Reinforcement Learning - TP1 Dynamic Programming and Reinforcement Learning

Chia-Man Hung

November 11, 2016

1 THE ONE-SITE TREE CUTTING PROBLEM

Q1. Explicit the MDP and the parameter chosen to model the random effects.
By continuing on the sample code (*mainTP1.m*, *tree_MDP.m*, *simu.m*), we define a *tree_sim.m* which receives a state and an action as input and return the next state and the reward. Since the current state and the action are given as input, we construct an array named dynamics associated to that action as in *tree_MDP*. It is the probability of getting into different states. Based on the dynamics, we draw a sample using *simu*. Then we compute the associated reward.

Q2. Policy evaluation.
We evaluate a given policy using the Monte-Carlo method and matrix inversion.
Monte-Carlo: We simulate n trajectories and compute the average. The simulation is done by *tree_sim*.

$$V_n(x_0) = \frac{1}{n} \sum_{k=1}^{n} \sum_{t=1}^{T_{max}} \gamma^{t-1} R_t^{(k)} \tag{1.1}$$

This is implemented in *monte_carlo_value*.

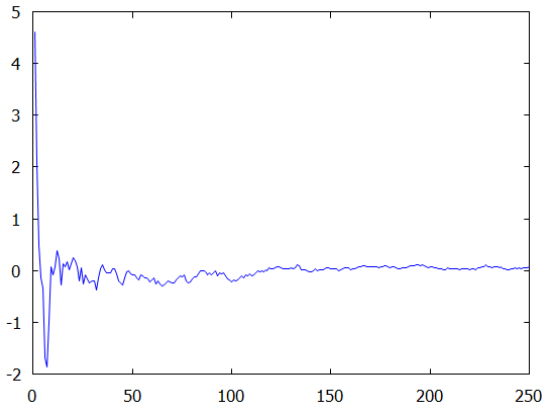
Matrix-inversion:
For any stationary policy π, the state value function at a state $x \in X$ follows the Bellman equation.

$$V^{\pi}(x) = r(x, \pi(x)) + \gamma \sum_{y} p(y|x, a) V^{\pi}(y) \tag{1.2}$$

We denote $V = \begin{pmatrix} V_1 \\ \vdots \\ V_S \end{pmatrix}$, $R = \begin{pmatrix} r(1, \pi(1)) \\ \vdots \\ r(S, \pi(S)) \end{pmatrix}$, $P = \begin{pmatrix} p(1|1, \pi(1)) & \cdots & p(S|1, \pi(1)) \\ \vdots & & \vdots \\ p(1|S, \pi(S)) & \cdots & p(S|S, \pi(S)) \end{pmatrix}$,
where S is the total number of states.

We have $V = R + \gamma PV$, which leads to $V = (I - \gamma P)^{-1} R$.
This is implemented in *bellman_evaluation*. Note that in the implementation, we use the *tree_MDP* to obtain R (reward) and P (dynamics).

Figure 1.1: Policy evaluation: $V_n(x_0) - V^{\pi}(x_0)$. $T_{max} = 1000$, $n_{max} = 250$.



As expected, the difference converges to 0 as n grows.

Q3. Optimal policy.
Value iteration:
The Q-value function associated to the Value function V is defined by

$$Q(x, a) = r(x, a) + \gamma \sum_{y} p(y|x, a) V(y) \tag{1.3}$$

$$Q_{k+1}(x, a) = r(x, a) + \gamma \sum_{y} p(y|x, a) \max_{a' \in A} Q_k(y, a'), k = 1, 2, ..., K. \tag{1.4}$$

This is implemented in *value_iteration*.

Policy iteration:
At each iteration $k = 1, 2, ..., K$, there is a policy evaluation done by matrix inversion and a policy improvement as below.

$$\pi_{k+1}(x) \in \underset{a \in A}{argmax}\{r(x,a) + \gamma \sum_{y} p(y|x,a) V^{\pi_k}(y)\} \tag{1.5}$$

This is implemented in *policy_iteration*.

Q4. Speed of convergence.

The policy iteration method converges with very few number of iterations. In our case, it converges within only two iterations. Compared to the value iteration method, every iteration is more costly, since it requires a policy evaluation, which is done by a matrix inversion in our case.

Figure 1.2: Speed of convergence of the value iteration method: $||V^* - V_k||_\infty$.