[AI18] Assignment 4
Jagadish Bapanapally

Consider the gird world search problem in Figure 1. Each state is generally indexed by its coordinate (i, j), where i is row and j is column. There are two terminal states (1, 3) and (2, 3). State (1,3) has reward -1, state (2,3) has reward +1, and the rest states have reward -0.04. Fix discount factor $\gamma = 0.8$. At each state, an agent can take one action from {up, down, left, right}; the actions will be encoded as 0 for up, 1 for down, 2 for left and 3 for right. As the environment is stochastic, the agent will not always end up in the state aimed by the executed action; with probability 0.8 it will reach the targeted state, but with probability 0.1 it will be perturbed right and with another probability 0.1 it will be perturbed left, as shown in Figure 1 (right). If the agent hit the wall or the rock at (1,1), it will stay at its original state.

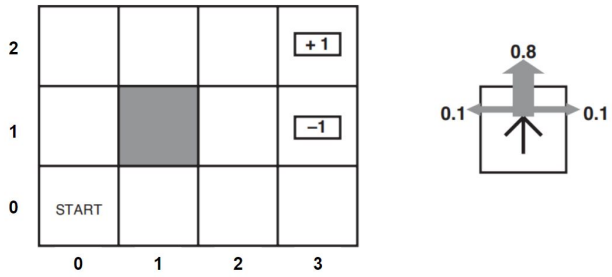

Fig. 1. A Grid World Search Problem

You are asked to implemented a set of popular algorithms from scratch for Markov Decision Process (MDP) and Reinforcement Learning (RL) based on the above context. A few functions and variables are already implemented for you; feel free to use them (but not any MPD or RL library) to speed up implementation.

[1] Policy evaluation (P.E.) is the task of estimating state utility associated with some policy. You are given a policy and asked to implement P.E. algorithms to estimate its associating utilities of the non-terminal states.

[1.1] In `LastName_PolicyEval_Linear.py`, implement P.E. with the assumption that the transition probabilities based on the policy is known – so we will solve linear equations to estimate utilities. [1]

[1.2] In `LastName_PolicyEval_MC.py`, implement P.E. with the assumption that the environment is unknown – so we will implement the Monte Carlo method from scratch to estimate utilities.[2]

Once done, document your utility estimates in Table 1 and send me your codes.

Table 1. Estimate of State Utility

Method	U(0,0)	U(0,1)	U(0,2)	U(0,3)	U(1,0)	U(1,2)	U(2,0)	U(2,1)	U(2,2)
Linear	0.70530822	0.65530822	0.61141553	0.38792491	0.76155822	0.66027397	0.81155822	0.86780822	0.91780822
MC	0.70132	0.65812	0.61384	0.40404	0.76452	0.6496	0.80576	0.86252	0.9156

[1] You can use existing Python libraries to solve linear equations.
[2] You can decide the number of trials for each estimation.

[2] You are given the state utilities and asked to implement an algorithm to identify a corresponding optimal policy based on the Bellman equation, using template `LastName_OptimalPolicy.py`. Once done, document your identified policies (up, down, left, right) in Table 2 and send me your code.

Table 2. Optimal Policy Identified based on the Bellman Equation

State	U(0,0)	U(0,1)	U(0,2)	U(0,3)	U(1,0)	U(1,2)	U(2,0)	U(2,1)	U(2,2)
Policy	0	2	2	2	0	0	3	3	3

[3] You are given an environment and asked to implement the policy iteration algorithm to find an optimal policy for it, using template `LastName_PolicyIteration.py`.

[3.1] Plot the estimates of all state utilities versus iteration number. Figure 2 is an example[3].

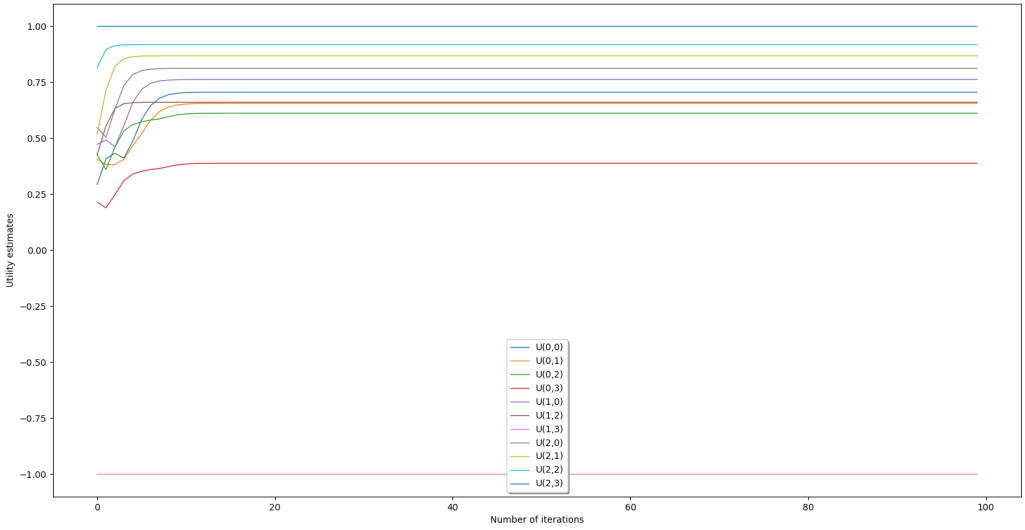

Fig. 2. State Utility Estimates versus Number of Iterations

[3.2] Document your identified optimal policies in Table 3 and send me your code.

Table 3. Optimal Policy Identified by the Policy Iteration Algorithm

State	U(0,0)	U(0,1)	U(0,2)	U(0,3)	U(1,0)	U(1,2)	U(2,0)	U(2,1)	U(2,2)
Policy	0	2	2	2	0	0	3	3	3

[3] However this does not mean your curves need to look like those in Fig 2.

[4] You are given an unknown environment and asked to implement the temporal different learning algorithm to identify its state utilities, using template `LastName_TDL.py`. You can fix the learning rate α[4]. When selecting an action at a state, use the GLIE strategy that selects a random action with probability $\frac{1}{t}$ and the optimal action with probability $1 - \frac{1}{t}$. Fix $t = 2$ when reporting results.

[4.1] Plot two figures. One is the utility estimate versus the number of trials (similar to Figure 3 left), and the other is the RMS error of the utility estimate (similar to Figure 3 right).[5]

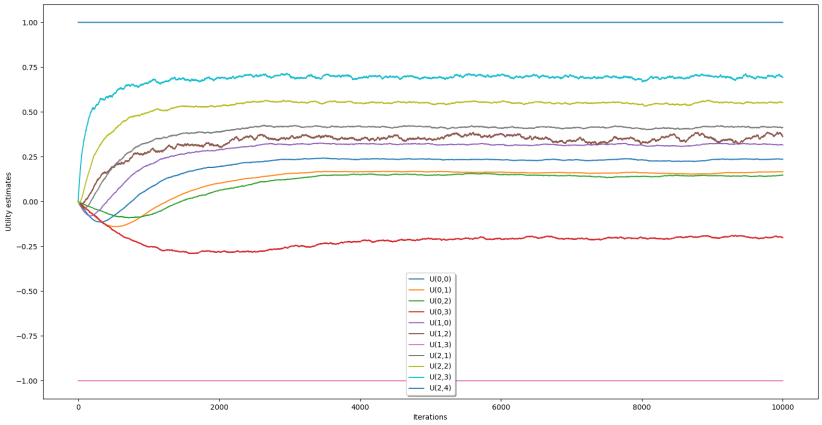

Fig. 3.

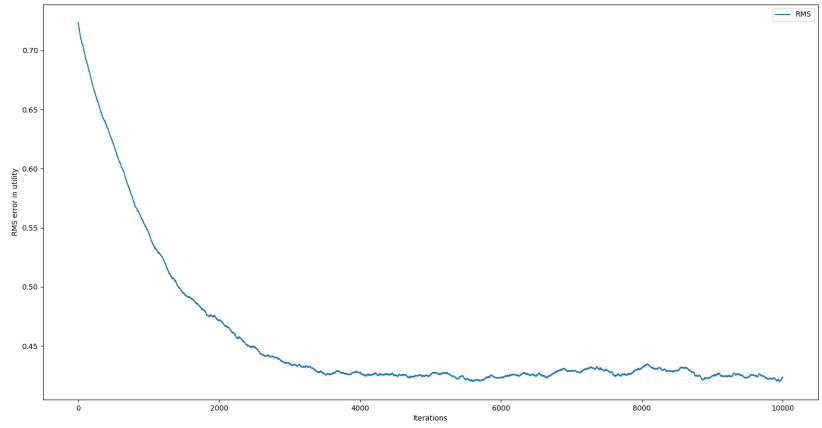

Fig. 4.

[4] However feel free to design an adaptive α that will decrease as visit frequency increases

[5] RMS is the rooted mean squared error between your estimated utilities and the true utilities.

[4.2] Document your estimated utilities in Table 4 and send me your code.

Table 4. State Utilities Estimated by Temporal Difference Learning

State	U(0,0)	U(0,1)	U(0,2)	U(0,3)	U(1,0)	U(1,2)	U(2,0)	U(2,1)	U(2,2)
Utility	0.23641003	0.16634668	0.14734012	-0.2011803	0.31604173	0.36413042	0.41389792	0.55319623	0.69345829

[5] You are given a number of observed state utilities and asked to implement a utility function approximation algorithm for predicting those unobserved utilities, using template `LastName_FuncApprox.py`. You will use the linear regression function for approximation, and need to design at least five features (including two features that are two coordinates of a state). Once done, document your designed features in Table 5 and your prediction errors[6] in Table 6 and send me your code.

Table 5. Feature Design

$f_1(s)$	first coordinate of state s
$f_2(s)$	second coordinate of state s
$f_3(s)$	reward + utility of the state
$f_4(s)$	distance to (2,3)
$f_5(s)$	distance to (1,3)

Table 6. Prediction Errors of State Utilities

State	U(0,1)	U(0,2)	U(1,0)	U(2,0)	U(2,2)	Mean
Error	0.05362785	0.11303595	0.20574556	0.01339605	0.14996531	0.10715414231440101

[6] The error at a state is the absolute difference between your predicted utility and the true utility of that state.