Lecture Notes 7

Dynamic Programming

In these notes, we will deal with a fundamental tool of dynamic macroeconomics: dynamic programming. Dynamic programming is a very convenient way of writing a large set of dynamic problems in economic analysis as most of the properties of this tool are now well established and understood.[1] In these lectures, we will not deal with all the theoretical properties attached to this tool, but will rather give some recipes to solve economic problems using the tool of dynamic programming. In order to understand the problem, we will first deal with deterministic models, before extending the analysis to stochastic ones. However, we shall give some preliminary definitions and theorems that justify all the approach

7.1 The Bellman equation and associated theorems

7.1.1 A heuristic derivation of the Bellman equation

Let us consider the case of an agent that has to decide on the path of a set of control variables, $\{y_t\}_{t=0}^{\infty}$ in order to maximize the discounted sum of its future payoffs, $u(y_t, x_t)$ where x_t is state variables assumed to evolve according to

$$x_{t+1} = h(x_t, y_t),\ x_0 \text{ given.}$$

[1] For a mathematical exposition of the problem see Bertsekas [1976], for a more economic approach see Lucas, Stokey and Prescott [1989].

We finally make the assumption that the model is markovian.

The optimal value our agent can derive from this maximization process is given by the *value function*

$$V(x_t) = \max_{\{y_{t+s} \in \mathscr{D}(x_{t+s})\}_{s=0}^{\infty}} \sum_{s=0}^{\infty} \beta^s u(y_{t+s}, x_{t+s}) \tag{7.1}$$

where $\mathscr{D}$ is the set of all feasible decisions for the variables of choice. Note that the value function is a function of the state variable only, as since the model is markovian, only the past is necessary to take decisions, such that all the path can be predicted once the state variable is observed. Therefore, the value in t is only a function of x_t. (7.1) may now be rewritten as

$$V(x_t) = \max_{\{y_t \in \mathscr{D}(x_t), \{y_{t+s} \in \mathscr{D}(x_{t+s})\}_{s=1}^{\infty}} u(y_t, x_t) + \sum_{s=1}^{\infty} \beta^s u(y_{t+s}, x_{t+s}) \tag{7.2}$$

making the change of variable $k = s - 1$, (7.2) rewrites

$$V(x_t) = \max_{\{y_t \in \mathscr{D}(x_t), \{y_{t+1+k} \in \mathscr{D}(x_{t+1+k})\}_{k=0}^{\infty}} u(y_t, x_t) + \sum_{k=0}^{\infty} \beta^{k+1} u(y_{t+1+k}, x_{t+1+k})$$

or

$$V(x_t) = \max_{\{y_t \in \mathscr{D}(x_t)} u(y_t, x_t) + \beta \max_{\{y_{t+1+k} \in \mathscr{D}(x_{t+1+k})\}_{k=0}^{\infty}} \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k}) \tag{7.3}$$

Note that, by definition, we have

$$V(x_{t+1}) = \max_{\{y_{t+1+k} \in \mathscr{D}(x_{t+1+k})\}_{k=0}^{\infty}} \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k})$$

such that (7.3) rewrites as

$$V(x_t) = \max_{y_t \in \mathscr{D}(x_t)} u(y_t, x_t) + \beta V(x_{t+1}) \tag{7.4}$$

This is the so–called *Bellman equation* that lies at the core of the dynamic programming theory. With this equation are associated, in each and every period t, a set of *optimal policy functions* for y and x, which are defined by

$$\{y_t, x_{t+1}\} \in \operatorname*{Argmax}_{y \in \mathscr{D}(x)} u(y, x) + \beta V(x_{t+1}) \tag{7.5}$$

Our problem is now to solve (7.4) for the function $V(x_t)$. This problem is particularly complicated as we are not solving for just a point that would satisfy the equation, but we are interested in finding a function that satisfies the equation. A simple procedure to find a solution would be the following

1. Make an initial guess on the form of the value function $V_0(x_t)$
2. Update the guess using the Bellman equation such that

$$V_{i+1}(x_t) = \max_{y_t \in \mathscr{D}(x_t)} u(y_t, x_t) + \beta V_i(h(y_t, x_t))$$

3. If $V_{i+1}(x_t) = V_i(x_t)$, then a fixed point has been found and the problem is solved, if not we go back to 2, and iterate on the process until convergence.

In other words, solving the Bellman equation just amounts to find the fixed point of the bellman equation, or introduction an operator notation, finding the fixed point of the operator T, such that

$$V_{i+1} = TV_i$$

where T stands for the list of operations involved in the computation of the Bellman equation. The problem is then that of the existence and the uniqueness of this fixed–point. Luckily, mathematicians have provided conditions for the existence and uniqueness of a solution.

7.1.2 Existence and uniqueness of a solution

Definition 1 *A metric space is a set S, together with a metric $\rho : S \times S \longrightarrow \mathbb{R}^+$, such that for all $x, y, z \in S$:*

1. *$\rho(x, y) \geqslant 0$, with $\rho(x, y) = 0$ if and only if $x = y$,*
2. *$\rho(x, y) = \rho(y, x)$,*
3. *$\rho(x, z) \leqslant \rho(x, y) + \rho(y, z)$.*

Definition 2 *A sequence* $\{x_n\}_{n=0}^{\infty}$ *in* S *converges to* $x \in S$*, if for each* $\varepsilon > 0$ *there exists an integer* N_ε *such that*

$$\rho(x_n, x) < \varepsilon \text{ for all } n \geqslant N_\varepsilon$$

Definition 3 *A sequence* $\{x_n\}_{n=0}^{\infty}$ *in* S *is a Cauchy sequence if for each* $\varepsilon > 0$ *there exists an integer* N_ε *such that*

$$\rho(x_n, x_m) < \varepsilon \text{ for all } n, m \geqslant N_\varepsilon$$

Definition 4 *A metric space* (S, ρ) *is complete if every Cauchy sequence in* S *converges to a point in* S*.*

Definition 5 *Let* (S, ρ) *be a metric space and* $T : S \to S$ *be function mapping* S *into itself.* T *is a contraction mapping (with modulus* β*) if for* $\beta \in (0, 1)$*,*

$$\rho(Tx, Ty) \leqslant \beta\rho(x, y), \text{ for all } x, y \in S.$$

We then have the following remarkable theorem that establishes the existence and uniqueness of the fixed point of a contraction mapping.

Theorem 1 (Contraction Mapping Theorem) *If* (S, ρ) *is a complete metric space and* $T : S \longrightarrow S$ *is a contraction mapping with modulus* $\beta \in (0, 1)$*, then*

1. T *has exactly one fixed point* $V \in S$ *such that* $V = TV$*,*
2. *for any* $V \in S$*,* $\rho(T^n V_0, V) < \beta^n \rho(V^0, V)$*, with* $n = 0, 1, 2 \ldots$

Since we are endowed with all the tools we need to prove the theorem, we shall do it.

Proof: In order to prove 1., we shall first prove that if we select any sequence $\{V_n\}_{n=0}^{\infty}$, such that for each n, $V_n \in S$ and

$$V_{n+1} = TV_n$$

this sequence converges and that it converges to $V \in S$. In order to show convergence of $\{V_n\}_{n=0}^{\infty}$, we shall prove that $\{V_n\}_{n=0}^{\infty}$ is a Cauchy sequence. First of all, note that the contraction property of T implies that

$$\rho(V_2, V_1) = \rho(TV_1, TV_0) \leqslant \beta\rho(V_1, V_0)$$

and therefore

$$\rho(V_{n+1}, V_n) = \rho(TV_n, TV_{n-1}) \leqslant \beta\rho(V_n, V_{n-1}) \leqslant \ldots \leqslant \beta^n \rho(V_1, V_0)$$

Now consider two terms of the sequence, V_m and V_n, $m > n$. The triangle inequality implies that

$$\rho(V_m, V_n) \leqslant \rho(V_m, V_{m-1}) + \rho(V_{m-1}, V_{m-2}) + \ldots + \rho(V_{n+1}, V_n)$$

therefore, making use of the previous result, we have

$$\rho(V_m, V_n) \leqslant \left(\beta^{m-1} + \beta^{m-2} + \ldots + \beta^n\right) \rho(V_1, V_0) \leqslant \frac{\beta^n}{1-\beta}\rho(V_1, V_0)$$

Since $\beta \in (0, 1)$, $\beta^n \to 0$ as $n \to \infty$, we have that for each $\varepsilon > 0$, there exists $N_\varepsilon \in \mathbb{N}$ such that $\rho(V_m, V_n) < \varepsilon$. Hence $\{V_n\}_{n=0}^{\infty}$ is a Cauchy sequence and it therefore converges. Further, since we have assume that S is complete, V_n converges to $V \in S$.

We now have to show that $V = TV$ in order to complete the proof of the first part. Note that, for each $\varepsilon > 0$, and for $V_0 \in S$, the triangular inequality implies

$$\rho(V, TV) \leqslant \rho(V, V_n) + \rho(V_n, TV)$$

But since $\{V_n\}_{n=0}^{\infty}$ is a Cauchy sequence, we have

$$\rho(V, TV) \leqslant \rho(V, V_n) + \rho(V_n, TV) \leqslant \frac{\varepsilon}{2} + \frac{\varepsilon}{2}$$

for large enough n, therefore $V = TV$.

Hence, we have proven that T possesses a fixed point and therefore have established its existence. We now have to prove uniqueness. This can be obtained by contradiction. Suppose, there exists another function, say $W \in S$ that satisfies $W = TW$. Then, the definition of the fixed point implies

$$\rho(V, W) = \rho(TV, TW)$$

but the contraction property implies

$$\rho(V, W) = \rho(TV, TW) \leqslant \beta\rho(V, W)$$

which, as $\beta > 0$ implies $\rho(V, W) = 0$ and so $V = W$. The limit is then unique.

Proving 2. is straightforward as

$$\rho(T^n V_0, V) = \rho(T^n V_0, TV) \leqslant \beta\rho(T^{n-1} V_0, V)$$

but we have $\rho(T^{n-1} V_0, V) = \rho(T^{n-1} V_0, TV)$ such that

$$\rho(T^n V_0, V) = \rho(T^n V_0, TV) \leqslant \beta\rho(T^{n-1} V_0, V) \leqslant \beta^2\rho(T^{n-2} V_0, V) \leqslant \ldots \leqslant \beta^n \rho(V_0, V)$$

which completes the proof. □

This theorem is of great importance as it establishes that any operator that possesses the contraction property will exhibit a unique fixed–point, which therefore provides some rationale to the algorithm we were designing in the previous section. It also insure that whatever the initial condition for this

algorithm, if the value function satisfies a contraction property, simple iterations will deliver the solution. It therefore remains to provide conditions for the value function to be a contraction. These are provided by the following theorem.

Theorem 2 (Blackwell's Sufficiency Conditions) *Let $X \subseteq \mathbb{R}^\ell$ and $B(X)$ be the space of bounded functions $V : X \longrightarrow \mathbb{R}$ with the uniform metric. Let $T : B(X) \longrightarrow B(X)$ be an operator satisfying*

1. *(Monotonicity) Let $V, W \in B(X)$, if $V(x) \leqslant W(x)$ for all $x \in X$, then $TV(x) \leqslant TW(x)$*

2. *(Discounting) There exists some constant $\beta \in (0, 1)$ such that for all $V \in B(X)$ and $a \geqslant 0$, we have*

$$T(V + a) \leqslant TV + \beta a$$

then T is a contraction with modulus β.

Proof: Let us consider two functions $V, W \in B(X)$ satisfying 1. and 2., and such that

$$V \leqslant W + \rho(V, W)$$

Monotonicity first implies that

$$TV \leqslant T(W + \rho(V, W))$$

and discounting

$$TV \leqslant TW + \beta\rho(V, W))$$

since $\rho(V, W) \geqslant 0$ plays the same role as a. We therefore get

$$TV - TW \leqslant \beta\rho(V, W)$$

Likewise, if we now consider that $V \leqslant W + \rho(V, W)$, we end up with

$$TW - TV \leqslant \beta\rho(V, W)$$

Consequently, we have

$$|TV - TW| \leqslant \beta\rho(V, W)$$

so that

$$\rho(TV, TW) \leqslant \beta\rho(V, W)$$

which defines a contraction. This completes the proof. □

This theorem is extremely useful as it gives us simple tools to check whether a problem is a contraction and therefore permits to check whether the simple algorithm we were defined is appropriate for the problem we have in hand.

As an example, let us consider the optimal growth model, for which the Bellman equation writes

$$V(k_t) = \max_{c_t \in \mathscr{C}} u(c_t) + \beta V(k_{t+1})$$

with $k_{t+1} = F(k_t) - c_t$. In order to save on notations, let us drop the time subscript and denote the next period capital stock by k', such that the Bellman equation rewrites, plugging the law of motion of capital in the utility function

$$V(k) = \max_{k' \in \mathscr{K}} u(F(k) - k') + \beta V(k')$$

Let us now define the operator T as

$$(TV)(k) = \max_{k' \in \mathscr{K}} u(F(k) - k') + \beta V(k')$$

We would like to know if T is a contraction and therefore if there exists a unique function V such that

$$V(k) = (TV)(k)$$

In order to achieve this task, we just have to check whether T is monotonic and satisfies the discounting property.

1. Monotonicity: Let us consider two candidate value functions, V and W, such that $V(k) \leqslant W(k)$ for all $k \in \mathscr{K}$. What we want to show is that $(TV)(k) \leqslant (TW)(k)$. In order to do that, let us denote by $\widetilde{k}'$ the optimal next period capital stock, that is

   $$(TV)(k) = u(F(k) - \widetilde{k}') + \beta V(\widetilde{k}')$$

   But now, since $V(k) \leqslant W(k)$ for all $k \in \mathscr{K}$, we have $V(\widetilde{k}') \leqslant W(\widetilde{k}')$, such that it should be clear that

   $$\begin{aligned} (TV)(k) &\leqslant u(F(k) - \widetilde{k}') + \beta W(\widetilde{k}') \\ &\leqslant \max_{k' \in \mathscr{K}} u(F(k) - k') + \beta W(k') = (TW(k')) \end{aligned}$$

Hence we have shown that $V(k) \leqslant W(k)$ implies $(TV)(k) \leqslant (TW)(k)$ and therefore established monotonicity.

2. Discounting: Let us consider a candidate value function, V, and a positive constant a.

$$\begin{aligned}(T(V+a))(k) &= \max_{k' \in \mathscr{K}} u(F(k)-k') + \beta(V(k')+a) \\ &= \max_{k' \in \mathscr{K}} u(F(k)-k') + \beta V(k') + \beta a \\ &= (TV)(k) + \beta a\end{aligned}$$

Therefore, the Bellman equation satisfies discounting in the case of optimal growth model.

Hence, the optimal growth model satisfies the Blackwell's sufficient conditions for a contraction mapping, and therefore the value function exists and is unique. We are now in position to design a numerical algorithm to solve the bellman equation.

7.2 Deterministic dynamic programming

7.2.1 Value function iteration

The contraction mapping theorem gives us a straightforward way to compute the solution to the bellman equation: iterate on the operator T, such that $V_{i+1} = TV_i$ up to the point where the distance between two successive value function is small enough. Basically this amounts to apply the following algorithm

1. Decide on a grid, $\mathscr{X}$, of admissible values for the state variable x

$$\mathscr{X} = \{x_1, \ldots, x_N\}$$

formulate an initial guess for the value function $V_0(x)$ and choose a stopping criterion $\varepsilon > 0$.

2. For each $x_\ell \in \mathscr{X}$, $\ell = 1, \ldots, N$, compute

$$V_{i+1}(x_\ell) = \max_{\{x' \in \mathscr{X}\}} u(y(x_\ell, x'), x_\ell) + \beta V_i(x')$$

3. If $\|V_{i+1}(x) - V_i(x)\| < \varepsilon$ go to the next step, else go back to 2.

4. Compute the final solution as

$$y^\star(x) = y(x, x')$$

and

$$V^\star(x) = \frac{u(y^\star(x), x)}{1 - \beta}$$

In order to better understand the algorithm, let us consider a simple example and go back to the optimal growth model, with

$$u(c) = \frac{c^{1-\sigma} - 1}{1 - \sigma}$$

and

$$k' = k^\alpha - c + (1 - \delta)k$$

Then the Bellman equation writes

$$V(k) = \max_{0 \leqslant c \leqslant k^\alpha + (1-\delta)k} \frac{c^{1-\sigma} - 1}{1 - \sigma} + \beta V(k')$$

From the law of motion of capital we can determine consumption as

$$c = k^\alpha + (1 - \delta)k - k'$$

such that plugging this results in the Bellman equation, we have

$$V(k) = \max_{(1-\delta)k \leqslant k' \leqslant k^\alpha + (1-\delta)k} \frac{(k^\alpha + (1 - \delta)k - k')^{1-\sigma} - 1}{1 - \sigma} + \beta V(k')$$

Now, let us define a grid of N feasible values for k such that we have

$$\mathscr{K} = \{k_1, \ldots, k_N\}$$

and an initial value function $V_0(k)$ — that is a vector of N numbers that relate each k_ℓ to a value. Note that this may be anything we want as we know — by the contraction mapping theorem — that the algorithm will converge. But, if we want it to converge fast enough it may be a good idea to impose a good initial guess. Finally, we need a stopping criterion.

Then, for each $\ell = 1, \ldots, N$, we compute the feasible values that can be taken by the quantity in left hand side of the value function

$$\mathscr{V}_{\ell,h} \equiv \frac{(k_\ell^\alpha + (1-\delta)k_\ell - k'_h)^{1-\sigma} - 1}{1-\sigma} + \beta V(k_h) \text{ for } h \text{ feasible}$$

It is important to understand what "h feasible" means. Indeed, we only compute consumption when it is positive and smaller than total output, which restricts the number of possible values for k'. Namely, we want k' to satisfy

$$0 \leqslant k' \leqslant k^\alpha + (1-\delta)k$$

which puts a lower and an upper bound on the index h. When the grid of values is uniform — that is when $k_h = \underline{k} + (h-1)d_k$, where d_k is the increment in the grid, the upper bound can be computed as

$$\overline{h} = \mathsf{E}\left(\frac{k_i^\alpha + (1-\delta)k_i - \overline{k}}{d_k}\right) + 1$$

Then we find

$$\mathscr{V}^\star_{\ell,h} \max_{h=1,\ldots,\overline{h}} \mathscr{V}_{\ell,h}$$

and set

$$V_{i+1}(k_\ell) = \mathscr{V}^\star_{\ell,h}$$

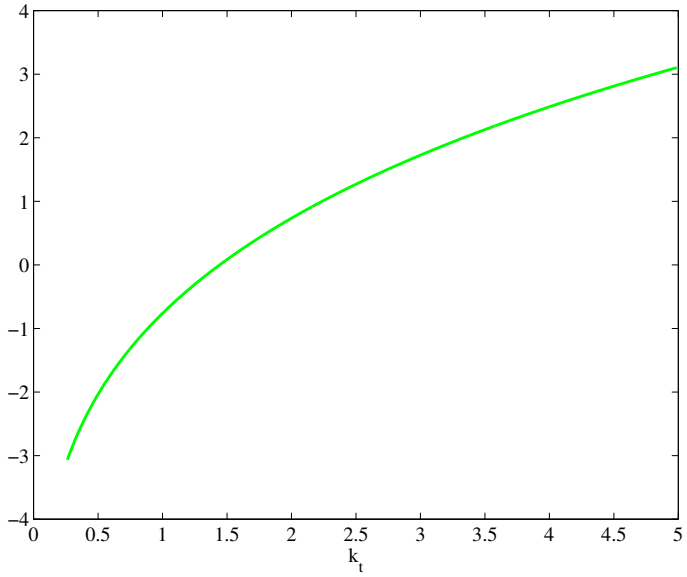
and keep in memory the index $h^\star = \text{Argmax}_{h=1,\ldots,N} \mathscr{V}_{\ell,h}$, such that we have

$$k'(k_\ell) = k_{h^\star}$$

In figures 7.1 and 7.2, we report the value function and the decision rules obtained from the deterministic optimal growth model with $\alpha = 0.3$, $\beta = 0.95$,

$\delta = 0.1$ and $\sigma = 1.5$. The grid for the capital stock is composed of 1000 data points ranging from $(1 - \Delta_k)k^\star$ to $(1 + \Delta_k)k^\star$, where $k^\star$ denotes the steady state and $\Delta_k = 0.9$. The algorithm[2] then converges in 211 iterations and 110.5 seconds on a 800Mhz computer when the stopping criterion is $\varepsilon = 1e^{-6}$.

Figure 7.1: Deterministic OGM (Value function, Value iteration)

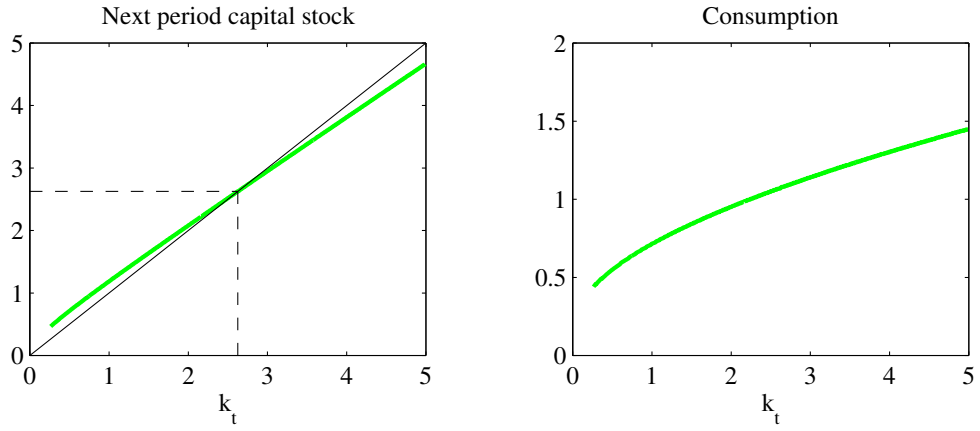


MATLAB CODE: VALUE FUNCTION ITERATION

```
sigma   = 1.5;                          % utility parameter
delta   = 0.1;                          % depreciation rate
beta    = 0.95;                         % discount factor
alpha   = 0.30;                         % capital elasticity of output

nbk     = 1000;                         % number of data points in the grid
crit    = 1;                            % convergence criterion
epsi    = 1e-6;                         % convergence parameter

ks      = ((1-beta*(1-delta))/(alpha*beta))^(1/(alpha-1));

dev     = 0.9;                          % maximal deviation from steady state
kmin    = (1-dev)*ks;                   % lower bound on the grid
kmax    = (1+dev)*ks;                   % upper bound on the grid
```

[2]This code and those that follow are not efficient from a computational point of view, as they are intended to have you understand the method without adding any coding complications. Much faster implementations can be found in the accompanying matlab codes.

Figure 7.2: Deterministic OGM (Decision rules, Value iteration)




```
dk      = (kmax-kmin)/(nbk-1);      % implied increment
kgrid   = linspace(kmin,kmax,nbk)'; % builds the grid
v       = zeros(nbk,1);             % value function
dr      = zeros(nbk,1);             % decision rule (will contain indices)

while crit>epsi;
  for i=1:nbk
    %
    % compute indexes for which consumption is positive
    %
    tmp         = (kgrid(i)^alpha+(1-delta)*kgrid(i)-kmin);
    imax        = min(floor(tmp/dk)+1,nbk);
    %
    % consumption and utility
    %
    c           = kgrid(i)^alpha+(1-delta)*kgrid(i)-kgrid(1:imax);
    util        = (c.^(1-sigma)-1)/(1-sigma);
    %
    % find value function
    %
    [tv(i),dr(i)] = max(util+beta*v(1:imax));
  end;

  crit  = max(abs(tv-v));   % Compute convergence criterion
  v     = tv;               % Update the value function
end
%
% Final solution
%
kp  = kgrid(dr);
```

```
c   = kgrid.^alpha+(1-delta)*kgrid-kp;
util= (c.^(1-sigma)-1)/(1-sigma);
```

7.2.2 Taking advantage of interpolation

A possible improvement of the method is to have a much looser grid on the capital stock but have a pretty fine grid on the control variable (consumption in the optimal growth model). Then the next period value of the state variable can be computed much more precisely. However, because of this precision and the fact the grid is rougher, it may be the case that the computed optimal value for the next period state variable does not lie in the grid, such that the value function is unknown at this particular value. Therefore, we use any interpolation scheme to get an approximation of the value function at this value. One advantage of this approach is that it involves less function evaluations and is usually less costly in terms of CPU time. The algorithm is then as follows:

1. Decide on a grid, $\mathscr{X}$, of admissible values for the state variable x

   $$\mathscr{X} = \{x_1, \ldots, x_N\}$$

   Decide on a grid, $\mathscr{Y}$, of admissible values for the control variable y

   $$\mathscr{Y} = \{y_1, \ldots, y_M\} \text{ with } M \gg N$$

   formulate an initial guess for the value function $V_0(x)$ and choose a stopping criterion $\varepsilon > 0$.

2. For each $x_\ell \in \mathscr{X}$, $\ell = 1, \ldots, N$, compute

   $$x'_{\ell,j} = h(y_j, x_\ell) \forall j = 1, \ldots, M$$

   Compute an interpolated value function at each $x'_{\ell,j} = h(y_j, x_\ell)$: $\widetilde{V}_i(x'_{\ell,j})$

   $$V_{i+1}(x_\ell) = \max_{\{y \in \mathscr{Y}\}} u(y, x_\ell) + \beta \widetilde{V}_i(x'_{\ell,j})$$

3. If $\|V_{i+1}(x) - V_i(x)\| < \varepsilon$ go to the next step, else go back to 2.

4. Compute the final solution as

$$V^\star(x) = \frac{u(y^\star, x)}{1-\beta}$$

I report now the matlab code for this approach when we use cubic spline interpolation for the value function, 20 nodes for the capital stock and 1000 nodes for consumption. The algorithm converges in 182 iterations and 40.6 seconds starting from initial condition for the value function

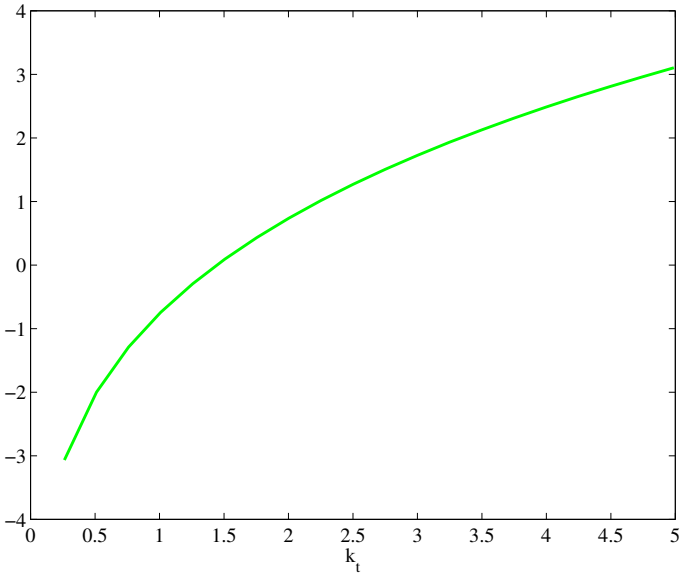
$$v_0(k) = \frac{((c^\star/y^\star) * k^\alpha)^{1-\sigma} - 1}{(1-\sigma)}$$

MATLAB CODE: VALUE FUNCTION ITERATION WITH INTERPOLATION

```
sigma   = 1.5;                      % utility parameter
delta   = 0.1;                      % depreciation rate
beta    = 0.95;                     % discount factor
alpha   = 0.30;                     % capital elasticity of output

nbk     = 20;                       % number of data points in the K grid
nbk     = 1000;                     % number of data points in the C grid
crit    = 1;                        % convergence criterion
epsi    = 1e-6;                     % convergence parameter

ks      = ((1-beta*(1-delta))/(alpha*beta))^(1/(alpha-1));

dev     = 0.9;                      % maximal deviation from steady state
kmin    = (1-dev)*ks;               % lower bound on the grid
kmax    = (1+dev)*ks;               % upper bound on the grid
kgrid   = linspace(kmin,kmax,nbk)'; % builds the grid
cmin    = 0.01;                     % lower bound on the grid
cmak    = kmax^alpha;               % upper bound on the grid
c       = linspace(cmin,cmax,nbc)'; % builds the grid
v       = zeros(nbk,1);             % value function
dr      = zeros(nbk,1);             % decision rule (will contain indices)
util    = (c.^(1-sigma)-1)/(1-sigma);

while crit>epsi;
   for i=1:nbk;
     kp             = A(j)*k(i).^alpha+(1-delta)*k(i)-c;
     vi             = interp1(k,v,kp,'spline');
     [Tv(i),dr(i)]  = max(util+beta*vi);
```

```
   end
  crit  = max(abs(tv-v));   % Compute convergence criterion
  v     = tv;               % Update the value function
end
%
% Final solution
%
kp  = kgrid(dr);
c   = kgrid.^alpha+(1-delta)*kgrid-kp;
util= (c.^(1-sigma)-1)/(1-sigma);
v   = util/(1-beta);
```

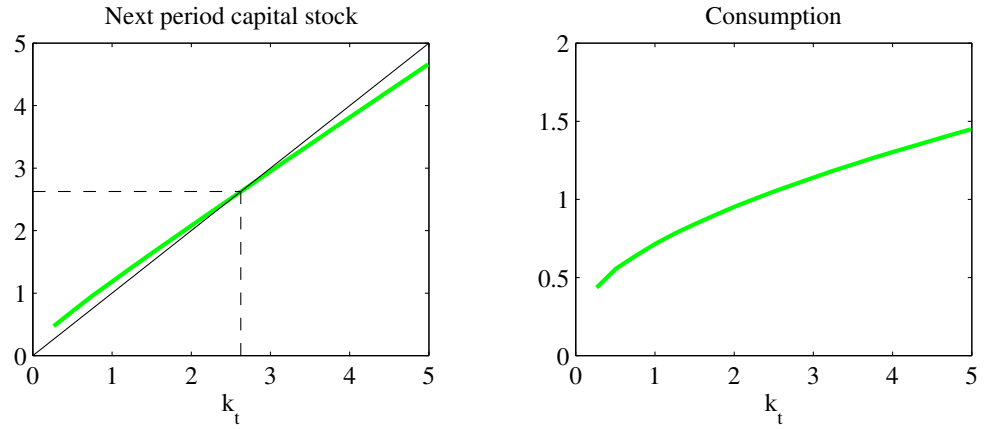
Figure 7.3: Deterministic OGM (Value function, Value iteration with interpolation)



7.2.3 Policy iterations: Howard Improvement

The simple value iteration algorithm has the attractive feature of being particularly simple to implement. However, it is a slow procedure, especially for infinite horizon problems, since it can be shown that this procedure converges at the rate β, which is usually close to 1! Further, it computes unnecessary quantities during the algorithm which slows down convergence. Often, com-

Figure 7.4: Deterministic OGM (Decision rules, Value iteration with interpolation)



putation speed is really important, for instance when one wants to perform a sensitivity analysis of the results obtained in a model using different parameterization. Hence, we would like to be able to speed up convergence. This can be achieved relying on Howard improvement method. This method actually iterates on policy functions rather than iterating on the value function. The algorithm may be described as follows

1. Set an initial feasible decision rule for the control variable $y = f_0(x)$ and compute the value associated to this guess, assuming that this rule is operative forever:

$$V(x_t) = \sum_{s=0}^{\infty} \beta^s u(f_i(x_{t+s}), x_{t+s})$$

   taking care of the fact that $x_{t+1} = h(x_t, y_t) = h(x_t, f_i(x_t))$ with $i = 0$. Set a stopping criterion $\varepsilon > 0$.

2. Find a new policy rule $y = f_{i+1}(x)$ such that

$$f_{i+1}(x) \in \underset{y}{\text{Argmax}}\, u(y, x) + \beta V(x')$$

   with $x' = h(x, f_i(x))$

3. check if $\|f_{i+1}(x) - f_i(x)\| < \varepsilon$, if yes then stop, else go back to 2.

Note that this method differs fundamentally from the value iteration algorithm in at least two dimensions

i one iterates on the policy function rather than on the value function;

ii the decision rule is used forever whereas it is assumed that it is used only two consecutive periods in the value iteration algorithm. This is precisely this last feature that accelerates convergence.

Note that when computing the value function we actually have to solve a linear system of the form

$$V_{i+1}(x_\ell) = u(f_{i+1}(x_\ell), x_\ell) + \beta V_{i+1}(h(x_\ell, f_{i+1}(x_\ell))) \quad \forall x_\ell \in \mathscr{X}$$

for $V_{i+1}(x_\ell)$, which may be rewritten as

$$V_{i+1}(x_\ell) = u(f_{i+1}(x_\ell), x_\ell) + \beta Q V_{i+1}(x_\ell) \quad \forall x_\ell \in \mathscr{X}$$

where Q is an $(N \times N)$ matrix

$$Q_{\ell j} = \begin{cases} 1 \text{ if } x'_j \equiv h(f_{i+1}(x_\ell), x_\ell) = x_\ell \\ 0 \text{ otherwise} \end{cases}$$

Note that although it is a big matrix, Q is sparse, which can be exploited in solving the system, to get

$$V_{i+1}(x) = (I - \beta Q)^{-1} u(f_{i+1}(x), x)$$

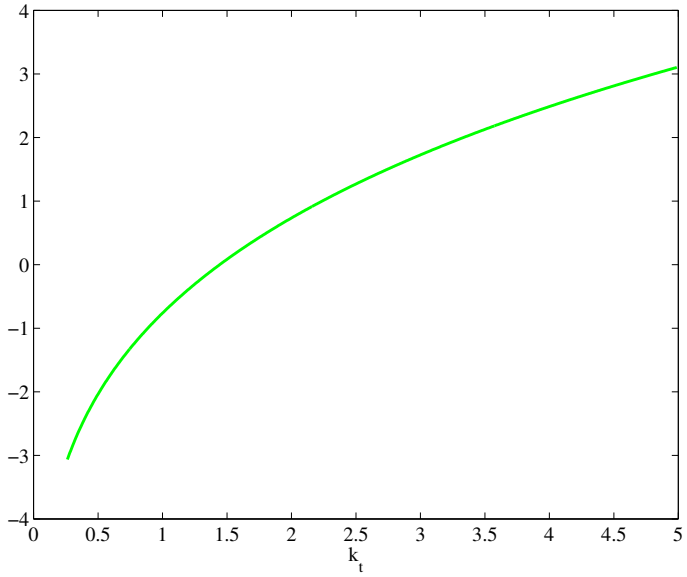
We apply this algorithm to the same optimal growth model as in the previous section and report the value function and the decision rules at convergence in figures 7.5 and 7.6. The algorithm converges in only 18 iterations and 9.8 seconds, starting from the same initial guess and using the same parameterization!

MATLAB CODE: POLICY ITERATION

```
sigma   = 1.50;                     % utility parameter
delta   = 0.10;                     % depreciation rate
beta    = 0.95;                     % discount factor
alpha   = 0.30;                     % capital elasticity of output

nbk     = 1000;                     % number of data points in the grid
crit    = 1;                        % convergence criterion
epsi    = 1e-6;                     % convergence parameter

ks      = ((1-beta*(1-delta))/(alpha*beta))^(1/(alpha-1));
dev     = 0.9;                      % maximal deviation from steady state
kmin    = (1-dev)*ks;               % lower bound on the grid
kmax    = (1+dev)*ks;               % upper bound on the grid
kgrid   = linspace(kmin,kmax,nbk)'; % builds the grid
v       = zeros(nbk,1);             % value function
kp0     = kgrid;                    % initial guess on k(t+1)
dr      = zeros(nbk,1);             % decision rule (will contain indices)
%
% Main loop
%
while crit>epsi;
  for i=1:nbk
    %
    % compute indexes for which consumption is positive
    %
    imax    = min(floor((kgrid(i)^alpha+(1-delta)*kgrid(i)-kmin)/devk)+1,nbk);
    %
    % consumption and utility
    %
    c       = kgrid(i)^alpha+(1-delta)*kgrid(i)-kgrid(1:imax);
    util    = (c.^(1-sigma)-1)/(1-sigma);
    %
    % find new policy rule
    %
    [v1,dr(i)]= max(util+beta*v(1:imax));
  end;
    %
    % decision rules
    %
    kp  = kgrid(dr)
    c   = kgrid.^alpha+(1-delta)*kgrid-kp;
    %
    % update the value
    %
    util= (c.^(1-sigma)-1)/(1-sigma);
    Q   = sparse(nbk,nbk);
    for i=1:nbk;
        Q(i,dr(i)) = 1;
```

```
    end
    Tv   = (speye(nbk)-beta*Q)\u;
    crit= max(abs(kp-kp0));
    v    = Tv;
    kp0 = kp;
end
```

Figure 7.5: Deterministic OGM (Value function, Policy iteration)



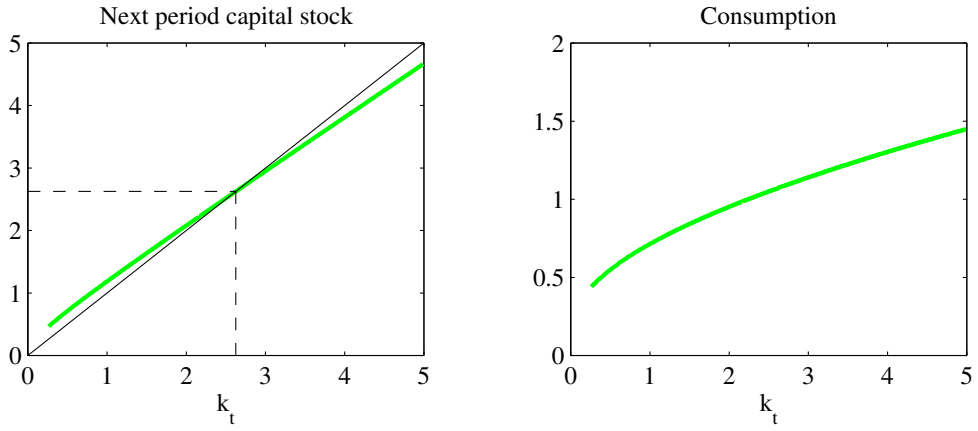
As experimented in the particular example of the optimal growth model, policy iteration algorithm only requires a few iterations. Unfortunately, we have to solve the linear system

$$(I - \beta Q)V_{i+1} = u(y, x)$$

which may be particularly costly when the number of grid points is important. Therefore, a number of researchers has proposed to replace the matrix inversion by an additional iteration step, leading to the so–called *modified policy iteration with k steps*, which replaces the linear problem by the following iteration scheme

1. Set $J_0 = V_i$

Figure 7.6: Deterministic OGM (Decision rules, Policy iteration)



2. iterate k times on

$$J_{i+1} = u(y, x) + \beta Q J_i, \ i = 0, \ldots, k$$

3. set $V_{i+1} = J_k$.

When $k \longrightarrow \infty$, J_k tends toward the solution of the linear system.

7.2.4 Parametric dynamic programming

This last technics I will describe borrows from approximation theory using either orthogonal polynomials or spline functions. The idea is actually to make a guess for the functional for of the value function and iterate on the parameters of this functional form. The algorithm then works as follows

1. Choose a functional form for the value function $\widetilde{V}(x; \Theta)$, a grid of interpolating nodes $\mathscr{X} = \{x_1, \ldots, x_N\}$, a stopping criterion $\varepsilon > 0$ and an initial vector of parameters Θ_0.

2. Using the conjecture for the value function, perform the maximization step in the Bellman equation, that is compute $w_\ell = T(\widetilde{V}(x, \Theta_i))$

$$w_\ell = T(\widetilde{V}(x_\ell, \Theta_i)) = \max_y u(y, x_\ell) + \beta \widetilde{V}(x', \Theta_i)$$

$$\text{s.t. } x' = h(y, x_\ell) \text{ for } \ell = 1, \ldots, N$$

3. Using the approximation method you have chosen, compute a new vector of parameters Θ_{i+1} such that $\widetilde{V}(x, \Theta_{i+1})$ approximates the data (x_ℓ, w_ℓ).

4. If $\|\widetilde{V}(x, \Theta_{i+1}) - \widetilde{V}(x, \Theta_i)\| < \varepsilon$ then stop, else go back to 2.

First note that for this method to be implementable, we need the payoff function and the value function to be continuous. The approximation function may be either a combination of polynomials, neural networks, splines. Note that during the optimization problem, we may have to rely on a numerical maximization algorithm, and the approximation method may involve numerical minimization in order to solve a non–linear least–square problem of the form:

$$\Theta_{i+1} \in \underset{\Theta}{\text{Argmin}} \sum_{\ell=1}^{N} (w_\ell - \widetilde{V}(x_\ell; \Theta))^2$$

This algorithm is usually much faster than value iteration as it may not require iterating on a large grid. As an example, I will once again focus on the optimal growth problem we have been dealing with so far, and I will approximate the value function by

$$\widetilde{V}(k; \Theta) = \sum_{i=1}^{p} \theta_i \varphi_i \left(2 \frac{k - \underline{k}}{\overline{k} - \underline{k}} - 1 \right)$$

where $\{\varphi_i(.)\}_{i=0}^{p}$ is a set of Chebychev polynomials. In the example, I set $p = 10$ and used 20 nodes. Figures 7.8 and 7.7 report the decision rule and the value function in this case, and table 7.1 reports the parameters of the approximation function. The algorithm converged in 242 iterations, but took less much time that value iterations.

Figure 7.7: Deterministic OGM (Value function, Parametric DP)

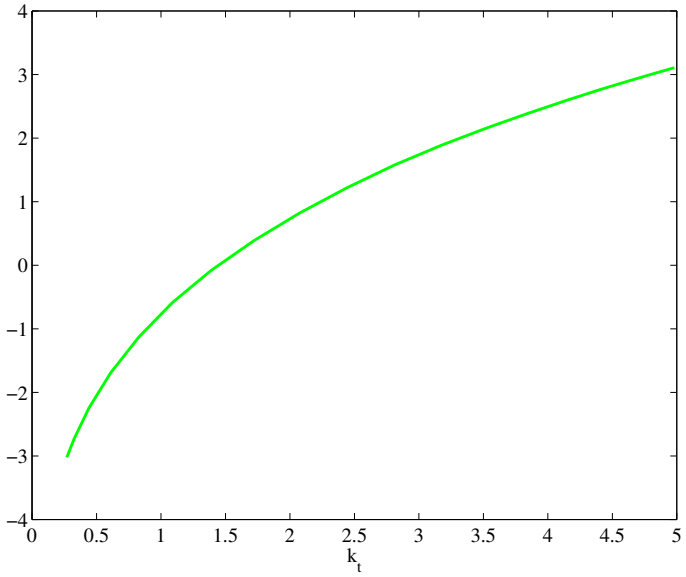


Figure 7.8: Deterministic OGM (Decision rules, Parametric DP)

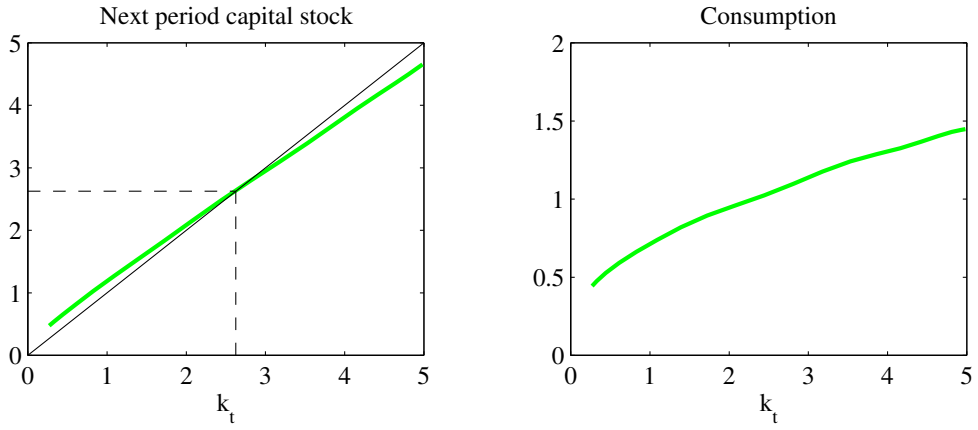


Table 7.1: Value function approximation

θ_0	0.82367
θ_1	2.78042
θ_2	-0.66012
θ_3	0.23704
θ_4	-0.10281
θ_5	0.05148
θ_6	-0.02601
θ_7	0.01126
θ_8	-0.00617
θ_9	0.00501
θ_{10}	-0.00281

MATLAB CODE: PARAMETRIC DYNAMIC PROGRAMMING

```
sigma   = 1.50;                              % utility parameter
delta   = 0.10;                              % depreciation rate
beta    = 0.95;                              % discount factor
alpha   = 0.30;                              % capital elasticity of output

nbk     = 20;                                % # of data points in the grid
p       = 10;                                % order of polynomials
crit    = 1;                                 % convergence criterion
iter    = 1;                                 % iteration
epsi    = 1e-6;                              % convergence parameter

ks      = ((1-beta*(1-delta))/(alpha*beta))^(1/(alpha-1));
dev     = 0.9;                               % maximal dev. from steady state
kmin    = (1-dev)*ks;                        % lower bound on the grid
kmax    = (1+dev)*ks;                        % upper bound on the grid
rk      = -cos((2*[1:nbk]'-1)*pi/(2*nnk));  % Interpolating nodes
kgrid   = kmin+(rk+1)*(kmax-kmin)/2;         % mapping
%
% Initial guess for the approximation
%
v       = (((kgrid.^alpha).^(1-sigma)-1)/((1-sigma)*(1-beta)));;
X       = chebychev(rk,n);
th0     = X\v
Tv      = zeros(nbk,1);
kp      = zeros(nbk,1);
%
% Main loop
%
options=foptions;
options(14)=1e9;

while crit>epsi;
   k0   = kgrid(1);
   for i=1:nbk
      param = [alpha beta delta sigma kmin kmax n kgrid(i)];
      kp(i) = fminu('tv',k0,options,[],param,th0);
      k0    = kp(i);
      Tv(i) = -tv(kp(i),param,th0);
   end;
   theta= X\Tv;
   crit = max(abs(Tv-v));
   v    = Tv;
   th0  = theta;
   iter= iter+1;
end
```

MATLAB CODE: EXTRA FUNCTIONS

```
function res=tv(kp,param,theta);
alpha   = param(1);
beta    = param(2);
delta   = param(3);
sigma   = param(4);
kmin    = param(5);
kmax    = param(6);
n       = param(7);
k       = param(8);
kp      = sqrt(kp.^2);                      % insures positivity of k'
v       = value(kp,[kmin kmax n],theta);    % computes the value function
c       = k.^alpha+(1-delta)*k-kp;          % computes consumption
d       = find(c<=0);                       % find negative consumption
c(d)    = NaN;                              % get rid off negative c
util    = (c.^(1-sigma)-1)/(1-sigma);       % computes utility
util(d) = -1e12;                            % utility = low number for c<0
res     = -(util+beta*v);                   % compute -TV (we are minimizing)
```

```
function v = value(k,param,theta);
kmin    = param(1);
kmax    = param(2);
n       = param(3);
k       = 2*(k-kmin)/(kmax-kmin)-1;
v       = chebychev(k,n)*theta;
```

```
function Tx=chebychev(x,n);
X=x(:);
lx=size(X,1);
if n<0;error('n should be a positive integer');end
switch n;
case 0;
   Tx=[ones(lx,1)];
case 1;
   Tx=[ones(lx,1) X];
otherwise
   Tx=[ones(lx,1) X];
   for i=3:n+1;
      Tx=[Tx 2*X.*Tx(:,i-1)-Tx(:,i-2)];
   end
end
```

A potential problem with the use of Chebychev polynomials is that they do not put any assumption on the shape of the value function that we know to be concave and strictly increasing in this case. This is why Judd [1998]

recommends to use shape–preserving methods such as Schumaker approximation in this case. Judd and Solnick [1994] have successfully applied this latter technic to the optimal growth model and found that the approximation was very good and dominated a lot of other methods (they actually get the same precision with 12 nodes as the one achieved with a 1200 data points grid using a value iteration technic).

7.3 Stochastic dynamic programming

In a large number of problems, we have to deal with stochastic shocks — just think of a standard RBC model — dynamic programming technics can obviously be extended to deal with such problems. I will first show how we can obtain the Bellman equation before addressing some important issues concerning discretization of shocks. Then I will describe the implementation of the value iteration and policy iteration technics for the stochastic case.

7.3.1 The Bellman equation

The stochastic problem differs from the deterministic problem in that we now have a ...stochastic shock! The problem then defines a value function which has as argument the state variable x_t but also the *stationary* shock s_t, whose sequence $\{s_t\}_{t=0}^{+\infty}$ satisfies

$$s_{t+1} = \Phi(s_t, \varepsilon_{t+1}) \tag{7.6}$$

where ε is a white noise process. The value function is therefore given by

$$V(x_t, s_t) = \max_{\{y_{t+\tau} \in \mathscr{D}(x_{t+\tau}, s_{t+\tau})\}_{\tau=0}^{\infty}} E_t \sum_{\tau=0}^{\infty} \beta^\tau u(y_{t+\tau}, x_{t+\tau}, s_{t+\tau}) \tag{7.7}$$

subject to (7.6) and

$$x_{t+1} = h(x_t, y_t, s_t) \tag{7.8}$$

Since both y_t, x_t and s_t are either perfectly observed or decided in period t they are known in t, such that we may rewrite the value function as

$$V(x_t, s_t) = \max_{\{y_t \in \mathscr{D}(x_t, s_t), \{y_{t+\tau} \in \mathscr{D}(x_{t+\tau}, s_{t+\tau})\}_{\tau=1}^{\infty}\}} u(y_t, x_t, s_t) + E_t \sum_{\tau=1}^{\infty} \beta^\tau u(y_{t+\tau}, x_{t+\tau}, s_{t+\tau})$$

or

$$V(x_t, s_t) = \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) + \max_{\{y_{t+\tau} \in \mathscr{D}(x_{t+\tau}, s_{t+\tau})\}_{\tau=1}^{\infty}} E_t \sum_{\tau=1}^{\infty} \beta^{\tau} u(y_{t+\tau}, x_{t+\tau}, s_{t+\tau})$$

Using the change of variable $k = \tau - 1$, this rewrites

$$V(x_t, s_t) = \max_{y_t \in \mathscr{D}(x_{t+\tau}, s_{t+\tau})} u(y_t, x_t, s_t) + \max_{\{y_{t+1+k} \in \mathscr{D}(x_{t+1+k}, s_{t+1+k})\}_{k=0}^{\infty}} \beta E_t \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k}, s_{t+1+k})$$

It is important at this point to recall that

$$\begin{aligned} E_t(X(\omega_{t+\tau})) &= \int X(\omega_{t+\tau}) f(\omega_{t+\tau}|\omega_t) \mathrm{d}\omega_{t+\tau} \\ &= \int \int X(\omega_{t+\tau}) f(\omega_{t+\tau}|\omega_{t+\tau-1}) f(\omega_{t+\tau-1}|\omega_t) \mathrm{d}\omega_{t+\tau} \mathrm{d}\omega_{t+\tau-1} \\ &= \int \dots \int X(\omega_{t+\tau}) f(\omega_{t+\tau}|\omega_{t+\tau-1}) \dots f(\omega_{t+1}|\omega_t) \mathrm{d}\omega_{t+\tau} \dots \mathrm{d}\omega_{t+1} \end{aligned}$$

which as corollary the law of iterated projections, such that the value function rewrites

$$\begin{aligned} V(x_t, s_t) &= \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) \\ &\quad + \max_{\{y_{t+1+k} \in \mathscr{D}(x_{t+1+k}, s_{t+1+k})\}_{k=0}^{\infty}} \beta E_t E_{t+1} \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k}, s_{t+1+k}) \end{aligned}$$

or

$$\begin{aligned} V(x_t, s_t) &= \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) \\ &\quad + \max_{\{y_{t+1+k} \in \mathscr{D}(x_{t+1+k}, s_{t+1+k})\}_{k=0}^{\infty}} \beta \int E_{t+1} \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k}, s_{t+1+k}) f(s_{t+1}|s_t) \mathrm{d}s_{t+1} \end{aligned}$$

Note that because each value for the shock defines a particular mathematical object the maximization of the integral corresponds to the integral of the maximization, therefore the max operator and the summation are interchangeable, so that we get

$$\begin{aligned} V(x_t, s_t) &= \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) \\ &\quad + \beta \int \max_{\{y_{t+1+k}\}_{k=0}^{\infty}} E_{t+1} \sum_{k=0}^{\infty} \beta^k u(y_{t+1+k}, x_{t+1+k}, s_{t+1+k}) f(s_{t+1}|s_t) \mathrm{d}s_{t+1} \end{aligned}$$

By definition, the term under the integral corresponds to $V(x_{t+1}, s_{t+1})$, such that the value rewrites

$$V(x_t, s_t) = \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) + \beta \int V(x_{t+1}, s_{t+1}) f(s_{t+1}|s_t) \mathrm{d}s_{t+1}$$

or equivalently

$$V(x_t, s_t) = \max_{y_t \in \mathscr{D}(x_t, s_t)} u(y_t, x_t, s_t) + \beta E_t V(x_{t+1}, s_{t+1})$$

which is precisely the Bellman equation for the stochastic dynamic programming problem.

7.3.2 Discretization of the shocks

A very important problem that arises whenever we deal with value iteration or policy iteration in a stochastic environment is that of the discretization of the space spanned by the shocks. Indeed, the use of a continuous support for the stochastic shocks is infeasible for a computer that can only deals with discrete supports. We therefore have to transform the continuous problem into a discrete one with the constraint that the asymptotic properties of the continous and the discrete processes should be the same. The question we therefore face is: *does there exist a discrete representation for s which is equivalent to its continuous original representation?* The answer to this question is **yes**. In particular as soon as we deal with (V)AR processes, we can use a very powerful tool: *Markov chains*.

Markov Chains: A Markov chain is a sequence of random values whose probabilities at a time interval depends upon the value of the number at the previous time. W will restrict ourselves to discrete-time Markov chains, in which the state changes at certain discrete time instants, indexed by an integer variable t. At each time step t, the Markov chain is in a state, denoted by $s \in \mathscr{S} \equiv \{s_1, \ldots, s_M\}$. $\mathscr{S}$ is called the state space.

The Markov chain is described in terms of *transition probabilities* π_{ij}. This transition probability should read as follows

> *If the state happens to be s_i in period t, the probability that the next state is equal to s_j is π_{ij}.*

We therefore get the following definition

Definition 6 *A Markov chain is a stochastic process with a discrete indexing $\mathscr{S}$, such that the conditional distribution of s_{t+1} is independent of all previously attained state given s_t:*

$$\pi_{ij} = Prob(s_t + 1 = s_j | s_t = s_i), s_i, s_j \in \mathscr{S}.$$

The important assumption we shall make concerning Markov processes is that the transition probabilities, π_{ij}, apply as soon as state s_i is activated no matter the history of the shocks nor how the system got to state s_i. In other words there is no hysteresis. From a mathematical point of view, this corresponds to the so–called Markov property

$$P(s_{t+1} = s_j | s_t = s_i, s_{t-1} = s_{i_{n-1}}, \dots s_0 = s_{i_0}) = P(s_{t+1} = s_j | s_t = s_i) = \pi_{ij}$$

for all period t, all states s_i, $s_j \in \mathscr{S}$, and all sequences $\{s_{i_n}\}_{n=0}^{t-1}$ of earlier states. Thus, the probability the next state s_{t+1} does only depend on the current realization of s.

The transition probabilities π_{ij} must of course satisfy

1. $\pi_{ij} \geqslant 0$ for all $i, j = 1, \dots, M$
2. $\sum_{j=1}^{M} \pi_{ij} = 1$ for all $i = 1, \dots, M$

All of the elements of a Markov chain model can then be encoded in a *transition probability matrix*

$$\Pi = \begin{pmatrix} \pi_{11} & \dots & \pi_{1M} \\ \vdots & \ddots & \vdots \\ \pi_{M1} & \dots & \pi_{MM} \end{pmatrix}$$

Note that Π_{ij}^k then gives the probability that $s_{t+k} = s_j$ given that $s_t = s_i$. In the long run, we obtain the steady state equivalent for a Markov chain: the *invariant distribution* or *stationary distribution*

Definition 7 *A stationary distribution for a Markov chain is a distribution π such that*

1. $\pi_j \geqslant 0$ *for all* $j = 1, \ldots, M$
2. $\sum_{j=1}^{M} \pi_j = 1$
3. $\pi = \Pi\pi$

Gaussian–quadrature approach to discretization Tauchen and Hussey [1991] provide a simple way to discretize VAR processes relying on gaussian quadrature.[3] I will only present the case of an AR(1) process of the form

$$s_{t+1} = \rho s_t + (1-\rho)\overline{s} + \varepsilon_{t+1}$$

where $\varepsilon_{t+1} \rightsquigarrow \mathcal{N}(0, \sigma^2)$. This implies that

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left\{ -\frac{1}{2} \left(\frac{s_{t+1} - \rho s_t - (1-\rho)\overline{s}}{\sigma} \right)^2 \right\} \mathrm{d}s_{t+1} = \int f(s_{t+1}|s_t) \mathrm{d}s_{t+1} = 1$$

which illustrates the fact that s is a continuous random variable. Tauchen and Hussey [1991] propose to replace the integral by

$$\int \Phi(s_{t+1}; s_t, \overline{s}) f(s_{t+1}|\overline{s}) \mathrm{d}s_{t+1} \equiv \int \frac{f(s_{t+1}|s_t)}{f(s_{t+1}|\overline{s})} f(s_{t+1}|\overline{s}) \mathrm{d}s_{t+1} = 1$$

where $f(s_{t+1}|\overline{s})$ denotes the density of s_{t+1} conditional on the fact that $s_t = \overline{s}$ (which amounts to the unconditional density), which in our case implies that

$$\Phi(s_{t+1}; s_t, \overline{s}) \equiv \frac{f(s_{t+1}|s_t)}{f(s_{t+1}|\overline{s})} = \exp\left\{ -\frac{1}{2} \left[\left(\frac{s_{t+1} - \rho s_t - (1-\rho)\overline{s}}{\sigma} \right)^2 - \left(\frac{s_{t+1} - \overline{s}}{\sigma} \right)^2 \right] \right\}$$

[3] Note that we already discussed this issue in lectures notre # 4.

then we can use the standard linear transformation and impose $z_t = (s_t - \overline{s})/(\sigma\sqrt{2})$ to get

$$\frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} \exp\left\{-\left((z_{t+1} - \rho z_t)^2 - z_{t+1}^2 j\right)\right\} \exp\left(-z_{t+1}^2\right) \mathrm{d}z_{t+1}$$

for which we can use a Gauss–Hermite quadrature. Assume then that we have the quadrature nodes z_i and weights ω_i, $i = 1, \ldots, n$, the quadrature leads to the formula

$$\frac{1}{\sqrt{\pi}} \sum_{j=1}^{n} \omega_j \Phi(z_j; z_i; \overline{x}) \simeq 1$$

in other words we might interpret the quantity $\omega_j \Phi(z_j; z_i; \overline{x})$ as an "estimate" $\widehat{\pi}_{ij} \equiv \text{Prob}(s_{t+1} = s_j | s_t = s_i)$ of the transition probability from state i to state j, but remember that the quadrature is just an approximation such that it will generally be the case that $\sum_{j=1}^{n} \widehat{\pi}_{ij} = 1$ will not hold exactly. Tauchen and Hussey therefore propose the following modification:

$$\widehat{\pi}_{ij} = \frac{\omega_j \Phi(z_j; z_i; \overline{s})}{\sqrt{\pi}\varsigma_i}$$

where $\varsigma_i = \frac{1}{\sqrt{\pi}} \sum_{j=1}^{n} \omega_j \Phi(z_j; z_i; \overline{x})$.

MATLAB CODE: DISCRETIZATION OF AR(1)

```
function [s,p]=tauch_huss(xbar,rho,sigma,n)
% xbar  : mean of the x process
% rho   : persistence parameter
% sigma : volatility
% n     : number of nodes
%
% returns the states s and the transition probabilities p
%

[xx,wx] = gauss_herm(n);        % nodes and weights for s
s       = sqrt(2)*s*xx+mx;      % discrete states
x       = xx(:,ones(n,1));
z       = x';
w       = wx(:,ones(n,1))';
%
% computation
%
p       = (exp(z.*z-(z-rx*x).*(z-rx*x)).*w)./sqrt(pi);
sx      = sum(p')';
p       = p./sx(:,ones(n,1));
```

7.3.3 Value iteration

As in the deterministic case, the convergence of the simple value function iteration procedure is insured by the contraction mapping theorem. This is however a bit more subtil than that as we have to deal with the convergence of a probability measure, which goes far beyond this introduction to dynamic programming.[4] The algorithm is basically the same as in the deterministic case up to the delicate point that we have to deal with the expectation. The algorithm then writes as follows

1. Decide on a grid, $\mathscr{X}$, of admissible values for the state variable x

   $$\mathscr{X} = \{x_1, \ldots, x_N\}$$

   and the shocks, s

   $$\mathscr{S} = \{s_1, \ldots, s_M\} \text{ together with the transition matrix } \Pi = (\pi_{ij})$$

   formulate an initial guess for the value function $V_0(x)$ and choose a stopping criterion $\varepsilon > 0$.

2. For each $x_\ell \in \mathscr{X}$, $\ell = 1, \ldots, N$, and $s_k \in \mathscr{S}$, $k = 1, \ldots, M$ compute

   $$V_{i+1}(x_\ell, s_k) = \max_{\{x' \in \mathscr{X}\}} u(y(x_\ell, s_k, x'), x_\ell, s_k) + \beta \sum_{j=1}^{M} \pi_{kj} V_i(x', s'_j)$$

3. If $\|V_{i+1}(x, s) - V_i(x, s)\| < \varepsilon$ go to the next step, else go back to 2.

4. Compute the final solution as

   $$y^\star(x, s) = y(x, x'(s, s), s)$$

As in the deterministic case, I will illustrate the method relying on the optimal growth model. In order to better understand the algorithm, let us consider a simple example and go back to the optimal growth model, with

$$u(c) = \frac{c^{1-\sigma} - 1}{1 - \sigma}$$

[4] The interested reader should then refer to Lucas et al. [1989] chapter 9.

and

$$k' = \exp(a)k^{\alpha} - c + (1-\delta)k$$

where $a' = \rho a + \varepsilon'$. Then the Bellman equation writes

$$V(k,a) = \max_{c} \frac{c^{1-\sigma}-1}{1-\sigma} + \beta \int V(k',a')\mathrm{d}\mu(a'|a)$$

From the law of motion of capital we can determine consumption as

$$c = \exp(a)k^{\alpha} + (1-\delta)k - k'$$

such that plugging this results in the Bellman equation, we have

$$V(k,a) = \max_{k'} \frac{(\exp(a)k^{\alpha} + (1-\delta)k - k')^{1-\sigma}-1}{1-\sigma} + \beta \int V(k',a')\mathrm{d}\mu(a'|a)$$

A first problem that we encounter is that we would like to be able to evaluate the integral involved by the rational expectation. We therefore have to discretize the shock. Here, I will consider that the technology shock can be accurately approximated by a 2 state markov chain, such that a can take on 2 values a_1 and a_2 ($a_1 < a_2$). I will also assume that the transition matrix is symmetric, such that

$$\Pi = \begin{pmatrix} \pi & 1-\pi \\ 1-\pi & \pi \end{pmatrix}$$

a_1, a_2 and π are selected such that the process reproduces the conditional first and second order moments of the AR(1) process

First order moments

$$\begin{aligned} \pi a_1 + (1-\pi)a_2 &= \rho a_1 \\ (1-\pi)a_1 + \pi a_2 &= \rho a_2 \end{aligned}$$

Second order moments

$$\begin{aligned} \pi a_1^2 + (1-\pi)a_2^2 - (\rho a_1)^2 &= \sigma_{\varepsilon}^2 \\ (1-\pi)a_1^2 + \pi a_2^2 - (\rho a_2)^2 &= \sigma_{\varepsilon}^2 \end{aligned}$$

From the first two equations we get $a_1 = -a_2$ and $\pi = (1+\rho)/2$. Plugging these last two results in the two last equations, we get $a_1 = \sqrt{\sigma_{\varepsilon}^2/(1-\rho^2)}$.

Hence, we will actually work with a value function of the form

$$V(k, a_k) = \max_c \frac{c^{1-\sigma} - 1}{1 - \sigma} + \beta \sum_{j=1}^{2} \pi_{kj} V(k', a'_j)$$

Now, let us define a grid of N feasible values for k such that we have

$$\mathscr{K} = \{k_1, \ldots, k_N\}$$

and an initial value function $V_0(k)$ — that is a vector of N numbers that relate each k_ℓ to a value. Note that this may be anything we want as we know — by the contraction mapping theorem — that the algorithm will converge. But, if we want it to converge fast enough it may be a good idea to impose a good initial guess. Finally, we need a stopping criterion.

In figures 7.9 and 7.10, we report the value function and the decision rules obtained for the stochastic optimal growth model with $\alpha = 0.3$, $\beta = 0.95$, $\delta = 0.1$, $\sigma = 1.5$, $\rho = 0.8$ and $\sigma_\varepsilon = 0.12$. The grid for the capital stock is composed of 1000 data points ranging from 0.2 to 6. The algorithm then converges in 192 iterations when the stopping criterion is $\varepsilon = 1e^{-6}$.

Figure 7.9: Stochastic OGM (Value function, Value iteration)

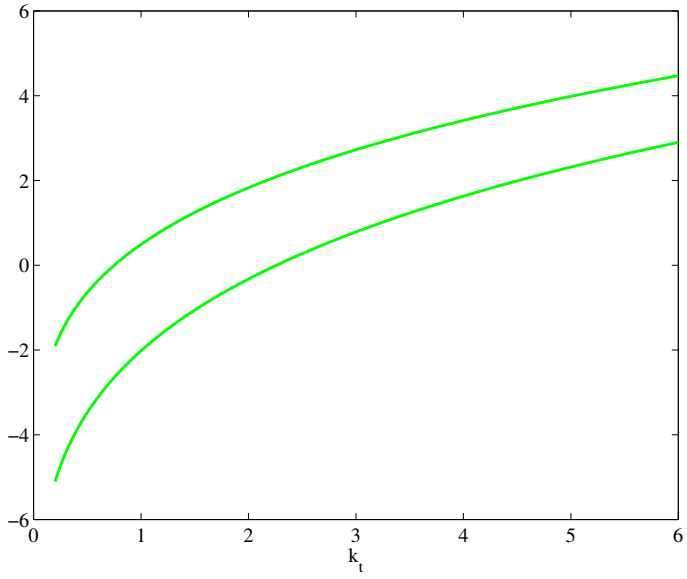
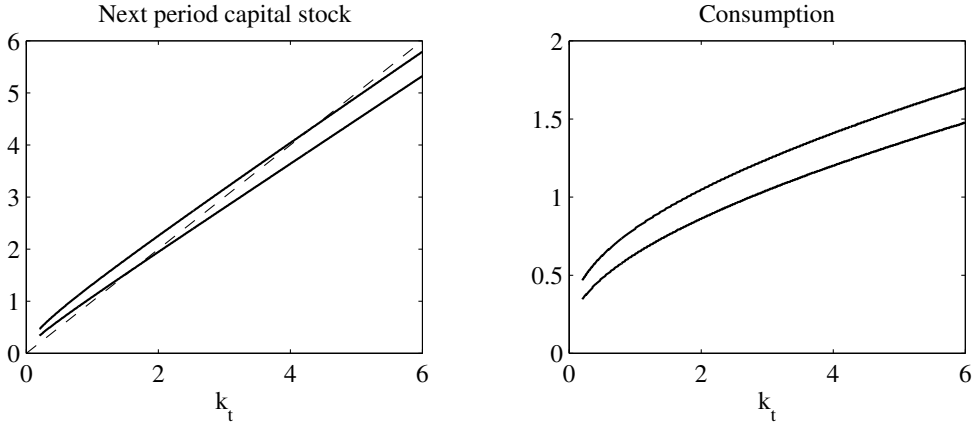


Figure 7.10: Stochastic OGM (Decision rules, Value iteration)



MATLAB CODE: VALUE FUNCTION ITERATION

```
sigma   = 1.50;                    % utility parameter
delta   = 0.10;                    % depreciation rate
beta    = 0.95;                    % discount factor
alpha   = 0.30;                    % capital elasticity of output
rho     = 0.80;                    % persistence of the shock
se      = 0.12;                    % volatility of the shock

nbk     = 1000;                    % number of data points in the grid
nba     = 2;                       % number of values for the shock
crit    = 1;                       % convergence criterion
epsi    = 1e-6;                    % convergence parameter
%
% Discretization of the shock
%
p       = (1+rho)/2;
PI      = [p 1-p;1-p p];
se      = 0.12;
ab      = 0;
a1      = exp(-se*se/(1-rho*rho));
a2      = exp(se*se/(1-rho*rho));
A       = [a1 a2];
%
% Discretization of the state space
%
kmin    = 0.2;
kmax    = 6;
kgrid   = linspace(kmin,kmax,nbk)';
c       = zeros(nbk,nba);
util    = zeros(nbk,nba);
v       = zeros(nbk,nba);
Tv      = zeros(nbk,nba);

while crit>epsi;
   for i=1:nbk
      for j=1:nba;
         c           = A(j)*k(i)^alpha+(1-delta)*k(i)-k;
         neg         = find(c<=0);
         c(neg)      = NaN;
         util(:,j)   = (c.^(1-sigma)-1)/(1-sigma);
         util(neg,j) = -1e12;
      end
      [Tv(i,:),dr(i,:)] = max(util+beta*(v*PI));
   end;
   crit = max(max(abs(Tv-v)));
   v        = Tv;
   iter = iter+1;
end
```

```
kp      = k(dr);
for j=1:nba;
   c(:,j)   = A(j)*k.^alpha+(1-delta)*k-kp(:,j);
end
```

7.3.4 Policy iterations

As in the deterministic case, we may want to accelerate the simple value iteration using Howard improvement. The stochastic case does not differ that much from the deterministic case, apart from the fact that we now have to deal with different decision rules, which implies the computation of different Q matrices. The algorithm may be described as follows

1. Set an initial feasible set of decision rule for the control variable $y = f_0(x, s_k)$, $k = 1, \ldots, M$ and compute the value associated to this guess, assuming that this rule is operative forever, taking care of the fact that $x_{t+1} = h(x_t, y_t, s_t) = h(x_t, f_i(x_t, s_t), s_t)$ with $i = 0$. Set a stopping criterion $\varepsilon > 0$.

2. Find a new policy rule $y = f_{i+1}(x, s_k)$, $k = 1, \ldots, M$, such that

$$f_{i+1}(x, s_k) \in \underset{y}{\text{Argmax}}\, u(y, x, s_k) + \beta \sum_{j=1}^{M} \pi_{kj} V(x', s'_j)$$

   with $x' = h(x, f_i(x, s_k), s_k)$

3. check if $\|f_{i+1}(x, s) - f_i(x, s)\| < \varepsilon$, if yes then stop, else go back to 2.

When computing the value function we actually have to solve a linear system of the form

$$V_{i+1}(x_\ell, s_k) = u(f_{i+1}(x_\ell, s_k), x_\ell, s_k) + \beta \sum_{j=1}^{M} \pi_{kj} V_{i+1}(h(x_\ell, f_{i+1}(x_\ell, s_k), s_k), s'_j)$$

for $V_{i+1}(x_\ell, s_k)$ (for all $x_\ell \in \mathscr{X}$ and $s_k \in \mathscr{S}$), which may be rewritten as

$$V_{i+1}(x_\ell, s_k) = u(f_{i+1}(x_\ell, s_k), x_\ell, s_k) + \beta \Pi_{k.} Q V_{i+1}(x_\ell, .) \quad \forall x_\ell \in \mathscr{X}$$

where Q is an $(N \times N)$ matrix

$$Q_{\ell j} = \begin{cases} 1 \text{ if } x'_j \equiv h(f_{i+1}(x_\ell), x_\ell) = x_\ell \\ 0 \text{ otherwise} \end{cases}$$

Note that although it is a big matrix, Q is sparse, which can be exploited in solving the system, to get

$$V_{i+1}(x, s) = (I - \beta \Pi \otimes Q)^{-1} u(f_{i+1}(x, s), x, s)$$

We apply this algorithm to the same optimal growth model as in the previous section and report the value function and the decision rules at convergence in figures 7.11 and 7.12. The algorithm converges in only 17 iterations, starting from the same initial guess and using the same parameterization!

Figure 7.11: Stochastic OGM (Value function, Policy iteration)

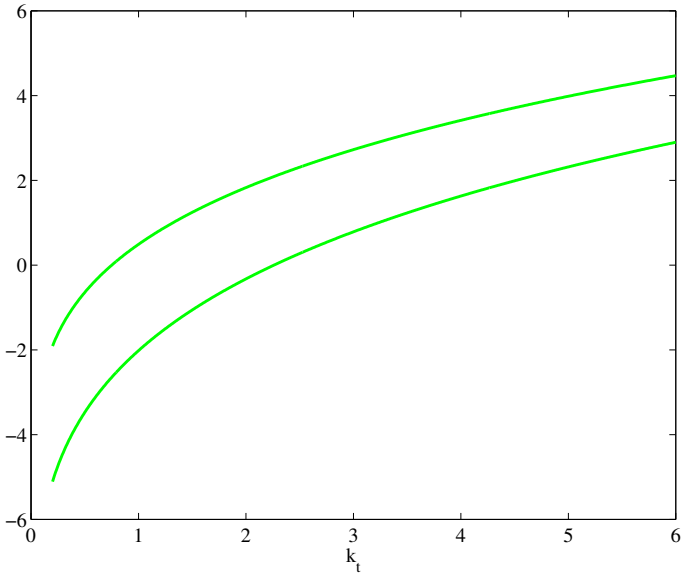
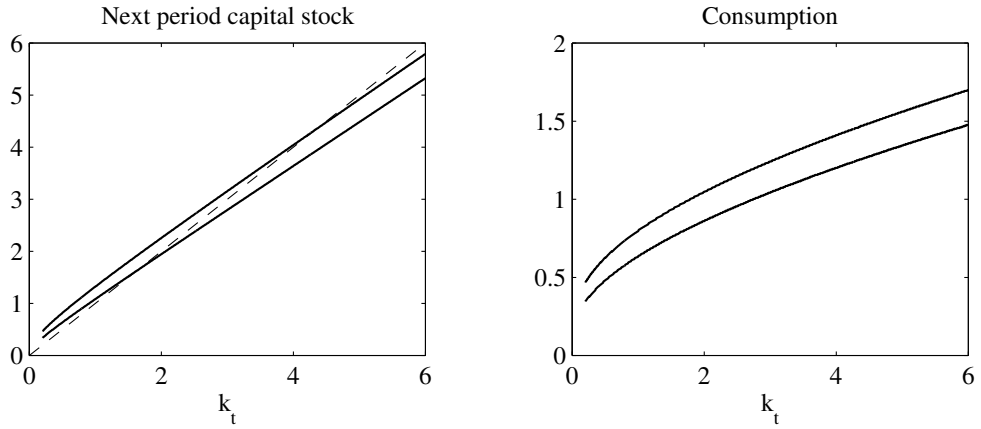


Figure 7.12: Stochastic OGM (Decision rules, Policy iteration)



MATLAB CODE: POLICY ITERATION

```
sigma   = 1.50;                 % utility parameter
delta   = 0.10;                 % depreciation rate
beta    = 0.95;                 % discount factor
alpha   = 0.30;                 % capital elasticity of output
rho     = 0.80;                 % persistence of the shock
se      = 0.12;                 % volatility of the shock

nbk     = 1000;                 % number of data points in the grid
nba     = 2;                    % number of values for the shock
crit    = 1;                    % convergence criterion
epsi    = 1e-6;                 % convergence parameter
%
% Discretization of the shock
%
p       = (1+rho)/2;
PI      = [p 1-p;1-p p];
se      = 0.12;
ab      = 0;
a1      = exp(-se*se/(1-rho*rho));
a2      = exp(se*se/(1-rho*rho));
A       = [a1 a2];
%
% Discretization of the state space
%
kmin    = 0.2;
kmax    = 6;
kgrid   = linspace(kmin,kmax,nbk)';
c       = zeros(nbk,nba);
util    = zeros(nbk,nba);
```

```
v       = zeros(nbk,nba);
Tv      = zeros(nbk,nba);
Ev      = zeros(nbk,nba);             % expected value function
dr0     = repmat([1:nbk]',1,nba);     % initial guess
dr      = zeros(nbk,nba);             % decision rule (will contain indices)
%
% Main loop
%
while crit>epsi;
   for i=1:nbk
      for j=1:nba;
         c           = A(j)*kgrid(i)^alpha+(1-delta)*kgrid(i)-kgrid;
         neg         = find(c<=0);
         c(neg)      = NaN;
         util(:,j)   = (c.^(1-sigma)-1)/(1-sigma);
         util(neg,j) = -inf;
         Ev(:,j)     = v*PI(j,:)';
      end
         [v1,dr(i,:)]= max(u+beta*Ev);
   end;
   %
   % decision rules
   %
   kp  = kgrid(dr);
   Q   = sparse(nbk*nba,nbk*nba);
   for j=1:nba;
      c = A(j)*kgrid.^alpha+(1-delta)*kgrid-kp(:,j);
      %
      % update the value
      %
      util(:,j)= (c.^(1-sigma)-1)/(1-sigma);

      Q0  = sparse(nbk,nbk);
      for i=1:nbk;
         Q0(i,dr(i,j)) = 1;
      end
      Q((j-1)*nbk+1:j*nbk,:) = kron(PI(j,:),Q0);
   end
   Tv  = (speye(nbk*nba)-beta*Q)\util(:);
   crit= max(max(abs(kp-kp0)));
   v   = reshape(Tv,nbk,nba);
   kp0 = kp;
   iter= iter+1;
end
c=zeros(nbk,nba);
for j=1:nba;
   c(:,j)   = A(j)*kgrid.^alpha+(1-delta)*kgrid-kp(:,j);
end
```

7.4 How to simulate the model?

At this point, no matter whether the model is deterministic or stochastic, we have a solution which is actually a collection of data points. There are several ways to simulate the model

- One may simply use the pointwise decision rule and iterate on it
- or one may use an interpolation scheme (either least square methods, spline interpolations ...) in order to obtain a continuous decision rule

Here I will pursue the second route.

7.4.1 The deterministic case

After having solved the model by either value iteration or policy iteration, we have in hand a collection of points, $\mathscr{X}$, for the state variable and a decision rule that relates the control variable to this collection of points. In other words we have *Lagrange data* which can be used to get a continuous decision rule. This decision rule may then be used to simulate the model.

In order to illustrate this, let's focus, once again on the deterministic optimal growth model and let us assume that we have solved the model by either value iteration or policy iteration. We therefore have a collection of data points for the capital stock $\mathscr{K} = \{k_1, \ldots, k_N\}$, which form our grid, and the associated decision rule for the next period capital stock $k_i' = h(k_i)$, $i = 1, \ldots, N$, and the consumption level which can be deduced from the next period capital stock using the law of motion of capital together with the resource constraint. Also note that we have the upper and lower bound of the grid $k_1 = \underline{k}$ and $k_N = \overline{k}$. We will use Chebychev polynomials to approximate the rule in order to avoid multicolinearity problems in the least square algorithm. In other words we will get a decision rule of the form

$$k' = \sum_{\ell=0}^{n} \theta_\ell T_\ell(\varphi(k))$$

where $\varphi(k)$ is a linear transformation that maps $[\underline{k}, \overline{k}]$ into [-1;1]. The algorithm works as follows

1. Solve the model by value iteration or policy iteration, such that we have a collection of data points for the capital stock $\mathscr{K} = \{k_1, \ldots, k_N\}$ and associated decision rule for the next period capital stock $k_i' = h(k_i)$, $i = 1, \ldots, N$. Choose a maximal order, n, for the Chebychev polynomials approximation.

2. Apply the transformation

$$x_i = 2\frac{k_i - \underline{k}}{\overline{k} - \underline{k}} - 1$$

   to the data and compute $T_\ell(x_i),\ \ell = 1, \ldots, n,$ and $i = 1 \ldots, N$.

3. Perform the least square approximation such that

$$\Theta = (T_\ell(x)' T_\ell(x))^{-1} T_\ell(x) h(k)$$

For the parameterization we have been using so far, setting

$$\varphi(k) = 2\frac{k - \underline{k}}{\overline{k} - \underline{k}} - 1$$

and for an approximation of order 7, we obtain

$$\theta = \{2.6008, 2.0814, -0.0295, 0.0125, -0.0055, 0.0027, -0.0012, 0.0007, \}$$

such that the decision rule for the capital stock can be approximated as

$$\begin{aligned} k_{t+1} \quad = \quad & 2.6008 + 2.0814 T_1(\varphi(k_t)) - 0.0295 T_2(\varphi(k_t)) + 0.0125 T_3(\varphi(k_t)) \\ & -0.0055 T_4(\varphi(k_t)) + 0.0027 T_5(\varphi(k_t)) - 0.0012 T_6(\varphi(k_t)) + 0.0007 T_7(\varphi(k_t)) \end{aligned}$$

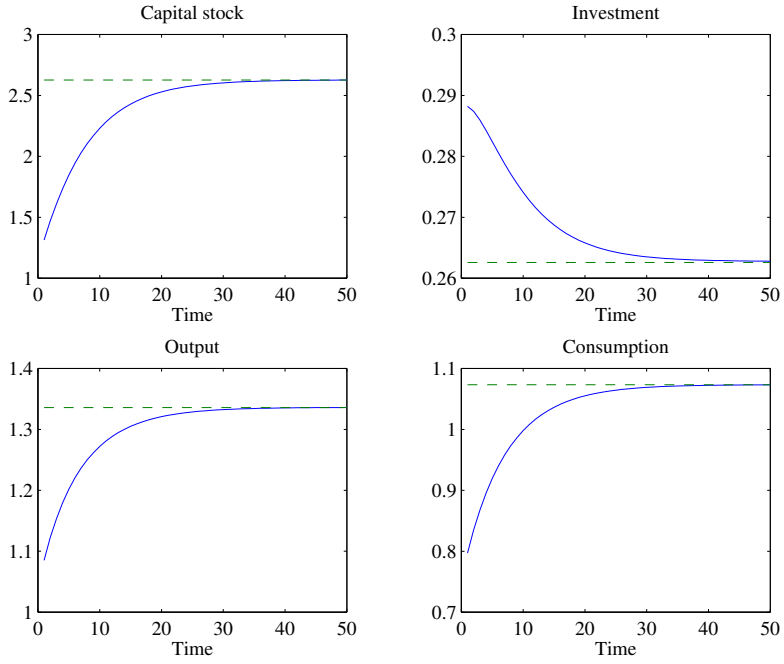
Then the model can be used in the usual way such that in order to get the transitional dynamics of the model we just have to iterate on this backward

looking finite–differences equation. Consumption, output and investment can be computed using their definition

$$\begin{aligned} i_t &= k_{t+1} - (1-\delta)k_t \\ y_t &= k_t^{\alpha} \\ c_t &= y_t - i - t \end{aligned}$$

For instance, figure 7.13 reports the transitional dynamics of the model when the economy is started 50% below its steady state. In the figure, the plain line represents the dynamics of each variable, and the dashed line its steady state level.

Figure 7.13: Transitional dynamics (OGM)



MATLAB CODE: TRANSITIONAL DYNAMICS

```
n       = 8;                            % Order of approximation+1
transk  = 2*(kgrid-kmin)/(kmax-kmin)-1; % transform the data
%
% Computes the matrix of Chebychev polynomials
%
```

```
Tk          = [ones(nbk,1) transk];
for i=3:n;
   Tk=[Tk 2*transk.*Tk(:,i-1)-Tk(:,i-2)];
end
b=Tk\kp;                                  % Performs OLS

k0  = 0.5*ks;                             % initial capital stock
nrep= 50;                                 % number of periods
k   = zeros(nrep+1,1);                    % initialize dynamics
%
% iteration loop
%
k(1)= k0;
for t=1:nrep;
   trkt = 2*(k(t)-kmin)/(kmax-kmin)-1;
   Tk   = [1 trkt];
   for i=3:n;
      Tk= [Tk 2*trkt.*Tk(:,i-1)-Tk(:,i-2)];
   end
   k(t+1)=Tk*b;
end
y   = k(1:nrep).^alpha;
i   = k(2:nrep+1)-(1-delta)*k(1:nrep);
c   = y-i;
```

Note: At this point, I assume that the model has already been solved and that the capital grid has been stored in `kgrid` and the decision rule for the next period capital stock is stored in `kp`.

Another way to proceed would have been to use spline approximation, in which case the matlab code would have been (for the simulation part)

Matlab Code: OGM simulation (cubic spline interpolation

```
k(1)= k0;
for t=1:nrep;
   k(t+1)=interp1(grid,kp,k(t),'cubic');
end
y   = k(1:nrep).^alpha;
i   = k(2:nrep+1)-(1-delta)*k(1:nrep);
c   = y-i;
```

7.4.2 The stochastic case

As in the deterministic case, it will often be a good idea to represent the solution as a continuous decision rule, such that we should apply the same

methodology as before. The only departure from the previous experiment is that since we have approximated the stochastic shock by a Markov chain, we have as many decision rule as we have states in the Markov chain. For instance, in the optimal growth model we solved in section 7.3, we have 2 decision rules which are given by[5]

$$\begin{aligned} k_{1,t+1} &= 2.8630 + 2.4761T_1(\varphi(k_t)) - 0.0211T_2(\varphi(k_t)) + 0.0114T_3(\varphi(k_t)) \\ &\quad -0.0057T_4(\varphi(k_t)) + 0.0031T_5(\varphi(k_t)) - 0.0014T_6(\varphi(k_t)) + 0.0009T_7(\varphi(k_t)) \end{aligned}$$

$$\begin{aligned} k_{2,t+1} &= 3.2002 + 2.6302T_1(\varphi(k_t)) - 0.0543T_2(\varphi(k_t)) + 0.0235T_3(\varphi(k_t)) \\ &\quad -0.0110T_4(\varphi(k_t)) + 0.0057T_5(\varphi(k_t)) - 0.0027T_6(\varphi(k_t)) + 0.0014T_7(\varphi(k_t)) \end{aligned}$$

The only difficulty we have to deal with in the stochastic case is to simulate the Markov chain. Simulating a series of length T for the markov chain can be achieved easily

1. Compute the cumulative distribution of the markov chain (Π^c) — *i.e.* the cumulative sum of each row of the transition matrix. Hence, Π^c_{ij} corresponds to the the probability that the economy is in a state lower of equal to j given that it was in state i in period $t-1$.

2. Set the initial state and simulate T random numbers from a uniform distribution: $\{p_t\}_{t=1}^T$. Note that these numbers, drawn from a $\mathscr{U}_{[0;1]}$ can be interpreted as probabilities.

3. Assume the economy was in state i in $t-1$, find the index j such that

$$\Pi^c_{i(j-1)} < p_t \leqslant \Pi^c_{ij}$$

then s_j is the state of the economy in period t

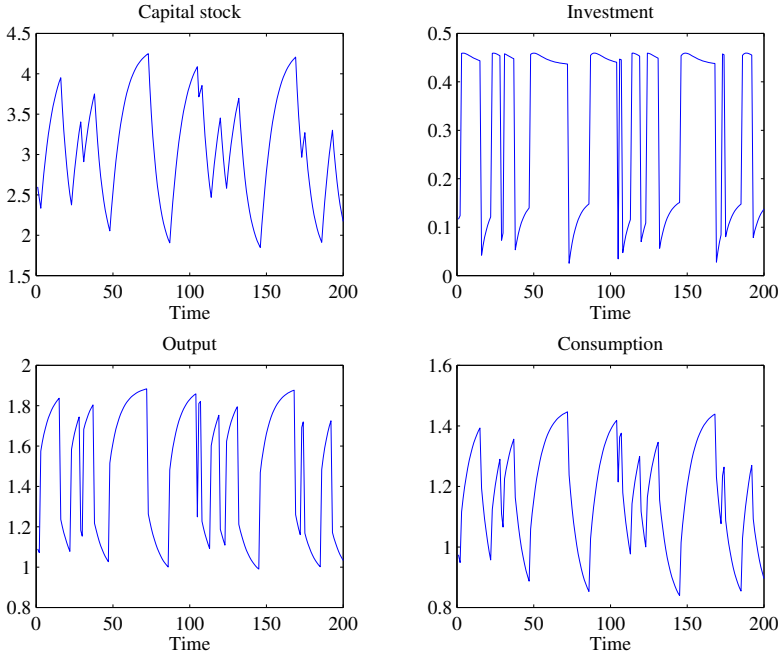
[5]The code to generate these two rules is exactly the same as for the deterministic version of the model since, when computing `b=Tk\kp` matlab will take care of the fact that `kp` is a $(N \times 2)$ matrix, such that `b` will be a $((n+1) \times 2)$ matrix where n is the approximation order.

MATLAB CODE: SIMULATION OF MARKOV CHAIN

```
s           = [-1;1];        % 2 possible states
PI          = [0.9 0.1;
               0.1 0.9];     % Transition matrix
s0          = 1;             % Initial state
T           = 1000;          % Length of the simulation
p           = rand(T,1);     % Simulated distribution
j           = zeros(T,1);
j(1)        = s0;            % Initial state
%
for k=2:n;
  j(k)=find(((sim(k)<=cum_PI(state(k-1),2:cpi+1))&...
              (sim(k)>cum_PI(state(k-1),1:cpi))));
end;
chain=s(j);                  % simulated values
```

We are then in position to simulate our optimal growth model. Figure 7.14 reports a simulation of the optimal growth model we solved before.

Figure 7.14: Simulated OGM



MATLAB CODE: SIMULATING THE OGM

```
n       = 8;                              % Order of approximation+1
transk  = 2*(kgrid-kmin)/(kmax-kmin)-1; % transform the data
%
% Computes approximation
%
Tk      = [ones(nbk,1) transk];
for i=3:n;
   Tk=[Tk 2*transk.*Tk(:,i-1)-Tk(:,i-2)];
end
b=Tk\kp;
%
% Simulate the markov chain
%
long    = 200;                            % length of simulation
[a,id]  = markov(PI,A,long,1);            % markov is a function:
                                          % a contains the values for the shock
                                          % id the state index

k0=2.6;                                   % initial value for the capital stock
k=zeros(long+1,1);                        % initialize
y=zeros(long,1);                          % some matrices
k(1)=k0;                                  % initial capital stock
for t=1:long;
    % Prepare the approximation for the capital stock
    trkt    = 2*(k(t)-kmin)/(kmax-kmin)-1;
    Tk          = [1 trkt];
    for i   = 3:n;
        Tk  = [Tk 2*trkt.*Tk(:,i-1)-Tk(:,i-2)];
    end

    k(t+1)  = Tk*b(:,id(t));              % use the appropriate decision rule
    y(t)    = A(id(t))*k(t).^alpha;       % computes output
end

i=k(2:long+1)-(1-delta)*k(1:long);        % computes investment
c=y-i;                                    % computes consumption
```

Note: At this point, I assume that the model has already been solved and that the capital grid has been stored in `kgrid` and the decision rule for the next period capital stock is stored in `kp`.

7.5 Handling constraints

In this example I will deal the problem of a household that is constraint on her borrowing and who faces uncertainty on the labor market. This problem has been extensively studied by Ayiagari [1994].

We consider the case of a household who determines her consumption/saving plans in order to maximize her lifetime expected utility, which is characterized by the function: [6]

$$E_t \sum_{\tau=0}^{\infty} \beta^{\tau} \left(\frac{c_{t+\tau}^{1-\sigma} - 1}{1-\sigma} \right) \text{ with } \sigma \in (0,1) \cup (1,\infty) \tag{7.9}$$

where $0 < \beta < 1$ is a constant discount factor, c denotes the consumption bundle. In each and every period, the household enters the period with a level of asset a_t carried from the previous period, for which she receives an constant real interest rate r. She may also be employed and receives the aggregate real wage w or may be unemployed in which case she just receives unemployment benefits, b, which are assumed to be a fraction of the real wage: $b = \gamma w$ where γ is the replacement ratio. These revenus are then used to consume and purchase assets on the financial market. Therefore, the budget constraint in t is given by

$$a_{t+1} = (1+r)a_t + \omega_t - c_t \tag{7.10}$$

where

$$\omega_t = \begin{cases} w & \text{if the household is employed} \\ \gamma w & \text{otherwise} \end{cases} \tag{7.11}$$

We assume that the household's state on the labor market is a random variable. Further the probability of being (un)employed in period t is greater if the household was also (un)employed in period $t-1$. In other words, the state on the labor market, and therefore ω_t, can be modeled as a 2 state Markov chain with transition matrix Π. In addition, the household is submitted to a

[6] $E_t(.)$ denotes mathematical conditional expectations. Expectations are conditional on information available at the beginning of period t.

borrowing contraint that states that she cannot borrow more than a threshold level $a^\star$

$$a_{t+1} \geqslant a^\star \tag{7.12}$$

In this case, the Bellman equation writes

$$V(a_t, \omega_t) = \max_{c_t \geqslant 0} u(c_t) + \beta E_t V(a_{t+1}, \omega_{t+1})$$

s.t. (7.10)–(7.12), which may be solved by one of the methods we have seen before. Anyway, implementing any of these method taking into account the constraint is straightforward as, plugging the budget constraint into the utility function, the Bellman equation can be rewritten

$$V(a_t, \omega_t) = \max_{a_{t+1}} u((1+r)a_t + \omega_t - a_{t+1}) + \beta E_t V(a_{t+1}, \omega_{t+1})$$

s.t. (7.11)–(7.12).

But, since the borrowing constraint should be satisfied in each and every period, we have

$$V(a_t, \omega_t) = \max_{a_{t+1} \geqslant a^\star} u((1+r)a_t + \omega_t - a_{t+1}) + \beta E_t V(a_{t+1}, \omega_{t+1})$$

In other words, when defining the grid for a, one just has to take care of the fact that the minimum admissible value for a is $a^\star$. Positivity of consumption — which should be taken into account when searching the optimal value for a_{t+1} — finally imposes $a_{t+1} < (1+r)a_t + \omega_t$, such that the Bellman equation writes

$$V(a_t, \omega_t) = \max_{\{a^\star \leqslant a_{t+1} \leqslant (1+r)a_t + \omega_t\}} u((1+r)a_t + \omega_t - a_{t+1}) + \beta E_t V(a_{t+1}, \omega_{t+1})$$

The matlab code `borrow.m` solves and simulate this problem.

Bibliography

Ayiagari, R.S., Uninsured Idiosyncratic Risk and Aggregate Saving, *Quarterly Journal of Economics*, 1994, *109* (3), 659–684.

Bertsekas, D., *Dynamic Programming and Stochastic Control*, New York: Academic Press, 1976.

Judd, K. and A. Solnick, *Numerical Dynamic Programming with Shape–Preserving Splines*, Manuscript, Hoover Institution 1994.

Judd, K.L., *Numerical methods in economics*, Cambridge, Massachussets: MIT Press, 1998.

Lucas, R., N. Stokey, and E. Prescott, *Recursive Methods in Economic Dynamics*, Cambridge (MA): Harvard University Press, 1989.

Tauchen, G. and R. Hussey, Quadrature Based Methods for Obtaining Approximate Solutions to Nonlinear Asset Pricing Models, *Econometrica*, 1991, *59* (2), 371–396.

Index

Contents

List of Figures

List of Tables