Discrete State Dynamic Programming and Dynamic Games

Kenneth L. Judd, Hoover Institution

March 19, 2020

Discrete State Space Problems

- Special structure
- Illustrate basic algorthmic ideas

Definition

- State space $X = \{x_i,\ i = 1, \cdots, n\}$
- Controls $\mathcal{D} = \{u_i | i = 1, ..., m\}$
- $q_{ij}^t(u) = \Pr\left(x_{t+1} = x_j | x_t = x_i, u_t = u\right)$
- $Q^t(u) = \left(q_{ij}^t(u)\right)_{i,j}$: Markov transition matrix at t if $u_t = u$.

Value Function: Definition and Algorithm

- Terminal value:

$$V_i^{T+1} = W(x_i), \ \ i = 1, \cdots, n.$$

- Bellman equation: time t value function is

$$V_i^t = \max_u \, [\pi(x_i, u, t) + \beta \sum_{j=1}^{n} q_{ij}^t(u) \, V_j^{t+1}], \ \ i = 1, \cdots, n$$

- Bellman equation can be directly computed.
  - Called *value function iteration*
  - It is only choice for finite-horizon problems because each period has a different value function.

Infinite Horizon Problems

- Infinite-horizon problems
- Bellman equation is now a simultaneous set of equations for V_i values:

$$V_i = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j \right], \quad i = 1, \cdots, n$$

  - Value function iteration is now

$$V_i^{k+1} = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j^k \right], \quad i = 1, \cdots, n$$

  - Can use value function iteration with arbitrary V_i^0 and iterate $k \to \infty$.
  - Error is given by contraction mapping property:

$$\left\| V^k - V^* \right\| \leq \frac{1}{1-\beta} \left\| V^{k+1} - V^k \right\|$$

Algorithm 12.1: Value Function Iteration Algorithm

Objective: Solve the Bellman equation, (12.3.4).

Step 0: Make initial guess V^0; choose stopping criterion $\epsilon > 0$.

Step 1: For $i = 1, ..., n$, compute
$V_i^{\ell+1} = \max_{u \in D} \; \pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u) V_j^{\ell}$.

Step 2: If $\| V^{\ell+1} - V^{\ell} \| < \epsilon$, then go to step 3; else go to step 1.

Step 3: Compute the final solution, setting
$U^* = \mathcal{U} V^{\ell+1}$,
$P_i^* = \pi(x_i, U_i^*), \qquad i = 1, \cdots, n$,
$V^* = (I - \beta Q^{U^*})^{-1} P^*$,
and STOP.

Output:

Policy Iteration (a.k.a. Howard improvement)

- Value function iteration is a slow process
- Linear convergence at rate β
  - Convergence is particularly slow if β is close to 1.
- Policy iteration is faster
  - Current guess:
$$V_i^k, \; i = 1, \cdots, n.$$
  - Iteration: compute optimal policy today if V^k is value tomorrow:
$$U_i^{k+1} = \arg\max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u) \, V_j^k \right], \; i = 1, \cdots, n,$$
  - Compute the value function if the policy U^{k+1} is used forever, which is solution to the linear system
$$V_i^{k+1} = \pi\left(x_i, U_i^{k+1}\right) + \beta \sum_{j=1}^{n} q_{ij}(U_i^{k+1}) \, V_j^{k+1}, \; i = 1, \cdots, n,$$

- Comments:
- Policy iteration depends on only monotonicity
  - Policy iteration is faster than value function iteration
    - If initial guess is above or below solution then policy iteration is between truth and value function iterate
    - Works well even for β close to 1.

Algorithm 12.2: Policy Function Algorithm

Objective: Solve the Bellman equation, (12.3.4).

Step 0: Choose stopping criterion $\epsilon > 0$.
EITHER make initial guess, V^0, for the value function and go to step 1,
OR make initial guess, U^1, for the policy function and go to step 2.

Step 1: $U^{\ell+1} = \mathcal{U} V^{\ell}$

Step 2: $P_i^{\ell+1} = \pi\left(x_i, U_i^{\ell+1}\right), \qquad i = 1, \cdots, n$

Step 3: $V^{\ell+1} = \left(I - \beta Q^{U^{\ell+1}}\right)^{-1} P^{\ell+1}$

Step 4: If $\| V^{\ell+1} - V^{\ell} \| < \epsilon$, STOP; else go to step 1.

- Modified policy iteration
- If n is large, difficult to solve policy iteration step
  - Alternative approximation: Assume policy $U^{\ell+1}$ is used for k periods:

$$V^{\ell+1} = \sum_{t=0}^{k} \beta^t \left(Q^{U^{\ell+1}}\right)^t P^{\ell+1} + \beta^{k+1} \left(Q^{U^{\ell+1}}\right)^{k+1} V^\ell$$

  - Theorem 4.1 points out that as the policy function gets close to U^*, the linear rate of convergence approaches β^{k+1}. Hence convergence accelerates as the iterates converge.

(Putterman and Shin) The successive iterates of modified policy iteration with k steps, (12.4.1), satisfy the error bound

$$\frac{\left\| V^* - V^{\ell+1} \right\|}{\left\| V^* - V^{\ell} \right\|} \leq \min \left[\beta, \; \frac{\beta(1-\beta^k)}{1-\beta} \parallel U^{\ell} - U^* \parallel + \beta^{k+1} \right]$$

Gaussian acceleration methods for infinite-horizon models

- Key observation: Bellman equation is a simultaneous set of equations

$$V_i = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j \right], \quad i = 1, \cdots, n$$

- Idea: Treat problem as a large system of nonlinear equations
- Value function iteration is the *pre-Gauss-Jacobi* iteration

$$V_i^{k+1} = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j^k \right], \quad i = 1, \cdots, n$$

- True Gauss-Jacobi is

$$V_i^{k+1} = \max_u \left[\frac{\pi(x_i, u) + \beta \sum_{j \neq i} q_{ij}(u)\, V_j^k}{1 - \beta q_{ii}(u)} \right], \quad i = 1, \cdots, n$$

- pre-Gauss-Seidel iteration
  - Value function iteration is a pre-Gauss-Jacobi scheme.
  - Gauss-Seidel alternatives use new information immediately
    - Suppose we have V_i^ℓ
    - At each x_i, given $V_j^{\ell+1}$ for $j < i$, compute $V_i^{\ell+1}$ in a pre-Gauss-Seidel

- Gauss-Seidel iteration
- Suppose we have V_i^ℓ
  - If optimal control at state i is u, then Gauss-Seidel iterate would be

$$V_i^{\ell+1} = \pi(x_i, u) + \beta \frac{\sum_{j<i} q_{ij}(u) V_j^{\ell+1} + \sum_{j>i} q_{ij}(u) V_j^\ell}{1 - \beta q_{ii}(u)}$$

  - Gauss-Seidel: At each x_i, given $V_j^{\ell+1}$ for $j < i$, compute $V_i^{\ell+1}$

$$V_i^{\ell+1} = \max_u \frac{\pi(x_i, u) + \beta \sum_{j<i} q_{ij}(u) V_j^{\ell+1} + \beta \sum_{j>i} q_{ij}(u) V_j^\ell}{1 - \beta q_{ii}(u)}$$

  - Iterate this for $i = 1, .., n$
- Gauss-Seidel iteration: better notation
  - No reason to keep track of ℓ, number of iterations
  - At each x_i,

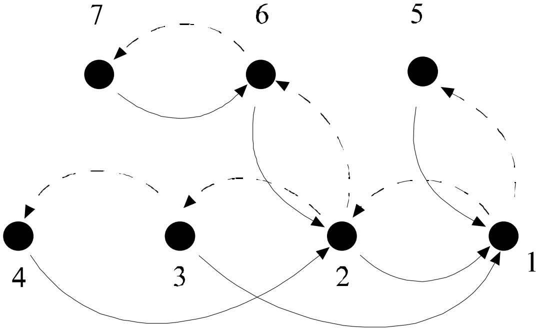
$$V_i \longleftarrow \max_u \frac{\pi(x_i, u) + \beta \sum_{j<i} q_{ij}(u) V_j + \beta \sum_{j>i} q_{ij}(u) V_j}{1 - \beta q_{ij}(u)}$$

  - Iterate this for $i = 1, .., n, 1, ..., $ etc.

State versus Information Flows

Consider the following graph:

- Solid arrows are permissible state transitions
- Broken arrows represent information flow



Upwind Gauss-Seidel

- Gauss-Seidel methods in (12.4.7) and (12.4.8)
- Sensitive to ordering of the states.
  - Need to find good ordering schemes to enhance convergence.
- Example:
  - Two states, x_1 and x_2, and two controls, u_1 and u_2
    - u_i causes state to move to x_i, $i = 1, 2$
    - Payoffs:

$$\begin{aligned} \pi(x_1, u_1) &= -1,\ \pi(x_1, u_2) = 0, \\ \pi(x_2, u_1) &= 0,\ \pi(x_2, u_2) = 1. \end{aligned}$$

    - $\beta = 0.9$.
  - Solution:
    - Optimal policy: always choose u_2, moving to x_2
    - Value function:

$$V(x_1) = 9,\ V(x_2) = 10.$$

    - x_2 is the unique steady state, and is stable

- Converges linearly:

$$
\begin{aligned}
&V^1(x_1) = 0,\ V^1(x_2) = 1,\ U^1(x_1) = 2,\ U^1(x_2) = 2,\\
&V^2(x_1) = 0.9,\ V^2(x_2) = 1.9,\ U^2(x_1) = 2,\ U^2(x_2) = 2,\\
&V^3(x_1) = 1.71,\ V^3(x_2) = 2.71,\ U^3(x_1) = 2,\ U^3(x_2) = 2,
\end{aligned}
$$

- Policy iteration converges after two iterations

$$
\begin{aligned}
&V^1(x_1) = 0,\ V^1(x_2) = 1,\ U^1(x_1) = 2,\ U^1(x_2) = 2,\\
&V^2(x_1) = 9,\ V^2(x_2) = 10,\ U^2(x_1) = 2,\ U^2(x_2) = 2,
\end{aligned}
$$

- Upwind Gauss-Seidel
- Value function at absorbing states is trivial to compute
  - Suppose s is absorbing state with control u
    - $V(s) = \pi(s, u)/(1 - \beta)$.
  - With absorbing state $V(s)$ we compute $V(s')$ of any s' that sends system to s.
$$V(s') = \pi(s', u) + \beta V(s)$$
  - With $V(s')$, we can compute values of states s'' that send system to s'; etc.

Alternative Orderings

It may be difficult to find proper order.

- Alternating Sweep
  - Idea: alternate between two approaches with different directions.

$$
\begin{aligned}
W &= V^k, \\
W_i &= \max_u \ \pi(x_i, u) + \beta \textstyle\sum_{j=1}^n q_{ij}(u) W_j, \ \ i = 1, 2, 3, ..., n \\
W_i &= \max_u \ \pi(x_i, u) + \beta \textstyle\sum_{j=1}^n q_{ij}(u) W_j, \ \ i = n, n-1, ..., 1 \\
V^{k+1} &= W
\end{aligned}
$$

  - Will always work well in one-dimensional problems since state moves either right or left, and alternating sweep will exploit this half of the time.
  - In two dimensions, there may still be a natural ordering to be exploited.
- Simulated Upwind Gauss-Seidel
  - It may be difficult to find proper order in higher dimensions
  - Idea: simulate using latest policy function to find downwind direction
    - Simulate to get an example path, $x_1, x_2, x_3, x_4, ..., x_m$
    - Execute Gauss-Seidel with states $x_m, x_{m-1}, x_{m-2}, ..., x_1$

Discrete-time Dynamic Games

- A discrete-time stochastic game with a finite number of states is often just called a "stochastic game"
  - Ericson-Pakes model of industry dynamics is an example
  - Pakes-Mcguire presents a computational method
- Definition of states and actions
  - State of the game in period t is $\omega_t \in \Omega$; finite number of states
  - N players.
  - Player i's action at t is $x_t^i \in \mathbb{X}^i(\omega_t)$, the set of feasible actions
  - The players' actions in period t is $x_t = \left(x_t^1, \ldots, x_t^N\right)$. As usual, x_t^{-i} denotes $\left(x_t^1, \ldots, x_t^{i-1}, x_t^{i+1}, \ldots, x_t^N\right)$.
- Apologies for change in notation. Here x_t^i denotes actions and ω_t^i denotes states

Dynamics and payoffs

- Dynamics
  - Changes in states are determined by a Markov process
  - Law of motion is

$$\Pr\left(\omega'|\omega_t, x_t\right) = \prod_{i=1}^{N} \Pr^i\left(\left(\omega'\right)^i|\omega_t^i, x_t^i\right),$$

    where $\Pr^i\left(\left(\omega'\right)^i|\omega_t^i, x_t^i\right)$ is the transition probability for player i's state.
- Payoff
  - Player i receives $\pi^i(x_t, \omega_t)$ when players' actions are x_t and the state is ω_t.
  - At the beginning of the next period player i receives a payoff $\Phi^i\left(x_t, \omega_t, \omega_{t+1}\right)$ IF there is a change in the state. For example, I may order a machine to come tomorrow but perhaps it does not.

Nash equilibrium

- Bellman equation for player i is

$$V^i(\omega) = \max_{x^i} \pi^i\left(x^i, X^{-i}(\omega), \omega\right) + \\ \beta \mathsf{E}_{\omega'}\left\{\Phi^i\left(x^i, X^{-i}(\omega), \omega, \omega'\right) + V^i(\omega') \,|\, \omega, x^i, X^{-i}(\omega)\right\}$$

- Player strategy is

$$X^i(\omega) = \arg\max_{x^i} \pi^i\left(x^i, X^{-i}(\omega), \omega\right) + \\ \beta \mathsf{E}_{\omega'}\left\{\Phi^i\left(x^i, X^{-i}(\omega), \omega, \omega'\right) + V^i(\omega') \,|\, \omega, x^i, X^{-i}(\omega)\right\}$$

- Nash equilibrium is a set of Bellman and policy solutions for the set of players

Computational considerations

- Equilibrium is a finite set of equations, each equation being a low-dimensional optimization problem
- LOOKS like dynamic programming but it is not
  - This is not a contraction mapping
  - There may be multiple solutions, in which case this cannot be a contraction mapping
  - Without a contraction factor you cannot use simple stopping rule form DP
- The system is a set of nonlinear equations
  - Can use Gauss-Jacobi, as did Pakes and Mcguire
  - Could use Gauss-Seidel, as later people did (to save memory)
  - Different algorithms may produce different solutions

More Computational considerations

- Parallelization?
  - Much more dangerous, but should be tried
  - Gauss-Jacobi is likely less dangerous
  - Random asynchronous synchronous could be a wild ride
- Crazy idea: Use Newton's method. Only nut cases would try that.
- We will do that next week.