Problem Set 1 (Dynamic Programming)

Due on May 13th. Use your favorite programming language & send a report in pdf, along with the code.

The goal of this homework is to gain some experience with standard dynamic programming methods for solving Markov decision processes with known transition and reward functions. The MDP to study is a simple **controlled queueing system** where the goal is to control the number of requests waiting in a queue. Requests are generated in a stationary stochastic fashion and are being served at a rate controlled by the learning agent. This can be formulated as a Markov decision process as follows:

The state space consists of the integers $0, 1, 2, \ldots, N-1$, with $N = 100$, corresponding to the length of the queue.

The action space consists of two actions: action a_{low} corresponding to a low service rate of $q(a_{\text{low}}) = q_{\text{low}} = 0.51$ and action a_{high} corresponding to a high service rate of $q(a_{\text{high}}) = q_{\text{high}} = 0.6$.

The reward function assigns a reward of $r(x, a) = -(x/N)^2 - c(a)$, with $c(a)$ being the cost of action a. This cost is defined for the two actions as $c(a_{\text{low}}) = 0$ and $c(a_{\text{high}}) = 0.01$.

The transition function describes the dynamics of the queue as follows. The arrival rate $p = 0.5$ models the rate at which new requests arrive into the queue and the controlled service rate models the rate at which requests leave the queue. In each round, the queue length increases by 1 with probability p and decreases by 1 with probability $q(a)$ corresponding to the action a taken. Precisely, letting x_t be the current queue length, a_t be the action taken by the agent, we define $I_t \in \{0, 1\}$ as the increment such that $\mathbb{P}[I_t = 1] = p$ and $S_t \in \{0, 1\}$ as the decrement such that $\mathbb{P}[S_t = 1] = q(a_t)$, the queue length is updated in each round as

$$x_{t+1} = \text{trunc}\,(x_t + I_t - S_t)\,,$$

where the "trunc" operator truncates the value to the interval $[0, N-1]$: $\text{trunc}(x) = \min\{N-1, \max\{x, 0\}\}$.

The discount factor will be set as $\gamma = 0.9$.

Problem 1: Policy evaluation

The first problem is to study the performance of the following simple policies:

Lazy policy: the policy that always uses the low service rate: $\pi_{\text{lazy}}(x) = a_{\text{low}}$.

Aggressive policy: the policy that uses the low service rate for short queues ($\pi_{\text{aggr}}(x) = a_{\text{low}}$ for $x < 50$) and the high service rate for long queues ($\pi_{\text{aggr}}(x) = a_{\text{high}}$ for $x \geq 50$).

Calculate the value functions of these policies using one of the following methods:

- The power iteration method covered in class. This method is described in detail in Section 4.1 of Sutton & Barto (2018) under the name "iterative policy evaluation".
- Directly solving the Bellman equations through the identity $V^\pi = \left(I - \gamma P^\pi\right)^{-1} r^\pi$, where r^π is the N-dimensional vector with its xth entry given as $r^\pi(x) = r(x, \pi(x))$ and P^π is the $N \times N$-dimensional matrix with entries $P^\pi(x, x') = P(x'|x, \pi(x))$. This identity follows from observing that the Bellman equations for policy evaluation can be written as the linear system of equations $V^\pi = r^\pi + \gamma P^\pi V^\pi$; reordering gives the result.

Having evaluated the policies, plot the difference between the resulting value functions, $V^{\pi_{\text{lazy}}} - V^{\pi_{\text{aggr}}}$. Which policy is better in state 50? What about state 80?

Problem 2: Value Iteration and Policy Iteration

The second problem is to calculate the optimal policy using the two elementary dynamic programming methods covered in class: Policy Iteration and Value Iteration. Plot the value functions produced by the two methods after 10, 20, 50, and 100 iterations! Which method converges faster to the optimal value function in terms of the number of iterations? What are the respective runtimes of performing 100 iterations with the two methods? What is your interpretation of these results?

Based on the results above, plot the difference between the optimal value function and the value functions of the two policies evaluated in Problem 1, $V^* - V^{\pi_{\text{lazy}}}$ and $V^* - V^{\pi_{\text{aggr}}}$! Can you verify that V^* strictly improves over these value functions? What is the optimal policy and how is it different from the basic ones evaluated in the first problem?

Hints

- For policy iteration, use the policy evaluation method developed in Problem 1 as a subroutine.
- If your code runs so fast that you can't measure a meaningful difference between the runtimes of policy iteration and value iteration, increase the size of the state space by setting a higher value of N (say, 1000).