Stanford CME 241 (Winter 2021) - Final Exam

Instructions:

- We trust that you will follow The Stanford Honor Code.
- You have about 50 hours to work on this test, and need to submit your answers by 11:59pm Tuesday March 16. However, the estimated amount of work for this test is about 3-6 hours, depending on your proficiency in typesetting mathematical notations and in writing code. There are 3 problems (with subproblems), with a total of 40 points.
- Include all of your writing, math formulas, code, graphs etc. into a single answers-document. Be sure to write your name and SUNET ID in your answers-document. You can do your work in LaTeX or Markdown or Jupyter or any other typesetting option, which should finally be converted to the single answers-document PDF to be submitted. Note: Code can be included in LaTeX using the *lstlisting* environment, and graphs can be included using *includegraphics*.
- Submit your answers-document on Gradescope. Please ensure you have access to Gradescope before starting the exam so there is ample time to correct the problem. The Gradescope assignment for this final is here.
- **DO NOT** upload your solutions to github

Problems:

1. **Problem 1 (Total of 18 points)**:

   We consider an extension of the midterm problem on shortest-path grid navigation. In this extension, we are given a grid comprising of cells arranged in the form of m rows and n columns, defined as $\mathcal{G} = \{(i, j) \mid 0 \leq i < m, 0 \leq j < n\}$. A subset of $\mathcal{G}$ (denoted $\mathcal{B}$) are uninhabitable cells known as *blocks*. A subset of $\mathcal{G} - \mathcal{B}$ (denoted $\mathcal{T}$) is known as the set of goal cells. We have to find a least-cost path from each of the cells in $\mathcal{G} - \mathcal{B} - \mathcal{T}$ to any of the cells in $\mathcal{T}$. Like the midterm problem, at each step, we are required to make a move to a non-block cell (we cannot remain stationary). However, unlike the midterm problem, after we make a move, a random vertical wind could move us one cell up or down, unless limited by a block or limited by the boundary of the grid.

   Each column has it's own random wind specification given by two parameters $0 \leq p_1 \leq 1$ and $0 \leq p_2 \leq 1$ with $p_1 + p_2 \leq 1$. The wind blows downwards with probability p_1, upwards with probability p_2, and there is no wind with probability $1 - p_1 - p_2$. If the wind makes us bump against a block or against the boundary of the grid, we incur a cost of $b \in \mathbb{R}^+$ in addition to the usual cost of 1 for each move we make. Thus, here the cost includes not just the time spent on making the moves, but also the cost of bumping against blocks or against the boundary of the grid (due to the wind). Minimizing the expected total cost amounts to finding our way to a goal state in a manner that combines minimization of the number of moves with the minimization of the hurt caused by bumping (assume discount factor of 1 when minimizing this expected total cost). If the wind causes us to bump against a wall or against a boundary, we bounce and settle in the cell we moved to just before being blown by the wind (note that the wind blows right after we make a move). The wind will never move us by more than one cell between two successive moves, and the wind is never horizontal. Note also that if we move to a goal cell, the process ends immediately without any wind-blow following the movement to the goal cell. The random wind for all the columns is specified as a sequence $[(p_{1,j}, p_{2,j}) \mid 0 \leq j < n]$.

   - **5 points:** Model this problem of minimizing the expected total cost while reaching a goal cell as a Finite Markov Decision Process with precise mathematical notation for states (non-terminal and terminal), actions, rewards and transition probabilities.
   - **5 points:** Implement your mathematical specification of the MDP within this Python code outline (download this code, but **be sure not to upload your code anywhere**). For this part of the problem, all you have to do is to **implement the method `get_transition_probabilities`**. Make sure you do not make any changes to the interface of any of the methods, including `get_transition_probabilities`. The remainder of the code, along with the driver code in `__main__`, is written for you (the driver code will run Value Iteration and print the Optimal Value Function and Optimal Policy of the MDP produced by your implementation of the method `get_transition_probabilities`).
   - **4 points:** Implement On-Policy TD Control (i.e., ϵ-greedy SARSA with declining ϵ) to solve this problem **by simply implementing the method `get_sarsa_vf_and_policy`**. Make sure you do not change the interface of `get_sarsa_vf_and_policy`. In particular, SARSA cannot access the transition probabilities of the MDP, and only has access to the dictionary of allowable moves for each non-block cell and to the function for sampling the next (post-wind) cell and reward (negative of cost), given as input the cell (from which the move was made) and the move one made from that cell. The comments within the method will provide some guidance.

- **4 points:** Implement Off-Policy TD Control (i.e., ϵ-Greedy Q-Learning) to solve this problem **by simply implementing the method get_q_learning_vf_and_policy**. Make sure you do not change the interface of get_q_learning_vf_and_policy. The comments within the method will provide some guidance.

For a complete submission, make sure to include in your answers-submission-document not just the code that you wrote but also the output grid displays as produced by running __main__.

2. **Problem 2 (Total of 12 points)**:

Consider the following discrete-time MDP for constrained consumption. At $t = 0$, the agent is given a finite amount $x_0 \in \mathbb{R}^+$ of a resource. In each time period, the agent can choose to consume any amount of the resource, with the consumption denoted as $c \in [0, x]$ where x is the amount of the resource remaining at the start of the time period. This consumption results in a reduction of the resource at the start of the next time period: $x' = x - c$. Consuming a quantity c of the resource provides a utility of consumption equal to $U(c)$, and we adopt the CRRA utility function:

$$U(c) = \frac{c^{1-\gamma}}{1-\gamma}, \ (\gamma > 0, \gamma \neq 1)$$

Our goal is to maximize the aggregate discounted utility of consumption until the resource is completely consumed. We assume a discount factor of $\beta \in [0, 1]$ when discounting the utility of consumption over any single time-period.

We model this as a discrete-time, continuous-state-space, continuous-action-space, stationary, deterministic MDP and so, our goal is to solve for the Optimal Value Function and associated Optimal Policy, which will give us the optimal consumption trajectory of the resource. Since this is a stationary MDP, the *State* is simply the amount x of the resource remaining at the start of a time period. The *Action* is the consumption quantity c in that time period. The *Reward* for a time period is $U(c)$ when the consumption in that time period is c. The discount factor over each single time period is β.

We assume that the Optimal Policy is given by: $c^* = \theta^* \cdot x$ for some $\theta^* \in [0, 1]$.

- **3 points:** Our first step is to consider a fixed deterministic policy, given by: $c = \theta \cdot x$ for some fixed $\theta \in [0, 1]$. Derive a closed-form expression for the Value Function $V_\theta(x)$ for a fixed deterministic policy, given by: $c = \theta \cdot x$. Specifically, you need to express $V_\theta(x)$ in terms of only β, γ, θ, x
- **3 points:** Use this closed-form expression for $V_\theta(x)$ to solve for the θ^* which maximizes $V_\theta(x)$ (thus fetching us the Optimal Policy given by: $c^* = \theta^* \cdot x$)
- **2 points:** Use this expression for θ^* to obtain an expression for the Optimal Value Function $V^*(x)$ in terms of only β, γ, x.
- **4 points:** Validate that the Optimal Policy (derived in part 2) and Optimal Value Function (derived in part 3) satisfy the Bellman Optimality Equation.

3. **Problem 3 (Total of 10 points)**:

Assume that every mortgage issued by a bank is characterized by a pair (L, M) where L is the loan amount of the mortgage (the money lent by the bank) and M is the mortgage rate. Once the loan L is extended, it is meant to be paid back to the bank in the form of end-of-month monthly payments (including principal and interest) over a period of n months (where n is fixed, say 360 months). In order to make the monthly payments exactly the same over the n months, the loan balance reduces in a specific manner over the n months (since monthly payments include principal). Denoting the interest and principal payments to be made at the end of month i (for all $0 \le i < n$) as I_i and P_i respectively and the loan balance at the start of month i (for all $0 \le i \le n$) as B_i, we need to have:

$$B_0 = L$$

$$B_{i+1} = B_i - P_i \text{ for all } 0 \le i < n$$

$$B_n = 0$$

$$P_i + I_i = P_j + I_j \text{ for all } 0 \le i < j < n$$

$$I_i = \frac{B_i \cdot M}{12} \text{ for all } 0 \le i < n$$

To satisfy the above, we get:

$$B_i = L \cdot \frac{(1 + \frac{M}{12})^n - (1 + \frac{M}{12})^i}{(1 + \frac{M}{12})^n - 1} \text{for all } 0 \le i \le n$$

$$P_i = \frac{\frac{B_i \cdot M}{12}}{(1 + \frac{M}{12})^{n-i} - 1} \text{ for all } 0 \le i < n$$

$$I_i = \frac{B_i \cdot M}{12} \text{ for all } 0 \le i < n$$

$$P_i + I_i = \frac{B_i \cdot M}{12} \cdot (1 + \frac{1}{(1 + \frac{M}{12})^{n-i} - 1}) = \frac{L \cdot M}{12} \cdot \frac{1}{1 - (1 + \frac{M}{12})^{-n}} \text{ for all } 0 \le i < n$$

Now assume that you have just bought an expensive house by taking out a mortgage with a loan amount L and mortgage rate M. After the first month of your mortgage, at the beginning of each month, the bank offers you an updated mortgage rate that reflects the updated market conditions (assumed to be stochastic). You can:

1. Choose to ignore the bank's offer of the updated mortgage rate and continue on with the payments for your current mortgage, or you can
2. Choose to cancel your current mortgage and enter into a new mortgage where the loan amount will be equal to the loan balance on your current mortgage and the mortgage rate will be the updated mortgage rate the bank just offered you.

If you choose Option 2., there is a fixed transaction cost of C to cancel your current mortgage and enter into the new mortgage (as an example, the fixed transaction cost C might be \$5000 for the requisite paperwork to cancel your current mortgage and issue a new mortgage). Note that borrowing a loan amount for the new mortgage that is exactly equal to the loan balance on

the current mortgage enables clean cancellation of the current mortgage and issuance of the new mortgage. Note that the new mortgage is for a duration of n months (since it's a fresh mortgage) whereas the current mortgage has a remaining duration of less than n months. Note also that the new mortgage has a loan amount L' that is smaller than the current mortgage's original loan amount L.

Your objective at any time is to minimize the expected aggregated discounted future payments (principal plus interest payments, as well as any transaction costs) to be made from that time onwards. To be precise on this objective, we are making the following two simplifying assumptions:

- We assume that "time value of money" is based on a constant discount rate $r \in \mathbb{R}$, i.e., a dollar paid at time t is equivalent to $(1+r)^{t'-t}$ dollars paid at time t' for all $t' > t$ with t and t' expressed in terms of number of years (i.e. r is a constant annualized discount rate).
- Utility of money is linear, i.e., Utility function is $U(x) = x$.

For simplicity, assume that you will live forever and that you will never default on your payments.

- **6 points**: Model this problem as a Markov Decision Process with precise mathematical notation (specifying the *State*, *Action*, *Reward*, *Transitions*, *Discount Factor*) such that the Optimal Policy of this MDP enables you to make financially optimal decisions (according to the above-defined objective) on whether or not to accept the bank's offer of an updated mortgage rate at the beginning of each month.
- **4 points**: If an economist provided you with a monthly time-steps stochastic process for the Mortgage Rate the bank would offer you over a long enough time period, say over the next 100 years, how would you go about solving the Control problem for this MDP? You don't have to write any code or even pseudo-code, but you should provide sufficient detail about your algorithm so that a programmer upon reading your description would be able to implement the Control algorithm.