Lecture 2: Dynamic Programming

Antoine Chapel

Today II: Dynamic Programming

- Introduction to dynamic programming
- The Finite Horizon framework
- The Stationary framework
- Value function iteration

Dynamic Programming: Introduction

Our basic problem is still an optimization problem, but it has a particular structure:

$$\begin{aligned} \max_{\{u_t\}_{t=0}^{T} \in U} \quad & \sum_{t=0}^{T} f(x_t, u) \\ \text{s.t.} \quad & x_{t+1} = g(x_t, u), \\ & x_0 \text{ given} \end{aligned}$$

- x_t is the **state** variable
- u_t is the **control** variable
- f is the **reward** function
- g is the **transition** function
- T is the time horizon. It may be finite or infinite.

Dynamic Programming: Introduction

Dynamic consumption problem: the ”Hello World” of dynamic programming for economists:

$$\begin{aligned} \max_{\{u_t\}_{t=0}^T \in U} \quad & \sum_{t=0}^{T} u(c_t) \\ \text{s.t.} \quad & x_{t+1} = \rho(1-u_t)x_t, \\ & c_t = u_t x_t, \\ & x_0 \text{ given} \end{aligned}$$

Dynamic Programming: backwards induction

- A necessary abstraction: the value function: $V(x_t)$
- At $t = s$, if you know already what the optimal behaviour u_t^* will be, you can define the value function for tomorrow as $V(x_{s+1}) = \sum_{t=s+1}^{T} f(x_t, u_t^*)$
- Since you know the impact that your action today will have on your state tomorrow (transition function), you must simply find u_s^* to maximize:

$$f(x_s, u_s) + V(g(x_s, u_s))$$

- Finally, you get your value function for today as:

$$V(x_s) = \max_{u_s \in U} \left[f(x_s, u_s) + V(x_{s+1}) \right]$$

Dynamic Programming: backwards induction

Now let us focus on the case with a finite horizon, that is there is a limited amount of periods T.

- At $t = T$, the agent simply solves $u_T^* = \arg\max_{u_T \imath U} \left[f(x_T, u_T) \right]$. There is no tomorrow, hence no value function for future state.
- So, by looking at the last period, you get $V(x_T) = \max_{u_T \imath U} \left[f(x_T, u_T) \right]$
- Then, you look at $t = T - 1$ and solve:

$$u_{T-1}^* = \arg \max_{u_{T-1} \in U} \left[f(x_{T-1}, u_{T-1}) + V(x_T) \right]$$

- This process, repeated until $t = 0$, is called backwards induction

Dynamic Programming: backwards induction

Let us look at an example, taken from Sydsaeter et al:

$$
\begin{aligned}
\max_{u} \quad & \sum_{t=0}^{2} (1 + x_t - u_t^2) \\
\text{s.t.} \quad & x_{t+1} = x_t + u_t, \\
& x_0 = 0, \\
& u_t \in \mathbf{R}
\end{aligned}
$$

Dynamic Programming with an infinite horizon

- The infinite horizon optimization problem:

$$\begin{aligned} \max_{\{u_t\}_{t=0}^{\infty} \in U} \quad & \sum_{t=0}^{\infty} \beta^t f(x_t, u_t) \\ \text{s.t.} \quad & x_{t+1} = g(x_t, u_t), \\ & x_0 \text{ given} \end{aligned}$$

- The conceptual jump from finite to infinite horizon.
- Goal: find the optimal policy function h: $u_t^* = h(x_t) \ \forall t$

Dynamic Programming with infinite horizon

Like before, let us denote the value function as ”how good is it to be in state x at time s:

$$V(x_s) = \max \sum_{t=s}^{\infty} \beta^t f(x_s, u_s)$$

$$\begin{aligned} V(x_0) &= \max \sum_{t=0}^{\infty} \beta^t f(x_t, u_t) \\ &= \max \beta^0 f(x_0, u_0) + \sum_{t=1}^{\infty} \beta^t f(x_t, u_t) \\ &= \max f(x_0, u_0) + \beta \sum_{t=1}^{\infty} \beta^{t-1} f(x_t, u_t) \\ &= \max f(x_0, u_0) + \beta V(x_1) \end{aligned}$$

Dynamic Programming with infinite horizon

Since the problem is stationary, the problem posed at $t = 0$ is exactly identical to the one posed at any t. Therefore, we write a fundamental object: the **Bellman equation**:

$$V(x) = \max f(x, u) + \beta V(x')$$

Solving this problem will require an algorithm called value function iteration.

Dynamic Programming: a consumption problem

We will present our algorithm with an example in mind: dynamic consumption.

$$\begin{aligned} \max_{\{c_t\}_{t=0}^{\infty}} \quad & \sum_{t=0}^{\infty} \beta^t \log(c_t) \\ \text{s.t.} \quad & k_{t+1} + c_t = Ak_t^{\alpha}, \\ & k_0 \ \text{ given} \end{aligned}$$

- At each t, the agent has some capital k_t, which she invests in a production machine that returns Ak_t^{α}
- That amount is shared between consumption c_t and what will be the capital at next period k_{t+1}.
- $\log(c_t)$ is our utility function

Dynamic Programming: A consumption problem

We write the Bellman equation, replacing c by $Ak^\alpha - k'$ and recognizing that the problem is stationary

$$V(k) = \max_{\{0<k'<A(k^\alpha)\}_{t=0}^{\infty}} \{\log(Ak^\alpha - k') + \beta V(k')\}$$

We see that our ”state” variable is k, and our ”control” variable has become k'.

Value Function Iteration: text description

- Discretize the state space: k_{grid}
- Initialize a first random guess of the value function by fixing $V^0(k')$ to a vector of ones
- Compute a matrix U, whose $(i,j)^{th}$ entry is $\log(Ak^{\alpha} - k') + \beta V(k')$, where k is the i^{th} element of k_{grid} and k' is the j^{th} element of k_{grid}
- Replace the n^{th} entry of $V(k')$ by the maximum value on row n of matrix U. That is your new $V^1(k')$.

Value Function Iteration: matrix description

Suppose you discretized your state space as a vector $[k^1, ..., k^m]$, for example $[1, ..., 35]$

At iteration j of the algorithm[1]:

$$\begin{bmatrix} V^{j+1}(k_1) \\ ... \\ V^{j+1}(k_m) \end{bmatrix} =$$

$$\begin{bmatrix} \max_{k'} \\ ... \\ \max_{k'} \end{bmatrix} \begin{bmatrix} \{ \log(Ak_1^\alpha - k'_1) + \beta V^j(k'_1), & ..., & \log(Ak_1^\alpha - k'_m) + \beta V^j(k'_m) \} \\ ... & ... & ... \\ \{ \log(Ak_m^\alpha - k'_1) + \beta V^j(k'_1), & ..., & \log(Ak_m^\alpha - k'_m) + \beta V^j(k'_m) \} \end{bmatrix}$$

[1] This matrix visualization is borrowed from Diego de Sousa, PhD candidate at Sciences Po

Value Function Iteration

- As you iterate, the V^{j+1} vector represents less and less of an update relative to the V^j vector.
- At each iteration, you should compute the norm:

$$norm^j = \max |V^{j+1} - V^j|$$

- If the norm is lower than some convergence criterion, you stop the iteration, and retrieve the policy function: $h(k) = k'^* = \arg\max_{k'} U$
- Let us look at Python code to make all of this clearer