Chapter 10

Computational tools

Lucas (1980) famously wrote "Our task as I see it...is to write a FORTRAN program that will accept specific economic policy rules as 'input' and will generate as 'output' statistics describing the operating characteristics of time series we care about, which are predicted to result from these policies." We do this work using computational tools rather than paper and pencil, because, in many cases, macroeconomic models do not have tractable analytical solutions. This chapter introduces some core tools that underpin these computational approaches. The methods introduced here—function approximation, root finding, optimization, discretization of stochastic processes, and linearization—form the backbone of modern computational macroeconomics. The goal here is to give the reader a bird's eye view of these methods and how they fit together within the macroeconomist toolkit.[1]

The chapter starts with some building block methods which are then combined to solve a dynamic programming problem in Sections 10.5. In practice, one can easily find software that performs the building block methods described below and one would not program these methods oneself. But understanding how these methods work is important in choosing which method to apply or diagnosing a problem when an algorithm does not behave as expected. Section 10.6 presents a perturbation method and is self-contained.

10.1 Approximating a function

We often encounter a situation where we need to store the information of a function in the computer. It is easy in some cases. For example, the information of a polynomial $f(x) = ax^2 + bx + c$ can be summarized by three numbers: a, b, and c. In many situations in macroeconomics, however, the function of interest does not have such a simple form. For example, when we solve Bellman equations in dynamic programming, most of the time, the value functions and policy functions do not have easy analytical expressions. Storing the value of the function at each point on an interval $[\underline{x}, \overline{x}]$ would in principle require infinite numbers, which is infeasible with finite computer memory. So, we need some way to approximate a function and this section introduces two popular methods.

[1] Readers looking for more details of these methods should see Numerical Recipes `http://numerical.recipes/book.html`, QuantEcon `https://quantecon.org/`, Judd (1998), Miranda and Fackler (2002).

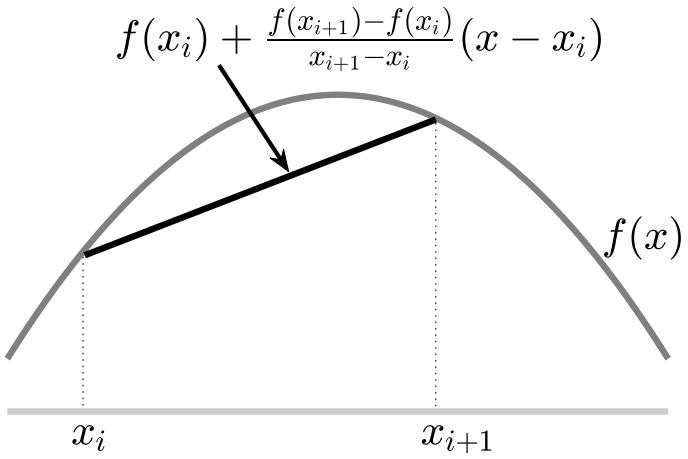

Figure 10.1: Linear interpolation

10.1.1 Interpolation

One natural method of approximation is to set discrete grid points on $[\underline{x}, \overline{x}]$ and store the function's value on these points. Let us index the grid points by $i \in \{1, ..., n\}$. It is natural to set $x_1 = \underline{x}$ and $x_n = \overline{x}$. In between, it is natural to have equally spaced points, but one might instead put more points in a subinterval where a more accurate approximation to the function is desired.

Linear interpolation

The most straightforward interpolation method is the linear interpolation. It only requires storing the values of $f(x_i)$ for all i. Then, given any value of x, we can find an approximate value of $f(x)$ by

1. Find the i where $x \in [x_i, x_{i+1})$.
2. Compute the approximated value $\hat{f}(x)$ by

$$\hat{f}(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i).$$

   As we can see from Figure 10.1, this formula provides the line that connects $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$.

This method has the advantage of using only local information ($f(x_{i+1})$ and $f(x_i)$) and is thus robust to what is happening elsewhere. The two disadvantages are that (i) the approximation error can be large if the underlying function is highly nonlinear, and (ii) the approximated function is not differentiable at the grid points.

Cubic spline interpolation

Another popular method is cubic spline interpolation, which can often achieve a better approximation of the underlying function than linear interpolation. Moreover, the approxi-

mated function is twice continuously differentiable. The latter property is often important in economics.

The idea is to interpolate using a (particular kind of) cubic function instead of a linear function. Then, we require the cubic function to have continuous first and second derivatives at each interior grid point, namely $x_2, ..., x_{n-1}$. We are able to devise a cubic function that only requires $f(x_i)$, $f(x_{i+1})$, $f''(x_i)$, and $f''(x_{i+1})$ to interpolate between x_i and x_{i+1}. Moreover, we do not need the actual information of the second derivatives—we can compute them by imposing continuity of the first derivatives at each grid point (plus some boundary conditions). Thus, once again, the only information we have to store is the values of $f(x_i)$ for all i. The downside compared with the linear interpolation is that step 1 uses the information of all $f(x_i)$s to compute a particular $f''(x_i)$, and thus, in principle, the value of $f(x_j)$ could affect the interpolation in $x \in [x_i, x_{i+1})$ even when x_j is far away from x_i.

10.1.2 Approximation by known functions

A second common approach, rather than relying on grid points, is to approximate directly with known functional forms (e.g., such as polynomials, log functions, and trigonometric functions) and consider their "weighted sum":

$$\hat{f}(x) = \sum_{i=1}^{n} a_i \phi_i(x). \tag{10.1}$$

Here, we choose n different functions $\phi_i(x)$, and the weights are a_i. A typical method of setting a_i is to choose various points in x, $\{x_1, ..., x_m\}$ and use the information of $\{f(x_1), ..., f(x_m)\}$ in addition to $\{\phi_i(x_1), ..., \phi_i(x_m)\}$ for all i. If $m = n$, (generically) a unique set of a_is solve (10.1) when the left-hand side is set as $f(x_j)$ (m equations with n unknowns). If $m \geq n$, we cannot set a_is to set (10.1) to hold with equality, but we can use (10.1) as a regression equation to choose the regression coefficients a_is so that the right-hand side is "close" to the left-hand side.

A popular choice of ϕ_is is a set of functions called the Chebyshev polynomials.[2] The Chebyshev polynomials are known to summarize the information of $f(x)$ in an efficient manner, and the methods of choosing $\{x_1, ..., x_m\}$ to find the a_i's are also well established. The downside of this method is similar to the cubic spline interpolation. Because this approximation method uses global information, an approximation of one part is influenced by the behavior of $f(x)$ and $\phi_i(x)$ in other parts of the domain.

10.2 Root finding

In economics, we often have to solve for the root of a nonlinear equation. Here, we focus on the one-dimensional case, that is, finding a scalar x that satisfies

$$f(x) = 0.$$

[2]Codes for implementing Chebyshev polynomial approximation can be found, for example, in the Numerical Recipes.

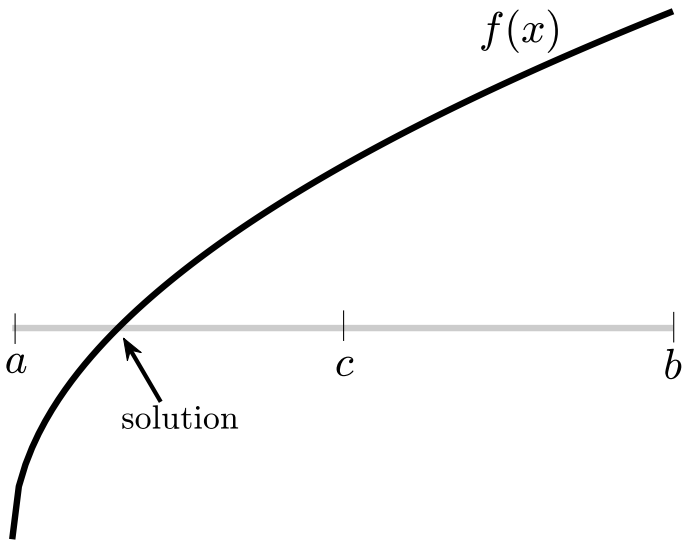

Figure 10.2: Bisection method

Methods for root finding in multi-dimensional cases often rely on similar principles.

The most obvious method of finding a root is brute-force grid search. If we know a root of $f(x) = 0$ exists within an interval $[\underline{x}, \overline{x}]$, create grid points $\{x_1, ..., x_n\}$ between $x_1 = \underline{x}$ and $x_n = \overline{x}$ for a large n. Evaluate $f(x_i)$ for each i and pick the x_i that corresponds to $f(x_i)$ closest to zero as the solution. This method always works and provides a good solution when n is a very large number. However, this method is extremely slow and is rarely the best choice.

10.2.1 Bisection

Bisection is related to brute-force grid search, but it is a lot more efficient. The steps are the following:

1. Find a and $b > a$ where $\text{sign}(f(a)) \neq \text{sign}(f(b))$. For example, in Figure 10.2, $f(a) < 0$ and $f(b) > 0$. This step is called "bracketing." If $f(x)$ is continuous, this condition implies that at least one solution to $f(x) = 0$ exists in the bracket $[a, b]$.

2. Let $c \equiv (a+b)/2$. Evaluate $f(c)$. If $\text{sign}(f(c)) = \text{sign}(f(a))$, a solution is in the bracket $[c, b]$ and we update the value of a to $a = c$. Otherwise, we update the value of b as $b = c$. In Figure 10.2, $\text{sign}(f(c)) = \text{sign}(f(b))$, and thus the solution is in $[a, c]$, and thus c becomes the new b.

3. Repeat step 2 until $b - a$ is less than a tolerance value you set (e.g., 10^{-6}). Check that $f(a)$ is close to zero, and use a as the solution.

The bisection method has several benefits. For one, it does not require differentiability of $f(x)$. In addition, even when $f(x)$ takes zero multiple times, this procedure always finds one of the solutions provided that the function is continuous. A big advantage of bisection is that it always ends after a finite number of steps. As each iteration cuts the interval in half,

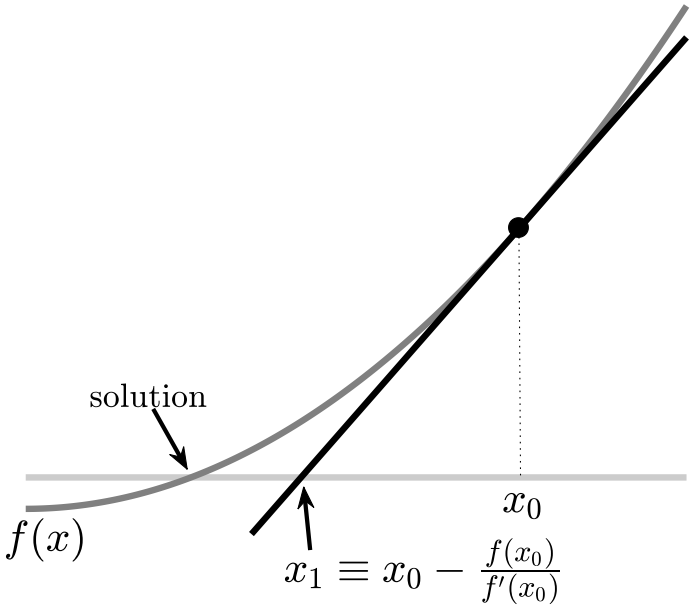

Figure 10.3: Constructing x_1 in the Newton-Raphson method

we know how many iterations it requires to shrink the starting interval down the tolerance level. The procedure always ends, even when no solutions exist, for example, because $f(x)$ is not continuous. If we don't know the properties of $f(x)$, checking $f(a) \approx 0$ in the final step is therefore very important.

The bisection procedure is much faster than the brute-force grid search method described earlier. Suppose we are trying to find a solution of $f(x) = 0$ on $[0, 1]$ with 10^{-3} accuracy in terms of x. This process requires 1000 evaluations of $f(x)$ in the case of the brute-force grid search, but only 10 evaluations in the case of bisection (because $(1/2)^{10} = 1/1024$). This method can be slower than other methods (e.g., the Newton-Raphson method described below can be much faster if $f(x)$ is close to linear), but it is quite a robust and useful method despite its simplicity.

10.2.2 Newton-Raphson

The idea of the Newton-Raphson method is to approximate a nonlinear function locally by a tangent line and use its root as the next guess. It is based on linear approximation of a function:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

for some starting point x_0. If this approximation is good, the solution for $f(x) = 0$ should, by plugging 0 into the left-hand side, be close to

$$\hat{x} = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{10.2}$$

If $f(x)$ is indeed linear, $\hat{x}$ delivers the solution in one step. The general procedure is as follows:

1. Make an initial guess x_0.

2. Construct x_1 from
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$
Figure 10.3 describes how x_1 is constructed.

3. Check whether $f(x_1) \approx 0$ using some tolerance value. That is, if $|f(x_1)| < \varepsilon$, where ε is the tolerance value, stop and call x_1 the solution. If not, now use x_1 and obtain x_2, going back to the previous step. Continue until $f(x_i) \approx 0$.

If $f(x)$ is close to linear, or x_0 is close to the true solution, this procedure can be much faster than bisection. The downside is that (i) it requires differentiability of $f(x)$, and we need to compute $f'(x)$ either analytically or numerically, and (ii) even when the solution exists, the procedure may not be able to find a solution if the function is not well behaved.

One method of finding a numerical derivative is to compute a finite-difference derivative
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$
In principle, a smaller h would provide a better approximation. In practice, because a computer does not recognize a value smaller-than-certain value (often called the "machine epsilon," typically around 10^{-16}), setting h too small is not desirable. A typical value used for h is around 10^{-5}.

10.3 Optimization

Let us consider maximizing a function $F(x)$. Similarly to how we find roots, we can, in principle, optimize $F(x)$ by a simple grid search: evaluate $F(x)$ on many grid points and pick the point x that yields the maximum value of $F(x)$. This method, although fail-safe, is very slow. Below, we introduce two popular methods.

We focus on the one-dimensional case where x is a scalar. For multi-dimensional cases, one approach is to optimize one dimension at a time. When maximizing $F(x, y)$, for example, fix x and maximize $F(x, y)$ in terms of y (this is a one-dimensional problem). By performing this procedure for many x, we can obtain the function of maximizing y: $y^*(x)$. Then, we can perform a one-dimensional problem of maximizing $F(x, y^*(x))$ over x.

10.3.1 Golden-section search

The first method, as in the bisection method in the previous section, only requires evaluating the values of the function. The procedure is as follows.

1. First, find three values a, b, and c such that $a < b < c$ and $F(a) < F(b)$ and $F(c) < F(b)$. These properties mean that a (local) maximum point exists between a and c. (If such values cannot be found, a corner solution is likely: either a or c is the maximum point.) These three points (a, b, c) form the initial "bracket."

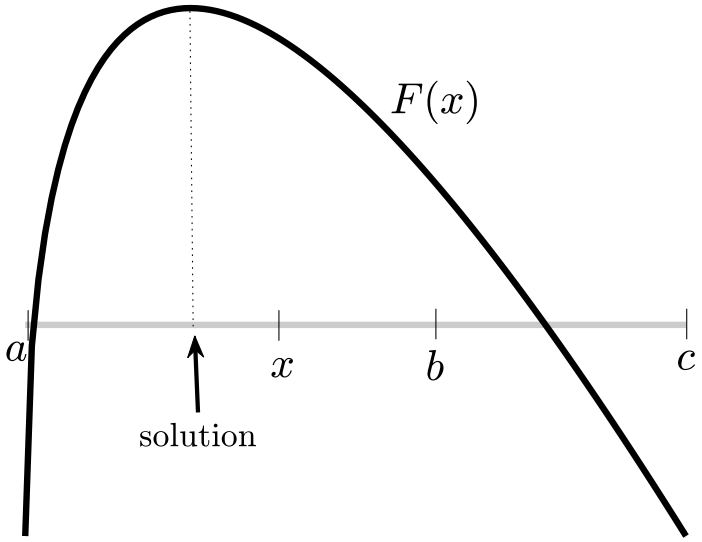

Figure 10.4: Golden section search

2. Take the longer segment between $[a, b]$ and $[b, c]$. Here, for exposition, suppose $(b - a) > (c - b)$; that is, $[a, b]$ is the longer one. Then, take a point x in $[a, b]$ so that $(b - x)/(b - a) = \omega^*$. Here, $\omega^* = (3 - \sqrt{5})/2 \approx 0.38197$ is a constant. As a result, we have now broken our initial interval up into three segments $[a, x]$, $[x, b]$, and $[b, c]$. Now compare $F(x)$ and $F(b)$. If $F(x) > F(b)$, remove the segment $[b, c]$ and create the new bracket (a, x, b). Otherwise, remove the segment $[a, x]$ and create the new bracket (x, b, c). In Figure 10.4, $F(x) > F(b)$, and thus the new bracket becomes (a, x, b).

3. Repeat step 2 until the size of the bracket is less than a tolerance value.

In step 2, we use the parameter ω^*. The ratio $(1 - \omega^*)/\omega^* \approx 1.61803$ is often called the "golden ratio" or "golden section." This ratio has been studied mathematically since ancient Greece and is widely used in the context of architecture and art. Here, ω^* is used because if we start from $(c-b)/(c-a) = \omega^*$ and follow the above procedure, every time, we can remove the ω^* fraction of the segment. After iterating n times, therefore, the length of the remaining bracket (that is, the bracket that contains the maximum point) is $(1 - \omega^*)^n(c - b)$.[3] As in the case of bisection, this method does not require differentiability of $F(x)$, and it ends in a pre-set number of iterations.

10.3.2 Newton's method

The second method starts by approximating the function with a quadratic equation:

$$F(x) \approx F(x_0) + F'(x_0)(x - x_0) + \frac{1}{2}F''(x_0)(x - x_0)^2.$$

[3] Ambitious readers can try to show that this result is guaranteed only with this particular value of ω^*. See Numerical Recipes `http://numerical.recipes/book.html` for further exposition.

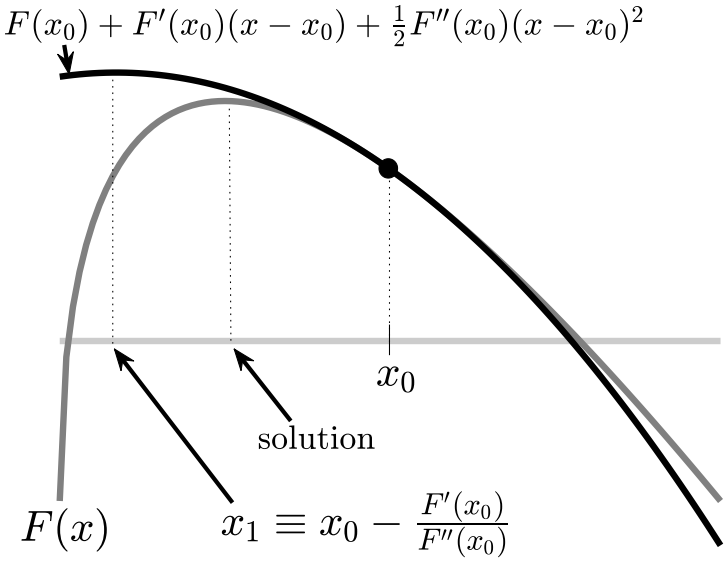

Figure 10.5: Constructing x_1 in the Newton method

Taking the first-order condition of the right-hand side with respect to x and equating it with zero, we obtain the solution for x:

$$\hat{x} = x_0 - \frac{F'(x_0)}{F''(x_0)}. \tag{10.3}$$

If $F(x)$ is indeed quadratic, $\hat{x}$ delivers the solution in one step.

The general procedure is as follows:

1. Make an initial guess x_0.

2. Construct x_1 from
$$x_1 = x_0 - \frac{F'(x_0)}{F''(x_0)}.$$
See Figure 10.5 for an example.

3. Check whether $|x_1 - x_0|$ is less than the pre-set tolerance value ε. If it is indeed the case ($|x_1 - x_0| < \varepsilon$), stop and call x_1 the solution. If not, now use x_1 to obtain x_2, going back to the previous step. Repeat until $|x_{i+1} - x_i| < \varepsilon$.

Compared with the golden-section search, the Newton method can be substantially faster, especially if $F(x)$ is close to quadratic. However, the same issues can arise as for the Newton-Raphson method above: it requires information on the first and second derivatives, and it is not guaranteed to find the maximum point. Starting from a good first guess x_0 is helpful.

10.3.3 Connections between root finding and optimization

Optimization and root finding are closely related. In fact, maximizing a function $F(x)$ typically requires solving the first-order condition $F'(x) = 0$, so that the maximization problem is equivalent to finding the root of $F'(x)$. This equivalence is clear when comparing Newton's method for optimization and the Newton–Raphson method for root finding. The Newton update rule for maximizing $F(x)$ is

$$x_{i+1} = x_i - \frac{F'(x_i)}{F''(x_i)}.$$

Applying the Newton–Raphson method to the equation $F'(x) = 0$ produces exactly the same iteration. Thus, Newton's method for optimization is simply Newton–Raphson applied to the first-order condition.

How does this compare to derivative-free approaches such as golden-section search? Golden-section search, however, appears to be less efficient than "finding a root of the first-order condition using bisection method," because it gets rid of about one-third of the interval in each iteration, as opposed to half. The golden section search provides two additional benefits instead: (i) it does not require $F(x)$ to be differentiable (it has to be differentiable to be able to take the first-order condition), and (ii) in every step, the procedure intends to maximize rather than minimize (one cannot tell the difference from just looking at the first-order condition). In practice, we use Newton–Raphson (or related methods) when F is smooth and derivatives are available; but use golden-section search when derivatives are not available or robustness is more important than speed.

Example 1: Root finding and optimization

Consider the following static labor-leisure choice problem. A consumer faces a problem

$$\max_{c,\ell} u(c, \ell)$$

subject to

$$c = f(\ell) + x,$$

where c is consumption, ℓ is labor supply, $u(\cdot, \cdot)$ is the utility function, $f(\cdot)$ is the production function, and x is other income.

Assume that the utility function is

$$u(c, \ell) = \ln(c) - \frac{\omega}{\gamma}\ell^{\gamma},$$

where $\omega > 0$ and $\gamma > 0$, and the production function is

$$f(\ell) = \ell^{\alpha},$$

where $\alpha \in (0,1)$. With this specification, the problem is to maximize

$$\ln(\ell^\alpha + x) - \frac{\omega}{\gamma}\ell^\gamma \tag{10.4}$$

or solve the first-order condition

$$\frac{\alpha\ell^{\alpha-1}}{\ell^\alpha + x} - \omega\ell^{\gamma-1} = 0. \tag{10.5}$$

Let $\alpha = 1/3$, $x = 0.5$, $\omega = 1$, and $\gamma = 2$. There are three accompanying MATLAB codes. `Ex1_bisection.m` solves the first-order condition (10.5) numerically using the bisection method to find the optimal $\ell \in [0,1]$. `Ex1_newton.m` solves the first-order condition (10.5) numerically using the Newton-Raphson method to find the optimal ℓ. The code for directly maximizing (10.4) with the Newton method would essentially be the same. Finally, `Ex1_GS.m` solves the optimization problem of maximizing (10.4) using golden section search.

10.4 Discretizing an AR(1) process

An AR(1) process is a common way to describe exogenous shock process. Computationally, we often wish to represent such a process with a Markov chain that has a similar persistence and volatility. Constructing a Markov chain with similar properties to an AR(1) is called "discretizing" the AR(1). We introduce two methods of discretizing the process

$$z_{t+1} = \rho z_t + \varepsilon_{t+1},$$

where $\rho \in (0,1)$. The shock ε_{t+1} has the properties $\mathbb{E}_t[\varepsilon_{t+1}] = 0$, $\text{Var}[\varepsilon_{t+1}] = \sigma_\varepsilon^2$, and $\Pr[\varepsilon \leq u] = F(u/\sigma_\varepsilon)$. That is, $F(\cdot)$ is the normalized distribution function for ε_{t+1}. Note this specification implies the unconditional variance of z_t is

$$\sigma_z^2 = \frac{1}{1-\rho^2}\sigma_\varepsilon^2.$$

Our discrete approximation will involve a set of grid points and a transition matrix between them. Let transition matrix have elements π_{ij}, which represents the probability of moving from state i to state j.

10.4.1 The Tauchen method

Tauchen (1986) suggests the following method:

1. Create equally spaced grid points, $\{\bar{z}^1, \bar{z}^2, \ldots, \bar{z}^N\}$. Let the distance between points be ω. A common recommendation is to set $\bar{z}^1 = -m\sigma_z$ and $\bar{z}^N = m\sigma_z$, where m represents how wide the coverage of the approximated process is. When $m = 3$, it covers three standard deviations of the original z_t.

2. Set the probabilities

$$\pi_{ij} = \begin{cases} F\left(\dfrac{\bar{z}^1 - \rho\bar{z}^i + \omega/2}{\sigma_\varepsilon}\right) & \text{for } j = 1 \\ F\left(\dfrac{\bar{z}^j - \rho\bar{z}^i - \omega/2}{\sigma_\varepsilon}\right) - F\left(\dfrac{\bar{z}^j - \rho\bar{z}^i + \omega/2}{\sigma_\varepsilon}\right) & \text{for } j = 2, \ldots, N-1 \\ 1 - F\left(\dfrac{\bar{z}^j - \rho\bar{z}^i - \omega/2}{\sigma_\varepsilon}\right) & \text{for } j = N, \end{cases}$$

This approximation is intuitive: for a given $i \in \{1, \cdots N\}$ and $j \in \{2, \cdots, N-1\}$, we suppose we begin at z_i and we compute the probability that the AR(1) assigns to the interval $[\bar{z}^j - \omega/2, \bar{z}^j + \omega/2]$. We then assign this probability to the transition π_{ij}. The end points $j \in \{1, N\}$ follow a similar logic but account for the fact that the support of z extends beyond the grid.

10.4.2 The Rouwenhorst method

Some researchers have argued the Tauchen approximation can be inaccurate when the persistence parameter ρ is close to 1. This issue may be problematic because many stochastic processes used in macroeconomics (e.g., the individual income process, the aggregate productivity process, or the firm productivity process) are often very persistent. Some researchers argue that the following method, originally proposed by Rouwenhorst (1995), has better approximation properties.

Below, we introduce a special case of Rouwenhorst's (1995) approximation.[4] The first step is the same as the Tauchen method: Create equally spaced grids, $\{\bar{z}^1, \bar{z}^2, \ldots, \bar{z}^N\}$. Note, though, that the upper bound and lower bound have to be set by $\bar{z}^1 = -m\sigma_z$ and $\bar{z}^N = m\sigma_z$, and m is not arbitrary (see below). Then, we construct the Markov matrix Π_N recursively as follows.

1. For $N = 2$,

$$\Pi_2 = \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}.$$

2. For $N \geq 3$, first construct the $N \times N$ matrix

$$\Pi_N = p\begin{bmatrix} \Pi_{N-1} & \mathbf{0} \\ \mathbf{0}' & 0 \end{bmatrix} + (1-p)\begin{bmatrix} \mathbf{0} & \Pi_{N-1} \\ 0 & \mathbf{0}' \end{bmatrix} + p\begin{bmatrix} \mathbf{0}' & 0 \\ \Pi_{N-1} & \mathbf{0} \end{bmatrix} + (1-p)\begin{bmatrix} 0 & \mathbf{0}' \\ \mathbf{0} & \Pi_{N-1} \end{bmatrix},$$

where $\mathbf{0}$ is an $(N-1) \times 1$ zero matrix (column vector). Then, divide all but the top and the bottom rows by two so that the elements in each row are equal to one.

We can set p and m by $p = (1+\rho)/2$ and $m = \sqrt{N-1}$. Then, we can show that the unconditional mean and unconditional variance of the Markov chain are the same as those of the original AR(1) process. This method only uses the information of mean and variance, and the invariant distribution of the Markov process converges to a normal distribution as $N \to \infty$. Thus, the Rouwenhorst method is especially recommended when ε_{t+1} follows a normal distribution. As was mentioned above, this method is particularly powerful when ρ is close to one.

[4] See Kopecky and Suen (2010) for further expositions and characterizations.

Example 2: Approximating an AR(1) process

Consider the following AR(1) process:

$$y_{t+1} = 0.95y_t + \varepsilon_{t+1},$$

where ε_{t+1} follows a normal distribution $N(0, \sigma^2)$ with $\sigma = 0.2$. The code `Ex2_tauchen.m` approximates this process by a 10-state first-order Markov process using the Tauchen method. `Ex2_rouwenhorst.m` performs the same task using the Rouwenhorst method.

10.5 Solving a dynamic programming problem with value function iteration

Many different methods are available for solving a dynamic programming problem and we will cover just one of them: value function iteration. This method is particularly robust because it relies on the contraction mapping theorem (see Chapter 4). Consider the Bellman equation of the form

$$V(k) = \max_{k'} \{F(k, k') + \beta V(k')\}, \tag{10.6}$$

where k is the state variable, k' is the next period value of k, $F(k, k')$ is the return function for the current period, and $\beta \in (0, 1)$ is the discount factor. Now consider the mapping T defined by:

$$TV_i(k) = \max_{k'} \{F(k, k') + \beta V_i(k')\}. \tag{10.7}$$

Given a function $V_i(k)$ the right-hand side of the equation above defines a new function, which we denote $TV_i(k)$. Now consider starting from $i = 0$ with some guess $V_0(k)$ and iterating $V_{i+1}(k) = TV_i(k)$. The contraction mapping theorem guarantees $V_i(k)$ approaches the true value function $V(k)$ for (10.6) as $i \to \infty$, regardless of the starting value function $V_0(k)$. This theorem suggests the following algorithm for finding $V(k)$:

1. Fix a grid on values of k.

2. Start from an arbitrary $V_0(k)$ represented as the function values on the grid for k.

3. For each k on the grid, solve the right-hand side of (10.7) with $V_0(k)$, that is, $\max'_k F(k, k') + \beta V_0(k')$. This step could involve interpolating the values of $V_0(k')$ off of the grid points. Let $V_1(k)$ by this maximized value for each k.

4. Now use $V_1(k)$ on the right-hand side and create $V_2(k)$ as in the previous step. Repeat for $i = 0, 1, 2, ...$ until $V_i(k)$ is similar to $V_{i+1}(k)$.

This algorithm is very robust in the sense that it is based on the theorem that says we will always get to the solution. The downside is that this algorithm can be slow, especially when β is close to one. Many other (potentially faster) algorithms exist, which we don't cover here. Below, we go over this method for several concrete examples.

10.5.1 A deterministic case

Consider the following problem. An infinitely-lived consumer receives a whole cake, whose quantity is a_0, at period 0. The cake can last forever, and the consumer chooses how to eat the cake over time. The consumer's preferences are

$$\sum_{t=0}^{\infty} \beta^t \sqrt{c_t},$$

and the constraint is

$$a_{t+1} = a_t - c_t,$$

where c_t is the consumption of the cake at period t, and a_t is the amount of leftover cake at the beginning of period t. The parameter $\beta \in (0, 1)$ is a discount factor. We can write the consumer's utility maximization problem as a Bellman equation:

$$V(a) = \max_{a'} \sqrt{a - a'} + \beta V(a').$$

The algorithm for solving this problem is as follows:

1. Create a grid for values of a. In this case, the natural lower bound is 0, and the natural upper bound is a_0. Call the resulting vector $\begin{bmatrix} a^1 & a^2 & \cdots & a^N \end{bmatrix}'$, where $a^1 = 0$ and $a^N = a_0$. N is the number of grid points.

2. Create the initial value function $V_0(a)$ on the grid points, that is, the vector of $V_0(a^j)$ $(j = 1, 2, ..., N)$. Because the initial value function is arbitrary, we start from the zero vector: $\begin{bmatrix} V_0(a^1) & V_0(a^2) & \cdots & V_0(a^N) \end{bmatrix}' = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}'$.

3. Using the value function $V_0(a)$ created above, for each grid point a^j, solve the right-hand side of the Bellman equation: $\max_{a'} \sqrt{a^j - a'} + \beta V_0(a')$. One way to solve the maximization problem is to restrict a' to lie on the grid points. Then, because we already know the values of $V_0(a')$ on the grid points for a', one can choose a^n that gives the maximum value of $\sqrt{a^j - a^n} + \beta V_0(a^n)$ (with the restriction that $a^n \leq a^j$). An alternative (and better) way is not to restrict the choice of a' to lie on the grid points. Off the grid points, instead, we can evaluate $V_0(a')$ using the approximation methods we learned in Section 10.1. If $V_0(a')$ is approximated by a differentiable function, we can use the first-order condition and solve the root using the methods in Section 10.2 or we can use the optimization methods in Section 10.3. Note the properties of a contraction mapping ensure the value function is concave in this particular case, and thus, we can use the first-order condition for optimization. When concavity is not obvious, grid search can be a slower but safer optimization method. Store the maximized value as $V_1(a^i)$.

4. Now use $\begin{bmatrix} V_1(a^1) & V_1(a^2) & \cdots & V_1(a^N) \end{bmatrix}'$ on the right-hand side of the Bellman equation, perform the optimization, and create $V_2(a^i)$. In general, one can use the vector of $V_i(a^j)$ to create the vector of $V_{i+1}(a^j)$. Repeat until the vector $V_i(a^j)$ is similar to the vector $V_{i+1}(a^j)$. More concretely, stop when $\max_j |V_{i+1}(a^j) - V_i(a^j)| < \varepsilon$ for a small ε.

Example 3: Deterministic dynamic programming

Consider the above cake-eating problem, with $\beta = 0.98$ and $a_0 = 10$. The provided codes conduct the value function iteration described above and simulate the first 100 periods to plot (i) the time series of the leftover cake a_t, (ii) the time series of consumption c_t, (iii) the value function $V(a)$, and (iv) policy function $a'(a)$. The code `Ex3_gridsearch.m` uses a simple grid search method by restricting the choice a' to be on the grid points for a. The code `Ex3_GS.m` allows the choice for a' to be off the grid points (using the golden section search for optimization) and interpolate the right-hand side values of the Bellman equation for each potential value for a'.

10.5.2 Stochastic cases

Consider a variation of the cake-eating problem. Assume that there is uncertainty: at the beginning of every period t, an additional cake z arrives. The amount of z is stochastic and can be z_H ("high") or z_L ("low": $z_L < z_H$), following a Markov process. The probability of the next-period state is z' given the current-period state is z is $\pi_{zz'}$. Now, the consumer maximizes the expected utility

$$\mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t \sqrt{c_t} \right],$$

where $\mathbb{E}_0[\cdot]$ represents the expectation at period 0, and the constraint is now

$$a_{t+1} = a_t - c_t + z_t.$$

The Bellman equation is

$$V(a, z) = \max_{a'} \sqrt{a - a' + z} + \beta \mathbb{E} \left[V(a', z') | z \right],$$

where $\mathbb{E}\left[\cdot|z\right]$ is the conditional expectation given the today's state z. Given the probability structure, we can rewrite it as

$$V(a, z) = \max_{a'} \sqrt{a - a' + z} + \beta \left[\pi_{zH} V(a', H) + (1 - \pi_{zH}) V(a', L) \right].$$

We can still apply the same procedure as the deterministic case. The only difference is that (i) the natural upper bound of the grids is not a_0, given that now a_t can potentially increase; (ii) instead of the vector $V_i(a^j)$, we need to work with two vectors $V_i(a^j, H)$ and $V_i(a^j, L)$ (or a matrix $V_i(a^j, z)$, where $z = H, L$).

As another example, let us consider a version of the stochastic neoclassical growth model introduced in Chapter 7.3. An important difference here is that we allow for elastic labor supply. The representative consumer's utility function is

$$\mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t \left((1 - \phi) \log(C_t) + \phi \log(1 - H_t) \right) \right] \tag{10.8}$$

and the resource constraint is

$$K_{t+1} + C_t = \exp(z_t) K_t^{\alpha} H_t^{1-\alpha} + (1-\delta) K_t. \tag{10.9}$$

Here, K_t is capital (which is the predetermined variable), C_t is consumption (non-predetermined), and $H_t \in [0,1)$ is labor supply (non-predetermined). z_t follows the stochastic process

$$z_{t+1} = \rho z_t + \varepsilon_{t+1}, \tag{10.10}$$

where ε_{t+1} is an i.i.d. random variable with mean zero and standard deviation σ. Here, $\beta \in (0,1)$, $\phi \in (0,1)$, $\alpha \in (0,1)$, $\delta \in (0,1)$, $\rho \in (0,1)$, and $\sigma > 0$ are parameters. We consider the social planner's problem of maximizing utility subject to the resource constraint.[5]

In the recursive formulation, the planner's problem can be written as

$$V(K,z) = \max_{K',H} (1-\phi) \log \left(\exp(z) K^{\alpha} H^{1-\alpha} + (1-\delta)K - K' \right) + \phi \log(1-H) + \beta \mathbb{E} \left[V(K',z') | z \right]. \tag{10.11}$$

This equation is very similar to the above cake-eating problem. Only two points are different. First, because the domain of z is now a real line, one has to create discrete grid points, and the stochastic process (10.10) has to be discretized using, for example, the methods introduced above. Second, the planner also chooses a second variable, H.

We can first solve

$$\max_{H} (1-\phi) \log \left(\exp(z) K^{\alpha} H^{1-\alpha} + (1-\delta)K - K' \right) + \phi \log(1-H) \tag{10.12}$$

for a given (z, K, K'). The solution to this problem can be written as $H(z, K, K')$. Note $H(z, K, K')$ may not be an analytically explicit function—we only need the numerical value of H corresponding to a given (z, K, K'). Then, plug this function in on the right-hand side, giving us

$$\begin{aligned} V(K,z) = \max_{K'} \quad & (1-\phi) \log \left(\exp(z) K^{\alpha} H(z,K,K')^{1-\alpha} + (1-\delta)K - K' \right) \\ & + \phi \log(1 - H(z,K,K')) + \beta \mathbb{E} \left[V(K',z') | z \right]. \end{aligned}$$

Now, we can use the same procedure as for the cake-eating problem.

Example 4: A stochastic neoclassical growth model

Consider the Brock-Mirman model (10.11). Let the stochastic process for $Z_t = \exp(z_t)$ be a two-state Markov process, with two values $H = 1.015$ and $L = 0.985$. The transition probability from i to j $(i, j = L, H)$ is π_{ij}. The parameter values are as follows: $\beta = 0.99$, $\pi_{HH} = \pi_{LL} = 0.95$, $\pi_{LH} = \pi_{HL} = 0.05$, $\alpha = 0.36$, and $\delta = 0.025$. The value of ϕ is set so that the steady-state value of H_t is $1/3$. The file `Ex4_gridsearch.m` restricts the choice of K' to be on the grid and uses grid search in the optimization of K'. The solution of H in (10.12) is obtained using the bisection method. After solving the Bellman equation, the program simulates the time series of Y_t, C_t, H_t, $I_t = K_{t+1} - (1-\delta)K_t$, and Z_t, for 3000 periods. After eliminating the first 100 periods, the code logs and HP-filters (using the subroutine `hpfilt.m`; see Chapter 14) each time series and compute the standard

[5] Because of the first and the second welfare theorem, the social planner's solution corresponds to the equilibrium outcome under perfect competition (when everyone has the same wealth).

deviation and the correlation with Y_t for the cyclical component of each variable. The file `Ex4_GS.m` repeats the same exercise (using the subroutines `Ex4_location.m` and `Ex4_labor.m`), allowing the choice of K' to be outside the grid points using the linear interpolation and the golden section search.

10.6 Solving a dynamic model with linearization

The previous section solved the dynamic programming problem on a grid of state variables. One would ideally construct the grid to cover the entire portion of the state space where the economy travels.[6] Solution methods of this type are often called "global methods" as they seek to approximate the solution at all relevant state variables. "Local methods" are a different approach in which we choose one point in the state space and examine how the solution changes locally around that point. We often take the deterministic steady state of the economy as our starting point and we can then ask, for example, how consumption changes following a small change in the capital stock. The approach we use is a perturbation method, which approximates the local behavior of the system around the steady state using simple functions, typically polynomials. In macroeconomics, various degrees of polynomials are used, but in this book, we will only cover the linear (or log-linear) approximation.

These methods are effective to the extent that the linear approximation is accurate. Thus, this method is mainly used for small deviations from the steady state. The deviation can occur for various reasons, but below we consider a situation where the deviation is due to a random AR(1) shock. In the stochastic neoclassical growth model, for example, we know that capital and consumption converge to their steady-state values when there are no shocks. Thus, it is reasonable to analyze the local dynamics around the steady state when the economy is subject to shocks. In particular, when the shocks are small, a linear solution can yield a reasonable approximation of the true solution.

The benefit of using a linearized system is that it is very fast compared to other methods, especially when the model involves many variables. Directly solving a dynamic programming problem, as in Example 4, is subject to the "curse of dimensionality"—when the number of variables, especially the number of endogenous state variables, is large, solving the problem becomes computationally burdensome very quickly. The solution methods of a large linear system using matrix algebra are well-established, and one can obtain the solution relatively quickly. In addition, because the expectations operator has the affine property, a linear solution has the "certainty-equivalence" property. That is, we can solve the model with uncertainty as if it were a deterministic system with corresponding expected values. The dynamics of the system are effectively the same whether there are uncertain shocks or the shock variables move around deterministically.

Below, we explain the principles of solving a linear (or linearized/log-linearized) rational expectations model. Versions of the methods we discuss below (and more advanced procedures) are incorporated into popular software packages such as Dynare (`https://www.dynare.org/`) and are very widely used.

[6] This may not always be possible. For example, if an exogenous state variable has a Gaussian distribution, there could be a realization of this variable that is very extreme. However, one would normally choose the grid so that the economy's state variables are inside the grid with high probability.

When the model is already linear, we can use the methods below directly. When the model is nonlinear, we first linearize or log-linearize the model. We will employ log-linearization below. We have already introduced the idea of log-linearization in Chapter 3. As a result of log-linearization, the equilibrium can be expressed as a system of linear equations with variables that represent log deviations from the steady-state values. Below, suppose we have a system of linear equations that are obtained from such procedures.

Here, we distinguish between two types of variables: predetermined variables and non-predetermined (jump) variables. Predetermined variables are variables whose values are already determined when entering period t. An example is the capital stock in the neoclassical growth model. Non-predetermined variables are variables whose values are determined within the period. An example is consumption in the neoclassical growth model.

What do we mean by "solving the model"? Here, our goal is to characterize the movement of endogenous variables as an explicit function of the predetermined variables and exogenous variables. As an example, let us consider the stochastic neoclassical growth model introduced earlier. Recall that the representative consumer's utility is given by (10.8) and the constraint is (10.9). The shocks are given by (10.10). Our eventual goal is to obtain the solution of this model, that is, representing K_{t+1}, C_t, H_t as functions of K_t and z_t.

The first-order conditions lead to the Euler equation

$$\frac{1}{C_t} = \mathbb{E}_t \left[\beta(1 + \alpha \exp(z_{t+1}) K_{t+1}^{\alpha-1} H_{t+1}^{1-\alpha} - \delta) \frac{1}{C_{t+1}} \right] \tag{10.13}$$

and the labor-supply condition

$$\frac{1-\phi}{C_t}(1-\alpha)\exp(z_t)K_t^{\alpha}H_t^{-\alpha} = \frac{\phi}{1-H_t}. \tag{10.14}$$

Log-linearizing (10.9), (10.13), and (10.14), we obtain

$$\bar{K}k_{t+1} + \bar{C}c_t = \bar{K}^{\alpha}\bar{H}^{1-\alpha}(z_t + \alpha k_t + (1-\alpha)h_t) + (1-\delta)\bar{K}k_t,$$

$$-c_t = \mathbb{E}_t[\beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(z_{t+1} + (\alpha-1)k_{t+1} + (1-\alpha)h_{t+1}) - c_{t+1}],$$

and

$$h_t = \theta(z_t + \alpha k_t - c_t), \tag{10.15}$$

where

$$\theta \equiv \frac{1-\bar{H}}{\bar{H}(1-\alpha)+\alpha}.$$

Here, the lower-case letter is the log deviation from the steady state, that is,

$$x_t \equiv \log\left(\frac{X_t}{\bar{X}}\right)$$

for a variable X_t, where $\bar{X}$ is the steady-state value.[7] For future use, let us use (10.15) to eliminate h_t from the first two equations. Rearranging, we obtain

$$\bar{K}k_{t+1} = \bar{K}^{\alpha}\bar{H}^{1-\alpha}(1+\theta(1-\alpha))z_t + (\bar{K}^{\alpha}\bar{H}^{1-\alpha}\alpha(1+\theta(1-\alpha)) + (1-\delta)\bar{K})k_t - (\bar{K}^{\alpha}\bar{H}^{1-\alpha}\theta(1-\alpha) + \bar{C})c_t, \tag{10.16}$$

[7] In log-linearizing (10.13), we applied the log-linearization technique ignoring the expectation parameter on the right-hand side. This step exploits the certainty equivalence property discussed in Chapter 7.3.4.

and (using $\mathbb{E}_t[z_{t+1}] = \rho z_t$ from (10.10))

$$\begin{aligned}\beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(1-\alpha)(1-\theta\alpha)k_{t+1} + (\beta\alpha\theta(1-\alpha)\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}+1)\mathbb{E}_t[c_{t+1}] \\ = \beta(1+\theta(1-\alpha))\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}\rho z_t + c_t.\end{aligned} \tag{10.17}$$

10.6.1 The Blanchard-Kahn condition

Blanchard and Kahn (1980) derive a condition for the uniqueness of the equilibrium in linear rational expectation models. This condition, called the Blanchard-Kahn condition, also helps solve the model. Before introducing the general result, let us fix the idea with a few examples.

One non-predetermined variable

Consider the dynamic system with one non-predetermined variable y_t:

$$y_t = \phi y_{t+1}, \tag{10.18}$$

where $\phi > 0$ is a parameter. For convenience, let us rewrite this relationship

$$y_{t+1} = \lambda y_t, \tag{10.19}$$

where $\lambda = 1/\phi$. The reason we wrote (10.18) first is to emphasize this relationship is not to imply y_{t+1} is predetermined by y_t. One can imagine the relationship between consumption today and consumption tomorrow in the consumer's Euler equation.

Suppose that one of the equilibrium conditions is that y_t does not "blow up," that is, $\lim_{T\to\infty} y_{t+T}$ remains finite. Two questions can be asked here. First, what condition on ϕ would guarantee the uniqueness of the equilibrium? Second, under that condition, what are the properties of the equilibrium?

Applying (10.19) repeatedly, we obtain

$$y_{t+T} = \lambda^T y_t.$$

When $\lambda > 1$, the only possible value of y_t for $\lim_{T\to\infty} y_{t+T}$ remaining finite is $y_t = 0$. Thus, the equilibrium is unique. When $\lambda \leq 1$, any value of y_t is consistent with $\lim_{T\to\infty} y_{t+T}$ being finite. This situation is often called "indeterminacy." Thus, the condition for the uniqueness is $\lambda > 1$, and the solution is

$$y_t = 0 \text{ for all } t. \tag{10.20}$$

Once again, the particular logic here is important: to guarantee the uniqueness of the equilibrium, λ has to be such that if we don't choose the right y_t, the economy will eventually "blow up" meaning the variables will diverge. Then, the only "right" y_t is the unique solution, which is (10.20) in this case.

One non-predetermined variable and one predetermined (exogenous) variable

Now, assume y_t is stochastic due to the influence of the exogenous (and predetermined) shock z_t. Consider the dynamic system

$$y_t = \phi \mathbb{E}_t[y_{t+1}] + \hat{z}_t,$$

where $\phi > 0$ is a parameter, $\hat{z}_t$ is a shock that follows

$$\hat{z}_{t+1} = \rho \hat{z}_t + \hat{\varepsilon}_{t+1}$$

with $\rho \in (0, 1)$, and $\hat{\varepsilon}_{t+1}$ is an i.i.d. random variable whose mean is 0 and the standard deviation is $\hat{\sigma} > 0$. Here, $\hat{z}_t$ is predetermined; that is, the value of $\hat{z}_t$ is determined at the value of $\hat{z}_{t-1}$ and a shock $\hat{\varepsilon}_t$, which is revealed at the beginning of period t.

Again, we rewrite

$$\mathbb{E}_t[y_{t+1}] = \lambda y_t + z_t, \tag{10.21}$$

where $\lambda = 1/\phi$ and $z_t = -\hat{z}_t/\phi$. Therefore,

$$z_{t+1} = \rho z_t + \varepsilon_{t+1}, \tag{10.22}$$

where ε has mean zero and standard deviation $\sigma = \hat{\sigma}/\phi$.

Let us look for the solutions where $\lim_{T\to\infty} E_t[y_{t+T}]$ is finite. The relationship (10.21) implies

$$\mathbb{E}_{t+1}[y_{t+2}] = \lambda y_{t+1} + z_{t+1},$$

and thus,

$$\mathbb{E}_t[y_{t+2}] = \mathbb{E}_t[\mathbb{E}_{t+1}[y_{t+2}]] = \lambda \mathbb{E}_t[y_{t+1}] + \mathbb{E}_t[z_{t+1}] = \lambda^2 y_t + \lambda z_t + \rho z_t = \lambda^2 \left[y_t + \frac{1}{\lambda}\left[1 + \frac{\rho}{\lambda}\right] z_t \right],$$

where the first equality utilizes the law of iterated expectations. Repeating this procedure (assuming $\lambda \neq \rho$),

$$\mathbb{E}_t[y_{t+T}] = \lambda^T \left[y_t + \frac{1}{\lambda}\left[1 + \frac{\rho}{\lambda} + \left(\frac{\rho}{\lambda}\right)^2 + \cdots + \left(\frac{\rho}{\lambda}\right)^{T-1}\right] z_t \right] = \lambda^T \left[y_t + \frac{1 - (\rho/\lambda)^T}{\lambda - \rho} z_t \right].$$

Thus, when $\lambda > 1$, the only situation where $\lim_{T\to\infty} E_t[y_{t+T}]$ is finite is when

$$y_t = -\lim_{T\to\infty} \frac{1 - (\rho/\lambda)^T}{\lambda - \rho} z_t = -\frac{1}{\lambda - \rho} z_t, \tag{10.23}$$

and this expression gives the unique solution of y_t. When $\lambda \leq 1$, indeterminacy exists, because many values of y_t can keep $\lim_{T\to\infty} E_t[y_{t+T}]$ finite.[8]

[8] In the above expression for $\mathbb{E}_t[y_{t+T}]$,

$$\lambda^T \left[y_t + \frac{1 - (\rho/\lambda)^T}{\lambda - \rho} z_t \right] = \lambda^T y_t + \frac{\lambda^T - \rho^T}{\lambda - \rho} z_t,$$

which would remain finite when $\lambda \leq 1$ (note we assumed $\rho < 1$).

The "guess and verify" method (the method of undetermined coefficients)

If we *know* the solution is unique, we can use the "guess and verify" (the method of undetermined coefficients) to find the solution, similarly to how we proceeded in Section 4.4.4. Suppose ("guess") that the solution of (10.21) takes the form

$$y_t = \mathcal{A} z_t, \tag{10.24}$$

where $\mathcal{A}$ is an unknown constant. Plugging this equation into (10.21), we obtain

$$\mathbb{E}_t[\mathcal{A} z_{t+1}] = \lambda \mathcal{A} z_t + z_t. \tag{10.25}$$

From (10.22), the left-hand side of (10.25) is equal to $\mathcal{A}\rho z_t$, and thus we can rearrange (10.25) as

$$(\mathcal{A}\rho - \mathcal{A}\lambda - 1) z_t = 0.$$

This equality has to hold for all z_t; thus, $\mathcal{A}\rho - \mathcal{A}\lambda - 1$ must be zero. Therefore,

$$\mathcal{A} = -\frac{1}{\lambda - \rho},$$

which, in the equation (10.24), yields the same solution as (10.23). This solution indeed satisfies (10.21) ("verify").

m non-predetermined variables and $n + k$ predetermined variables

Now, let us consider a more general situation. Suppose the model now has m non-predetermined variables and $n + k$ predetermined variables. Predetermined variables include n endogenous variables, such as K_t in the growth model example, and k exogenous variables, such as z_t in the growth model. Suppose the model can be expressed as the system of equations

$$B \begin{bmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{bmatrix} = A \begin{bmatrix} x_t \\ y_t \end{bmatrix} + E a_t, \tag{10.26}$$

where x_t is an $n \times 1$ vector of endogenous predetermined variables, y_t is an $m \times 1$ vector of non-predetermined variables, and B and A are $(n + m) \times (n + m)$ matrices.[9] The vector a_t represents exogenous predetermined variables, which is a $k \times 1$ vector. E is an $(n + m) \times k$ matrix. Below, we consider a situation where each element of a_t follows the AR(1) process

$$a^i_{t+1} = \rho a^i_t + \varepsilon^i_{t+1},$$

where a^i_t is the element i of a_t vector, $\rho \in [0, 1)$ is the common persistence parameter, and ε_{t+1} is the mean zero i.i.d. shock for element i.[10]

[9] Some models cannot be expressed in this manner, but a broad class of macroeconomic models can be transformed to fit into this expression (after linearizing or log-linearizing).

[10] A common ρ is assumed here for the ease of exposition.

In our stochastic growth example, $x_t = k_t$, $y_t = c_t$, and $a_t = z_t$. In matrix form, (10.16) and (10.17) can be expressed as

$$B \begin{bmatrix} k_{t+1} \\ \mathbb{E}_t[c_{t+1}] \end{bmatrix} = A \begin{bmatrix} k_t \\ c_t \end{bmatrix} + E z_t, \tag{10.27}$$

where

$$B = \begin{bmatrix} \bar{K} & 0 \\ \beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(1-\alpha)(1-\theta\alpha) & \beta\alpha\theta(1-\alpha)\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}+1 \end{bmatrix},$$

$$A = \begin{bmatrix} \bar{K}^{\alpha}\bar{H}^{1-\alpha}\alpha(1+\theta(1-\alpha))+(1-\delta)\bar{K} & -(\bar{K}^{\alpha}\bar{H}^{1-\alpha}\theta(1-\alpha)+\bar{C}) \\ 0 & 1 \end{bmatrix},$$

and

$$E = \begin{bmatrix} \bar{K}^{\alpha}\bar{H}^{1-\alpha}(1+\theta(1-\alpha)) \\ \beta(1+\theta(1-\alpha))\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}\rho \end{bmatrix}.$$

Suppose B is invertible in (10.26).[11] Then we can rewrite it as

$$\begin{bmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{bmatrix} = F \begin{bmatrix} x_t \\ y_t \end{bmatrix} + G a_t, \tag{10.28}$$

where $F = B^{-1}A$ and $G = B^{-1}E$. The matrix F can be decomposed into (Jordan decomposition)

$$F = HJH^{-1}. \tag{10.29}$$

The matrix J is a diagonal matrix with eigenvalues of F on the diagonal, and H is a matrix that consists of the corresponding eigenvectors:

$$J = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{n+m} \end{bmatrix}$$

and

$$H = \begin{bmatrix} v_1 & v_2 & \cdots & v_{n+m} \end{bmatrix}.$$

(For exposition, we are limiting ourselves to the situation where eigenvalues are real and distinct.) Here, we line up the eigenvalues with the ascending order of the absolute value: $|\lambda_1| < |\lambda_2| < \cdots < |\lambda_{n+m}|$. Let h denote the number of eigenvalues that satisfies $|\lambda_i| > 1$. Blanchard and Kahn (1980) show that the equilibrium (i.e., the solutions for x_t and y_t that don't blow up in the future) is unique if $h = m$. That is, the number of eigenvalues that are outside the unit circle is the same as the number of non-predetermined variables. This condition is often called the Blanchard-Kahn condition.

[11] In general, B may not be invertible. For example, in our stochastic growth example, if we did not use (10.15) to eliminate h_t earlier and treated h_t as another non-predetermined variable, the resulting 3×3 matrix B would not have been invertible. In such a case, we can use the Generalized Schur decomposition (also called the QZ decomposition) instead of the Jordan decomposition below. See, for example, Heer and Maußner (2024), Chapters 2 and 3.

The intuition is the same as in the earlier case with one non-predetermined variable. Here, we deal with the matrix F. In general, a matrix multiplication to a vector is a combination of expanding (or contracting) and rotating the vector. The eigenvalues tell us whether the multiplication expands ($|\lambda_i| > 1$) or contracts ($|\lambda_i| < 1$) the vector in the directions that are indicated by the eigenvectors. When $h = m$, we can choose a unique set of non-predetermined variables such that the vector $\begin{bmatrix} x_t' & y_t' \end{bmatrix}'$ is "zero" in the direction of expansions. The following procedure can identify such a $\begin{bmatrix} x_t' & y_t' \end{bmatrix}'$.

Using (10.29) on (10.28) and multiplying H^{-1} from the left,

$$H^{-1} \begin{bmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{bmatrix} = JH^{-1} \begin{bmatrix} x_t \\ y_t \end{bmatrix} + H^{-1}Ga_t. \tag{10.30}$$

Let us partition H^{-1} and J with n and m rows and n and m columns and call

$$H^{-1} = \begin{bmatrix} \tilde{H}_{11} & \tilde{H}_{12} \\ \tilde{H}_{21} & \tilde{H}_{22} \end{bmatrix}$$

and

$$J = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}.$$

Note J_1 and J_2 are both diagonal matrices. Let

$$H^{-1}G = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix},$$

where Γ_1 is an $n \times k$ matrix and Γ_2 is an $m \times k$ matrix. Let

$$\begin{bmatrix} \tilde{x}_t \\ \tilde{y}_t \end{bmatrix} = \begin{bmatrix} \tilde{H}_{11} & \tilde{H}_{12} \\ \tilde{H}_{21} & \tilde{H}_{22} \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix}. \tag{10.31}$$

Then (10.30) can be rewritten as

$$\begin{bmatrix} \tilde{x}_{t+1} \\ \mathbb{E}_t[\tilde{y}_{t+1}] \end{bmatrix} = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix} \begin{bmatrix} \tilde{x}_t \\ \tilde{y}_t \end{bmatrix} + \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix} a_t.$$

Now let us look at the last m elements of the equation:

$$\mathbb{E}_t[\tilde{y}_{t+1}] = J_2\tilde{y}_t + \Gamma_2 a_t.$$

One can immediately see the similarities to (10.21). In fact, because J_2 is diagonal, we can apply exactly the same steps as before to each element of this set of equations and show that the solution (that guarantees $\mathbb{E}_t[\tilde{y}_{t+T}]$ remains finite as $T \to \infty$) is

$$\tilde{y}_t = \Lambda\Gamma_2 a_t,$$

where Λ is an $m \times m$ diagonal matrix whose diagonal elements are $-1/(\lambda_i - \rho)$ (i runs from $n+1$ to $n+m$). From (10.31), this expression for $\tilde{y}_t$ implies

$$\Lambda\Gamma_2 a_t = \tilde{H}_{21}x_t + \tilde{H}_{22}y_t.$$

Therefore, the solution for y_t is

$$y_t = -\tilde{H}_{22}^{-1}\tilde{H}_{21}x_t + \tilde{H}_{22}^{-1}\Lambda\Gamma_2 a_t.$$

From (10.28),

$$x_{t+1} = F_{11}x_t + F_{12}y_t + G_1 a_t = (F_{11} - F_{12}\tilde{H}_{22}^{-1}\tilde{H}_{21})x_t + (G_1 + F_{12}\tilde{H}_{22}^{-1}\Lambda\Gamma_2)a_t.$$

Thus, we obtained the solution.

Example 5: Solving the log-linearized stochastic neoclassical growth model

Consider the log-linearized stochastic neoclassical growth model (10.27). The parameter values are the same as Example 4, except that the shock process follows (10.10) with $\rho = 0.95$. The code `Ex5_BK.m` follows the above steps to solve the model. One can see that in the matrix J, $J(1,1) = 0.9537$ and $J(2,2) = 1.0592$, so that the Blanchard-Kahn condition is satisfied. The solution is

$$c_t = 0.5691k_t + 0.3920z_t,$$

$$k_{t+1} = 0.9537k_t + 0.1132z_t,$$

and

$$h_t = -0.2431k_t + 0.7070z_t.$$

The code also plots the log-linearized impulse response functions (see Section 3.5.2) and also compute the same HP-filtered statistics as in Example 4, with $\sigma = 0.007$.