Homework #2

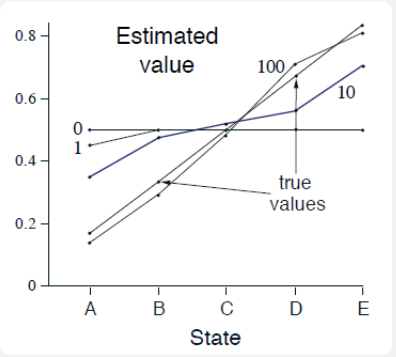
Q1. Obtain RSM error curves shown on the right side of the Figure for both TD(0) and MC methods.

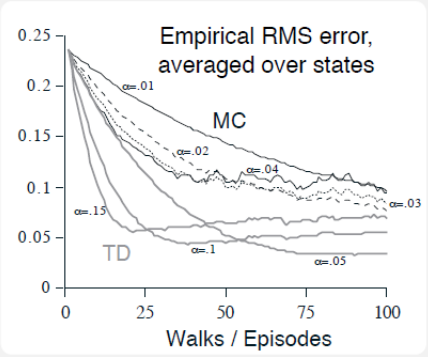
I am encouraging you to use python code for the task.

Example 6.2 Random Walk

In this example we empirically compare the prediction abilities of TD(0) and constant-α MC when applied to the following Markov reward process:

A *Markov reward process*, or MRP, is a Markov decision process without actions. We will often use MRPs when focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment from those due to the agent. In this MRP, all episodes start in the center state, C, then proceed either left or right by one state on each step, with equal probability. Episodes terminate either on the extreme left or the extreme right. When an episode terminates on the right, a reward of +1 occurs; all other rewards are zero. For example, a typical episode might consist of the following state-and-reward sequence: C, 0, B, 0, C, 0, D, 0, E, 1. Because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state. Thus, the true value of the center state is $v_\pi(\mathrm{C}) = 0.5$. The true values of all the states, A through E, are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$, and $\frac{5}{6}$.





The left graph above shows the values learned after various numbers of episodes on a single run of TD(0). The estimates after 100 episodes are about as close as they ever come to the true values—with a constant step-size parameter ($\alpha = 0.1$ in this example), the values fluctuate indefinitely in response to the outcomes of the most recent episodes. The right graph shows learning curves for the two methods for various values of α. The performance measure shown is the root mean-squared (RMS) error between the value function learned and the true value function, averaged over the five states, then averaged over 100 runs. In all cases the approximate value function was initialized to the intermediate value $V(s) = 0.5$, for all s. The TD method was consistently better than the MC method on this task.

Q2. Obtain RSM error curves shown in Example 7.1. I am also encouraging you to use python code for the task.

Example 7.1: n-step TD Methods on the Random Walk Consider using n-step TD methods on the 5-state random walk task described in Example 6.2. Suppose the first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1. Recall that the estimated values of all the states started at an intermediate value, $V(s) = 0.5$. As a result of this experience, a one-step method would change only the estimate for the last state, $V(\mathrm{E})$, which would be incremented toward 1, the observed return. A two-step method, on the other hand, would increment the values of the two states preceding termination: $V(\mathrm{D})$ and $V(\mathrm{E})$ both would be incremented toward 1. A three-step method, or any n-step method for $n > 2$, would increment the values of all three of the visited states toward 1, all by the same amount.

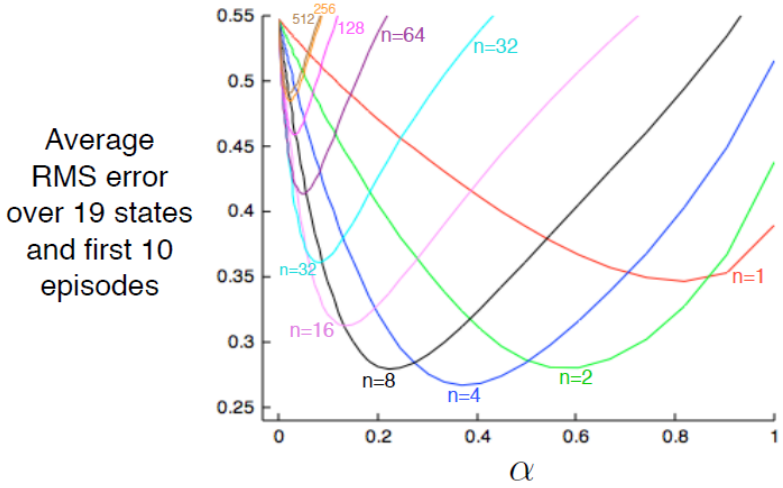


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1).

Q3. Obtain Episodes versus time steps curves shown in Example 6.3. I am also encouraging you to use python code for the task.

Example 6.5: Windy Gridworld Shown inset in Figure 6.3 is a standard gridworld, with start and goal states, but with one difference: there is a crosswind upward through the middle of the grid. The actions are the standard four—`up`, `down`, `right`, and `left`—but in the middle region the resultant next states are shifted upward by a "wind," the strength of which varies from column to column. The strength of the wind is given below each column, in number of cells shifted upward. For example, if you are one cell to the right of the goal, then the action `left` takes you to the cell just above the goal. Let us treat this as an undiscounted episodic task, with constant rewards of -1 until the goal state is reached.

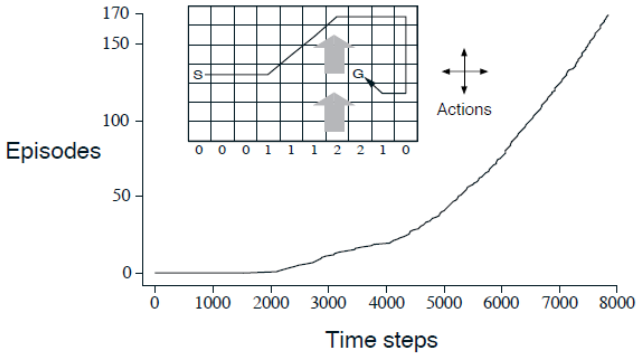


Figure 6.3: Results of Sarsa applied to a gridworld (shown inset) in which movement is altered by a location-dependent, upward "wind." A trajectory under the optimal policy is also shown.

The graph in Figure 6.3 shows the results of applying ε-greedy Sarsa to this task, with $\varepsilon = 0.1$, $\alpha = 0.5$, and the initial values $Q(s,a) = 0$ for all s, a. The increasing slope of the graph shows that the goal is reached more and more quickly over time. By 8000 time steps, the greedy policy was long since optimal (a trajectory from it is shown inset); continued ε-greedy exploration kept the average episode length at about 17 steps, two more than the minimum of 15. Note that Monte Carlo methods cannot easily be used on this task because termination is not guaranteed for all policies. If a policy was ever found that caused the agent to stay in the same state, then the next episode would never end. Step-by-step learning methods such as Sarsa do not have this problem because they quickly learn *during the episode* that such policies are poor, and switch to something else. ■