Optimization under uncertainty - Homework 1

Thomas Boudier, M2 ORCO

November 12, 2021

1 Properties of Bellman Operators

Let $\mathcal{M} := (\mathcal{S}, A, r, P, \lambda)$ be a MDP. We denote by L_π the linear operator for a static deterministic policy π. For any vector V:

$$L_\pi V := r_\pi + \lambda P_\pi V$$

L is the bellman operator:

$$LV := max_\pi L_\pi V$$

1. Observe L and L_π are monotonous:
Let V, V' such that $V \leq V'$ componentwise. Let's fix some policy π.
As P_π is a stochastic matrix and $\lambda > 0$:

$$V > V' \Leftrightarrow r_\pi + \lambda P_\pi V > r_\pi + \lambda P_\pi V' \Leftrightarrow L_\pi V > L_\pi V'$$

This being true for any fixed policy; it is also true for the best one:

$$V > V' \Leftrightarrow LV > LV'$$

2. L and L_π are homogenous: Let $c \in \mathbb{R}$, and let π be any policy.
Observe that, as P_π is a stochastic matrix, $P_\pi \mathbb{I} = \mathbb{I}$; $\mathbb{I}$ being the unit vector.
Hence:

$$L_\pi(V + c\mathbb{I}) = r_\pi + \lambda P_\pi V + \lambda P_\pi c\mathbb{I} = L_\pi V + \lambda c\mathbb{I}$$

This being true for any fixed policy, it is also true for the best one:

$$L(V + c\mathbb{I}) = LV + \lambda c\mathbb{I}$$

3. It is clear that L, L_π are contracting and increasing. Hence, we have $V \leq V_\pi \Rightarrow V \leq L_\pi V$ and $V \geq V_\pi \Rightarrow V \leq L_\pi V$.
This being true for any policy π, it is in particular true for the best one: $V \leq V^* \Rightarrow V \leq LV$ and $V \geq V^* \Rightarrow V \leq LV$.

2 Multiarmed Bandits

We consider a bandit as a finite set of n arms. Each arm i has S possible states, a reward vector r_i and a transition probability matrix P_i . Here is the evolution of the bandit used by a player:
At each round $t \in \mathbb{N}$, the arms are in states, say $s = (s_1(t), s_2(t), ..., s_n(t))$. The player decides to use one arm among the S possible arms (say arm i). He gets a reward $\lambda^t r_i(s_i(t))$ and arm i moves to a new state s_i' with probability $P_i(s_i'|s_i)$. The other arms stay in their current state. The player wants to maximize the sum of its rewards over an infinite horizon.

1. Let $\mathcal{S} := \{s = (s_{1,j}, .., s_{n,j})|j \in [[1, S]]\}$ be the set of states.
Let $\mathcal{A}_s := \{j|j \in [[1, S]]\}$ be the action set; which do not depends on s. For all arm i, let $(P_i)_{l,c} := P_i(s_l|s_c)$, the probability to pass from state s_c to state s_l when using arm i, and let $r_i(s) = r_i(s_i)$ be the reward vector. If $\lambda \in]0, 1[$; then $(\mathcal{S}, \mathcal{A}_s, P_i, r_i)$ defines a Markov Decision Process discounted by λ.

2. Now let's consider a particular arm i_0. In the next few questions assume the dropping of indexes. Consider a new game where the controller has the choice at each step to stop and earn M[1]; or action the arm, move to a new state according to the probability matrix associated earn his reward [1], and start a new step.
Let $W(s, M)$ be the optimal gain expected to earn over an infinite horizon, starting in state s.
It is clear that it is equal either to M, either to the gain of the arm in state s, plus $W(s', M)$; where s' is a state reached from state s; discounted by λ. In other words:

$$W(s, M) = max(M, r(s) + \lambda \sum_{s'} P(s'|s)W(s', M))$$

We can write the $W(s, M)$ inside a vector $W_M := (W(s, M))_{s\in[[1,S]]}$ and $R := (r(s))_{s\in[[1,S]]}$ so that W_M verifies:

$$W_M = max(M, R + \lambda PW)$$

3. Let $M* := R + \lambda PW$.

- Suppose $M < M*$. Then $W(s, M) = M*$. Consequently, if $M_1 < M_2 < M*$, then $M* = W(s, M_1) \leq W(s, M_2) = M*$.

[1] being discounted by λ, of course.

- If $M* < M$, then $W(s, M) = M$. Consequently, if $M_1 > M_2 > M*$, then $M1 = W(s, M_1) \geq W(s, M_2) = M2$.
- Finally, if $M_1 < M* < M_2$, then $M* = W(s, M_1) \leq W(s, M_2) = M_2$.

We conclude that $W(s, M)$ is increasing in M.

Suppose $M < \frac{r_{min}}{1-\lambda}$, where $r_{min} = min_s(R)$.
Imagine a scenario in which we always use the lever, and always earn the minimal reward of the machine. The total gain over an infinite horizon would be: $\sum_{t=0}^{\infty} \lambda^t r_{min} = \frac{r_{min}}{1-\lambda}$. Logically, if M induce a lower gain than the one in the worst scenario possible, then it is never a good choice to stop to earn M. So we must have $W = R + \lambda PW = LW$ where L is the Bellman operator. Hence W is the fixed point of L.

Suppose $M > \frac{r_{max}}{1-\lambda}$, where $r_{max} = max_s(R)$.
Similarly; the best scenario possible would give us $\frac{r_{max}}{1-\lambda}$; so it is always a better choice to stop and take M. Hence $W(s, M) = M$.

4. Consider $M = \frac{r_min}{1-\lambda}$. In this situation, we already know that $W(s, M) = V(s)$ for all $s \in S$, where V is the fixed point of L; ie V verifies:

$$V = R + \lambda PV \Leftrightarrow V = R(I - \lambda P)^{-1}$$

Imagine M gradually increases up to the moment where we have one state $s*$ for which $W(s*, M_{s*}) = V(s*)$. It is easy to see that $s* = argmin_s(V(s))$ where V is the fixed point of L; hence $M_{s*} = V(s*)$. For any $M \geq M_{s*}$; we have thanks to question ???. It is clear that for any other state $s'*$; the behaviour of $W(s'*, M)$ is identical. Hence, $W(s, M)$ is linear in M and:

$$\forall s \in \mathcal{S}, W(s, M) = max(M, V(s))$$

5. We call the *index policy* the policy $\bar{\pi}$ in which at each step s we action the lever i that have the highest index $I_i(s) := min\{M | W_i(s, M) = M\}$. We want to do a simulation of the problem, in particular we want to compute the value $V^{\bar{\pi}}$ of some instances using the formula:

$$\forall \pi, V^{\pi} = R_{\pi}(I - \lambda P_{\pi})^{-1}$$

Now the vector R_π and the matrix P_π are not trivial as in the previous questions because we are considering all the arms at the same time. To handle the things with more ease, we will introduce a bijection:

$$\begin{aligned} c_P \colon S &\to [[0, S^n - 1]] \\ s &\mapsto p. \end{aligned}$$

We will call c_S it's inverse: $c_S := c_P^{-1}$. We call p the *p-state* corresponding to a state s if and only if $p = c_P(s)$. Conversely s is the *s-state* associated to p.
These two functions use the classical method of conversion from base 10 to base S. The numerical functions are called $conv_P$ and $conv_S$.

This conversion allows us to consider the vector $R_{\bar{\pi}}$ and the matrix $P_{\bar{\pi}}$ as if they were indexed by the p-states. Now we will determine how to fill these. We will start with the matrix.
First, notice it is sparse; the reason is that if you take any two s-states s, s', it is unlikely that it is possible to do from one to the other: it is only possible (whatever the policy is) if they have at most one element of difference. The numerical function that handle this is a boolean one called *diffp*. Note that diffp takes in argument two p-states.
Suppose now we are in a p-state p_1 associated to the s-state s_1. For any p-state p_2 (associated to s_2), we want to determine the value of $(P_{\bar{\pi}})_{p1,p2}$.

- Call *diffp* on $p1, p2$. If it returns False, returns 0.
- Compute which arm has the highest index of every arm on s_1. Say the answer is i. Check that s_1 and s_2 do not differ on any other index than i. If this is false, return 0. Otherwise, return $P_i((s_2)_i|(s_1)_i)$.

Now let's focus on $R_{\bar{\pi}}$. Say we are in p-state p. Compute it's associated s-state s, and the which arm i has the highest index on s. Say the answer is i. Returns $r_i((s)_i)$.

You can verify in the hw.py file that the algorithm is working correctly (according to me). Some parameters may be modified, they are at the beggining of the file. The execution with $S = 3$ and $n = 4$ takes about a second on my computer, but I wouldn't make these too big.

To finish the homework, we simply have to execute the value iteration algorithm on the same instances and check it has the same (or "similar enough"

as VI do not give an exact result) that the value computed. Of course, this is not a proof that the index policy is an optimal policy; it could just be optimal by chance on this particular instance, which is why we want to reiterate the experiment on different instance by modifying the *seed* parameter for instance.