COMP3702 Artificial Intelligence
Semester 2, 2022
Tutorial 7 - Sample Solutions

Exercise 7.1

Note that for this question, once the agent arrives on a square with a value, it has only one action available to it, that is, to *exit* the environment. The rewards stated on the squares with values are the reward for *exiting* the square and the environment, so the reward is not repeatedly earned. For example, $R(s_5, exit) = 10$, and $R(s_{4b}, exit) = 0$ (where s_{4b} is the square below s_4).

a) The value of the policy π_R, which is to always move right when possible, is:

s0	*s1*	*s2*	*s3*	*s4*	*s5*
5	$10\gamma^4p^2$	$10\gamma^3p^2$	$10\gamma^2p^2$	$10\gamma p$	10
			0	0	

You could write this as:

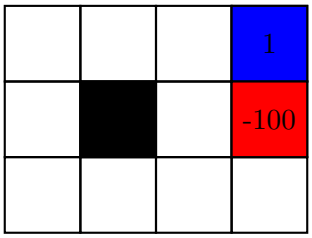
$$V^{\pi_R}(s_1) = 10\gamma^4p^2$$
$$V^{\pi_R}(s_2) = 10\gamma^3p^2$$
$$V^{\pi_R}(s_3) = 10\gamma^2p^2$$
$$V^{\pi_R}(s_4) = 10\gamma p$$

b) The value of the policy π_L, which is to always move left when possible, is:

s0	*s1*	*s2*	*s3*	*s4*	*s5*
5	5γ	$5\gamma^2$	$5\gamma^3p$	$5\gamma^4p^2$	10
			0	0	

Gridworld

Consider the gridworld below:



States in this environment are the positions on the tiles. The world is bounded by a boundary wall, and there is one obstacle, at (1,1) (using python or zero indexing starting from the bottom left corner). In addition, there are two terminal states, indicated by the coloured squares.

Terminal states: In practice, we would say that the two coloured tiles are *terminal states* of the MDP. For mathematical convenience, we often prefer to deal with an infinite horizon MDP (see more below). To convert this model to an infinite horizon MDP, we will add an extra pseudo-state to the list of MDP states called `'EXIT_STATE'`, which is absorbing (the agent cannot leave it irrespective of its action, ie. $T(EXIT\_STATE, a, EXIT\_STATE) = 1$ for all a).

Actions and Transitions: In this world, an agent can in general choose to move in four directions — *up*, *down*, *left* and *right*, which we might sometimes refer to as `^`, `v`,`<` and `>`, respectively. However, the agent moves successfully with only $p = 0.8$, and moves perpendicular to its chosen direction with $p = 0.1$ in each perpendicular direction. If it hits a wall or obstacle, the agent stays where it is. Denote this action `'.'`. In addition, once the agent arrives on a coloured square with a value, it has only one special action available to it; that is, to *exit* the environment and move to the exited state E; denote this action `'e'`.

Rewards: The values stated on the coloured squares are the reward for *exiting* the square and the environment, so the reward is not repeatedly earned; that is, $R([3, 2], exit) = 1$, and $R([3, 1], exit) = -100$. All other states have 0 reward.

Discount factor: $\gamma = 0.9$.

Exercise 7.2

a) : See `grid_world_solution.py`.

b) : What is the one-step probability of arriving in each state s' when starting from (0,0) for each a, i.e what is $P(s'|a, (0, 0)) \; \forall \; a, s'$?
These can be worked out by hand, or by calling `get_transition_probabilities((0, 0), action)` for each action.

$P((1, 0)|right, (0, 0)) = 0.8$
$P((0, 1)|right, (0, 0)) = P((0, 0)|right, (0, 0)) = 0.1$

$P((0, 1)|up, (0, 0)) = 0.8$
$P((1, 0)|up, (0, 0)) = P((0, 0)|up, (0, 0)) = 0.1$

$P((0, 0)|left, (0, 0)) = 0.9$
$P((0, 1)|left, (0, 0)) = 0.1$

$P((0, 0)|down, (0, 0)) = 0.9$
$P((1, 0)|down, (0, 0)) = 0.1$

All other transition probabilities are zero.

c) : What is the probability of arriving in state (1,0) from each a and s, i.e what is $P((1,0)|a,s) \; \forall \; (a,s)$?

These can be worked out by hand, or using a call to `get_transition_probabilities` in a loop over all state/action pairs.

$P((1,0)|right,(0,0)) = 0.8$
$P((1,0)|up,(0,0)) = 0.1$
$P((1,0)|down,(0,0)) = 0.1$

$P((1,0)|left,[2,0]) = 0.8$
$P((1,0)|up,[2,0]) = 0.1$
$P((1,0)|down,[2,0]) = 0.1$

$P((1,0)|up,(1,0)) = 0.8$
$P((1,0)|down,(1,0)) = 0.8$
$P((1,0)|left,(1,0)) = 0.2$
$P((1,0)|right,(1,0)) = 0.2$

All other transition probabilities are zero.

Exercise 7.3

See `grid_world_solution.py` for the VI code used to generate these solution.

a) : What is the value function estimate after 4 iterations? What is the policy according to the value function estimate after 4 iterations?

Values after iteration 4 (rounded to 4 decimal places)
(0, 0) 0.0
(0, 1) 0.0
(0, 2) 0.3732
(1, 0) 0.0
(1, 2) 0.6584
(2, 0) 0.0467
(2, 1) 0.1173
(2, 2) 0.7965
(3, 0) 0.0
(3, 1) -100.0
(3, 2) 1.0
(-1, -1) 0.0

Policy after iteration 4
(0, 0) UP
(0, 1) UP
(0, 2) RIGHT
(1, 0) UP

(1, 2) RIGHT
(2, 0) UP
(2, 1) LEFT
(2, 2) RIGHT
(3, 0) DOWN
(3, 1) UP (any action results in exit)
(3, 2) UP (any action results in exit)
(-1, -1) UP (any action results in exit)

b) : What is the value function estimate after 10 iterations? What is the policy according to the value function estimate after 10 iterations?

Values after iteration 10 (rounded to 4 decimal places)
(0, 0) 0.4491
(0, 1) 0.5362
(0, 2) 0.6163
(1, 0) 0.3678
(1, 2) 0.7155
(2, 0) 0.2805
(2, 1) 0.2860
(2, 2) 0.8174
(3, 0) 0.0523
(3, 1) -100.0
(3, 2) 1.0
(-1, -1) 0.0

Policy after iteration 10
(0, 0) UP
(0, 1) UP
(0, 2) RIGHT
(1, 0) LEFT
(1, 2) RIGHT
(2, 0) LEFT
(2, 1) LEFT
(2, 2) RIGHT
(3, 0) DOWN
(3, 1) UP (any action results in exit)
(3, 2) UP (any action results in exit)
(-1, -1) UP (any action results in exit)

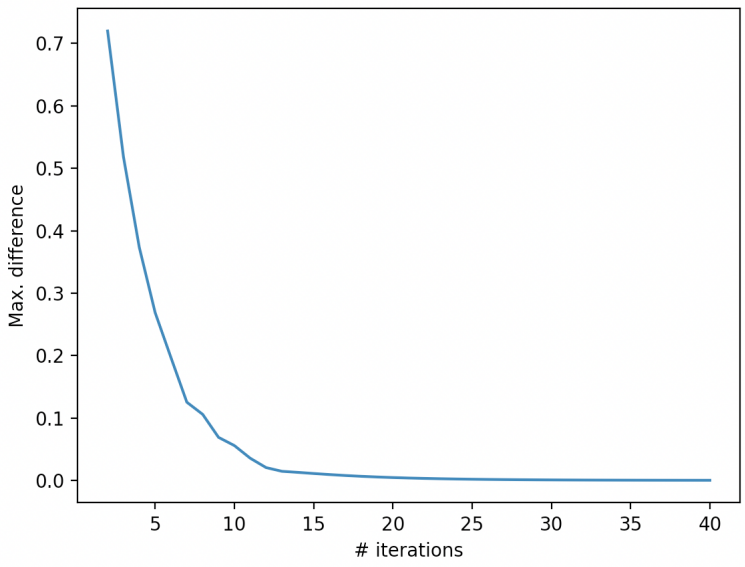
c) Run value iteration until the maximum change in values is less than EPSILON (i.e. the values have 'converged'). At this point, what is the value function estimate? What is the policy according to the value function?

Values after iteration 40 (rounded to 4 decimal places)
(0, 0) 0.4800
(0, 1) 0.5540

(0, 2) 0.6310
(1, 0) 0.4215
(1, 2) 0.7282
(2, 0) 0.3717
(2, 1) 0.3860
(2, 2) 0.8294
(3, 0) 0.1756
(3, 1) -100.0
(3, 2) 1.0
(-1, -1) 0.0

Policy after iteration 40
(0, 0) UP
(0, 1) UP
(0, 2) RIGHT
(1, 0) LEFT
(1, 2) RIGHT
(2, 0) LEFT
(2, 1) LEFT
(2, 2) RIGHT
(3, 0) DOWN
(3, 1) UP (any action results in exit)
(3, 2) UP (any action results in exit)
(-1, -1) UP (any action results in exit)

d) On each iteration, print the iteration number and the largest difference in the value function estimate for any state. What do you observe in the results?



Exercise 7.4

a) See `grid_world_solution.py` for code.

b) Let $P^\pi \in R^{|S|\times|S|}$ be a matrix containing probabilities for each transition under the policy π, where:

$$P^\pi_{ij} = P(s_{t+1} = j \mid s_t = i, a_t = \pi(s_t))$$

What is the size of P^π in this gridworld, when the special pseudo-state `EXIT_STATE` is included?

If you exclude the obstacle at (1,1) as an (unreachable) state, but include `EXIT_STATE`, then $dim(P^\pi) = 12 \times 12$. (Also, you could include the obstacle, so that $dim(P^\pi) = 13 \times 13$, but the transition probability to (1,1) is 0 from all states.)

c) Set the policy to move *right* everywhere, $\pi(s) = \text{RIGHT}\ \forall\ s \in S$. Calculate the row of P^{RIGHT} corresponding to an initial state at the bottom left corner, (0,0) of the gridworld.

Let the state indices be ordered:
$((0,0),(1,0),(2,0),(3,0),(0,1),(2,1),(3,1),(0,2),(1,2),(2,2),(3,2),\texttt{EXIT\_STATE})$

The row is: $P^{\text{RIGHT}}_{((0,0),j)} = \begin{bmatrix}0.1 & 0.8 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$

d) Write a function that calculate the row of P^{RIGHT} corresponding to an initial state at the bottom left corner, (0,0) of the gridworld for any action or deterministic $\pi((0,0))$.

Reuse code answers from Exercise 7.3 to compute these values.

e) Turn this into a function that computes P^π for any deterministic π. Note that

$$P([3,1],a,\text{EXIT\_STATE}) = P([3,2],a,\text{EXIT\_STATE}) = P(\text{EXIT\_STATE},a,\text{EXIT\_STATE}) = 1$$

for all a, so the rows of P^π for these states s are all zeros except for a 1 corresponding to the transition to $s' = \text{EXIT\_STATE}$.

As above, reuse code answers from Exercise 7.3 to compute these values.

f) Compute

$$V^{\text{RIGHT}} = (I - \gamma P^{\text{RIGHT}})^{-1} r.$$

To do this, define r as the reward for landing on a square, which holds because $R(s,a,s') = R(s)$ for the red and blue squares at [3,1] and [3,2], respectively. (*Hint*: Rather than explicitly computing the matrix inverse, you may want to use `numpy.linalg.solve()` for large $|S|$ as it implements the LU decomposition to solve for V^π using forward and backward substitution, so can result in a big speed-up.)

Using the same indexing as above: $r = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & -100 & 0 & 0 & 0 & 1 & 0\end{bmatrix}^T$ (i.e. transposed to a column vector).

Using your pre-defined P^π, code to solve for v is:

```
import numpy as np
# you should already have these defined
size_S = len(P)
discount_factor = 0.9
```

```
A = np.identity(size_S) - discount_factor*np.array(P)
r = np.array([0.0 for s in range(size_S)])
r[7]  = -100
r[11] = 1
v = np.linalg.solve(A, r)
```

g) Replace your iterative approach to policy iteration with a linear algebra approach using what you have done over steps b-f.

See `grid_world_solution.py` for code.

f) How many iterations does PI take to converge?

If it takes more than 5 iterations, you are doing something wrong!

How long does each iteration take? It depends on the machine.

Exercise 7.5

Derive V^π from the MDP objective function.

Start with:

$$V(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

Expand and collect:

$$V(s_0) = \mathbb{E}\left[R(s_0, a_0) + \gamma\left(R(s_1, a_1) + \gamma\left(R(s_2, a_2)\ldots\right)\right)\right]$$

Introduce a recursion of V:

$$V(s_0) = \mathbb{E}\left[R(s_0, a_0) + \gamma V(s_1)\right]$$

Evaluate the expectation using $\sum_{s_1} P(s_1 \mid s_0, a_0)$, and recalling $R(s, a) = \sum_{s'} P(s' \mid s, a) R(s, a, s')$:

$$V(s_0) = \sum_{s_1} P(s_1 \mid s_0, a_0)\left[R(s_0, a_0, s_1) + \gamma V(s_1)\right]$$

Introduce the policy π, such that $a = \pi(s)$:

$$V^\pi(s_0) = \sum_{s_1} P(s_1 \mid s_0, \pi(s_0))\left[R(s_0, \pi(s_0), s_1) + \gamma V^\pi(s_1)\right]$$

... and drop the time index (ie. set $s_0 = s$ and $s_1 = s'$):

$$V^\pi(s) = \sum_{s'} P(s' \mid s, \pi(s))\left[R(s, \pi(s), s') + \gamma V^\pi(s')\right]$$