Chapter 6

Infinite State Space

In this chapter we begin working with stochastic systems on infinite state space. While a completely rigorous treatment of this area requires measure theory (chapter 7 and onward), we can build a good understanding of the key topics (dynamics, optimization, etc.) by heuristic arguments, simulation, and analogies with the finite case. Along the way we will meet some more challenging programming problems.

6.1 First Steps

In this section we study dynamics for stochastic recursive sequences (SRSs) taking values in $\mathbb{R}$. Our main interest is in tracking the evolution of probabilities over time, as represented by the marginal distributions of the process. We will also look at stationary distributions—the infinite state analogue of the stationary distributions discussed in §4.3.1—and how to calculate them.

6.1.1 Basic Models and Simulation

Our basic model is as follows: Let the state space S be a subset of $\mathbb{R}$ and let $Z \subset \mathbb{R}$. Let $F \colon S \times Z \to S$ be a given function, and consider the SRS

$$X_{t+1} = F(X_t, W_{t+1}), \quad X_0 \sim \psi, \quad (W_t)_{t \geq 1} \overset{\text{IID}}{\sim} \phi \tag{6.1}$$

Here $(W_t)_{t \geq 1} \overset{\text{IID}}{\sim} \phi$ means that $(W_t)_{t \geq 1}$ is an IID sequence of shocks with *cumulative* distribution function ϕ. In other words, $\mathbb{P}\{W_t \leq z\} = \phi(z)$ for all $z \in Z$. Likewise ψ is the cumulative distribution function of X_0, and X_0 is independent of $(W_t)_{t \geq 1}$. Note that X_t and W_{t+1} are independent, since X_t depends only on the initial condition and the shocks $W_1, \ldots, W_t$, all of which are independent of W_{t+1}.

Example 6.1.1 Consider a stochastic version of the Solow–Swan growth model, where output is a function f of capital k and a real-valued shock W. The sequence of productivity shocks $(W_t)_{t\geq 1}$ is $\overset{\text{IID}}{\sim} \phi$. Capital at time $t+1$ is equal to that fraction s of output that was saved last period, plus undepreciated capital, giving law of motion

$$k_{t+1} = F(k_t, W_{t+1}) := sf(k_t, W_{t+1}) + (1-\delta)k_t \tag{6.2}$$

Consumption is given by $c_t = (1-s)f(k_t, W_{t+1})$. The production function satisfies $f\colon \mathbb{R}^2_+ \to \mathbb{R}_+$ and $f(k,z) > 0$ whenever $k > 0$ and $z > 0$. For the state space we can choose either $S_0 = \mathbb{R}_+$ or $S = (0,\infty)$, while $Z := (0,\infty)$.

Exercise 6.1.2 Show that if $k \in S_0$ (resp., S) and $z \in Z$, then next period's stock $F(k,z)$ is in S_0 (resp., S).

Example 6.1.3 Let $Z = S = \mathbb{R}$, and consider the smooth transition threshold autoregression (STAR) model

$$X_{t+1} = g(X_t) + W_{t+1}, \quad (W_t)_{t\geq 1} \overset{\text{IID}}{\sim} \phi \tag{6.3}$$

$$g(x) := (\alpha_0 + \alpha_1 x)(1 - G(x)) + (\beta_0 + \beta_1 x)G(x)$$

Here $G\colon S \to [0,1]$ is a smooth transition function, such as the logistic function, satisfying $G' > 0$, $\lim_{x\to-\infty} G(x) = 0$ and $\lim_{x\to\infty} G(x) = 1$.

Code for simulating time series from an arbitrary SRS is given in listing 6.1. The listing defines a class `SRS` implementing the canonical SRS in (6.1).[1] The behavior of the class is similar to that of the class `MC` in listing 4.4 (page 73), and the methods are explained in the doc strings.

Listing 6.2 provides an example of usage. The code creates an instance of `SRS`, corresponding to the Solow model (6.2) when $f(k,W) = k^\alpha W$ and $\ln W_t \sim N(0,\sigma^2)$. The two sample paths generated by this code are shown in figure 6.1.

Exercise 6.1.4 Repeat exercise 5.2.2 on page 109 using the class `SRS`.

Returning to the SRS (6.1), let's consider the distribution of X_t for arbitrary $t \in \mathbb{N}$. This distribution will be denoted by ψ_t, and you can think of it for now as a cumulative distribution function (i.e., $\psi_t(x)$ is the probability that $X_t \leq x$). It is also called the marginal distribution of X_t; conceptually it is equivalent to its discrete state namesake that we met in §4.2.2.

In order to investigate ψ_t via simulation, we need to sample from this distribution. The simplest technique is this: First draw $X_0 \sim \psi$ and generate a sample path stopping at time t. Now repeat the exercise, but with a new set of draws $X_0, W_1, \ldots, W_t$,

[1]The benefit of designing an abstract class for SRSs is code reuse: The class can be used to study any system. Readers are encouraged to add functionality to the class as they do the exercises in the chapter.

Listing 6.1 (srs.py) Simulation of SRSs

```
class SRS:

    def __init__(self, F=None, phi=None, X=None):
        """Represents X_{t+1} = F(X_t, W_{t+1}); W ~ phi.
        Parameters: F and phi are functions, where phi()
        returns a draw from phi. X is a number representing
        the initial condition."""
        self.F, self.phi, self.X = F, phi, X

    def update(self):
        "Update the state according to X = F(X, W)."
        self.X = self.F(self.X, self.phi())

    def sample_path(self, n):
        "Generate path of length n from current state."
        path = []
        for i in range(n):
            path.append(self.X)
            self.update()
        return path
```

Listing 6.2 (testsrs.py) Example application

```
from srs import SRS                  # Import from listing 6.1
from random import lognormvariate

alpha, sigma, s, delta = 0.5, 0.2, 0.5, 0.1
# Define F(k, z) = s k^alpha z + (1 - delta) k
F = lambda k, z: s * (k**alpha) * z + (1 - delta) * k
lognorm = lambda: lognormvariate(0, sigma)

solow_srs = SRS(F=F, phi=lognorm, X=1.0)
P1 = solow_srs.sample_path(500)    # Generate path from X = 1
solow_srs.X = 60                   # Reset the current state
P2 = solow_srs.sample_path(500)    # Generate path from X = 60
```

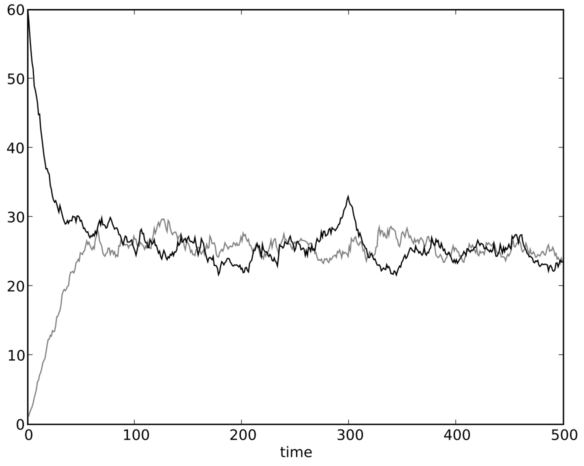

Figure 6.1 Time series plot

leading to a new draw of X_t that is independent of the first. If we do this n times, we get n independent samples $X_t^1, \ldots, X_t^n$ from the target distribution ψ_t. Algorithm 6.1 contains the pseudocode for this operation. Figure 6.2 is a visualization of the algorithm after 3 iterations of the outer loop.

Exercise 6.1.5 Investigate the mean of ψ_t for the Solow–Swan model when $f(k, W) = k^\alpha W$ and $\ln W_t \sim N(0, \sigma^2)$. Carry out a simulation with $k_0 = 1$, $t = 20$, $\delta = 0.1$, $s = 1/2$, $\sigma^2 = 0.2$ and $\alpha = 0.3$. Draw $n = 1000$ samples. Compute $\mathbb{E} k_t$ using the statistic $n^{-1} \sum_{i=1}^n k_t^i$. Explain how theorem 4.3.31 on page 94 justifies your statistic.

Exercise 6.1.6 Repeat exercise 6.1.5, but now setting $s = 3/4$. How does your estimate change? Interpret.

Exercise 6.1.7 Repeat exercise 6.1.5, but now set $k_0 = 5$, $k_0 = 10$, and $k_0 = 20$. To the extent that you can, interpret your results.

Exercise 6.1.8 Repeat exercise 6.1.5, but now set $t = 50$, $t = 100$, and $t = 200$. What happens to your estimates? Interpret.

Exercise 6.1.9 Repeat exercise 6.1.7, but with $t = 200$ instead of $t = 20$. Try to interpret your results.

Algorithm 6.1 Draws from the marginal distribution

```
for $i$ in 1 to $n$ do
    draw $X$ from the initial condition $\psi$
    for $j$ in 1 to $t$ do
        draw $W$ from the shock distribution $\phi$
        set $X = F(X, W)$
    end
    set $X_t^i = X$
end
return $(X_t^1, \ldots, X_t^n)$
```

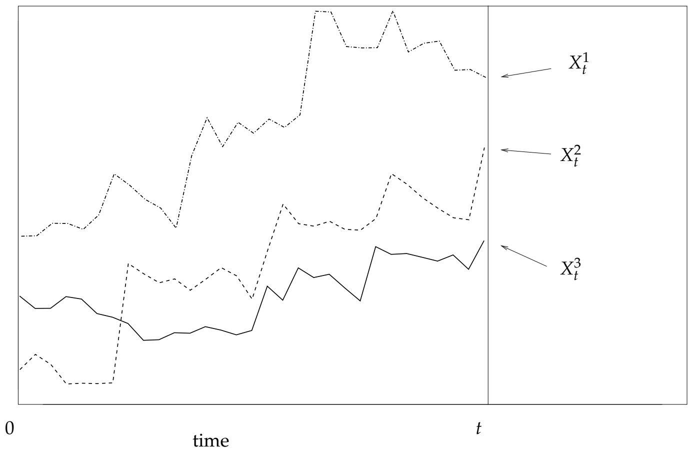

Figure 6.2 Sampling from the marginal distribution

Recall that if $X_1, \ldots, X_n$ is a sample of IID random variables, then the *sample mean* is defined as $\bar{X}_n := \frac{1}{n}\sum_{i=1}^n X_i$, while the *sample variance* is

$$\hat{\sigma}_n^2 := \frac{1}{n-1}\sum_{i=1}^n (X_i - \bar{X}_n)^2$$

Assuming that the second moment of X_i is finite, the central limit theorem and a convergence result often referred to as Slutsky's theorem give

$$\frac{\sqrt{n}(\bar{X}_n - \mathbb{E}X_1)}{\hat{\sigma}_n} \xrightarrow{d} N(0,1) \text{ as } n \to \infty, \text{ where } \hat{\sigma}_n := \sqrt{\hat{\sigma}_n^2}$$

Exercise 6.1.10 Based on this fact, construct a 95% confidence interval for your estimate of $\mathbb{E}k_t$. (Use the parameters from exercise 6.1.5.)

Exercise 6.1.11 Consider the same model as in exercise 6.1.5, but now set $\delta = 1$. The classical golden rule optimization problem is to choose the savings rate s in the Solow–Swan model to maximize steady state consumption. Let's consider the stochastic analogue. The simplest criterion is to maximize *expected* steady state consumption. For this model, by $t = 100$ the distribution of c_j varies little for $j \geq t$ (we'll learn more about this later on). As such, let's consider $\mathbb{E}c_{100}$ as expected steady state consumption. Compute $n = 5,000$ observations of c_{100}, and take the sample average to obtain an approximation of the expectation. Repeat for s in a grid of values in $(0,1)$. Plot the function, and report the maximizer.

6.1.2 Distribution Dynamics

While the mean conveys some information about the random variable k_t, at times we wish to know about the entire (cumulative) distribution ψ_t. How might one go about computing ψ_t by simulation?

The standard method is with the *empirical distribution function*, which, for independent samples $(X_i)_{i=1}^n$ of random variable $X \in \mathbb{R}$, is given by

$$F_n(x) := \frac{1}{n}\sum_{i=1}^n \mathbb{1}\{X_i \leq x\} \qquad (x \in \mathbb{R}) \tag{6.4}$$

Thus $F_n(x)$ is the fraction of the sample that falls below x. The LLN (theorem 4.3.31, page 94) can be used to show that if X has cumulative distribution F, then $F_n(x) \to F(x)$ with probability one for each x as $n \to \infty$.[2] These results formalize the fundamental idea that empirical frequencies converge to probabilities when the draws are independent.

[2] Later we will cover how to do these kinds of proofs. In this case much more can be proved—interested readers should refer to the Glivenko–Cantelli theorem.

Listing 6.3 (`ecdf.py`) Empirical distribution function

```
class ECDF:

    def __init__(self, observations):
        self.observations = observations

    def __call__(self, x):
        counter = 0.0
        for obs in self.observations:
            if obs <= x:
                counter += 1
        return counter / len(self.observations)
```

An implementation of the empirical distribution function is given in listing 6.3, based on a class called `ECDF`.[3] The class is initialized with samples $(X_t)_{t=1}^n$ stored in a sequence (list or tuple) called `observations`, and an instance is created with a call such as `F = ECDF(data)`. The method `__call__` evaluates $F_n(x)$ for a given value of x. (Recall from §2.2.3 that `__call__` is a special method that makes an instance `F` *callable*, so we can evaluate $F_n(x)$ using the simple syntax `F(x)`.) Here is an example:

```
from ecdf import ECDF          # Import from listing 6.3
from random import uniform
samples = [uniform(0, 1) for i in range(10)]
F = ECDF(samples)
F(0.5)  # Returned 0.29
F.observations = [uniform(0, 1) for i in range(1000)]
F(0.5)  # Returned 0.479
```

Figure 6.3 gives four plots of the empirical distribution function corresponding to the time t distribution of the Solow–Swan model. The plots are for $n = 4$, $n = 25$, $n = 100$, and $n = 5,000$. The parameters are $k_0 = 1$, $t = 20$, $\delta = 0.1$, $s = 1/2$, $\sigma^2 = 0.2$, and $\alpha = 0.3$.

Exercise 6.1.12 Add a method to the `ECDF` class that uses Matplotlib to plot the empirical distribution over a specified interval. Replicate the four graphs in figure 6.3 (modulo randomness).

Consider now a variation of our growth model with additional nonlinearities. In

[3]As usual, the code is written for clarity rather than speed. See the text home page for more optimized solutions.

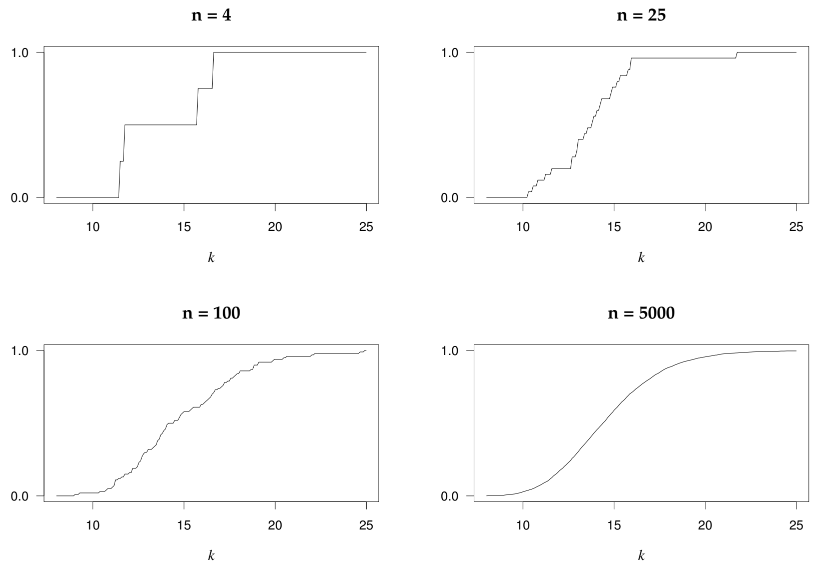

Figure 6.3 Empirical distribution functions

example 4.1.8 (page 57) we looked at a model with "threshold" nonconvexities. A stochastic version is

$$k_{t+1} = sA(k_t)k_t^{\alpha} W_{t+1} + (1-\delta)k_t \tag{6.5}$$

where the shock is assumed to be lognormally distributed (and independent), and A is the step function

$$A(k) = A_1 \mathbb{1}\{0 < k < k_b\} + A_2 \mathbb{1}\{k_b \le k < \infty\} = \begin{cases} A_1 & \text{if } 0 < k < k_b \\ A_2 & \text{if } k_b \le k < \infty \end{cases}$$

with $k_b \in S = (0, \infty)$ interpreted as the threshold, and $0 < A_1 < A_2$.

Figures 6.4 and 6.5 each show two time series generated for this model, with initial conditions $k_0 = 1$ and $k_0 = 80$. The parameters for figure 6.4 are set at $\alpha = 0.5$, $s = 0.25$, $A_1 = 15$, $A_2 = 25$, $\sigma^2 = 0.02$, and $k_b = 21.6$, while for figure 6.5, $k_b = 24.1$. Notice how initial conditions tend to persist, although time series occasionally cross the threshold k_b—what might be referred to in physics as a "phase transition." Informally, the state variable moves from one locally attracting region of the state space to another.

Exercise 6.1.13 Compute the empirical distribution functions at $t = 100$ for the two

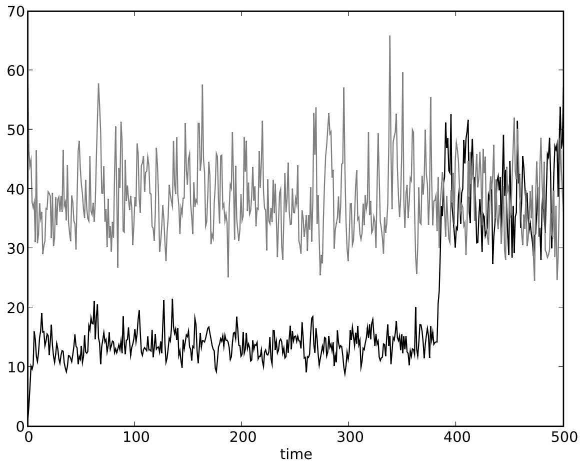

Figure 6.4 Persistence in time series

sets of parameters used in figures 6.4 and 6.5. Plot the functions and interpret their shapes.

Aside from computing the distributions, another interesting question is: How long do we expect it to take on average for the transition (crossing of the threshold k_b) to occur for an economy with initial condition $k_0 = 1$? More mathematically, what is the expectation of $\tau := \inf\{t \geq 0 : k_t > k_b\}$ when regarded as a random variable on $\mathbb{N}$? (Here τ is usually called the *first passage time* of $(k_t)_{t \geq 0}$ to (k_b, ∞).)

Exercise 6.1.14 Using $\alpha = 0.5$, $s = 0.25$, $A_1 = 15$, $A_2 = 25$, $\sigma^2 = 0.02$, $k_0 = 1$ and $k_b = 21.6$, compute an approximate expectation of τ by sample mean. (Set $n = 5,000$.) Do the same for $k_b = 24.1$, which corresponds to figure 6.5. How does your answer change? Interpret.

6.1.3 Density Dynamics

Now let's look more deeply at distribution dynamics for SRSs, with an emphasis on density dynamics. In reading this section, you should be aware that all densities create distributions but not all distributions are created by densities. If f is a density function on $\mathbb{R}$, then $F(x) := \int_{-\infty}^{x} f(u)du$ is a cumulative distribution function. However, if F is a cumulative distribution function with jumps—corresponding to positive probability

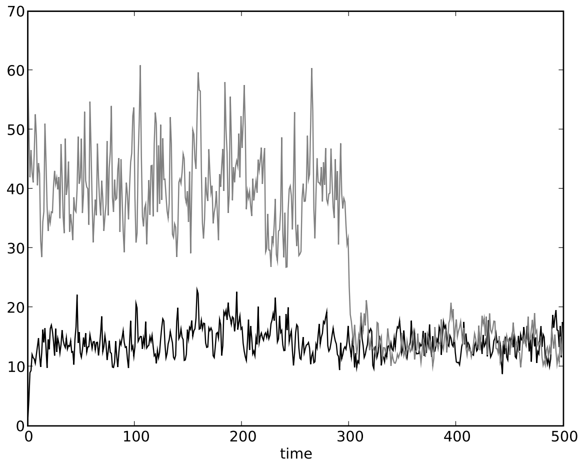

Figure 6.5 Persistence in time series

mass on individual points—then there exists no density f with $F(x) := \int_{-\infty}^{x} f(u)du$ for all $x \in \mathbb{R}$. (More about this later on.)

For the sake of concreteness, let's focus on a model of the form

$$X_{t+1} = g(X_t) + W_{t+1}, \quad X_0 \sim \psi, \quad (W_t)_{t \geq 1} \overset{\text{IID}}{\sim} \phi \tag{6.6}$$

where $Z = S = \mathbb{R}$, and both ψ and ϕ are *density* functions on $\mathbb{R}$. For this model, the distribution of X_t can be represented by a density ψ_t for any $t \geq 1$, and ψ_t and ψ_{t+1} are linked by the recursion

$$\psi_{t+1}(y) = \int p(x,y)\psi_t(x)dx, \quad \text{where } p(x,y) := \phi(y - g(x)) \tag{6.7}$$

Here p is called the *stochastic density kernel* corresponding to (6.3). It represents the distribution of $X_{t+1} = g(X_t) + W_{t+1}$ given $X_t = x$ (see below). The left-hand side of (6.7) is a continuous state version of (4.10) on page 75. It links the marginal densities of the process from one period to the next. In fact it defines the whole sequence of densities $(\psi_t)_{t \geq 1}$ for the process once an initial condition is given.

Let's try to understand why (6.7) holds, leaving fully rigorous arguments until later. First we need the following lemma.

Lemma 6.1.15 *If $W \sim \phi$, then $Y := g(x) + W$ has density $\phi(y - g(x))dy$.*[4]

[4]Here the symbol dy indicates that $\phi(y - g(x))$ is a density in y rather than in x.

Proof. Let F be the cumulative distribution function (cdf) of Y, and let Φ be the cdf corresponding to ϕ (i.e., $\Phi' = \phi$). We have

$$F(y) = \mathbb{P}\{g(x) + W \leq y\} = \mathbb{P}\{W \leq y - g(x)\} = \Phi(y - g(x))$$

The density of Y is $F'(y) = \phi(y - g(x))$ as claimed. □

Returning to (6.7), recall that if X and Y are random variables with joint density $p_{X,Y}(x,y)$, then their marginal densities satisfy

$$p_X(x) = \int p_{X,Y}(x,y)dy, \quad p_Y(y) = \int p_{X,Y}(x,y)dx$$

Moreover the conditional density $p_{Y|X}(x,y)$ of Y given $X = x$ is given by

$$p_{Y|X}(x,y) = \frac{p_{X,Y}(x,y)}{p_X(x)} \qquad (x, y \in S)$$

Some simple manipulations now yield the expression

$$p_Y(y) = \int p_{Y|X}(x,y)p_X(x)dx \qquad (y \in S)$$

We have almost established (6.7). Letting $X_{t+1} = Y$ and $X_t = X$, we have

$$\psi_{t+1}(y) = \int p_{X_{t+1}|X_t}(x,y)\psi_t(x)dx \qquad (y \in S)$$

The function $p_{X_{t+1}|X_t}(x,y)$ is the density of $g(X_t) + W_{t+1}$ given $X_t = x$, or, more simply, the density of $g(x) + W_{t+1}$. By lemma 6.1.15, this is $\phi(y - g(x)) =: p(x,y)$, confirming (6.7).

Now let's look at the dynamics implied by the law of motion (6.7). The initial condition is $\psi_0 = \psi$, which is the density of X_0 (regarded as given). From this initial condition, (6.7) defines the entire sequence $(\psi_t)_{t\geq 0}$. There are a couple of ways that we can go about computing elements of this sequence. One is numerical integration. For example, ψ_1 could be calculated by evaluating $\psi_1(y) = \int p(x,y)\psi(x)dx$ at each $y \in S$. Come to think of it, though, this is impossible: there is an infinity of such y. Instead, we would have to evaluate on a finite grid, use our results to form an approximation $\hat{\psi}_1$ of ψ_1, then do the same to obtain $\hat{\psi}_2$, and so on.

Actually this process is not very efficient, and it is difficult to obtain a measure of accuracy. So let's consider some other approaches. Say that we wish to compute ψ_t, where t is a fixed point in time. Since we know how to compute empirical distribution functions by simulation, we could generate n observations of X_t (see algorithm 6.1), compute the empirical distribution function F_t^n, and differentiate F_t^n to obtain an approximation ψ_t^n to ψ_t.

It turns out that this is not a good plan either. The reason is that F^n_t is not differentiable everywhere on S. And although it is differentiable at many points in S, at those points the derivative is zero. So the derivative of F^n_t contains *no* information about ψ_t.[5]

Another plan would be to generate observations of X_t and histogram them. This is a reasonable and common way to proceed—but not without its flaws. The main problem is that the histogram converges rather slowly to its target ψ_t. This is because histograms have no notion of neighborhood, and do not make use of all knowledge we have at hand. For example, if the number of bins is large, then it is often the case that no points from the sample will fall in certain bins. This may happen even for bins close to the mean of the distribution. The result is a "spiky" histogram, even when the true density is smooth.[6]

We can include the prior information that the density of ψ_t is relatively smooth using *Parzen windows*, or nonparametric kernel density estimates. The kernel density estimate f_n of unknown density f from observations $(Y_i)_{i=1}^n$ is defined as

$$f_n(x) := \frac{1}{n \cdot \delta_n} \sum_{i=1}^{n} K\left(\frac{x - Y_i}{\delta_n}\right) \qquad (x \in \mathbb{R}) \tag{6.8}$$

where K is some density on $\mathbb{R}$, and δ_n is either a parameter or a function of the data, usually referred to as the *bandwidth*.

Essentially, f_n is a collection of n "bumps," one centered on each data point Y_i. These are then summed and normalized to create a density.

Exercise 6.1.16 Using a suitable change of variables in the integral, show that f_n is a density for every n.

The bandwidth parameter plays a role similar to the number of bins used in the histogram: A high value means that the densities we place on each data point are flat with large tails. A low value means they are concentrated around each data point, and f_n is spiky.

Exercise 6.1.17 Implement the nonparametric kernel density estimator (6.8) using a standard normal density for K. Base your code on the empirical distribution function class in listing 6.3. Generate a sample of 100 observations from the standard normal distribution and plot the density esimate for bandwidth values 0.01, 0.1, and 0.5.

Although the nonparametric kernel density estimator produces good results in a broad range of environments, it turns out that for the problem at hand there is a better

[5]Readers familiar with the theory of ill-posed problems will have a feel for what is going on here. The density computation problem is ill-posed!

[6]We get ψ_t by integrating ψ_{t-1} with respect to a kernel $\phi(y - g(x))$, and functions produced in this way are usually smooth rather than spiky.

way: The *look-ahead estimator* ψ_t^n of ψ_t is defined by generating n independent draws $(X_{t-1}^1, \ldots, X_{t-1}^n)$ of X_{t-1} and then setting

$$\psi_t^n(y) := \frac{1}{n} \sum_{i=1}^{n} p(X_{t-1}^i, y) \qquad (y \in \mathbb{R}) \tag{6.9}$$

where $p(x,y) = \phi(y - g(x))$.[7] This estimator has excellent asymptotic and finite sample properties. While we won't go into them too deeply, note that

Lemma 6.1.18 *The look-ahead estimator ψ_t^n is pointwise unbiased for ψ_t, in the sense that $\mathbb{E}\psi_t^n(y) = \psi_t(y)$ for every $y \in S$. Moreover $\psi_t^n(y) \to \psi_t(y)$ as $n \to \infty$ with probability one.*

Proof. Fix $y \in S$, and consider the random variable $Y := p(X_{t-1}, y)$. The look-ahead estimator $\frac{1}{n}\sum_{i=1}^n p(X_{t-1}^i, y)$ is the sample mean of IID copies of Y, while the mean is

$$\mathbb{E}Y = \mathbb{E}p(X_{t-1}^i, y) = \int p(x,y)\psi_{t-1}(x)dx = \psi_t(y)$$

(The last equality is due to (6.7)). The desired result now follows from the fact that the sample mean of an IID sequence of random variables is an unbiased and consistent estimator of the mean.[8] □

Figure 6.6 shows a sequence of densities from the STAR model (6.3), computed using the look-ahead estimator with 1,000 observations per density. The transition function G is the cumulative distribution function for the standard normal distribution, $\alpha_0 = 1$, $\alpha_1 = 0.4$, $\beta_0 = 10$, and $\beta_1 = 0.8$. The density ϕ is standard normal.

Exercise 6.1.19 Implement the look-ahead estimator for the STAR model using the same parameters used for figure 6.6. Replicate the figure (modulo the influence of randomness).

6.1.4 Stationary Densities: First Pass

The sequence of densities $(\psi_t)_{t \geq 0}$ in figure 6.6 appears to be converging.[9] Indeed it can be shown (see chapter 8) that there is a limiting distribution ψ^* to which $(\psi_t)_{t \geq 0}$ is converging, and that the limit ψ^* is independent of the initial condition ψ_0. The density ψ^* is called a stationary density, and it satisfies

$$\psi^*(y) = \int p(x,y)\psi^*(x)dx \qquad (y \in \mathbb{R}) \tag{6.10}$$

[7] The independent draws $(X_{t-1}^1, \ldots, X_{t-1}^n)$ can be obtained via algorithm 6.1 on page 121.

[8] If you don't know the proof of this fact then try to do it yourself. Consistency follows from the law of large numbers (theorem 4.3.31 on page 94).

[9] See figure 6.12 on page 142 for another sequence of densities converging to a limit.

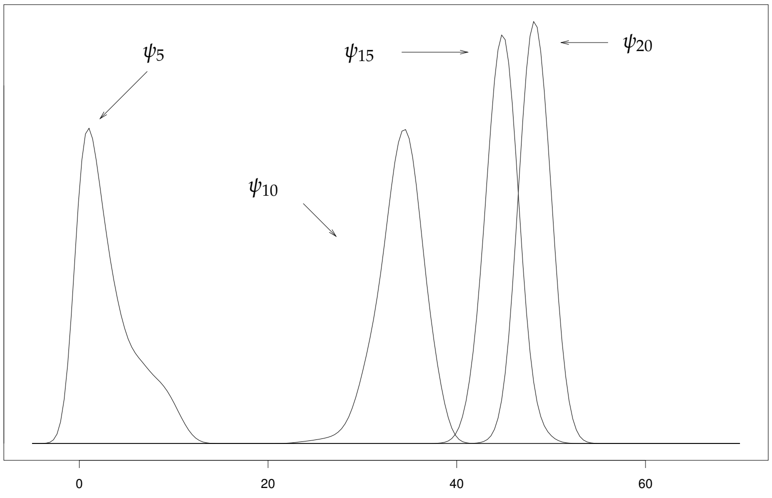

Figure 6.6 Density sequence

More generally, a density ψ^* on $\mathbb{R}$ is called *stationary* for the SRS (6.6) if (6.10) holds, where the density kernel p satisfies $p(x, y) = \phi(y - g(x))$. The SRS is called globally stable if there exists one and only one such density on $\mathbb{R}$, and the sequence of marginal distributions $(\psi_t)_{t \geq 0}$ converges to it as $t \to \infty$. (A more formal definition is given in chapter 8.)

You will recall that in the finite case a distribution ψ^* is called stationary if $\psi^* = \psi^* \mathbf{M}$, or equivalently, $\psi^*(y) = \sum_{x \in S} p(x, y) \psi^*(x)$ for all $y \in S$. The expression (6.10) simply replaces the sum with an integral, and the basic idea is the same: if the current marginal density is stationary, then updating to the next period leaves probabilities unchanged. Note, however, that when the state space is infinite a stationary density may fail to exist. You will be asked to give an example in exercise 8.2.2.

Recall that in the finite state case, when the stochastic kernel p is globally stable, each Markov chain generated by the kernel satisfies a law of large numbers (theorem 4.3.33, page 94). Here we have an analogous result. As shown in theorem 8.2.15 (page 206), given global stability and a function h such that $\int |h(x)| \psi^*(x) dx$ is finite, we have

$$\frac{1}{n} \sum_{t=1}^{n} h(X_t) \to \int h(x) \psi^*(x) dx \quad \text{as } n \to \infty \tag{6.11}$$

with probability one, where $(X_t)_{t \geq 0}$ is a time series generated by the model.

Exercise 6.1.20 Consider the STAR model (6.3) with $\alpha_0 = \beta_0 = 0$ and $\alpha_1 = \beta_1 = a$, where a is a constant with $|a| < 1$. Suppose that ϕ is standard normal. We will see later that this is a stable parameter configuration, and $\psi^* = N(0, 1/(1-a^2))$ is stationary for this kernel. From (6.11) we have

$$\frac{1}{n}\sum_{t=1}^{n} X_t^2 \simeq \frac{1}{1-\alpha^2} \quad \text{for large } n$$

Write a simulation that compares these two expressions for large n.

The LLN gives us a method to investigate the steady state distribution ψ^* for globally stable systems. For example, we can form the empirical distribution function

$$F^n(x) := \frac{1}{n}\sum_{t=1}^{n} \mathbb{1}\{X_t \leq x\} = \frac{1}{n}\sum_{t=1}^{n} \mathbb{1}_{(-\infty,x]}(X_t) \qquad (x \in \mathbb{R})$$

where $\mathbb{1}_{(-\infty,x]}(y) = 1$ if $y \leq x$ and zero otherwise and $(X_t)_{t\geq 0}$ is a simulated time series generated by the model. The empirical distribution function was discussed previously in §6.1.2. We will see in what follows that $\int \mathbb{1}_{(-\infty,x]}(y)\psi^*(y)dy$ is the probability that a draw from ψ^* falls below x. In other words, $F(x) := \int \mathbb{1}_{(-\infty,x]}(y)\psi^*(y)dy$ is the cumulative distribution function associated with ψ^*. Setting $h = \mathbb{1}_{(-\infty,x]}$ in (6.11), we then have $F^n(x) \to F(x)$ with probability one, $\forall x \in \mathbb{R}$, and the empirical distribution function is consistent for F.

Exercise 6.1.21 Use the empirical distribution function to compute an estimate of F for (6.3) under the same parameters used in figure 6.6.

There is, however, a more powerful technique for evaluating ψ^* when global stability holds. Taking our simulated time series $(X_t)_{t=1}^n$, define

$$\psi_n^*(y) := \frac{1}{n}\sum_{t=1}^{n} p(X_t, y) \qquad (y \in \mathbb{R}) \tag{6.12}$$

This expression is almost identical to the look-ahead estimator developed in §6.1.3 (see (6.9) on page 129), with the difference being that the random samples are now a single time series rather than repeated draws at a fixed point in time. To study the properties of ψ_n^*, observe that for any fixed $y \in S$, the LLN (6.11) gives us

$$\psi_n^*(y) := \frac{1}{n}\sum_{t=1}^{n} p(X_t, y) \to \int p(x,y)\psi^*(x)dx = \psi^*(y)$$

where the last equality is by (6.10). Thus $\psi_n^*(y)$ is consistent for $\psi^*(y)$.

In fact much stronger results are true, and ψ_n^* is an excellent estimator for ψ^* (see, e.g., Stachurski and Martin 2008). The reason is that while estimators such as F^n use

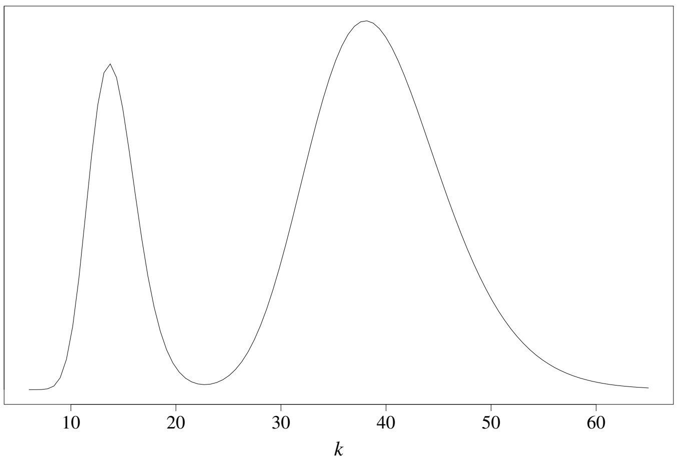

Figure 6.7 Look-ahead estimator

only the information contained in the sampled time series (X_t), the look-ahead estimator ψ_n^* also incorporates the stochastic kernel p, which encodes the entire dynamic structure of the model.

Exercise 6.1.22 Use the look-ahead estimator (6.12) to compute an estimate of ψ^* for (6.3) under the same parameters used in figure 6.6.

Here is a second application. Consider the nonconvex growth model in (6.5) on page 124 with $\delta = 1$. We will prove below that the stochastic density kernel for this model is

$$p(x,y) = \phi\left(\frac{y}{sA(x)x^\alpha}\right)\frac{1}{sA(x)x^\alpha} \qquad (x, y > 0) \tag{6.13}$$

and that the model is globally stable. From stability we obtain the LLN (6.11), and hence the look-ahead estimator (6.12) is consistent for the unique stationary density ψ^*. Figure 6.7 shows a realization of ψ_n^* when A is the step function

$$A(k) := A_1 \mathbb{1}\{k \le k_b\} + A_2 \mathbb{1}\{k > k_b\} \qquad (k > 0)$$

and $W_t = e_t^\xi$, where $\xi_t \sim N(0, \sigma^2)$. The parameters are $\alpha = 0.5$, $s = 0.25$, $A_1 = 15$, $A_2 = 25$, $\sigma^2 = 0.02$, $k_b = 22.81$, and $k_0 = k_b$.

Exercise 6.1.23 Replicate figure 6.7. Sample a time series $(k_t)_{t \ge 0}$ from the model, and implement ψ_n^* with $(k_t)_{t \ge 0}$ and the kernel in (6.13).[10]

[10]You will require a value of n around 100,000 to get a reasonable estimate, and even then some variation

6.2 Optimal Growth, Infinite State

Let's now look at a simple optimal growth model on an infinite state space. We will compute the optimal policy for the model numerically using value iteration and policy iteration. We also study via simulation the dynamics of the model under that policy.

6.2.1 Optimization

Consider again the optimal growth model discussed in chapter 1. At time t an agent receives income y_t, which is split into consumption c_t and savings k_t. Given k_t, output at $t+1$ is $y_{t+1} = f(k_t, W_{t+1})$, where $(W_t)_{t\geq 1}$ is IID and takes values in $Z := (0, \infty)$ according to density ϕ. The agent's behavior is specified by a policy function σ, which is a map from $S := \mathbb{R}_+$ to $\mathbb{R}$ satisfying $0 \leq \sigma(y) \leq y$ for all $y \in S$. The value $\sigma(y)$ should be interpreted as the agent's choice of savings when income $= y$, while $0 \leq \sigma(y) \leq y$ is a feasibility constraint ensuring that savings is nonnegative and does not exceed income. The set of all such policies will be denoted by Σ.

As with the finite state case, choice of a policy function $\sigma \in \Sigma$ also determines an SRS for the state variable, given by

$$y_{t+1} = f(\sigma(y_t), W_{t+1}), \quad (W_t)_{t\geq 1} \overset{\text{IID}}{\sim} \phi, \quad y_0 = y \tag{6.14}$$

where y is initial income. Letting U be the agent's utility function and $\rho \in (0,1)$ be the discount factor, the agent's decision problem is

$$\max_{\sigma \in \Sigma} v_\sigma(y), \quad \text{where} \quad v_\sigma(y) := \mathbb{E}\left[\sum_{t=0}^{\infty} \rho^t U(y_t - \sigma(y_t))\right] \tag{6.15}$$

Here $v_\sigma(y)$ is the expected discounted value of following policy σ when initial income is $y_0 = y$. For now we assume that $U \colon \mathbb{R}_+ \to \mathbb{R}_+$ is bounded and continuous, and that $f \colon \mathbb{R}_+ \times Z \to \mathbb{R}_+$ is continuous.

As discussed in the finite case—see §5.1.1, and in particular the discussion surrounding (5.4) on page 100—a rigorous definition of the expectation in (6.15) requires measure theory. The details are deferred until chapter 10. Among other things, we will see that the expectation can be passed through the sum to obtain

$$v_\sigma(y) = \sum_{t=0}^{\infty} \rho^t \mathbb{E} U(y_t - \sigma(y_t)) \qquad (y \in S = \mathbb{R}_+) \tag{6.16}$$

will be observable over different realizations. This is due to the nonlinearity in the model and resulting slow convergence.

This expression is simpler to interpret, with each term $\mathbb{E}U(y_t - \sigma(y_t))$ defined in terms of integrals over $\mathbb{R}$. Specifically, we integrate the function $y \mapsto U(y - \sigma(y))$ with respect to the marginal distribution ψ_t of y_t, where y_t is defined recursively in (6.14).

Given v_σ in (6.16), we can define the value function v^* in exactly the same way as (5.8) on page 102: $v^*(y) := \sup\{v_\sigma(y) : \sigma \in \Sigma\}$.[11] Just as in §5.1.2, the value function satisfies a Bellman equation: Letting $\Gamma(y) := [0, y]$ be the feasible savings choices when income is y, we have

$$v^*(y) = \max_{k \in \Gamma(y)} \left\{ U(y-k) + \rho \int v^*(f(k,z))\phi(z)dz \right\} \qquad (y \in S) \tag{6.17}$$

The intuition behind (6.17) is similar to that for the finite state Bellman equation on page 102, and won't be repeated here. A proof that v^* satisfies (6.17) will be provided via theorem 10.1.11 on page 234. In the same theorem it is shown that v^* is continuous.

Recall that bcS is the set of continuous bounded real-valued functions on S. Given a $w \in bcS$, we say that $\sigma \in \Sigma$ is *w-greedy* if

$$\sigma(y) \in \operatorname*{argmax}_{k \in \Gamma(y)} \left\{ U(y-k) + \rho \int w(f(k,z))\phi(z)dz \right\} \qquad (y \in S) \tag{6.18}$$

Later in the text we will see that continuity of w implies continuity of the objective function in (6.18), and since $\Gamma(y)$ is compact, the existence of a maximizer $\sigma(y)$ for each y is guaranteed by theorem 3.2.22 (page 49).

We will also prove that a policy σ^* is optimal in terms of maximizing expected discounted rewards if and only if it is v^*-greedy (theorem 10.1.11). In view of continuity of v^* and the previous comment regarding existence of maximizers, this result shows that at least one optimal policy exists. Moreover we can compute σ^* by first solving for v^* and then obtaining σ^* as the maximizer in (6.18) with v^* in place of w.

In order to compute v^*, we define the Bellman operator T, which maps $w \in bcS$ into $Tw \in bcS$ via

$$Tw(y) = \max_{k \in \Gamma(y)} \left\{ U(y-k) + \rho \int w(f(k,z))\phi(z)dz \right\} \qquad (y \in S) \tag{6.19}$$

We prove in chapter 10 that T is a uniform contraction of modulus ρ on the metric space (bcS, d_∞), where $d_\infty(v, w) := \sup_{y \in S} |v(y) - w(y)|$. In view of Banach's fixed point theorem (page 53), T then has a unique fixed point $\bar{v} \in bcS$, and $T^n v \to \bar{v}$ in d_∞ as $n \to \infty$ for all $v \in bcS$. Moreover it is immediate from the definition of T and the Bellman equation that $Tv^*(y) = v^*(y)$ for all $y \in S$, so $\bar{v} = v^*$. We conclude that all trajectories of the dynamical system (bcS, T) converge to v^*.

[11]From boundedness of U it can be shown that this supremum is taken over a bounded set, and hence v^* is well defined. See exercise 10.1.8 on page 233. We treat unbounded rewards in §12.2.

These observations suggest that to solve for an optimal policy we can use the value iteration technique presented in algorithm 5.1 on page 103, replacing bS with bcS for the set from which the initial condition is chosen. The algorithm returns a v-greedy policy σ, computed from a function $v \in bcS$ that is close to v^*. If v is close to v^*, then v-greedy policies are "almost optimal." See §10.2.1 for details.

6.2.2 Fitted Value Iteration

Let's turn to numerical techniques. With regard to value iteration, the fact that the state space is infinite means that implementing the sequence of functions generated by the algorithm on a computer is problematic. Essentially, the issue is that if w is an arbitrary element of bcS, then to store w in memory we need to store the values $w(y)$ for every $y \in S$. For infinite S this is not generally possible.

At the same time, some functions from S to $\mathbb{R}$ can be stored on a computer. For example, if w is a polynomial function such as $w(y) = \sum_{i=0}^{n-1} a_i y^i$, then to store w in memory, we need only store the n coefficients $(a_i)_{i=0}^{n-1}$ and the instructions for obtaining $w(y)$ from these coefficients. Functions that can be recorded in this way (i.e., with a finite number of parameters) are said to have *finite parametric representation*.

Unfortunately, iterates of the Bellman operator do not naturally present themselves in finite parametric form. To get Tv from v, we need to solve a maximization problem at each y and record the result. Again, this is not possible when S is infinite. A common kludge is discretization, where S is replaced with a grid of size k, and the original model with a "similar" model that evolves on the grid. This is rarely the best way to treat continuous state problems, since a great deal of useful information is discarded, and there is little in the way of theory guaranteeing that the limiting policy converges to the optimal policy as $k \to \infty$.[12]

Another approach is fitted value iteration, as described in algorithm 6.2. Here $\mathscr{F}$ is a class of functions with finite parametric representation. The map $v \mapsto w$ defined by the first two lines of the loop is, in effect, an approximate Bellman operator $\hat{T}$, and fitted value iteration is equivalent to iteration with $\hat{T}$ in place of T. A detailed theoretical treatment of this algorithm is given in §10.2.3. At this stage let us try to grasp the key ideas, and then look at implementation.

The first thing to consider is the particular approximation scheme to be used in the step that sends Tv into $w \in \mathscr{F}$. A number of schemes have been used in economic modeling, from Chebychev polynomials to splines and neural nets. In choosing the best method we need to consider how the scheme interacts with the iteration process

[12]One reason is that the resulting policy is not an element of the original policy space Σ, making it difficult to discuss the error induced by approximation. As an aside, some studies actually treat discrete state problem using continuous approximations in order to reduce the number of parameters needed to store the value function.

Algorithm 6.2 Fitted value iteration

```
initialize $v \in bcS$
repeat
    sample the function $Tv$ at finite set of grid points $(y_i)_{i=1}^k$
    use the samples to construct an approximation $w \in \mathscr{F}$ of $Tv$
    set $e = d_\infty(v, w)$
    set $v = w$
until $e$ is less that some tolerance
solve for a $v$-greedy policy $\sigma$
```

used to compute the fixed point v^*. A scheme that approximates individual functions well with respect to some given criterion does not always guarantee good dynamic properties for the sequence $(\hat{T}^n v)_{n \geq 1}$.

To try to pin down a suitable technique for approximation, let's decompose $\hat{T}$ into the action of two operators L and T. First T is applied to v—in practice Tv is evaluated only at finitely many points—and then an approximation operator L sends the result into $w = \hat{T}v \in \mathscr{F}$. Thus, $\hat{T} = L \circ T$. Figure 6.8 illustrates iteration of $\hat{T}$.

We aim to choose L such that (1) the sequence $(\hat{T}^n v)_{n \geq 1}$ converges, and (2) the collection of functions $\mathscr{F}$ is sufficiently rich that the limit of this sequence (which lives in $\mathscr{F}$) can be close to the fixed point v^* of T (which lives in bcS).[13] The richness of $\mathscr{F}$ depends on the choice of the approximation scheme and the number of grid points k in algorithm 6.2. In the formal results presented in §10.2.3, we will see that the approximation error depends on $d_\infty(Lv^*, v^*)$, which indicates how well v^* can be approximated by an element of $\mathscr{F}$.

Returning to point (1), any serious attempt at theory requires that the sequence $(\hat{T}^n v)_{n \geq 1}$ converges in some sense as $n \to \infty$. In this connection, note the following result.

Exercise 6.2.1 Let M and N be operators sending metric space (U, d) into itself. Show that if N is a uniform contraction with modulus ρ and M is nonexpansive, then $M \circ N$ is a uniform contraction with modulus ρ.

As T is a uniform contraction on (bcS, d_∞), we see that $\hat{T}$ is uniformly contracting whenever L is nonexpansive on (bcS, d_∞). While for some common approximation architectures this fails, it does hold for a number of useful schemes. When attention is restricted to these schemes the sequence $(\hat{T}^n v)_{n \geq 1}$ is convergent by Banach's fixed point theorem, and we can provide a detailed analysis of the algorithm.

[13]More correctly, the limit of the sequence lives in $\text{cl}\,\mathscr{F} \subset bcS$.

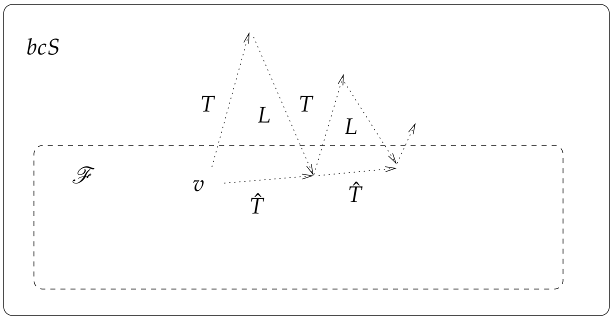

Figure 6.8 The map $\hat{T} := L \circ T$

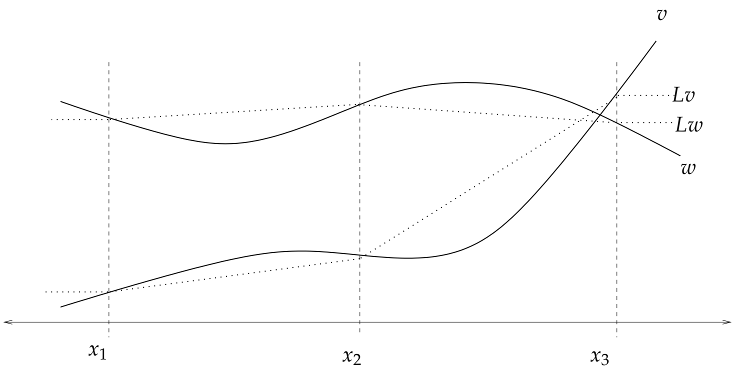

Figure 6.9 Approximation via linear interpolation

Listing 6.4 (`lininterp.py`) An interpolation class

```
from scipy import interp

class LinInterp:
    "Provides linear interpolation in one dimension."

    def __init__(self, X, Y):
        """Parameters: X and Y are sequences or arrays
        containing the (x,y) interpolation points."""
        self.X, self.Y = X, Y

    def __call__(self, z):
        """If z is a float (or integer) returns a float;
        if z is a sequence or array returns an array."""
        if isinstance(z, int) or isinstance(z, float):
            return interp([z], self.X, self.Y)[0]
        return interp(z, self.X, self.Y)
```

Let's move on to implementation, deferring further theory until §10.2.3. The approximation scheme we will use is piecewise linear interpolation, as shown in figure 6.9. (Outside the set of grid points, the approximations are constant.) With reference to the figure, it is not difficult to see that for any $v, w \in bcS$, and any x in the domain, we have

$$|Lv(x) - Lw(x)| \leq \sup_{1 \leq i \leq k} |v(x_i) - w(x_i)| \leq \|v - w\|_\infty$$

Taking the supremum over x gives $\|Lv - Lw\|_\infty \leq \|v - w\|_\infty$, so L is nonexpansive on bcS.

Consider the class `LinInterp` in listing 6.4. This class provides an interface to SciPy's `interp()` function, which implements linear interpolation. The latter is called using syntax `interp(Z, X, Y)`, where `X` and `Y` are arrays of equal length providing the (x, y) interpolation points, and `Z` is an array of arbitrary points where the interpolant is to be evaluated. The call returns an array of length `len(Z)` containing these evaluations.

The class `LinInterp` makes two modifications. First, an object of class `LinInterp` stores the interpolation points `X` and `Y` internally, so after instantiation of the object with these arrays, evaluations of the interpolant can be obtained without having to pass the interpolation points each time. Second, we may wish to evaluate the inter-

Listing 6.5 (`fvi.py`) Fitted value iteration

```
from scipy import linspace, mean, exp, randn
from scipy.optimize import fminbound
from lininterp import LinInterp        # From listing 6.4

theta, alpha, rho = 0.5, 0.8, 0.9      # Parameters
def U(c): return 1 - exp(- theta * c)  # Utility
def f(k, z): return (k**alpha) * z     # Production
W = exp(randn(1000))                   # Draws of shock

gridmax, gridsize = 8, 150
grid = linspace(0, gridmax**1e-1, gridsize)**10

def maximum(h, a, b):
    return h(fminbound(lambda x: -h(x), a, b))

def bellman(w):
    """The approximate Bellman operator.
    Parameters: w is a vectorized function (i.e., a
    callable object which acts pointwise on arrays).
    Returns: An instance of LinInterp.
    """
    vals = []
    for y in grid:
        h = lambda k: U(y - k) + rho * mean(w(f(k,W)))
        vals.append(maximum(h, 0, y))
    return LinInterp(grid, vals)
```

polant at a single point (float or integer) and receive a single number in return (rather than always having to pass and receive arrays). The **if** statement in the `__call__` method checks the type of the call parameter z and deals with it appropriately.[14]

The main part of the code is in listing 6.5. The utility function is set to $U(c) = 1 - e^{-\theta c}$, where the risk aversion parameter θ determines the curvature of U. The production function is $f(k,z) = k^{\alpha} z$. The shock is assumed to be lognormal; $W = e^{\xi}$, where ξ is standard normal.

Stepping through the logic of listing 6.5, the grid is formed by a call to SciPy's `linspace()` function, which returns an evenly spaced sequence on $[0, 8^{0.1}]$. Algebraic

[14]For discussion of the `__call__` method see §2.2.3.

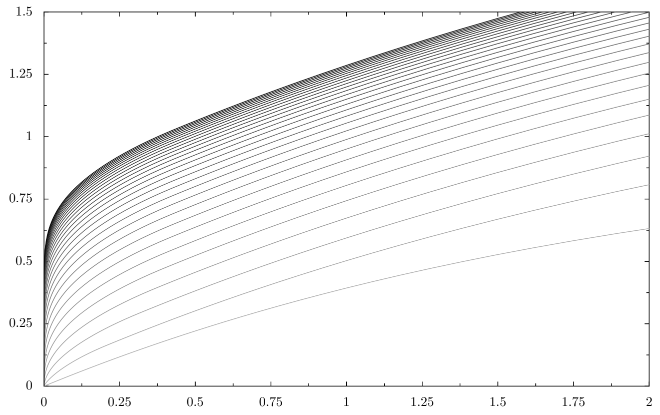

Figure 6.10 FVI algorithm iterates

operations on SciPy (NumPy) arrays are performed elementwise, so appending `**10` to the end of the line raises each element of the grid to the power of 10. The overall effect is to create a grid on $[0,8]$ such that most grid points are close to zero. This is desirable since most the value function's curvature is close to zero, and more curvature requires closer grid points to achieve the same level of error.

Next we define the function `maximum()` that takes a function `h` and two points `a` and `b`, and returns the maximum of `h` on the interval $[a,b]$. As defined here, `maximum()` is just a wrapper for SciPy's `fminbound()` function. The latter computes minimizers (not minima), and our wrapper uses this functionality to compute maxima. Specifically, we exploit the fact that the maximizer of h on $[a,b]$ is the minimizer x^* of $-h$ on $[a,b]$. Hence $h(x^*)$ is the maximum.

The function `bellman()` is the approximate Bellman operator $\hat{T} = L \circ T$, taking a function w and returning a new function $\hat{T}w$. The **for** loop steps through each grid point y_i, computing $Tw(y_i)$ as defined in (6.19) and recording this value in `vals`. Note how the expression `mean(w(f(k,W)))` is used to approximate the expection in $\int w(f(k,z))\phi(z)dz$. We are using the fact that `w` is vectorized (i.e., acts elementwise on NumPy arrays), so `w(f(k,W))` is an array produced by applying $w(f(k,\cdot))$ to each element of the shock array `W`. Vectorized operations are typically faster than **for** loops.[15]

After collecting the values $Tw(y_i)$ in the **for** loop, the last line of the function definition returns an instance of `LinInterp`, which provides linear interpolation between the set of points $(y_i, Tw(y_i))$. This object corresponds to $\hat{T}w = L(Tw)$. Figure 6.10 shows convergence of the sequence of iterates, starting from initial condition U.

[15] Alternatively, the integral can be computed using a numerical integration routine.

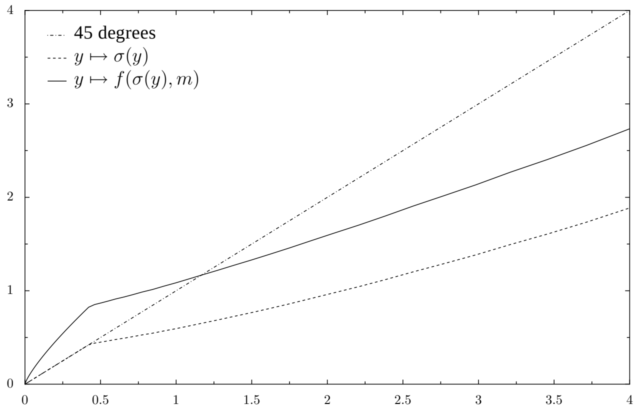

Figure 6.11 Approximate optimal policy

We can now compute a greedy policy σ from the last of these iterates, as shown in figure 6.11; along with the function $y \mapsto f(\sigma(y), m)$, where m is the mean of the shock. If the shock is always at its mean, then from every positive initial condition the income process $(y_t)_{t \geq 0}$ would converge to the unique fixed point at $\simeq 1.25$. Notice that when income is low, the agent invests all available income.

Of course, what happens when the shock is at its mean gives us little feel for the true dynamics. To generate the sequence of densities corresponding to the process $y_{t+1} = f(\sigma(y_t), W_{t+1})$, we can use the look-ahead estimator (6.9) on page 129. To use the theory developed there it is easiest to operate in logs. Given our parameterization of f, taking logs of both sides of $y_{t+1} = f(\sigma(y_t), W_{t+1}) = \sigma(y_t)^\alpha W_{t+1}$ yields

$$x_{t+1} = \alpha \ln \sigma(\exp(x_t)) + w_{t+1} := g(x_t) + w_{t+1}$$

where $x_t := \ln y_t$, $w_t := \ln W_t$ and $g(x) := \alpha \ln \sigma(\exp(x))$. This SRS is of the form (6.6) on page 126, and the look-ahead estimator of the density ψ_t of x_t can be computed via

$$\psi_t^n(y) := \frac{1}{n} \sum_{i=1}^{n} p(x_{t-1}^i, y) \qquad (y \in \mathbb{R}) \tag{6.20}$$

where $p(x, y) = \phi(y - g(x))$ and $(x_{t-1}^i)_{i=1}^n$ is n independent draws of x_{t-1} starting from a given initial condition x_0. Here ϕ is the density of w_t—in this case it's $N(0, 1)$. Figure 6.12 shows the densities ψ_1 to ψ_{15} starting at $x_0 \equiv -7.5$ and using $n = 1{,}000$.

Exercise 6.2.2 Using your preferred plotting utility and extending the code in listing 6.5, replicate figures 6.10 through 6.12. Starting from initial condition U, about

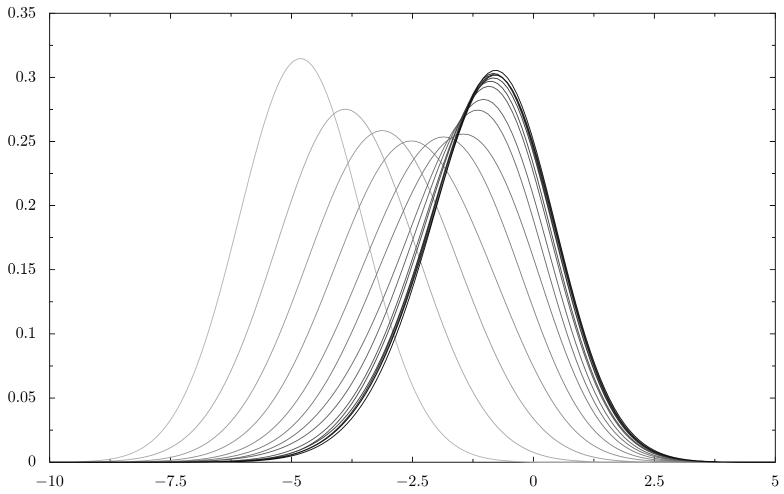

Figure 6.12 Densities of the income process

30 iterates of $\hat{T}$ produces a good approximation of v^*. When you compute the densities via the look-ahead estimator, try experimenting with different initial conditions. Observe how the densities always converge to the same limit.

6.2.3 Policy Iteration

Recall that in §5.1.3 we solved the finite state problem using a second algorithm, called policy iteration. (See in particular algorithm 5.2 on page 105.) We can do the same thing here, although we will need to use approximation techniques similar to those we used for fitted value iteration. The basic idea is presented in algorithm 6.3. (In practice, the functions v_σ and σ will have to be approximated at each step.)

Algorithm 6.3 Policy iteration algorithm

```
pick any $\sigma \in \Sigma$
**repeat**
    compute $v_\sigma$ from $\sigma$
    solve for a $v_\sigma$-greedy policy $\sigma'$
    set $\sigma = \sigma'$
**until** *until some stopping condition is satisfied*
```

The theory behind policy iteration is presented in §10.2.2. In this section our interest will be in implementation. When considering implementation the most difficult part is to compute v_σ from σ (i.e., to calculate the value of a given policy). Let's think about how we might do this given the definition $v_\sigma(y) = \sum_{t=0}^{\infty} \rho^t \mathbb{E} U(y_t - \sigma(y_t))$ in (6.16).

Fix y and consider the evaluation of $v_\sigma(y)$. We need to sum the terms $\mathbb{E} U(y_t - \sigma(y_t))$ discounted by ρ^t, at least for all $t \leq T$ where T is large. How should we evaluate $\mathbb{E} U(y_j - \sigma(y_j))$ for fixed j? One idea would be to use Monte Carlo. The initial condition is y, and from there the process $(y_t)_{t \geq 0}$ follows the SRS in (6.14). We can generate n independent observations $y_j^1, \ldots, y_j^n$ of y_j using a version of algorithm 6.1 (page 121). The mean $n^{-1} \sum_{i=1}^{n} U(y_j^i - \sigma(y_j^i))$ is close to $\mathbb{E} U(y_j - \sigma(y_j))$ for large n.[16]

While Monte Carlo methods can be useful for high-dimensional problems, this approach is not the easiest to implement. The reason is that even if we obtain good approximations to $\mathbb{E} U(y_t - \sigma(y_t))$ for each t and then sum them (discounting by ρ^t) to obtain $v_\sigma(y)$, we have only obtained an evaluation of the function v_σ at a single point y. But for algorithm 6.3 we need to evaluate v_σ at every point y (or at least on a grid of points so we can approximate v_σ).

Rather than us going down this path, consider the following iterative technique. For given $\sigma \in \Sigma$, define the operator T_σ that sends $w \in bcS$ into $Tw \in bcS$ by

$$T_\sigma w(y) = U(y - \sigma(y)) + \rho \int w(f(\sigma(y), z)) \phi(z) dz \qquad (y \in S) \tag{6.21}$$

For us the pertinent features of T_σ are summarized in the following result.

Lemma 6.2.3 *For each $\sigma \in \Sigma$, the operator T_σ is a uniform contraction on (bcS, d_∞), and the unique fixed point of T_σ in bcS is v_σ.*

The proof is not overly difficult but does involve some measure theory. As such we defer it until §10.1.3. What is important for us now is that from any initial guess $v \in bcS$ we have $T_\sigma^n v \to v_\sigma$ so, by iterating with T_σ, we can obtain an approximation to v_σ. In doing so we need to approximate at each iteration, just as we did for fitted value iteration (algorithm 6.2, but with T_σ in place of T). Listing 6.6 provides an implementation of this scheme, along with some additional routines.

The first line of the listing imports everything from the module `fvi`, which is the file in listing 6.5. This provides the primitives of the model and the grid. Next some functions are imported from `scipy`. The first function `maximizer()` is similar to `maximum()` in listing 6.5 but returns maximizers rather than maxima.

[16] An alternative Monte Carlo strategy would be to use the densities computed by the look-ahead estimator (see figure 6.12). Numerical integration of $y \mapsto U(y - \sigma(y))$ with respect to the density of y_j then gives another approximation to $\mathbb{E} U(y_j - \sigma(y_j))$.

Listing 6.6 (`fpi.py`) Fitted policy iteration

```
from fvi import *  # Import all definitions from listing 6.5
from scipy import absolute as abs

def maximizer(h, a, b):
    return fminbound(lambda x: -h(x), a, b)

def T(sigma, w):
    "Implements the operator L T_sigma."
    vals = []
    for y in grid:
        Tw_y = U(y - sigma(y)) + rho * mean(w(f(sigma(y), W)))
        vals.append(Tw_y)
    return LinInterp(grid, vals)

def get_greedy(w):
    "Computes a w-greedy policy."
    vals = []
    for y in grid:
        h = lambda k: U(y - k) + rho * mean(w(f(k, W)))
        vals.append(maximizer(h, 0, y))
    return LinInterp(grid, vals)

def get_value(sigma, v):
    """Computes an approximation to v_sigma, the value
    of following policy sigma. Function v is a guess.
    """
    tol = 1e-2          # Error tolerance
    while 1:
        new_v = T(sigma, v)
        err = max(abs(new_v(grid) - v(grid)))
        if err < tol:
            return new_v
        v = new_v
```

The function `T` takes functions `sigma` and `w`, representing σ and w respectively, and returns $T_\sigma w$ as an instance of the class `LinInterp` defined in listing 6.4. The code is relatively self-explanatory, as is that for the function `get_greedy()`, which takes as its argument a function `w` and returns a `w`-greedy policy as an instance of `LinInterp`.

The function `get_value` is used to calculate v_σ from σ. It takes as arguments the functions `sigma` and `v`, which represent σ and a guess for v_σ. We then iterate on this guess with T_σ (actually $L \circ T_\sigma$ where L is the linear interpolation operator) to obtain an approximation to v_σ. Iteration proceeds until the maximum distance over the grid points between the new and previous iterate falls below some tolerance.

Exercise 6.2.4 Compute an approximate optimal policy from listing 6.6. Check that this policy is similar to the one you computed in exercise 6.2.2.[17]

6.3 Stochastic Speculative Price

This section applies some of the ideas we have developed to the study of prices in a commodity market with consumers and speculators. After specifying and solving the model, we will also investigate how the solution can be obtained using the optimal growth model of §6.2. In this way, we will see that the optimal growth model, which at first pass seems rather limited, can in fact be applied to the study of decentralized economies with a large number of agents.

6.3.1 The Model

Consider a market for a single commodity, whose price is given at t by p_t. The "harvest" of the commodity at time t is W_t. We assume that the sequence $(W_t)_{t\geq 1}$ is IID with common density function ϕ. The harvests take values in $S := [a, \infty)$, where $a > 0$. The commodity is purchased by "consumers" and "speculators." We assume that consumers generate demand quantity $D(p)$ corresponding to price p. Regarding the inverse demand function $D^{-1} =: P$ we assume that

Assumption 6.3.1 The function $P\colon (0,\infty) \to (0,\infty)$ exists, is strictly decreasing and continuous, and satisfies $P(x) \uparrow \infty$ as $x \downarrow 0$.

Speculators can store the commodity between periods, with I_t units purchased in the current period yielding αI_t units in the next, $\alpha \in (0,1)$. For simplicity, the risk free interest rate is taken to be zero, so expected profit on I_t units is

$$\mathbb{E}_t p_{t+1} \cdot \alpha I_t - p_t I_t = (\alpha \mathbb{E}_t p_{t+1} - p_t) I_t$$

[17]Hint: Suppose that σ_n is the policy computed at the n-th iteration. A good initial condition for the guess of v_{σ_n} in `get_value()` is $v_{\sigma_{n-1}}$.

Here $\mathbb{E}_t p_{t+1}$ is the expectation of p_{t+1} taken at time t. Speculators are assumed to be risk neutral. Nonexistence of arbitrage requires that

$$\alpha \mathbb{E}_t p_{t+1} - p_t \leq 0 \tag{6.22}$$

Profit maximization gives the additional condition

$$\alpha \mathbb{E}_t p_{t+1} - p_t < 0 \text{ implies } I_t = 0 \tag{6.23}$$

We also require that the market clears in each period. Supply X_t is the sum $\alpha I_{t-1} + W_t$ of carryover by speculators and the current harvest, while demand is $D(p_t) + I_t$ (i.e., purchases by consumers and speculators). The market equilibrium condition is therefore

$$\alpha I_{t-1} + W_t =: X_t = D(p_t) + I_t \tag{6.24}$$

The initial condition $X_0 \in S$ is treated as given.

Now to find an equilibrium. Constructing a system $(I_t, p_t, X_t)_{t \geq 0}$ for investment, prices, and supply that satisfies (6.22)–(6.24) is not trivial. Our path of attack will be to *seek a system of prices that depend only on the current state.* In other words, we take a function $p \colon S \to (0, \infty)$ and set $p_t = p(X_t)$ for every t. The vector $(I_t, p_t, X_t)_{t \geq 0}$ then evolves as

$$p_t = p(X_t), \quad I_t = X_t - D(p_t), \quad X_{t+1} = \alpha I_t + W_{t+1} \tag{6.25}$$

For given X_0 and exogenous process $(W_t)_{t \geq 1}$, the system (6.25) determines the time path for $(I_t, p_t, X_t)_{t \geq 0}$ as a sequence of random variables. We seek a p such that (6.22) and (6.23) hold for the corresponding system (6.25).[18]

To this end, suppose that there exists a particular function $p^* \colon S \to (0, \infty)$ satisfying

$$p^*(x) = \max \left\{ \alpha \int p^*(\alpha I(x) + z) \phi(z) dz, P(x) \right\} \qquad (x \in S) \tag{6.26}$$

where

$$I(x) := x - D(p^*(x)) \qquad (x \in S) \tag{6.27}$$

It turns out that such a p^* will suffice, in the sense that (6.22) and (6.23) hold for the corresponding system (6.25). To see this, observe first that[19]

$$\mathbb{E}_t p_{t+1} = \mathbb{E}_t p^*(X_{t+1}) = \mathbb{E}_t p^*(\alpha I(X_t) + W_{t+1}) = \int p^*(\alpha I(X_t) + z) \phi(z) dz$$

[18]Given (6.25) we have $X_t = I_t + D(p_t)$, so (6.24) automatically holds.

[19]If the manipulations here are not obvious don't be concerned—we will treat random variables in detail later on. The last inequality uses the fact that if U and V are independent and V has density ϕ then the expectation of $h(U, V)$ given U is $\int h(U, z) \phi(z) dz$.

Thus (6.22) requires that

$$\alpha \int p^*(\alpha I(X_t) + z)\phi(z)dz \leq p^*(X_t)$$

This inequality is immediate from (6.26). Second, regarding (6.23), suppose that

$$\alpha \int p^*(\alpha I(X_t) + z)\phi(z)dz < p^*(X_t)$$

Then by (6.26) we have $p^*(X_t) = P(X_t)$, whence $D(p^*(X_t)) = X_t$, and $I_t = I(X_t) = 0$. (Why?) In conclusion, both (6.22) and (6.23) hold, and the system $(I_t, p_t, X_t)_{t \geq 0}$ is an equilibrium.

The only issue remaining is whether there does in fact exist a function $p^* \colon S \to (0, \infty)$ satisfying (6.26). This is not obvious, but can be answered in the affirmative by harnessing the power of Banach's fixed point theorem. To begin, let $\mathscr{C}$ denote the set of decreasing (i.e., nonincreasing) continuous functions $p \colon S \to \mathbb{R}$ with $p \geq P$ pointwise on S.

Exercise 6.3.2 Argue that $\mathscr{C} \subset bcS$, the set of all bounded continuous functions from S into $\mathbb{R}$.

Exercise 6.3.3 Show that if $(h_n) \subset \mathscr{C}$ and $d_\infty(h_n, h) \to 0$ for some function $h \in bcS$, then h is decreasing and dominates P.

Lemma 6.3.4 *The metric space $(\mathscr{C}, d_\infty)$ is complete.*

Proof. By theorem 3.2.3 on page 45 (closed subsets of complete spaces are complete) and the completeness of bcS (theorem 3.2.9 on page 46) we need only show that $\mathscr{C}$ is closed as a subset of bcS. This follows from exercise 6.3.3. □

As $\mathscr{C}$ is complete, it provides a suitable space in which we can introduce an operator from $\mathscr{C}$ to $\mathscr{C}$ and—appealing to Banach's fixed point theorem—show the existence of an equilibrium. The idea is to construct the operator such that (1) any fixed point satisfies (6.26), and (2) the operator is uniformly contracting on $\mathscr{C}$. The existence of an operator satisfying conditions (1) and (2) proves the existence of a solution to (6.26).

So let p be a given element of $\mathscr{C}$, and consider the new function on S constructed by associating to each $x \in S$ the real number r satisfying

$$r = \max \left\{ \alpha \int p(\alpha(x - D(r)) + z)\phi(z)dz,\ P(x) \right\} \tag{6.28}$$

We denote the new function by Tp, where $Tp(x)$ is the r that solves (6.28), and regard T as an operator sending elements of $\mathscr{C}$ into new functions on S. It is referred to below as the pricing functional operator.

Theorem 6.3.5 *The following results hold:*

1. *The pricing functional operator T is well-defined, in the sense that $Tp(x)$ is a uniquely defined real number for every $p \in \mathscr{C}$ and $x \in S$. Moreover*

$$P(x) \leq Tp(x) \leq v(x) := \max\left\{\alpha \int p(z)\phi(z)dz, P(x)\right\} \qquad (x \in S)$$

2. *T maps $\mathscr{C}$ into itself. That is, $T(\mathscr{C}) \subset \mathscr{C}$.*

The proof is only sketched. You might like to come back after reading up on measure theory and fill out the details. To start, let $p \in \mathscr{C}$ and $x \in S$. Define

$$h_x(r) := \max\left\{\alpha \int p(\alpha(x - D(r)) + z)\phi(z)dz,\ P(x)\right\} \qquad (P(x) \leq r \leq v(x))$$

Although we skip the proof, this function is continuous and decreasing on the interval $[P(x), v(x)]$. To establish the claim that there is a unique $r \in [P(x), v(x)]$ satisfying (6.28), we must show that h_x has a unique fixed point in this set. As h_x is decreasing, uniqueness is trivial.[20] Regarding existence, it suffices to show that there exist numbers $r_1 \leq r_2$ in $[P(x), v(x)]$ with

$$r_1 \leq h_x(r_1) \quad \text{and} \quad h_x(r_2) \leq r_2$$

Why does this suffice? The reason is that if either holds with equality then we are done, and if both hold inequalities are strict, then we can appeal to continuity of h_x and the intermediate value theorem (page 335).[21]

A suitable value for r_1 is $P(x)$. (Why?) For r_2 we can use $v(x)$, as

$$\begin{aligned} h_x(r_2) &= \max\left\{\alpha \int p(\alpha(x - D(r_2)) + z)\phi(z)dz,\ P(x)\right\} \\ &\leq \max\left\{\alpha \int p(z)\phi(z)dz,\ P(x)\right\} = v(x) = r_2 \end{aligned}$$

The claim is now established, and with it part 1 of the theorem.

To prove part 2, we must show that Tp (1) dominates P, (2) is decreasing on S, and (3) is continuous on S. Of these, (1) is implied by previous results, while (2) and (3) hold but proofs are omitted—we won't cover the necessary integration theory until chapter 7.[22]

[20]This was discussed in exercise 3.2.31 on page 51.

[21]Can you see why? Apply the theorem to $g(r) = r - h(r)$.

[22]The proof of (3) uses theorem B.1.4 on page 341.

Exercise 6.3.6 Verify that if p^* is a fixed point of T, then it solves (6.26).

Theorem 6.3.7 *The operator T is a uniform contraction of modulus α on $(\mathscr{C}, d_\infty)$.*

It follows from theorem 6.3.7 that there exists a unique $p^* \in \mathscr{C}$ with $Tp^* = p^*$. In view of exercise 6.3.6, p^* satisfies (6.26), and we have solved our existence problem. Thus it only remains to confirm theorem 6.3.7, which can be proved using Blackwell's sufficient condition for a uniform contraction. To state the latter, consider the metric space (M, d_∞), where M is a subset of bU, the bounded real-valued functions on arbitrary set U.

Theorem 6.3.8 (Blackwell) *Let M be a subset of bU with the property that $u \in M$ and $\gamma \in \mathbb{R}_+$ implies $u + \gamma \mathbb{1}_U \in M$. If $T \colon M \to M$ is monotone and*

$$\exists\, \lambda \in [0,1) \quad \text{s.t.} \quad T(u + \gamma \mathbb{1}_U) \leq Tu + \lambda \gamma \mathbb{1}_U \quad \forall\, u \in M \text{ and } \gamma \in \mathbb{R}_+ \tag{6.29}$$

then T is uniformly contracting on (M, d_∞) with modulus λ.

Monotonicity means that if $u, v \in M$ and $u \leq v$, then $Tu \leq Tv$, where all inequalities are pointwise on U. The proof of theorem 6.3.8 is given in on page 344, and we now return to the proof of theorem 6.3.7.

Exercise 6.3.9 Let h_1 and h_2 be decreasing functions on S with (necessarily unique) fixed points x_1 and x_2. Show that if $h_1 \leq h_2$, then $x_1 \leq x_2$.

Using this exercise it is easy to see that T is a monotone operator on $\mathscr{C}$: Pick any $p, q \in \mathscr{C}$ with $p \leq q$, and any $x \in S$. Let $r \mapsto h_p(r)$ be defined by

$$h_p(r) := \max \left\{ \alpha \int p(\alpha(x - D(r)) + z) \phi(z) dz,\ P(x) \right\}$$

and let $h_q(r)$ be defined analogously. Clearly, $Tp(x)$ is the fixed point of $r \mapsto h_p(r)$, as is $Tq(x)$ the fixed point of $r \mapsto h_q(r)$. Since $h_p(r) \leq h_q(r)$ for all r, it must be that $Tp(x) \leq Tq(x)$. As x was arbitrary we have $Tp \leq Tq$.

To apply Blackwell's condition, we need to show in addition that if $p \in \mathscr{C}$ and $\gamma \in \mathbb{R}_+$, then (1) $p + \gamma \mathbb{1}_S \in \mathscr{C}$, and (2) there exists a $\lambda < 1$ independent of p and γ and having the property

$$T(p + \gamma \mathbb{1}_S) \leq Tp + \lambda \gamma \mathbb{1}_S \tag{6.30}$$

Statement (1) is obviously true. Regarding statement (2), we make use of the following easy lemma:

Lemma 6.3.10 *Let a, b, and c be real numbers with $b \geq 0$. We have*

$$\max\{a + b, c\} \leq \max\{a, c\} + b$$

If you're not sure how to prove these kinds of inequalities, then here is how they are done: Observe that both

$$a + b \leq \max\{a, c\} + b \quad \text{and} \quad c \leq \max\{a, c\} + b$$

$$\therefore \quad \max\{a + b, c\} \leq \max\{a, c\} + b$$

To continue, let p and γ be as above, and let $q := p + \gamma \mathbb{1}_S$. Pick any $x \in S$. Let r_p stand for $Tp(x)$ and let r_q stand for $Tq(x)$. We have

$$\begin{aligned} r_q &= \max\left\{\alpha \int q(\alpha(x - D(r_q)) + z)\phi(z)dz,\ P(x)\right\} \\ &\leq \max\left\{\alpha \int q(\alpha(x - D(r_p)) + z)\phi(z)dz,\ P(x)\right\} \\ &= \max\left\{\alpha \int p(\alpha(x - D(r_p)) + z)\phi(z)dz + \alpha\gamma,\ P(x)\right\} \\ &\leq \max\left\{\alpha \int p(\alpha(x - D(r_p)) + z)\phi(z)dz,\ P(x)\right\} + \alpha\gamma \\ &= r_p + \alpha\gamma \end{aligned}$$

Here the first inequality follows from the fact that $r_p \leq r_q$ (since $p \leq q$ and T is monotone), and the second from lemma 6.3.10.

We have show that $T(p + \gamma\mathbb{1}_S)(x) \leq Tp(x) + \alpha\gamma$. Since x is arbitrary and $\alpha < 1$, the inequality (6.30) is established with $\lambda := \alpha$.

6.3.2 Numerical Solution

In this section we compute the rational expectations pricing functional p^* numerically via Banach's fixed point theorem. To start, recall that in §6.3.1 we established the existence of a function $p^* \colon S \to (0, \infty)$ in $\mathscr{C}$ satisfying (6.26). In the proof, p^* was shown to be the fixed point of the pricing operator $T \colon \mathscr{C} \ni p \mapsto Tp \in \mathscr{C}$. In view of Banach's theorem we have $d_\infty(T^n p, p^*) \to 0$ as $n \to \infty$ for any $p \in \mathscr{C}$, so a natural approach to computing p^* is by iterating on an arbitrary element p of $\mathscr{C}$ (such as P). In doing so, we will need to approximate the iterates $T^n p$ at each step, just as for fitted value iteration (algorithm 6.2, page 136). As before we use linear interpolation, which is nonexpansive with respect to d_∞.

So suppose that $p \in \mathscr{C}$ and $x \in S$ are fixed, and consider the problem of obtaining $Tp(x)$, which, by definition, is the unique $r \in [P(x), v(x)]$ such that (6.28) holds. Regarding this r,

Exercise 6.3.11 Show that $r = P(x)$ whenever $\alpha \int p(z)\phi(z)dz \leq P(x)$.

Exercise 6.3.12 Show that if $\alpha \int p(z)\phi(z)dz > P(x)$, then r satisfies

$$r = \alpha \int p(\alpha(x - D(r)) + z)\phi(z)dz$$

Algorithm 6.4 Computing $Tp(x)$

```
evaluate $y = \alpha \int p(z)\phi(z)dz$
**if** $y \leq P(x)$ **then** return $P(x)$
**else**
    define $h(r) = \alpha \int p(\alpha(x - D(r)) + z)\phi(z)dz$
    return the fixed point of $h$ in $[P(x), y]$
**end**
```

Together, exercises 6.3.11 and 6.3.12 suggest the method for finding r presented in algorithm 6.4, which returns $Tp(x)$ given p and x. An implementation of the algorithm is given in listing 6.7. The demand curve D is set to $1/x$, while for the shock we assume that $W_t = a + cB_t$, where B_t is beta with shape parameters $(5, 5)$. The function `fixed_point()` computes fixed points inside a specified interval using SciPy's `brentq` root-finding algorithm. The function `T()` implements algorithm 6.4.[23]

Once we can evaluate $Tp(x)$ for each p and x, we can proceed with the iteration algorithm, as shown in algorithm 6.5. A sequence of iterates starting at P is displayed in figure 6.13.

Algorithm 6.5 Computing the pricing function

```
set $p = P$
**repeat**
    sample $Tp$ at finite set of grid points $(x_i)_{i=1}^k$
    use samples to construct linear interpolant $q$ of $Tp$
    set $p = q$
**until** *a suitable stopping rule is satisfied*
```

Exercise 6.3.13 Implement algorithm 6.5 and replicate figure 6.13.[24]

[23]Although the `else` statement from algorithm 6.4 is omitted in the definition of `T()`, it is unnecessary because the last two lines in the definition of `T()` are only executed if the statement `y <= P(x)` is false. Note also that the function `p()` passed to `T()` must be vectorized.

[24]Hint: You can use the LinInterp class in listing 6.4 (page 138) for interpolation.

Listing 6.7 (cpdynam.py) Computing $Tp(x)$

```
from scipy import mean
from scipy.stats import beta
from scipy.optimize import brentq

alpha, a, c = 0.8, 5.0, 2.0
W = beta(5, 5).rvs(1000) * c + a   # Shock observations
D = P = lambda x: 1.0 / x

def fix_point(h, lower, upper):
    """Computes the fixed point of h on [upper, lower]
    using SciPy's brentq routine, which finds the
    zeros (roots) of a univariate function.
    Parameters: h is a function and lower and upper are
    numbers (floats or integers).  """
    return brentq(lambda x: x - h(x), lower, upper)

def T(p, x):
    """Computes Tp(x), where T is the pricing functional
    operator.
    Parameters: p is a vectorized function (i.e., acts
    pointwise on arrays) and x is a number.  """
    y = alpha * mean(p(W))
    if y <= P(x):
        return P(x)
    h = lambda r: alpha * mean(p(alpha*(x - D(r)) + W))
    return fix_point(h, P(x), y)
```

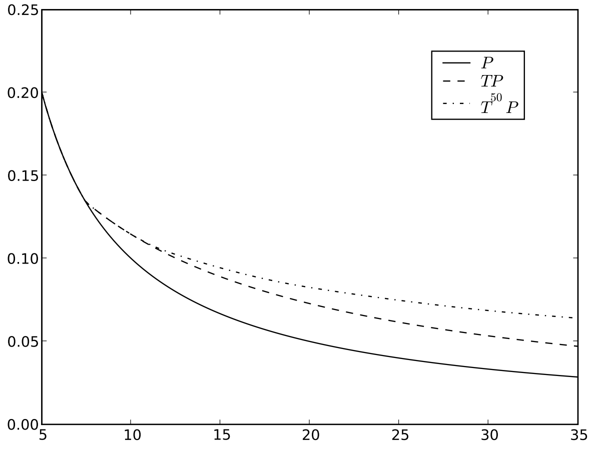

Figure 6.13 The trajectory $T^n P$

Given p^*, we have a dynamic system for quantities defined by

$$X_{t+1} = \alpha I(X_t) + W_{t+1}, \qquad (W_t)_{t \geq 1} \overset{\text{IID}}{\sim} \phi \tag{6.31}$$

where $I(x) := x - D(p^*(x))$. As shown in §6.1.3, the distribution ψ_t of X_t is a density for each $t \geq 1$, and the densities satisfy

$$\psi_{t+1}(y) = \int p(x, y) \psi_t(x) dx \qquad (y \in S)$$

where $p(x, y) = \phi(y - \alpha I(x))$, and ϕ is the density of the harvest W_t.[25]

Exercise 6.3.14 We will see later that the process (6.31) is stable, with a unique stationary density ψ^*, and that the look-ahead estimator given in (6.12) on page 131 can be used to estimate it. Using this estimator, show graphically that for these particular parameter values, speculators do not affect long-run probabilities for the state, in the sense that $\psi^* \simeq \phi$.[26]

[25]We regard ϕ as defined on all of $\mathbb{R}$, and zero off its support. Hence, if $y < \alpha I(x)$, then $p(x, y) = 0$.

[26]The latter is the distribution that prevails without speculation.

6.3.3 Equilibria and Optima

In §6.3.1 we used Banach's fixed point theorem to show the existence of a pricing functional p^* such that the resulting system for prices and quantities was a competitive equilibrium. There is another way we can obtain the same result using dynamic programming and the optimal growth model. Solving the problem this way illustrates one of the many fascinating links between decentralized equilibria and optimality.

Before getting started we are going to complicate the commodity pricing model slightly by removing the assumption that the interest rate is zero. With a positive and constant interest rate r, next period returns must be discounted by $\rho := 1/(1+r)$. As a result the no arbitrage and profit maximization conditions (6.22) and (6.23) become

$$\rho \alpha \mathbb{E}_t p_{t+1} - p_t \leq 0 \tag{6.32}$$

$$\rho \alpha \mathbb{E}_t p_{t+1} - p_t < 0 \text{ implies } I_t = 0 \tag{6.33}$$

As in §6.3.1 we seek a pricing function p^* such that the system

$$p_t = p^*(X_t), \quad I_t = X_t - D(p_t), \quad X_{t+1} = \alpha I_t + W_{t+1} \tag{6.34}$$

satisfies (6.32) and (6.33). This can be accomplished directly using the fixed point arguments in §6.3.1, making only minor adjustments to incorporate ρ. However, instead of going down this path we will introduce a fictitious planner who solves the optimal growth model described in §6.2.1. Through a suitable choice of primitives, we show that the resulting optimal policy can be used to obtain such a p^*. Since the optimal policy exists and can be calculated, p^* likewise exists and can be calculated.

Regarding the planner's primitives, the production function f is given by $f(k,z) = \alpha k + z$; the discount factor is $\rho := 1/(1+r)$; the utility function U is defined by $U(c) := \int_0^c P(x)dx$, where P is the inverse demand function for the commodity pricing model; and the distribution ϕ of the shock is the distribution of the harvest.

We assume that P is such that U is bounded on $\mathbb{R}_+$. By the fundamental theorem of calculus we have $U' = P$. The conditions of assumption 6.3.1 on page 145 also hold, and as a result the function U is strictly increasing, strictly concave and satisfies $U'(c) \uparrow \infty$ as $c \downarrow 0$. We remove the assumption in §6.3.1 that the shock is bounded away from zero, as this restriction is not needed.

Using the arguments in §6.2.1, we know that there exists at least one optimal policy. In fact concavity of the primitives implies that there is only one such policy. (For the proof, see §12.1.2.) The policy, denoted simply by σ, is v^*-greedy, which is to say that

$$\sigma(x) = \operatorname*{argmax}_{0 \leq k \leq x} \left\{ U(x-k) + \rho \int v^*(f(k,z))\phi(z)dz \right\} \tag{6.35}$$

for all x. One can also show that v^* is differentiable with $(v^*)'(x) = U'(x - \sigma(x))$, and that the objective function in (6.35) is likewise differentiable. Using these facts and taking into account the possibility of a corner solution, it can be shown that σ satisfies

$$U' \circ c(x) \geq \rho \int U' \circ c[f(\sigma(x), z)] f'(\sigma(x), z) \phi(z) dz \qquad \forall\, x \in S \tag{6.36}$$

Moreover if the inequality is strict at some $x > 0$, then $\sigma(x) = 0$. Here $c(x) := x - \sigma(x)$ and $f'(k, z)$ is the partial derivative of f with respect to k. This is the famous Euler (in)equality, and full proofs of all these claims can be found via propositions 12.1.23 and 12.1.24 in §12.1.2.

The main result of this section is that by setting p^* equal to marginal utility of consumption we obtain an equilibrium pricing functional for the commodity pricing model.

Proposition 6.3.15 *If p^* is defined by $p^*(x) := U' \circ c(x) =: U'(c(x))$, then the system defined in* (6.34) *satisfies* (6.32) and (6.33).

Proof. Substituting the definition of p^* into (6.36) we obtain

$$p^*(x) \geq \rho \int p^*[f(\sigma(x), z)] f'(\sigma(x), z) \phi(z) dz \qquad \forall\, x \in S$$

with strict inequality at x implying that $\sigma(x) = 0$. Using $f(k, z) = \alpha k + z$ this becomes

$$p^*(x) \geq \rho\alpha \int p^*(\alpha\sigma(x) + z) \phi(z) dz \qquad \forall\, x \in S$$

Now observe that since $c(x) = x - \sigma(x)$, we must have

$$p^*(x) = U'(x - \sigma(x)) = P(x - \sigma(x)) \qquad \forall\, x \in S$$
$$\therefore \quad D(p^*(x)) = x - \sigma(x) \qquad \forall\, x \in S$$

Turning this around, we get $\sigma(x) = x - D(p^*(x))$, and the right-hand side is precisely $I(x)$. Hence $\sigma = I$, and we have

$$\rho\alpha \int p^*(\alpha I(x) + z) \phi(z) dz - p^*(x) \leq 0 \qquad \forall\, x \in S$$

with strict inequality at x implying that $I(x) = 0$. Since this holds for all $x \in S$, it holds at any realization $X_t \in S$, so

$$\rho\alpha \int p^*(\alpha I(X_t) + z) \phi(z) dz - p^*(X_t) \leq 0$$

with strict inequality implying $I(X_t) = 0$. Substituting I_t and p_t from (6.34), and using the fact that

$$\mathbb{E}_t p_{t+1} = \int p^*(\alpha I(X_t) + z) \phi(z) dz$$

as shown in §6.3.1, we obtain (6.32) and (6.33). □

6.4 Commentary

Further theory and applications of stochastic recursive sequences in economics and finance can be found in Sargent (1987), Stokey and Lucas (1989), Farmer (1999), Duffie (2001), Miranda and Fackler (2002), Adda and Cooper (2003), or Ljungqvist and Sargent (2004). The simulation-based approach to computing marginal and stationary densities of stochastic recursive sequences in §6.1.3 and §6.1.4 was proposed by Glynn and Henderson (2001), and a detailed analysis of the technique and its properties can be found in Stachurski and Martin (2008).

The one-sector, infinite horizon, stochastic optimal growth model in §6.2 is essentially that of Brock and Mirman (1972). Related treatments can be found in Mirman and Zilcha (1975), Donaldson and Mehra (1983), Stokey and Lucas (1989), Amir (1996), and Williams (2004). A survey of the field is given in Olsen and Roy (2006). For discussion of the stability properties of the optimal growth model, see §12.1.3. The commentary to that chapter contains additional references.

We saw in §6.3.3 that optimal policies for the growth model coincide with the market equilibria of certain decentralized economies. For more discussion of the links between dynamic programming and competitive equilibria, see Stokey and Lucas (1989, ch. 16), or Bewley (2007). Seminal contributions to this area include Samuelson (1971), Lucas and Prescott (1971), Prescott and Mehra (1980), and Brock (1982).

Under some variations to the standard environment (incomplete markets, production externalities, distortionary taxes, etc.), equilibria and optima no longer conincide, and the problem facing the researcher is to find equilibria rather than optimal policies. For a sample of the literature, see Huggett (1993), Aiyagari (1994), Greenwood and Huffman (1995), Rios-Rull (1996), Krusell and Smith (1998), Kubler and Schmedders (2002), Reffett and Morand (2003), Krebs (2004), Datta et al. (2005), Miao (2006), and Angeletos (2007).

The commodity price model studied in §6.3 is originally due to Samuelson (1971), who connected equilibrium outcomes with solutions to dynamic programming problems. Our treatment in §6.3.1 and §6.3.2 follows Deaton and Laroque (1992), who were the first to derived the equilibrium price directly via Banach's fixed point theorem. The technique of iterating on the pricing functional is essentially equivalent to Coleman's algorithm (Coleman 1990). For more on the commodity pricing model, see, for example, Scheinkman and Schectman (1983) or Williams and Wright (1991).