COMPUTATIONAL ECONOMICS

Dynamic Programming with Value Function Iteration

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

1 Introduction

In the previous chapter the value function from dynamic programming was used to solve a growth model by first solving the model backward in time from the terminal condition to the initial condition before solving it forward in time over the same time frame. In this chapter, we use the same value function and we iterate on it again but we focus not on the entire time frame covered by the model but rather only on the steady state. So the iterations in this chapter begin with an initial guess about the value function value and employ the value function to get a better approximation. These iterations then continue until the change in the value function from one iteration to the next is sufficiently small that a satisfactory level of convergence is obtained.

If it seems somewhat simpler, and perhaps less useful, to solve the model only for the steady state rather than for the entire time frame, that is the case. However, much of economic research on dynamic economic models is focused only on steady state solutions and this chapter can be used to open a window on this methodology.

As in the previous chapter, we continue here to use a discrete state space. Therefore, much of the development of the model here is easy to follow after sorting through the use of grids, meshes and paths in the previous chapter.

The previous chapter was confined to a deterministic model. In this chapter we will begin with a deterministic version of the steady state model but will then progress to a stochastic version in which there is uncertainty about one of the coefficients in the model.[1]

2 The Deterministic Model

The growth model used in the chapter is the same as the one used in the previous chapter with one exception – instead of covering a finite time horizon this model covers an infinite period and thus there is no terminal capital stock target. Therefore the criterion function is changed from

$$J = \sum_{t=0}^{N-1} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)} \tag{1}$$

to

$$J = \sum_{t=0}^{\infty} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)} \tag{2}$$

The initial condition is retained as

$$K_0 \quad \text{given} \tag{3}$$

but the terminal condition for capital is dropped. Thus in summary the model used in this chapter can be stated as find

$$(C_0, C_1, \cdots)$$

[1]In this chapter, as in the previous one, the programs were developed in large part by Firat Yaman while drawing on ideas and material from courses taught at The University of Texas by Dean Corbae and Russell Cooper and **MATLAB** programs developed for one of those courses by Pablo D'Erasmo.

to maximize

$$J = \sum_{t=0}^{\infty} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)} \tag{4}$$

subject to

$$K_{t+1} = K_t + \theta K_t^{\alpha} - C_t \tag{5}$$

$$K_0 \quad \text{given} \tag{6}$$

Also, the parameters used in this chapter are the same as those in the previous chapter with the exception that the initial capital stock is different and we no longer need a terminal capital stock parameter. Thus the parameters used in this chapter are

$$\alpha = 0.33 \quad \beta = 0.98 \quad \theta = 0.3 \quad \tau = 0.5 \quad K_0 = 0.1$$

In summary, in this chapter, the essential problem is to choose those levels of consumption over an infinite time horizon that strikes the right balance between consumption and investment. Lower consumption in any given period means less utility in that period but more investment and therefore larger capital stocks and more production. However, the difference between this chapter and the previous one is that we will be working to solve only for the steady state solution of the model.

3 The Methodology for the Deterministic Model

As was discussed above, we will use the same value function in this chapter as in the previous chapter and will once again draw on the dynamic programming approach outlined by Bertsekas (2005) Chapter 1 (see also Adda and Cooper (2003) Chapter 2 and Stokey and Lucas (1989)). The value function is

$$V_t(k_t) = \max_{k_{t+1}} \{u(c_t) + \beta V_{t+1}(k_{t+1})\} \tag{7}$$

We will use this value function in a recursive fashion but will not start at the terminal time and iterate to the initial time. Rather we will start with some initial guess about the value function and use equation (7) to get an improved estimate. We will then use that improved estimate in the value function again and get another estimate. So when V changes by less than a certain tolerance from one iteration to the next, convergence is declared and the iterations are halted.

Therefore, the program in this chapter includes only one loop, namely a backward loop similar to the one used in the previous chapter. However, the loop is not over time periods but rather over iterations in an attempt to reach convergence. Also, in this chapter, as in the previous one, our goal will be to obtain a map. The map will show what point to move to and we will be looking for that point where the best move is to itself and is thus the steady state.

4 The Program

The initialization of the program in this chapter is almost identical to the one in the previous chapter, namely

```
alpha = 0.33;
beta = 0.98;
theta = 0.3;
tau = 0.5;
delta=0.1;
k0 = .1;
kN = 2.1;
n = 20;
step = (kN-k0) / (n-1);
k_grid = [k0:step:kN]
```

We will display many sections of the code in the chapter; however, for a complete listing of the program see Appendix A. Note in the statements above that `k0` has a different value in this chapter. Also, `kN` in this chapter will no longer be a target for the terminal capital stock but rather simply an upper bound for the capital stock grid in the capital grid, `k_grid`. Also, instead of the small grid of five points that was used for n in the previous chapter we use 20 grid points here from the outset. We make use of the **MATLAB** convention in the `k_grid` statement that the absence of a semicolon will cause the result in that line of the code to be printed. The first five points and the last two points in `k_grid` are shown below

$$0.1000 \quad 0.2053 \quad 0.3105 \quad 0.4158 \quad 0.5211 \quad \ldots 1.9947 \quad 2.1000$$

In the first runs of the program it is advisable to set the iteration limit, iterlim, very small as is done below. This limits the output file size and makes it easier to understand the program

```
iterlim = 5;
tolera = 0.000001 ;
```

The tolerance, `tolera`, is set very small to assure convergence. If the model does not converge quickly enough it may be necessary to use a larger tolerance. The program can be divided into two segments:

- the definition of the grid and mesh for the capital stocks and other variables
- the iteration loop in the value function.

4.1 Definition of the Grid and the Mesh

The initialization of the grids and meshes for output, capital stocks in the current and next time period, investment and consumption are the same here as in the program used in the previous chapter, namely

```
y_grid = theta*(k_grid.^alpha);

%vector of ones
aux = ones(n,1);

%current capital stocks vary along a row in kt_mesh
%future capital stocks vary along a column in kt1_mesh
%i_mesh is the corresponding mesh for investment
%since i = k(t+1) - (1-delta) * k(t)
```

```
kt_mesh = aux*k_grid;
kt1_mesh = (aux*k_grid)';
i_mesh = kt1_mesh - (1-delta) * kt_mesh;

y_mesh = aux*y_grid;

%then consumption is c(t)= y(t)-i(t)
%to create the consumption mesh array

c_mesh = y_mesh - i_mesh;

%replace negative consumption with zero

cc = find(c_mesh<0);
c_mesh(cc) = 0;
clear cc;
```

Also, the mesh over utility is defined here the same way as in the program of the previous chapter, namely

```
%utility; to avoid negative and zero consumption we assign the
%lowest possible utility to entries with zero consumption

u_mesh = (1/(1-tau))*(c_mesh.^(1-tau));
uu = find(u_mesh == 0);
%u_mesh(uu) = -realmax
u_mesh(uu) = -10;
```

Here also we comment out the statement with the use of realmax and replace it with one that uses a small negative number, namely -10. However the user will probably want to switch this after first learning to use the program. The twenty by twenty `u_mesh` matrix is too large to show here so we print below only the first five rows and columns in the upper left corner of the matrix.

```
>> u_mesh(1:5,1:5)

ans =

    0.7220    1.0250    1.2384    1.4125    1.5632
    0.3166    0.7934    1.0548    1.2546    1.4221
  -10.0000    0.4566    0.8316    1.0738    1.2655
  -10.0000  -10.0000    0.5201    0.8555    1.0864
  -10.0000  -10.0000  -10.0000    0.5576    0.8714
```

This completes the preliminary work in the program and opens the door to the use of the value function iteration as in equation (7) above.

4.2 The Iteration Loop

Before beginning the loop itself it is necessary to initialize it. To do this we need first to obtain an expression for consumption and then use that to compute the initial value function. To obtain an expression for consumption we begin with equation (5) above, i.e.

$$K_{t+1} = K_t + \theta K_t^{\alpha} - C_t \tag{8}$$

and substitute output, Y_t, for the production function, θK_t^{α}. Then solve equation (8) for consumption to obtain

$$C_t = Y_t - (K_{t+1} - K_t) = Y_t - I_t \tag{9}$$

Then set investment I_t equal to zero to obtain

$$C_t = Y_t \tag{10}$$

This is an unrealistic assumption but is good enough to provide an initial guess for the value function in the value function iteration. The corresponding **MATLAB** statement for this with the grids can be written

```
 clast = ygrid'
```

where `clast` is a column vector of consumption for each of the grid points. We show below only the first *5* points in the *20* element vector

```
y_grid(1:5)'

ans =

    0.1403
    0.1779
    0.2039
    0.2246
    0.2419
```

Then we can compute the corresponding value function using the utility function for each time period

$$U\left(C_t\right) = \frac{1}{(1-\tau)} C_t^{(1-\tau)} \tag{11}$$

with the statement

```
 Vlast = (1/(1 - tau)) * (clast .^(1-tau))
```

The top five elements in the twenty element `Vlast` column vector are shown below

```
Vlast(1:5)

ans =

    0.7492
    0.8436
    0.9032
    0.9478
    0.9837
```

Also in the iteration loop we will make use of the `Vlast_mesh` which is computed by using the aux vector of n ones with the statement

```
Vlast_mesh = Vlast * (aux')
```

`Vlast` is a column vector and aux is also a column vector but is transposed to a row vector. Therefore, `Vlast_mesh` is a 20 x 20 matrix. We show below only the 5×5 elements at the top left corner of that matrix.

```
 Vlast_mesh(1:5,1:5)

ans =

    0.7492    0.7492    0.7492    0.7492    0.7492
    0.8436    0.8436    0.8436    0.8436    0.8436
    0.9032    0.9032    0.9032    0.9032    0.9032
    0.9478    0.9478    0.9478    0.9478    0.9478
    0.9837    0.9837    0.9837    0.9837    0.9837
```

With this initialization in hand the iteration loop can be written as

```
tol = 1
iter = 0
while ( (tol > tolera) & (iter < iterlim) )

    iter = iter + 1
    V = u_mesh + beta*(Vlast*(aux'));
    [VV,I] = max(V,[],1);
    VV_diff = (VV - Vlast')'
    tol=max(abs(VV-Vlast'))
    Vlast = VV';
end
```

We begin by setting the tolerance, `tol`, equal to 1 and the iteration counter, `iter`, equal to 0. Then a while statement is used to continue the loop so long as the tolerance is greater than the `tolera` and the iteration count is less than the iteration limit, `iterlim`. In each pass through the loop the iteration counter is augmented with the statement

```
    iter = iter + 1
```

This is followed by the value function itself. Consider the portion of this function in braces from equation (7), i.e.

$$u\left(c_t\right) + \beta V_{t+1}\left(k_{t+1}\right) \tag{12}$$

and consider first only the $V_{t+1}\left(k_{t+1}\right)$ portion of equation (12). This is the `Vlast_mesh` element that was computed above and is a 20×20 matrix of which only the upper left hand corner 5×5 elements are shown above. Then the program statement for the full mathematics in equation (12) is written

```
      V = u_mesh + beta*(Vlast*(aux'));
```

Next from equation (7) the maximization is

$$V_t\left(k_t\right) = \max_{k_{t+1}} \left\{u\left(c_t\right) + \beta V_{t+1}\left(k_{t+1}\right)\right\} \tag{13}$$

This can be written **MATLAB** using the max function the same as was described in the previous chapter with the optimal values being put in the array VV and the index for these values begin put in the array `I`. The next line in the code, namely

```
     VV_diff = (VV - Vlast')'
```

is used only to display and print the difference between the value function grids from one iteration to the next. This `VV_diff` vector is shown below for the first pass through the iteration loop.

```
 >> VV_diff

VV_diff =

    0.7136
    0.7136
    0.7453
    0.7639
    0.7749
    0.7810
    0.8016
    0.8166
    0.8185
    0.8215
    0.8407
    0.8555
    0.8591
    0.8667
    0.8758
    0.8862
    0.8945
    0.9015
    0.9116
    0.9189
```

One can see here that the change in the value function grid between the first and second iteration is substantial for all the grid points. Of course as the iterations precede these differences will become much smaller. Then the absolute value of this difference between the current and previous values for each of the grid points is computed with the statement

```
 abs ( VV -- Vlast')
```

Also for the tolerance calculation we want the worst of these differences so the `max` function is used again in the statement to compute `tol`, namely

```
 tol = max (abs (VV -- Vlast'))
```

and this value on the first pass through the iteration loop is

```
  2.4647
```

as can be confirmed by looking at the `VV_diff` vector above. Finally, at the end of the loop the current value is used to update the value for the next iteration with the statement

```
  Vlast = VV'
```

So in summary, the iteration loop continues until the value function converges and when convergence is obtained it leaves the index for the look up grid in the array `I`. The transpose of that vector is shown below

```
  I'

ans =

     2
     2
     3
     4
     5
```

```
    6
    6
    7
    8
    8
    9
   10
   10
   11
   12
   13
   13
   14
   15
   15
```

This vector is like the *look-up* map which was discussed in the previous chapter. It indicates that if one is at the first grid point the best move to make is to second grid point. If one is at the fifth grid point, counting down from the top of the vector, the best move is to the fifth grid point. Finally, if one is at the twentieth grid point, at the bottom of the vector, the best move to make is to the fifteenth grid point. This is illustrated nicely with the graph which is generated with the final statements in the program, namely

```
index=1:n;
plot(index,index,'-',index,I,'--');
title('Path of capital');
legend('steady state','decision rule');
```

The result of this plot is shown in Figure 1 below. So the steady state occurs when the decision rule indicates that the best move to make is the current grid point. In this case, if convergence has been obtained with only 20 grid points and only five iterations, all of the grid points between 2 and 6 would have been steady state points.

In contrast, if we change the number of grid points, n, in the program from 20 to 100 and raise the iteration limit, iterlim, to 1000 while also placing semicolons at the end of all the statements in the iteration loop so as to decrease the amount of output, the program generates the plot shown in Figure 2 below. This shows the results after 677 iterations when the tolerance falls below 0.000001 and convergence is obtained.

Though it is difficult to tell exactly from the graph, it appears that the steady state is at about the 34th grid point in `k_grid`. This corresponds to the 34-th grid point in `k_grid` which would be

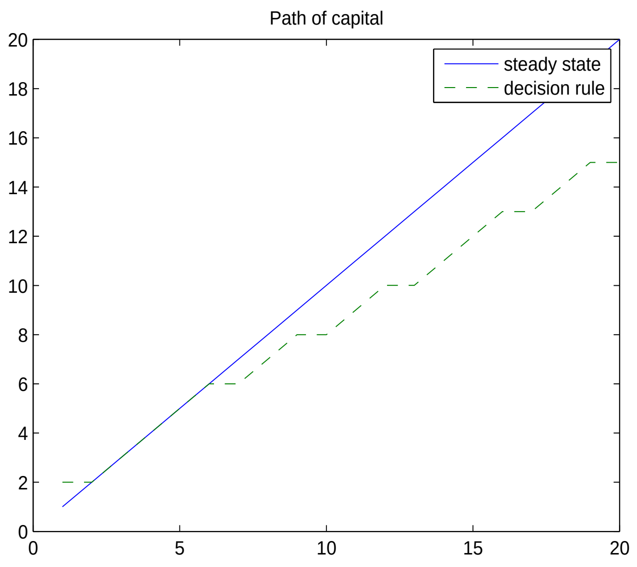
$$\begin{aligned} k0 + 33\left(\tfrac{kN-k0}{n}\right) &= 0.1 + 33\left(\tfrac{2.1-0.1}{99}\right) \\ &= 0.1 + 33\,(0.0202) \\ &= 0.1 + 0.666 \\ &= 0.766 \end{aligned}$$

So a capital stock of 0.766 would be steady state solution for this growth model.

5 The Stochastic Model

The growth model used in the chapter is the same as the infinite horizon model used earlier in this chapter with one exception – one of the coefficients of the model is now treated as stochastic rather than as deterministic. The stochastic parameter is in the production function, so that instead of the production function

Figure 1: The Steady State and the Decision Rule with n = 20 after Five Iterations



$$Y_t = \theta K_t^\alpha \tag{14}$$

we introduce a new random variable, z_t and write the production function as

$$Y_t = z_t \theta K_t^\alpha \tag{15}$$

It is useful to think of this production function as having a stochastic technology parameter, γ_t, so that the function is written as

$$Y_t = \gamma_t K_t^\alpha \tag{16}$$

where

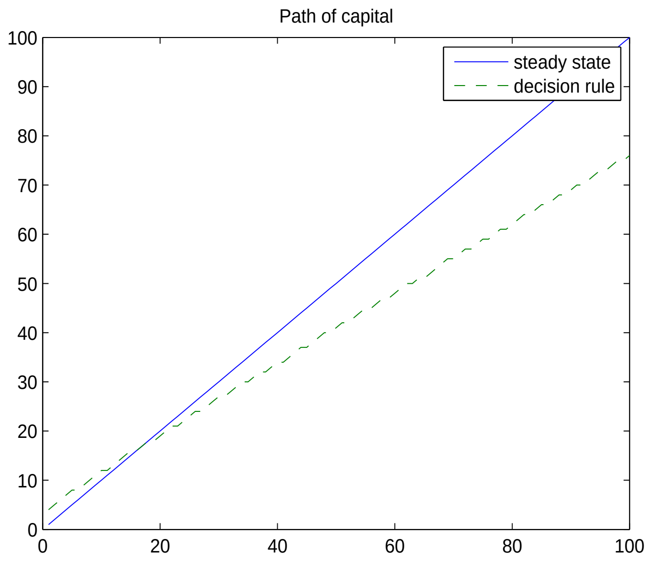
$$\gamma_t = z_t \theta \tag{17}$$

with γ_t = a stochastic technology parameter. However, in this chapter we will keep the separate z_t and θ instead of the combined γ_t. Unlike the development in Chapter 18 where the uncertainty parameters were modeled with continuous distributions in this chapter we will use discrete probability distributions. Thus the stochastic parameter, z_t, will be characterized as having only two values, namely *low* and *high*. Also the probabilities we use here will not be static in the sense of the probability that z_t takes on its low or high value but rather that in the dynamic sense of transition, for example the probability of the transition from low to high. For example we will use the two values

$$\begin{aligned} z^{low} &= 0.95 \\ z^{high} &= 1.05 \end{aligned} \tag{18}$$

and a Markov probability matrix that gives the probabilities of making transitions between these two values in each period. This matrix for the model here is

Figure 2: The Steady State and the Decision Rule with n = 100 after Convergence



$$P = \begin{bmatrix} 0.5 & 0.5 \\ 0.1 & 0.9 \end{bmatrix} \tag{19}$$

Thus the P transition probability matrix for this model may be thought of as

$$P = \begin{bmatrix} low, low & low, high \\ high, low & high, high \end{bmatrix} \tag{20}$$

So, for example, the probability that the z_t parameter will make the transition from high this period to low next period is 0.1 and the probability that it will remain high next period if it is high this period is *0.9*. Thus for the model used in this section with a technology parameter, θ, equal to 0.3 and with z^{low} and z^{high} as given in equation (18) above one can think of a stochastic γ_t technology parameter that moves randomly between the values of $0.95 \times 0.30 = 0.285$ and $1.05 \times 0.3 = 0.315$. So in *low* years the economy will have an effective technology parameter of 0.285 and in *high* years a technology parameter of 0.315.

Thus in summary the model used in this section can be stated as find

$$(C_0, C_1, \cdots)$$

to maximize the expected value of

$$J = \sum_{t=0}^{\infty} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)} \tag{21}$$

subject to

$$K_{t+1} = K_t + z_t \theta K_t^{\alpha} - C_t \tag{22}$$

$$K_0 \quad \text{given} \tag{23}$$

and with P the transition probability matrix for the z_t parameters.

6 The Methodology for the Stochastic Model

The value function is for the stochastic model is computed separately for the low case and then for the high case with the probabilities used to weight each case in the previous iteration to get the corresponding value for the next period, i.e.

$$\begin{aligned} V_t^l(k_t) &= \max_{k_{t+1}} \left\{ u^l(c_t) + \beta \left(p^{ll} V_t^l(k_{t+1}) + p^{lh} V_t^h(k_{t+1}) \right) \right\} \\ V_t^h(k_t) &= \max_{k_{t+1}} \left\{ u^h(c_t) + \beta \left(p^{hl} V_t^l(k_{t+1}) + p^{hh} V_t^h(k_{t+1}) \right) \right\} \end{aligned} \tag{24}$$

where

u^l	$=$	utility function in the low case
u^h	$=$	utility function in the high case
p^{ll}	$=$	probability of moving from low to low
p^{lh}	$=$	probability of moving from low to high
p^{hl}	$=$	probability of moving from high to low
p^{hh}	$=$	probability of moving from high to high
V_t^l	$=$	value function in the low case
V_t^h	$=$	value function in the high case

In order to facilitate this use of the value function it is necessary to compute the utility in the low case and in the high case. This turn requires the computation of output, investment and consumption in each of the cases. Therefore the methodology for the stochastic model requires the computation of most of the variables for the low and then the high case and then the use of the appropriate probability weighs in the value function calculations in equation (24).

7 The Program for the Stochastic Model

The program for the stochastic model is almost identical to the program for the deterministic infinite horizon model except that the calculations are repeated for both the low and then the high case. For example, in the initialization of the parameters near the top of the code one finds the statements

```
z_low = 0.90;
z_high = 1.10;
```

that set the value of z for each of the cases. This is followed by the lines of code which set the elements of the Markov probability transition matrix

```
%probability of going from low to low, low to high
P(1,1)=0.5;  P(1,2)=1-P(1,1);

%probability of going from high to low, high to high
P(2,1)=0.1;  P(2,2)=1-P(2,1);
```

It is not necessary to compute the capital grid, `k_grid`, for both the low and high cases since the stochastic element does not enter the model until output is computed from the capital stock. However, it is necessary to compute output for both cases and this is done with the statements

```
y_grid_l = z_low*theta*(k_grid.^alpha);
y_grid_h = z_high*theta*(k_grid.^alpha);
```

Likewise it is not necessary to compute investment for both cases since it depends only on the capital stocks; however it is necessary to compute consumption for both cases since it depends on the stochastic element, output. Thus the calculation for the consumption mesh for each case depend on output mesh for each case but on investment with no separation for the two cases

```
c_mesh_l = y_mesh_l - i_mesh;
c_mesh_h = y_mesh_h - i_mesh;
```

Similarly the calculations for utility are done for both cases with the statements

```
u_mesh_l = (1/(1-tau))*(c_mesh_l.^(1-tau));
u_mesh_h = (1/(1-tau))*(c_mesh_h.^(1-tau));
```

Just as in the infinite horizon model earlier in this chapter the clast calculation is done to get the V_last variable that is used in turn as the starting point for the iterations in the while loop, so it is necessary here to repeat these calculations for both cases, i.e.

```
clast_l = y_grid_l';
Vlast_l = (1/(1-tau))*(clast_l.^(1-tau));
clast_h = y_grid_h';
Vlast_h = (1/(1-tau))*(clast_h.^(1-tau));
```

Notice that for the first iteration investment is set to zero so that consumption equals to output. This is an unrealistic assumption but is good enough to provide a starting point for the iterations that will converge. Then, following the value function mathematics from equation (24) above

$$\begin{aligned} V_t^l\left(k_t\right) &= \max_{k_{t+1}}\left\{u^l\left(c_t\right)+\beta\left(p^{ll}V_t^l\left(k_{t+1}\right)+p^{lh}V_t^h\left(k_{t+1}\right)\right)\right\} \\ V_t^h\left(k_t\right) &= \max_{k_{t+1}}\left\{u^h\left(c_t\right)+\beta\left(p^{hl}V_t^l\left(k_{t+1}\right)+p^{hh}V_t^h\left(k_{t+1}\right)\right)\right\} \end{aligned} \tag{25}$$

the iteration in each pass through the while loop can be written

```
  V_l=u_mesh_l+beta*(P(1,1)*Vlast_l*(aux')+P(1,2)*Vlast_h*(aux'));
  V_h=u_mesh_h+beta*(P(2,1)*Vlast_l*(aux')+P(2,2)*Vlast_h*(aux'));
```

and the maximization is done with the statements

```
  [VV_l,I_l] = max(V_l,[],1);
  [VV_h,I_h] = max(V_h,[],1);
```

The tolerance, tol, on each pass is then calculated with the statements

```
  tol_l=max(abs(VV_l-Vlast_l'));
  tol_h=max(abs(VV_h-Vlast_h'));
  tol=max(tol_l,tol_h);
```

that gets `tol_l` for the low case first and `tol_h` for the high case in order to then compute tol as the maximum over the two cases. The final step in the while loop is then to update the value functions `Vlast_l` and `Vlast_h`, i.e.

```
    Vlast_l = VV_l';
    Vlast_h = VV_h';
```

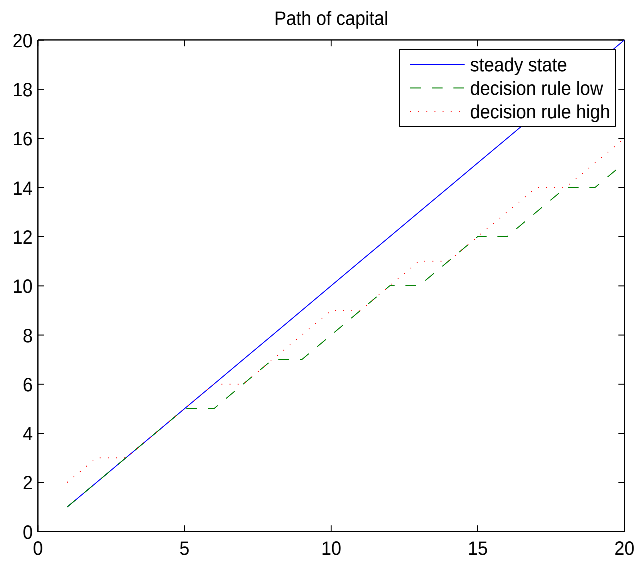
with the optimum before beginning the next pass through the loop. After exit from the while loop the results are printed with the statements

```
iter
tol
Vlast_l
Vlast_h
I_l
I_h
```

and key look-up map results in `I_l` and `I_h` are plotted with the statements

```
index=1:n;
plot(index,index,'-',index,I_l,'--',index,I_h,':');
title('Path of capital');
legend('steady state','decision rule low','decision rule high');
```

Figure 3: Look-up Map Results with 20 Grid Points and Iteration Limit *5*



Of course we are not really interested in the results for the case with only 20 grid points and an iteration limit of 5. Therefore the program is run again by changing n from *20* to *100*, i.e.

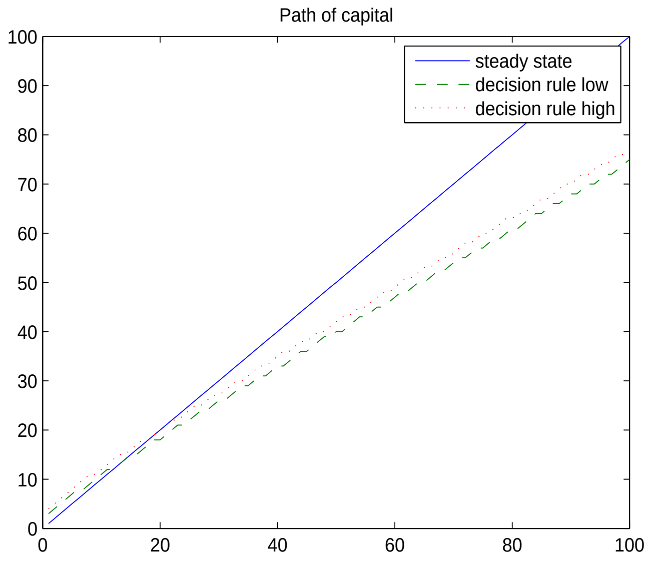
```
n = 100;
```

and changing the limit on the number of iterations from *5* to *1000*, i.e.

```
iterlim = 1000;
```

This yields the look-up map results shown in Figure 6.2 below

Figure 4: Look-up Map Results with 100 Grid Points and Iteration Limit of *1000*



8 Summary

In this chapter dynamic programming with discrete state space is used to solve first a deterministic and then a stochastic growth model. In both cases the models have infinite horizons and are solved for the steady state solution. The models are first solved with a small number of grid points and with a limited number of iterations in order to facilitate learning by producing output files that are small enough to decipher with some ease. In both of these cases convergence is not obtained before the iteration limit is reached. Then each of the models is solved again with a larger number of grid points and with an iteration limit that is large enough to permit the models to reach convergence.

References

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

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

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

Appendix

A Running GAMS

Appendix A provides the details for running the GAMS software on a PC. In order to use GAMS with other input files, substitute the appropriate file name for trnsport.gms in the following. For help and information about obtaining GAMS go to the GAMS Development Corporation web site at

`http://www.gams.com`

There is a student version of the software that can be downloaded and that can solve all or almost all of the models used in this book. It the model is too large, usually a small change in the number of time periods or some other set is sufficient to reduce the size so that it runs on the student version.

- Go to the book web site at

  `http://www.eco.utexas.edu/compeco`

  and to the *Input Files for Chapters in the Book* section of the web site. Right click on the `trnsport.gms` file name and select the `Save Target As` ... option in order to save the file in your preferred directory.

- Choose Programs from the Start menu and then choose GAMS and gamside. Choose Open from the File menu, navigate to the trnsport.gms file, and open it for editing. Note in the complete GAMS statement of the model that, as is the usual case in GAMS, the model is defined in steps:

  1. first the sets
  2. then the parameters
  3. then the variables
  4. then the equations
  5. and finally the model and solve statements.

- Solve the model by choosing Run from the File menu and then check the solution log to be sure that you have

  ```
  SOLVER STATUS: 1 NORMAL COMPLETION
  ```

  and

  ```
  MODEL STATUS: 1 OPTIMAL
  ```

  Then close the log file window.

- Click on the trnsport.lst file window and scroll through this listing file to see the solution. Note that the `*.lst` file extension used here is an abbreviation for a *listing* of the output file. Note that the GAMS output has the following structure:

  Echo Print shows a listing of the input file with the line numbers added.

  Error Messages In the case of errors in the input file, they will be signaled by GAMS with `****` on the leftmost part of the corresponding line of input where the error was found, and with `$number` just below the part of the line of input where the error is located, where `number` contains a specific error code. Then, at the end of the list of the input file, GAMS displays the explanation of each of the error codes found.

  `Equation Listing` Shows each equation of the model with the corresponding values for sets, scalars, and parameters.

  `Column Listing` Shows a list of the equations' individual coefficients classified by columns.

  `Model Statistics` Shows information such as model number of equations, number of variables, and so on.

  `Solve Summary` Shows information such as solver and model status at the end of the GAMS run, and so on.

  `Solution Listing` Shows the solution values for each equation and variable in the model. Each solution value is listed with four pieces of information, where a dot `.` means a value of zero, `EPS` a value near zero, and `+INF` and `-INF` mean plus or minus infinite, respectively:

  `LOWER (the lower bound)`

  `LEVEL (the solution level value)`

  `UPPER (the upper bound)`

  `MARGINAL` (the solution marginal value; for linear or nonlinear programming problems, it corresponds to the increase in the objective value owing to a unit increase in the corresponding constraint)

  `Report Summary` Shows the count of rows or columns that are infeasible, nonoptimal, or unbounded.