ECE183DA (Winter 2020)

Design of Robotic Systems I

Prof. Ankur Mehta
mehtank@ucla.edu

Problem set 1
Due 3pm Tue. Feb. 24, 2020

1 Lab Overview

1.1 Objectives

The goal of this lab is to explore Markov Decision Processes (MDPs) to control a simple discretized robot. You will develop and implement a model of the robot behavior and use it to optimally accomplish a prescribed task.

1.2 Deliverables

As an individual or a team, you will create a well documented git repository containing all your code and data. Using comments in your code, organized and indexed in your `README.md`, you will clearly identify answers to the numbered questions below. You can work with up to 2 other students provided that you clearly indicate who you worked with, and everyone agrees to equally share credit for the assignment.

Submit a link to your code repository on CCLE by 3pm Tue. Feb. 24, 2020. Submissions that are up to 24 hours late will be accepted for a 10 percentage point reduction in final grade. No submissions will be accepted more than 24 hours late.

2 Lab specification

Preliminaries

0(a). Who did you collaborate with?

0(b). What other resources did you consult?

Coding notes

- You will be running MDP solvers for various values of the discount factor γ and error probability p_e. You may want to wrap your code in an MDP class that has initializers for γ and p_e so that you can efficiently explore the effects of these values.
- You may need to create data objects to represent states $s \in S$ and/or actions $a \in A$. Note that the state s will later be used as an index to a matrix/array when computing policies and values.
- For the sake of evaluation, please create a single top level function / script that generates all the relevant results in one command, and include installation / running instructions in the `README.md` file. If we can't run it, we can't grade it.

2.1 MDP System

Consider a simple robot in a 2D grid world of length L and height H. That is, the robot can be located at any lattice point $(x, y) : 0 \leq x < L, 0 \leq y < H; x, y \in \mathbb{N}$. Some of these lattice points may be obstacles that the robot cannot occupy.

In each time step, the robot can either attempt to take one step in either of the four cardinal directions {Up, Down, Left, Right}, or choose to Not move. If the robot does choose to move, it may **instead** experience an error with probability p_e, and **actually** take one step in any of the four cardinal directions with equal probability. Note

that sometimes the result of the error is the same as the originally chosen motion. If the robot chooses not to move, it will not experience any error.

If a motion would result in moving off of the grid or into an obstacle (whether commanded or as a result of an error), the robot will instead stay where it is.

Create code to simulate this system:

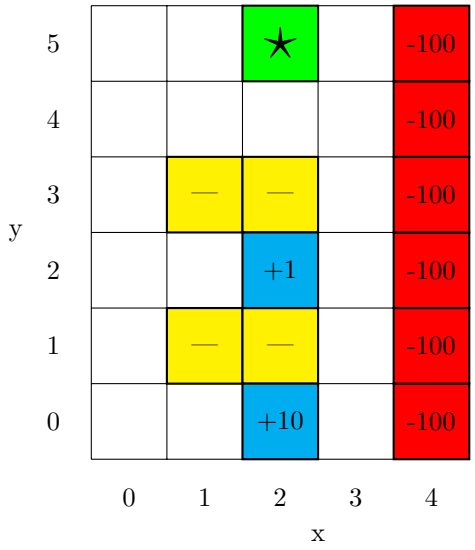
1(a). Create (in code) your state space $S = \{s\}$. What is the size of the state space N_S?

1(b). Create (in code) your action space $A = \{a\}$. What is the size of the action space N_A?

1(c). Write a function that returns the probability $p_{sa}(s')$ given inputs s, a, s' (and global constant error probability p_e). Assume for now that there are no obstacles.

2.2 Planning problem

Consider the grid world shown below, with $L = 5, H = 6$:



Obstacles are colored yellow and marked with a —; the robot cannot occupy these states. Some states, colored blue or red, have a listed reward (either +1, +10, or -100). The remaining states have 0 reward. The rewards for each state are based on starting a transition in that state, and are independent of both the action taken from that state and the next state. The robot might for example start in the inital state colored green and marked with a ★.

2(a). Update the state transition function from 1(c) to incorporate the displayed obstacles.

2(b). Write a function that returns the reward $r(s)$ given input s.

2.3 Policy iteration

Assume an initial policy π_0 of always taking a step to the Left. In this section, assume $\gamma = 0.9$, $p_e = 0.01$.

3(a). Create and populate a matrix/array that contains the actions $\{a = \pi_0(s)\}$, $\pi_0(s) \in A$ prescribed by the initial policy π_0 when indexed by state s.

3(b). Write a function to display any input policy π, and use it to display π_0.

3(c). Write a function to compute the policy evaluation of a policy π. That is, this function should return the matrix/array of values $\{v = V^\pi(s)\}$, $V^\pi(s) \in \mathbb{R}$ when indexed by state s. The input will be a matrix/array storing π as above (and will use global constant discount factor γ).

3(d). Write a function that returns a matrix/array π giving the optimal policy under a one-step lookahead (Bellman backup) when given an input value function V. Display the policy that results from a one-step improvement on π_0.

3(e). Combine your functions above to create a new function that computes policy iteration on the MDP, returning optimal policy π^* with optimal value V^*. Display π^*.

3(f). How much compute time did it take to generate your optimal policy in 3(e)? You may want to use your programming language's built-in runtime analysis tool.

3(g). Starting in the initial state as drawn above, plot a trajectory under the optimal policy. What is the total discounted reward of that trajectory? What is the expected discounted sum-of-rewards for a robot starting in that inital state?

2.4 Value iteration

4(a). Using an initial condition $V(s) = 0 \; \forall s \in S$, write a function (and any necessary subfunctions, perhaps derived from the functions above) to compute value iteration, again returning optimal policy π^* with optimal value V^*.

4(b). Run this function to recompute and display the optimal policy π^*. Also generate a trajectory for a robot and compute its reward as described in 3(g). Compare these results with those you got from policy iteration.

4(c). How much compute time did it take to generate your results from 4(b)? Use the same timing method as in 3(f).

2.5 Additional scenarios

5(a). Explore different values of γ, p_e to find different optimal policies, and characterize / explain your observations.