Lecture 1: Introduction to Dynamic Programming

Xin Yi

January 5, 2019

1. Motivation

What is dynamic programming? Consider the following “Maximum Path Sum I” problem listed as problem 18 on website Project Euler. The task at hand is to find a path, which connects adjacent numbers from top to bottom of a triangle, with the largest sum. An example of such a path is

```
   3
  7 4
 2 4 6
8 5 9 3
```

The example above with 4 rows may be easy. But the following with 15 rows is not. Although one can crack the problem by brute force (16384 routes in total), the problem will be geometrically more complicated if the number of rows extends to hundreds or thousands.

```
                            75
                          95 64
                        17 47 82
                      18 35 87 10
                    20 04 82 47 65
                  19 01 23 75 03 34
                88 02 77 73 07 63 67
              99 65 04 28 06 16 70 92
            41 41 26 56 83 40 80 70 33
          41 48 72 33 47 32 37 16 94 29
        53 71 44 65 25 43 91 52 97 51 14
      70 11 33 28 77 73 17 78 39 68 17 57
    91 71 52 38 17 14 91 43 58 50 27 29 48
  63 66 04 68 89 53 67 30 73 16 69 87 40 31
04 62 98 27 23 09 70 98 73 93 38 53 60 04 23
```

So is there any intelligent solution to this problem? Instead of working out the solution from top to bottom, an alternative is to work out the solution from backwards. Starting from the second last row, say number 63, conditional on the path already reaches 63, there are only two choices left: 04 and 62. Given the past, history already doesn't matter. What matters is which number in the last row will give the largest payoff. Clearly, between 04 and 62, the choice should be 62. Thus the payoff of reaching 63 in the second last row is $63 + 62 = 125$. Hence, we can erase 63 from the row and replace it with 125.

Repeating this logic for every number in the second last row, we can update it to, from left to right, “125, 164, 102, 95, 112, 123, 165, 128, 166, 109, 122, 147, 100, 54”. The new row will be the expected sum of future payoffs for reaching corresponding position in the second last row. From this, we also know the direction of the path in the next period to achieve such future payoffs. Deleting the last row and treating the updated second last row as the terminal row, we can iterate this “algorithm” and find the corresponding path with the largest total sum. Instead of comparing among 16384 paths, we can finish the problem in 15+14+13+...+1 = 120 steps. This solution method is called *backward induction*.

This problem, although elementary and abstract, is central to our discussions of dynamic programming. It is inherently dynamic. Each row can be seen as one time period and the starting point is from the top. The numbers are the *payoffs* corresponding to different *states* in each period. Maximizing a simple total sum along the path is then equivalent to saying that the *discount factor* is 1. There are two *choices* in each period, either to choose the number on the left or to choose the number on the right. If you remember what you have learnt about Arrow-Debreu equilibrium in your PhD Microeconomics II class, the 16384 possible routes then correspond to 16384 possible *histories*.

The fact that a minimalist dynamic problem, with only 15 periods and 2 choices, has 16384 possible histories reveals the complex nature of any dynamic model. Any exhaustive algorithm that compares payoffs along each history from forwards is inefficient. In contrast, a simple backward induction is useful because although the set of histories is huge, the number of states is limited. Working from backwards, one can ignore the past payoffs or the past histories and instead focus on the one-period payoffs at hand. Iterating this logic, one can then easily solve for any dynamic problem in principle, even if the number of states is infinite. This approach is also useful in many areas. We can apply it to solve for macroeconomic models, structural labor models, or even microeconomic dynamic games.

Now I should introduce dynamic programming in more formal settings.

Recap: Dynamic problems are all about backward induction, as we usually do not have enough computing power to tackle the problem using an exhaustive search algorithm.[1]

Remark: In fact, backward induction is not the accurate phrase to characterize dynamic programming. A more precise description should be ”solve suboptimal problems first and store it, so that you don’t have to solve it again for multiple times”.

Exercise: How does backward induction in “Maximum Path Sum” problem relate to ”solve suboptimal problems first and store it, so as to avoid solving it for multiple times”?

[1]This is true, even one uses modern metaheuristic algorithms such as differential evolution or particle swarm optimization. More importantly, we are interested in the analytical properties most of the time, so that we can offer some principles on how to solve a “Maximum Path Sum” problem in general, not just numerically.

2. A First Look

The "Maximum Path Sum" problem introduced earlier is essentially a dynamic problem with finite number of states. There are many applications for this class of dynamic models. One classic example would be the structural occupational choice model in Keane and Wolpin (1997). However, the state space in the class of dynamic models that we are going to see in the Macroeconomic classes is usually infinite. More formally, consider the following canonical neoclassical growth model,

$$\max_{\{c_t,k_{t+1}\}_{t=0}^{T}} \quad \sum_{t=0}^{T}\beta^t U(c_t)$$

$$\begin{aligned} \text{s.t.} \quad & c_t + i_t \le y_t = f(k_t), && \forall t = 0,\dots,T; \\ & k_{t+1} = (1-\delta)k_t + i_t, && \forall t = 0,\dots,T-1; \\ & k_0 > 0 \quad \text{is given}. \end{aligned} \tag{1}$$

where k_t is the capital stock at time t, i_t is the amount of investment, c_t is the quantity of consumption, and the usual regularity conditions apply[2]. Readers with a quick eye should immediately see that, a naive backward induction is no longer feasible. Unlike the "Maximum Path Sum" problem, where the cardinality of state space is 10 in its maximum, in each period the cardinality of state space is infinite. It is almost impossible to compare all the possible values of $\{c_t, n_t, k_{t+1}\}$ even if one works from backwards.

So how should we tackle this class of problems? In what follows, I will first offer a method which utilizes Kuhn-Tucker conditions to solve the problem. Then I will introduce another method or, more precisely, another formulation called "Principle of Optimality" which essentially generalizes the intuition of backward induction for this class of problems. The final section of this lecture is then dedicated to "UTDs" (unnecessary technical details)[3] to establish that, formulations under "Principle of Optimality" is really equivalent to the formulation in (1).

3. Kuhn-Tucker Solution Methods

Finite horizon

I will skip most of the UTDs. Readers who demand a higher-level of mathematical rigour may follow the reasoning in chapter 2.1 of SLP (Stokey et al., 1989). After eliminating i_t, the

[2] $f(0) = 0, f'(k) > 0, \lim_{k\to 0} f'(k) = \infty, \lim_{k\to\infty} f'(k) = 1-\delta$

[3] This is a phrase borrowed from Prof. Steven Durlauf's Social Interaction class.

Lagrangian of the problem in section 2 is as follows,

$$\mathcal{L} = \sum_{t=0}^{T} \beta^t U(c_t) - \sum_{t=0}^{T} \lambda_t [c_t + k_{t+1} - (1-\delta)k_t - f(k_t)]$$

The relevant first-order conditions are

$$\beta^t U'(c_t) = \lambda_t$$

$$\lambda_t = [(1-\delta) + f'(k_{t+1})]\,\lambda_{t+1}$$

Note that how the second FOC is obtained by using both k_{t+1} in the term $\lambda_t[c_t + k_{t+1} - (1-\delta)k_t - f(k_t)]$ and the term $\lambda_{t+1}[c_{t+1} + k_{t+2} - (1-\delta)k_{t+1} - f(k_{t+1})]$. Furthermore, since we know that U is strictly increasing and $U'(0) = \infty$, λ_t must be greater than 0 for every t. Hence, by the complementary slackness condition $\lambda_t[c_t + k_{t+1} - (1-\delta)k_t - f(k_t)] = 0$, the inequality constraint $c_t + k_{t+1} - (1-\delta)k_t - f(k_t) \leq 0$ must be binding, i.e.,

$$c_t + k_{t+1} - (1-\delta)k_t - f(k_t) = 0$$

Substituting this equation back into the FOCs to eliminate c_t, we have that

$$\beta^t U'[f(k_t) + (1-\delta)k_t - k_{t+1}] = \lambda_t \tag{2}$$

$$\lambda_t = [(1-\delta) + f'(k_{t+1})]\lambda_{t+1} \tag{3}$$

Taking the inter-temporal ratio of (2), we have that

$$\beta \frac{\lambda_t}{\lambda_{t+1}} U'[f(k_{t+1}) + (1-\delta)k_{t+1} - k_{t+2}] = U'[f(k_t) + (1-\delta)k_t - k_{t+1}]$$

From (3), we have that $\lambda_t/\lambda_{t+1} = (1-\delta) + f'(k_{t+1})$. Substituting this back into the previous equation, gives the following

$$\beta\,[(1-\delta) + f'(k_{t+1})]\,U'[f(k_{t+1}) + (1-\delta)k_{t+1} - k_{t+2}] = U'[f(k_t) + (1-\delta)k_t - k_{t+1}], \quad \text{for} \quad t = 0, 1, \ldots, T-1$$

which is essentially equation (5) on page 11 of SLP. We also know that k_{T+1} must be zero, since there is no need to invest if there is no future. Combine these with the initial condition k_0, we have a system of T equations (not necessarily linear) with T unknowns. Solving this system of equations shall give us a path of $\{k_t\}_{t=1}^{T}$. We can then utilize this and obtain the corresponding paths of $\{c_t\}_{t=0}^{T}$ and $\{i_t\}_{t=0}^{T}$. To solve this system of second-order difference equations, readers are required to do Exercise 2.2 on page 12 of SLP. You can use Irigoyen et al. (2003) to check your answer.

Infinite horizon

With infinite horizon, everything is similar except for that the terminal condition $k_{T+1} = 0$ is now replaced by a transversality condition. We will come back to this later.

4. Principle of Optimality

The above example is essentially a *"sequence problem"* (SP), that is, given the objective and the constraints, we want to find a *sequence* of $\{k_t\}$ to maximize (or to find the supremum of) the objective. This is almost analogous to the forward exhaustive search algorithm that we used in the "Maximum Path Sum" problem. One may argue that we have turned the previous neoclassical dynamic model into a system of second-order difference equations, so unlike the "Maximum Path Sum" problem, it is not that hard to solve for the solutions. However, this approach is not optimal in general. As you can already see in the "Maximum Path Sum" problem, there is no way that you can write the problem in difference equations. In what follows, I will introduce a second approach called *"Principle of Optimality"* to tackle the issue. This will be the cornerstone in our study of dynamic programming.

Before we start, it is always good to hear from the man who invented "Principle of Optimality" himself, Richard Bellman.

> PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

The above definition is an intuitive quotation from page 83 of Bellman (1957). To understand this sentence, we should first understand what a "*state*" is.

State variables

To paraphrase Bellman, a state variable is essentially a small set of parameters or variables that *characterize* the system at any stage. We need these state variables because they provide all the necessary information we need to make a decision at any stage. For example, in the neoclassical growth model the relevant *choice variables* are $\{k_{t+1}\}$, the amount of capital stock we have in the next period.The relevant *state variables* are then $\{k_t\}$, because we need to know the situations we are in, we need to know how much capital stock we already have in order to choose what's optimal for the future. In the "Maximum Path Sum" example, the state variable is essentially the current node or number your are in, because this number tells you what your future payoffs are, whether you choose left or right, so as to inform you on which decisions are optimal conditional on you are already in that node.

Policy and contingency plans

After knowing what a "state" is, it comes down to interpret the sentence that "the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." To understand this sentence, we must first define what constitutes "an optimal policy". Broadly speaking, an optimal policy is a "contingency plan" which defines what you should do in every possible scenario or in any possible state. For example, if your breakfast choice depends on the weather, then a policy function or a contingency plan is a complete list of what you should eat tomorrow morning in every possible weather scenario. In our example, a policy is a function $k^*_{t+1}(k_t)$ that defines what amount of capital you should achieve in the next period given any possible state k_t which is the amount of capital you already have.

Sub-problems and sub-optimality

To finish interpreting the definition, we need to look at another slightly more modern definition of "Principles of Optimality", which is adapted from the definition in Bertsekas (2017) and modified for our purpose. Consider the canonical neoclassical growth model. Suppose that we are at state where $k_t = \tilde{k}$ for some particular $\tilde{k}$ at particular time t_0, then a subproblem is to maximize lifetime utility for the remaining life cycle starting from time t_0, conditional on $k_{t_0} = \tilde{k}$, i.e.,

$$
\begin{aligned}
&\max_{\{c_t, k_{t+1}\}_{t=t_0}^{T}} \quad \sum_{t=t_0}^{T} \beta^t U(c_t) \\
&\text{s.t.} \quad c_t + i_t \le y_t = f(k_t), \qquad \forall t = t_0, \dots, T; \\
&\qquad k_{t+1} = (1-\delta)k_t + i_t, \qquad \forall t = t_0, \dots, T-1; \\
&\qquad k_{t_0} = \tilde{k} \qquad \text{is given.}
\end{aligned} \tag{4}
$$

To paraphrase Bertsekas (2017), Principle of Optimality states that if $\{k^*_{t+1}\} = \{\tilde{k}_1, \tilde{k}_2, \dots, \tilde{k}_T\}$ is an optimal policy to the original problem, then the truncated policy $\{\tilde{k}_{t_0+1}, \dots, \tilde{k}_T\}$ is optimal for the subproblem (4). We will delegate the formal proof to the later section.

Intuition

An intuitive proof is that if the truncated policy is not optimal for the subproblem and there is an alternative optimal policy $\{\hat{k}_{t+1}\}_{t_0}^{T-1}$, then there must be an alternative policy $\{k_{t+1}\}$ where $k_{t+1} = \tilde{k}_{t+1}$ for $t = 0, \dots, t_0 - 1$ and $k_{t+1} = \hat{k}_{t+1}$ for $t = t_0, \dots, T-1$ such that this policy beats the optimal policy $\{k^*_{t+1}\}$. This would be a contradiction. To supply some real-life intuitions, the best example is found in Bertsekas (2017), which I quote here, "[S]uppose that the fastest route from Los Angeles to Boston passes through Chicago. The principle of optimality translates to the obvious fact that the Chicago to Boston portion of the route is also the fastest route for a trip that starts from Chicago and ends in Boston". The intuition of this example is that, ...

In essence, Principle of Optimality (henceforth PO) lays a foundation for dynamic programming, which is a more efficient way to solve dynamic optimization problem. The cornerstone of this approach is to formalize the "backward induction" intuition that we have seen earlier. If PO holds, then we can always start from the last period T and consider the associated subproblem. By PO, the optimal policy to this subproblem will also be part of the optimal policy to the global problem. As a byproduct, we will also know about payoffs associated to reaching each possible state in the terminal period. Hence, we can substitute these payoffs into the subproblem associated with period $T-1$ and solve for the optimal policy to this subproblem. Iterating this logic, we will eventually find the optimal policies to each subproblem for each time period $t = 2, \dots, T$. By PO, they are all part of the optimal policy to the global problem. Combining them returns us the optimal policy of the entire dynamic optimization problem.

How much more efficiency does this dynamic programming give us? In the context of problems with finite state space such as the "Maximum Path Sum" problem, dynamic programming only requires to solve the subproblems T times. Brute force approach that evaluates every possible 'path' would require compare payoffs of $T!$ different paths. In general, the computational burden for brute force approaches is exponential in T while dynamic programming approach is polynomial in T. That is, the computational time for brute force is $O(n^T)$ and that of dynamic programming is $O(T^n)$. We can see that dynamic programming is clearly superior than a brute force approach for larger values of T (for $n = 2$),

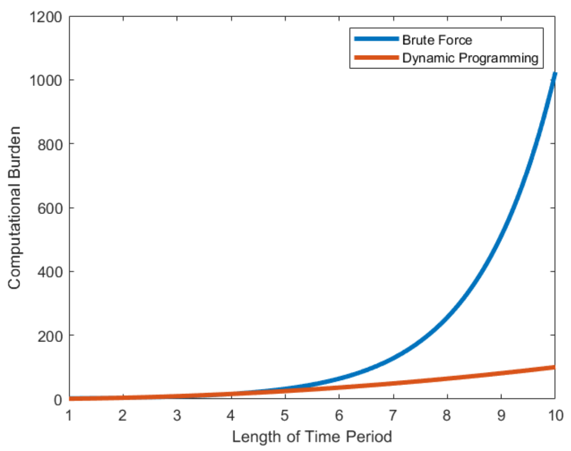

Figure 1: Computational burden of different approaches

5. Functional Equation Formulation

So how does this definition of Principle of Optimality relate to the one in SLP? To see the comparison clearly, I will first reproduce the definition of Principle of Optimality on page 67, with slight modifications to cater our notations. For any *sequence problem* (SP) such as problem (1), there is a *functional equation* representation (FE) such that,

$$v_t(k_t) = \max_{c_t, k_{t+1}} [U(c_t) + \beta v_{t+1}(k_{t+1})]$$

$$s.t. \quad c_t + k_{t+1} - (1-\delta)k_t \leq f(k_t)$$

where $v_t(\cdot)$ is the "value function" at time t and $v_t(k_t)$ is the value of the optimal program from period t at state k_t,[4]

$$v_t(k_t) \equiv \max_{\{c_s, k_{s+1}\}_{s=t}^{T}} \sum_{s=t}^{T} \beta^{s-t} U(k_s)$$

$$s.t. \quad c_s + k_{s+1} - (1-\delta)k_s \leq f(k_s), \quad \forall s \geq t$$

such that $v_t(\cdot)$ maps all the possible values of k_t to the corresponding maximized objective function starting from k_t. SLP's version of Principle of Optimality then states that a solution v attains the maximum in a SP problem if and only if it is an optimal solution to the FE problem in every t. We shall discuss the intuition of this statement first while delegating a more rigorous mathematical proof that operates in the metric space to later lectures. Two remarks are in order.

First, the definition of value function exactly maps to the PO definition in Bertsekas (2017). The version of PO in Bertsekas (2017) states that an optimal policy to a subproblem must also be part of the optimal policy to the global problem. This is exactly what $v_t(k_t)$ is, that is, it is the maximum value to the subproblem starting from k_t. Hence, the functional equation formulation says that, an optimal policy attains a maximum in the global problem if and only if it also attains a maximum for each subproblem.

Second, if the definition of $v_t(k_t)$ is exactly the optimal policy to a SP formulation of subproblem, then what is the difference between the FE formulation and the SP formulation? The answer is that the FE formulation is a *recursive* representation of the problem. That is, the value function $v(\cdot)$ will appear on both sides of the function equation. To see how this is a recursive representation, we can substitute the definition for v_{t+1} into the FE problem

[4] Note that in this example the value function has a time subscript, because the life cycle is finite and the environment is not stationary. If the problem is infinite and stationary, then we should drop the time subscript.

such that it becomes

$$v_t(k_t) = \max_{c_t, k_{t+1}} \left[U(c_t) + \max_{\{c_s, k_{s+1}\}_{s=t+1}^{T}} \sum_{s=t+1}^{T} \beta^{s-(t+1)} U(k_{s+1}) \right]$$

$$\begin{aligned} s.t. \quad & c_t + k_{t+1} - (1-\delta)k_t \leq f(k_t) \\ & c_s + k_{s+1} - (1-\delta)k_s \leq f(k_s), \quad \forall s \geq t+1. \end{aligned}$$

We can quickly prove that this is further equivalent to the following problem,

$$v_t(k_t) = \max_{c_s, k_{s+1}} \sum_{s=t}^{T} \beta^{s-t} U(k_s)$$

$$s.t. \quad c_s + k_{s+1} - (1-\delta)k_s \leq f(k_s), \quad \forall s \geq t,$$

which is not surprising since this is precisely the definition of the "value function" starting from k_t. Furthermore, this problem is also the SP formulation of the subproblem starting from k_t. To conclude, despite that this reasoning seems cyclic and tautological, it illustrates an important point, that the FE formulation is a *recursive representation* of the SP formulation for the same subproblem.

So why should we care about recursive formulation or a FE formulation when the SP formulation is available? One reason is that it is **simple** and often supplies useful economic intuitions. Another, perhaps a more important reason, is related to computational efficiency. Going back to the "Maximum Path Sum" problem, there is another key insight that maps exactly to the FE or the recursive formulation. When we do the backward induction, the payoffs of each state in the last period is computed first and then attached to the payoffs in the second last period. Then, we *iterate* this logic until the first period. By doing this, we are only considering one period forward in each step. That is, the payoffs in each step has been amended so as to include all future payoffs along the optimal path of the corresponding subproblems. This maps precisely to the definition of the value function. Furthermore, this one-period forward value function is constructed *recursively* by our iteration of the logic.

This is the reason that we prefer FE formulation to the SP formulation of the problem in the language of dynamic programming. By PO, we can incorporate the idea of backward induction both into SP and FE formulation. However, the FE formulation is more efficient than the SP formulation.[5] Because by solving the problem recursively, we have essentially "stored" the optimal policy to each subproblem because the payoffs are added up recur-

[5]While this statement is in principle true, in practice it is more complicated. When the discount factor approaches 1, value function iteration becomes very slow and the SP formulation or the Euler Methods become more appealing. There are also other more efficient methods for certain class of dynamic problems. However, the discussion of such methods should be delegated to a second-year numerical courses and is beyond the purpose of the current course. Interested readers may search for Prof. Violante's second-year course at NYU.

sively. This saves additional effort to compute the optimal policies again when solving a larger subproblem in the earlier time period.

Recap: Principle of Optimality allows us to do backward induction in dynamic optimization. The FE formulation of the problem then further saves computational efforts because by considering the problem recursively, the results from the subproblems are stored instead of being recomputed in every step.

6. Conclusion

In this lecture, we have briefly introduced and explained the key concepts in dynamic programming which is the basic mathematical language to analyze models in macroeconomics. As a language, it has two advantages. First, it is simple and provides economic intuition on the tradeoffs that agents facde in each period. Second, it is computationally efficient and incorporates both the ideas of backward induction and recursive storing of information. Principle of Optimality is introduced so that we can safely do backward induction in the optimization. The FE formulation then helps us to formulate the question recursively, so as to store the optimal policies already computed and to avoid unnecessary recomputations.

In the subsequent lectures, we will first go through the mathematics behind dynamic programming and formally prove SLP's version of Principle of Optimality. Then, we will use the FE or the recursive formulation of the problem to write down an equilibrium and discuss how to solve it. We will do this for both finite and infinite horizons, also with and without uncertainty. Finally, we will move on to search and matching models which supplies even more microfoundations to macroeconomic problems. Interested readers can directly skip the mathematical proof of Principle of Optimality in lecture 2, as it is mostly adapted from the SLP textbook and also in my view, not essential.

References

Bellman, Richard, *Dynamic Programming*, Princeton, New Jersey: Princeton University Press, 1957.

Bertsekas, Dimitri, *Dynamic Programming and Optimal Control*, 4 ed., Vol. 1, Nashua, New Hampshire: Athena Scientific, 2017.

Irigoyen, Claudio, Esteban Rossi-Hansberg, and Mark L. J. Wright, *Solutions Manual for Recursive Methods in Economic Dynamics*, Cambridge, Massachusetts: Harvard University Press, 2003.

Keane, Michael P. and Kenneth I. Wolpin, "The Career Decisions of Young Men," *Journal of Political Economy*, 1997, *105* (3), 473–522.

Stokey, Nancy L., Robert E. Jr. Lucas, and Edward C. Prescott, *Recursive Methods in Economic Dynamics*, Cambridge, Massachusetts: Harvard University Press, 1989.

Lecture 2: Mathematics of Dynamic Programming

Xin Yi

December 23, 2018

Disclaimer

This lecture is largely adapted from Chapter 3 of Stokey et al. (1989) with many parts being identical. I do not claim any credit to this notes and forbid redistribution beyond the purpose of the course. All errors are mine.[1]

1. Space

The first set of mathematical definitions we will study is associated with the concept of space. In general, a space is just a set of numbers with certain structures (properties) on the set. We are interested in the concept of space, because we want to operate on a restricted set of numbers where the properties/structures of these numbers help us to establish certain theorems that will be useful for our purposes. For example, a commonly used technique in dynamic optimization is called "Value Function Iteration" which iterates on the Bellman equation until a fixed point is found. However, to ensure that this iteration procedure actually converges, we will want to operate on a space called Banach space because the properties in that space helps us to establish an assurance for convergence. In what follows, we will introduce some basic definitions and gradually arrive at the definition of Banach space.

Definition 1. A **real vector space** X is a set of vectors with the following properties. For any two vectors $x, y \in X$, real numbers $\alpha, \beta \in \mathbf{R}$, and a zero vector $\theta \in X$,

a. $x + y \in X$.

b. $\alpha x \in X$.

c. $x + y = y + x$.

d. $(x + y) + z = x + (y + z)$.

e. $\alpha(x + y) = \alpha x + \alpha y$.

[1] Readers may want to skip this lecture as most of the material is already covered in Math Camp 1B under the section of "Banach Fixed Point Theorem". Nevertheless, I will try my best to supply as much intuition as possible to the mathematics, so that it may help the readers to refresh theor understandings.

f. $x + \theta = x$.
g. $0x = \theta$.
h. $1x = x$.

The first notion of space that we will encounter is the *real* vector space. This space operates over the set of vectors that satisfy a number of properties. First of all, this vector space satisfy the usual rules for addition (items a.). Second, the vector in this space can be scaled by a *real number* $\alpha \in \mathbf{R}$ (item b.). Furthermore, the addition and scaling operation must satisfy some normal behaviors (addition: items c. d. f.; scaling: items e. g. h.).

As the name suggests, we call this space a real vector space because the scalar we used for scaling is a real number. In contrast, a complex vector space is a space where the scalar is a complex number.

Definition 2. A **metric space** is a set S, together with a metric $\rho : S \times S \to \mathbf{R}$, such that for all $x, y, z \in S$,
a. $\rho(x, y) \geq 0$, with equality if and only if $x = y$.
b. $\rho(x, y) = \rho(y, x)$.
c. $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$.

In general, we can interpret a metric as a way of measuring distances between any two points. There can be any sorts of metric. One example could be that we have three points on the Cartesian coordinates, such that $(a, b, c) = \{(2, 5), (3, 2), (13, 7)\}$. In this example, the set S corresponds to all points in the Cartesian coordinates. However, one is free to define *any* sort of metric he wants. He can assign the distance between a and b to be 1, the distance between b and c to be 2, and so on. What this definition says is that, for the (S, ρ) pair to be a metric space, the metric we define has to satisfy some regularity conditions. These conditions are that, the distance must be nonnegative and only be zero if and only if the two points are identical; the distance must be symmetric; and if we stop somewhere in between the two points, it can at best be located along the shortest path such that there is no way we can find a shortcut to beat the direct path connecting the two points. If these regularity condition are satisfied for the distance measure (metric) ρ we defined, then the (S, ρ) pair will be a metric space.

Definition 3. A **normed vector space** is a vector space S, together with a **norm** $\|\cdot\| : S \to \mathbf{R}$, such that for all $x, y \in S$, $\alpha \in \mathbf{R}$, zero vector $\theta \in \mathbf{R}$,
a. $\|x\| \geq 0$, with equality if and only if $x = \theta$.
b. $\|\alpha x\| = |\alpha| \cdot \|x\|$.
c. $\|x + y\| \leq \|x\| + \|y\|$.

For the normed vector space, we can interpret the "norm" as a measure for the length of the vector. In general we can assign any length we want for a given vector. For example, we can just define the length of the vector $a = (1, 2)$ to be 100. What this definition says is that given a vector space S and a length measure $\|\cdot\|$, we will only call the $(S, \|\cdot\|)$ a normed vector space if the norm measure satisfies some regularity conditions. The conditions are that the length must be nonnegative (equals zero iff the vector itself is 0); the vector is "scalable" such that if it is multiplied by a constant, the length will also be scaled up by the constant without changing the direction of the vector; and that the sum of lengths of the two vectors must be no smaller than the length of a vector that is the addition of the two vectors (with equality iff the two vectors are heading the same direction).

One example of such a norm is the "ℓ_1 norm" where the length of the vector is defined as the sum of absolute values of the coordinates, $\|x\|_1 = \sum_i |x_i|$. Another example of a norm is the "ℓ_2 norm" where the length of the vector is defined as the the square root of sum of squares of the coordinates, $\|x\|_2 = \sqrt{\sum_i x_i^2}$. In terms of the language of econometrics, a variant of ℓ_1 norm can be interpreted as a "Absolute Value Deviation" criterion while a variant of ℓ_2 norm is a "Least Squares" criterion. The key difference between the two criterion is that a ℓ_2 loss function penalizes high-value elements more. (as it is quadratic). As for the definition of "variants", readers are invited to do the following exercise and connect the dots.

Exercise 1. Suppose that $(S, \|\cdot\|)$ with a norm $\|\cdot\| : S \to \mathbf{R}$ is a normed vector space and define a distance $\rho : S \times S \to \mathbf{R}$ with $\rho(a, b) = \|a - b\|$. Prove that (S, ρ) is a metric space.

Definition 4. A sequence $\{x_n\}_{n=0}^{\infty}$ **converges** to $x \in S$, if for each $\varepsilon > 0$, there exists N_ε such that $\rho(x_n, x) < \varepsilon, \forall n > N_\varepsilon$.

The interpretation is as follows. We have a sequence $\{x_n\}_{n=0}^{\infty}$ that we want to show it converges. What this definition essentially says is that, given a correct guess of the limit x, the sequence converges to this limit if it satisfies the ε-δ criterion. The criterion says, for any choice of small distance ε, there will be a "turning point" N_ε such that the numbers after this turning point will be sufficiently close to the prespecified limit x; and the distance between the numbers and the limit is within the prespecified distance ε. Since we can arbitrarily choose a small enough ε, this definition then depicts a picture that satisfies what we want for a definition of "convergence".

However, this definition is not perfect. The problem is that it requires a prespecified "guess" of what the limit x is. What if we have a sequence, we don't now what the limit is, and we still want to show that it converges? The following definition of the "Cauchy sequence" fills this gap.

Definition 5. A sequence $\{x_n\}_{n=0}^{\infty}$ in S is a **Cauchy sequence** (satisfies the **Cauchy criterion**) if for each $\varepsilon > 0$, there exists N_ε such that $\rho(x_n, x_m) < \varepsilon, \forall n, m \geq N_\varepsilon$.

As discussed previously, Cauchy sequence aims to provide a definition of convergence without knowing where the limit is. Unlike the previous definition, Cauchy sequence says that it is sufficient to demand that any two numbers after the turning point are sufficiently close to each other and the distance must be smaller than the prescribed (arbitrary) distance ε. In this sense, we don't have to know what the limit is. In fact, we are pining down the limit by the relative position of the numbers in the sequence. The previous definition works by setting a "target" (the limit) and tries to show that a convergence sequence will be staying infinitesimally near that target in the limit. What Cauchy says is that, we don't have to set up a target, in order to show that the sequence converges, all we need to do is to show that the numbers stay infinitesimally near to each other, so that they stay infinitesimally near to *something*.

Exercise 2. Show that a sequence converges if and only if it is a Cauchy sequence.

Definition 6. A metric space (S, ρ) is **complete** if every Cauchy sequence in S converges to an element in S. A complete normed vector space is called a **Banach Space**.

Definition 6 contains two definitions. The first definition is about completeness of a metric space. The motivation of this definition is that we want a notion that says the metric space is "dense" and has no "holes". For example, if we consider the set of positive numbers, $\mathbf{R}^+$, this set is not "dense" as we can add the zero number next to it. Hence, a natural characterization of such space is that every sequence of numbers in this set must converge to a point that belongs to the same set. You may check that over the realm of positive numbers, the sequence $y_n = \frac{1}{n}$ converges to zero which is not in the set. Hence, $\mathbf{R}^+$ is not complete. In fact, if we think about the number line that represents positive numbers $(0, \infty)$, there is literally a "hole" on the position of 0.

The second definition is about Banach Space. It is just defined as a normed vector space where every Cauchy sequence in it converges to a point that is also in the space. Banach Space is important for our purposes, because later when we prove the Contraction Mapping Theorem, we can show that the numbers mapped by the contraction form a Cauchy sequence and hence they will converge to a point inside the same space, which is the fixed point for the contraction mapping.

2. The Contraction Mapping Theorem

Theorem 1. Let $X \subseteq \mathbf{R}^l$, and let $C(X)$ be the set of bounded continuous functions $f : X \to \mathbf{R}$ with the sup norm, $\|f\| = \sup_{x \in X} |f(x)|$. Then $C(X)$ is a complete normed vector space.

Before we begin to prove it, perhaps it's better to first discuss why we need this theorem. Previously we claimed that completeness in a normed vector space is important because we want the mapping based on the Bellman equation to be a Cauchy sequence and we want it to converge to a fixed point. However, what Bellman equation does is to map a value function to another value function. That is, the mapping operates over the domain of *functions* instead of set of numbers. Hence, in order for us to apply the properties of a Banach space, we must first prove that the set of bounded continuous function is a Banach space.

Proof. We can show that $C(X)$ is a normed vector space. It suffices to show that this normed vector space is complete, that is, to show that if $\{f_n\}$ is a Cauchy sequence, then there exists $f \in C(X)$ such that for any $\varepsilon > 0$ there exists N_ε such that $\|f_n - f\| \leq \varepsilon, \forall n \geq N_\varepsilon$. SLP then proceeds to prove this claim in three steps.

Step 1. Find a "candidate" function f. Fix $x \in X$, then the sequence of real numbers $\{f_n(x)\}$ satisfies $|f_n(x) - f_m(x)| \leq \sup_{y \in X} |f_n(y) - f_m(y)| = \|f_n - f_m\|$, where the first inequality is obvious and the second equality follows from the sup norm definition. Obviously, this sequence satisfies the Cauchy criterion. By the completeness of the real numbers, it converges to a limit point, we can denote it by $f(x)$. This will be our candidate function f.

Step 2. Show that $\{f_n\}$ converges to f in the sup norm. Hence, we need to show that $\|f_n - f\| \to 0$ as $n \to \infty$. Let $\varepsilon > 0$ be given, we can choose N_ε so that $n, m \geq N_\varepsilon$ implies $\|f_n - f_m\| \leq \varepsilon/2$. This can be done, because $\{f_n\}$ satisfies the Cauchy criterion. Then for any fixed $x \in X$ and all $m \geq n \geq N_\varepsilon$,

$$\begin{aligned}
|f_n(x) - f(x)| &\leq |f_n(x) - f_m(x)| + |f_m(x) - f(x)| \\
&\leq \|f_n - f_m\| + |f_m(x) - f(x)| \\
&\leq \varepsilon/2 + |f_m(x) - f(x)|
\end{aligned}$$

Since $\{f_m(x)\}$ converges to $f(x)$, we can choose m separately for each fixed $x \in X$ so that $|f_m(x) - f(x)| \leq \varepsilon/2$. Since the choice of x was arbitrary, it follows that $\|f_n - f\| \leq \varepsilon, \forall n \geq N_\varepsilon$. Since $\varepsilon > 0$ was arbitrary, the convergence claim follows.

Step 3. Show that f is bounded and continuous. Boundedness is obvious. The definition of continuity requires that for every $\varepsilon > 0$ and $x \in X$, there exists $\delta > 0$ such that $|f(x) - f(y)| < \varepsilon$ if $\|x - y\|_E < \delta$, where $\|\cdot\|_E$ denotes the Euclidean norm on $\mathbf{R}^l$. Let ε and x be given. Choose k so that $\|f - f_k\| < \varepsilon/3$. This can be done because $f_n \to f$ in the sup norm.

Then we can choose δ such that $\|x-y\|_E < \delta$ implies $|f_k(x) - f_k(y)| < \varepsilon/3$. This can be done, because f_k is continuous. Then

$$\begin{aligned} |f(x) - f(y)| &\leq |f(x) - f_k(x)| + |f_k(x) - f_k(y)| + |f_k(y) - f(y)| \\ &\leq 2\,\|f - f_k\| + |f_k(x) - f_k(y)| \\ &< \varepsilon. \end{aligned}$$

□

Definition 7. Let (S, ρ) be a metric space and $T : S \to S$ be a function mapping S into itself. T is a **contraction mapping** (with **modulus** β) if for some $\beta \in (0, 1)$, $\rho(Tx, Ty) \leq \beta\rho(x, y)$, for all $x, y \in S$.

Recall that our strategy is to show that the Bellman operator over the Banach space maps to a Cauchy sequence and hence it converges to a fixed point. To establish this argument, we also need a concept called contraction mapping. Literally, contraction mapping means that it is a "contraction". The definition suggests that the distance ρ between the set elements (x and y) after the mapping (Tx and Ty) would "shrink" or "contract" ($\rho(Tx, Ty) \leq \beta\rho(x, y)$) since $\beta < 1$. This is perhaps best illustrated in the following figure which is Figure 3.1 in Hunter and Nachtergaelem (2001).

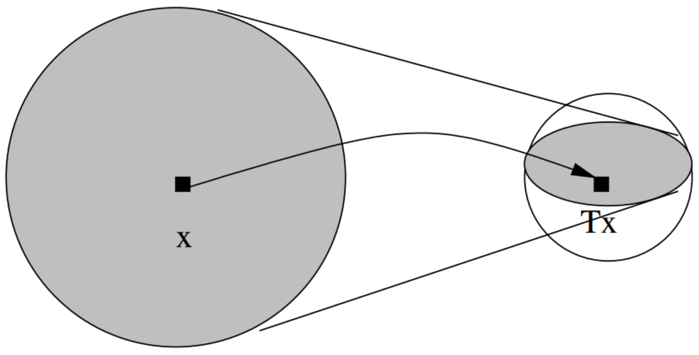

Fig. 3.1 T is a contraction.

Before we talk about a sufficient condition characterizing the set of mappings that are contractions, it is perhaps best to state and explain the Contraction Mapping Theorem first. The reason is that this theorem brings more intuition on the concept of contraction and why we need them.

Theorem 2. (*Contraction Mapping Theorem*) If (S, ρ) is a complete metric space and $T : S \to S$ is a contraction mapping with modulus β, then

a. T has exactly one fixed point v in S.

b. for any $v_0 \in S, \rho(T^n v_0, v) \leq \beta^n \rho(v_0, v), n = 0, 1, 2, \dots$

The Contraction Mapping Theorem (CMT) has two parts. The first part claims that if we iterate on the contraction mapping operator, then we will reach an *unique* fixed point as long as we are in a complete metric space. This is in fact very intuitive given our explanation on the contraction mapping that it shrinks distance between points. Imagine ourselves starting from a point, v_0, in a Banach space. Iterate the contraction mapping operator T over this point we will get a sequence $\{v_n\}_{n=1}^{\infty}$ where $v_n = T^n(v_0)$. Because a contraction maps the points in a space into another point in the same space, we know that v_n must also belong to the same Banach space. Furthermore, we know that the distance between any two points over the mapping process is getting smaller and smaller by the property of a contraction. If we take n to the limit, then it must be that v_{n+1} is arbitrarily close to v_n and we know that $\lim_{n\infty} v_n$ must converge to a point in the same space by the property of the Banach space. Thus, this limit will be a fixed point of the contraction mapping.

The above explanation, while intuitive, is merely a verbal representation of the mathematical proof in Stokey et al. (1989) which we reproduce in below. Another, perhaps the best, analogy that explains the intuition of the CMT proceeds as follows. Suppose we are holding a map of Singapore and we lay it on the ground of a seminar room at SMU. Then there must be a point in the map that is lying exactly on top of the point in Singapore it represents. If we continue to do the "mapping of maps" and lay the new map on top of the old map, this is still the case. Iterate over this mapping process infinitely, we will reach a fixed point.

Proof. To prove (a), we need to find a unique v such that $Tv = v$. Define the mappings $\{T^n\}$ such that $T^0 x = x$ and $T^n x = T(T^{n-1}x)$, for $n = 1, 2, \dots$. Choose $v_0 \in S$ and define $\{v_n\}_{n=0}^{\infty}$ by $v_{n+1} = Tv_n$. It follows that $v_n = T^n v_0$. Since by assumption T is a contraction mapping with modulus β, $\rho(v_2, v_1) = \rho(Tv_1, Tv_0) \leq \beta\rho(v_1, v_0)$. By induction, $\rho(v_{n+1}, v_n) \leq \beta^n \rho(v_1, v_0)$, for $n = 1, 2, \dots$. Hence, for any $m > n$,

$$\begin{aligned}
\rho(v_m, v_n) &\leq \rho(v_m, v_{m-1}) + \cdots + \rho(v_{n+2}, v_{n+1}) + \rho(v_{n+1,v_n}) \\
&\leq [\beta^{n-1} + \cdots + \beta^{n+1} + \beta^n]\rho(v_1, v_0) \\
&= \beta^n[\beta^{m-n-1} + \cdots + \beta + 1]\rho(v_1, v_0) \\
&\leq \frac{\beta^n}{1-\beta}\rho(v_1, v_0),
\end{aligned}$$

where the first inequality follows from triangle inequality and the second inequality follows from the previous induction that $\rho(v_{n+1}, v_n) \leq \beta^n \rho(v_1, v_0)$. It follows that $\{v_n\}$ is a

Cauchy sequence. Since S is complete, it follows that $v_n \to v \in S$ for some v. This is our candidate v. We now show that $Tv = v$ and that v is unique. First, note that for all n and $v_0 \in S$,

$$\begin{aligned}\rho(Tv, v) &\leq \rho(Tv, T^n v_0) + \rho(T^n v_0, v)\\ &\leq \beta\rho(v, T^{n-1}v_0) + \rho(T^n v_0, v).\end{aligned}$$

Notice that since $\lim_{n\to\infty} \rho(v_m, v_n) \leq \lim_{n\infty} \frac{\beta^n}{1-\beta}\rho(v_1, v_0) = 0$, it follows that $\rho(Tv, v) = 0$ and $Tv = v$.

Second, suppose there exists another $\hat{v} \neq v$ such that $T\hat{v} = \hat{v}$. Then,

$$0 < a = \rho(\hat{v}, v) = \rho(T\hat{v}, Tv) \leq \beta\rho(\hat{v}, v) = \beta a.$$

This is a contradiction since $\beta < 1$.

To prove part (b), observe that for any $n \geq 1$,

$$\rho(T^n v_0, v) = \rho[T(T^{n-1}v_0), Tv] \leq \beta\rho(T^{n-1}v_0, v).$$

(b) then follows by induction. □

Corollary 1. Let (S, ρ) be a complete metric space, and let $T : S \to S$ be a contraction mapping with fixed point $v \in S$. If S' is a closed subset of S and $T(S') \subseteq S'$, then $v \in S'$. If in addition $T(S') \subseteq S'$, then $v \in S'$. If in addition $T(S') \subseteq S'' \subseteq S'$, then $v \in S''$.

Corollary 2. (*N-Stage Contraction Theorem*) Let (S, ρ) be a complete metric space, let $T : S \to S$, and suppose that for some integer N, $T^N : S \to S$ is a contraction mapping with modulus β. Then

a. T has exactly one fixed point in S.

b. for any $v_0 \in S$, $\rho(T^{kN}v_0, v) \leq \beta^k \rho(v_0, v), k = 0, 1, 2, \ldots$.

Theorem 3. (*Blackwell's sufficient conditions for a contraction*) Let $X \subseteq \mathbf{R}^l$, and let $B(X)$ be a space of bounded functions $f : X \to \mathbf{R}$, with the sup norm. Let $T : B(X) \to B(X)$ be an operator satisfying

a. (monotonicity) $f, g \in B(X)$ and $f(x) \leq g(x)$, for all $x \in X$, implies $(Tf)(x) \leq (Tg)(x)$, for all $x \in X$.

b. (discounting) there exists some $\beta \in (0, 1)$ such that

$$[T(f + a)](x) \leq (Tf)(x) + \beta a, \text{ all } f \in B(X), a \geq 0, x \in X.$$

[Here $(f+a)(x)$ is the function defined by $(f+a)(x) = f(x) + a$.] Then T is a contraction with modulus β.

3. Principle of Optimality

Suppose we have the following [more general] sequence problem (SP),

$$\sup_{\{x_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, x_{t+1})$$

$$\text{s.t.} \quad x_{t+1} \in \Gamma(x_t), \quad t = 0, 1, 2, \ldots,$$

and $x_0 \in X$ is given. Then Principle of Optimality claims that a sequence $\{x_{t+1}\}_{t=0}^{\infty}$ attains the supremum of SP if and only if it satisfies the functional equation (FE) representation,

$$v(x) = \sup_{y \in \Gamma(x)} \left[F(x, y) + \beta v(y) \right].$$

Let us look into the details of the general problem. x_{t+1} corresponds to the choice variable. $F(x_t, x_{t+1})$ corresponds to the utility function. It has two arguments in it, because it depends on both the state variable x_t in the current period and also the [dynamic] choice of x in the next period. x_t is the state variable because it summarizes all the information relevant regarding the state we are in in period t. The $\Gamma(x_t)$ notation then corresponds to the set of possible values of the choice variables (which will also be the state variables in the next period) given that the current state variable is x_t; we can interpreted it as a feasibility constraint. In particular, for our neoclassical models, x_{t+1} corresponds to the choice of capital in the next period, k_{t+1}. $F(x_t, x_{t+1})$ then corresponds to the utility function, only with consumption being substituted by the binding constraint, $c_t + k_{t+1} = f(k_t) + (1-\delta)k_t$, such that

$$U(c_t) = U\left[f(k_t) + (1-\delta) - k_{t+1}\right] \equiv F(x_t, x_{t+1}).$$

The feasibility constraint $\Gamma(x_t)$ then corresponds to the range of possible k_{t+1} in the next period which is $[(1-\delta)k_t, f(k_t) + (1-\delta)k_t]$ in this problem. We will now state a set of assumptions under which Principle of Optimality holds.

Definition 8. A **plan** is a sequence $\{x_t\}_{t=0}^{\infty}$ in X. Given $x_0 \in X$, the set of plans that are **feasible** from x_0 is then defined as

$$\Pi(x_0) = \left\{ \{x_t\}_{t=0}^{\infty} : x_{t+1} \in \Gamma(x_t), \quad t = 0, 1, \ldots \right\}.$$

Assumption 1. $\Gamma(x)$ is nonempty, for all $x \in X$.

This assumption is rather trivial and standard. It implies that the set of feasible plans are non-empty.

Assumption 2. For all $x_0 \in X$ and $\underline{x} \in \Pi(x_0)$, $\lim_{T\to\infty} \sum_{t=0}^{T} \beta^t F(x_t, x_{t+1})$ exists.

This assumption says that we can evaluate the utility for all kinds of feasible plans. There are several sufficient conditions that ensure Assumption 2 hold. Readers are required to read page 69-70 on Stokey et al. (1989). The basic message is that the usual functional forms we see in macroeconomics satisfy these sufficient conditions and hence Assumptions 1 and 2 will hold. We are now ready to prove the Principle of Optimality by showing that SP implies FE and FE implies SP.

Definition 9. The **supreme function** $v^* : X \to \bar{\mathbf{R}}$ is defined as

$$v^*(x_0) = \sup_{\underline{\mathbf{x}} \in \Pi(x_0)} u(\underline{\mathbf{x}})$$

where $u : \Pi(x_0) \to \bar{\mathbf{R}}$ is

$$u(\underline{\mathbf{x}}) = \lim_{T\to\infty} \sum_{t=0}^{T} \beta^t F(x_t, x_{t+1}).$$

The interpretation of this definition is just that $v^*(x_0)$ is the supreme of the SP function.

Theorem 4 (*SP implies FE*). Let X, Γ, F, and β satisfy Assumption 1 and 2. Let $\underline{\mathbf{x}}^* \in \Pi(x_0)$ be a feasible plan that attains the supremum in SP for initial state x_0. Then

$$v^*(x_t^*) = F(x_t^*, x_{t+1}^*) + \beta v^*(x_{t+1}^*), \quad t = 0, 1, 2, \ldots$$

Theorem 5 (*FE implies SP*). X, Γ, F, and β satisfy Assumption 1 and 2. Let $\underline{\mathbf{x}}^* \in \Pi(x_0)$ be a feasible plan that satisfies

$$v^*(x_t^*) = F(x_t^*, x_{t+1}^*) + \beta v^*(x_{t+1}^*), \quad t = 0, 1, 2, \ldots$$

and with

$$\limsup_{t\to\infty} \beta^t v^*(x_t^*) \leq 0,$$

then $\underline{\mathbf{x}}^*$ attains the supremum in SP for initial state x_0.

Since the proofs for these theorems including the ones following are mechanical and follow similar arguments to what I have covered previously, I list them as take-home exercises.

4. SP: Euler Equation and Transversality

Exercise 3. Show that the Euler equation for SP is

$$0 = F_y(x_t^*, x_{t+1}^*) + \beta F_x(x_{t+1}^*, x_{t+2}^*), \quad t = 0, 1, 2, \dots$$

Exercise 4. Prove the sufficiency of the Euler conditions and transversality condition

$$\lim_{T\to\infty} \beta^t F_x(x_t^*, x_{t+1}^*) x_t^* = 0$$

as being the optimal solution for the SP problem.

5. FE: Bellman Equation and Functional Euler Equation

Deterministic Environment

Exercise 5. Show that the functional Euler equation for FE is

$$0 = F_y[x, g(x)] + \beta F_x[x, g(x)].$$

Exercise 6. Show that the Bellman equation for the FE under deterministic environment is

$$v(x) = \max_{y\in\Gamma(x)} [F(x, y) + \beta v(y)].$$

Stochastic Environment

Exercise 7. Show that the Bellman equation for the FE problem under stochastic environment with transition function $Q(z, z')$ is as follows,

$$v(x, z) = \sup_{y\in\Gamma(x,z)} \left\{ F(x, y, z) + \beta \int_z v(y, z') Q(z, dz') \right\}$$

Interested readers should read Chapter 8 of Stokey et al. (1989) on the transition functions. It will be beneficial for you when studying the materials covered in Macroeconomics 2. However, I will not cover them in details here as this should have been the content of the other math camp on Macroeconomics 2. The computational aspect of this should also be covered by the lecture on numerical methods in Macroeconomics 2.

References

Hunter, John K. and Bruno Nachtergaelem, *Applied Analysis*, WSPC; Reprint edition, 2001.

Stokey, Nancy L., Robert E. Jr. Lucas, and Edward C. Prescott, *Recursive Methods in Economic Dynamics*, Cambridge, Massachusetts: Harvard University Press, 1989.

Lecture 3: Dynamic Optimization Under Certainty

Xin Yi

1. Introduction

In this lecture, we will study deterministic dynamic optimization in greater details. Although the absence of uncertainty may be unrealistic, it serves as an useful benchmark because a stochastic equilibrium can be interpreted as a deterministic one. We will talk about two approaches to solve dynamic optimization problems. The first approach will be sequential optimization while the second approach uses dynamic programming. We will discuss each method for both finite and infinite horizon. Note that the context of these problems is from an agent's perspective, as it should be. Later in the Macroeconomics course, you are required to define and characterize *general equilibrium*. Second, readers need to be clear on the nature of the optimization. In some context, it is a planning problem. In other contexts, prices are involved and it is a partial equilibrium. Third, for the sequential methods, we will derive a system of equations that precisely characterize the equilibrium. Solving this system of equations using a computer will be straightforward. However, for dynamic programming, we are interested in optimal policies where close form solutions are usually unavailable. We will try to introduce cases with analytical solutions and also some [but not exhaustive] algorithms that explain how to numerically solve for the policy.

2. Sequential Optimization: Finite Horizon

Consider the optimization problem presented in lecture 1,

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{T}} \quad \sum_{t=0}^{T} \beta^t U(c_t)$$

$$\begin{aligned}
\text{s.t.} \quad & c_t + i_t \leq y_t = f(k_t), && \forall t = 0, \dots, T; \qquad (1)\\
& k_{t+1} = (1-\delta)k_t + i_t \geq 0, && \forall t = 0, \dots, T;\\
& c_t \geq 0, && \forall t = 0, \dots, T;\\
& k_0 > 0 \quad \text{is given.}
\end{aligned}$$

The Lagrangian is then

$$\mathcal{L} = \sum_{t=0}^{T} \beta^t U(c_t) - \sum_{t=0}^{T} \lambda_t[c_t + k_{t+1} - (1-\delta)k_t - f(k_t)] + \mu_t c_t + \nu_t k_{t+1}$$

The corresponding Kuhn-Tucker conditions consist of the First-Order Conditions (FOC),

$$\frac{\partial \mathcal{L}}{\partial c_t} : \beta^t U'(c_t) - \lambda_t + \mu_t = 0$$

$$\frac{\partial \mathcal{L}}{\partial k_{t+1}} : -\lambda_t + \lambda_{t+1}(1-\delta) + \lambda_{t+1} f'(k_{t+1}) + \nu_t = 0$$

the Complementary-Slackness Conditions (CS),

$$\lambda_t[c_t + k_{t+1} - (1-\delta)k_t - f(k_t)] = 0$$

$$\mu_t c_t = 0$$

$$\nu_t k_{t+1} = 0$$

and the Non-Negativity constraints (NN),

$$\lambda_t \geq 0, \mu_t \geq 0, \nu_t \geq 0.$$

Since it cannot be that $c_t = 0$ for any t, then by CS, $\mu_t = 0$ for all t. In addition, since $U'(c) > 0 \; \forall c$, it must be that $\lambda_t > 0$ by the FOC with respect to c_t. Hence, by CS, $c_t = f(k_t) - k_{t+1} + (1-\delta)k_t$. Substituting this expression for c_t and $\mu_t = 0$ back to the FOC yields,

$$\beta^t U' \left[f(k_t) - k_{t+1} + (1-\delta)k_t \right] = \lambda_t.$$

Manipulating this equation gives an inter-temporal expression

$$\frac{\beta U'[f(k_{t+1} - k_{t+2} + (1-\delta)k_{t+1})]}{U'[f(k_t) - k_{t+1} + (1-\delta)k_t]} = \frac{\lambda_{t+1}}{\lambda_t}.$$

Furthermore, it is easy to show that $k_{t+1} > 0$ for all t except for T. It follows that $\mu_t = 0$ for all $t = 0, \ldots, T$ and $k_{T+1} = 0$. Hence, from the FOC with respect to k_{t+1}, we can obtain,

$$\frac{1}{f'(k_{t+1}) + (1-\delta)} = \frac{\lambda_{t+1}}{\lambda_t}.$$

Combining these expressions then yields

$$U'[f(k_t) + (1-\delta)k_t - k_{t+1}] = \beta U'[f(k_{t+1}) + (1-\delta)k_{t+1} - k_{t+2}][(1-\delta) + f'(k_{t+1})].$$

This is the *Euler equation*. Given this equation for each period $t = 0, \ldots, T$, we have a system of second-order difference equations with $T + 1$ unknowns. Since k_0 is given and $k_{T+1} = 0$, the above system is exactly identified. To interpret this equation, note that it is equivalent to

$$\underbrace{U'(c_t)}_{\text{marignal cost}} = \underbrace{\beta U'(c_{t+1})}_{\text{marginal benefit}} \underbrace{[(1-\delta) + f'(k_{t+1})]}_{\text{investment return}}$$

$U'(c_t)$ is essentially the marginal cost of investing, since by investing i_t amount of capital for the future, the agent forgoes some consumption in the present. The marginal utility is then the utility that would be lost if the agent invested one more unit of capital. $\beta U'(c_{t+1})$ is the discounted marginal benefit of investing, since by investing i_t amount the agent is able to consume more goods in the future. The marginal utility is the [discounted] utility that would be enjoyed if the agent increases his consumption by one more unit in the next period. $(1-\delta) + f'(k_{t+1})$ then summarizes the state of investing technology. It governs how much consumption will increase in the next period, by investing one more unit of capital in the present.

Therefore, this Euler equation essentially describes the *intertemporal* relationship between the future and the present for *optimal* decisions. It is intertemporal because it connects future with the present through the time subscript, the discounted utility, and the intertemporal transformation of input to output (capital to consumption). It also describes the optimal decisions because this expression is equating marginal benefits with marginal costs. Furthermore, since the utility function $u(\cdot)$ is strictly concave, agents also prefer "consumption smoothing".

Recap 1: For sequential optimization with finite horizon, the optimal choice of capital $\{k_t\}$ is solved from the system of Euler equations with boundary conditions k_0 given and $k_{T+1} = 0$.

3. Recursive Formulation: Finite Horizon

Characterization of Optimal Policies

We now consider solving the problem using dynamic programming or to formulate the problem *recursively*. To formulate this problem recursively, first we define the value of the optimal program from period t at state k_t as

$$v_t(k_t) \equiv \max_{\{c_s, k_{s+1}\}_{s=t}^{T}} \sum_{s=t}^{T} \beta^{s-t} U(k_s)$$

$$s.t. \quad c_s + k_{s+1} - (1-\delta)k_s \leq f(k_s), \quad \forall s \geq t$$

As discussed in lecture 1, this can be further broke down to a “maximization-by-step” formulation,

$$v_t(k_t) = \max_{c_t, k_{t+1}} \left[U(c_t) + \max_{\{c_s, k_{s+1}\}_{s=t+1}^T} \sum_{s=t+1}^{T} \beta^{s-(t+1)} U(k_{s+1}) \right]$$

$$\begin{aligned} s.t. \quad & c_t + k_{t+1} - (1-\delta)k_t \le f(k_t) \\ & c_s + k_{s+1} - (1-\delta)k_s \le f(k_s), \quad \forall s \ge t+1. \end{aligned}$$

By definition of $v_t(k_t)$, this is equivalent to

$$v_t(k_t) = \max_{c_t, k_{t+1}} [U(c_t) + \beta v_{t+1}(k_{t+1})]$$

$$s.t. \quad c_t + k_{t+1} - (1-\delta)k_t \le f(k_t)$$

This is called the “Bellman Equation”. It is also a functional equation because the optimal policy, which is the “variable” of interest, is a function. Suppose that function $g_t(\cdot)$ denotes the optimal policy at time t. Then given a state k_t, $g_t(\cdot)$ will map k_t to a choice of capital in the next period, k_{t+1}. Now it is useful to stop and make comparisons between the recursive formulation and the sequence optimization method. In sequential optimization methods, we are only solving for the optimal path of capital. However, with recursive formulation, we are solving a contingency plan for every possible state of affair. That is, even if the current state is *not* along the global optimal path, the policy function $g_t(\cdot)$ still solves an optimal path for the subproblem. This is the key distinction between sequential optimization method and the recursive method.[1]

We now proceed to characterize the optimal policy $g_t(\cdot)$. First, the functional equation can be further written as

$$v_t(k_t) = \max_{k_{t+1}} \{U[f(k_t) - k_{t+1} + (1-\delta)k_t] + \beta v_{t+1}(k_{t+1})\}$$

The first-order condition to this problem is then

$$-U'[f(k_t) - k^*_{t+1}(k_t) + (1-\delta)k_t] + \beta v'_{t+1}(k^*_{t+1}(k_t)) = 0$$

We can denote the optimal policy as $g_t(\cdot)$ such that the optimal choice at state k_t is $k^*_{t+1}(k_t) =$

[1]Readers may wonder that from this perspective, sequential method seems to be computationally more efficient than the recursive method, since the sequential method only requires solving for the optimal path characterized by the second-order difference equations while recursive method does much more and solve for every subproblem. While it is true for this model, however in practice, researcher often, if not always, do not encounter such nice characterizations of the optimal path. Then the power of dynamic programming kicks in and this is why it is dominant in economic research.

$g_t(k_t)$. Then the FOC evaluates to,

$$-U'[f(k_t)-g_t(k_t)+(1-\delta)k_t]+\beta v'_{t+1}(g_t(k_t))=0$$

Note that this equation already supplies some intuition about the intertemporal tradeoff of the problem. $U'(\cdot)$ is the marginal cost of forgoing consumption in the present while $v'_{t+1}(\cdot)$ is the marginal benefit of increasing consumption in all future. To derive the complete functional Euler equation, note that after substituting the optimal policy for k_{t+1}, the Bellman equation can be written as

$$v_t(k_t)=U[f(k_t)-g_t(k_t)+(1-\delta)k_t]+\beta v_{t+1}(g_t(k_t))$$

Taking derivative with respect to k_t, we can obtain

$$v'_t(k_t)=U'[f(k_t)-g_t(k_t)+(1-\delta)k_t][f'(k_t)-g'_t(k_t)+(1-\delta)]+\beta v'_{t+1}(g_t(k_t))g'_t(k_t).$$

which is further equivalent to

$$v'_t(k_t)=\underbrace{U'[f(k_t)-g_t(k_t)+(1-\delta)k_t][f'(k_t)+(1-\delta)]}_{\text{first-order effect of } k_t}+\underbrace{g'_t(k_t)[\beta v'_{t+1}(g_t(k_t))-U'(\cdot)]}_{\text{indirect effect of } k^*_{t+1}(k_t)=g_t(k_t)}$$

This expression can be interpreted as follows. $v'(k_t)$ is the changes in the value from now and all future. It can be decomposed into two components. The first component is the direct, first-order effect of changes in k_t. It is essentially the change in utility due to the changes in current consumption. This effect is further magnified by the changes in the marginal amount of capital in the next period due to the changes in k_t. The second component is then the indirect effect through the optimal adjustment of $k^*_{t+1}(k_t)$ in response to changes in k_t. Substituting the FOC that $U'[f(k_t)-g_t(k_t)+(1-\delta)k_t]=\beta v'_{t+1}(g_t(k_t))$ into the this expression, we can eliminate the indirect effect and obtain

$$v'_t(k_t)=U'[f(k_t)-g_t(k_t)+(1-\delta)k_t][f'(k_t)+(1-\delta)].$$

The reason that the indirect term disappears is a consequence of the Envelope Theorem. The indirect effect of k_t through k_{t+1} will change two things, the value function in the future and the current utility. However, since the agent chooses $k^*_{t+1}(k_t)$ to optimize intertemporal tradeoff, it is necessary that the agent will balance the marginal cost of current utility and the marginal benefit of future payoffs which is also characterized by the FOC. Hence, the indirect term will be canceled out.

To make a more transparent comparison between the functional Euler equation under recursive method and the Euler equation under sequential method, note that we can update the previous expression to time $t+1$ so that

$$v_{t+1}(g_t(k_t)) = U'[f(g_t(k_t)) - g_{t+1}(g_t(k_t)) + (1-\delta)g_t(k_t)][f'(g_t(k_t)) + (1-\delta)].$$

Substituting this expression into the FOC, we can obtain the functional Euler equation. We also list the Euler equation under the sequential method in the second row for comparison purposes

$$U'[f(k_t) - g_t(k_t) + (1-\delta)k_t] = \beta U'[f(g_t(k_t)) - g_{t+1}(g_t(k_t)) + (1-\delta)g_t(k_t)][f'(g_t(k_t)) + (1-\delta)] \tag{FE}$$

$$U'[f(k_t) - k_{t+1} + (1-\delta)k_t] = \beta U'[f(k_{t+1}) - k_{t+2} + (1-\delta)k_{t+1}][f'(k_{t+1}) + (1-\delta)] \tag{SP}$$

It is straight forward to see that under sequential method, the variable of interest is the optimal path of capital $\{k_{t+1}\}$ and it is determined by the system of second-order difference equations which are the Euler equations. However, for recursive method, the unknown is the sequence of functions or the optimal policies $\{g_t(\cdot)\}$ and it is determined by the system of functional Euler equations. Each functional Euler equation is also of "second order" since it involves both $g_t(\cdot)$ and $g_{t+1}(\cdot)$ as unknowns. For the purpose of the Macroeconomics course, it is sufficient to characterize the optimal solution by the Euler equations or the functional Euler equations without actually describing how to solve them. Only in rare cases, you will be asked to actually solve for the analytical expression of the functional Euler equation. In most cases, there is no close-form solution and we have to rely on numerical algorithms in solving them. I will come back and explain both the analytical example and the numerical algorithm after we have studied recursive methods under infinite horizon.

Recap 2: For recursive method with finite horizon, the optimal policies $\{g_t(\cdot)\}$ is characterized by the system of functional Euler equations.

4. Sequential Optimization: Infinite Horizon

Transversality conditions

Now consider the previous sequence optimization problem in infinite horizon,

$$\begin{aligned} &\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t U(c_t) \\ \text{s.t.}\quad & c_t + i_t \le y_t = f(k_t), && \forall t = 0, \ldots; \\ & c_t, k_{t+1} = (1-\delta)k_t + i_t > 0, && \forall t = 0, \ldots \end{aligned} \tag{2}$$

One can set up the Lagrangian similarly which and derive the [identical] Kuhn-Tucker conditions and the the corresponding Euler equations that characterize the optimal path,

$$U'[f(k_t) - k_{t+1} + (1-\delta)k_t] = \beta U'[f(k_{t+1}) - k_{t+2} + (1-\delta)k_{t+1}][f'(k_{t+1}) + (1-\delta)] \quad \forall t$$

However, instead of having $k_{T+1} = 0$, there is no terminal period under infinite horizon. In this case, we rely on a similar condition, the *transversality condition*, to pin down the behavior of the optimal path in the limit. Note that for the case of finite horizon, the FOC conditions with respect to c_t and k_{T+1} imply,

$$\beta^t U'(c_t) = \lambda_t > 0$$

$$-\lambda_T + \nu_T = 0$$

Hence, by complementary slackness, $k_{T+1} = 0$ and

$$\beta^T U'(c_T) k_{T+1} = 0.$$

Extending this to infinite horizon gives the transversality condition,

$$\lim_{t\to\infty} \beta^t U'(c_t) k_{t+1} = 0.$$

The intuition to this expression is as follows. Under finite-horizon settings, $k_{T+1} = 0$ because there shouldn't be any "savings" in the terminal period. For otherwise, the agent could have opted for a different path of capital and consumed the "saving" in the terminal period to achieve a higher level of utility. Similarly in infinite horizon, the generalized intuition is that $\beta^t U'(c_t) k_{t+1}$ represents how much marginal utility of consumption one can have by shifting one unit of capital to the next period at time t. If this term is positive in the limit, that is, the agent has some "excess" marginal utility in infinity, then potentially he can be better by consuming that "savings of utility" instead.

I now state a more general version of it, which is copied word-for-word from Per Krusell's lecture notes, "Real Macroeconomic Theory". Consider the optimization problem,

$$\max_{\{k_{t+1}\}_{t=0}^{\infty}} \quad \sum_{t=0}^{\infty} \beta^t F(k_t, k_{t+1})$$

$$\text{s.t.} \quad k_{t+1} \geq 0 \quad \forall t$$

If $\{k^*_{t+1}\}_{t=0}^{\infty}$ and $\{\mu^*_t\}_{t=0}^{\infty}$ satisfy

(i) $k^*_{t+1} \geq t \quad \forall t$;

(ii) Euler Equation: $F_2(k^*_t, k^*_{t+1}) + \beta F_1(k^*_{t+1}, k^*_{t+2}) + \mu^*_t = 0 \quad \forall t$;

(iii) Non-negativity constraints and complementary slackness: $\mu_t^* \geq 0$ and $\mu_t^* k_{t+1}^* = 0 \quad \forall t$;

(iv) Transversality condition: $\lim_{t\to\infty} \beta^t F_1(k_t^*, k_{t+1}^*) k_{t+1}^* = 0$;

and $F(k_t, k_{t+1})$ is concave in (k_t, k_{t+1}) and increasing in its first argument, then $\{k_{t+1}^*\}_{t=0}^{\infty}$ maximizes the objective. The proof of this is straightforward and can be found on Per Krussell's textbook.

No Ponzi-Game Condition

Besides transversality conditions, we will also encounter another condition for a certain class of problems, called "No Ponzi-Game" condition. This condition does not appear in every dynamic optimization problem and it is conceptually very different from the transversality condition. Suppose that the economy is as follows. In each period, the agents are endowed with a fixed amount of income ω. The agents then decide how much to consume for each period. Besides the consumption decisions, agents can also borrow and lend at an interest rate r. Denote the amount of bond an individual is holding at time t as b_t, then the optimization problem of the individual is

$$\max_{c_t, b_{t+1}} \sum_t \beta^t U(c_t)$$

$$s.t. \quad c_t + b_{t+1} \leq (1+r)b_t + \omega$$

To see why a "no-Ponzi game" condition is necessary, I will borrow an example as presented in Krussell (2014). Suppose that the optimal stream of consumption is $\{c_t^*\}$. Then absent the no-Ponzi game condition, an individual can do the following to improve his utility without violating budget constraint in any period,

1. Consume $\tilde{c}_0 = c_0^* + 1$. Then the asset holding in period 1 will be $\tilde{b}_1 = b_1^* - 1$, since a negative b_t indicates borrowing.
2. In the subsequent period, consume $\tilde{c}_t = c_t^*$ in each t and hold $\tilde{b}_{t+1} = b_{t+1}^* - (1+r)^t$ amount of assets.

It is easy to see that $\{\tilde{c}_t\}$ yields a higher level of utility for the agent. To see why this arrangement does not violate any budget constraint, we should first acknowledge that as a set of optimal decisions, $\{c_t^*, b_{t+1}^*\}$ must satisfy the budget constraint period by period such that,

$$c_t^* + b_{t+1}^* \leq (1+r)b_t^* + \omega$$

For the new arrangement $\{\tilde{c}_t, \tilde{b}_{t+1}\}$, the budget constraint in period 0 is satisfied because

$$\tilde{c}_0 + \tilde{b}_1 = c_0^* + b_1^* \leq (1+r)b_0^* + \omega = (1+r)\tilde{b}_0 + \omega$$

since the initial level of asset endowment $\tilde{b}_0$ remains the same. Second, the new arrangement does not violate the budget constraint in any subsequent period because,

$$\begin{aligned}
\tilde{c}_t + \tilde{b}_{t+1} &= c_t^* + b_{t+1}^* - (1+r)^t \\
&\leq (1+r)b_t^* + \omega - (1+r)^t \\
&= (1+r)\left[\tilde{b}_t + (1+r)^{t-1}\right] + \omega - (1+r)^t \\
&= (1+r)\tilde{b}_t + \omega + (1+r)^t - (1+r)^t \\
&= (1+r)\tilde{b}_t + \omega.
\end{aligned}$$

Intuitively, this is true because in period 1, the agent will purchase $\tilde{b}_2 = b_2^* - (1+r)$ amount of asset. This means that he is borrowing $(1+r)$ of monies in comparison to that of the original decisions b_2^*. The agent needs to do so, because he has to count on the new borrowing to finance the interest incurred on the extra debt of $\$1$ he borrowed in period 0 ($\tilde{b}_1 = b_1^* - 1$). This requires the agent to borrow exactly an amount of $(1+r)$ in the second period, for otherwise he will violate the budget constraint. Then in the third period, the agent will have to borrow $(1+r)\cdot(1+r)$ amount of monies to pay back the principal and the interest on the $(1+r)$ amount he borrowed in period 2. If he iterates on this logic, then the agent can be better off than the optimal decisions $\{c_t^*, b_{t+1}^*\}$ because he is borrowing more monies in each period to finance the debt from the last period. By doing so, the individual will accumulate a larger and larger amount of debt, but he never has to really pay it back. This is the precise definition of Ponzi Scheme.

Note that while it is in principle possible for *one* agent to use the Ponzi Scheme to better his utility, it will be impossible for *everyone* to do this. The reason is Ponzi games involve with more and more borrowing, and someone has to lend the monies.

To rule out such behavior, we need to invoke a "no-Ponzi" game condition which states that,

$$\lim_{t\to\infty} \frac{b_t}{(1+r)^t} \geq 0.$$

This expression simply says that in the limit, the growth of asset held by the individual must exceed the rate debt is accumulating in the economy. So that an agent is forbidden to borrow more debt than what's necessary.

I should emphasize on an important point before moving on to recursive formulation. The transversality condition is conceptually different from the no-Ponzi condition. Transversality condition is an *optimality* condition while no-Ponzi condition is just a constraint for the economy to behave regularly and it does not say anything about the optimality of the solution.

5. Recursive Formulation: Infinite Horizon

Suppose the problem is stationary and the value of the optimal program from period t is

$$v(k_t) \equiv \max_{\{c_s, k_{s+1}\}_{s=t}^{\infty}} \sum_{s=t}^{\infty} \beta^{s-t} U(k_s)$$

$$\text{s.t.} \quad c_s + k_{s+1} - (1-\delta)k_s \leq f(k_s), \quad \forall s \geq t$$

Note that we get rid of the t subscript in the value function $v(\cdot)$, since the problem is stationary by assumption and the value will be the same for all t. Following the same steps as in the finite horizon case, the Bellman equation is then

$$v(k) = \max_{k'} \Big\{ U[f(k) - k' + (1-\delta)k] + \beta v(k') \Big\}$$

where we adopt the prime notation to denote values in the next period. The corresponding Euler equation is

$$U'[f(k) - g(k) + (1-\delta)k] = \beta U'[f(g(k)) - g(g(k)) + (1-\delta)g(k)][f'(g(k)) + (1-\delta)]$$

We are interested in solving for either the policy functions $g(k)$ or the value functions $v(k)$. Knowing one function is sufficient to solve for the other, since $g(k)$ is the maximizer for solving $v(k)$ using the Bellman equations. However, only under rare circumstances we can solve for analytical solutions. In the sections that follow, we will first introduce a special case where analytical solution is available and then discuss how we can solve for the values and policies numerically. Both approaches utilize a mathematical concept called "contraction mapping" which we will discuss in length during the last lecture.

6. Analytical Solutions

Consider a special case where $U(c) = \log(c)$ and $f(k) = Ak^{\alpha}$ for some constants A and $\alpha < 1$. For simplicity, we can assume that capital is perishable and there is full depreciation such that $\delta = 1$. Hence, the Bellman equation to this problem is as follows,

$$v(k) = \max_{k'} \{\log[Ak^{\alpha} - k'] + \beta v(k')\}$$

To solve for the associated policy function, we want to first solve for the value functions since the policies are just the maximizer to the Bellman equation. To do this, we need to invoke a mathematical concept called "contraction mapping theorem". We will only discuss it in details later. For now, all we need to know is that given the functional form assumptions, if we iterate over the Bellman equation starting from an initial guess of the value function, it is guaranteed that we will find an unique fixed point. This fix point will be the desired solution.

Let's first specify our initial guess of the value function. Suppose that we guess $v^0(k) = 0$ for all k. Then we can write the Bellman equation as follows,

$$v^1(k) = \max_{k'} \left\{ \log[Ak^\alpha - k'] + \beta v^0(k') \right\} = \max_{k'} \left\{ \log[Ak^\alpha - k'] \right\}$$

Clearly, the maximizer to this problem is to choose $k' = 0$ regardless the value of k. Hence, the associated value function is

$$v^1(k) = \log\left(Ak^\alpha\right) = \log(A) + \alpha \log(k).$$

We can substitute this value function into the Bellman equation again to solve for $v^2(k)$,

$$v^2(k) = \max_{k'} \left\{ \log[Ak^\alpha - k'] + \beta v^1(k') \right\} = \max_{k'} \left\{ \log[Ak^\alpha - k'] + \beta \log\left(Ak^\alpha\right) \right\}$$

The solution to this problem is then

$$k' = \frac{\alpha\beta A k^\alpha}{1 + \alpha\beta}$$

and the associated value function is

$$v^2(k) = (\alpha + \alpha^2\beta)\log k + \log\left(A - \frac{\alpha\beta A}{1+\alpha\beta}\right) + \beta \log A + \alpha\beta \log \frac{\alpha\beta A}{1+\alpha\beta}.$$

Notice that a pattern has emerged during the iterations. Each value function is written in a specific form such that,

$$v^n(k) = a_n + b_n \log k$$

To see this, for the first iteration, we have $a_1 = logA$ and $b_1 = \alpha$. For the second iteration, we have

$$\begin{aligned} a_2 &= \log\left(A - \frac{\alpha\beta A}{1+\alpha\beta}\right) + \beta \log A + \alpha\beta \log \frac{\alpha\beta A}{1+\alpha\beta} \\ b_2 &= \alpha + \alpha^2\beta \end{aligned}$$

Then without loss of generality, we can guess that the "fixed point" of the value function, in which the sequence $v^n(k)$ converges to $v(k)$ is of the following functional form,

$$v(k) = a + b \log k.$$

We can substitute this into the Bellman equation and obtain

$$v(k) = a + b\log k = \max_{k'} \{log(Ak^\alpha - k') + \beta(a + b\log k')\}$$

Solving this optimization problem we can obtain the policy function as

$$k' = \frac{\beta b}{1+\beta b} Ak^\alpha$$

Substituting this policy function back into the Bellman equation, we can obtain two separate expressions for the LHS and the RHS of the Bellman equation. The left-hand side is simply

$$LHS = a + b\log k$$

and the right-hand side is

$$RHS = (1+b\beta)\log A + \log\left(\frac{1}{1+b\beta}\right) + a\beta + b\beta\log\left(\frac{\beta b}{1+\beta b}\right) + (\alpha + \alpha\beta b)\log k.$$

By comparing coefficients, we then know that the following relationships must be true

$$\begin{aligned} a &= (1+b\beta)\log A + \log\left(\frac{1}{1+b\beta}\right) + a\beta + b\beta\log\left(\frac{\beta b}{1+\beta b}\right) \\ b &= \alpha + \alpha\beta b \end{aligned}$$

This is system of linear equations with two unknowns a and b. The solution to this system of equations is

$$\begin{aligned} a &= \frac{1}{1-\beta}\frac{1}{1-\alpha\beta}[\log A + (1-\alpha\beta)\log(1-\alpha\beta) + \alpha\beta\log(\alpha\beta)] \\ b &= \frac{\alpha}{1-\alpha\beta} \end{aligned}$$

Substituting these values back to the expression for policy function, we obtain that

$$k' = \frac{b\beta}{1+b\beta}Ak^\alpha = \alpha\beta Ak^\alpha.$$

7. Numerical Solutions

We have shown how to solve for the analytical solutions of dynamic programming problems. However, for many classes of dynamic programming problem, this is infeasible. In this section, we introduce two approaches to numerically solve for dynamic programming problems: value function iteration and policy function iteration.

Value Function Iteration

Consider the deterministic growth model. The unknowns we are interested in are $\{c_t, k_{t+1}, v(\cdot)\}$. Notice that to solve for all these unknowns, it is sufficient to solve for the value function $v(\cdot)$. Given the value function and an initial capital stock k_0, we can substitute the value function back to the Bellman equation and solve for the optimal path of capital recursively. To be more specific, to solve for the choice of capital stock in period 1, we can substitute the value function $v^*(\cdot)$ such that k_1^* is the argument max to the Bellman equation,

$$k_1^* = \arg\max_{k_1} \Big\{ U[f(k_0) - k_1 + (1-\delta)k_0] + \beta v^*(k_1) \Big\}$$

Knowing k_1^*, we can then substitute it back to the Bellman equation and find k_2^*.

$$k_2^* = \arg\max_{k_2} \Big\{ U[f(k_1^*) - k_2 + (1-\delta)k_1^*] + \beta v^*(k_2) \Big\}$$

Applying this logic recursively, we can solve for the optimal path of capital ad infinitum. Hence, the problem reduces to finding the value function $v^*(\cdot)$. The conventional method to achieve this is through *value function iteration.*

As literally suggested by its name, value function iteration operates by iterating on the value function. To start, we should guess for an initial value function $v(k)$ for any k and call this guess $v^0(k)$. Given *every* value of the state variable k, we can substitute the associated value $v^0(k)$ and compute the value $v^1(k)$ *as implied* by the Bellman equation, such that

$$v^1(k) = \max_{k'} \Big\{ U[f(k) - k' + (1-\delta)k] + \beta v^0(k') \Big\}$$

Collecting *all* the values of $v^1(k)$ for *every* value of state variable k, we then have another value function $v^1(\cdot)$. We can substitute this value function for every possible state k and compute again the implied values by the Bellman equation as,

$$v^2(k) = \max_{k'} \Big\{ U[f(k) - k' + (1-\delta)k] + \beta v^1(k') \Big\}$$

Iterating this logic recursively, we will stop the iteration until the $v^{n+1}(k)$ is close enough to $v^n(k)$ for every state k, such that the ℓ_1 distance (sum of differences) $\big\|v^{n+1}(k) - v^n(k)\big\|_1$ between the values is smaller than a prescribed error criterion ϵ. The following pseudo-code provides a summary of this algorithm. (See "Algorithm 1" on next page).

A Practical Version of Value Function Iteration

The algorithm that we outlined earlier is valid in principle, yet it is not practical. One reason is that the choice of k_{t+1} is continuous and involves an +infinite number of possible k_{t+1}. To

Algorithm 1: Value Function Iteration

```
1 Guess an initial value function $v^0(k)$, e.g., $v^0(k) = 0$ for all $k$.
2 Specify a stopping criterion $\epsilon$, set $n = 0$, and set Error $> \epsilon$.
3 while Error $>= \epsilon$, do
      foreach $k \in [0, \infty]$ do
          Compute $v^{n+1}(k) = \max_{k'} \left\{ U[f(k) - k' + (1-\delta)k] + \beta v^n(k') \right\}$.
      Error $= \left\| v^{n+1}(k) - v^n(k) \right\|_1$.
      Update $n = n + 1$.
```

circumvent this problem, a technique that is often invoked is to *discretize* the state space. Instead of evaluating infinitely many states, we can set up a *grid* as a discretization of the state space. In particular, assuming that the value of k can only be between the interval of $[\underline{k}, \bar{k}]$, we can set up a grid with a step value Δk. Then the grid consists of points $\{\underline{k}, \underline{k} + \Delta k, \underline{k} + 2\Delta k, \ldots, \bar{k}\}$ and there will be a total of $(\bar{k} - \underline{k})/\Delta k$ elements in the grid.[2] The modified algorithm will evaluate the value function only at these points on the grid.[3] The pseudo-code is as follows,

Algorithm 2: Value Function Iteration – Discretization

```
1 Specify a step size $\Delta k$.
2 Given that $k$ falls in the range of $[\underline{k}, \bar{k}]$, set up a vector/grid consists of $[\underline{k} : \Delta k : \bar{k}]$.
3 Guess an initial value function $v^0(k)$, e.g., $v^0(k) = 0$ for all $k \in [\underline{k} : \Delta k : \bar{k}]$.
4 Specify a stopping criterion $\epsilon$, set $n = 0$, and set Error $> \epsilon$.
5 while Error $>= \epsilon$, do
      foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
          Compute $v^{n+1}(k) = \max_{k'} \left\{ U[f(k) - k' + (1-\delta)k] + \beta v^n(k') \right\}$.
      Error $= \left\| v^{n+1}(k) - v^n(k) \right\|_1$.
      Update $n = n + 1$.
```

The modified algorithm is not without flaws. Readers will quickly realize that to update value function based on the Bellman equation, we have to evaluate $v^n(k')$ for a continuum of k'. However, we only have $v(\cdot)$ for a finite number of k given the discretization of state space. The technique employed by most economists to address this concern is called *interpolation*. We will introduce linear interpolation which is the simplest interpolation method. For other more advanced and more practical methods such as Schumaker splines, interested readers may refer to Violante (2015) for further information.

[2]The interpretation is that if Δk becomes small enough, the grid will converge to a continuous one.
[3]The grid we introduced here is essentially a uniform grid. For non-uniform grid, quadrature grid, and stochastic grid, readers may refer to...

Linear Interpolation of Value Function

Suppose that we have a grid of points $k(i)$ for $i = 1, ..., (\bar{k} - \underline{k})/\Delta k$. Then in each step where we update the value function $v^n(k)$, we need to know the values of $v^n(k)$ for some k lies between say $k(i)$ and $k(i + 1)$, in order to find the maximizer to the LHS of the Bellman equation. Linear interpolation method says that we can interpolate the value of $v(k)$ as follows,

$$v(k) \approx v(k(i)) + \frac{v(k(i+1)) - v(k(i))}{k(i+1) - k(i)} [k - k(i)]$$

The intuition of the linear interpolation method is simple. As the name suggests, it is a method that interpolates the values between two points via a linear, straight line. Suppose we have two points, $(x_0, f(x_0))$ and $(x_1, f(x_1))$, on a Cartesian plane. Then geometry dictates that for any point (x, y) that lies on the *linear interpolant*,

$$\frac{y - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad \Rightarrow \quad f(x) = y = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Value Function Iteration: Limitations

While value function iteration is considered as an accurate method, in practice it can be very slow. One reason is that if we choose a finer grid, there will be too many grid points/states to evaluate for *each* iteration. To deal with this problem, researchers typically utilize the power of modern computers and resort to parallel computations using Central Processing Units (CPU) or Graphical Processing Units (GPU). Modern CPUs are usually equipped with multiple cores/nodes. The CPU on your personal laptop may carry two nodes, while the High Performance Computing Cluster at SMU carries 272 cores.[4] If one is willing to compromise on the nature of the operation (i.e., arithmetic vs logical), a GPU may be even more powerful and cost-effective as one consumer-grade GPU could carry more than 2,000 CUDA cores. Given these computational power, one way to circumvent the computational burden of evaluating every grid point is simply to divide the grid points equally among the CPUs. These CPUs will then *simultaneously* evaluate the value functions at those points. Very loosely speaking, if there are n cores available, we can naively expect an n-times speed up. (though this is an inaccurate characterization).

The second reason why value function iteration is slow, is that it is in particular slow to find a maximizer/optimal policy for the Bellman equation. This is especially true when the functional forms are non-linear. To address this concern, we will introduce Howard's Improvement Algorithm (Howard, 1960) in the next subsection. Later on, we will also introduce Policy Function Iteration which can also be interpreted as an application of Howard's algorithm.

[4]This is considered as a small HPC cluster. Many universities have multiple HPC cluster each equipped with more than tens of thousands cores.

Howard's Improvement Step

The basic idea of Howard's Improvement Step, is that instead of finding a maximizer for each iteration, we should use the same maximizer repeatedly for some number of iterations and then update the maximizer. To be more specific,suppose that we have an initial maximizer $k' = \tilde{k}$ when we update the value function from $v^n(k)$ to $v^{n+1}(k)$,

$$g^{n+1}(k) = \tilde{k}(k) \equiv \arg\max_{k'} \; U[f(k) - k' + (1-\delta)k] + \beta v^n(k')$$

Notice that this $\tilde{k}$ should be a function (a sequence of values) instead of a scalar value. The reason is that we are finding the argument max for every single value of k. Hence, we need a corresponding $\tilde{k}$ for every k. For the sake of accuracy, we denote this function to be $g^{n+1}(k)$ and the corresponding $v^{n+1}(k)$ to be $v_0^{n+1}(k)$, such that by the Bellman equation,

$$v_0^{n+1}(k) \equiv U\left[f(k) - g^{n+1}(k) + (1-\delta)k\right] + \beta v^n(g^{n+1}(k))$$

Howard (1960)'s idea is that instead of straight away jumping from $v^{n+1}(k)$ to $v^{n+2}(k)$, we should exploit $g^{n+1}(k)$ repeatedly for many times before updating to $v^{n+2}(k)$. Specifically, we should update $v_0^{n+1}(k)$ for H times using the Bellman equation and the same policy function $g^{n+1}(k)$ [every time],

$$v_{h+1}^{n+1}(k) = U\left[f(k) - g^{n+1}(k) + (1-\delta)k\right] + \beta v_h^{n+1}(g^{n+1}(k))$$

Given the updated $v_H^{n+1}(k)$, we can then find the maximizer $g^{n+2}(k)$ again and update $v^{n+2}(k)$. (it will be used as $v_0^{n+2}(k)$). The following pseudo-code summarizes the algorithm.

Algorithm 3: Value Function Iteration - Howard's Improvement

```
1 Specify a step size $\Delta k$.
2 Given that $k$ falls in the range of $[\underline{k}, \bar{k}]$, set up a vector/grid consists of $[\underline{k} : \Delta k : \bar{k}]$.
3 Guess an initial value function $v^0(k)$, e.g., $v^0(k) = 0$ for all $k \in [\underline{k} : \Delta k : \bar{k}]$.
4 Specify a stopping criterion $\epsilon$, set $n = 0$, and set Error $> \epsilon$.
5 while Error $>= \epsilon$, do
      foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
          Compute $v_0^{n+1}(k) = \max_{k'} \left\{U[f(k) - k' + (1-\delta)k] + \beta v^n(k')\right\}$.
          Store the maximizer $g^{n+1}(k)$ in a vector.
      Set $h = 0$.
      while $h < H$, do
          foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
              Compute $v_{h+1}^{n+1}(k) = U\left[f(k) - g^{n+1}(k) + (1-\delta)k\right] + \beta v_h^{n+1}(g^{n+1}(k))$.
          Update $h = h + 1$.
      Set $v^{n+1}(k) = v_H^{n+1}(k)$ and compute Error $= \left\|v^{n+1}(k) - v^n(k)\right\|_1$.
      Update $n = n + 1$.
```

Several observations are in order. First, the fact that we add more intermediate steps in updating $v_h^{n+1}(k)$ does not mean that there are more computational burden. The purpose of these improvement steps are meant to cut down the number of iterations needed for $v^{n+1}(k)$ to converge, that is N will become smaller. Hence, total computational burden is smaller. This is true because we are facing a tradeoff between adding more intermediate steps updating from $v_h^{n+1}(k)$ to $v_{h+1}^{n+1}(k)$ and cutting down the number of iterations needed to update from $v^n(k)$ to $v^{n+1}(k)$. The computational burden of the latter trumps the former, as for the intermediate steps we are evaluating the Bellman equation using a given maximizer $g^{n+1}(k)$ repeatedly,

$$v_{h+1}^{n+1}(k) = U\left[f(k) - g^{n+1}(k) + (1-\delta)k\right] + \beta v_h^{n+1}(g^{n+1}(k))$$

whereas for the latter steps we are trying to *find* a maximizer which is time-consuming,

$$v^{n+1}(k) = \max_{k'} \Big\{ U[f(k) - k' + (1-\delta)k] + \beta v^n(k') \Big\}.$$

Second, for the intermediate steps or the "Howard's improvement steps", the value function is "improving" for each iteration. This is true because $U[\cdot]$ at the policy (maximizer) $g^{n+1}(k)$ is positive. The improvement comes from a tendency that the policy functions usually converge faster that the value functions. By using the policy functions as a proxy repeatedly, we are "fine-tuning" or "improving" the value function so that it "catches up" with the current policy function, in the sense that the value function will become closer to the true value. (Because the improvement step is also a contraction mapping hence bound to converge).

Policy Function Iteration

An alternative to value function iteration is to iterate over the policy functions. The basic idea is to start with a guess of the policy function $g^0(k)$ and then solve for the associated value functions based on the system of [linear] Bellman equations,

$$v(k) = U\left[f(k) - g^0(k) + (1-\delta)k\right] + \beta v(g^0(k)).$$

Note that that there are in total of N unknown $v(k)$ in N equations, where N is the number of grid points for k. Denote these solved values to be $v^0(k)$. After solving for these $v^0(k)$, the next step is to find a new policy function based on the value functions, such that

$$g^1(k) = \arg\max_{k'} \quad \{U\left[f(k) - k' + (1-\delta)k\right] + \beta v^0(k')\}$$

Given the new policy function $g^1(k)$, one can iterate over these steps until the policy function converges. The following pseudo-code summarizes the algorithm for policy function iteration.

Algorithm 4: Policy Function Iteration

```
1 Specify a step size $\Delta k$.
2 Given that $k$ falls in the range of $[\underline{k}, \bar{k}]$, set up a vector/grid consists of $[\underline{k} : \Delta k : \bar{k}]$.
3 Guess an initial policy function $g^0(k)$, e.g., $g^0(k) = 0$ for all $k \in [\underline{k} : \Delta k : \bar{k}]$.
4 Specify a stopping criterion $\epsilon$, set $n = 0$, and set Error $> \epsilon$.
5 while Error $>= \epsilon$, do
      Solve the system of linear equations with $(\bar{k} - \underline{k})/\Delta k$ unknown $v^n(k)$, where the
        system of equations is $v^n(k) = U\left[f(k) - g^n(k) + (1-\delta)k\right] + \beta v^n(g^n(k))$.
      foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
          Compute $g^{n+1}(k) = \arg\max_{k'} \{U\left[f(k) - k' + (1-\delta)k\right] + \beta v^n(k')\}$.
      Error $= \left\| g^{n+1}(k) - g^n(k) \right\|_1$.
      Update $n = n + 1$.
```

The advantage of policy function iteration over value function iteration is that, it usually takes less number of iterations for the algorithm to converge. However, it is unclear how computational time for *each round* of iteration compares with that of value function iteration; it can go either way.

8. Parallelization of Value Function Iteration (Optional)

We have briefly discussed that one way to speed up the computation time is through parallel programming. In this section, we discuss it in greater details. Interested readers may also refer to Fernández-Villaverde and Valencia (2018), Aldrich (2014), and Aldrich et al. (2011) for further information.

Parallelization via CPU

One way to implement parallelization is to utilize the multi-node CPUs in our personal desktops or the HPC cluster at SMU. Recall that the algorithm for value function iteration with discretization is as follows (Algorithm 5 on next page). The difficulty arises because we need to compute the value function $v^{n+1}(k)$ for each grid point k in the grid $\left[\underline{k} : \Delta k : \bar{k}\right]$. Now, suppose that we have $N = 16$ cores available in the HPC cluster. We can assign $(\bar{k} - \underline{k})/16\Delta k$ grid points to each core in the CPU. This is easy to implement in Matlab, as one need only to change "`for`" to "`parfor`" during the iteration over grid points, which corresponds to the "**foreach** $k \in \left[\underline{k} : \Delta k : \bar{k}\right]$ **do**" in the pseudo-code. Matlab will automatically divide the grid points among each node/core/processor and compute the value function simultaneously. Theoretically, though inaccurately, we could achieve a speed up close to

N times. This is a huge improvement for the feasibility of a research project. Suppose that originally without parallelization, the program converges in 3 months. If one can speed it up by 16 times, then it will converge within 1 week.[5]

Algorithm 5: Value Function Iteration – Discretization

```
1 Specify a step size $\Delta k$.
2 Given that $k$ falls in the range of $[\underline{k}, \bar{k}]$, set up a vector/grid consists of $[\underline{k} : \Delta k : \bar{k}]$.
3 Guess an initial value function $v^0(k)$, e.g., $v^0(k) = 0$ for all $k \in [\underline{k} : \Delta k : \bar{k}]$.
4 Specify a stopping criterion $\epsilon$, set $n = 0$, and set $Error > \epsilon$.
5 while $Error >= \epsilon$, do
      foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
          Compute $v^{n+1}(k) = \max_{k'} \left\{ U[f(k) - k' + (1-\delta)k] + \beta v^n(k') \right\}$.
      $Error = \left\| v^{n+1}(k) - v^n(k) \right\|_1$.
      Update $n = n + 1$.
```

One problem commonly seen among researchers new to HPC is that, they think their program will automatically run much faster on a HPC cluster. Therefore, they do not utilize parallelization in their programs. While it is true to some extent the HPC cluster is better than the personal desktops, without parallelization, the computing power is often unexploited. To establish this point, we run an experiment where a computationally-intensive program is implemented with and without parallelization on a 6-core CPU. The following figure is a snapshot from a CPU-monitoring software that records the utilization of each core/node.

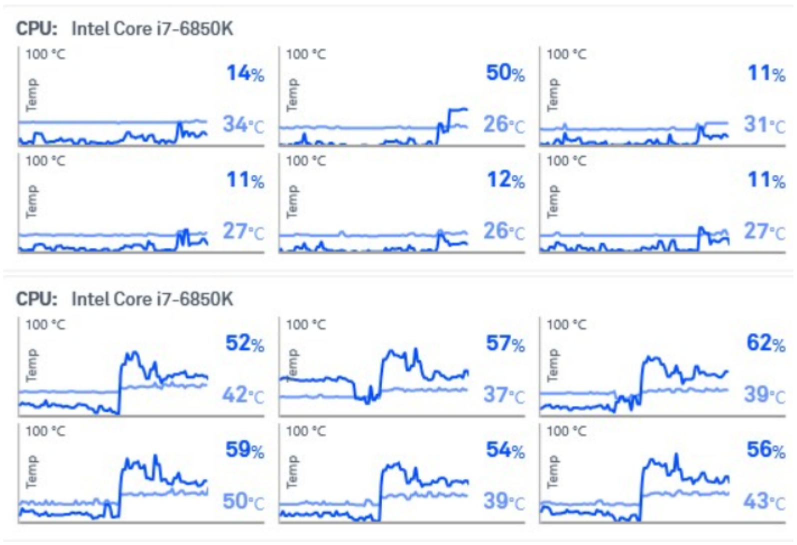

Figure 1: Utilization of CPU with and without parallelization.

[5]However, one should not take parallelization as a silver bullet. First of all, there are many bottlenecks that limits the extent of parallelization, such as hardware optimization and programming optimization. Second, parallelizaton should not be used as a substitute for bad programming.

The top panel records the activities in all 6 cores under the program that there is no parallelization. It is clear that only 1 core is being used intensively by the computer where the level of utilization is around 50% and all other 5 cores are idle. In contrast, for the bottom panel where parallelization is employed (i.e., change "`for`" to "`parfor`"), all 6 cores are being used heavily for the computation. The key message that I wish to convey is that, while it is true HPC cluster helps us to speed up the computation [to a small extent] as many built-in Matlab command already utilizes the multi-thread architecture, the HPC cluster will only be most useful when an explicit parallelization structure is present in the program.

Parallelization via GPU

Another approach to implement parallel computing is to use the Graphical Processing Units (GPU). The basic idea is that NVIDIA GPUs are usually equipped with hundreds and thousands of GPUs. If we can reduce the parallel operations to arithmetic operations (the basic matrix computations) instead of logical operations (true or false operations), then each GPU core will be as good as a CPU core for our purposes. However, the unit price is much cheaper. The current price of a GTX 1080Ti GPU with 3,500 CUDA cores is about $600 USD whereas for a 16-core CPU the cost will be about $800 USD. This implies that the per-core unit price of CPU is 3,000 times higher than that of GPU.

To make a comparison of running parallelization using CPU and GPU, I reproduce the following table, which is part of Table 3 in Aldrich et al. (2011).[6] [7] [8]

Number of grid points	16	32	64	128	256	512	1,024
CPU time in seconds	0.03	0.08	0.19	0.5	1.36	3.68	10.77
GPU time in seconds	1.42	1.45	1.49	1.52	1.57	1.69	1.93

Number of grid points	2,048	4,096	8,192	16,384	32,768	65,536
CPU time in seconds	34.27	117.32	427.50	1615.40	6270.37	24588.50
GPU time in seconds	2.45	3.65	6.37	11.86	23.56	48.32

Table 1: Comparison of running time, adapted from Aldrich et al. (2011) Table 3

The CPUs they used are two 2.66 GHz Intel Xeon E5345 CPUs with a total of 8 cores, whereas the GPU they used is a 1.30 GHz NVIDIA GeForce GTX 280 with 240 CUDA cores. It is clear from this comparison that GPU really speeds up computation especially for a finer

[6]The number of grid points represents the grid points for capital alone.

[7]Both parallelization implementations are written in C language, so this is a fair comparison.

[8]The authors set the precision to double precision, that is the number as accurate up to 15 or 16 decimal digits. However, many GPUs, including the GPU used by the authors, are optimized for single precision computations. If one is willing to compromize on the level of precision and accept an accuracy of 6 to 7 decimal digits, then the performance of the GPU could double.

grid, which is often demanded in quantitative work. For the finest grid with 65,536 grid points, the amount of time for CPU parallelization is about 7 hours. With parallelization using an entry-level consumer-grade GPU, the computational time is reduced to under 1 minute.

I wish to further mention two more observations. The first observation is that if we consider a case without any parallelization, the difference is even more pronounced. By my experiences, an 8-core CPU in general will only achieve a speed-up of factor of 4 in comparison to a case without any parallel programming. This means that without parallelization, even if we write down the program in C, the computation time is about 28 hours or a whole day.

The second observation is that, although the performance of GPU is already impressive, there are more room for improvement. The GPU that Aldrich et al. (2011) used is only an entry level GPU. More recent consumer-grade GPU such as NVIDIA GeForce GTX 1080Ti comes with more than 3,500 CUDA cores. This implies the computational power is about 11.3 TFLOPS for 1080Ti which is about 20 times that of GTX 280 used by Aldrich et al. (2011).

So what's the general rule of thumb? Which computational approach should we follow in practice? My answer consists of two parts. The first part is whether we should use parallelization to solve for value function iteration. The answer is yes. There is virtually no cost associated with learning parallelization using CPU. In Matlab, all you need is "`parfor`". This is similar in other languages. The second part then relates to the question on whether we should use CPU or GPU. For this part, my answer is, it depends. There is an inherent tradeoff between saving the computational time and spending the time to grasp the extra techniques. Learning to code in CUDA C is not an trivial experience. My personal take is that, we should only use GPU when the parallel program using CPU takes more than 1 week to finish running (in total). It may take you perhaps a month to fully apply CUDA C in your own research. If all it gets for you is to reduce the computation from 1 day to 1 minute, it is not very worthwhile.

9. Conclusion

In this lecture, we discussed dynamic optimization in a deterministic environment. We first introduce different ways to characterize the solution system. The first method exploits the Euler equations using the sequence problem. The second method which is dynamic programming characterize the optimal policies using Bellman equations. We then explain on how we can solve the dynamic programs both analytically and numerically. For the next lecture, we will talk more about how to characterize dynamic optimization in a stochastic environment.

References

Aldrich, Eric M., "GPU Computing in Economics," in Karl Schmedders and Kenneth L. Judd, eds., *Handbook of Computational Economics*, Vol. 3, North-Holland, 2014, chapter 10, pp. 557–598.

—, Jesús Fernández-Villaverde, Ronald A. Gallant, and Juan F. Rubio-Ramírez, "Tapping the Supercomputer Under Your Desk: Solving Dynamic Equilibrium Models with Graphics Processors," *Journal of Ecnomic Dynamics and Control*, 2011, *35* (3), 386–393.

Fernández-Villaverde, Jesús and David Zarruk Valencia, "A Practical Guide to Parallelization in Economics," *Working Paper*, 2018.

Howard, Ronald A., *Dynamic Programming and Markov Decision Processes*, 1 ed., Cambridge, Massachusetts: The MIT Press, 1960.

Krussell, Per, *Real Macroeconomic Theory*, Draft Textbook, 2014.

Violante, Gianluca, *Lecture Notes for Quantitative Macroeconomics*, NYU PhD Lecture Notes, 2015.

Lecture 4: Dynamic Optimization with Uncertainty

Xin Yi

1. Introduction

In this lecture, we will study dynamic optimizations in stochastic environments where there are random shocks in the economy. However, the presence of random shocks does not necessarily complicate the picture. If you think about random shocks in an Arrow-Debreu environment, then all stochastic models are static.[1] Therefore, the techniques presented here can be seen as a generalization from what we are doing in the previous lecture.

It is worthwhile to emphasize again that, the models presented here are all partial equilibrium models. That is, we only focus on the behavior from the perspective of an individual and we do not think about market clearing conditions. A large part of these lectures will be dedicated to solving the model. In the PhD Macroeconomics 1 class, you will need to apply the techniques from this class to construct a model that describes the general equilibrium for the whole economy. Then in the PhD Macroeconomics 2 class, you will [partly] learn about how to solve the general equilibrium model numerically. It is also important to keep in mind on the difference between a planning problem and a market economy. Though this is an obvious distinction to all economics students, people do sometimes get confused when they first touch dynamic macroeconomic theory.

2. Common Stochastic Processes in Macroeconomics

By now you should have a rigorous understanding about stochastic processes from your Econometrics class and previous math camps. In this section, we will briefly revisit some stochastic environments that are commonly seen in macroeconomics.

Markov Chains

A formal definition of stationary Markov chains is omitted in here as it is already covered in the previous courses. The most important thing to know about Markov chains for our pur-

[1] We must also assume that agents are risk-neutral. Otherwise, there will be an additional second-moment effect for the stochastic models.

pose, is the properties of a first-order Markov chain, where only the previous state matters for the current state, and any histories beyond that will not matter. Formally, we define a history at time t as a sequence of states reversely ordered by time,

$$z^t = (z_t, z_{t-1}, \dots, z)$$

where $z_t \in \mathcal{Z}$ is the state at time t. A first-order Markov chain then satisfies the following properties,

$$\pi[(z_{t+1}, z^t)|z^t] = \pi[(z_{t+1}, z_t)|z_t]$$

as opposed to higher-order models where an n-th order Markov model satisfies the following

$$\pi[(z_{t+1}, z^t)|z^t] = \pi[z_{t+1}|(z_t, z_{t-1}, \dots, z_{t-n+1})].$$

Linear Stochastic Difference Equations

As the name suggests, linear stochastic difference equation is the class of processes that are recursively defined by linear functional forms in a random environment. Readers who are interested in a more formal definition may refer to Krussell (2014). The most common linear stochastic difference equation that we will see in macroeconomics is the first-order autoregressive process, i.e., the AR(1) process. In the context of a neoclassical growth model, suppose now that A_t represents a random Hicks-neutral productivity shock that follows an AR(1) process, then

$$A_{t+1} = \rho_0 + \rho_1 A_t + \varepsilon_{t+1}$$

where it is assumed that

$$E_t[\varepsilon_{t+1}] = 0, \quad E_t[\varepsilon_{t+1}^2] = \sigma^2, \quad \text{and } E_t[\varepsilon_{t+k}\varepsilon_{t+k+1}] = 0.$$

and we can show that given a A_0, we will have a stationary process so long as $|\rho_1| < 1$.

3. Sequential Optimization under Stochastic Environments

The following example is largely drawn from Krussell (2014). Consider the neoclassical growth model under stochastic environments. Suppose that the production function now has an extra element, the productivity shock z_t, and it is random,

$$y_t = z_t f(k_t).$$

Further assume that $z_t \in \mathcal{Z}$ follows a first order Markov process such that

$$\pi\left[(z_{t+1}, z^t)|z^t\right] = \pi\left[(z_{t+1}, z_t)|z_t\right]$$

where z^t is a history of realizations of productivity shocks at time t such that $z^t = (z_t, z_{t-1}, \ldots, z_0)$. (z_t, z^{t-1}) is another way to denote the history at time t, z^t. $\pi\left[(z_{t+1}, z^t)|z^t\right]$ then denotes the probability of observing state z_{t+1} at time $t+1$ conditional on the past history is z^t. This is a first-order Markov process because history beyond the previous time period does not matter.

Given these assumptions, the optimization problem for a stochastic neoclassical growth model is as follows

$$\max_{c_t(z^t), k_{t+1}(z^t)} \quad \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t U(c_t)\right] \equiv \sum_{t=0}^{\infty} \sum_{z^t \in \mathcal{Z}^t} \beta^t \pi(z^t) U\left[c_t(z^t)\right]$$

$$s.t. \quad c_t(z^t) + k_{t+1}(z^t) \leq z_t f\left[k_t(z^{t-1})\right] + (1-\delta) k_t(z^{t-1}) \quad \forall t, \forall z^t,$$

where $\mathcal{Z}^t$ is the set of possible history at time t. The Lagrangian to this problem is,

$$\mathcal{L} = \sum_{t=0}^{\infty} \sum_{z^t \in \mathcal{Z}} \beta^t \pi(z^t) U\left[c_t(z^t)\right] - \sum_{t=0}^{\infty} \sum_{z^t \in \mathcal{Z}^t} \lambda_t(z^t) \Big\{ c_t(z^t) + k_{t+1}(z^t) - z_t f\left[k_t(z^{t-1})\right] - (1-\delta) k_t(z^{t-1}) \Big\}$$

The associated first-order conditions with respect to $c_t(z^t)$ and $k_{t+1}(z^t)$ are

$$\beta^t \pi(z^t) U'[c_t(z^t)] = \lambda_t(z^t)$$

$$\lambda_t(z^t) = \sum_{z_{t+1} \in \mathcal{Z}} \lambda_{t+1}(z_{t+1}, z^t) \Big\{ z_{t+1} f'\left[k_{t+1}(z^t)\right] + (1-\delta) \Big\}.$$

The second FOC with respect to $k_{t+1}(z^t)$ may be a bit difficult to comprehend. The reason we are summing over z_{t+1} is as follows. When we differentiate with respect to $k_{t+1}(z^t)$, we need to do this for the term $\lambda_{t+1}(z^{t+1}) \Big\{ z_{t+1} f\left[k_{t+1}(z^t) - (1-\delta) k_{t+1}(z^t)\right] \Big\}$ in the Lagrangian. Notice that this term involves with both z^{t+1} and z^t. However, we are only differentiating with respect to $k_{t+1}(z^t)$ for *one* particular history z^t at time t. Given z^t, we can reach different histories z^{t+1} at time $t+1$, depending on the particular value of productivity shock z_{t+1} at time $t+1$. Hence, a summation sign is necessary.

Given the FOCs, we should first use the FOC with respect to consumption to yield an inter-temporal expression,

$$\beta \frac{\pi(z^{t+1}) U'\left[c_{t+1}(z^{t+1})\right]}{\pi(z^t) U'\left[c_t(z^t)\right]} = \frac{\lambda_{t+1}(z^{t+1})}{\lambda_t(z^t)}$$

which is equivalent to

$$\beta \frac{\pi(z_{t+1}, z^t) U' \left[c_{t+1}(z_{t+1}, z^t) \right]}{\pi(z^t) U' \left[c_t(z^t) \right]} = \frac{\lambda_{t+1}(z_{t+1}, z^t)}{\lambda_t(z^t)}$$

Multiplying both sides of the equation by $z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta)$ and summing it over all possible z_{t+1} yields

$$\begin{aligned} & \sum_{z_{t+1} \in \mathcal{Z}} \beta \frac{\pi(z_{t+1}, z^t) U' \left[c_{t+1}(z_{t+1}, z^t) \right]}{\pi(z^t) U' \left[c_t(z^t) \right]} \Big\{ z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta) \Big\} \\ = & \sum_{z_{t+1} \in \mathcal{Z}} \frac{\lambda_{t+1}(z_{t+1}, z^t)}{\lambda_t(z^t)} \Big\{ z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta) \Big\}. \end{aligned}$$

But the RHS of this equation is just 1, based on the FOC with respect to $k_{t+1}(z^t)$. Hence, we have that

$$\pi(z^t) U' \left[c_t(z^t) \right] = \sum_{z_{t+1} \in \mathcal{Z}} \beta \pi(z_{t+1}, z^t) U' \left[c_{t+1}(z_{t+1}, z^t) \right] \Big\{ z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta) \Big\}.$$

Furthermore, by the definition of conditional probability that $\pi \left[(z_{t+1}, z^t) | z^t \right] = \pi(z_{t+1}, z^t) / \pi(z^t)$, we can change the above to the final expression of stochastic Euler equation,

$$U' \left[c_t(z^t) \right] = \sum_{z_{t+1} \in \mathcal{Z}} \beta \pi \left[(z_{t+1}, z^t) | z^t \right] U' \left[c_{t+1}(z_{t+1}, z^t) \right] \Big\{ z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta) \Big\}.$$

Readers may compare this with the deterministic Euler equation and see that the functional form is really identical, except that stochastic Euler equation is written in expectation terms. The interpretation of the stochastic Euler equation is that an optimal solution to the dynamic optimization problem must be balance the inter-temporal trade off. The marginal utility of consuming one more unit of goods in the present state must be equal to the *expected* discounted marginal utility of consumption foregone in the next period, given that saving one more unit of consumption in the present yields $z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta)$ more goods available in a future state z_{t+1}.

Back to the optimization itself, the quantities of interest from this dynamic optimization are the contingent consumption plans and investment plans, $\{c_t(z^t), k_{t+1}(z^t)\}$. Notice that we can use the binding budget constraint and substitute the contingent consumption plans by the contingent investment plans,

$$\begin{aligned} & U' \left[z_t f \left[k_t(z^{t-1}) \right] + (1-\delta) k_t(z^{t-1}) - k_{t+1}(z^t) \right] \\ = & \sum_{z_{t+1} \in \mathcal{Z}} \beta \pi \left[(z_{t+1}, z^t) | z^t \right] U' \left[z_{t+1} f \left[k_{t+1}(z^t) \right] + (1-\delta) k_{t+1}(z^t) - k_{t+2}(z_{t+1}, z^t) \right] \Big\{ z_{t+1} f' \left[k_{t+1}(z^t) \right] + (1-\delta) \Big\} \end{aligned}$$

This is a system of second-order stochastic difference equations, where the unknowns are $\{k_{t+1}(z^t)\}$, $\forall t, \forall z^t$. The functional form of $U[\cdot]$ and $f(\cdot)$ would often make it nonlinear. The fact it is both stochastic and nonlinear may make it difficult to solve. The conventional approach to solve this system of equations is to log-linearize (first order Taylor expansion) around the deterministic steady state. As this is rather tedious and mechanical, interested readers may refer to page 100-104 in Krussell (2014).

4. Recursive Formulation under Stochastic Environments

For the recursive formulation of the problem, we first define the value of the optimal program starting from period t, given a history z^t as,

$$v(k_t, z^t) \equiv \max_{\{c_s(z^s), k_{s+1}(z^s)\}_{s=t}^{\infty}} \mathbb{E}\left[\sum_{s=t}^{\infty} \beta^{s-t} U\left[c_s(z^s)\right] \ \mid z^t\right]$$

$$s.t. \quad c_s(z^s) + k_{s+1}(z^s) \leq f\left[k_s(z^{s-1})\right] + (1-\delta)k_s(z^{s-1})$$

Notice that the value is now expressed in conditional expectations because we are given with a particular history z^t. This is equivalent to

$$\begin{aligned} v(k_t, z^t) &= \max_{\{c_s(z^s), k_{s+1}(z^s)\}_{s=t}^{\infty}} \sum_{s=t}^{\infty} \beta^{s-t} \pi(z^s|z^t) U\left[c_s(z^s)\right] \\ &= \max_{\{c_s(z^s), k_{s+1}(z^s)\}_{s=t}^{\infty}} \left\{ U\left[c_t(z^t)\right] + \sum_{s=t+1}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-t} \pi(z^s|z^t) U\left[c_s(z^s)\right] \right\} \end{aligned}$$

$$s.t. \quad c_s(z^s) + k_{s+1}(z^s) \leq f\left[k_s(z^{s-1})\right] + (1-\delta)k_s(z^{s-1})$$

We can invoke the first-order Markov process assumption that $\pi\left[(z_{t+1}, z^t)|z^t\right] = \pi\left[(z_{t+1}, z_t)|z_t\right] = \pi\left[z_{t+1}|z_t\right]$ and write the value as,

$$\begin{aligned} v(k_t, z_t) &= \max_{\{c_s(z^s), k_{s+1}(z^s)\}_{s=t}^{\infty}} \left\{ U\left[c_t(z_t)\right] + \sum_{s=t+1}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-t} \pi(z^s|z_t) U\left[c_s(z^s)\right] \right\} \\ &= \max_{c_t(z_t), k_{t+1}(z_t)} \left\{ U\left[c_t(z_t)\right] + \max_{\{c_s(z^s), k_{s+1}(z^s)\}_{s=t+1}^{\infty}} \sum_{s=t+1}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-t} \pi(z^s|z_t) U\left[c_s(z^s)\right] \right\} \end{aligned}$$

The second term on the RHS can be further written as

$$
\begin{aligned}
&\max_{\{c_s(z^s),k_{s+1}(z^s)\}_{s=t+1}^{\infty}} \beta \sum_{s=t+1}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z_t) U\left[c_s(z^s)\right] \\
=&\max_{\{c_s(z^s),k_{s+1}(z^s)\}_{s=t+1}^{\infty}} \beta \left\{ \sum_{z^{t+1} \in \mathcal{Z}^{t+1}} \pi(z^{t+1}|z_t) U\left[c_{t+1}(z^{t+1})\right] + \sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z_t) U\left[c_s(z^s)\right] \right\} \\
=&\max_{\{c_s(z^s),k_{s+1}(z^s)\}_{s=t+1}^{\infty}} \beta \left\{ \sum_{z_{t+1} \in \mathcal{Z}} \pi(z_{t+1}|z_t) U\left[c_{t+1}(z_{t+1})\right] + \sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z_t) U\left[c_s(z^s)\right] \right\} \\
=&\max_{\{c_s(z^s),k_{s+1}(z^s)\}_{s=t+1}^{\infty}} \beta \left\{ \sum_{z_{t+1} \in \mathcal{Z}} \pi(z_{t+1}|z_t) \left[U\left[c_{t+1}(z_{t+1})\right] + \sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z^{t+1}) U\left[c_s(z^s)\right] \right] \right\} \\
\equiv&\beta \sum_{z_{t+1} \in \mathcal{Z}} \pi(z_{t+1}|z_t) v(k_{t+1}, z_{t+1})
\end{aligned}
$$

where the second last equality is true because by the property of first-order Markov processes, for any $s \geq t+2$, $\pi(z^s|z_t) = \pi(z^s|z^t) = \pi(z^s|z^{t+1}) \sum_{z^{t+1} \in \mathcal{Z}^{t+1}} \pi(z^{t+1}|z^t)$; hence,

$$
\begin{aligned}
&\sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z_t) U\left[c_s(z^s)\right] \\
=&\sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z^{t+1}) \sum_{z^{t+1} \in \mathcal{Z}^{t+1}} \pi(z^{t+1}|z^t) U\left[c_s(z^s)\right] \\
=&\sum_{z^{t+1} \in \mathcal{Z}^{t+1}} \pi(z^{t+1}|z^t) \sum_{s=t+2}^{\infty} \sum_{z^s \in \mathcal{Z}^s} \beta^{s-(t+1)} \pi(z^s|z^{t+1}) U\left[c_s(z^s)\right].
\end{aligned}
$$

Given these derivations, the stochastic Bellman equation is then

$$
v(k_t, z_t) = \max_{c_t(z_t),k_{t+1}(z_t)} \left\{ U\left[c_t(z_t)\right] + \beta \sum_{z_{t+1} \in \mathcal{Z}} \pi(z_{t+1}|z_t) v(k_{t+1}, z_{t+1}) \right\}
$$

$$
\text{s.t.} \quad c_s(z_s) + k_{s+1}(z_s) \leq f\left[k_s(z_{s-1})\right] + (1-\delta) k_s(z_{s-1})
$$

Without loss of generality, we can use "prime" to denote the value in next period and rewrite the Bellman equation as follows,

$$
v(k, z) = \max_{c,k'} \left\{ U(c) + \beta \sum_{z'} \pi(z'|z) v(k', z') \right\}
$$

$$
\text{s.t.} \quad c + k' \leq f(k) + (1-\delta) k
$$

Stochastic Bellman Equation without Markov

So far we have exploited the Markov properties in order to derive at the stochastic Bellman equation. In this subsection, I will present a more general derivation that exploits the law of iterated expectation in order to show the stochastic Bellman equation.

Recall that we can define the value as (I omitted the constraints),

$$v(k_t, z_t) = \max_{c_s, k_{s+1}} \mathbb{E}_t \left[\sum_{s=t}^{\infty} \beta^{s-t} U(c_s) \right]$$

Notice that by having a subscript t for the expectation operation, we move all the z^t notations to the subscript which is the information set available at time t. Expanding the summation for the initial period, we arrive at

$$v(k_t, z_t) = \max_{c_s, k_{s+1}} \mathbb{E}_t \left[U(c_t) + \sum_{s=t+1}^{\infty} \beta^{s-t} U(c_s) \right]$$

By the law of iterated expectation,

$$\mathbb{E}_t[x] \equiv \mathbb{E}(x|I_t) = \mathbb{E}(\mathbb{E}(x|I_{t+1})|I_t) \equiv \mathbb{E}_t(E_{t+1}(x)),$$

where the equality is true by LIE because we know that the information set at time t is a subset of the information set at time $t+1$, i.e., $I_t \subseteq I_2$.[2] Hence, we have

$$v(k_t, z_t) = \max_{c_s, k_{s+1}} \mathbb{E}_t \left[U(c_t) + \mathbb{E}_{t+1} \sum_{s=t+1}^{\infty} \beta^{s-t} U(c_s) \right].$$

This can be further written as

$$\begin{aligned}
v(k_t, z_t) &= \max_{c_s, k_{s+1}} \mathbb{E}_t \left[U(c_t) + \mathbb{E}_{t+1} \left(\beta U(c_{t+1}) + \beta \sum_{s=t+2}^{\infty} \beta^{s-(t+1)} U(c_s) \right) \right] \\
&= \max_{c_s, k_{s+1}} \mathbb{E}_t \left[U(c_t) + \beta \mathbb{E}_{t+1} \left(U(c_{t+1}) + \sum_{s=t+2}^{\infty} \beta^{s-(t+1)} U(c_s) \right) \right] \\
&= \max_{c_t, k_{t+1}} \mathbb{E}_t \left[U(c_t) + \beta \max_{c_s, k_{s+1}} \mathbb{E}_{t+1} \left(U(c_{t+1}) + \sum_{s=t+2}^{\infty} \beta^{s-(t+1)} U(c_s) \right) \right] \\
&= \max_{c_t, k_{t+1}} \mathbb{E}_t \left[U(c_t) + \beta v(k_{t+1}, z_{t+1}) \right] \\
&= \max_{c_t, k_{t+1}} U(c_t) + \beta \mathbb{E}_t v(k_{t+1}, z_{t+1})
\end{aligned}$$

The resultant equation will be general form for stochastic Bellman equation.

[2]The generalized law of iterated expectation states that if $\mathcal{G}_1 \subseteq \mathcal{G}_2$, then $\mathbb{E}[\mathbb{E}[X|\mathcal{G}_2]|\mathcal{G}_1] = \mathbb{E}[X|\mathcal{G}_1]$. Readers interested in a rigorous proof may read page 208 of Stokey et al. (1989).

5. Solving Stochastic Dynamic Programs

Given the stochastic Bellman equation, the next step is to study how we can actually solve for the optimal policies. The strategy is much similar to the deterministic case, in that we aim to solve for the value functions through iteration over Bellman equations. Once we know the value functions, the policies are automatically solved as the maximizer to the Bellman equation.

However, there is another issue that is how we should model stochastic shocks in practice. This depends on the nature of the underlying stochastic processes. For our purpose, suppose that z can take three values, $(1, 2, 3)$. $z_H = 3$ would represent the case that there is a high productivity shock; $z_M = 2$ and $z_L = 1$ would represent cases there are medium and low productivity shock. Further suppose that the transition probabilities are represented by a matrix $\mathbf{P}$ where rows represent the initial states in a period and columns represent the end states. The first column and row represent the high state; the second column and row represent the medium state; and the third represents the low state. Each element of the matrix $\mathbf{P}_{ij}$ is then the probability of transitioning from state i to state j. For example, $\mathbf{P}_{11} = 0.7$ means that the probability of starting from a high state and ending also in the high state is 0.7. In terms of the notations of our Bellman equation, this is just $\mathbf{P}_{11} = \pi(z_H|z_H) = 0.7$. Notice that this matrix has the property that each state tends to perpetuate in itself. But there are also some small chances for state switching.

$$\mathbf{P} = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.2 & 0.7 \end{pmatrix}$$

Value function iteration under this stochastic environment works as follows. We should start with a guess of value function $v^0(k, z)$ for *each combination* of (k, z). Given this guess, we can use the Bellman equation and the transition probabilities to find $v^1(k, z)$.

$$v^1(k, z) = \max_{c,k'} \left\{ U(c) + \beta \sum_{z'} \pi(z'|z) v^0(k', z') \right\}$$

$$\text{s.t.} \quad c + k' \leq f(k) + (1 - \delta)k$$

Iterating this logic recursively, by Contraction Mapping Theorem, we will find a fixed point where the value function converges under some regularity conditions. The following pseudo-code summarizes the algorithm.

Algorithm 1: Value Function Iteration – Stochastic Environment

```
1 Specify a step size $\Delta k$.
2 Given that $k$ falls in the range of $[\underline{k}, \bar{k}]$, set up a vector/grid consists of $[\underline{k} : \Delta k : \bar{k}]$.
3 Guess an initial $v^0(k, z)$, e.g., $v^0(k, z) = 0$ for all $k \in [\underline{k} : \Delta k : \bar{k}]$ and $z \in [z_H, z_M, z_L]$.
4 Specify a stopping criterion $\epsilon$, set $n = 0$, and set *Error* $> \epsilon$.
5 while *Error* $>= \epsilon$, do
      foreach $k \in [\underline{k} : \Delta k : \bar{k}]$ do
          foreach $z \in [z_H, z_M, z_L]$ do
              Compute $v(k, z) = \max_{c,k'} \left\{ U(c) + \beta \sum_{z'} \pi(z'|z) v(k', z') \right\}$,
              subject to $c + k' \leq f(k) + (1 - \delta)k$.
      *Error* $= \left\| v^{n+1}(k, z) - v^n(k, z) \right\|_1$.
      Update $n = n + 1$.
```

6. Conclusion

In this lecture, we briefly introduce dynamic programming under stochastic environments. Solutions are characterized for both sequential optimization and dynamic programs, with a follow up discussions on how to solve them in practice.

References

Krussell, Per, *Real Macroeconomic Theory*, Draft Textbook, 2014.

Stokey, Nancy L., Robert E. Jr. Lucas, and Edward C. Prescott, *Recursive Methods in Economic Dynamics*, Cambridge, Massachusetts: Harvard University Press, 1989.

Lecture 5: Search and Matching

Xin Yi

1. Introduction

In this lecture, we will be applying dynamic programming to labor market models. In particular, we will go through the basic algebras that will be useful for these models. We will first start with a partial equilibrium model first studied by McCall (1970). Then, we will move on to a more complete equilibrium matching model as presented in Pissarides (1990). However, I will be refrained from discussing the economic intuition behind the models, as this will be the central task in the Macroeconomics 1 class. The materials presented here are meant to lay a foundation, so that you will make more efficient use of your time in term 2. A standard reference to this literature is Rogerson et al. (2005).

2. The McCall-Mortensen Search Model

Consider the problem for an unemployed worker searching for jobs, which I imported from Prof. Jacquet's lecture notes. Each period, the worker will find a job offer with wage $w \in [\underline{w}, \bar{w}]$ drawn from some cumulative distribution $F(w)$. The worker may choose to reject the offer if w falls below his reservation wage $\bar{w}$. In this case, the worker will receive an unemployment benefit b. This timeline of events will repeat in every period until the worker finds a job. The model also does not allow any possibility of quitting or firing after the worker has accepted a job. That is, once he starts working the worker will continue to do so in perpetuity. The problem is then, to solve for the level of reservation wage, w_R. We further assume that the utility of having an income w is simply $u(w) = w$.

First, we should note that the worker in this scenario is making *discrete* choices, i.e., whether he should accept or reject an offer. Second, the worker can only be in two scenarios, that is either he is employed or he is not. Hence, we can write down the value of accepting a job offer w as

$$V(w) = \sum_{t=0}^{\infty} \beta^t w = \frac{w}{1-\beta}$$

and the value of being unemployed as

$$U = b + \beta \mathbb{E} \max \Big\{ U, V(w) \Big\}$$

By the definition of reservation wage w_R, that the worker is indifferent between working and not working, it must be that the values under the two scenarios be equalized,

$$V(w_R) = U$$

Hence, we can expand the value function for U as follows

$$\begin{aligned} \frac{w_R}{1-\beta} = V(w_R) = U &= b + \beta \mathbb{E} \max \Big\{ U, V(w) \Big\} \\ &= b + \beta \mathbb{E} \max \left\{ \frac{w_R}{1-\beta}, \frac{w}{1-\beta} \right\} \\ &= b + \beta \Big\{ \int_{\underline{w}}^{w_R} \frac{w_R}{1-\beta} \, dF(w) + \int_{w_R}^{\bar{w}} \frac{w}{1-\beta} \, dF(w) \Big\} \end{aligned}$$

This equation then characterizes the reservation wage w_R. I wouldn't go further to discuss any economic intuition as it is beyond the scope of this mini-course. For a detailed exposition on this topic, readers may refer to Ljungqvist and Sargent (2012).

Alternative Solution: the Dynamic Program

In this model, it is sufficient to solve for the reservation wage, as this wage level governs all economic behaviors. However, for the sake of technical interest, I will also present a solution, where we do not solve for the reservation wage. Instead, we are interested in solving a decision rule, or a policy function for the individual facing an offer w. Note that such policy function is inherently equivalent to a cutoff criterion. That is, there will be a cutoff wage such that for every w above the cutoff, the individual will accept the offer. This is just the reservation wage we characterized previously. But again for the sake for technicality, we should learn how to solve a dynamic program for such labor market models.

Let's define another type of value function $v(w)$. Previously, we define $V(w)$ as the lifetime utility of a worker who has already *accepted* an offer w. In contrast, we will define $v(w)$ as the value of an individual facing an offer w and he is making a decision on whether he should accept the offer. Under this scenario, if the individual accepts the offer, then his value would be $V(w) = \frac{w}{1-\beta}$. If he does not accept the offer, then his value would be the unemployment benefit for this period, b, and the discounted expected value of a wage offer in the future $\beta\mathbb{E}[v(w')]$,

$$b + \beta \int_{\underline{w}}^{\overline{w}} v(w')\, dF(w').$$

Hence, the Bellman equation of this problem is,

$$v(w) = \max_{\text{accept, reject}} \left\{ \frac{w}{1-\beta}, b + \beta \int_{\underline{w}}^{\overline{w}} v(w')\, dF(w'). \right\}$$

Given the Bellman equation, we should again solve for the value function and hence the associated policy function through value function iteration. The algorithm is essentially identical to the problems we discussed in lecture 2 and 3. You can also find a Julia code solving a similar dynamic program in Sargent and Stachurski (2017).

3. Continuous Time Search

We now move on to study search and matching models in continuous time. The basic strategy for deriving value function in continuous time, is to start with a small time interval Δ in discrete time and takes the limit when the time interval become infinitesimally small. In this sense, we will start with discrete time which we are comfortable with and then approach continuous time in the limit. Before we start to study the algebra in continuous time, it is useful to first study a basic concept called "arrival rate".

Arrival Rate

Let's first think about why we need an arrival rate of job offers. Suppose that we stay in the previous environment and workers will draw a job offer in each period, say the time interval for each period is Δ. Recall that the basic strategy for continuous time is to take the time interval Δ to the limit $\Delta \to 0$. By this logic, in the limit, the individual is constantly drawing an offer at every instant. This is clearly not realistic and it may cause many troubles in our modeling. Hence, we want to have an environment in which the individual will only receive a job offer from time to time, and an "arrival rate" of job offers is called for.

The standard way of modeling arrival rate is to exploit the properties of Poisson distribution. Suppose that instead of drawing a job offer every period, the individual now will draw n (which is random) job offers in each period. This changes everything because n is can be zero, whereas in the previous environment, n is fixed at 1. In particular, we assume that n is random and follows a Poisson distribution. Denote $p(n, \Delta)$ be the probability that an individual receives n offers in an interval of length Δ. Then, by the property of Poisson distribution, we can write the probability in terms of some constant α, the time interval Δ, and the number of offers n.

$$p(n, \Delta) = \frac{(\alpha\Delta)^n e^{-\alpha\Delta}}{n!}.$$

The expected number of job offer the worker will receive in the time interval Δ is

$$\mathbb{E}[N] = \sum_{n=0}^{\infty} np(n,\Delta) = \sum_{n=0}^{\infty} n\frac{(\alpha\Delta)^n e^{-\alpha\Delta}}{n!} = \alpha\Delta e^{-\alpha\Delta}\sum_{n=1}^{\infty}\frac{(\alpha\Delta)^{n-1}}{(n-1)!} = \alpha\Delta e^{-\alpha\Delta}e^{\alpha\Delta} = \alpha\Delta.$$

Given that we have derived $\alpha\Delta$ to be the expected number of job offers in time interval Δ, the natural interpretation of α is that it is the arrival rate of job offers. In Prof. Jacquet's lecture slides, you will encounter an alternative derivation where the average time of waiting until the arrival of next offer is shown to be $1/\alpha$ and hence the arrival rate of jobs is α. These two approaches are essentially equivalent.

Continuous-Time Algebra

Suppose now that the discount factor between a time interval Δ is $\beta(\Delta) = 1/(1+r\Delta)$, so that $\beta(\Delta)(1+r\Delta) = 1$. Then the value of unemployed is

$$\begin{aligned}
& U = b\Delta + \frac{1}{1+r\Delta}\left[Q(0,\Delta)U + Q(1,\Delta)\mathbb{E}\max\{V(w),U\} + \sum_{n=2}^{\infty} p(n,\Delta)\mathbb{E}\max\{V(w),U\}\right] \\
\Leftrightarrow \quad & (1+r\Delta)U = b\Delta(1+r\Delta) + Q(0,\Delta)U + Q(1,\Delta)\mathbb{E}\max\{V(w),U\} + \sum_{n=2}^{\infty} p(n,\Delta)\mathbb{E}\max\{V(w),U\} \\
\Leftrightarrow \quad & r\Delta U = b\Delta(1+r\Delta) + [Q(0,\Delta)-1]\,U + Q(1,\Delta)\mathbb{E}\max\{V(w),U\} + \sum_{n=2}^{\infty} p(n,\Delta)\mathbb{E}\max\{V(w),U\} \\
\Leftrightarrow \quad & rU = b(1+r\Delta) + \frac{Q(0,\Delta)-1}{\Delta}U + \frac{Q(1,\Delta)}{\Delta}\mathbb{E}\max\{V(w),U\} + \sum_{n=2}^{\infty}\frac{p(n,\Delta)}{\Delta}\mathbb{E}\max\{V(w),U\}
\end{aligned}$$

Note that as Δ approaches 0, the followings is true,

$$\lim_{\Delta\to 0}\frac{Q(1,\Delta)}{\Delta} = \lim_{\Delta\to 0}\alpha e^{-\alpha\Delta} = \alpha; \qquad \lim_{\Delta\to 0}\frac{p(n,\Delta)}{\Delta} = \lim_{\Delta\to 0}\frac{\alpha^n\Delta^{n-1}e^{-\alpha\Delta}}{n!} = 0, \quad \forall n > 1.$$

Furthermore, since $\lim_{\Delta\to 0} Q(0,\Delta) - 1 = 0$ and $\lim_{\Delta\to 0}\Delta = 0$, by L'Hôpital's rule,

$$\lim_{\Delta\to 0}\frac{Q(0,\Delta)-1}{\Delta} = \lim_{\Delta\to 0}\frac{\frac{d[Q(0,\Delta)-1]}{d\Delta}}{\frac{d\Delta}{d\Delta}} = \lim_{\Delta\to 0}\frac{\frac{d\left[e^{-\alpha\Delta}-1\right]}{d\Delta}}{\frac{d\Delta}{d\Delta}} = \lim_{\Delta\to 0}\frac{-\alpha e^{-\alpha\Delta}}{1} = -\alpha.$$

Hence, the expression for rU is further equivalent to the following as Δ approaches 0,

$$\begin{aligned}
rU &= b - \alpha U + \alpha\mathbb{E}\max\{V(w),U\} = b + \alpha\mathbb{E}\max\{V(w)-U,0\} \\
&= b + \alpha\int_{w_R}^{\bar{w}} V(w) - U \; dF(w),
\end{aligned}$$

where the value of being employed with wage w can be derived as follows,

$$\begin{aligned}
& V(w) = w\Delta + \frac{1}{1+r\Delta}V(w) \\
\Leftrightarrow \quad & (1+r\Delta)V(w) = (1+r\Delta)\Delta w + V(w) \\
\Leftrightarrow \quad & rV(w) = (1+r\Delta)w \\
\Leftrightarrow \quad & rV(w) = w, \quad \text{as } \Delta \text{ approaches } 0.
\end{aligned}$$

Hence, we have the continuous-time version of values of being employed and being unemployed, or the continuous-time Bellman equations,

$$\begin{aligned}
& rU = b + \alpha \int_{w_R}^{\bar{w}} V(w) - U \; dF(w) \\
& rV(w) = w.
\end{aligned}$$

To characterize the reservation wage w_R, we need only follow the same strategy under discrete time and invoke the condition that $w_R/r = V(w_R) = U$. Substituting this expression into the value of unemployed yields the following analytical expression that characterizes w_R,

$$w_R = b + \alpha \int_{w_R}^{\bar{w}} \frac{w - w_R}{r} \; dF(w).$$

4. Basic Diamond-Mortensen-Pissarides

In this section, we will study the basic algebra behind an *equilibrium* labor market model, called the Diamond-Mortensen-Pissarides model (DMP), which has won the triple a shared Nobel Prize. This model starts from the assumption that there are in total mass of v vacant jobs and mass of u unemployed workers in the economy. The key feature of this model is that, at any point in time, there will be a matching between some vacant jobs and unemployed workers, such that the unemployed will find a job and fill the vacant position. In particular, they assume that the number of matching is a function of the number of vacant positions and unemployed workers, $m(u,v)$. $m(u,v)$ is called the matching function. It is assumed to be CRS and increasing & concave in each of the two arguments.

Given this assumption, the probabilities for an unemployed person to find a job and for a vacant position to be filled are

$$p(\theta) = \frac{m(u,v)}{u} = m(1, \frac{v}{u}) \equiv m(1,\theta); \quad q(\theta) = \frac{m(u,v)}{v} = \frac{m(1,v/u)}{v/u} \equiv \frac{m(1,\theta)}{\theta}$$

where $\theta \equiv \frac{v}{u}$ can be interpreted as "market tightness". These probabilities can be interpreted as the parameters to a Poisson process such that the probabilities that a vacant position meets n unemployed workers, or an unemployed worker meets n vacant positions in an interval of length Δ are,

$$P(n,\Delta) = \frac{(p(\theta)\Delta)^n \, e^{-p(\theta)\Delta}}{n!}$$

$$Q(n,\Delta) = \frac{(q(\theta)\Delta)^n \, e^{-p(\theta)\Delta}}{n!}.$$

We should also assume that a firm pay pc units of cost per unit time to maintain the opening of a vacant position and once it is filled the productivity of the filled position (the employed worker) is p. Furthermore, there is an exogenous rate of destruction δ, which is also a parameter to a Poisson process such that the probability of n worker-position pairs being destroyed in an interval of length Δ is $\text{Prob}(n,\Delta) = \frac{(\delta\Delta)^n e^{-\delta\Delta}}{n!}$. Denote the values of an vacant position, a filled position, an unemployed worker, and an employed worker as J, V, U, W respectively. Then, the value functions are

$$\begin{aligned}
rV &= -pc + q(\theta)(J - V)\\
rJ &= p - w - \delta(J - V)\\
rU &= b + p(\theta)(W - U)\\
rW &= w - \delta(W - U).
\end{aligned}$$

We will derive the first continuous-time Bellman function step-by-step. The rest follows a similar procedure. First, we know that the discrete time version of the Bellman equation for the vacant position is,

$$\begin{aligned}
& V = -pc\Delta + \frac{1}{1+r\Delta}\left[Q(0,\Delta)V + Q(1,\Delta)J + \sum_{n=2}^{\infty} Q(n,\Delta)J\right]\\
\Leftrightarrow\quad & (1+r\Delta)V = -pc\Delta(1+r\Delta) + Q(0,\Delta)V + Q(1,\Delta)J + \sum_{n=2}^{\infty} Q(n,\Delta)J\\
\Leftrightarrow\quad & r\Delta V = -pc\Delta(1+r\Delta) + [Q(0,\Delta) - 1]\,V + Q(1,\Delta)J + \sum_{n=2}^{\infty} Q(n,\Delta)J\\
\Leftrightarrow\quad & rV = -pc(1+r\Delta) + \frac{Q(0,\Delta)-1}{\Delta}V + \frac{Q(1,\Delta)}{\Delta}J + \sum_{n=2}^{\infty}\frac{Q(n,\Delta)}{\Delta}J\\
\Leftrightarrow\quad & rV = -pc - q(\theta)V + q(\theta)J, \quad \text{as } \Delta \text{ approaches } 0\\
\Leftrightarrow\quad & rV = -pc + q(\theta)(J - V)
\end{aligned}$$

5. Quickly Writing Continuous-Time Bellman Equations

This section is copied from a recitation note I wrote when I was a TA for the Macroeconomics 1 course. However, only Bellman equations for the simplest continuous-time model has been covered in this math camp. You may want to refer back to this section again after finishing the relevant lectures of the Macroeconomics 1 class.

Up to now we have had the following continuous-time Bellman equations. First, for the basic model, we have

$$\begin{aligned} rV(w) &= w \\ rU &= b + \alpha \int_{w_R}^{\bar{w}} V(w) - U \; dF(w) \end{aligned}$$

For models with exogenous job destruction, the Bellman equations are

$$\begin{aligned} rV(w) &= w + \delta \left[U - V(w) \right] \\ rU &= b + \alpha \int_{w_R}^{\bar{w}} V(w) - U \; dF(w) \end{aligned}$$

I claim that all similar Bellman equations, in the context of our labor market models, can be written in the following way.

$$\text{Flow}(\equiv r \times \text{Value}) = \text{Instantaneous flow} + \text{Hazard rate} \times \text{Capital Change in Values}$$

Hence, for the basic model, the continuous-time equations can be interpreted as

$$\underbrace{r \cdot \underbrace{V(w)}_{\textit{value}}}_{\textit{flow}} = \underbrace{w}_{\textit{instantaneous flow}} + \text{No Change in State}$$

$$r \cdot \underbrace{U}_{\textit{value}} = \underbrace{b}_{\textit{instantaneous flow}} + \underbrace{\alpha}_{\textit{hazard rate}} \cdot \underbrace{\int_{w_R}^{\bar{w}} V(w) - U \; dF(w)}_{\textit{capital change in value when the event occurs}}$$

Similarly for the exogenous job destruction model, we have,

$$\underbrace{r \cdot \underbrace{V(w)}_{\textit{value}}}_{\textit{flow}} = \underbrace{w}_{\textit{instantaneous flow}} + \underbrace{\delta}_{\textit{hazard rate}} \cdot \underbrace{\left[U - V(w) \right]}_{\textit{capital change in value when the event occurs}}$$

The equation for unemployment remains the same. For the models on endogenous job destruction with quits, δ becomes the rate of wage shocks. Hence, the continuous-time equation for the value of employment now becomes

$$rV(w) = \underbrace{w}_{\textit{instantaneous flow}} + \underbrace{\delta}_{\textit{hazard rate}} \cdot \underbrace{\int_{\underline{w}}^{w_R} U - V(w)\ dF(w')}_{\textit{capital change in the event of endogenous quit facing the wage shock } w'}$$
$$+ \underbrace{\delta}_{\textit{hazard rate}} \cdot \underbrace{\int_{w_R}^{\bar{w}} V(w') - V(w)\ dF(w')}_{\textit{capital change in the event of staying on the job despite the wage shock}}$$

The equation for the value of unemployment is more or less the same. And for the models of on-the-job search, α_0 and α_1 are the rate of job offer arrivals for unemployed and employed workers, δ is the rate of exogenous destruction. Hence, the continuous-time equation Bellman equation for employment with current wage w is

$$rV(w) = w + \underbrace{\delta}_{\textit{hazard rate for exogenous job destruction}} \cdot \underbrace{(U - V(w))}_{\textit{capital change in the event of exogenous job destruction}}$$
$$+ \underbrace{\alpha_1}_{\textit{hazard rate for a new job offer}} \cdot \underbrace{\int_{w}^{\bar{w}} V(w') - V(w)\ dF(w')}_{\textit{capital change in the event of a more lucrative job offer}}$$

Again, the equation for the value of unemployment is similar to the previous cases.

References

Ljungqvist, Lars and Thomas J. Sargent, *Recursive Macroeconomic Theory*, Cambridge, Massachusetts: The MIT Press, 2012.

McCall, John J., "Economics of Information and Job Search," *Quarterly Journal of Economics*, 1970, *84* (1), 113–126.

Pissarides, Christopher A., *Equilibriu Unemployment Theory*, Cambridge, U.K.: Basil Blackwell, 1990.

Rogerson, Richard, Robert Shimer, and Randall Wright, "Search-Theoretic Models of the Labor Market: A Survey," *Journal of Economic Literature*, 2005, *43* (4), 959–988.

Sargent, Thomas J. and John Stachurski, *Lectures in Quantitative Economics* 2017.