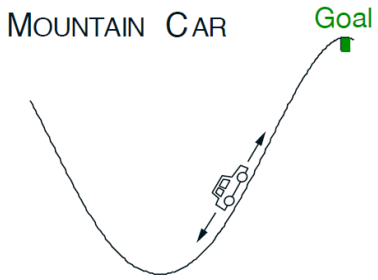
Homework 6: On-policy Control with Approximation

Due: Wednesday, November 11th 11:59 pm

The purpose of this project is to study different properties of Function Approximation with on-policy control methods.

Task Description

Consider the task of driving an underpowered car up a steep mountain road, as suggested by the diagram in the upper left of the following figure. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. The only solution is to first move away from the goal and up the opposite slope on the left. Then by applying full throttle the car can build up enough inertia to carry it up the steep slope even though it is slowing down the whole way.



This is a continuous control task where things have to get worse in a sense (farther from the goal) before they can get better. The reward in this problem is -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode. There are three possible actions: full throttle forward (+1), full throttle reverse (-1), and zero throttle (0). The car moves according to a simplified physics. Its position x_t and velocity $\dot{x}_t$ are updated by

$$x_{t+1} = bound[x_t + \dot{x}_{t+1}]$$

$$\dot{x}_{t+1} = bound[\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)]$$

where the *bound* operation enforces $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. In addition, when x_{t+1} reached the left bound, $\dot{x}_{t+1}$ was reset to zero. When it reached the right bound, the goal was reached and the episode was terminated. Each episode starts from a random position $x_t \in [-0.6, -0.4)$ and zero velocity.

Part I

You have been given a simple implementation of the Mountain Car task.

1. Your first task is to check and confirm that the given code simulates the above formulae and task description. Then write a function that generates random episodes for this task. You should use the given code for the Mountain Car task.
2. Use the Pygame library to develop a simple function that animates a given episode/trajectory. The equation for the mountain is $y = 0.45sin(3x) + 0.55$. Use a randomly generated episode from the function you developed above and pass it to your animation function, then plot the results.

Part II

Develop a function approximation procedure based on either **Polynomials** or **Fourier basis** (recommended). Given the current $\boldsymbol{w}$, the developed function approximation method should return the value for each specific state.

Part III

1. Implement the Episodic Semi-gradient n-step SARSA (details in page 247).
2. Use the algorithm to learn the Mountain Car task with $n \in \{1, 8, 16\}$. Tune the step-size parameter (α), select a proper Function Approximation order, discount factor (γ), exploration probability (ε). Plot step-per-episode (in log scale) vs. number of episodes. This plot should be averaged over 50-100 runs.
3. Show an animation of the task for each n value.
4. Which value of n results in faster learning? Why?

Evaluation: we will grade your submission according to the following table:

Item	COMP4600	COMP5300
Part I	10+20	10+20
Part II	30	30
Part II	20+10+5+5	20+10+5+5

Note 1: The parts marked with * are optional for COMP4600 (undergraduates) but mandatory for COMP5300 (graduates). There are no optional parts in this homework.

Note 2: We do not accept code in any programming language other than Python 3 (do not use Python 2).

Note 3: For the implementation of the algorithms, you are not allowed to rely on any python library other than `numpy` and `matplotlib` (for plotting), and `pygame`.

Note 4: All the code, plots, explanations should be included in a single Jupyter Notebook (`.ipynb`) file. Include your name as part of the filename and submit through Blackboard.

Submission: By 11:59pm on Wednesday, November 11th 2020, submit your `student_name.ipynb` file on Blackboard. Make sure everything is entirely contained within this file and it runs without any error.