COMPUTATIONAL ECONOMICS

Dynamic Programming in MATLAB

Revised Edition

David A. Kendrick
Ruben P. Mercado
Hans M. Amman

1 Introduction

Dynamic optimization encompasses a group of mathematical techniques used in economics to model the intertemporal behavior of economic agents under the assumption of forward-looking optimizing behavior. For example, it can be used to model, among others, the behavior of:

- a policy maker who tries to determine the optimal path of policy variables in order to achieve some specified targets for GDP or the inflation rate;
- firms that are assumed to choose the optimal path of investment in order to maximize intertemporal profits or their present value;
- consumers who are assumed to face intertemporal choices between present and future consumption.

In general terms, dynamic optimization deals with the problem of obtaining a sequence of optimal choices under given dynamic constraints. Calculus of variations, optimal control, and dynamic programming are the most commonly used techniques for dynamic optimization. In this chapter, we focus on what is known as discrete time dynamic programming, a technique particularly suited for computational implementation, given its recursive structure. Specifically, we deal with a special case of dynamic optimization that is known as the quadratic linear problem (QLP), a very popular kind of problem in which the goal is to optimize an intertemporal quadratic objective function subject to dynamic linear constraints that hold as equalities.[1,2] The QLP is used here for a deterministic model because it is also well adapted and widely used for the types of stochastic models that we progress to. We dealt with a QLP in Chapter **??**, but did not exploit the recursive nature of the typical QLP and solved the problem with nonlinear programming in GAMS. That approach can easily deal with inequality constraints. However, rather than a recursive solution method, it uses a *stacking* method, which transforms a dynamic problem into a larger static one.

[1] Though the choice of the quadratic criterion can be somewhat limiting many nonlinear models can be usefully approximated by QLP models and then solved with successive approximations.

[2] For dynamic models in which there are inequalities, mathematical programming methods like those used with GAMS are more appropriate than the *Riccati* methods discussed in this chapter. On the other hand, the quadratic linear control theory models with equality constraints are most useful when one wants to deal with stochastic elements in the form of additive noise terms and uncertain parameters.

In this chapter we turn to **MATLAB**, which uses a vector-matrix paradigm more suitable for dealing with the standard QLP since, as we will see later, the solution of these problems involves a series of vector and matrix operations. We introduced **MATLAB** in earlier chapters. In Appendix A we provide the listing and the book web site includes the file for the **MATLAB** representation of the model we develop in this chapter. This code was based on an earlier one in GAUSS by Hans Amman and was created in **MATLAB** by Huber Salas and Miwa Hattori.

This chapter begins with a brief introduction to the mathematics of QLP. Then, as a simple example, a small macroeconometric model is introduced. Finally, the model is input to **MATLAB** and solved.

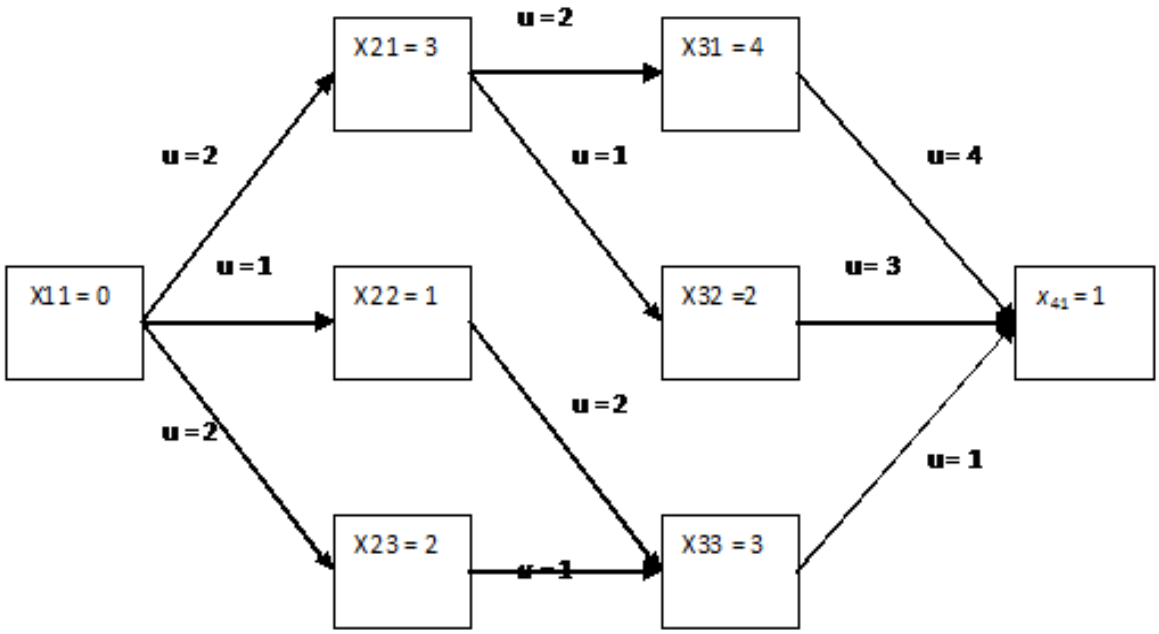
2 The Concept of Dynamic Programming

The dynamic programming approach, developed by Richard Bellman (1957), can be illustrated with a simple example. The diagram in Figure 1 represents different ways of going from node x_{11} to x_{41} applying a specific action u to move from one node to another. As a concrete example, we can interpret the nodes as towns and the actions as means of transportation (e.g., car, plane, train). Each town has an associated cost (e.g., room and board), as does each means of transportation. The problem is to find the minimum cost path or, more precisely in the case of dynamic programming, a feedback rule to determine the optimal action u as a function of the node we are at.

In a more general formulation, we can think of the diagram in Figure 1 as representing the time path of a system that can be driven from one state (node) to another by manipulating controls, the u's, Going back to a more concrete example, now in time and not in space as above, you can think of a macroeconomic example in which the state variable is the inflation rate, or alternatively the level of GDP, and the control variable is the money supply. The problem is the one that would confront a monetary authority trying, at a minimum cost to society, to drive inflation down or the GDP level up, facing a number of alternative economic paths to achieve that goal.

To solve the problem, the dynamic programming approach uses a recursive method that works backward. For the example at hand, the method proceeds

Figure 1: Dynamic Programming



as follows.

1. Compute the cost J of each segment in the last stage (i.e., add the cost of x_{41} and the cost of the corresponding u. There are obviously three values:

$$J(x_{31}, x_{41}) = 5; \quad J(x_{32}, x_{41}) = 4; \quad J(x_{33}, x_{41}) = 2$$

2. Compute the cost of each feasible optimal sequence of segments from the $x_{2..}$ nodes:

$$\begin{aligned} J(x_{21}, x_{41}) &= min[J(x_{31}, x_{41}) + \text{cost of segment } (x_{21}, x_{31}); \\ &\qquad J(x_{32}, x_{41}) + \text{cost of segment } (x_{21}, x_{32})] \\ &= min[5+6; 4+3] = 7 \end{aligned}$$

which implies that the optimal sequence of controls to go from x_{21} to x_{41} is $[u(x_{21}, x_{32}), u(x_{32}, x_{41})]$. Obviously, the optimal sequence (x_{22}, x_{41})

is $[u(x_{22}, x_{33}),\ u(x_{33}, x_{41})]$ (the only one feasible) with $J(x_{22}, x_{41}) = 7$, whereas for (x_{23}, x_{41}) it is $[u(x_{23}, x_{33})u(x_{33}, x_{41})]$ with $J(x_{23}, x_{41}) = 6$.

3. Compute the cost of the optimal feasible sequence of segments fromx_{11}:

$$\begin{aligned} J(x_{11}, x_{41}) &= min[J(x_{21}, x_{41}) + \text{cost of segment } (x_{11}, x_{21}); \\ &\qquad J(x_{22}, x_{41}) + \text{cost of segment } (x_{11}, x_{22}); \\ &\qquad J(x_{23}, x_{41}) + \text{cost of segment } (x_{11}, x_{23})] \\ &= min[7 + 5; 7 + 2; 6 + 4] = 9 \end{aligned}$$

Thus the optimal sequence of controls for the problem is

$$u(x_{11}, x_{22}), u(x_{22}, x_{33}), u(x_{33}, x_{41})$$

3 A Simple Quadratic Linear Problem

In most economic applications we find problems in which we represent an economic agent, institution, or the economy as a whole as a system of state variables that evolves through time. This system can be manipulated by means of a set of control variables in order to minimize (or maximize) an intertemporal cost (or value) function. A very typical problem is one in which the cost function is quadratic and the economic system is represented by linear equations. For a very simple one-state–one-control case, the problem is expressed as finding the controls

$$(u_k)_{k=0}^{N-1}$$

to minimize a quadratic criterion function J of the form

$$J = \sum_{k=0}^{N} L_k(x_k) \tag{1}$$

with

$$L_k(x_k) = \frac{1}{2} x_k^2 \tag{2}$$

subject to the dynamic equation

$$x_{k+1} = ax_k + bu_k \tag{3}$$

and the initial condition

$$x_0 \tag{4}$$

where

$$\begin{aligned} L_k &= \text{loss in period } k \\ x_k &= \text{a state variable at time } k \\ u_k &= \text{a control variable at time } k \\ a &= \text{a state parameter} \\ b &= \text{a control parameter} \end{aligned}$$

As we already know, the dynamic programming approach works by solving the problem backward in time, determining optimal feedback rules for choosing the control vector as a function of the state vector at each stage—each time period—of the problem. Thus it transforms the original optimization problem into a sequence of subproblems. Its crucial notion is the optimal cost-to-go, which is the cost along the minimum-cost path from a given time period to the terminal period of the problem. For QLP, the cost-to-go is a quadratic function of the state of the system at time k, which for our particular problem is

$$J_k^* = \min_{u_k} \left\{ J_{k+1}^* + L_k \left(x_k \right) \right\} \tag{5}$$

J_k^* is the optimal value of J at time k. Starting from the terminal period, the cost is

$$J_N^* = \frac{1}{2} x_N^2 \tag{6}$$

Applying the principle of dynamic programming, the optimal cost-to-go at period $N-1$ is the minimum of the optimal cost-to-go at state x_N in time N and the cost incurred in time period $N-1$:

$$J_{N-1}^* = \min_{u_{N-1}} \left\{ J_N^* + L_{N-1} \left(x_{N-1} \right) \right\} \tag{7}$$

where L_{N-1} is the cost function J for $N-1$ in equation (2). Thus we have

$$J_{N-1}^* = \min_{u_{N-1}} \left\{ \frac{1}{2} x_N^2 + \frac{1}{2} x_{N-1}^2 \right\} \tag{8}$$

To carry on the minimization we need all the variables in equation (8) expressed at time $N-1$. Substituting equation (3) for x_N into (8) and expanding, we obtain

$$J_{N-1}^{*} = \min_{u_{N-1}} \left\{ \frac{1}{2} x_{N-1}^2 a^2 + x_{N-1} u_{N-1} ab + \frac{1}{2} u_{N-1}^2 b^2 + \frac{1}{2} x_{N-1}^2 \right\} \tag{9}$$

The first-order condition for the minimization is

$$\frac{\partial J_{N-1}^{*}}{\partial u_{N-1}} = x_{N-1} ab + u_{N-1} b^2 = 0 \tag{10}$$

Solving (10) for u_{N-1}, we obtain a feedback rule that gives us the optimal control as a function of the state:

$$u_{N-1} = G_{N-1} x_{N-1} \tag{11}$$

where G_{N-1}, known as the feedback gain coefficient, is

$$G_{N-1} = -\frac{a}{b} \tag{12}$$

If we repeat the procedure for J_{N-2}^{*}, and so on, we observe that a general form for the feedback rule emerges:

$$u_k = G_k x_k = -\frac{a}{b} x_k \tag{13}$$

Thus, the feedback rule (13) tells us what the optimal action is at each point in time, depending of the state of the system. For our particular problem the feedback gain coefficient is a constant. However, as we will see later, this is not the case for more general problems.

To obtain the solution paths for the controls and the states, we have to start from the initial condition equation (4) to obtain the optimal control from equation (13). Then we can solve equation (3) to get the next optimal state and go back to equation (13) for the optimal control and so on. Thus we can see that the solution paths are obtained from a forward loop. Knowing the optimal states, we can compute the corresponding criterion value from equation (1).

The **MATLAB** representation of the solution procedure of the simple problem we presented is straightforward and is available in the book web site in file `qlpsimple.m`. We begin by initializing the problem for four periods (from zero to three), and we assign values for the parameters and the initial condition. We also set to zero the vectors containing the optimal states and controls; the variable sum, which contains the criterion value; and the index variable `k`:

```
t = 3; a = 0.7; b = -0.3; x0 = -1;
u = zeros(1,t); x = zeros(1,t);
sum = 0; k = 0;
```

Next we write the forward loop:

```
xold = x0;
while k <= t;
   glarge = - a / b;
   uopt = glarge * xold;
   xnew = a * xold + b * uopt;
   sum = sum + 0.5 * xold\^{}2;
   x(1,k+1) = xold;
   u(1,k+1) = uopt;
   xold = xnew;
   k = k+1;
end;
```

The loop begins with the assigment of the initial condition to the `xold` variable, and `runs` as long as $k <= t$. At each pass, the values of the feedback gain coefficient `glarge`, the optimal control `uopt`, and the optimal state `xnew` are computed and stored in the corresponding positions of vectors x and u, and the corresponding value of the criterion function is computed and added to the sum variable. Finally, the results are printed, previously transposing the vectors so that the results are displayed in colums:

```
u = u'
x = x'
Criterion = sum
```

Table 1: Optimal States and Controls

k	x	u
0	–1	$-2\frac{1}{3}$
1	0	0
2	0	0
3	0	0

The solution values for the optimal states and controls are shown in Table 1. Note that the value of x is driven to zero in just one period and with a single control action. Even if we extend the number of periods, regardless of the initial condition, just a single control action suffices. Why? The answer lies in the fact

that the control variable is not part of the criterion function to be minimized. Thus, there is no cost associated with the use of the control and one can immediately jump to any necessary value to bring the state variable to zero. However, in most cases this variable is included in the criterion function. Moreover, both the state variable and the control variable have associated specific weights in the function to represent relative priorities in terms of the cost of having the state variable off-target versus the cost of using the control.

In the next section we present a more comprehensive and general problem—a many-state–many-control problem. There are weights on states and controls and cross terms in the criterion function, and the system of equations contains constant terms. We will see that the main logic to obtain a solution is the same as we used in this section. However, some new elements appear as part of it, such as Riccati matrices and vectors, and the solution procedure involves a backward loop together with a forward loop similar to the one we presented in this section.

4 A More General Quadratic Linear Problem

The quadratic linear problem (QLP) has linear system equations and a quadratic criterion and may be written as: find

$$(u_k)_{k=0}^{N-1}$$

to minimize the criterion

$$J = L_N(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k) \tag{14}$$

with

$$L_N(x_N) = \frac{1}{2} x_N' W_N x_N + w_N' x_N \tag{15}$$

$$L_(x_k, u_k) = \frac{1}{2} x_k' W_k x_k + w_k' x_k + x_k' F_k u_k + \frac{1}{2} u_k' \Lambda_k u_k + \lambda_k' u_k \tag{16}$$

subject to the system equations

$$x_{k+1} = Ax_k + Bu_k + c \quad k = 0, 1, \cdots, N-1 \tag{17}$$

and the initial conditions

$$x_0 \tag{18}$$

where

$$\begin{aligned} L_k &= \text{loss in period } k \\ x_k &= \text{state vector at time } k \\ u_k &= \text{control vector at time } k \\ W_k &= \text{state vector priority matrix at time } k \\ F_k &= \text{cross state–controll priority matrix at time } k \\ \Lambda_k &= \text{control vector priority matrix at time } k \\ A, B, c &= \text{parameter matrices and vector} \end{aligned}$$

In addition, the notation

$$(u_k)_{k=0}^{N-1}$$

indicates the set of control vectors from period zero through period $N-1$, that is, $(u_0, u_1, u_2, \cdots, u_{N-1})$. Period N is the terminal period of the model. Thus the problem is to find the time paths for the m control variables in each period for the time periods from 0 to $N-1$ to minimize the quadratic form in equations (14)-(16) while starting with the initial conditions (18) and following the difference equations (17). The derivation of the solution for this model is described in detail in Kendrick (1981, chapter 2). Here we provide an outline of the procedure.

We know that for QLP, the cost-to-go is a quadratic function of the state of the system at time k, which for our problem is

$$J_k^* = \frac{1}{2} x_k' K_k x_k + p_k' x_k + v_k \tag{19}$$

where K_k and p_k are called the Riccati matrix and vector, respectively, and where v_k is a scalar. Starting from the terminal period we have

$$J_N^* = \frac{1}{2} x_N' K_N x_N + p_N' x_N + v_N \tag{20}$$

From equation (15), the cost at the terminal period is

$$\frac{1}{2} x_N' W_N x_N + w_N' x_N \tag{21}$$

Thus, from equations (20) and (21) we obtain the result that $v_N = 0$ and the terminal conditions

$$\begin{aligned} K_N &= W_N && (22) \\ p_N &= w_N && (23) \end{aligned}$$

The optimal cost-to-go at period $N-1$ is the minimum of the optimal cost-to-go at state x_N in time N and the cost incurred in time period $N-1$:

$$J^*_{N-1} = \min_{u_{N-1}} \left\{ J^*_N + L_{N-1}\left(x_{N-1}, u_{N-1}\right) \right\} \tag{24}$$

where L_{N-1} is the cost function J for $N-1$ as defines in equation (16). Analogously, the optimal cost-to-go at period $N-2$ is

$$J^*_{N-2} = \min_{u_{N-2}} \left\{ J^*_{N-1} + L_{N-2}\left(x_{N-2}, u_{N-2}\right) \right\} \tag{25}$$

and so on. Carrying out the corresponding substitutions in equations (24) and (25), minimizing, solving the first-order conditions with respect to the control vectors, and rearranging, we observe the emergence of a general solution that has the form of a feedback rule:

$$u_k = G_k x_k + g_k \tag{26}$$

where

$$\begin{aligned} G_k &= -\left[B' K_{k+1} B + \Lambda'_k\right]^{-1} \left[F_k + B' K_{k+1} A\right] && (27) \\ g_k &= -\left[B' K_{k+1} B + \Lambda'_k\right]^{-1} \left[B'\left(K_{k+1} c_+ p_{k+1}\right) + \lambda_k\right] && (28) \end{aligned}$$

and the expressions for the Riccati matrix and vector are

$$\begin{aligned} K_k &= A' K_{k+1} A + W_k - \left[A' K_{k+1} B + F\right] \left[B' K_{k+1} B + \Lambda'_k\right]^{-1} \cdot \\ &\quad \left[F' + B' K_{k+1} A\right] && (29) \\ p_k &= -\left[A' K_{k+1} B + F\right] \left[B' K_{k+1} B + \Lambda'_k\right]^{-1} \\ &\quad \left[B'\left(K_{k+1} c_+ p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1} c_+ p_{k+1}\right) + w_k && (30) \end{aligned}$$

These equations look formidable but essentially they involve only the matrices and vector A, B, and c from the system equations (17) and the matrices W and Λ from the criterion function (14) in addition to the lead values of the

Riccati matrix K_{k+1} and the Riccati vector p_{k+1}.

To obtain the solution paths for the controls and the states, we have to start at the end and work backward. We follow that procedure here by beginning with the terminal conditions and working back to the initial period while solving for the Riccati matrices and vectors. Then we use the initial conditions, the feedback rule, and the system equations to solve forward in time while computing the state and control variables.

Thus we begin by integrating backward in time starting with the terminal conditions

$$K_N = W_N \tag{31}$$
$$p_N = w_N \tag{32}$$

The backward integration is done by solving the Riccati matrix and vector equations:

$$\begin{aligned} K_k &= A'K_{k+1}A + W_k \\ &- \left[A'K_{k+1}B + F\right] \left[B'K_{k+1}B + \Lambda_k'\right]^{-1} \left[F' + B'K_{k+1}A\right] \end{aligned} \tag{33}$$

$$\begin{aligned} p_k &= -\left[A'K_{k+1}B + F\right] \left[B'K_{k+1}B + \Lambda_k'\right]^{-1} \\ &\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1}c + p_{k+1}\right) + w_k \end{aligned} \tag{34}$$

As will be shown later in the computational section of this chapter, these calculations can be programmed into **MATLAB** and solved in a very straightforward way. Using equations (31) through (34) we begin the procedure with K_N, which is obtained from equation (31). This matrix is then used first in equation (33) to compute K_{N-1}, and then K_{N-1} is used in that equation again to compute K_{N-2}, and so on, until the K_0 matrix has been computed. A similar procedure is used to calculate the Riccati vectors p_k using equations (32) and (34).

Once all the Riccati matrices and vectors have been computed, then a forward integration loop is started. In this loop the feedback gain matrix G_k and vector g_k from equations (27) and (28) are computed for each time period using those expressions, that is,

$$G_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1} \left[F_k + B'K_{k+1}A\right] \tag{35}$$
$$g_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1} \left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] \tag{36}$$

These expressions, like those for the Riccati matrix and vector shown earlier, are somewhat complicated but are easily programmed and solved in **MATLAB**. The feedback gain matrix and vector are then used in the feedback rule

$$u_k = G_k x_k + g_k \tag{37}$$

along with the initial condition, x_0, to compute the control vector, u_0. This value and the initial state are then used in the system equations from equation (17):

$$x_{k+1} = Ax_k + Bu_k + c \tag{38}$$

to compute the value of the state vector for the next period, x_1. Then the process is repeated beginning with equations (35) and (36) and using equations (37) and (38) until the control vectors, u_k, and the state vectors, x_k, have been computed for all time periods.

Further, in each pass through the forward loop the criterion value for that period is computed and added to the amount already accumulated using part of the criterion function, which is shown here in the tracking version rather than in the quadratic form[3]:

$$\frac{1}{2}\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \Lambda_k \left(u_k - \tilde{u}_k\right) \tag{39}$$

where

$$\begin{aligned} \tilde{x}_k &= \text{desired state vector} \\ \tilde{u}_k &= \text{desired control vector} \end{aligned}$$

We shall see shortly how intuitive it is to represent the mathematics of the solution procedure in **MATLAB**, but before doing so, we comment on other types of problems and solution procedures that arise when we move from the deterministic framework of the standard QLP problem to an environment in which uncertainty is taken into account.

The simplest way of introducing uncertainty is by assuming that it takes the form of additive uncertainty, that is, assuming that the system of linear equations is shocked in each period by additive noise. When we do this, the mathematical representation of the QLP problem is modified in two ways. First, the objective function is now an expected value, so equations (14)-(16) becomes

[3]Later in the chapter we discuss the transformation of the quadratic tracking version of the criterion function to the quadratic form.

$$J = E\{L_N(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k)\} \tag{40}$$

with

$$L_N(x_N) = \frac{1}{2}x_N' W_N x_N + w_N' x_N \tag{41}$$

$$L_(x_k, u_k) = \frac{1}{2}x_k' W_k x_k + w_k' x_k + x_k' F_k u_k + \frac{1}{2}u_k' \Lambda_k u_k + \lambda_k' u_k \tag{42}$$

where E is the mathematical expectation operator. Second, the system of equations now has an additive noise term denoted by ξ_k, that is,

$$x_{k+1} = Ax_k + Bu_k + c + \xi_k \tag{43}$$

with

$$E\{\xi_k\} = 0$$

$$E\{\xi_k \xi_k'\} = Q_k$$

$$E\left\{\begin{matrix}\xi_j \xi_k' \\ j \neq k\end{matrix}\right\} = 0$$

However, it can be shown that the solution procedure for this stochastic problem is the same as the one for the QLP problem, equations (31) to (37)][4]. This is why the solution procedure when additive uncertainty is present is named the certainty equivalent (CE). Note that this does not mean that the observed optimal paths for the states and the controls are the same in a QLP and a CE simulation, since in the CE simulation the system of equations is shocked by additive noise at each time period.

Consider now the case of multiplicative uncertainty. In this case, we have information about the variances and covariances of the dynamic equation parameters (matrices A, B, and vector c) and we want to exploit that knowledge when computing the optimal values of the controls to be applied period after period. In formal terms equations (40)-(43) still characterize the problem. However, it can be shown that the solution procedure is now somewhat different from the

[4]See Kendrick (1981), chapter 5.

one corresponding to equations (31) through (37). Indeed, the expectations operator now appears in those equations, as shown in equations (46) through (5):

$$K_N = W_N \tag{44}$$
$$p_N = w_N \tag{45}$$

as the fixed end condition and for the transition

$$K_k = E\left\{A'K_{k+1}A\right\} + W_k - \left[E\left\{A'K_{k+1}B\right\} + F\right] \left[E\left\{B'K_{k+1}B\right\} + \Lambda_k'\right]^{-1} \left[F' + E\left\{B'K_{k+1}A\right\}\right] \tag{46}$$

$$p_k = -\left[E\left\{A'K_{k+1}B\right\} + F\right] \left[E\left\{B'K_{k+1}B\right\} + \Lambda_k'\right]^{-1} \left[E\left\{B'K_{t+1}c\right\} + E\left\{B\right\}' p_{k+1} + \lambda_k\right] + E\left\{A'K_{k+1}c\right\} + E\left\{A\right\}' p_{k+1} + w_k \tag{47}$$

$$G_k = -\left[E\left\{B'K_{k+1}B\right\} + \Lambda_k'\right]^{-1} \left[F_k + E\left\{B'K_{k+1}A\right\}\right] \tag{48}$$

$$g_k = -\left[E\left\{B'K_{k+1}B\right\} + \Lambda_k'\right]^{-1} \left[E\left\{B'K_{k+1}c\right\} + E\left\{B\right\}' p_{k+1} + \lambda_k\right] \tag{49}$$

Note that there are now several terms involving the expectations of matrix products. To compute these expectations, following Chapter 6 in Kendrick (1981) we have to proceed as follows. In general terms, we define

$$D \equiv A'KB \tag{50}$$

where D, A, K, and B are all matrices, A and B being random and K deterministic. Thus

$$E\left\{D\right\} = E\left\{A'KB\right\} \tag{51}$$

A single element in D is d_{ij}. Then

$$E\left\{d_{ij}\right\} = E\left\{a_i'Kb_j\right\} \tag{52}$$

where a_i is the ith column of A and b_j is the jth column of B. It can be shown that

$$E\left\{d_{ij}\right\} = \left(E\left\{a_i\right\}\right)' KE\left\{b_j\right\} + tr\left[K\Sigma_{b_j a_i}\right] \tag{53}$$

where

$$\Sigma_{b_j a_i} = E\left\{\left[b_j - E\left\{b_j\right\}\right]\left[a_i - E\left\{a_i\right\}\right]'\right\} \tag{54}$$

is the covariance matrix for the *jth* column of B and the *ith* column of A and $\text{tr}[\cdot]$ is the trace operator, that is, the sum of the diagonal elements of the matrix in brackets.

To obtain the optimal paths for the controls and the states, we have to apply equations (44) through (49) in the same manner with backward and forward loops as we did for the deterministic QLP case presented earlier.

When a procedure like the one just presented is used to solve a problem in which multiplicative uncertainty is present, we say that we have an open loop feedback without update problem (OLF w/o update). Why do we say *without update*? In more complex simulations like the ones presented in Section 5, given the knowledge of the variances and covariances of the state equation parameters, we can consider a passive learning process. To do so, in each time period of the solution a projection-updating mechanism—usually a Kalman filter—is added to the solution method of the optimization problem in order to obtain, in each period, updated values of the next period parameters and of their variance-covariance matrix. You will have a better idea of this once you get to Section 5. Details of the mathematical form of these procedures are in Kendrick (1981, 2002).

In the following we turn to the **MATLAB** representation of the QLP solution procedure. However, first, in the next section, we introduce a small macroeconomic model to use as an example. CE, OLF, and parameter-updating procedures and examples are introduced Chapter **??** using Duali, a high-level software especially designed to deal with these types of problems.

5 The Macroeconomic Model

The macroeconomic model we deal with here, based on the work of Chow (1967) and Abel (1975), is a very simple one with two state variables and two control variables, that was used early in the control literature to perform some policy experiments. We did not choose it because of its nice properties but rather because it is very easy to implement in **MATLAB** and thus provides a good starting point for handling the more complex and realistic models like the ones to be presented in the next two chapters. The two state variables are

$$\begin{aligned} C_k &= \text{consumption} \\ I_k &= \text{investment} \end{aligned}$$

and the control variables are

$$\begin{aligned} G_k &= \text{government expenditures} \\ M_k &= \text{money supply} \end{aligned}$$

The reduced form of the model when estimated with data for the period 1954-II to 1963-IV as reported in Kendrick (1982), is

$$C_{k+1} = 0.914C_k - 0.016I_k + 0.305G_k + 0.424M_k - 59.437 \tag{55}$$

$$I_{k+1} = 0.097C_k + .424I_k - 0.101G_k + 1.459M_k - 184.766 \tag{56}$$

Note that the model exhibits *crowding out* behavior since the sign on the government expenditure variable in the investment equation is negative. The model can be written in the notation of the system equations (17) as

$$x_{k+1} = Ax_k + Bu_k + c \qquad k = 0, 1, \cdots, N-1,$$

with

$$x_k = \begin{bmatrix} C_k \\ I_k \end{bmatrix} \quad u_k = \begin{bmatrix} G_k \\ M_k \end{bmatrix}$$

$$A = \begin{bmatrix} 0.914 & -0.016 \\ 0.097 & 0.424 \end{bmatrix} \quad B = \begin{bmatrix} 0.305 & 0.424 \\ -0.101 & 1.459 \end{bmatrix} \quad c = \begin{bmatrix} -59.437 \\ -184.766 \end{bmatrix}$$

The initial conditions for the model are given by the values of consumption and investment respectively in 1964-I as

$$x_0 = \begin{bmatrix} 387.9 \\ 85.3 \end{bmatrix} \tag{57}$$

The criterion for the model is of the form

$$J = L_N(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k) \tag{58}$$

with

$$L_N(x_N) = \frac{1}{2}(x_N - \tilde{x}_N)'\hat{W}_N(x_N - \tilde{x}_N) \tag{59}$$

$$L_(x_k, u_k) = \frac{1}{2}[(x_k - \tilde{x}_k)'\hat{W}_k(x_k - \tilde{x}_k + (u_k - \tilde{u}_k)'\hat{\Lambda}_k(u_k - \tilde{u}_k)] \tag{60}$$

This is called a *tracking function* since it is minimized by having the optimal state and control vectors x_k and u_k track as closely as possible the desired state and control vectors $\tilde{x}_k$ and $\tilde{u}_k$. So the decision maker chooses the optimal time paths for the desired states and controls and then solves the model to compute the optimal controls, which come as close as possible to these desired paths while satisfying the dynamic relationships in the system equations (17).

Compare equations (59)-(60) to the criterion that was discussed in Section 4, that is, equations (14)-(16):

$$L_N(x_N) = \frac{1}{2}x_N'W_Nx_N + w_N'x_N \tag{61}$$

$$L_(x_k, u_k) = \frac{1}{2}x_k'W_kx_k + w_k'x_k + x_k'F_ku_k + \frac{1}{2}u_k'\Lambda_ku_k + \lambda_k'u_k \tag{62}$$

Equations (59)-(60) and (61)-(62) are similar since they are both quadratic functions. In fact equation (58) can be transformed to the form of equation (14) by expanding the quadratic terms. This results in the following relationship between the matrices of equations (59) and (60):

$$W_k = \hat{W}_k \tag{63}$$

$$w_k = -\hat{W}_k\tilde{x}_k \tag{64}$$

$$F = 0 \tag{65}$$

$$\Lambda_k = \hat{\Lambda}_k \tag{66}$$

$$\lambda_k = -\hat{\Lambda}_k\tilde{u}_k \tag{67}$$

In the **MATLAB** statement we input the data using the matrices and vectors in the tracking function (58) and then compute the matrices and vectors for the quadratic form (14) that was used to derive the algorithm that is implemented in the code.

The data for equation (58), which are taken from Kendrick (1982), are as follows:

$$\tilde{x}_k = (1.0075)^k \begin{bmatrix} 387.9 \\ 85.3 \end{bmatrix} \tag{68}$$

$$\tilde{u}_k = (1.0075)^k \begin{bmatrix} 110.4 \\ 147.17 \end{bmatrix} \tag{69}$$

These two equations indicate that the desired paths for both the states and the controls grow at approximately 3 percent per year or 0.75 percent per quarter over the time horizon covered by the model.

The priorities (penalty weights) in the objective function equation (58) are

$$\hat{W}_N = \begin{bmatrix} 6.25 & 0 \\ 0 & 100 \end{bmatrix} \quad \hat{W}_k = \begin{bmatrix} 0.0625 & 0 \\ 0 & 1 \end{bmatrix} \tag{70}$$

$$\hat{\Lambda}_k = \begin{bmatrix} 1 & 0 \\ 0 & 0.444 \end{bmatrix} \tag{71}$$

All of these priorities are the same (relative to the square of the size of the variables) except those for the terminal state variables in $\hat{W}_N$, where the priorities are 100 times as great. This is done to represent the fact that politicians usually care more about the state of the economy in the period just before an election than they do at other times.

This completes the statement of the model. Now we are ready to incorporate both the mathematics and the model into the **MATLAB** representation.

6 The MATLAB Representation

In this section we discuss the **MATLAB** representation a few statements at a time. The complete listing is in Appendix A. The code statement begins with the dimensions of the model. This version has seven time periods, two state variables, and two control variables:

```
t = 7; n = 2; m = 2;
```

One of the nice features of the **MATLAB** language in comparison to older languages such as Fortran and C is that the dimensioning of matrices is done automatically by the code. Therefore it is not necessary to use something like

```
Dimension A(2,2), B(2,2), c(2,1)
```

Rather one can just input the matrices A and B and the vector c as shown below and the **MATLAB** system takes care of the memory management:

```
a = [0.914 -0.016;
     0.097 0.424];
b = [0.305 0.424;
    -0.101 1.459];
c = [-59.437;
    -184.766];
```

Note that the semicolon is used to mark the end of a row in the matrix input.

We can also input the initial conditions, x_0, for the state and the base values for the desired states and controls as vectors. The base values for the states are called xtar to indicate that they represent target values for the states, x. Likewise, for the control targets, which are called utar,


```
x0 = [387.9;
      85.3];
xtar = [387.9;
        85.3];
utar = [110.4;
      147.17];
```

Finally we input the criterion function matrices:

```
w = [0.0625 0;
         0 1];
wn = [6.25 0;
      0 100];
f = [0 0;
    0 0];
lambda = [1 0;
     0 0.444];
```

Now all the data have been input and we are ready to start the matrix Riccati loop. In preparation for doing so we have to initialize the Riccati matrix K_{k+1} and vector p_{k+1}. These are called kold and pold to distinguish them from $K_k$and p_k, which are called knew and pnew, respectively. Since the Riccati loop proceeds from the last time period toward the first, period $k+1$ values are the old values and period k values are the new ones:

```
% The Riccati Loop
kold = wn; % Boundary condition
pold = -wn*xtar*(1.0075)^t; % Boundary condition
```

Recall from equations (19) and (20), which were the terminal conditions for the Riccati equations, that

$$K_N = W_N$$

$$p_N = w_N$$

Also remember that we have to use input data for the tracking function equation (58) and transform it for use in the quadratic form of equation (14). Thus we need to use the transformations from equations (63) and (6), which, for the terminal period, become

$$W_N = \hat{W}_N \tag{72}$$

$$w_N = -\hat{W}_N \tilde{x}_N \tag{73}$$

Substitution of equation (63) into equation (31) and equation (6) into equation (32) yields

$$K_N = \hat{W}_N \tag{74}$$

$$p_N = -\hat{W}_N \tilde{x}_N \tag{75}$$

In equation (68) the desired value for the state is based on the initial period target values grown over the time horizon covered by the model at a rate of 3 percent per year or 0.75 percent per quarter, so that

$$\tilde{x}_N = (1.0075)^N \tilde{x}_0 \tag{76}$$

Substitution of equation (76) into equation (75) then provides the relationship that is used in the **MATLAB** representation, namely

$$p_N = -\hat{W}_N \tilde{x}_0 (1.0075)^N \tag{77}$$

This is written in **MATLAB** as

```
pold = -wn*xtar*(1.0075)^t; % Boundary condition
```

where t is the number of time periods in the **MATLAB** representation. Next we need to compute and store the Riccati matrices K_k and for all time periods as we integrate backward from the terminal time period to the initial time period. Thus we have to store a series of (n, n) matrices. We do this by using a three-dimensional matrix with dimensions (n, n, t) for the Riccati matrices K_k and (n, t) for the Riccati vectors p_k. This is specified in **MATLAB** with the statements

```
kstore = zeros(n,n,t); % storage for dynamic Riccati matrices
pstore = zeros(n,t);   % storage for dynamic Riccati vectors
```

So kstore is our place to store all the Riccati matrices and pstore is used to store the Riccati vectors. These matrices are filled with zeroes as they are created and we then replace the zeros with the computed values.

Moreover, we have to create a place to store the optimal controls and states, and that is accomplished with the **MATLAB** statementsbelow. Here again the arrays are filled with zeroes as they are created and then are filled with the computed values later. This is done with the following statements:

```
u = zeros(m,t+1);
x = zeros(n,t+1);
```

The time dimension of these arrays is set to t+1 rather than to t to accommodate the fact that the x array must hold values for period zero as well as for the last period.

As a final step before we begin the backward recursion, we have to store the terminal values of kold in the matrix t of kstore and pold in column t of pstore:

```
kstore(:,:,t) = kold(:,:);
pstore(1:n,t) = pold;
```

This completes all of the setup required before we begin the backward recursion to compute the Riccati matrices and vectors. The loop itself is begun with the **MATLAB** statements

```
k = t-1;
while k > = 1;
```

Here the running index k is used for time periods. It is initialized to t-1 because we have already done the calculations for the terminal period t and are ready to do them for period t-1. Then the while command is used to indicate that the loop operation should continue as long as k $\geqslant$ 1. At the bottom of this loop we find the statements

```
k = k-1;
end; % End of the Riccati loop
```

that decrease k by one each time the calculation passes thorough the loop; and the end statement indicates the point to which the calculation jumps once the condition in the while statement no longer holds true.

The next step is to compute the desired paths for the state and control vectors, which are called utark and xtark for u target and x target, respectively, for all the time periods. Those variables are then used in turn in equations (57) and (60), that is,

$$w_k = -\hat{W}_k \tilde{x}_k$$

$$\lambda_k = -\hat{\Lambda}_k \tilde{u}_k$$

to compute w_k, which is called `wsmall` and λ_k, which is called `lambdas`:

```
utark = (1.0075^k).*utar; % Time dependent targets
xtark = (1.0075^k).*xtar;
wsmall = -w*xtark;
lambdas = -lambda*utark;
```

Now we are finally in position to compute the Riccati matrix using equation (29),

$$K_k = A'K_{k+1}A + W_k - \left[A'K_{k+1}B + F\right] \left[B'K_{k+1}B + \Lambda_k'\right]^{-1} \left[F' + B'K_{k+1}A\right]$$

The representation in **MATLAB** of this equation is

```
knew = a'*kold*a+w- ...
      (a'*kold*b+f)*inv(b'*kold*b+lambda')*(f'+b'*kold*a);
```

This is a good demonstration of the power of **MATLAB** to represent a complex expression in a form that is very close to the mathematical representation. The differences between the mathematical and **MATLAB** representations are quickly apparent. For the inverse of a matrix enclosed in parentheses mathematics uses $()^{-1}$, whereas **MATLAB** uses the function inv( ). **MATLAB** does not include the Greek alphabet so the mathematical symbol Λ is represented in **MATLAB** as lambda. Further, as was discussed earlier, the Riccati matrices K_k and K_{k+1} are represented as knew and kold, respectively.

Similarly the equation for the Riccati vectors in mathematics is

$$p_k = -\left[A'K_{k+1}B + F\right]\ \left[B'K_{k+1}B + \Lambda_k'\right]^{-1}$$

$$\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1}c + p_{k+1}\right) + w_k$$

and its **MATLAB** representation is

```
pnew=
-(a'*kold*b+f)*inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas)+...
  a'*(kold*c+pold)+wsmall;
```

Note that the ... notation is used in **MATLAB** to signal to the complier that the rest of the equation continues on the following line.

Having now used kold and pold, we need to transfer knew and pnew, respectively, to them for use in the next pass through the while loop. This is done with

```
kold = knew; % Setup next period
pold = pnew;
```

Then the Riccati matrix and vector for period k can be placed in the storage arrays that hold these matrices and vectors for all time periods, that is,

```
kstore(:,:,k) = knew(:,:);
pstore(1:n,k) = pnew;
```


Now we are at the bottom of the backward loop and, as promised earlier, this loop ends with a statement to decrease k by one and then to end the while loop:

```
k = k-1;
end;
```

No sooner do we finish to backward loop than it is time to start the forward loop with the following statements:

```
k = 0;
xold = x0;
sum = 0;
while k<= t-1;
      utark = (1.0075\^{}k).*utar;
      xtark = (1.0075\^{}k).*xtar;
      wsmall = -w*xtark;
      lambdas = -lambda*utark;
```

After `k` is set to zero the state vector is initialized using `x0` and then the desired paths for the states and controls are computed and used to calculate w_k and λ_k, which are represented in **MATLAB** as `wsmall` and `lambdas`. Note that the while loop this time uses a less than or equal to sign. At the bottom of this forward while loop we find the statements

```
k = k+1;
end;
```

that increment the `k` and provide the close for the while loop. Now we are at the stage where we want to make use of the Riccati matrices and vectors that we computed and stored in the backward loop:

```
kold(:,:) = kstore(:,:,k+1);
pold = pstore(1:n,k+1);
```

The elements for these matrices are pulled from storage in kstore and the vectors are pulled from pstore. Once the K_k matrix is available in `kold` it can be used in the computation of the feedback gain matrix, G_k, as described in equation (34),

$$G_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F_k + B'K_{k+1}A\right]$$

The **MATLAB** representation of this mathematical expression is

```
glarge = -inv(b'*kold*b+lambda')*(f'+b'*kold*a);
```

Here again we see how closely the mathematical and **MATLAB** representations parallel one another and how much this aids the user in being sure that the mathematics of the solution procedure is correctly mimicked in the computer code.

Similarly the mathematical expression for the feedback gain vector is

$$g_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right]$$

and the **MATLAB** representation is

```
gsmall = -inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas);
```

Once the feedback gain matrix and vector have been computed they can be used in the feedback rule (37) along with the state vector x_k to compute the control vector u_k:

$$u_k = G_k x_k + g_k$$

```
uopt = glarge*xold + gsmall;
```

Finally, the system equation (37) is used along with x_k and u_k to compute x_{k+1}:

$$x_{k+1} = Ax_k + Bu_k + c$$

```
xnew = a*xold + b*uopt + c;
```

Next we compute the portion of the cost terms in the criterion function that are incurred during period k. This is done with the mathematical expression

$$\frac{1}{2}\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \Lambda_k \left(u_k - \tilde{u}_k\right)$$

and the corresponding **MATLAB** expression

```
sum = sum + 0.5*(xold-xtark)'*w*(xold-xtark) + 0.5*(uopt-...
      utark)'*lambda*(uopt-utark);
```

The variable sum was set to zero at the top of the forward loop and is added to with each pass through the loop. Then the values of the state and control vectors are stored


```
x(1:n,k+1) = xold;
u(1:m,k+1 )= uopt;
```

and xold is set to xnew for the next pass through the forward loop, and the k index is incremented:

```
xold = xnew;
k = k+1;
```

The forward loop is then closed with the statement

```
end;
```

One more small bit of cleanup is required before we are through with the computations. We need to store the last state vector and add on the portion of the cost function for the terminal period. This is done using a portion of the mathematics from equation (58), that is,

$$\frac{1}{2}\left(x_N - \tilde{x}_N\right)' \hat{W}_N \left(x_N - \tilde{x}_N\right)$$

These two steps are represented in **MATLAB** as

```
x(1:n,t+1) = xold;
utark = (1.0075^k).*utar;
xtark = (1.0075^k).*xtar;
sum = sum + 0.5*(xold-xtark)'*wn*(xold-xtark);
```

The results for the optimal controls, states, and criterion value are then printed with the statements

```
u = u'; % The optimal control vector
u
x = x'; % The optimal state vector
x
Criterion = sum % The value of the criterion function
```

In the preceding we have shown almost all the lines of the **MATLAB** code for this solution procedure and model. While it is nice to learn to code a few lines at a time, it is not comfortable to view it that way after you have gained some familiarity with it. Therefore Appendix A includes the complete listing of the **MATLAB** input file.

7 Experiments

The macroeconomic model used as an example here is delightfully small while you are learning how to represent and solve it in **MATLAB**, but after you have used it for a bit you will discover that it has serious limitations. Therefore, we suggest that you modify it according to your own taste while you are experimenting with it.

One limitation is that the coefficients on the control variable are so small that there is not much latitude to alter the state variables' paths:

$$C_{k+1} = 0.914C_k - 0.016I_k + 0.305G_k + 0.424M_k - 59.437$$

$$I_{k+1} = 0.097C_k + .424I_k - 0.101G_k + 1.459M_k - 184.766$$

The coefficients of G_k in equations (57) and (58) are 0.305 and –0.101, respectively, so you might want to increase the magnitude of these coefficients in order to make the model more responsive to fiscal policy. Alternatively, if you want more clout for monetary policy you might increase the size of the coefficients on M.

Moreover, some of you will not like the fact that government spending *crowds*

out investment, so you may want to change the coefficient on government expenditure in the investment equation from negative to positive.

Econometricians will be somewhat dismayed that we are suggesting that coefficients be altered since these are, after all, estimated from data and should not be changed arbitrarily. While we agree that one should be careful about empirical work, the spirit here is to learn about the dynamic response of the model. The estimation of this model on data from different time periods does yield different parameter estimates and small changes in specification also result in changes in the parameter values. Thus for purposes of this kind of experiment we encourage the user to try some parameter modifications in order to see how those change the optimal state and control variables.

Aside from these limitations, the model is reasonably good for becoming acquainted with the use of optimal control theory to determine policy in macroeconomics. For this purpose you are encouraged to alter either the desired paths of states and controls or the priorities W_k and Λ_k to see how this affects the results. For example, some of you will want to assign high priority to consumption and others will prefer to do so for investment. Some will want to change the desired path for government expenditures so that it declines rather than rises and then provide a high priority weight in the *(1,1)* element in Λ_k to ensure that this result is obtained. Some will want to ensure that the economy follows the desired paths in all periods and others will want instead to use high priorities only on the terminal period to be sure that the economy is in good shape just before the next election.

8 Further Readings

For a general treatment of dynamic programming methods and their applications to economics see Sargent (1987) and Adda and Cooper (2003). More advanced treatments can be found in Bertsekas (2005) and Stokey and Lucas (1989). For detailed derivations of the QLP, CE, and OLF procedures and for projection-updating mechanisms, see Kendrick (1981, 2002). For a book on econometric and financial analysis with GAUSS, a language similar to **MATLAB** that is widely used in econometrics, see Lin (2001).

References

Abel, A. B.: 1975, A comparison of three control algorithms to the monetarist-fiscalist debate, *Annals of Economic and Social Measurement* **4**, 239–252.

Adda, J. and Cooper, R.: 2003, *Dynamics Economics: Quantitative Methods and Applications*, MIT Press, Cambridge, Massachusetts.

Bellman, R. E.: 1957, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey.

Bertsekas, D. P.: 2005, *Dynamic programming and optimal control*, Vol. 1, 3rd edn, Athena Scientific, Boston, USA.

Chow, G. C.: 1967, Multiplier, Accelerator, and Liquidity Preference in the Determination of National Income in the United States, *Review of Economics and Statistics* **49**, 1–15.

Kendrick, D. A.: 1981, *Stochastic control for economic models*, first edn, McGraw-Hill Book Company, New York, New York, USA. See also Kendrick (2002).

Kendrick, D. A.: 1982, Caution and probing in a macroeconomic model, *Journal of Economic Dynamics and Control* **4**, 149–170.

Kendrick, D. A.: 2002, Stochastic control for economic models. 2nd edition available at url: http://www.eco.utexas.edu/faculty/Kendrick.

Lin, K.: 2001, *Computational Econometrics: Gauss Programming for Econometricians and Financial Analysts*, Etext, Los Angeles, California, USA. Avialable at url: http://www.etext.net.

Sargent, T. J.: 1987, *Dynamic Macroeconomic Theory*, Harvard University Press, Cambridge, Massachussets, USA.

Stockey, N. L. and Lucas, R. E.: 1989, *Recursive methods in economic dynamics*, Harvard University Press, Cambridge, Massachussets.

Appendix

A MATLAB Representation of the Abel Model

```
 Title: Quadratic-Linear Tracking problem for Abel
% Program name: qlpabel.m

% Based on the Chapter 4 of Stochastic Control for Economic Models
% example by David Kendrick.
% GAUSS version by Hans Amman, modified to MATLAB by Huber Salas
% with subsequent changes by Miwa Hattori and David Kendrick
% to implement a deterministic,
% two-control version of the Abel (1975) model (Jan 2005)

% Computes the optimal cost-to-go, control and state vectors.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Preliminaries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t = 7; n = 2; m = 2;


a = [0.914 -0.016;
    0.097  0.424];
b = [0.305 0.424;
    -0.101 1.459];
c = [-59.437;
    -184.766];
x0 = [387.9;
       85.3];
xtar = [387.9;
       85.3];
utar = [110.4;
       147.17];
w = [0.0625 0;
         0  1];
wn = [6.25 0;
       0 100];
f = [0 0;
     0 0];
lambda = [1    0;
          0 0.444];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   The Riccati Loop
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
kold = wn;                      %   Boundary condition
pold = -wn*xtar*(1.0075)^t;     %   Boundary condition

kstore = zeros(n,n,t);      % storage for dynamic Riccati matrices
pstore = zeros(n,t);        % storage for dynamic Riccati vectors

u = zeros(m,t+1);
x = zeros(n,t+1);

kstore(:,:,t) = kold(:,:);
pstore(1:n,t) = pold;


k = t-1;
while k >= 1;
  utark = (1.0075^k).*utar;   %   Time dependent targets
  xtark = (1.0075^k).*xtar;
  wsmall = -w*xtark;
  lambdas = -lambda*utark;

  knew = a'*kold*a+w-(a'*kold*b+f)*inv(b'*kold*b+lambda')*(f'+b'*kold*a);
                              %   Computing the Riccati matrices
  pnew = -(a'*kold*b+f)*inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas)+...
             a'*(kold*c+pold)+wsmall;
                              %   Computing the tracking equation

  kold = knew;                  %   Setup next period
  pold = pnew;
  kstore(:,:,k) = knew(:,:);
  pstore(1:n,k) = pnew;
  k = k-1;
end;                          %   End of the Riccati loop


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   The Forward loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k = 0;
xold = x0;
sum = 0;

while k <= t-1;
  utark = (1.0075^k).*utar;
  xtark = (1.0075^k).*xtar;

  wsmall = -w*xtark;
  lambdas = -lambda*utark;
```

```
  kold(:,:) = kstore(:,:,k+1);
  pold = pstore(1:n,k+1);

  glarge = -inv(b'*kold*b+lambda')*(f'+b'*kold*a);
  gsmall = -inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas);

  uopt = glarge*xold+gsmall;
  xnew = a*xold+b*uopt+c;
  sum = sum+0.5*(xold-xtark)'*w*(xold-xtark)+0.5*(uopt-utark)'*lambda*(uopt-utark);

  x(1:n,k+1) = xold;
  u(1:m,k+1) = uopt;
  xold = xnew;
  k = k+1;

end;                          %   End of the forward loop


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   The Last Period
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x(1:n,t+1) = xold;
utark = (1.0075^k).*utar;
xtark = (1.0075^k).*xtar;
sum = sum+0.5*(xold-xtark)'*wn*(xold-xtark);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Print the solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u = u';                       %   The optimal control vector
u

x = x';                       %   The optimal state vector
x

Criterion = sum               %   The value of the criterion function
```