Stanford CME 241 (Winter 2020) - Final Exam Solutions

1. Consider an MDP with infinite number of states $\mathcal{S} = \{1, 2, 3, \ldots\}$ and finite number of actions $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ for some fixed $m \geq 1$. The transition probabilities are as follows: For all states $s \in \mathcal{S}$,

$$Pr[(s+j)|(s, A_i)] = \begin{cases} \frac{1}{i} & \text{for each } 1 \leq j \leq i \\ 0 & \text{otherwise} \end{cases}$$

For all states $s, s' \in \mathcal{S}$ and for all actions $a \in \mathcal{A}$, $R(s, a, s')$ denotes the reward when transitioning from s to s' upon taking action a

$$R(s, a, s') = (s' - s)^n \text{ for some fixed } 1 \leq n \in \mathbb{R}$$

Let the discount factor be denoted by $0 \leq \gamma < 1$.

- **6 points**: Derive analytical expressions for the Optimal Value Function $V_*(\cdot)$ and for an Optimal Deterministic Policy (each in terms of m, n, γ).
- **4 points**: Now assume that an agent, instead of following an optimal policy, decides to follow the stochastic policy: $\pi(A_i|s) = \frac{1}{m}$ for all $1 \leq i \leq m$ for all states $s \in \mathcal{S}$. Derive an analytical expression for the Value Function $V_\pi(\cdot)$ for this policy π (in terms of m, n, γ).

Answer: Let us write out the Bellman Optimality Equation.

$$V_*(s) = \max_{1 \leq i \leq m} \{ \sum_{j=1}^{i} \frac{j^n + \gamma \cdot V_*(s+j)}{i} \}$$

We note that since this has an infinite number of states, since the movements are only in one direction, and since transitions and rewards do not depend on the state s (they depend on the action A_i and on $s' - s$ respectively), $V_*(s)$ should be independent of s. So we write $V_*(s)$ as simply the single number V_*. So we rewrite the Bellman Optimality Equation as:

$$V_* = \max_{1 \leq i \leq m} \{ \sum_{j=1}^{i} \frac{j^n + \gamma \cdot V_*}{i} \}$$

$$\Rightarrow V_* = \max_{1 \leq i \leq m} \{ \frac{1}{i} \sum_{j=1}^{i} j^n + \gamma \cdot V_* \}$$

$$\Rightarrow V_* - \gamma V_* = \max_{1 \leq i \leq m} \{ \frac{1}{i} \sum_{j=1}^{i} j^n \}$$

Since j^n is monotonically increasing as a function of j, we note that:

$$\underset{1 \leq i \leq m}{\text{argmax}} \{ \frac{1}{i} \sum_{j=1}^{i} j^n \} = m$$

. Therefore,

$$(1 - \gamma) V^* = \frac{1}{m} \sum_{j=1}^{m} j^n$$

$$\Rightarrow V_* = \frac{1}{m(1-\gamma)} \sum_{j=1}^{m} j^n$$

Therefore, the optimal policy $\pi_* : \mathcal{S} \to \mathcal{A}$ is given by:

$$\pi_*(s) = m \text{ for all } s \in \mathbb{Z}^+$$

Now we derive the analytical expression for the Value Function $V_\pi(\cdot)$ for the stochastic policy $\pi(A_i|s) = \frac{1}{m}$ for all $1 \leq i \leq m$ for all states $s \in \mathcal{S}$. We start by noting (based on the same argument as $V_*(\cdot)$) that $V_\pi(s)$ is independent of s. So we write $V_\pi(s)$ as simply the single number V_π. The Bellman Expectation Equation is:

$$V_\pi = \sum_{i=1}^{m} \{ \frac{1}{m} \cdot \{ \sum_{j=1}^{i} \frac{j^n + \gamma \cdot V_\pi}{i} \} \}$$

$$\Rightarrow V_\pi = \frac{1}{m(1-\gamma)} \sum_{i=1}^{m} \{ \frac{1}{i} \sum_{j=1}^{i} j^n \}$$

2. Assume you have fixed data available as N complete episodes where each episode is in the form:

$$S_1, R_1, S_2, R_2, \ldots, S_n, R_n, T$$

where $S_1, S_2, \ldots, S_n$ is the sequence of states visited in the episode and $R_1, R_2, \ldots, R_n$ are the associated rewards following the visited states. T is the terminal state. Note that n (the length of an episode) can vary across the N episodes. Note that there are no actions here, so the setting for this data is an underlying MRP and not MDP. Assume discount factor $\gamma = 1$.

Given only this data of fixed N episodes, your task is to implement working Python code that will estimate the Value Function for discount factor $\gamma = 1$ based on a variety of methods. An outline of the code is made available for you here. A couple of functions (get_state_return_samples, required for tabular Monte-Carlo, and get_state_reward_next_state_samples, required for the other methods) have been implemented for you and you may use them as helper functions. Your task is to implement the following methods, each compatible with their respective function interfaces provided in this outline code.

- **Tabular Monte-Carlo - 3 points:** Implement get_mc_value_function.
- **MRP - 6 points:** Implement get_probability_and_reward_functions and using its output, implement get_mrp_value_function (based on MRP Bellman Equation).
- **Tabular TD(0) with Experience Replay - 5 points:** Implement get_td_value_function.
- **LSTD - 6 points:** Implement get_lstd_value_function .

Make sure to read the comments/hints provided in the outline code (within each of the above functions you need to implement).

If you run the __main__ code, it will evaluate each of your 4 methods on the data that is set up in the __main__ code. In your answers-document, include the Python code you implemented and write the Value Function outputs you obtained for each of the 4 methods (up to 2 decimal places). **Make sure your output Value Functions correspond to discount factor $\gamma = 1$.**

Answer: The completed code is here. The output upon running __main__ is as follows:

- Monte-Carlo - A: 9.57, B: 5.64
- MRP - A: 12.93, B: 9,60
- TD - A: 12.88, B: 9.56
- LSTD - A: 12.93, B: 9.60

3. Assume you are the owner of a bank where customers come in randomly everyday to make cash deposits and to withdraw cash from their accounts. At the end of each day, you can borrow (from another bank, without transaction costs) any cash amount $y > 0$ at a constant daily interest rate R, meaning you will need to pay back a cash amount of $y(1 + R)$ at the end of the next day. Also, at the end of each day, you can invest a portion of your bank's cash in a risky (high return, high risk) asset. Assume you can change the amount of your investment in the risky asset each day, with no transaction costs (this is your mechanism to turn any amount of cash into risky investment or vice-versa). The key point here is that once you make a decision to invest a portion of your cash in the risky asset at the end of a day, you will not have access to this invested amount as cash that otherwise could have been made available to customers who come in the next day for withdrawals. More importantly, if the cash amount c in your bank at the start of a day is less than C, the banking regulator will make you pay a penalty of $K \cdot \cot(\frac{\pi \cdot c}{2C})$ (for a given constant K).

   For convenience, we make the following assumptions:

   - Assume that the borrowing and investing is constrained so that we end the day (after borrowing and investing) with positive cash ($c > 0$) and that any amount of regulator penalty can be immediately paid (meaning $c \geq K \cdot \cot(\frac{\pi \cdot c}{2C})$ when $c \leq C$).
   - Assume that the deposit rate customers earn is so small that it can be ignored.
   - Assume for convenience that the first half of the day is reserved for only depositing money and the second half of the day is reserved for only withdrawal requests.
   - Assume that if you do not have sufficient cash to fulfill a customer withdrawal request, you ask the customer to make the withdrawal request again the next day.
   - Assume all quantities are continuous variables.

   - **5 points:** Your first task is to model an MDP so you can run the bank in the most optimal manner, i.e., maximizing the Expected Utility of assets less liabilities at the end of a T-day horizon, conditional on any current situation of assets and liabilities. Specify the states, actions, transitions, rewards with precise mathematical notation (make sure you do the financial accounting from one day to the next precisely).
   - **5 points:** In a practical setting, we do not know the exact probability distributions of the customer deposits and withdrawals. Neither do we know the exact stochastic process of the risky asset. But assume we have access to a large set of historical data detailing daily customer deposits and withdrawal requests, as well as daily historical market valuations of the risky asset. Assume we also have data on new customers as well as leaving customers (sometimes due to their withdrawal requests not being satisfied promptly). Describe your approach to solve this problem with Reinforcement Learning by using the historical data described above. Specify which Reinforcement Learning algorithm you would use, including any customizations for the specifics of this problem. Although you are not expected to write any code for this problem, provide sufficient detail that will enable a programmer with knowledge of RL to implement your ideas.

Answer: *State* (when the bank closes its office for the day, i.e., after all deposits and withdrawal requests of the day are done) on day $1 \leq t \leq T$ is given by the 5-tuple $(t, x_t, y_t, z_t, w_t) \in \mathbb{R}^5$ where x_t is the current cash in the bank, y_t is the current price of one share of the stock, z_t is previous $(t-1)$ day's cash borrowing, and w_t is the shares invested in on the previous $(t-1)$ day.

Action (when the bank closes its office for the day, i.e., after all deposits and withdrawal requests of the day are done) is given by the 2-tuple $(a_t, b_t) \in \mathbb{R}^2$ where a_t is the cash to be borrowed on day t and b_t is the number of shares to invest in on day t. (a_t, b_t) is constrained on each day to ensure that the start-of-next-day cash $c_t = x_t + w_t y_t - z_t(1+R) + a_t - b_t y_t > 0$ and when $c_t \leq C$, we also need to ensure that $c_t \geq K \cot(\frac{\pi \cdot c_t}{2C})$.

Let the environment random variables be: Next day's deposits p_{t+1}, next day's withdrawal requests q_{t+1} and the percentage change s_{t+1} in the price of one share of the stock from current day to the next day. Then, the next day's state $(t+1, x_{t+1}, y_{t+1}, z_{t+1}, w_{t+1})$ (for $1 \leq t \leq T-1$) is given by:

$$x_{t+1} = x_t + w_t y_t - z_t(1+R) + a_t - b_t y_t - K \cot(\frac{\pi \cdot (x_t + w_t y_t - z_t(1+R) + a_t - b_t y_t)}{2C}) + p_{t+1} - q_{t+1}$$

$$y_{t+1} = y_t(1 + s_{t+1})$$

$$z_{t+1} = a_t$$

$$w_{t+1} = b_t$$

Reward on day t for $1 \leq t \leq T-1$ is 0, and *Reward* on day T is $U(x_T + w_T y_T - z_T(1+R))$ (where $U(\cdot)$ is the Utility function defined by the bank owner's risk-aversion).

We'd like to solve this problem with Reinforcement Learning. Although there are many different ways by which we could approach solving this problem and many different potential RL algorithms, here are some ideas.

- We use the historical data on a) Deposits b) Withdrawals c) Daily Stock Price movements to build a model of the above MDP (i.e., state transition probabilities based on the statistically-estimated probabilities of deposits, withdrawals and stock moves on any given day). One can get more sophisticated here by recording various explanatory factors for deposits, withdrawals and stock moves (eg: Day of week, demographics of current customers, econometric factors etc.). We would then have to predict the future movements of some of these explanatory factors and use those to predict the probabilities of future deposits, withdrawals and stock moves. We would use this model of the MDP to generate a large set of simulation episodes that account for above-mentioned probabilities. A much simpler way to generate these simulation episodes is to simple database the historical deposits, withdrawals, stock moves, and then to randomly pull a sample from the database anytime our simulation requires generation of the next state. This would operate as an experience-replay and this simple approach would make for a good first-version prototype before developing a more sophisticated simulator.
- Since the action space is large (large number of possible (a_t, b_t) tuples), it would be suitable to employ a Policy Gradient algorithm. Secondly, the state space is also large, so will need a good function approximation for the Q-value function (that Policy Gradient requires). This would be an Actor-Critic algorithm. One would need to make a prudent choice of features for both the Actor and the Critic neural networks (although an initial prototype could be quickly built with linear-function approximations).