Homework 3: Infinite Horizon Dynamic Programming

Collaboration in the sense of discussion is allowed, however, the work you turn in should be your own - you should not split parts of the assignments with other students and you should certainly not copy other students' code or papers. See the collaboration and academic integrity statement here: https://natanaso.github.io/ece276b. *Books may be consulted but not copied from.*

Submission

You should submit the following four files on **Gradescope** by the deadline shown in the top right corner.

1. **FirstnameLastname_HW3_P1-2.pdf**: upload your solutions to the theoretical problems (Problems 1-2). You may use latex, scanned handwritten notes (write legibly!), or any other method to prepare a pdf file. Do not just write the final result. Present your work in detail, explaining your approach at every step.

2. **FirstnameLastname_HW3_P3.zip**: upload the code you have written for Problem 3 in a zip file with the following structure:

   ```
   src/
   main.py
   README.txt
   ```

3. **FirstnameLastname_HW3_P4.zip**: upload the code you have written for Problem 4 in a zip file with the following structure:

   ```
   src/
   inverted_pendulum_animation.py
   main.py
   README.txt
   ```

4. **FirstnameLastname_HW3_P5.pdf**: upload the report for Problem 5. You are encouraged but not required to use an IEEE conference template[1] for your report.

Problems

1. [10 pts] A tennis player has two strategies to serve the ball: bold (B) or timid (T). The probability of B (or T) landing in bounds is p_B (or p_T, respectively). The probability of winning a point assuming the serve is in bounds is q_B (or q_T, respectively). For the strategy to make sense, we assume $p_B < p_T$ and $q_B > q_T$. Find the serving strategy in a single game in order to maximize the probability of winning that game. Formulate this as a stochastic shortest path problem and write the Bellman equation. (Tennis rules in a game: the points for each player are 0, 15, 30, or 40 and there is a chance of a second serve if the first serve does not land in bounds.)

2. [15 pts] You are enjoying flying your drone but it runs on limited battery. Suppose the state of the battery is either good (1) or bad (2). At the good state you get a reward r_1, and there is a probability p_{12} that the state goes to bad, otherwise it remains good. At the bad state you get a lower reward $r_2 < r_1$ and the battery remains at this state. At every time step, you observe the current state, collect the current reward, and decide whether to recharge or not. Recharging resets your drone to the good state but costs an amount c.

   (a) Formulate the problem as minimizing an infinite-horizon γ-discounted cost. Write the Bellman equation. (Alternatively, you can write the problem as maximizing a reward, and in this case the statements below are symmetric.)

   (b) Prove that the optimal cost-to-go function is increasing in the state of the battery, i.e., $J^*(1) \leq J^*(2)$.

[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html

(c) Prove that optimal policies are of threshold nature: if it is optimal to recharge at state 1, then it is optimal to recharge at state 2. If it is optimal to not recharge at state 2 then it is optimal to not recharge at state 1.

(d) More generally, suppose the state takes values $\{1, \ldots, n\}$ from better to worse. Your rewards r_i are decreasing in i. At each step, p_{ij} describes the probability of transitions from state i to j. The worse the state is, the more likely it is to get worse, i.e., the quantity $\sum_{j=l}^{n} p_{ij}$ is increasing in i for all $l \in \{1, \ldots, n\}$ (stochastic dominance). You have the option to reset to state 1 with cost c. Prove that the optimal cost-to-go $J^*(i)$ is increasing in i, and that optimal policies are of threshold nature: continue if $i \leq t$ and reset if $i > t$ for some threshold t.

3. [25 pts] **Programming Assignment** Implement a shortest path planner for a robot using Rapidly-Exploring Random Trees (RRTs). You are allowed to use any version of RRT or RRT* that we have covered in class. We will restrict our attention to robots that can move in any direction instantaneously and whose configuration space is $\mathbb{R}^2$.

   The input to your program is a text file called `input.txt`. The first two lines in this file specify the coordinates of the starting and the goal configuration respectively. Each line after the second will contain the coordinates of the configuration space obstacles specified as $x_1\ y_1, x_2\ y_2, \ldots$. That is, the X and Y coordinate of a C-space obstacle corner are separated by a space, and the coordinates of consecutive corners are separated by a comma. The corners are listed in the order they appear on the boundary. You will have to implement a subroutine to check if a configuration or a path in the configuration space is in collision with the configuration space obstacles.

   Run your algorithm on the maze configuration space in Fig. 1. The start and goal configurations are shown in red and green, respectively. The corresponding input file, `input_maze.txt` is provided. Another input file named `input_triangle.txt` corresponds to the solution of Problem 2 in Homework 2. Compare the cost of the shortest path found by hand in Problem 2 with the cost of the path returned by RRT.

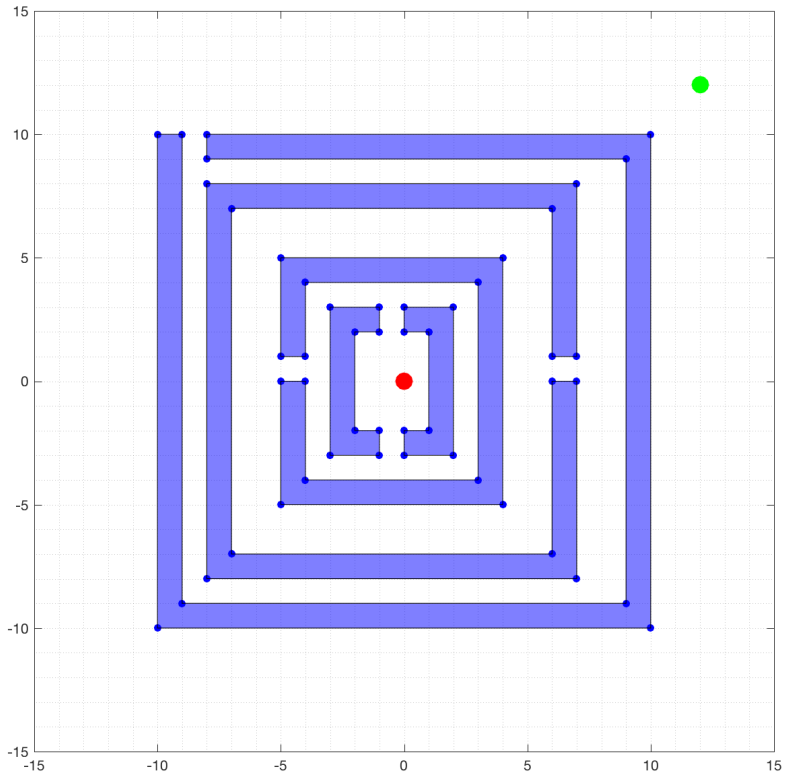

Figure 1: Configuration space for the maze.

4. [35 pts] **Programming Assignment** Our task is to solve an optimal control problem in order to balance an inverted pendulum. We will discretize the state and control spaces to formulate a finite-state MDP problem, and then solve it using value iteration and policy iteration. Consider the pendulum state $\mathbf{x} = (x_1, x_2)^T$ where x_1 is the angle of the pendulum and x_2 is the angular velocity. The continuous-time dynamics are:

$$d\mathbf{x} = f(\mathbf{x}, u)\, dt + \sigma\, d\omega \qquad f(\mathbf{x}, u) := \begin{bmatrix} x_2 \\ a \sin x_1 - b\, x_2 + u \end{bmatrix}$$

where $a > 0$ summarizes the effects of gravity, mass and length of the pendulum, $b > 0$ represents damping and friction, u is the control input and ω is Gaussian motion noise (Brownian motion). The stage-cost at each state $\mathbf{x}$ and control u is:

$$g(\mathbf{x}, u) = 1 - \exp(k \cos x_1 - k) + \frac{r}{2} u^2$$

where $k > 0$ determines the shape of the cost and $r > 0$ scales the control cost. Note that the cost penalizes x_1 that is away from 0 and is quadratic in the control u. You should formulate a Discounted infinite-horizon problem with discounted factor $0 < \gamma < 1$. Experiment with the constants $(a, b, \sigma, k, r, \gamma)$.

Next, we will formulate an MDP. First, you should decide a time step δt, maximum velocity v_{max} and maximum control u_{max} to be represented in the MDP. You can discretize the state and control spaces into (n_1, n_2, n_u) number of grid points. Be careful that x_1 is an angle and should enforce wrap-around. x_2 and u are discretized over the intervals $[-v_{max}, v_{max}]$ and $[-u_{max}, u_{max}]$. Experiment with the constants $(\delta t, n_1, n_2, n_u, v_{max}, u_{max})$. For each discrete state $\mathbf{x}$ and each discrete control u, the transition probability $p_f(\mathbf{x}'|\mathbf{x}, u)$ is Gaussian with mean $\mathbf{x} + f(\mathbf{x}, u)\, \delta t$ and covariance $\sigma\sigma^T\, \delta t$. We have to create transition probabilities in the discretized MDP to approximate the Gaussian distribution defined by the continuous dynamics. We will choose possible next states/grid points around the mean, evaluate the Gaussian at the chosen grid points, and normalize so that the outgoing transition probabilities sum up to 1 at each state. The stage cost of the MDP is $g(\mathbf{x}, u)\, \delta t$.

Generally, denser grids produce more accurate solutions. It is also useful to think about "compatibility" of the grid in the following sense. Suppose that the time step δt is small and the angular velocity discretization step is small, while the angle discretization step is large. Then, it will be very difficult for the MDP to make any state transitions, i.e., the transition probabilities will be close to delta functions centered at the current state. This is to be avoided. Instead, we should aim for

$$\frac{2 v_{max}}{n_2}\, \delta t \approx \frac{2\pi}{n_1}$$

and apply similar reasoning to the velocity and control discretization.

Now that we have formulated an MDP, we can use policy iteration and value iteration to obtain an optimal control strategy. For your choice of parameters, compare the convergence of policy and value iteration. After you solve the MDP, you have a control policy defined on the grid. Use interpolation to extend the control policy to continuous space and simulate several trajectories for the continuous system, starting at different initial states. Use `inverted_pendumlum_animation.py` to help visualize your controller and if your controller (and problem and discretization parameters) are sensible. You should see the pendulum going to the vertical state and balancing there. Gradually increase the noise and discuss its effect. At noise leves are you not able to achieve stable equilibrium in the vertical state?

5. [15 pts] Write a project report describing your approach to the two programming assignments. Your report should include the following sections:
   - **Introduction**: discuss why the problem is important and present a brief overview of your approach
   - **Problem Formulation**: state the problem you are trying to solve in mathematical terms. This section should be short and clear and should define the quantities you are interested in.
   - **Technical Approach**: describe your approach to sampling-based planning and pendulum balancing. In the former case, explain your choice of algorithm, collision checking subroutine, distance function used, nearest neighbor calculation, etc. Include plots showing the tree and paths generated by your algorithm. In the latter case, describe your MDP discretization and the policy and value iteration algorithms.
   - **Results**: present the results for both problems and discuss your choice of parameters and algorithm performance – what worked as expected and what did not and why.