ECON 512: Empirical Methods **Spring 2019**

Lecture 2: Dynamic Programming

Instructor: Paul Grieco *Date: January 7*

These lecture notes are for my personal use. If you are reading them, I decided to distribute them as an experiment. There are typos and probably outright errors, if you find them please accept my apologies and report them to me.

These notes are a review of dynamic programming theory with a focus on how one would numerically solve a dynamic programming problem using value function iteration. Good background material for these notes is:

- Miranda & Fackler, Chapter 7.
- Adda & Cooper, Chapters 1 & 5.
- Stokey & Lucas, Chapter 4.

2.1 Dynamic Programming is Indirect Utility

Before jumping to a dynamic problem, consider a simple household consumption problem:

$$V(I,p) = \max_{c} u(c)$$

$$\text{s.t.: } p \cdot c \leq I$$

Where:

- c is a vector representing the consumption bundle.
- p is a vector of prices.
- I is persons income or budget.
- $u(\cdot)$ is a persons utility function.

The first order condition for every good i is:

$$\frac{u_j(c)}{p_j} = \lambda$$

Where λ is the Lagrange multiplier on the budget constraint.

The key for us is that the function $V(I,p)$ gives us the utility of the consumer of being facing price vector p with income I, and we *don't* need to know exactly how he acts if we know V. For example,

$$V_I(I,p) = \frac{u_j(c)}{p_j} \text{ for all } j$$

Tells us the marginal value of additional income to the consumer, even though it doesn't directly encode consumption functions.

2.2 Direct Attack on a Finite Dynamic Problem

As opposed to the problem of allocating consumption across multiple goods, dynamic programming is concerned with allocating consumption across multiple periods. Consider a "cake-eating" problem:

- Start with a finite time problem $t = 1, \ldots, T$.
- Agent starts with an endowment of cake W_1.
- Cake doesn't go bad, but doesn't regenerate, so law of motion is,

$$W_{t+1} = W_t - c_t$$

- Consider the utility function with discount factor $0 \leq \beta < 1$:

$$\sum_{t=1}^{T} \beta^{(t-1)} u(c_t)$$

We could formulate this problem as a large constrained optimization,

$$V_T(W_1) = \max_{\{c_t\}_1^T, \{W_t\}_2^{T+1}} \sum_{t=1}^{T} \beta^{(t-1)} u(c_t)$$

$$\begin{aligned} \text{s.t.: } W_{t+1} &= W_t - c_t \\ W_t &\geq 0 \\ c_t &\geq 0 \end{aligned}$$

The first thing to notice is that many of the choices are redundant since choosing c_t determines how much cake will be left, therefore re-write constraints:

$$\begin{aligned} \text{s.t.: } \sum_{t=1}^{T} c_t + W_{T+1} &= W_1 \\ W_{T+1} &\geq 0 \end{aligned}$$

So the choice variables are $\{c_t\}_1^T$ and W_{T+1}. So the first order condition for c_t is, where λ is the lagrange multiplier on the consumption constraint is,

$$\beta^{t-1} u'(c_t) = \lambda.$$

The foc with respect to W_{t+1} where ϕ is the Lagrange multiplier is,

$$\lambda = \phi$$

This clearly gives us a terminal condition, since $0 < \lambda = \phi$ it must be that $W_{t+1} = 0$.

So we can solve the problem this way, but notice that it grows in the number of periods we consider. If we want to solve an infinite horizon problem, we'll be stuck.

Moreover, we aren't using the full structure. For example, we can use the consumption focs to generate the Euler Equations:

$$u'(c_t) = \beta u'(c_{t+1})$$

Moreover, once the problem is solved it is easy to calculate the marginal value of additional cake:

$$V_T'(W_1) = \lambda = \beta^{t-1} u'(c_t)$$

Which will hold for all time periods. The insight here is that if we assume the agent is acting optimally, we only need to know V, it doesn't matter when the new cake is eaten.

2.3 Dynamic Programming

The point of dynamic programming is to use the indirect utility function to convert a T period problem into a series of 2-period problems.

2.3.1 Finite Horizon

Suppose we solved our sequence problem (for all W_1), but now we need to add a "period 0". We don't want to re-write a slightly larger problem and solve it, instead, solve:

$$\max_{c_0} u(c_0) + \beta V_T(W_1)$$

$$\text{s.t.: } W_1 = W_0 - c_0$$

Everything we need to know about periods 1 through T is contained in $V_T(\cdot)$ already. Notice that the first order condition for this problem is

$$u'(c_0) = \beta V_T'(W_1)$$

and if we substitute out $V'(\cdot)$ we are right back to the Euler equation.

Of course, in reality we must "build up" V_T though backward induction. However the final period's problem is static and easy to solve, then we can apply dynamic programming for each previous problem.

Notice: there may be a bit of a memory issue since each period has its own value function V_T. This is a direct result of the problem being non-stationary.

2.3.2 Infinite Horizon

The backward induction approach to solving the value function isn't useful if the number of periods is infinite. E.g.,

$$V(W_0) = \max_{\{c_t\}_1^\infty, \{W_t\}_2^\infty} \sum_{t=1}^{\infty} \beta^{(t-1)} u(c_t)$$

$$\text{s.t.: } W_{t+1} = W_t - c_t$$

Because there is no endpoint at which to begin. However, note that this problem is stationary, time itself doesn't matter, just the current state of the world (in our case, how much cake is left).

Our dynamic problem becomes a fixed point problem:

$$V(W) = \max_{c \in [0,W]} u(c) + \beta V(W - c)$$

Equivalently, changing the control variable such that we simply choose the next period's level of cake left, instead of consumption:

$$V(W) = \max_{W' \in [0,W]} u(W - W') + \beta V(W')$$

So we've arrived at the Bellman equation, which defines an operator that maps from a space of functions into itself. That is,

$$B(F)(W) = \max_{W' \in [0,W]} u(W - W') + \beta F(W')$$

And our value function is the fixed point of this operator, $B(V) = V$. Under Blackwell's sufficient conditions, (monotonicity and discounting) we know that such a fixed point exists and is unique.

Even better, we know that by repeatedly applying the operator, we will eventually converge to V. Now let's turn to operationalizing this on a computer.

2.4 A Simple Deterministic Growth Model

We'll work through coding up a simple deterministic growth model where an agent can save capital.

- An agent has period utility $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$, or $\log(c)$ if $\sigma = 1$. Lifetime utility is,

$$U(\{c_t\}) = \sum_{t=0}^{\infty} \beta^t u(c_t)$$

- On the production side, labor is inelasticity supplied. Agent has initial capital endowment k_0 and production function:

$$y_t = f(k_t) = k_t^{\alpha}$$

  where we assume $\alpha < 1$.

- Agent's key decision is how much to spend and how much to invest, this leads to the budget constraint:

$$y_t \geq c_t + i_t$$

- Reason to invest is to have more capital in future years, there may also be depreciation of capital, so the law of motion for capital is,

$$k_{t+1} = (1 - \delta)k_t + i_t$$

This is a stationary infinite horizon problem (nothing depends on t if we know k). Also, it is clear that the budget constraint should bind. Therefore we can write the Bellman equation as

$$V(k) = \max_{k'} u(\underbrace{f(k) + (1 - \delta)k - k'}_{c_t}) + \beta V(k')$$

We want to solve the full model, but first it's useful to note that it has a steady state:

- The first order condition is,

$$-u'(f(k)+(1-\delta)k-k')+\beta V'(k')=0$$

- By the envelope theorem,

$$V'(k)=u'(c)(f'(k)+1-\delta)$$

- So we find the Euler condition,

$$u'(c)=\beta u'(c')(f'(k')+1-\delta)$$

- In steady state, $c=c'=c^*$, so

$$1=\beta(f'(k^*)+1-\delta)$$

  Given our form on f,

$$k^*=\left(\frac{1}{\beta\alpha}-\frac{1-\delta}{\alpha}\right)^{\frac{1}{\alpha-1}}$$

You won't always be able to find the steady state of the problem you want to solve analytically (suppose there are shocks) in this case you will need to find a steady state of an approximate problem, and use a wider state space for value function iteration.

2.4.1 Discretizing the State Space

We allow the state space to be discretized so that agents must choose from a finite set of next period capital. The easiest way to do this is to choose capital levels on an equally spaced grid around the steady state:

```
Klo=Kstar*.9;
Khi=Kstar*1.1;
step=(Khi-Klo)/N;
K=Klo:step:Khi;
```

Let $k^{(j)}$ represent the capital value of the jth gridpoint.

2.4.2 Setting up Period Utility Matrix and Initial Guess

Notice that by discretizing the state space we've limited the agent's possible actions to a finite set. We can therefore represent an agents consumption options in a given state as a vector, and the full set of consumption options in all states as a matrix:

$$C(i,j)={k^{(j)}}^{\alpha}+(1-\delta)k^{(j)}-k^{(i)}$$

This $K\times K$ matrix gives consumption if you are in state $k^{(j)}$ and consume such that next period you will be in state $k^{(i)}$.

This immediately maps into period utility,

$$U(i,j)=\frac{C(i,j)^{1-\sigma}}{1-\sigma}$$

The only other thing we need is an initial guess of the value function. A convenient feasible guess is that the agent consumes its entire capital stock:

$$V^0(j) = u(k^{(j)^\alpha} + (1-\delta)k^{(j)}).$$

This is just an $K \times 1$ vector.

2.4.3 Value Function Iteration Step

Given V^r we can apply the Bellman operator to get V^{r+1}. For each state, we have the maximization problem:

$$V^{r+1}(j) = \max_i \{U(i,j) + \beta V^r(i)\}$$

Since we've discretized, this max is just picking the highest value across the discretized set of next period capital. In MATLAB this might look like:

```
q = repmat(V',1,n);
r = U + beta*q ;
v=max(r);
```

2.4.4 Checking for Convergence

We keep iterating until the change between V^r and V^{r+1} is small enough in some norm. However how small is small enough? The contraction mapping theorem gives us a hint, recall that,

$$||V^* - V^r||_\infty \leq \frac{1}{1-\beta}||V^{r+1} - V^r||_\infty$$

Where V^* is the fixed point and β is the modulus of the contraction mapping (the discount factor for our problem).

This implies if we want accuracy of ϵ^* to the true solution, we need, to iterate such that,

$$(1-\beta)\epsilon^* \leq ||V^{r+1} - V^r||_\infty$$

Since β is typically between .95 and .99, a good rule of thumb is we want to iterate until we have a difference in iterations of 1e-6 in order to ensure our result is within 1e-4 of the true solution.

2.4.5 Policy Function and Transition Path

Since we discretized the problem, the optimal policy is just the best choice in each state using the converged value function. In MATLAB, you can recover the index of the optimal choice using the second argument of `max`. E.g.,

```
[R,m]=max(r);
Kprime=K(m);
```

Once you have the policy function, it is easy to simulate the problem from some initial capital stock level by simply iterating on the policy funciton. This is more exciting when you have some productivity shocks, but we'll see an example in the code.

2.5 Endogenous Labor Choice

Oftentimes, a model will involve a static optimization that must be done numerically. The point of this section is to point out that this can be solved outside the value function iteration loop.

In our example, let's introduce a labor-leisure tradeoff, we can adopt the utility function:

$$u(c, \ell) = (1 - \mu) \log c + \mu \log(1 - \ell)$$

The production function becomes,

$$f(k, \ell) = k^{\alpha} \ell^{1-\alpha}$$

The Bellman equation is now,

$$V(k) = \max_{k', \ell} u(f(k, \ell) + (1 - \delta)k - k', \ell) + \beta V(k')$$

Notice that we have *not* introduced a new state variable. In addition, the first order condition with respect to ℓ will not involve V:

$$u_c(c, \ell) f_l(k, \ell) + u_l(c, \ell) = 0$$

Or given our functional forms,

$$(1 - \mu) \frac{1}{f(k, \ell) + (1 - \delta)k - k'} (1 - \alpha) \left(\frac{k}{\ell} \right)^{\alpha} - \mu \frac{1}{1 - \ell} = 0.$$

This is an easy problem to solve if we know both k and k', so we will solve it for all pairs of current and future capital stock. Define,

$$U(i, j) = \max_{\ell} u(f(k^{(j)}, \ell) + (1 - \delta)k^{(j)} - k^{(i)}, \ell)$$

Which gives us K^2 problems to solve numerically. However, once this is done, we can proceed exactly as we did above (Section 2.4.3).

2.6 Stochastic Growth Model

In this section, we add a persistent productivity shock two our growth model this does two things:

- Expand the state space to multiple dimensions.
- Introduce uncertainty to the agents problem.

2.6.1 Extending the Problem

We'll use our deterministic growth model (without labor choice) EXCEPT production will involve a productivity shock:

$$y_t = A_t f(k_t) = A_t k_t^{\alpha}$$

We assume productivity evolves according to an AR(1) process and is log-normally distributed.

$$A_t = \exp(a_t)$$

$$a_{t+1} = \rho a_t + \varepsilon_{t+1}$$

Where $\rho \in (-1, 1)$ and $\varepsilon \sim N(0, \sigma_\varepsilon)$.

Therefore productivity,

- Affects production today.
- Affects agent's expectations about productivity in the future.

Hence, it is a new state variable. However, we've been working with discrete variables, and an AR(1) process is continuous. Therefore, we will approximate it with Tauchen's method.

2.6.2 Tauchen's Method

The method described here is from Tauchen (1986) and discretizes the AR(1) process into equally spaced gird points. There are other methods that may use grid points more efficiently, but they have the same basic intuition.

We want to approximate the process for a_t, note that this process has unconditional variance,

$$\sigma_a = \frac{\sigma_\varepsilon}{\sqrt{1-\rho^2}}$$

This gives us an idea of how wide we need to make the discrete grid. Suppose we will have N gridpoints $\{a^1, \ldots, a^N\}$. Then define

$$\begin{aligned} a^1 &= -m\sigma_a \\ a^N &= m\sigma_a \\ a^i &= a^1 + \frac{a^N - a^1}{n-1}(i-1) = a^1 + w(i-1) \end{aligned}$$

Where m governs the coverage of the mass of the unconditional distribution of a. We usually use $m = 2$ (96% of mass is inside the grid) or $m = 3$ (99%).

We will approximate the AR(1) with a discrete-state Markov process.

- Suppose in time t the state was on a gridpoint
- To follow the AR(1) you would draw ϵ_{t+1} and go to $rhoa_t + \varepsilon_{t+1}$.

- However, since we need to stay on the grid, we simply remap a_{t+1} to the closest available gridpoint. This leads to:

$$\pi(a^j|a^i) = \begin{cases} \Phi\left(\frac{a^1+\frac{w}{2}-\rho a^i}{\sigma_\varepsilon}\right) & \text{if } a^j = a^1 \\ \Phi\left(\frac{a^j+\frac{w}{2}-\rho a^i}{\sigma_\varepsilon}\right) - \Phi\left(\frac{a^j-\frac{w}{2}-\rho a^i}{\sigma_\varepsilon}\right) & \text{if } a^1 < a^j < a^M \\ 1 - \Phi\left(\frac{a^j-\frac{w}{2}-\rho a^i}{\sigma_\varepsilon}\right) & \text{if } a^j = a^M \end{cases}$$

Note that we now have a discrete state markov chain that approximates the AR(1):

$$\Pi = \begin{bmatrix} \pi(a^1|a^1) & \pi(a^2|a^1) & \cdots & \pi(a^m|a^1) \\ \pi(a^1|a^2) & \pi(a^2|a^2) & \cdots & \pi(a^m|a^2) \\ \vdots & & & \vdots \\ \pi(a^1|a^m) & \pi(a^2|a^m) & \cdots & \pi(a^m|a^m) \end{bmatrix}$$

2.6.3 Stochastic Bellman Equation

Now that we have have discretized capital and the productivity process, we can write the Bellman equation for this problem, where the state is (k, a). Let's cut down on notation and set $\delta = 1$ (easy to add back).

$$V(k, a) = \max_{k'} \{u(e^a k^\alpha - k') + \beta E[V(k', a')|a']\}$$

Complications:

- The state space is now $K \times A$. Since the number of states is discrete, this just means more memory, but you need to decide how you want to do bookeeping. It will conveinent to store V as a $K \times A$ matrix.
- Firms have expectations over what future state you will be in next period. Need to integrate over transition probabilities.

$$E[V(k', a')|a] = \sum_{a' \in A} \pi(a'|a) V(k', a')$$

- Using matrix notation we can write EV as a $K \times A$ matrix where the rows are the next period capital stock and the columns are the current productivity:

$$EV = V\Pi'$$

  Notice that EV depends on the current productivity, but not the current capital stock.

2.6.4 Value Function Iteration

We will loop over k and vectorize a. This is just a programming choice and may not be the fastest way:

1. Given V^{r-1}, a $(K \times A)$ matrix, compute $EV = V^{r-1}\Pi'$.
2. For each $\bar{k} \in K$ (current capital level):

(a) Construct matrix of possible flow utilities over possible next period capital states (rows) and current productivity states (columns).

$$U = \begin{bmatrix} u(a^1\bar{k}^\alpha - k^1) & u(a^2\bar{k}^\alpha - k^1) & \cdots & u(a^m\bar{k}^\alpha - k^1) \\ u(a^1\bar{k}^\alpha - k^2) & u(a^2\bar{k}^\alpha - k^2) & \cdots & u(a^m\bar{k}^\alpha - k^2) \\ \vdots & & & \vdots \\ u(a^1\bar{k}^\alpha - k^K) & u(a^2\bar{k}^\alpha - k^K) & \cdots & u(a^m\bar{k}^\alpha - k^K) \end{bmatrix}$$

(b) The $\bar{k}$ row of the new value function maximizes $U + \beta EV$ columnwise. Using MATLAB-like notation:

$$V^r(\bar{k}, :) = \max_{k'}\{U + \beta EV\}$$

3. Once V^r is fully constructed, check for convergence as before.

2.6.5 Stationary Distribution of Capital

The model gives rise to a decision rule, what firms should do in every state, $k'(k, a)$, but the transition for the model is stochastic. Let $s = (k, a)$ the state transtion matrix can be written

$$Pr(s'|s) = \pi(a'|a)\mathbf{1}[k' = k'(a, s)]$$

Call this matrix P to find the stationary distribution of states we need to solve:

$$\rho P = \rho$$

We can either solve this linear system (using the condition that $\sum \rho_s = 1$). But often it's easier to just iterate it, as we know for any distribution of states q,

$$\lim_{n\to\infty} qP^n = \rho$$

This is what we do in the code. Once we have the stationary distribution of all states, the stationary distribution of capital is just integrates ρ over productivity stats:

$$Pr(k) = \sum_a \rho(k, a)$$