Planning and Learning in Dynamical Systems (046194)
Programming Homework 2, due 10/6/2019

Question 1 – Blackjack

Black Jack is a popular casino card game. The object is to obtain a hand with the maximal sum of card values, but without exceeding 21. All face cards count as 10, and the ace counts as 11 (unlike the original game). In our version, each player competes independently against the dealer, and the card deck is infinite (i.e., the probability of drawing a new card from the deck does not depend on the cards in hand).

The game begins with two cards dealt to the player and one to the dealer. If the player starts with 21 it is called a *natural* (an ace and a 10 card), and he wins (reward = 1).

If the player did not start with 21 he can request additional cards one by one (*hits*), until he either chooses to stop (*sticks*) or exceeds 21 (*goes bust*). If he goes bust, he loses (reward=-1), if he sticks – then it becomes the dealer's turn.

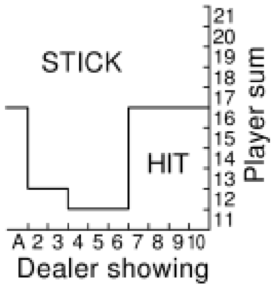
The dealer first draws a second card. Then, the dealer hits or sticks according to a fixed policy: he sticks on any sum of 17 or greater.

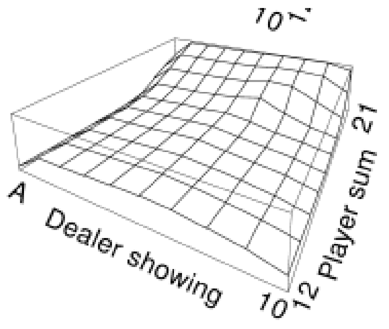
If the dealer busts, then the player wins (reward = 1). Otherwise, the outcome--win, lose, or draw--is determined by whose final sum is closer to 21.

We represent a state as (X,Y) where X is the current player sum and Y is the dealer's first card.

a. Describe the problem as a Markov decision process. What is the size of the state space?
b. Use value iteration to solve the MDP. Plot the optimal value function V^* as a function of (X,Y).
c. Use the optimal value function to derive an optimal policy. Plot the optimal policy as follows: for each value of the dealer's card (Y), plot the minimal value for which the policy sticks.

Here's an example of the plots you should provide (the values should be different though)





<u>Question 2</u> – The $c\mu$ rule revisited

Consider again the job processing domain of theoretical homework 3 (question 2):
N jobs are scheduled to run on a single server. At each time step (t=0,1,2,…), the sever may choose one of the remaining unfinished jobs to process. If job i is chosen, then with probability $\mu_i > 0$ it will be completed, and removed from the system; otherwise the job stays in the system, and remains in an unfinished state. Notice that the job service is memoryless – the probability of a job completion is independent of the number of times it has been chosen.
Each job is associated with a waiting cost $c_i > 0$ that is paid for each time step that the job is still in the system. The server's goal is minimizing the total cost until all jobs leave the system.

This time, we will solve the problem numerically, testing the various dynamic programming and reinforcement learning algorithms we have learned so far.

We consider the following specific case, with $N = 5$:

i	1	2	3	4	5
μ_i	0.6	0.5	0.3	0.7	0.1
c_i	1	4	6	2	9

<u>Part 1 – Planning</u>

In this part all the Python functions may use the true model (i.e., the μ_i's and c_i's) .

a. How many states and actions are in the problem?

Let S denote the state space. A deterministic policy π is a mapping from states to actions. In our case, it will be represented in Python as a vector of length $|S|$, in which each element denotes the selected action $(1,\ldots,N)$ for the corresponding state.

b. Write a function (in Python) that take as input a policy, and returns the corresponding value function V^{π}, also represented as a vector of length $|S|$. You can solve it either directly by matrix inversion, or iteratively. Remember that the value of the terminal state (no more jobs left) is zero by definition.

c. Plot the values of the policy π_c that selects the job with the maximal cost c_i, from the remaining unfinished jobs.

d. Write a function that calculates the optimal policy π^* using the **policy iteration** algorithm, and execute it, starting from the initial policy π_c. For each iteration of the algorithm, plot the value of the initial state s_0 (no jobs completed) for the current policy. How many steps are required for convergence?

e. Compare the optimal policy π^* obtained using policy iteration to the $c\mu$ law. Also plot V^{π^*} vs. V^{π_c}.

f. Write a **simulator** of the problem: a function that takes in a state s and action a, and returns the cost of the state $c(s)$, and a random next state s', distributed according to the transition probabilities of the problem.

Part 2 – Learning

In this part the learning algorithms cannot use the true model parameters, but only have access to the simulator function written above.

g. Policy evaluation: consider again the policy π_c, and use the TD(0) algorithm to learn the value function V^{π_c}. Start from $\hat{V}_{TD}(s)=0$ for all states. Experiment with several step size α_n schedules:

(i) : $a_n = 1/\left(no.of\ visits\ to\ s_n\right)$

(ii) : $a_n = 0.01$

(iii) : $a_n = \dfrac{10}{100+\left(no.of\ visits\ to\ s_n\right)}$

For each step-size schedule, plot the errors $\left\|V^{\pi_c} - \hat{V}_{TD}\right\|_\infty$ and $\left|V^{\pi_c}(s_0) - \hat{V}_{TD}(s_0)\right|$ as a function of iteration n. Explain the motivation behind each step-size schedule, and how it reflects in the results.

h. Now, run policy evaluation using TD(λ). Choose your favorite step-size schedule, and plot the errors $\left\|V^{\pi_c} - \hat{V}_{TD}\right\|_\infty$ and $\left|V^{\pi_c}(s_0) - \hat{V}_{TD}(s_0)\right|$ as a function of iteration n for several choices of λ. Repeat each experiment 20 times and display average results.

i. Q-learning: run Q-learning to find the optimal policy. Start from $\hat{Q}(s,a)=0$ for all states and actions. Experiment with the 3 previous step-size schedules. Use $\epsilon - greedy$ exploration, with $\epsilon = 0.1$. For some $\hat{Q}$, let $\pi_{\hat{Q}}$ denote the greedy policy w.r.t. $\hat{Q}$, i.e., $\pi_{\hat{Q}}(s) = \arg\min_a \hat{Q}(s,a)$.

For each step-size schedule, plot the errors $\left\|V^* - V^{\pi_{\hat{Q}}}\right\|_\infty$ and $\left|V^*(s_0) - \arg\min_a \hat{Q}(s_0,a)\right|$ as a function of iteration n. You may use the policy evaluation function you wrote in (b) to calculate $V^{\pi_{\hat{Q}}}$. If this step takes too long, you may plot the errors only for $n = 100, 200, 300, \ldots$ etc.

j. Repeat the previous Q-learning experiment for your favorite step-size schedule but now with $\epsilon = 0.01$. Compare.