MAE 242: Robot Motion Planning
Project#2

Assigned April 18, due May April 29

Homework Instructions:

This is hw2 programming part. Please upload files on gradescope with your solutions. The Collaboration Policy for this course is detailed on the syllabus and you have to provide the names of at most 2 other students you have collaborated with to do this homework.

Homework exercise:

1. (Programming problem) In this exercise, you will implement and demonstrate the performance of stochastic search algorithms in different environments. To do this, please download the folder "mdps" from canvas. In it you will find several updated files:

   - The file `gridWorld.py` plots a grid world, O, of dimensions n, m (imported from the grid files below), as well as a table in your console / terminal with the value function at each grid cell. The grid plot shows the start and final states and, given a sequence of legal states defining a path, it plots the path in the grid environment.
   - The files `smallGrid.py` and `mediumGrid.py` are Grid Worlds O of small sizes. Their format is similar to the obstacle environments in `mazemods` of the previous assignment.
   - The files `costFunction.py` implements stage cost functions.
   - The file `nextState.py` defines a Markov chain function `nextState`(x, u, η, O), to simulate the state transition. The function inputs are the current state $x = i$, a control action u which can be "up" $= (0, 1)$, "down" $= (0, -1)$, "left"$= (-1, 0)$, or "right" $= (0, 1)$, a parameter to randomize actions η, and a grid World O. In this way, with probability $q_{ij}(u) = 1 - \eta$, `nextState`(i, u, η, O) outputs $j = i + u$, and with probability $q_{ij}(u) = \eta/2$ or 0 it outputs $j = i + v$, where v is another action in the set "up", "down", "left" or "right'. For example, if the choice is "up", then the agent moves "up" with probability $1 - \eta$, otherwise it moves either "left" or "right" with probability $\eta/2$, and "down" with probability 0. Other motions can be found in the code. From here you should be able to define state transition probabilities $(p_{ij}(u))$. Note that if the transition to cell j is not legal (gets into an obstacle), then `nextState`$(i, u, \eta, O) = i$. This changes the transition probabilities as follows: the probability of staying at $x = i$ will be $p_{ij}(u) = \sum_{j:j \text{ is in obstacle}} q_{ij}(u)$.

   You have to edit and submit the file `valueIteration.py` as your coding solution to this problem. Please submit the other results answering the following questions as a pdf file.

   (a) (10 points) Using the discount parameter γ and the cost function associated with the 'small' grid, defined in `getCost` complete the function `valueIteration`, to use `smallGrid`. The iterations should use the discount parameter γ and the associated cost, defined in `getCost`. For how long you need to iterate achieve convergence error $\epsilon = 1\text{e}{-3}$ with $\gamma = 0.9$ and $\eta = 0.2$? Does the number of iterations change for different value function initializations?

(b) (10 points) Using the discount parameter γ and the cost function associated with the 'small' grid, defined in `getCost` complete the function `policyIteration`, to use `smallGrid`. The iterations should use the discount parameter γ and the cost function defined in `getCost`. For how long you need to iterate until convergence with $\gamma = 0.9$ and $\eta = 0.2$? Is the policy the same as in part (a)?

(c) (5 points) Using `mediumGrid` and the standard `getCost` function, play with the discount factor γ and the default noise of η in the Markov chain, to obtain different results in this environment with `getCost` function. Can your optimal policy from value iteration go through the cell next to the start cell, given by (2,2)?

(d) (15 points) Consider the `mediumGrid` layout and the `getCostBridge` function. This grid has two terminal states with positive payoff at (5,3) and at (3,3). The bottom row of the environment have states with negative payoff -10 (cliff states). The starting state is at (1,2). We call the state $(2, 2)$ the "bridge" between the start and the cliff states. We distinguish between two types of paths: (1) paths that "risk the cliff" and travel near the bottom row of the grid; these paths are shorter but risk earning a large negative payoff. (2) Paths that "avoid the cliff" and travel along the top edge of the grid. These paths are longer but are less likely to incur huge negative payoffs (or large costs.)

Give an assignment of parameter values for the discount γ, and noise value η in the actions which produce an optimal policy that results in the following paths:

1) Prefer the close exit (with cost -1), risking the cliff (with cost +10)
2) Prefer the close exit (with cost -1), but avoiding the cliff (with cost +10)
3) Prefer the distant exit (with cost -10), risking the cliff (with cost +10)
4) Prefer the distant exit (with cost -10), but avoiding the cliff (with cost +10)

Programming Instructions:

- You will fill in portions of valueIteration.py for solving this assignment. You can add more functions within this file. You should submit valueIteration.py with your code and comments. Please do not change the other files in this distribution or submit any of our original files other than valueIteration.py.
- The appropriate function calls used for evaluating your functions are:
  1. `value, policy, iteration = valueIteration(gamma,cost,eta,gridname)`,
  2. `value, policy, iteration = policyIteration(gamma,cost,eta,gridname)`,
  3. `gamma, eta = optimalValues(question)`,

  where the outputs are in the order of numpy array of values, policy and number of iterations for convergence with error 1e-3. The inputs include discount factor γ, probability η, cost type ('cost' or 'bridge'), and grid type ('small' or 'medium'). The test cost and grid is meant for code evaluation. The `optimalValues` function holds your answer for discount factor and probability for each of the questions. Please modify the values as you see fit. The outputs `values` and `policy` are $n \times m$ sized arrays. The `policy` variable should consist of numbers $\{0, 1, 2, 3\}$ corresponding to the moves: right $\{(1, 0)\}$, top $\{(0, 1)\}$, left $\{(-1, 0)\}$ and down $\{(0, -1)\}$ respectively.
- Your code will be verified for basic technical correctness. Please do not change the names of any provided functions, classes or file names, as this will confuse the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. The autograder acts primarily as a filter. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.
- Getting Help:
  1. Some sample code has been provided towards the end of the search file to understand the use of functions. Please start early on the coding assignments. If you are stuck on something, please use office hours and Piazza.
- The autograder currently considers numpy and matplotlib libraries. If you are using other libraries, please email the TA to add those Python libraries in the autograder environment.