# Stanford CME 241 (Winter 2019) - Midterm Exam

1. **(4 points) Bellman Expectation Equations: Aim to do this question in 10 minutes**. Given a finite MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ and given a *Deterministic* Policy $\pi : \mathcal{S} \to \mathcal{A}$, write with precise notation the 4 Bellman Expectation Equations for the state value function and action value function for fixed policy $\pi$, i.e., $v_\pi$ in terms of $q_\pi$, $q_\pi$ in terms of $v_\pi$, $v_\pi$ in terms of $v_\pi$, $q_\pi$ in terms of $q_\pi$.

2. **Manual Value Iteration: Aim to do this Question in 30 minutes**. Consider a simple MDP with 3 states $s_1$, $s_2$, $s_3$ and 2 actions $a_1$, $a_2$. The transition probabilities and expected rewards are:

```
{
  s1: {
    a1: ({s1: 0.2, s2: 0.6, s3: 0.2}, 8.0),
    a2: ({s1: 0.1, s2: 0.2, s3: 0.7}, 10.0)
  },
  s2: {
    a1: ({s1: 0.3, s2: 0.3, s3: 0.4}, 1.0),
    a2: ({s1: 0.5, s2: 0.3, s3: 0.2}, -1.0)
  },
  s3: {
    a1: ({s3: 1.0}, 0.0),
    a2: ({s3: 1.0}, 0.0)
  }
}
```

   Assume discount factor $\gamma = 1$.

   Your task is to determine an Optimal Deterministic Policy *by manually working out* simply the first two iterations of Value Iteration algorithm.

   - **3 points**: Initialize the Value Function for each state to be it's max (over actions) reward, i.e., we initialize the Value Function to be $v_0(s_1) = 10.0, v_0(s_2) = 1.0, v_0(s_3) = 0.0$. Then manually calculate $q_k(\cdot, \cdot)$ and $v_k(\cdot)$ from $v_{k-1}(\cdot)$ using the Value Iteration update, and then calculate the greedy policy $\pi_k(\cdot)$ from $q_k(\cdot, \cdot)$ for $k = 1$ and $k = 2$ (hence, 2 iterations).
   - **3 points**: Now argue informally (no need for a formal proof) that $\pi_k(\cdot)$ for $k > 2$ will be the same as $\pi_2(\cdot)$. Hint: You can make the argument by examining the structure of how you get $q_k(\cdot, \cdot)$ from $v_{k-1}(\cdot)$. With this argument, there is no need to go beyond the two iterations you performed above, and so you can proclaim $\pi_2(\cdot)$ as an Optimal Deterministic Policy for this MDP.

3. **Dynamic Price Optimization: Aim to do this Question in 40 minutes**. You own a supermarket and you are $T$ days away from Halloween ☺. You have just received $M$ Halloween masks from your supplier. You want to dynamically set the selling price of the Halloween masks at the start of each day in a manner that maximizes your *Expected Total Sales Revenue* for Halloween masks this season (assume no one will buy Halloween masks after Halloween).

   Assume that for each of the $T$ days, you are required to select a price for that day from one of $N$ prices $p_1, p_2, \ldots, p_N \in \mathbb{R}$, and that price is the selling price for all masks on that day. Assume that the customer demand for number of Halloween masks on any day is governed by a Poisson probability distribution with mean $\lambda_i \in \mathbb{R}$ if you select that day's price to be $p_i$ (where $i$ is a choice among $1, 2, \ldots, N$).

   Note that on any given day, the demand could exceed the number of Halloween masks you have in the store, in which case the number of masks sold on that day will be equal to the number of Halloween masks you had at the start of that day.

   - **6 points**: Write the Bellman Optimality Equation customized to this MDP. Essentially you need to express the Optimal Value Function $v_*$ recursively based on taking the best action in the current state and based on the subsequent random customer demand that would produce the appropriate reward and take you to the next state. Note: The probability mass function of a Poisson distribution with mean $\lambda \in \mathbb{R}$ is given by:

$$f(k) = \frac{e^{-\lambda} \cdot \lambda^k}{k!} \text{ for } k = 0, 1, 2, \ldots$$

   - **2 points**: To be able to solve the $v_*$ recursion, you need to know the values of $v_*$ for the boundary case(s) (boundary states). Write down the values of $v_*$ for the boundary states.
   - **7 points**: You can solve this $v_*$ recursion (hence, solve for the Optimal Policy $\pi_*$) with a numerical recursive algorithm (essentially a special form of Dynamic Programming algorithm customized to this problem). Write Python-like pseudo-code for this algorithm that would enable you to dynamically set the selling price at the start of each day. Clearly define the inputs and output of your algorithm with their types (int, float, List, Mapping etc.).