Assignment 2

Markov Decision Processes, Classical Reinforcement Learning, and Model Predictive Control

General Information

Due date:	You can find the due date in the syllabus handout, on *Gradescope*, and in the course calendar. Your solution must be submitted on *Gradescope* before 23h59.
Submission:	Please submit your solution and the requested Matlab scripts (highlighted in `blue`) as a single PDF document. Both typed and scanned handwritten solutions are accepted. It is your responsibility to provide enough detail such that we can follow your approach and judge your solution. Students may discuss assignments. However, each student must code up and write up their solutions independently. We will check for plagiarism. The points for each question are shown in the left margin.

Introduction

In this assignment, we will explore different methods for solving discrete and continuous Markov Decision Processes (MDPs). This assignment consists of two problems[1]. In the first problem, we will consider a grid world scenario with discrete state and action spaces and implement both model-based and sample-based reinforcement learning (RL) methods to solve the MDP. In the second problem, we will use the classical mountain car problem as a benchmark to examine methods for discretizing a continuous MDP such that the RL methods developed in the first problem can be effectively applied. After solving the mountain car problem with RL-based methods, we will further explore the use of control approaches, specifically, model predictive control (MPC), to solve this problem.

Problem 2.1 Grid World

We consider the grid world problem shown in Figure 1. The goal is to plan a path from the start state 'S' to the goal state 'G' without going into the obstacles indicated by the shaded boxes. The possible actions for the agent in each state are {'up', 'down', 'left', 'right'}. We assume the system dynamics is deterministic for this problem (i.e., the agent always transitions into the state that it intends to be in). The agent receives a reward of -1 for each state transition and a reward of -100 if transitioning into an obstacle state. Each episode is terminated when the agent reaches the goal state.

We will explore both model-based and sample-based methods to solve the grid world problem. The main script for this problem is `./p1_grid_world/main_p1_gw.m`. Follow the instructions below and the comments in the main script to complete the corresponding functions in the algorithm directory `./p1_grid_world/algorithms`.

[1]This assignment is partially adapted from [1].

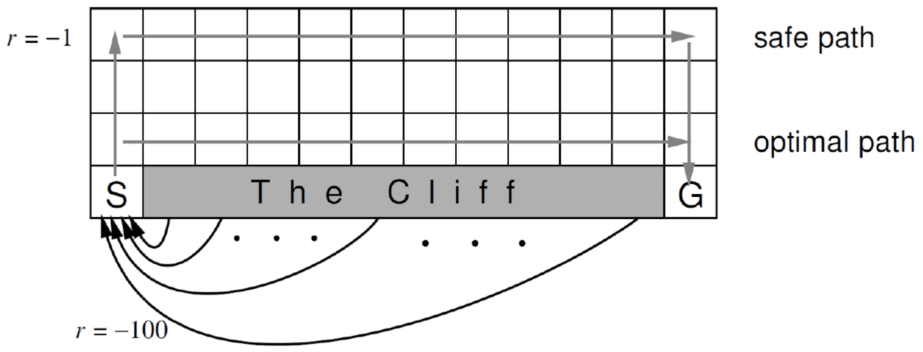


Figure 1: Gridworld problem [1]: The goal is to guide an agent to walk from the start state indicated by 'S' to the goal state 'G' without falling into the obstacle states indicted by the shaded boxes.

Model-Based Methods

2 (a) *Generalized Policy Iteration:* Implement the Generalized Policy Iteration (GPI) algorithm in `generalized_policy_iteration.m`. Use appropriate terminal conditions for Policy Evaluation and Policy Improvement processes and implement the Policy Iteration (PI) and the Value Iteration (VI) algorithms. Report the heatmaps of the converged value functions and solutions to the grid world problem. Comment on the differences between the PI and VI algorithms.

+2 (b) *Linear Programming (Extra Credit):* From the optimal Bellman Equation, we know that the optimal state value function v_* is a solution to the following set of equations:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v_*(s') \right], \; \forall s \in \mathcal{S}, \tag{1}$$

where $s, s' \in \mathcal{S}$ denote the current and the next states, respectively, $a \in \mathcal{A}$ is the action of the agent, $p(s' \mid s, a)$ is the transition model, and $r(s, a, s')$ is the expected immediate reward, and γ is the discounting rate. Due to the presence of the maximization operation, it is non-trivial to directly solve for v_* from Eqn. (1). In part (a), we saw that we can use the GPI algorithm to iteratively find v_* for the grid world problem. In this part, we explore an alternative Linear Programming (LP)-based approach to solve for the optimal state value function v_*. In particular, it can be shown that the optimal solution to the following LP is the optimal state value function v_*:

$$\begin{aligned} \min_{\xi} \quad & \sum_{s} c(s)\xi(s) \\ \text{s.t.} \quad & \xi(s) \geq \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma \xi(s') \right], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \end{aligned} \tag{2}$$

where $\xi(s)$ is the set of decision variables, and $c(s)$ is a set of positive weights. For MDPs with a moderate number of discrete states, we can obtain v_* by solving (2) with standard LP algorithms.

(i) Implement the LP approach in `linear_programming.m` and solve the grid world problem. Report the heatmap of the optimal value function. **Note:** You may use the Matlab built-in function `linprog` or tools such as `cvx` [2] to solve the LP.

(ii) Give an intuition behind the LP formulation in (2). **Hint:** The LP in (2) is equivalent to

$$\begin{aligned} \min_{\xi} \quad & \sum_{s} c(s)\xi(s) \\ \text{s.t.} \quad & \xi(s) \geq \max_{a\in\mathcal{A}(s)} \sum_{s'} p(s' \mid s, a)\left[r(s,a,s') + \gamma\xi(s')\right], \quad \forall s \in \mathcal{S}. \end{aligned} \tag{3}$$

Show that the inequity constraints in (3) require $\xi(s) \geq v^*(s)$ for all $s \in \mathcal{S}$, and thus the optimal solution to the minimization problem in (3) is v_*.

Sample-Based RL

1.5 (c) *Monte Carlo (On-Policy):* Solve the grid world MDP with the Monte-Carlo algorithm with ε-soft policy. To do so, you should generate training episodes in which the agent follows some ε-soft policy $\pi(a \mid s)$. After each episode, the policy should be improved based on the observations of that episode using model-free Policy Improvement. For this problem you only need to consider a constant ε (i.e., with `k_epsilon` set to 1).

Note: At each step in a training episode, you need to sample an action from the ε-soft policy that is represented as a probability distribution. Denote $\hat{f}_x(x)$ as a generic probability density function (PDF) over a discrete random variable (DRV) $x \in \{..., -1, 0, 1, ...\}$ and $\hat{F}_x(x) = \sum_{\bar{x}=-\infty}^{x} f(\bar{x})$ as the associated cumulative density function (CDF). One approach to sample from the distribution $\hat{f}_x(x)$ is to first generate a random number u_s from a uniform distribution $f_u(u)$ that has a support over the closed interval $[0, 1]$, and the corresponding sample x_s is the one that satisfies $\hat{F}_x(x_s - 1) < u \leq \hat{F}_x(x_s)$. Details of this approach and the sampling methods for more general cases can be found in [3].

(i) Complete `monte_carlo.m` and report the heatmap of the converged value function with the optimized path. Can the algorithm find the optimal greedy policy?

(ii) What is the impact of varying the soft policy parameter ε?

1.5 (d) *Q Learning (Off-Policy):* Solve the grid world problem the Q-learning algorithm. Similar to part (c), you should generate training episodes which the agent follows an ε-greedy policy. The difference is that, this time, the updates of the action value function happens during the episode.

(i) Complete `q_learning.m` and report the heatmap of the converged value function with the optimized path. First start with a fixed ε (i.e., with `k_epsilon` set to 1) and then with decreasing ε (i.e., with `k_epsilon` set to a value between 0 and 1). Comment on the impact of changing the decay parameter `k_epsilon`.

(ii) Briefly comment on the differences between the Monte-Carlo algorithm and the Q-learning algorithm in terms of the converged solution and their computational efficiency.

(iii) Rerun the Q-learning algorithm with simulated measurement noise. The 'noisiness' of the measurement is controlled by the parameter `noise_alpha`. The parameter `noise_alpha` takes values between 0 and 1. Observations are noise-free when `noise_alpha` is set to 1 and are nosier when `noise_alpha` is set to smaller values. Explore different values of `noise_alpha` and comment on the quality of the solution as `noise_alpha` decreases.

Problem 2.2 Mountain Car

In this problem, we consider the mountain car problem that is initially proposed in [4] and later adapted as a benchmark for RL algorithms. The goal of the mountain car problem is to drive an under-powered

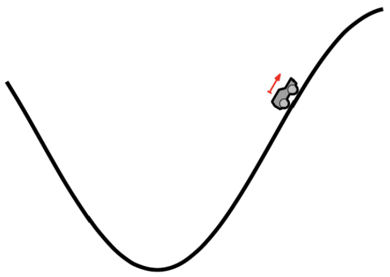


Figure 2: The mountain car problem [1]: The goal is to drive an under-powered car to the top of the hill. Since the car is under-powered, it does not have enough power to directly drive to the goal position at the right edge. It instead needs to build up momentum by swinging back and forth.

car to reach the top of a hill (Figure 2). Since the car is under-powered, it cannot drive up to the top of the hill directly; instead, it needs to swing back and forth to gain enough momentum to climb up the hill. We consider a two-dimensional state space of position x and velocity v. The position and velocity of the car are bounded between $[-1.2, 0.5]$ and $[-0.07, 0.07]$, respectively. The car's input is the acceleration a, which is bounded between $[-1, 1]$. The dynamics of the car is given by

$$\begin{aligned} v_{k+1} &= v_k + 0.001a_k - 0.0025\cos(3x_k) \\ x_{k+1} &= x_k + v_{k+1}, \end{aligned} \tag{4}$$

where k is the discrete-time index. Whenever the car reaches the position limits, its velocity is set to zero so that it remains there indefinitely. When the car reaches the top of the hill, it gets a reward of +10. Otherwise, it gets a reward of -1 for every time step in which it does not reach the top.

We will solve the mountain car problem with a classical RL approach and MPC. The main scripts for the two subcomponents of this problem are `main_p2_mc_rl.m` and `main_p2_mc_mpc.m` in `./p2_mountain_car/`. The main script for building MDP models for the mountain car problem is `create_mountain_car.m`. Follow the instructions below and comments in the scripts to complete the discretization functions in `./p2_mountain_car/discretization` and the missing part in `./p2_mountain_car/main_p2_mc_mpc.m`.

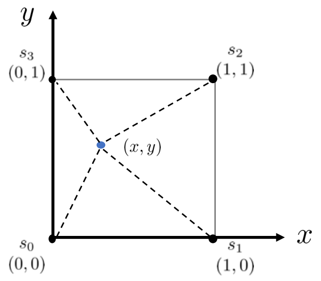
Discretization for Solving Continuous MDPs

Since the system has continuous state and action spaces, we need to discretize the system so that MDP-based RL methods can be applied. You will see that discretization is not trivial for this system. To effectively discretize the system, a stochastic discrete model should be created, even though the continuous system is deterministic. In this part, we will explore two approaches to create a discrete, stochastic MDP.

1.5 (a) *Nearest Neighbour:* One approach to discretize a continuous state space is to uniformly discretize the state space and map continuous states to the closest node in the uniform mesh. This approach is inherently deterministic but we will create a stochastic MDP by adding noise to the system.

(i) Complete the function `build_stochastic_mdp_nn.m` and run the script `create_mountain_car.m` to build a probabilistic model of the system based on the Nearest Neighbour approach. What is the stochastic element in the modeling process and what is its significance? What modeling parameters would have the most impact on the quality of the solution?

(ii) Load the MDP model and run the main script `main_p2_mc_rl.m` to solve the Mountain Car problem with the GPI algorithm. Note that you may use the GPI function developed in Problem 2.1 and make changes if necessary. Think about what the optimal solution to this task should



$$p(s' = s_0 \mid s, a) = (1-x)(1-y)$$
$$p(s' = s_1 \mid s, a) = x(1-y)$$
$$p(s' = s_2 \mid s, a) = xy$$
$$p(s' = s_3 \mid s, a) = (1-x)y$$

Figure 3: Illustration of the Linear Interpolation approach for building a stochastic transition model $p(s' \mid s, a)$. In contrast to the Nearest Neighbour approach that snaps s' to the closest node, the Linear Interpolation approach defines a distribution over four adjacent nodes based on the distance between s' and the nodes. The coordinates here are normalized between $[0,1] \times [0,1]$.

be for the Mountain Car system. Was the learning algorithm able to find this solution? If not, why do you think that is the case? Report the converged value function heatmap and the plot showing the state and input trajectories of the car.

1 (b) *Linear Interpolation:* An alternative the Nearest Neighbour approach to build a discrete stochastic MDP is to instead assign probabilities to adjacent nodes based on the distance to the continuous state [5]. An illustration is shown in Figure 3. Complete the function `build_stochastic_mdp_li.m`. Change the parameter `mdp_approach` in the script `create_mountain_car.m` correspondingly and run the script to build a stochastic MDP for the mountain car problem based on the Linear Interpolation approach. Solve the MDP with the GPI algorithm by running `main_p2_mc_rl.m` with the correct model loaded. Report the state value function heatmap and state and action trajectory plots. Briefly comment on the effectiveness of the two approaches to discretize continuous MDPs.

Model Predictive Control

We will now solve the same mountain car problem with MPC. We can formulate the mountain car problem as an optimal control problem with the following cost function

$$\min_{a_1,..,a_{N-1}} \quad (\mathbf{x}_N - \mathbf{x}_g)^T \mathbf{Q} (\mathbf{x}_N - \mathbf{x}_g) + \sum_{k=0}^{N-1} (\mathbf{x}_k - \mathbf{x}_g)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_g) + r a_k^2, \tag{5}$$

where $\mathbf{x}_k = [x_k,\ v_k]^T$ is the state, a_k is the input, $\mathbf{x}_g$ is the goal state, $\mathbf{Q}$ is a symmetric positive semi-definite matrix, r is a non-negative scalar, and N is the length of the trajectory. Note that r is typically required to be strictly positive to penalize large inputs; however, here we will set r to zero and explicitly limit the values of the inputs as constraints in our MPC formulation.

0.5 (c) Assume a prediction horizon of T timesteps. Write out the MPC optimization problem with the nonlinear system dynamics given in (4).

2 (d) Due to the number of decision variables and constraints, it can be often difficult to implement the nonlinear MPC real-time in practice. To speed up the computation, we can instead use a sequential quadratic programming (SQP) approach to solve the optimization problem at each time step more efficiently. The idea is similar to the ILQC implementation in Assignment 1. Here, we assume that we solve the nonlinear MPC problem only once at the initial time step and rollout a sequence of predicted states $\{\bar{\mathbf{x}}_{k+r}\}_{r=1}^{T}$ and inputs $\{\bar{a}_{k+r}\}_{r=0}^{T-1}$ over the prediction horizon. At each subsequent time step k, we linearize the system dynamics and quadratize the the cost around the perdition trajectory from the previous time step (i.e., $\{\bar{\mathbf{x}}_{k+r}\}_{r=1}^{T}$ and $\{\bar{a}_{k+r}\}_{r=0}^{T-1}$). Define $\delta \mathbf{x}_{k+r} = \mathbf{x}_{k+r} - \bar{\mathbf{x}}_{k+r}$ and $\delta a_{k+r} = a_{k+r} - \bar{a}_{k+r}$, formulate the Quadratic Program (QP) to be solved at each time step and complete the implementation in `main_p2_mc_mpc.m`. Note that, in typical SQP algorithms, we usually solve the QP multiple times until the solution converges or a maximum number of iteration is reached; for this problem, it suffices to solve the QP once at each time step.

(i) What is the effect of changing the length of the prediction horizon?

(ii) Test the MPC controller with noisy measurements. To do so, change `cur_state_mpc_update` in `main_p2_mc_mpc.m` from `cur_state` to `cur_state_noisy`. Try different values of `noise`. Is the car is still able to reach the goal state?

References

[1] Jonas Buchli. *Course on Optimal and Learning Control for Autonomous Robots*, April 2015. Course Number 151-0607-00L, Swiss Federal Institute of Technology in Zurich (ETH Zurich). Accessed on: Mar. 07, 2020. [Online]. Available: http://www.adrlab.org/doku.php/adrl:education:lecture:fs2015.

[2] Michael Grant and Stephen Boyd. *CVX: Matlab Software for Disciplined Convex Programming*, 2014. Webpage: http://cvxr.com/cvx.

[3] Angela Schoellig. *Remarks on the Implementation of Probabilistic Approaches*, 2015. Course Notes, University of Toronto Institute for Aerospace Studies (UTIAS). Accessed on: Mar. 07, 2020. [Online]. Available: https://drive.google.com/open?id=1yPjhVgh_4HR5lmfITxn15VxNJiiEiVF2.

[4] Andrew William Moore. *Efficient Memory-Based Learning for Robot Control*, 1990. Technical Report UCAM-CL-TR-209, University of Cambridge. Accessed on: Mar. 08, 2020. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf.

[5] Pieter Abbeel. *Solving Continuous MDPs with Discretization*, 2019. Course Slides, UC Berkeley. Accessed on: Mar. 08, 2020. [Online]. Available: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/slides/Lec3-discretization-of-continuous-state-space-MDPs.pdf.