n$; the effect of higher capital on the marginal product of labor ($af_{kn}$) occurs at the intersection of the $n$ row and the $k$ column. This notation is
4.2 LINEAR APPROXIMATION (ANALYTICAL)

not used in coding, but just in presentation.

\[
\begin{bmatrix}
  c & l & i & n & p & \lambda & \omega & k & m \\
  c & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  k' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  \lambda & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
  \omega & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  ET & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  d\lambda_{t+1} \\
  d\omega_{t+1} \\
  d\kappa_{t+1} \\
  d\mu_{t+1} \\
  E_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  d\lambda_t \\
  d\omega_t \\
  d\kappa_t \\
  d\mu_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  d\lambda_{t+1} \\
  d\omega_{t+1} \\
  d\kappa_{t+1} \\
  d\mu_{t+1} \\
  E_t \\
\end{bmatrix}
\begin{bmatrix}
  af_{kn} \\
  a_{f_k} \\
  1 - \delta \\
  paf_{kk} - 1 \\
\end{bmatrix}
\]

\[\ldots a g\]

\[
\begin{bmatrix}
  dc_t + 1 \\
  dl_t + 1 \\
  di_t + 1 \\
  dm_t + 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  d\lambda_{t+1} \\
  d\omega_{t+1} \\
  d\kappa_{t+1} \\
  d\mu_{t+1} \\
  E_t \\
\end{bmatrix}
\begin{bmatrix}
  af_{kn} \\
  a_{f_k} \\
  1 - \delta \\
  paf_{kk} - 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  d\lambda_{t+1} \\
  d\omega_{t+1} \\
  d\kappa_{t+1} \\
  d\mu_{t+1} \\
  E_t \\
\end{bmatrix}
\begin{bmatrix}
  af_{kn} \\
  a_{f_k} \\
  1 - \delta \\
  paf_{kk} - 1 \\
\end{bmatrix}
\]

\[\ldots a g\]

\[
\begin{bmatrix}
  dc_t + 1 \\
  dl_t + 1 \\
  di_t + 1 \\
  dm_t + 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
  di_t \\
  dm_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  da_t \\
  dg_t \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_{t+1} \\
  dl_{t+1} \\
  di_{t+1} \\
  dm_{t+1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  dc_t \\
  dl_t \\
Note that these equations suggest that there is a reduced dimension dynamic system of no more than two variables, since all other rows of $A$ are zero.

### 4.3 Steps in specifying the model

We parallel the steps in model specification of the introductory Muth model of chapter 1, highlighting the new elements implied by the RBC model.

In broad form, the RBC model is a system of possibly equations,

$$E_t F(y_{t+1}, y_t, x_{t+1}, x_t, \theta) = 0$$

where the variables are defined as in (1) and $\theta$ is a vector of parameters.

A stationary point of this model is a $(y, x, \text{ and } \theta)$ such that

$$F(y, y, x, x, \theta) = 0$$

Suppose that there are $n$ equations in this setting. Then, we can choose to satisfy these equations by various choices of endogenous variables $y$, exogenous variables $x$, and parameters $\theta$.

### 4.3.1 Step 1: Variable List Specification

We begin by specifying the list of variables in vlstrbc.m. The variable list matches the vector of endogenous variables derived in the previous section (aside from a few notational changes for coding purposes). However, rather than code the lists in detail, we make use of the ADDEQN.m utility, which has a parallel structure to ADDEQN.m. Since the reader is familiar with the process from above, we simply provide a listing for the program vlstrbc.m

```matlab
1 % vlstrbc.m
2 3 % ******************************* %
4 % List of Names of Elements of Y(t) %
5 % ******************************* %
6
7 ylist=[];
8 count=0;
9 [ylist,count]=ADDVAR(’c’,ylist,count);
```
4.3 STEPS IN SPECIFYING THE MODEL

[\texttt{ylist,count]}=\texttt{ADDVAR('l',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('i',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('n',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('p',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('lam',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('om',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('k',ylist,count)};
\[ \texttt{ylist,count]}=\texttt{ADDVAR('m',ylist,count)};

\% ********************************* \%
\% List of Names of Elements of x(t) \%
\% ********************************* \%
\[ \texttt{xlist=}[;]
\[ \texttt{count=0;}
\[ \texttt{[xlist,count]}=\texttt{ADDVAR('a',xlist,count)};
\[ \texttt{[xlist,count]}=\texttt{ADDVAR('g',xlist,count)};
\[ \texttt{MAKEID(ylist,xlist,'idrbc')}\]

Note that the last line again calls the \texttt{MAKEID} utility to create IDs for the variables and to store them in \texttt{idrbc.mat}.

4.3.2 Step 2A: Steady State Calculation

In the RBC model, the specification of the linear system cannot be undertaken until the steady-state of the model is first solved. For this steady state, there are typically certain features of the data which we wish to match. Given these values, we can analytically calculate the steady state values for the remaining parameters. One program to perform such steady-state computations is \texttt{ssrbc.m}.

\%ssrbc.m

\% Calculate steady state of model
% Features of data to match

sg=.2; % g/y
sn=.67; % wn/y
sk=1-sn; % qk/y = 1 - wn/y
r=.01; % four percent real rate per year (1% per quarter)
delta=.025; % 10 percent depreciation rate per year (2.5% per quarter)
gamma=.5; % inverse of labor supply elasticity
sigma=1; % intertemporal
bstar=1/(1+r);

% choose whether to calibrate utility parameter chi
% to hit specific labor value or whether to fix utility
% parameter and determine particular labor value.

cal=1;

if cal==1
    nbar=.2;
else
    chi=5;
end

% Normalization

a=1;

% Determine other elements of steady-state from
% Calculate capital-labor ratio
% from MPK=r+delta

knratio=((a*sk)/(r+delta))^(1/(1-sk));

% Calculate the real wage rate

wss=a*sn*(knratio^sk);
% Calculate the ratio of output to labor input
ynratio=a*(knratio^sk);

% Calculate the ratio of consumption to labor input
cnratio=(1-sg)*ynratio-delta*knratio;

% The requirement that
% w-(mun)/muc=chi*(n^gamma)*(c^sigma)
% now is "one equation in one unknown" given the
% information above since the rhs is
% chi*(n^(gamma+sigma))*(cnratio^sigma)
% We can use it to "calibrate" the labor preference
% parameter chi given a specified value of nbar or
% to "calculate" the ss level of labor given
% a particular chi value.

if (cal==1)
    denom=(nbar^(gamma+sigma))*(cnratio^sigma);
    chi=(wss/denom);
    nss=nbar;
else
    nss=(wss/(chi*(cnratio^sigma)));
    ns=ns^(1/(gamma+sigma));
end

css=cnratio*nss;
yss=a*(knratio^sk)*nss;
kss=knratio*nss;
lamss=css^(-sigma);
pss=lamss;
lss=1-nss;
omss=wss*pss;

% Checking 4 nonlinear equations at these parameter values
gap=zeros(4,1);
CHAPTER 4 RBC BASICS

81 gap(1) = (css^(-sigma)) - lamss;
82 gap(2) = chi * nss * gamma - lamss * wss;
83 gap(3) = bstar * lamss * (a * sk * ((nss / kss) ^ sn) + (1 - delta)) - lamss;
84 yss2 = a * (kss ^ sk) * (nss ^ sn);
85 gap(4) = (1 - sg) * yss2 + (1 - delta) * kss - css;
86
87 disp('Equation gaps for calibrated model')
88 disp(gap')
89
90 x = [lamss kss css chi];
91
92 disp('solutions for [lamss kss css chi]');
93 disp(x)
94
95 % Calculate derivative information
derivinfo
96
97 % Save ss values
98 save ssrbc.mat css nss lss yss kss pss lamss omss

4.3.3 Step 2B: Implications for model coefficients

Many of the model coefficients involve derivatives evaluated at the steady state. Given the steady state parameters above, these derivative information is computed in derivinfo.m, which is called by ssrbc.m.

1 % derivinfo.m
2
3 % Compute some derivative information that will be used in linear system
4 Mss = 1 / bstar;
5
6 DccU = -sigma * css ^ (-sigma - i);
4.3 STEPS IN SPECIFYING THE MODEL

DclU=0;
DncU=0;
DllU=-chi*gamma*(nss^(gamma-1));

F=(nss^sn)*(kss^sk);
DnF=sn*(nss^(sn-1))*(kss^sk);
DkF=sk*(nss^sn)*(kss^(sk-1));
DnnF=sn*(sn-1)*(nss^(sn-2))*(kss^sk);
DnkF=sn*sk*(nss^(sn-1))*(kss^(sk-1));
DknF=DnkF;
DkkF=sk*(sk-1)*(nss^sn)*(kss^(sk-2));

4.3.4 Step 2C: Coefficient matrix specification

The coefficient matrices are specified by linrbc.m, but for clarity of presentation this is accomplished in another program eqnsrbc.m, which is called by linrbc.m

% eqnsrbc.m

% EQN

[elist,eqn]=addeqn('FOC:c',elist,eqn);
B(eqn,ic)=DccU;
B(eqn,il)=DclU;
B(eqn,ip)=-1;

% EQN

[elist,eqn]=addeqn('FOC:l',elist,eqn);
B(eqn,ic)=DclU;
B(eqn,il)=DllU;
B(eqn,iom)=-1;

% EQN
[elist, eqn] = addeqn('FOC:i', elist, eqn);
B(eqn, ip) = -1;
B(eqn, ilam) = 1;

% EQN

[elist, eqn] = addeqn('FOC:n', elist, eqn);
B(eqn, in) = pss*a*DnnF;
B(eqn, ik) = pss*a*DnkF;
B(eqn, iom) = -1;
B(eqn, ip) = a*DnF;
CF0(eqn, ia) = pss*DnF;

% EQN

[elist, eqn] = addeqn('FOC:kprime', elist, eqn);
B(eqn, ilam) = 1;
A(eqn, im) = bstar;

% EQN

[elist, eqn] = addeqn('FOC: p (commodity resource)', elist, eqn);
B(eqn, ik) = a*DkF;
B(eqn, in) = a*DnF;
B(eqn, ic) = -1;
B(eqn, ii) = -1;
CF0(eqn, ia) = F;
CF0(eqn, ig) = -1;

% EQN

[elist, eqn] = addeqn('FOC: lam (k accumulation)', elist, eqn);
4.3 STEPS IN SPECIFYING THE MODEL

4.3.5 Listing of linrbc.m

The linrbc.m program is a linear system program (similar to sysmuth2.m). As with that program, we highlight the user-specified elements by indenting these.

```
% linrbc.m

% LISTS OF VARIABLES
% AND ASSOCIATED INDICES
```
% ******************************

% USER INPUT 1: Create lists and load ids
vlstrbc
load idrbc

% USER INPUT 2: Specify Location of predetermined variables
lpd=[ik];

% USER INPUT 3: Specify Length of leads in C matrices
nlead = 0;

% ******************************
% system dimension
% ******************************

ny=ROWS(ylist);
xn=ROWS(xlist);

% ******************************
% DEFINING THE MATRICES A, B AND C
% ******************************

% Initialize Matrices:
A=zeros(ny,ny);
B=zeros(ny,ny);
for i=0:nlead;
eval(['CF' num2str(i) '=zeros(ny,nx);']);
end
CF=zeros(ny,nx);
4.3 STEPS IN SPECIFYING THE MODEL

% initialize equation list and equation number
elist=[];
eqn=0;

% USER INPUT 4A: Solve steady state and specify model parameters
ssrbc

% USER INPUT 4B: Specify Equations
eqnsrbc

% ********************
% COMBINE Ci matrices
% ********************

CF=[];
for i=0:nlead;
    loc=eval(['CF' num2str(i)]);
    CF=[CF loc];
end

4.3.6 Step 3: Driving Process Specification

The driving process is specified in drvrbc.m and contains the coefficient matrices for the forcing process equations $x_t = Q\xi_t$ and $\xi_t = \rho\xi_{t-1} + G_{\xi_t}$:

% drvrbc.m

rhoa=.9;
rhog=.75;

nx=2;
ndelta=2;
nshock=2;
38 CHAPTER 4 RBC BASICS

Q=eye(2);
RHO=eye(2);
Gbar=eye(2);

RHO(1,1)=rhoa;
RHO(2,2)=rhog;

4.3.7 Step 4: Compute solution using RESOLKW

The program masterbbc.m sets the MATLAB path; uses RESOLKW to compute a solution; produces an impulse response; and plots the results.

% master.m

% Set path for KWRE package
setpath

% calculate RE solution
[M,PI,G,ylist,xlist,lpd,elist]=resolkw('linrbc','drvrbc');

% Set shock vector
mm=size(M);
mm=mm(1);
shock=zeros(1,mm);

sc=input('Choose 1 for A shock and 2 for G shock')

if (sc==1)
  % A shock
  shock(2)=1;
else
  % G shock
  shock(3)=1;
end
4.3 STEPS IN SPECIFYING THE MODEL

% Compute impulse response
nir=20;
IR=impkw(M,PI,G,ylist,xlist,lpd,shock,nir);

% plot impulse responses;
plotrbc
Chapter 5

Using the RE Solution

We now turn to a variety of applications of the RE solution, using the RBC case as an example.

5.1 Impulse Responses

The state space model (4,5) makes it easy to compute an impulse response to a shock vector $e_t$. This takes the form of

$$E_t Z_{t+j} - E_{t-1} Z_{t+j} = \Pi M^j G e_t$$

where $Z = \begin{bmatrix} y \\ x \end{bmatrix}$.

The program IMPKW.m is an example of a function file used to produce outputs from a set of RE models. It computes the impulse responses recursively

$$E_t s_t - E_{t-1} s_t = G e_t$$
$$E_t s_{t+j} - E_{t-1} s_{t+j} = M(E_t s_{t+j-1} - E_{t-1} s_{t+j-1})$$
$$E_t Z_{t+j} - E_{t-1} Z_{t+j} = \Pi(E_t s_{t+j} - E_{t-1} s_{t+j})$$

function IR=IMPKW(M,PI,G,ylist,xlist,lpd,shock,nir);
% Matlab File: impkw.m
% calculation of impulse response (for nir periods) using the
% Markov decision rules. The inputs and outputs are summarized by the statement:
% IR=impkw(M,PI,G,ylist,xlist,lpd,shock,nir)
% where G is the matrix in the driving process specification,
% x(t) = Q delta(t) and delta(t) = RHO * delta(t-1) + G e(t),
% lpd indicates the location of the predetermined variables and
% shock indicates the value that e(t) should take. If no shock vector
% is specified, then a menu allows a choice of shocks.

nM=rows(M);
nPI=rows(PI);

if (nargin<7)
    clc
disp('No shock vector specified. User can supply shock if G=eye(M)')
disp(['The matrix M contains ', num2str(nM), ' elements'])
klist=ylist(lpd,:)
nk=rows(klist);
for i=1:1:nk;
    sklist(i,:)=['%2.0f', ' ', ylist(lpd(i),:)];
end
disp(['The first ', num2str(nk), ' elements are predetermined variables'])
disp(sklist)

nx=rows(xlist);
for i=1:1:nx;
    sxlist(i,:)=['%2.0f', ' ', xlist(i,:)];
end
disp(['The next ', num2str(nx), ' elements are exogenous variables (x)'])
disp(sxlist)
5.1 IMPULSE RESPONSES

shk=input('select shock # ');
szshk=input('choose size of shock ');
shock=zeros(1,nM);
shock(:,shk)=szshk
pause
nir=input('choose length of impulse response ');
end

nM=rows(M);
nPI=rows(PI);

% IMPULSE RESPONSES

% The calculation of impulse responses involves two elements. First, it is
% necessary to specify initial conditions for the controlled state(k). Second,
% one needs to specify an values of the exogenous variables, i. e.,
% an (nM x 1) vector of initial conditions.

if (rows(shock)<cols(shock))
disp('transposing shock vector')
shock=shock';
end

if max(size(shock))~=max(size(M));
disp('inadmissible run')
IR=0;
else
S = shock;

% Define initial value of state vector for use in later work,
% since S will be updated in computation of impulse response.
SI=S;
% Initialize the impulse response matrix
IR=zeros(nir,nPI+1);

% Calculate the impulse response for the nir periods.
for i=1:nir;
    %Index of Impulse Response Period;
    IR(i,1)=i;
    %Observables;
    IR(i,2:nPI+1)=(PI*S)';
    % Update the state vector:
    S = M*S;
end
end

The impulse responses are calculated and selective responses are plotted by plotrbc.m

% plotrbc.m
vlistrbc
load idrbc
load ssrbc

ny=ROWS(ylist);

% Plot selected variables
date=IR(:,1);
if (sc==1)
    subplot(221),plot(date,IR(:,1+ia+ny),'k-'),
    title('Productivity')
5.2 SIMULATIONS

5.2 Simulations

5.2.1 Artificial Shock Inputs

5.2.2 VAR Inputs

5.2.3 Filtering Results

5.3 Population moments

5.4 Spectra

5.4.1

5.5 Population moments of filtered series
Chapter 6

Strategies for Larger Models

–Ex: IRBC
Chapter 7

Additional Examples

7.1 New IS-LM

7.2 New Keynesian Taylor-Levin Staggering

7.3 New Neo-Classical Synthesis (NNS)
   – King-Wolman, King-Watson 1996

7.4 Open Economy NNS

7.5 State-Dependent Pricing

7.6 Optimal Monetary Policy
Appendix

Debugging and Error Messages

Researchers commonly make mistakes in setting up a dynamic model. This section discusses methods for (1) locating mistakes using the included debugging program (DEBUGKW.m), and (2) the interpretation of error messages encountered while attempting to use solution code (RESOLKW.m and its components).

1. Debugging

After running the system file to set up the matrices A, B and C, run DEBUGKW.m. The following options are presented:

(0) quit
(1) check for rows of zeros in [A B]
(2) check for det(Az-B) nonzero graphically
(3) check LR multipliers
(4) check # stable roots versus # predetermined variables
(5) check equations of the model

1.1 Checking for Rows of Zeros

If one or more rows of zeros are present in [A B], then the system does not meet the requirements for solvability. This can occur quite easily given that the matrices A and B are initialized to be square, conformable to y, and filled with zeros.
The debug program checks for rows of zeros and reports their location in the event that any are found. The default tolerance level is $10^{-6}$ and is easily modified by editing the appropriate lines in `DEBUGkw.m`.

**Example 1 (Muth Model Two)** The program `sysmuth2.m` in the directory `\kwre\muth2debug` is an example of a model in which this error results from a misspecified system file. Because the number of equations was not updated, the third equation overwrites the second and the third row is left as a row of zeros. The code reads:

```plaintext
% EQN2
[elist, eqn] = ADDEQN('demand', elist, eqn);
A(eqn, ik) = 1; B(eqn, iq) = 1; B(eqn, ip) = alpha; CF(eqn, ix) = -1;
% EQN3
[elist, eqn] = ADDEQN('inventory', elist, eqn);
A(eqn, ik) = 1; A(eqn, ip) = -b*theta; B(eqn, ip) = -theta;
```

Clearly, if the first line of the inventory equation code is omitted, the `ADDEQN` function is not executed and the variable `eqn` is not updated its proper value.

### .1.2 Checking $|Az - B| \neq 0$

Theoretical analysis of these models shows it to be a necessary condition that $|Az - B| \neq 0$ for some $z$. It is easy to check this condition by computing the determinant at various values of $z$ and investigating the results.

This condition can be violated for a variety of reasons, including the existence of a row of zeros as discussed above. Given that $[A B]$ does not contain a row of zeros, it is likely that the system contains a repeated equation. Removing one of the duplicate equations solves the problem.

### .1.3 Checking Long-Run Multipliers

When considering a model of the general form

$$AE_{ty_{t+1}} = By_t + C(F)x_t$$

the long-run response of a permanent change in the exogenous vector of
variables \( x_t \) affects only the steady-state,

\[
Ay = By + C(1)x \\
(A - B)y = C(1)x \\
y = (A - B)^{-1}C(1)x
\]

and the change can be written as

\[
\Delta yy = (A - B)^{-1}C(1)\Delta x
\]

where \( C(1) = C_0 + C_1 + ...C_p \). This is included as an option because the analysis of long-run multipliers is sometimes useful in checking the specification of a system. The importance of properties like this may already be familiar because of their frequent use as identifying assumptions in VAR orthogonalization.

For example, the analytical properties of a class of utility functions commonly used in RBC models (King and Rebelo [1999]) are such that there is no long run response of labor to permanent changes in wages. Checking the long-run multipliers is an easy way to see if such a property holds. Finding it fails to hold can be a sign of a major misspecification or a coding error (under appropriate conditions).

.1.4 Checking the Roots

As described in Blanchard and Kahn [1980]: if a non-singular model is to possess a rational expectations solution which is both unique and stable, the number of stable roots must be equal to the number of predetermined variables. The same restriction carries over to non-singular systems, in which some roots can be viewed as being infinite. (The first chapter yields a bit of intuition to the idea of infinite roots as representative of dynamic identities within the system.)

The condition described in Blanchard and Kahn [1980] can be checked using the debugging program or by looking at the roots of \(|Az - B| = 0\) directly.

In theory, system reduction should have no effect on the generalized eigenvalues of the system. That is, the generalized eigenvalues given by the polynomial \(|Az - B| = 0\) are unaffected by transformations arising from the
multiplication of nonsingular matrices. For $|T| \neq 0$,

$$0 = |A^*z - B^*|$$
$$= |TAz - TB|$$
$$= |T(Az - B)|$$
$$= |T||Az - B|$$
$$= |Az - B|.$$  

However, it is certainly possible that numerical difficulties present such an effect. As an added option, one may check to see that the generalized eigenvalues of the system are the same before and after system reduction takes place, thereby checking a number of potential numerical issues.

**Example 2 (Checking Roots)** Recall that in the Muth model discussed above, the inventory equation is given by $k_{t+1} = \beta E_t p_{t+1} - p_t$. If a negative sign is accidentally entered in front as $k_{t+1} = -[\beta E_t p_{t+1} - p_t]$, the model contains too few stable eigenvalues. As a result of this error the roots appear in a complex pair and no solution exists according to the analysis of Blanchard and Kahn [1980].

### 1.5 Examining the Equations

In the example above ("Checking Roots"), it is necessary to examine the code in the system file or to examine the specification of the equations of the model. The fifth option presented by the debugging program conveniently displays the equations of the model complete with their numerical coefficients. Continuing from above, the improperly entered equations are displayed by the debugging program as,

**Example 3**

EQN 1: $0 = -1 \cdot q(t) + 0.1 \cdot p(t) + 1 \cdot k(t)$

EQN 2: $1 \cdot k(t+1) = 1 \cdot q(t) + 0.2 \cdot p(t) + 0.25253 \cdot x(t)$

EQN 3: $3.96 \cdot p(t+1) + 1 \cdot k(t+1) = 4 \cdot p(t) - 1 \cdot x(t)$

From this we can see that the effect of current price on the inventory stock is positive rather than negative.
.2 Error Messages

.2.1 Error messages from REDKW.m

System reduction is the initial step in the solution algorithm. As discussed previously, this amounts to finding blocks of zeros in the coefficient matrix $A$ (perhaps after transformations to $A$, $B$, and $C$) and then using equations of the form

$$0 = b_{1A} \Lambda_t + b_{1k} k_t + c_1 x_t$$

to acquire solutions for some of the nonpredetermined variables as functions of the others ($k_t$ and $x_t$). The issues here are not really dynamic, but pertain only to the solution of a linear system. We illustrate possible issues by considering a system with two equations in which there are two nonpredetermined variables ($\lambda_{it}$), one predetermined variable ($k_t$) and one exogenous state variable ($x_t$):

$$0 = \begin{bmatrix} \theta_1 & \theta_2 \\ 0 & \theta_3 \end{bmatrix} \begin{bmatrix} \lambda_{1t} \\ \lambda_{2t} \end{bmatrix} + \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} k_t + \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} x_t$$

$$k_{t+1} = \mu k_t + \psi_3 x_t, \text{ for } \mu \in (0, 1)$$

The entire issue is whether the first two equations are well-specified. There are three possible error messages, each of which is illustrated by by three examples.

Restrictions Between Nonpredetermined and Exogenous Variables

If exist one or more equations in which all coefficients on the nonpredetermined ($\lambda_{it}$) variables are zero, then the model is misspecified. It specifies nonsensical restrictions between predetermined variables and exogenous variables. In our example, such misspecifications arise if $\theta_3 = 0$, $\theta_1 = \theta_2 = 0$, or both. In this event, the error message displayed is

REDKW.M: There are no new flows that can be isolated (nnf=0) and the matrix A is singular.
The system cannot be reduced.
Inconsistent Equations

The equations of the model may be inconsistent with one another. In our example, such inconsistencies arise if $\theta_1 = 0$ and $\theta_1 = \theta_2$. Clearly, the first and second equations are the same and it is not possible for the nonpredetermined variables to be distinct linear combinations of the other variables. It is also possible that the equations of the model are nearly inconsistent and are considered inconsistent by the solution algorithm at the tolerance level in use. In either event, the error message displayed is

\[ \text{REDKW.M: The system reduction program (redkw.m) has encountered major problems with flow equations and predetermined variables. Either:} \]

(1) The model is well specified, but there is a computational problem. If the model has an equation of the form $0 = sm*\text{lamj}(t) + cl(F)*k(t)$ where \( \text{lamj}(t) \) is nonpredetermined, the coefficient $sm$ may be small enough that $cl(F)/sm$ falls above the tolp value (1.0000e-010 by default).

(2) The model is misspecified. KW prove that any model which has a solution is reducible by this code.

.2.2 Error messages from DYNKW.m

The model is in Blanchard and Kahn [1980] form after system reduction is complete, leaving two requirements for the existence of a rational expectations solution which is both stable and unique. (1) The “counting rule”: the number of stable eigenvalues is equal to the number of predetermined variables. Equivalently, the number of unstable eigenvalues is equal to the number of nonpredetermined variables. (2) It must be possible to associate unstable canonical variables and nonpredetermined variables. This is a requirement on the rank of a particular matrix.

Consider the following simple rational expectations model

\[
E_t \begin{bmatrix} y_{1,t+1} \\ y_{2,t+1} \end{bmatrix} = \begin{bmatrix} \mu_1 & 0 \\ 0 & \mu_2 \end{bmatrix} \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} x_t
\]
.2 ERROR MESSAGES

Case 1: A Unique and Stable Solution

If (1) the number of unstable eigenvalues is equal to the number of nonpredetermined variables and (2) unstable canonical variables are associated with nonpredetermined variables, then a unique and stable rational expectations equilibrium exists.

This occurs if (a) $y_1$ and $y_2$ are predetermined, and $\mu_1$ and $\mu_2$ are stable; (b) $y_1$ is predetermined and $y_2$ is nonpredetermined, and $\mu_1$ is stable and $\mu_2$ is unstable; (c) $y_1$ is nonpredetermined and $y_2$ is predetermined, and $\mu_1$ is unstable and $\mu_2$ is stable; or (d) $y_1$ and $y_2$ are nonpredetermined, and $\mu_1$ and $\mu_2$ are unstable.

Case 2: Too Many Stable Roots and Multiple Equilibria

If the number of stable eigenvalues is greater than the number of predetermined variables, then a unique solution does not exist.

This occurs if (a) $y_1$ and $y_2$ are nonpredetermined, and either $\mu_1$ is stable, $\mu_2$ is stable or both; (b) either $y_1$ or $y_2$ is nonpredetermined, but both $\mu_1$ and $\mu_2$ are stable. The error given in this case is

Using a critical value of $bcrit = 0.99999$, there are too few unstable roots (i.e., roots such that $bcrit*\mu >= 1$). Hence there are multiple stable RE solutions to the model. This can be due to:

1. An inappropriate choice of $bcrit$,
2. A model that cannot be uniquely solved, or
3. An error in the system specification.

To investigate system errors, use debugkw.m.

All results obtained after this point are meaningless.

Case 3: Too Few Stable Roots and Nonexistence

If the number of stable roots is less than the number of predetermined variables, then a non-explosive solution does not exist. Equivalently, if the number of unstable roots is greater than the number of nonpredetermined variables, then a non-explosive solution does not exist. In this event, the error given is similar to the one given in Case 2:

Using a critical value of $bcrit = 0.99999$, 

there are too many unstable roots (i.e., roots such that 
bcrit*mu >=1). Hence, there is no stable solution to
the model. This can be due to:
(1) An inappropriate choice of bcrit,
(2) An intrinsically unsolvable model, or
(3) An error in the specified system.
To investigate system errors, use debugw.m.
All results obtained after this point are meaningless.

Case 4: Failure of the Rank Condition and Nonexistence

After system reduction, the decoupled system is partitioned into two sys-
tems: one describing variables with stable dynamics (s_t) and one describing
variables with dynamics (u_t):

\[
E_t u_{t+1} = J_u u_t + \Psi^*_ux_t + \Psi^*_uy_t x_{t+1}
\]
\[
E_t s_{t+1} = J_s s_t + C^*_ux_{t+1} + C^*_ys_t x_{t+1}
\]

where \(J_u\) and \(J_s\) are partitions from the Jordan matrix, one containing stable
roots and the other containing unstable roots.

The unstable equations of this model can be solved by unwinding forward
(Sargent’s procedure):

\[
u_t = J_u^{-1} [E_t u_{t+1} + \Psi^*_ux_t + \Psi^*_uy_t x_{t+1}]
\]
\[= - \sum_{i=0}^{\infty} (J_u^{-1})^{i+1} E_t [\Psi^*_ux_{t+i} + \Psi^*_uy_t x_{t+i+1}]
\]

The nonpredetermined and predetermined variables in which we are truly
interested are related to the solution derived for the stable and unstable
partitions above by

\[
\begin{bmatrix}
V_{u\lambda} & V_{uk} \\
V_{sk} & V_{sk}
\end{bmatrix}
\begin{bmatrix}
\lambda_t \\
k_t
\end{bmatrix}
= \begin{bmatrix}
v_t \\
 s_t
\end{bmatrix}
\]

Hence, the solution for the nonpredetermined variables is found by

\[\lambda_t = V_{u\lambda}^{-1} [u_t - V_{uk}k_t]\]
If the matrix $V_{u\lambda}$ cannot be inverted, a solution for the nonpredetermined variables does not exist. In this event, the error given is,

\texttt{DYNKW.m: rank condition on Llam is violated.}
\texttt{It is not possible to associate the unstable canonical variables with the unstable roots: the RE rank condition is violated and the model cannot be solved.}

\section*{2.3 Errors from MDRKW.m}
This program is involved in the computation of the linear state space system (Markov Decision Rules). It issues a single error message which has not been encountered frequently. If there are imaginary components to the roots of the solution which are “too large,” the program may issue an error.
Jordan form (canonical form)

An $l \times l$ Jordan matrix $J$ is a matrix with block entries on its diagonal and zero elements off-diagonal.

$$J \equiv \begin{bmatrix}
J_1 & 0 & 0 & \cdots & 0 \\
0 & J_2 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & 0 \\
0 & 0 & 0 & 0 & J_l
\end{bmatrix}$$

A Jordan matrix contains one block entry for each unique root (in the Jordan matrix above, there are $l$ roots). The block for the $i$th root $z_i$ is

$$J_i \equiv \begin{bmatrix}
z_i & 1 & 0 & \cdots & 0 \\
0 & z_i & 1 & \cdots & 0 \\
0 & 0 & z_i & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & 1 \\
0 & 0 & 0 & 0 & z_i
\end{bmatrix}$$

The dimension of each block $J_i$ corresponds to the number of repetitions of the $i$th root $z_i$. The upper diagonal elements of each block are equal.
to 1 and all other elements are zero. Where a root is finite and unique, its Jordan block entry is simply a scalar equal to the value of the root. In solving the model, we partition the Jordan matrix based on the stability properties of the roots:

$$J \equiv \begin{bmatrix} J_u & 0 \\ 0 & J_s \end{bmatrix}$$

For singular systems, there is an additional submatrix $N$ which contains zeros on and below the diagonal, and a combination of ones and zeros above the diagonal. In this case, the partition is:

$$J \equiv \begin{bmatrix} N & 0 & 0 \\ 0 & J_u & 0 \\ 0 & 0 & J_s \end{bmatrix}$$
Bibliography


