Macroeconomics 1 - lecture notes 3

Niku Määttänen

September 23, 2018

1 Dynamic programming and recursive competive equilibrium

This lecture note is, first of all, a VERY brief introduction to dynamic programming.[1] We also present an example of a recursive competitive equilibrium, a concept that is closely related to dynamic programming and often used in contemporary macroeconomics.

For pedagogical reasons, we discuss dynamic programming in the context of the neoclassical growth model presented in lecture notes 1. We start by considering a version of the model that contains productivity shocks and explain why the solution methods we used previously no longer suffice. We then discuss dynamic programming in the context of simpler examples where dynamic programming is not as essential. Then we get back to the neoclassical growth model with productivity shocks and show how to formulate the problem recursively so as to be able apply dynamic programming.

[1]For an introduction to the underlying mathematical theory behind dynamic programming, I recommend Acemoglu's textbook *Introduction to Modern Economic Growth* (MIT press). *Recursive Macroeconomic Theory* by Ljunkvist and Sargent (MIT press) provides many different applications of dynamic programming in macroeconomics. For different computational methods and recipes related to dynamic programming, I recommend *Dynamic General Equilibrium Modeling* by Heer and Maussner (Springer)

Finally we define the recursive competive equilibrium for a stochastic version of the neoclassical growth model.

1.1 Neoclassical growth model with productivity shocks

Consider the problem of a social planner that maximizes

$$E\sum_{t=1}^{\infty}\beta^{t-1}u(c_t, 1-n_t) \tag{1}$$

subject to

$$c_t + k_{t+1} = z_t f(k_t, n_t) + (1-\delta)k_t \tag{2}$$

where E is the expectation operator and z_t is an aggregate productivity shock. For concreteness, assume that $z_t \in \{z^1, z^2\}$ and let $P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}$ denote a probability transition matrix such that

$$\Pr(z_{t+1} = z^j \mid z_t = z^i) = P_{ij} = p_{ij}. \tag{3}$$

Given our interpretation of P as a probability matrix, its rows must sum up to one (and all its entries must be non-negative). We can thus write it as

$$P = \begin{pmatrix} p_{11} & 1-p_{11} \\ 1-p_{22} & p_{22} \end{pmatrix}. \tag{4}$$

Think about solving this problem. In lecture notes 1, we considered deterministic models and found the solution by a solving system of equations that consisted of the first-order conditions and aggregate resources constraints for $t = 1, 2, ..., T$, where T was assumed to be large enough so that the economy should be in a steady state by then.

That approach does not work here. To see this, note that optimal consumption, labor supply and investment in any period presumably depend on the whole history of shocks until that period. (They cannot depend on aggregate shocks after that period, because we assume that not even the social planner knows future productivity shocks in advance.) That is, the optimal consumption in period t, for instance, should be written as $c_t(z_1, z_2, ..., z_t)$. As t increases, the number of different possible shock histories increases rapidly: By period 10, for instance, there are already 2^{10} different histories. Moreover, there is generally no period T after which the economy would be in a steady state.

So how do we solve this problem? The answer is dynamic programming.

1.2 A deterministic finite-horizon example

Consider the following problem:

$$\max_{\{k_{t+1}\}_{t=1}^{T}} \sum_{t=1}^{T} \beta^{t-1} u(c_t) \tag{5}$$

subject to

$$c_t = f(k_t) - k_{t+1} \tag{6}$$

$$k_{t+1} \geq 0 \tag{7}$$

$$k_1 \textit{ given} \tag{8}$$

Here we assume that capital depreciates fully from one period to the next. Following the approach we took in the first two lecture notes, we could solve this problem by solving the following system of equations:

$$u'(f(k_t) - k_{t+1}) = \beta f'(k_{t+1}) u'(f(k_{t+1}) - k_{t+2}), \; \textit{for } t = 1, 2, ..., T-1 \tag{9}$$

$$k_{T+1} = 0 \tag{10}$$

Alternatively, we can solve the problem recursively starting from the last period. In period T, the social planner takes the capital stock as given. The optimal investment policy, as a function of the capital stock, is obviously $k_{T+1}(k_T) = 0$. Let $V_T(k_T)$ denote the last period utility given the optimal policy and capital stock. We have $V_T(k_T) = u(f(k_T))$. In period $T-1$, the social planner takes k_{T-1} as given and its problem is:

$$\max_{k_T} \{u(f(k_{T-1}) - k_T) + \beta u(f(k_T))\} \tag{11}$$

Denoting the value to this problem by $V_{T-1}(k_{T-1})$ and substituting $V_T(k_T) = u(f(k_T))$, we have

$$V_{T-1}(k_{T-1}) = \max_{k_T}\{u(f(k_{T-1}) - k_T) + \beta V_T(k_T)\}. \quad (12)$$

We can proceed backwards to any period $t \geq 1$ and write

$$V_t(k_t) = \max_{k_{t+1}}\{u(f(k_t) - k_{t+1}) + \beta V_{t+1}(k_{t+1})\} \quad (13)$$

This is an example of a Bellmann equation. Here V_t is the value function, k_t is a state variable and k_{t+1} is the control or policy variable. The value function V_t gives the discounted sum of periodic utilities (payoffs) given the value of the state variable. Solving for this dynamic programming problem involves finding the value functions V_t (for all t) and the policy functions $k_{t+1}(k_t)$. This can be done recursively by starting from the last period: we already know V_T and $k_{T+1}(k_T)$, so next we would solve V_{T-1} and $k_T(k_{T-1})$.

It is useful to show explicitly that the allocation resulting from the recursive optimization is the same as the one characterized by (9) and (10). The first-order condition related to (13) in period t reads as

$$u'(f(k_t) - k_{t+1}) = \beta V'_{t+1}(k_{t+1}). \quad (14)$$

This equation determines $k_{t+1}(k_t)$. By plugging in the optimal policy to (13), we get

$$V_t(k_t) = \{u(f(k_t) - k_{t+1}(k_t)) + \beta V_{t+1}(k_{t+1}(k_t))\}. \quad (15)$$

Differentiating this w.r.t. k_t gives us:

$$V'_t(k_t) = \{u'(f(k_t) - k_{t+1}(k_t))(f'(k_t) - k'_{t+1}(k_t)) + \beta V'_{t+1}(k_{t+1}(k_t))k'_{t+1}(k_t)\} \quad (16)$$

Using (14) this simplifies to

$$V'_t(k_t) = u'(f(k_t) - k_{t+1}(k_t))f'(k_t) \quad (17)$$

Forwarding (17) by one period and substituting into (14) results in (9).

1.3 A deterministic infinite-horizon example

Consider the same problem as above but with $T = \infty$. Again, let $V_t(k_t)$ denote the discounted sum of periodic utilities from period t onwards given optimal policies. However, since the horizon is now infinite, it is clear that this function must be time-invariant. Related to that, it makes sense to drop the time indices from everywhere. Hence, we write the Bellmann equation in the following form:

$$V(k) = \max_{k'} \{u(f(k) - k') + \beta V(k')\}, \tag{18}$$

where prime refers to next period variables (not to be confused with a derivative!).

This is an example of a functional fixed point problem. We are looking for a function $V(k)$ such that (18) holds for every k. Unlike in the finite horizon example, there is no last period from which to start with.

However, as we will see, we can still solve this problem recursively by first guessing a value function and then iterating until the value function has converged. That is, given a first guess for the value function (for instance $V(k) \equiv 0$), which we put into right hand side of the Bellman equation, we solve for the optimal k' (in principle for every k). This gives us a new guess for the value function (as the value function in the left-hand side), which we then take as our new guess (and put into the right-hand side). This is the most straightforward approach to solve the Bellman equation and it is called value function iteration.

Sometimes we can solve for the value- and policy functions analytically. This is case in this example if we assume $u(c) = \log(c)$ and $f(k, n) = Ak^{\alpha}$. The answer turns out to be of the form

$$V(k) = E + F \log(k) \tag{19}$$

$$k'(k) = \frac{\beta F}{1 + \beta F} A k^{\alpha} \tag{20}$$

where E and F are constants.

1.4 A stochastic infinite-horizon example

We can now get back to the stochastic neoclassical growth model defined in (1)-(2). The Bellmann equation is very similar to (18). However, here we need to optimize over both k' and n. In addition, we need to include a second state variable, namely the current productivity level. This is because generally, information about the current productivity is useful when forming the expectation about next period productivity. Hence, we have

$$V(k,z) = \max_{k',n}\{u(c,n) + \beta E_{z'|z} V(k',z')\} \tag{21}$$

$$s.t. \tag{22}$$

$$c + k' = zf(k,n) + (1-\delta)k \tag{23}$$

Given k', computing the expectation is easy, of course. For instance, if $z = z^1$ we have $E_{z'|z}V(k',z') = p_{11}V(k',z^1) + (1-p_{11})V(k',z^2)$.

1.5 About the general theory of dynamic programming

The fundamental questions related to the Bellman equations of the form (21) are the following: i) Does the value function exist?, ii) Is the value function unique?, iii) Is the policy function unique? We would also like to know about some basic properties of the value function. For instance, whether it is differentiable or not (notice that to derive some of the results above, we needed to assume that it is differentiable).

The key theorems related to existence and uniqueness (in infinite horizon problems) are contraction mapping theorems, which tell us under which conditions there is unique solution to a functional fixed point problem. They can be used to show that in the case of (18) and (21) the answer to the first two questions is "yes" as long as $\beta < 1$. (The "contraction mapping" in our example, is the act of multiplying the value function by β, taking the expectations, adding the periodic utility to it, and optimizing over k'.) The contraction mapping theorems also tells us that value function iteration works.

The uniqueness of the value function implies that the policy function is unique unless two policies result in the same utility. In economic applications, this is usually ruled out by concavity of the periodic utility function. Finally, it can be shown that differentiability of the periodic utility function guarantees that the value function is also differentiable under quite general conditions.

1.6 Numerical dynamic programming

Consider solving the problem in (21) numerically by value function iteration. Given a guess for the value function, we solve the optimization problem and then update the value function. That is, we have the following recursive structure:

$$V^{j+1}(k,z) = \max_{k',n}\{u(f(k,n)-k') + \beta E_{z'|z}V^j(k',z')\} \tag{24}$$

$$s.t. \tag{25}$$

$$c + k' = zf(k,n) + (1-\delta)k \tag{26}$$

Given V^j we solve for V^{j+1}, given V^{j+1} we solve for V^{j+2} and so on.

One obvious practical problem is that the value function is defined over a continuum of capital stocks. For instance, in order to solve for V^{j+1} for a given level of capital, we need to know the value of V^j *for all* possible values of k'. Hence it is seems that we must have solved V^j for uncountably many different values of k.

We overcome this problem by discretizing the state space and by interpolating the value function. Since we cannot solve the value function for all possible levels capital, we solve for it a finite set of different capital levels. That is, we discretize this state space along the k-dimension and consider only values $k^1 < k^2 < .... < k^N$. In other words, the value function that we compute is actually just a vector of $2N$ elements (again assuming that z can take only two values). The values k^1 and k^N should be chosen so that they are not restricting the optimal consumption decision.

When optimizing over k' (given k and V^j), we typically need to determine $V^j(k',z')$ (or its derivative) for some k' that do not correspong to any point in the grid (i.e.

$k' \neq k^i$ for all $i = 1, 2, , , .N$). This we do by interpolation. The simplest way is linear interpolation: $V(k) \approx V(k^n) + \frac{V(k^m) - V(k^n)}{k^m - k^n}(k - k^n)$ where $k^n < k < k^m$. There are also alternative, and much more efficient, interpolation schemes (such as interpolating with different splines or polynomials).

Value function iteration is the most straigthforward and often also the most robust way of solving dynamic problems. The problem is that it is often very time consuming. One way to speed up the computation is called Howard's improvement algorithm. The idea is very simple: instead of finding new optimal policies (here $k'(k, z)$ and $n(k, z)$) within each value function iteration, we update the value function most of the time with fixed policies. That is, once we have found optimal policies for a given value function, we use those policies to iterate the value function not just once but several times. Then we reoptimize and iterate again the value function several times. This process is repeated until both the optimal policies and the value function have converged. Typically, Howard's improvement algorithm speeds up the computation substantially. This is because it allows to greatly reduce the number of times that we need to optimize.

1.7 Recursive competive equilibrium

In the above examples, we illustrated dynamic programming by considering the neoclassical growth model and the problem of the social planner. Let us now reconsider the problem of a representative household and the competive equilibrium. For simplicity, we consider here the deterministic version of the model and abstract from the labor supply decision.

When writing down the problem of the representative household (as opposed to the problem of a social planner), we need to distinguish between individual capital holdings and the aggregate capital stock. This is because the representative household takes the aggregate capital stock as given. At the same time, the household needs to know what the aggregate capital stock is. There are two reasons for this. First, current aggregate capital stock determines current wage and interest rates. Second, current aggregate capital stock helps to predict next capital stock which in turn determines next period prices.

Let us denote individual capital holdings by k and aggregate capital stock by K. We write the problem of the representative household as follows:

$$\begin{aligned}
v(k,K) &= \max_{k'}\{u(c)+\beta v(k',K')\} \\
s.t. & \\
c+k' &= (1+r(K))k+w(K). \\
k' &\geq \underline{k} \\
K' &= H(K).
\end{aligned}$$

where $H(K)$ denotes the law of motion for the aggregate capital stock that the representative household has in mind. In other words, given current aggregate capital stock K, the household expects next period capital stock to be $H(K)$.

The solution of this household problem includes a policy function $k'=h(k,K)$ that determines household savings as a function of its own capital holdings and aggregate capital stock.

In lecture notes 1, the competitive equilibrium was defined directly in terms of the allocation and prices. The recursive competitive equilibrium, in contrast, consists of a set of functions that map quantities and prices to the current state. We define it as follows:

The recursive competitive equilibrium consists of a value function $v(k,K)$, policy function $h(k,K)$, aggregate law-of-motion $H(K)$ and prices $r(K)$ and $w(K)$ such that.

i) $v(k,K)$ and $h(k,K)$ solve the household problem.

ii) Prices are competitively determined:

$$\begin{aligned}
r(K) &= f_k(K)+1-\delta \\
w(K) &= f_n(K)
\end{aligned}$$

iii) Consistency is satisfied:

$$h(K,K)=H(K) \textit{ for all } K$$

The third condition means that whenever the representative household's capital holdings equal the aggregate capital stock, its own capital holdings evolve exactly as the aggregate capital stock. In other words, households expectations regarding next period aggregate capital stock are consistent with the actual behaviour of individual households.

Of course, also the standard aggregate resource constraint needs to hold in the equilibrium. However, as usual, the household budget constraint, together with the assumption of a constant returns to scale technology, already guarantees that the aggregate resource constraint holds.