DSC5211C QUANTITATIVE RISK MANAGEMENT

SESSION 7

Stochastic Dynamic Programming
Infinite Horizon

Objectives for Session 7

- Introduction to Markov Decision Processes.
- Computation of *n*-step probabilities and steady state probabilities.
- Introduction to Stochastic Dynamic Programming.
- Solution of MDP:
    - Policy Iteration
    - Value Iteration
    - Linear Programming.
- Estimation of Transition Probabilities in Markov Processes.

Applications of Stochastic Dynamic Programming Infinite Horizon

- Inventory management.
- Wealth management. E.g., sovereign funds.
- Credit risk.
- Modelling commodity prices.
- Health care. Modelling the probability of illness.

Markov Chains

- In this course we are going to consider discrete states and time.
- The first concept presented is the *Markov property* (Howard, 1971). Let

$$P\left(s_{t+1} = s' \mid s_t, s_{t-1}, \ldots, s_0\right),\ \forall s', s_t, s_{t-1}, \ldots, s_0$$

random walk

represent the probability of transition from a state s_t to a state s', where s_i is the state of a dynamic system at time *i*.

- If the state has the Markov Property, it contains all the information to determine the transition probabilities and, in this case, the model transition dynamics is $P\left(s_{t+1} = s' \mid s_t\right)$.

Markov Chains (Cont.)

Let $P(i, j)$ represent the one-step transition probability from state *i* to *j*.

For all *i*:

$$\sum_j P(i, j) = 1$$

Markov Chains – *n*-step transition probabilities

- Given the chain is in state *i* at a given time, what is the probability it will be in state *j* after *n* transitions?

For all *i* and *j*:

$$P^n(i,j) = \sum_z P(z,j) \times P^{n-1}(i,z)$$

- We solve a system of *n* equations.

Markov Chains – Steady-State Probabilities

- Steady-state probabilities $\pi(i)$ represent the limiting probabilities of a given state of the system being visited in the long-run.
- Again, we can define these transition probabilities iteratively as

$$\pi(i) = \sum_j P(j,i) \times \pi(j) \quad .$$

- In this case we have a linear system of with a number of equations equal to the number of states.

Markov Chains – classification of states

- State j is accessible from state i if $P^n(i, j) > 0$ for some n.
- If j is accessible from i, and i is accessible from j, we say that states i and j communicate ($i \leftrightarrow j$).
- State i communicates with itself.
- If i communicates with j then j communicates with i
- If $i \leftrightarrow j$ and $j \leftrightarrow k$, then $i \leftrightarrow k$.
- If all the states communicate, the Markov chain is *irreducible*.

Markov Chains – Recurrence v Transience

- Let f_i be the probability that, starting in state *i*, the process will reenter state *i*. If $f_i = 1$, the state is *recurrent*, otherwise it is *transient*.
    - If state *i* is recurrent then, starting from state *i*, the process will reenter state *i* infinitely often (with prob. 1).
- A special case of a recurrent state is if *the absorbing state*.

Markovian Decision Process

- If every state has the Markov Property, the environment and the task as a whole also have the Markov Property.
- A Markovian decision process is a tuple (*policy*, *reward function*, *value function*, *model of the environment*).
- A *policy* $\pi : A(s) \rightarrow a$ is a rule that transforms states into actions.
- Under a stationary process a Markovian decision Process is a Markov Chain.

Reward Functions

- A *reward function* $u_{ij}^{a_t}$ defines the utility that an agent receives from choosing an action a_t in a certain state *i*, at a given time *t*.
- The ultimate goal is to maximise the total reward received in the long-run, the *Return* (R_t).
- If the horizon is finite with *T* stages, then $R_t = u_{s_t, s_{t+1}}^{a_t} + u_{s_{t+1}, s_{t+2}}^{a_{t+1}} \ldots + u_{s_T, s_{T+1}}^{a_T}$

  .

Reward Functions (Cont.)

- If the time horizon is infinite, the return is the discounted sum (where $0 \leq \gamma \leq 1$ is the *discount* parameter) of each one of the rewards

  $R_t = \sum_{k=0}^{+\infty} \gamma^k u_{t+k+1}$.

- A *value function*, *V(s)*, specifies the value of the state *s*, i.e., the total value of rewards an agent expects to receive until the end of the horizon by entering that state immediately.

- If there is a perfect knowledge of the transition probabilities and rewards the optimal policy is computed by dynamic programming.

Dynamic Programming - Definition

- Bellman (1957) defines dynamic programming as the mathematical theory of multi-stage decision processes.
- Bellman defines a *decision process* as a system (possibly stochastic) in which the decision-maker has a choice of transformations that may be applied to the system at any time.
- He also distinguishes single-stage (only one decision has to be made) from multi-stage decision processes (where the decision-maker has to take a sequence of decisions).

Dynamic Programming - Principle of Optimality

- A decision maker solves a dynamic programming problem when he understands the *structure* of the optimal policy.
- ***Principle of Optimality***: an optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions are an optimal policy with regard to the state resulting from the first decision.

Dynamic Programming – The Problems

- The dynamic programming problem is the one of computing the *optimal policies*, given that the environment follows a Markovian process and a known model of the environment.
- It implies the solution of two related problems:
    - prediction (policy evaluation)
    - control.

Dynamic Programming – The Prediction Problem

- The *prediction* problem consists of estimating the value of each state of the environment for a certain policy π .
- Let E_π represent the expected value of a policy π.
- Let $\pi(s,a)$ represent the probability of executing action *a* in state *s*, by following a policy π .
- Further, let $P_{ss'}^{a}$ represent the probability of the system moving from state s to state s', conditional on the agent choosing action *a*.

Dynamic Programming – The Prediction Problem (Cont.)

- The *state-value function* $V^{\pi}(s)$ represents the expected return of state *s*, following a policy π:

$$V^{\pi}(s) = E_{\pi}[R_t \setminus s_t = s]$$

or equivalently

$$V^{\pi}(s) = u_s^{\pi} + \gamma \sum_{s'} P_{ss'}^{\pi} V^{\pi}(s')$$

variable

Dynamic Programming – Policy Evaluation

- In order to determine the value of a given policy π we solve a system of #S linear equations:

$$V^{\pi}(s) = u_s^{\pi} + \gamma \sum_{s'} P_{ss'}^{\pi} V^{\pi}(s') .$$

- The variables are the $V^{\pi}(s)$ and the solution is straightforward.

Dynamic Programming – The Control Problem

- The control problem takes into account that a given policy influences the value of a given action.
- Let the *action-value function* $Q^{\pi}(s,a)$ represent the expected return of an action *a*, at state *s*, following policy π.
- $Q^{\pi}(s,a) = E_{\pi}[R_t \setminus s_t = s, a_t = a]$ represents the action-value function, which is equivalent to

$$Q^{\pi}(s,a) = u_s^a + \gamma \sum_{s'} P_{ss'}^{a} V^{\pi}(s') .$$

Dynamic Programming – The Control Problem (Cont.)

- It is then intuitive that a value function imposes a partial order on the space of policies.

- A policy π is better than or equal to a policy π' if and only if its expected value is higher than or equal to the expected return of policy π':

$$\pi \succ \pi' \Leftrightarrow V^{\pi}(s) \geq V^{\pi'}(s)\text{, for all } s \in S\,.$$

Dynamic Programming – Optimal Value Functions

- The optimal state-value function $V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$ represents the maximum expected value of a state *s*.

- The optimal action-value function $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in S, a \in A(s)$ represents the maximum expected value of an action *a* in state *s*. An optimal policy π^* gives the association between states of the world and actions that maximise the expected return:

$$\pi^*(s) = \arg\max_{a} \left[u_s^a + \gamma \sum_{s'} P_{ss'}^{a} V^{\pi^*}(s') \right], \ \forall s \in S \ .$$

Dynamic Programming – Solution Methods

- Infinite Horizon:
  - Policy Iteration: The *policy iteration* algorithm is an approximation method, in the space of policies, which converges to the optimal policy, in the limit.
  - Value Iteration: The *value iteration* algorithm is a successive approximation method, in the space of the value functions *V(s)*, which converges to the optimal value function.
  - Linear Programming: we re-write the D.P. problem as a linear program.

D.P. Infinite Horizon – Policy Iteration

- We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*,$$

where $\xrightarrow{E}$denotes a policy *evaluation* and $\xrightarrow{I}$denotes a policy *improvement*.

- Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal).

- Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations.

D.P. Infinite Horizon – Policy Iteration Algorithm

Step 1. For each state *s* choose action *a*(*s*) arbitrarily. Evaluate V(*s*).

Step 2. Policy Evaluation

```
Repeat
        $\Delta \leftarrow 0$
        For each $s \in S$ :
        $v \leftarrow V(s)$
        $V(s) \leftarrow u_s^\pi + \gamma \sum_{s'} P_{ss'}^\pi V^\pi(s')$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
```

(Continued next slide...)

Policy Iteration Algorithm (Cont.)

Step 3. Policy Improvement

```
$Policy\_stable \leftarrow True$
For each $s \in S$ :
        $b \leftarrow \pi(s)$
        $\pi(s) = \arg\max_a \left[ u_s^a + \gamma \sum_{s'} P_{ss'}^a V^\pi(s') \right]$
        If $b \neq \pi(s)$ then $Policy\_stable \leftarrow False$
If *policy-stable*, then stop; else go to Step 2
```

D.P. Infinite Horizon – Value Iteration

- This algorithm is called *value iteration*. It can be written as a particularly simple backup operation that combines the policy-improvement and truncated policy-evaluation steps:

- Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

- This algorithm converges to an optimal policy for discounted finite MDPs

D.P. Infinite Horizon – Value Iteration Algorithm

For each state *s* choose action *a*(*s*) arbitrarily. Evaluate V(*s*).

```
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$: $v \leftarrow V(s)$
            $V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [u_{ss'}^a + \gamma V(s')]$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
```

Output a deterministic policy, π^*, such that:

$$\pi^*(s) = \arg\max_a \left[u_s^a + \gamma \sum_{s'} P_{ss'}^a V^{\pi^*}(s') \right]$$

D.P. Infinite Horizon – Linear Programming

Optimal stationary policy for a maximization problem

$$\pi^*(s) = \arg\max_a \left[u_s^a + \gamma \sum_{s'} P_{ss'}^a V^{\pi^*}(s') \right] \quad \forall s \in S$$

Solution:

$$\min Z = \sum_s V(s)$$

s.t., for each s and *a* in *a*(s):

$$V(s) - \gamma \sum_{s'} P_{ss'}^a V(s') \geq u_s^a$$

All variables are free.

- If a constraint for state *s* and action *a* is binding then action *a* is optimal in state *s*.

D.P. Infinite Horizon – Linear Programming

Optimal stationary policy for a minimization problem

$$\pi^*(s) = \arg\min_a \left[u_s^a + \gamma \sum_{s'} P_{ss'}^a V^{\pi^*}(s') \right] \quad \forall s \in S$$

Solution:

$$\max Z = \sum_s V(s)$$

s.t., for each s and *a* in *a*(s):

$$V(s) - \gamma \sum_{s'} P_{ss'}^a V(s') \le u_s^a$$

All variables are free.

- If a constraint for state *s* and action *a* is binding then action *a* is optimal in state *s*.

Estimating the Transition Probabilities in Markov Processes

- Let p_{ij} stand for the transition probability from state i to state j.

$$p_{ij} = \Pr\left(X_{t+1} = j | X_t = i\right)$$

- We observe a sample from the Markov Chain:

$$x_1^n \equiv x_1, x_2, \ldots x_n$$

- This is a realization of the random variable X_1^n.

Estimating the Transition Probabilities

$$\begin{aligned}\Pr\left(X_1^n = x_1^n\right) &= \Pr\left(X_1 = x_1\right)\prod_{t=2}^{n}\Pr\left(X_t = x_t | X_1^{t-1} = x_1^{t-1}\right)\\ &= \Pr\left(X_1 = x_1\right)\prod_{t=2}^{n}\Pr\left(X_t = x_t | X_{t-1} = x_{t-1}\right)\end{aligned}$$

- The Likelihood of a transition matrix p_{ij}:

$$L(p) = \Pr\left(X_1 = x_1\right)\prod_{t=2}^{n} p_{x_{t-1}x_t}$$

Log-Likelihood Estimation of the Transition Probabilities

- Take the log of the likelihood function:

$$\mathcal{L}(p) = \log L(p) = \log \Pr\left(X_1 = x_1\right) + \sum_{i,j} n_{ij} \log p_{ij}$$

- Setting the partial derivative of the likelihood function to p_{ij} equal to zero:

$$\boxed{\hat{p}_{ij} = \frac{n_{ij}}{\sum_{j=1}^{m} n_{ij}}}$$

Log-Likelihood Estimation of the Transition Probabilities

- We just need to count how many transitions we have observed from state *i* to *j* and calculate the conditional probability.
- As usual, we need to define a time window to compute the transition probabilities representative of the future.
- The lower the number of states considered in the system the more robust these transition probabilities are.
  - There is a trade-off between the quality of the estimated returns and transition probabilities.

Summary

- We have presented Markov Decision Processes as a way of modelling and optimize the actions in inter-temporal problems.

- We have shown how to solve MDP using several possible algorithms:
    - Policy Iteration
    - Value Iteration
    - Linear Programming.

- We have analysed the use of Maximum Likelihood to estimate the transition probabilities in Markov Processes.

References

- Sutton, R. S., Barto, A. G., 1998. Reinforcement Learning: An Introduction. MIT Press.

- Winston, W. 2004. “Operations Research, Applications and Algorithms,” 4th edition, Thomson.