Advanced Topics on Robotics
Homework 2

Alexandre M. F. Dias
Department of Electrical and Computer Engineering
Instituto Superior Técnico - University of Lisbon
alexandre.f.dias@tecnico.ulisboa.pt

1 Introduction

The problem at hand is a finite Markov Decision Process (finite MDP), as the state, action and reward sets are finite sets. In Fig. 1, the problem is illustrated, according to [1]. It consists of 60 states, which correspond to the 4 possible orientations (north, south, west, and east) in each of the 15 rooms. One of the rooms (room 15) corresponds to the 4 goal states (states 57 to 60) and 4 of the rooms contain landmarks (visible to the robot when facing south in a room north of a landmark room). There are 5 possible actions in each room (stay in place, move forward, turn right, turn left, turn around). There are also 21 observations, which correspond to i) each possible combination of the presence of a wall in each of the 4 orientations, ii) the four landmarks and iii) reaching each of the 4 goal states.

Tabular solution methods are those where the state and action spaces are small enough so that value functions can be represented as arrays or tables. In this homework, three tabular solution methods for solving finite MDPs are explored. The first two, Policy Iteration and Value Iteration, are Dynamic Programming (DP) methods and on-policy methods, while the other one, Q-learning, is a Temporal-Difference (TD) method and an off-policy method. DP methods require a complete model of the environment (they are model based), while TD methods require no model (they are model free). Although Policy Iteration was not required to be experimented with, it is explored as its results provide a baseline to the other methods (optimal policy and optimal value function). Under certain conditions, all of these methods are guaranteed to converge to the optimal solution, i.e., the value function converges to the optimal value function, with the corresponding policy or policies being optimal[1].

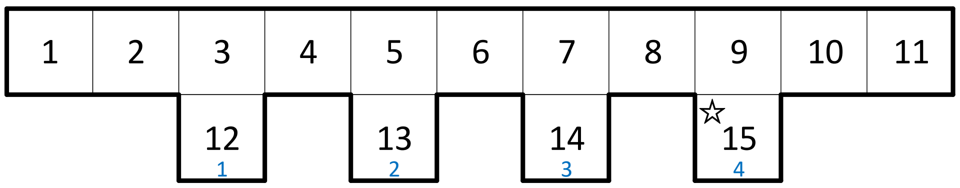

Figure 1: The problem at hand: 60 states, corresponding to 15 rooms in which the robot can have 4 possible orientations (facing north, south, west and east); the room marked with a star corresponds to the 4 goal states (states 57 to 60) and the rooms numbered from 1 to 4, in blue, contain a landmark each.

[1]A value function may correspond to one policy or several distinct policies, and in the same way an optimal value function may correspond to one optimal policy or several distinct optimal policies.

2 Full observability: Value Iteration

The update operation performed in Value Iteration combines a *truncated* Policy Evaluation — a single sweep through all non-goal states $s \in \mathcal{S}$ — and Policy Improvement:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_k(s')], \forall s \in \mathcal{S}, \forall s' \in \mathcal{S}^+. \tag{1}$$

For details on Policy Evaluation, Policy Improvement and the Policy Iteration algorithm the reader is referred to Section A of the appendix. Sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_* (either $\gamma < 1$ or termination is guaranteed from all states).

The Value Iteration algorithm is implemented according to Algorithm 2, which is a slightly modified version of that in Section 4.4 of [2] — similarly to Policy Iteration (Algorithm 1), it also uses the relative change in the value function norm as convergence criterion instead of the maximum absolute component change of the value function.

In Fig. 2, the results of running Value Iteration are shown. The upper chart of Fig. 2 depicts the policy difference[2] between the policy corresponding to current value function in Value Iteration and the optimal policy previously determined in Policy Iteration (Table 2).

The middle chart of Fig. 2 depicts the evolution of the relative change in the value function norm (convergence criterion). When comparing both upper and middle charts of Fig. 2, one can easily observe that the policy underlying the value function converges to the optimal policy much earlier than the value function is considered to have converged to the optimal value function — the policy is already optimal after 9 iterations, with the convergence criterion being of the order of 10^{-2}, while convergence of the value function is only attained after 43 iterations ($\theta = 10^{-6}$).

The lower chart of Fig. 2 shows how the value function evolves throughout iterations for each state. One can observe that the value function does not increase monotonically for each state as the algorithm is executed. The reason for this is that a *truncated* rather than *full* Policy Evaluation — as in the Policy *Iteration* algorithm — is employed. On the other hand, the value function converges to the optimal value function, which can be verified by comparing the line corresponding to the last iteration (iteration 43) with Fig. 7.

3 Full observability: Q-learning

Two sets of experiments were carried out to explore the Q-learning algorithm. The first set of experiments consisted in trying values for α_n and ε such that convergence was ensured, while in the second set of experiments there was no convergence guarantee, although convergence could still be attained through proper choice of values for α_n and ε. In the Q-learning algorithm, the update rule is

$$Q(s,a) \leftarrow Q(s,a) + \alpha_n \left[r + \gamma \max_a Q(s',a) - Q(s,a)\right]$$

To ensure convergence, it is required that all state-action pairs continue to be updated, i.e., in the limit (infinite algorithm steps) each state-action pair is taken an infinite number of times; it is also required that the sequence of step-size values α_n fulfills two conditions:

$$\sum_{n=1}^{\infty} \alpha_n = \infty \tag{2}$$

$$\sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{3}$$

[2]For details on the policy difference evaluation metric, see Section B of the Appendix

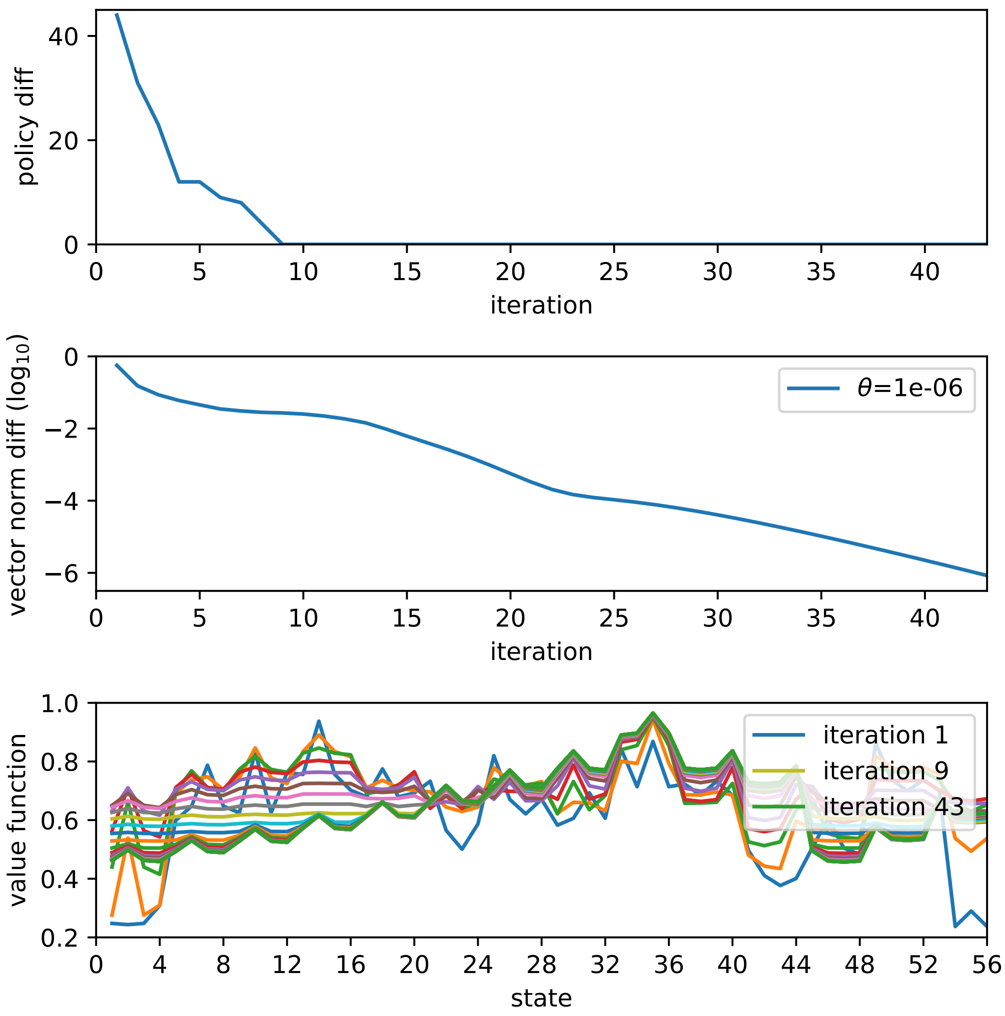

Figure 2: Policy difference $\Delta\pi$, relative change in the value function norm (convergence criterion) and state value function $V(s)$ for the execution of the Value Iteration algorithm ($\theta = 10^{-6}$).

According to these conditions, in the first set of experiments[3] $\alpha_n = \alpha/n$ for some $\alpha > 0$, in order to ensure convergence, and $\varepsilon > 0$ to ensure that each state-action pair is, in the limit, taken an infinite number of times. In the second set of experiments, $\alpha_n = \alpha$, for some $\alpha > 0$, thus convergence is not ensured as condition (3) is not fulfilled.

In Fig. 3, the two tables summarize the number of episodes (upper cell value) and total number of steps (lower cell value) necessary until convergence to the optimal *policy*, for distinct combinations of values for α and ε in each of the two sets of experiments. The number of steps needed until convergence is considered a proxy for the computational effort required. In Fig. 3.(b), the designation *oscill.* conveys that, although the optimal policy is attained, during the remainder of the algorithm execution the policy underlying the state-action value function $Q(s, a)$ *oscillates* between the optimal policy and non-optimal policies.

In Figs. 5 and 6, the evolution of the Q-learning algorithm is illustrated for a set of values achieving the optimal policy within a relatively low total number of steps in each of the sets of experiments

[3]Although conditions (2) and (3) refer to *each* state-action pair, in the simulations carried out $\alpha_n = \alpha/n$ refers to the *overall* total number of steps, not to the total number of steps of a *particular* state-action pair; this still ensures that both conditions are met and has proved in simulations to converge *faster* to the true state-action value function $Q(s, a)$ and for a *broader* range of values for α and ε.

α \ ε	0.7	0.8	0.9	1.0
2000	2674	280	65	16
	144791	29683	15584	**15483**
1000	no	76	64	29
	convergence	**10569**	14488	32441
500	464	142	78	37
	27281	**17031**	19892	40762
200	3463	279	132	78
	191443	**29660**	30889	72484
100	no	no	no	178
	convergence	convergence	convergence	145952

(a) Convergence ensured

α \ ε	0.5	0.6	0.7	0.8	0.9	1.0
0.01	3041	3775	1302	774	361	76
	92578	146477	75894	77268	78162	**68131**
0.02	1489	796	359	476	108	41
	46102	34178	**24795**	49618	25306	44911
0.05	4744	3340	270	156	102	32
	133808	123588	18418	**18075**	23666	35714
0.10	2421	2005	760	151	63	28
oscill.	72555	76703	42724	18102	**14482**	32441
0.20	1303	738	517	299	44	36
oscill.	41303	32450	29838	30374	**11178**	40762

(b) Convergence not ensured

Figure 3: Number of episodes (upper cell value) and number of steps (lower cell value) necessary for convergence to the optimal policy, for distinct combinations of hyperparameter values in the two sets of experiments.

(convergence ensured and convergence not ensured, respectively). In the figures, the policy difference, the root mean squared error (RMSE) of the norm difference between $Q(s, a)$ and $Q_*(s, a)$ (averaged over states), the relative change in the norm of $Q(s, a)$ and the number of steps in each episode are shown.

Similarly to the Value Iteration algorithm, it can be observed in both Figs. 5 and 6 that in the Q-learning algorithm the value function (now the state-action value function, instead of the state value function) also takes much longer to converge than what is required for the policy underlying the value function to converge to the optimal policy. This is a downside of Q-learning: although convergence to the optimal value function can be ensured, the computation to achieve so is too expensive — one possibility is to rely on the stabilization of the change in the state-action value function, as shown in both figures, and even then care must be taken (such a metric oscillates by over one order of magnitude).

In both sets of experiments, Q-learning benefits greatly from exploration in the problem at hand, as convergence is more quickly achieved with relatively high values of ε (greater or equal than 0.5). Moreover, even a completely random choice of actions ($\varepsilon = 1.0$) for the update of $Q(s, a)$ allows achieving convergence within a reasonable total number of steps. This shows that it is very important that each state-action pair keeps being visited as the algorithm executes and not just the state-action pairs considered optimal.

4 Partial observability: point-based Value Iteration

In this section, the problem at hand is solved in the POMDP framework, i.e., instead of assuming full state observability as before, partial observability is considered, with observations being used to update belief states (probability distributions over the set of possible states). Therefore, additionally to the uncertainty associated with action outcome (transition model), there is now also uncertainty associated with observation of the state (observation model).

In POMDPs, the (optimal) value function is piecewise linear and convex. Exact value iteration algorithms for POMDPs exist, however the computation of exact solutions for POMDPs is, in general, an intractable problem. For this reason, approximate solution techniques using a sample set of points from the belief space have been devised. Perseus is a randomized point-based approximate value iteration algorithm for solving POMDPs. The algorithm allows approximating the value function with a relatively small number of vectors (relative to the size of the belief points set).

In Table 1, the results of running the Perseus algorithm and sampling statistics for different belief points set sizes are summarized. In Fig. 4, further results of the Perseus algorithm execution and sampling statistics for different belief points set sizes are illustrated. In the upper chart, the state value function approximation, computed as the mean[4] of the cumulative reward over all the simulation runs per non-goal state, is depicted. The lower chart conveys how the algorithm time complexity varies with the size of the belief points set.

The upper chart of Fig. 4 indicates that even a fairly small belief points set size of $|\mathcal{B}| = 100$ already provides a very good approximation of the value function $V(s)$, with the number of vectors defining such approximation being relatively small ($|V| = 36$), while also showing that a size of $|\mathcal{B}| = 10$ is clearly insufficient to attain a good approximation. Naturally, since in the POMDP setting the state is not fully observable and the agent must rely on observations, the value of each state can never exceed that of the MDP setting, and in the problem at hand is typically quite lower than the value in the fully observability case.

Regarding algorithmic complexity, Table 1 shows that the number of approximating vectors is typically of one order of magnitude below that of the belief points set size. The lower chart of Fig. 4 conveys that an increase of one order of magnitude in the belief points set size translates into a increase of less than one order of magnitude (0.60 to 0.70, in $\log_{10}$ scale). Therefore, both memory and timewise, the algorithm execution scales well with the belief points set size.

Belief points set size, $\log_{10}|\mathcal{B}|$	Value function vector set size, $|V|$	Iterations	Value iteration time (s)	Success rate (%)
1	9	30	4	73
2	36	72	13	100
3	192	94	53	100
4	706	94	259	100

Table 1: Execution results of the Perseus algorithm and sampling statistics for different belief points set sizes (convergence criterion of 10^{-3}). The sampling leading to the success rate and discounted cumulative reward metrics was carried out over 1000 runs for each of the 56 non-goal states as start state.

5 Conclusions

In this homework, experimentations considering full and partial observability in the problem at hand were carried out.

For full observability, the Value Iteration algorithm learned the optimal policy and converged to the optimal value function through backup operations, resorting to the transition model. The Q-learning algorithm learned the optimal policy through experimentation (simulated state transitions), without knowledge of the transition model itself. In Q-learning, there is a clear interplay between the step size α and the amount of exploration provided by the ε-greedy policy behavior in the attainment of the optimal policy. Also, even if convergence is not ensured by the values taken by the step size, convergence can still be achieved.

For partial observability, the Perseus randomized point-based approximate value iteration algorithm was able to attain successful policies with relatively low computational effort, even lower than that

[4]The reason for this is the state value function being defined as an expectation, and therefore the mean is an estimate of the expectation.

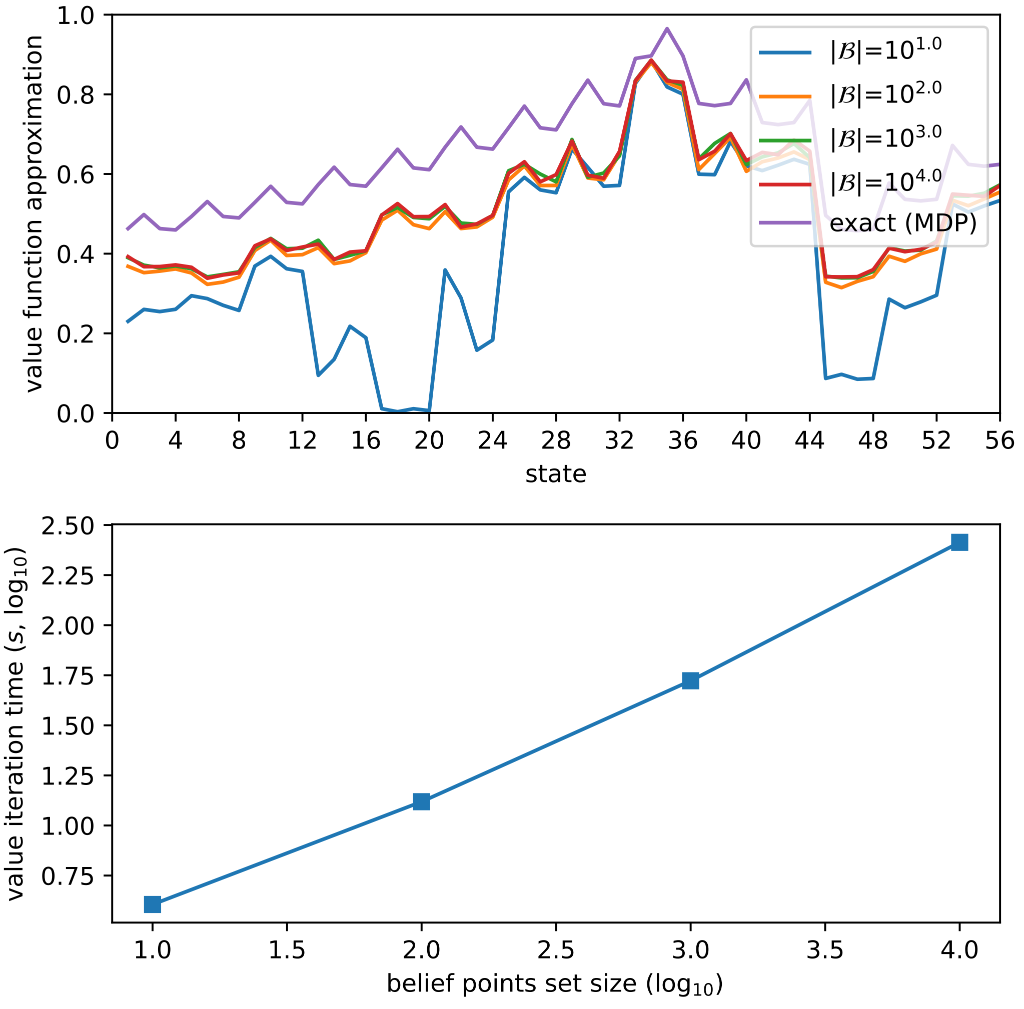

Figure 4: Execution results of the Perseus algorithm and sampling statistics for different belief points set sizes (convergence criterion of 10^{-3}): value function approximation (mean of the cumulative reward over 1000 runs) and algorithm execution time.

of the tabular methods, proving the merit of approximate methods for the value function based on a sample set of the belief space.

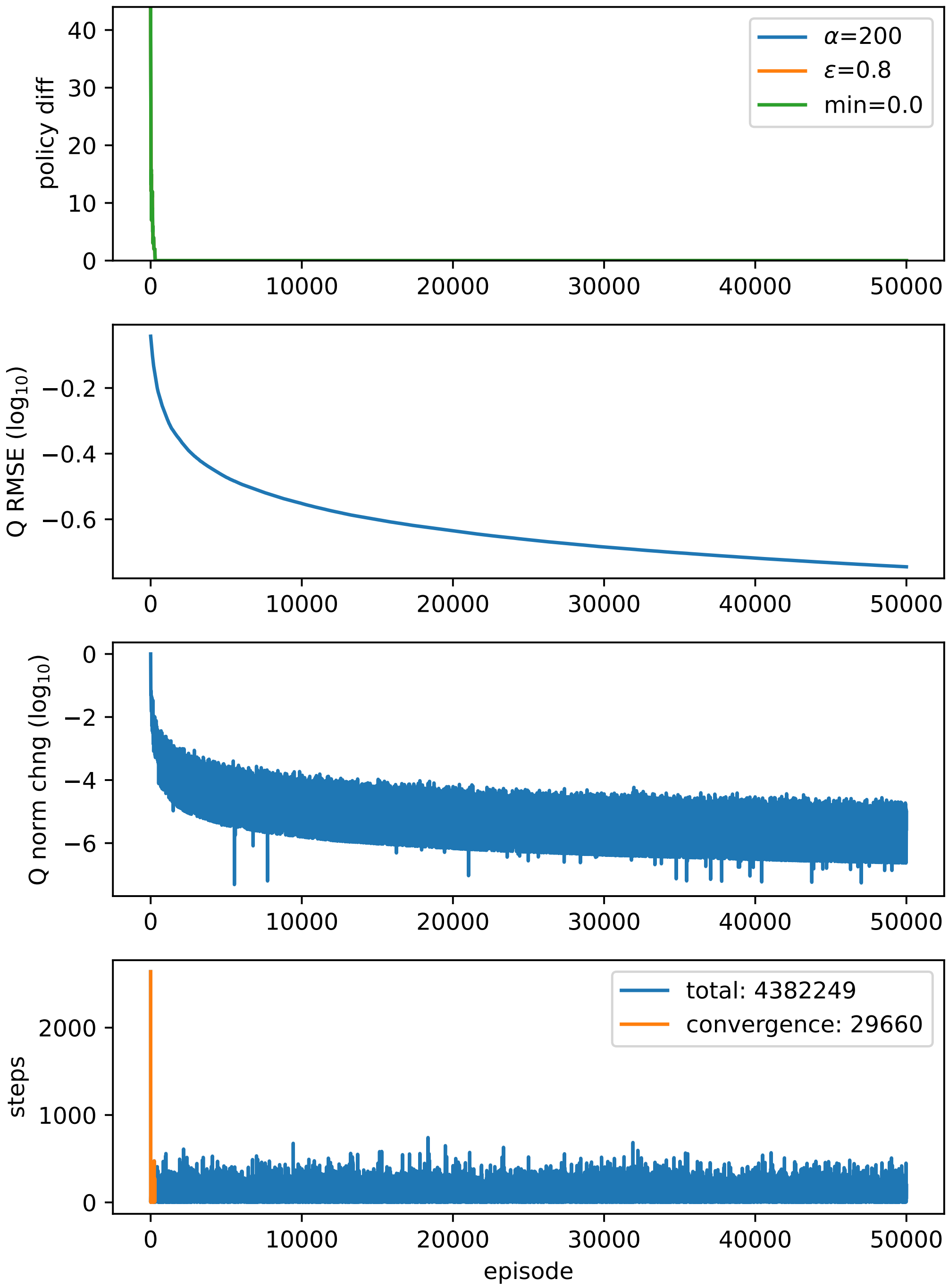

Figure 5: Policy difference $\Delta\pi$, RMSE error of the state-action value function $Q(s, a)$ relative to the optimal state-action value function $Q_*(s, a)$, relative change in the state-action value function norm and number of steps required to reach a terminal state in each episode. The hyperparameters α and ε take values of 1000 and 0.8, respectively, with $\alpha_n = \alpha/n$, where n is the total number of steps so far.

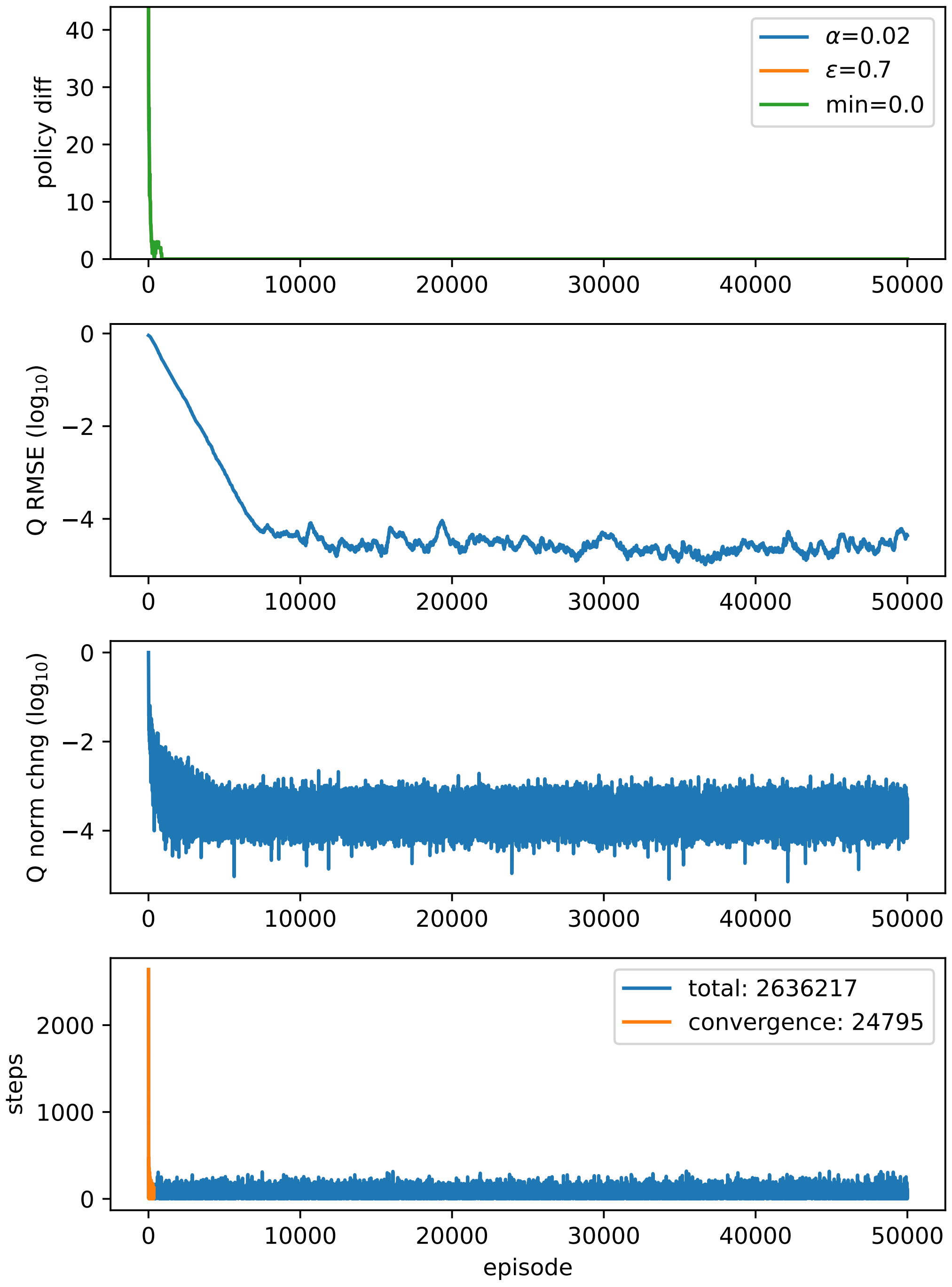

Figure 6: Policy difference $\Delta\pi$, RMSE error of the state-action value function $Q(s,a)$ relative to the optimal state-action value function $Q_*(s,a)$, relative change in the state-action value function norm and number of steps required to reach a terminal state in each episode. The hyperparameters α and ε take values of 0.02 and 0.7, respectively, with $\alpha_n = \alpha$.

A Policy Iteration

The optimal state value function v_* satisfies the Bellman optimality equation,

$$\begin{aligned} v^*(s) &= \max_a \mathbb{E}\left[G_t | S_t = s, A_t = a\right] && (4)\\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a\right] && (5)\\ &= \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v^*(s')\right], && (6) \end{aligned}$$

$\forall s \in \mathcal{S}$, $\forall s' \in \mathcal{S}^+$, $\forall a \in \mathcal{A}$, and by convention $v(s'') = 0, s'' \in \mathcal{S}^+ \setminus \mathcal{S}$ (s'' is a terminal or goal state).

If a policy π is being followed, the *state value function for policy* π is given by:

$$\begin{aligned} v^\pi(s) &= \mathbb{E}^\pi\left[G_t | S_t = s\right] && (7)\\ &= \mathbb{E}^\pi\left[R_{t+1} + \gamma v^\pi(S_{t+1}) | S_t = s\right] && (8)\\ &= \sum_a^{|\mathcal{A}|} \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v^\pi(s')\right], && (9) \end{aligned}$$

where $\pi(a|s)$ is the *probability* of taking action a in state s under policy π. The existence and uniqueness of v^π is guaranteed if either $\gamma < 1$ or termination is guaranteed from all states under policy π. The associated update rule is

$$v_{k+1}(s) = \sum_a^{|\mathcal{A}|} \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_k(s')\right], \quad (10)$$

which converges to v^π as $k \to \infty$, and this algorithm is called *iterative* policy evaluation. The value function updates for each state s can be performed in place, with the updated values being immediately available, or only be available after $v_{k+1}(s)$ has been computed for all $s \in \mathcal{S}$. In Policy Evaluation, the estimated value function $v(s)$ is updated for a policy $\pi(s)$ and for each non-terminal state, until convergence — this is also known as the *prediction* problem.

In Policy Improvement, given the state value function for a given policy, v^π, one intends to find a new policy π' such that $v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$, and if the inequality is strict for at least one state $s \in \mathcal{S}$, then policy π' is better than policy π. Policy Improvement yields a policy π' that is strictly better than policy π, unless policy π is already optimal. The new policy π' is obtained through

$$\begin{aligned} \pi'(s) &= \operatorname*{argmax}_a q^\pi(s, a) && (11)\\ &= \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v^\pi(S_{t+1}) | S_t = s, A_t = a\right] && (12)\\ &= \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v^\pi(s')\right], && (13) \end{aligned}$$

$\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}^+$.

The Policy *Iteration* method consists in alternating between Policy Evaluation and Policy Improvement until convergence to the optimal policy and corresponding optimal value function. Policy Iteration execution is outlined in Algorithm 1. Algorithm 1 is a slightly modified version of that in Section 4.3 of [2], in that it

i. uses the relative change in the value function norm as convergence criterion instead of the maximum absolute component change of the value function — this makes the convergence criterion sensitive to small absolute changes that are significant in relative terms, e.g., when a relatively great number of steps is required to achieve a reward only in a terminal state and/or the discount factor is relatively small

ii. considers an equiprobability apportioning scheme — all optimal actions for a given state are assigned equal probability; this works for both stochastic and deterministic policies (the latter being the case in the problem at hand), as the value function for a given policy is determined *in expectation*, i.e., assigning probability 1 to a single optimal action (deterministic) leads to the same result, in expectation, as assigning probability $1/N$ to N optimal actions (stochastisc or deterministic with more than one optimal action for a given state)

Moreover, the algorithm performs in-place updates (in line 7, $V(s')$ is used, which is eventually an already updated value of the value function for state s'). However, the code implementation does not perform in-place updates; it does, nonetheless, perform vectorized operations, which are still faster than carrying out updates for each individual state $s \in \mathcal{S}$.

In Fig. 7 and Table 2, the results of running Policy Iteration are shown — respectively, the optimal value function for the non-terminal states and the optimal action in each state (optimal policy).

Important: explain why action 1 (stay in the same state) is never selected

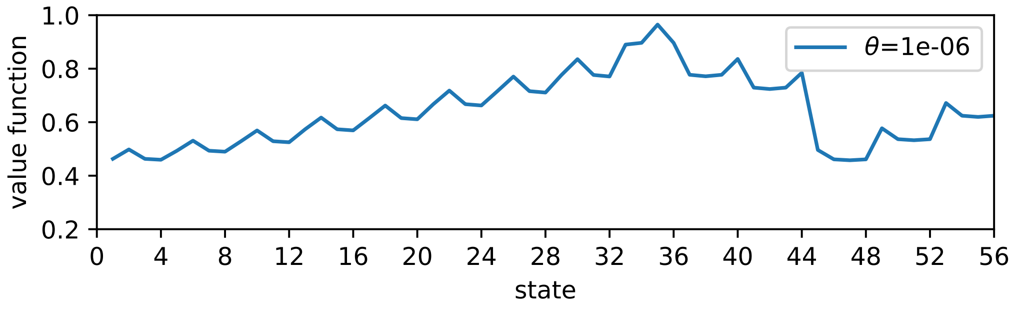

Figure 7: Optimal state value function $v(s)$ after the execution of the Policy Iteration algorithm ($\theta = 10^{-6}$).

room	1	2	3	4	5
state	1 2 3 4	5 6 7 8	9 10 11 12	13 14 15 16	17 18 19 20
optimal action	3 2 5 4	3 2 5 4	3 2 5 4	3 2 5 4	3 2 5 4
room	6	7	8	9	10
state	21 22 23 24	25 26 27 28	29 30 31 32	33 34 35 36	37 38 39 40
optimal action	3 2 5 4	3 2 5 4	3 2 5 4	4 3 2 5	5 4 3 2
room	11	12	13	14	15
state	41 42 43 44	45 46 47 48	49 50 51 52	53 54 55 56	57 58 59 60
optimal action	5 4 3 2	2 5 4 3	2 5 4 3	2 5 4 3	- - - -

Table 2: Optimal policy for the problem at hand. In each room, each state corresponds to one of the four possible orientations (north, south, west, and east) and there are five possible actions (stay in place, move forward, turn right, turn left, turn around) in each state.

B Policy difference

The policy difference between policy π_1 and policy π_2, $\Delta\pi$, is defined as the norm[5]:

$$\Delta\pi = \frac{1}{2}\left\|\pi_1(s,a) - \pi_2(s,a)\right\|_{1,1} = \frac{1}{2}\sum_{i}^{|\mathcal{S}|}\sum_{j}^{|\mathcal{A}|}\left|\pi_1(s_i,a_j) - \pi_2(s_i,a_j)\right| \tag{14}$$

The intuition behind (14) is that, if for a certain state s_i the *only* optimal action a' to be taken according to policy π_1 is different from the *only* optimal action a'' to be taken according to policy π_2, then

$$\begin{aligned}\frac{1}{2}\sum_{j}^{|\mathcal{A}|}\left|\pi_1(s_i,a_j) - \pi_2(s_i,a_j)\right| &= \frac{1}{2}\left(\left|\pi_1(s_i,a') - \pi_2(s_i,a')\right| + \left|\pi_1(s_i,a'') - \pi_2(s_i,a'')\right|\right) \\ &= \frac{1}{2}(1+1) = 1\end{aligned}$$

since

$$\begin{aligned}&\left|\pi_1(s_i,a_j) - \pi_2(s_i,a_j)\right| = |1-0| = 1 \vee \left|\pi_1(s_i,a_j) - \pi_2(s_i,a_j)\right| = |0-1| = 1, \\ &a_j = a' \vee a_j = a'',\end{aligned}$$

and

$$\begin{aligned}&\left|\pi_1(s_i,a_j) - \pi_2(s_i,a_j)\right| = |0-0| = 0, \\ &a_j \neq a' \wedge a_j \neq a''.\end{aligned}$$

In short, for deterministic policies where $\pi(s_i,a_j) = 1$ or $\pi(s_i,a_j) = 0$, $\forall s_i \in \mathcal{S}$, $\forall a_j \in \mathcal{A}$, which is the case in the problem at hand, the policy difference $\Delta\pi$ is equal to the number of states for which the action to be taken according to policy π_1 differs from the action to be taken according to policy π_2.

[5] `https://en.wikipedia.org/wiki/Matrix_norm`

C Algorithms

Algorithm 1 Policy Iteration

```
Require: $V(s) \in \mathbb{R} \setminus \{0\}, \forall s \in \mathcal{S}, V(s) = 0, \forall s \in \mathcal{S}^+ \setminus \mathcal{S}$ (value function initialization)
Require: $\pi(a|s) \geq 0$, $\sum_a^{|\mathcal{A}|} \pi(a|s) = 1, \forall s \in \mathcal{S}$ (policy initialization)
Require: $\theta > 0$ (estimation accuracy)
 1: procedure PolicyIteration($V$, $\theta$)
 2:     repeat
 3:         procedure PolicyEvaluation($V$, $\theta$)
 4:             repeat
 5:                 for all $s \in \mathcal{S}$ do
 6:                     $v(s) \leftarrow V(s)$
 7:                     $V(s) \leftarrow \sum_a^{|\mathcal{A}|} \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
 8:                 end for
 9:                 $\Delta \leftarrow \|V - v\|_2 / \|v\|_2$
10:             until $\Delta < \theta$
11:         end procedure
12:         procedure PolicyImprovement($V$)
13:             $\mathrm{policystable} \leftarrow \mathrm{true}$
14:             for all $s \in \mathcal{S}$ do
15:                 $\mathrm{oldactions} \leftarrow \{\pi(a|s) | \pi(a|s) > 0, \forall a \in \mathcal{A}\}$
16:                 $\{\pi(a|s)\} \leftarrow \mathrm{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
17:                 if $\{\pi(a|s)\} \neq \mathrm{oldactions}$ then
18:                     $\mathrm{policystable} \leftarrow \mathrm{false}$
19:                 end if
20:             end for
21:         end procedure
22:     until $\mathrm{policystable}$
23: end procedure
```

Algorithm 2 Value Iteration

```
Require: $V(s) \in \mathbb{R} \setminus \{0\}, \forall s \in \mathcal{S}, V(s) = 0, \forall s \in \mathcal{S}^+ \setminus \mathcal{S}$ (value function initialization)
Require: $\theta > 0$ (estimation accuracy)
 1: procedure ValueIteration($V$, $\theta$)
 2:     repeat
 3:         for all $s \in \mathcal{S}$ do
 4:             $v(s) \leftarrow V(s)$
 5:             $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
 6:         end for
 7:         $\Delta \leftarrow \|V - v\|_2 / \|v\|_2$
 8:     until $\Delta < \theta$
 9:     for all $s \in \mathcal{S}$ do
10:         $\{\pi(a|s)\} \leftarrow \mathrm{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
11:     end for
12: end procedure
```

References

[1] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 362–370. Morgan Kaufmann, San Francisco (CA), 1995.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2nd edition, 2018.