CMPUT 655
Assignment 2

September 14, 2024

Dikshant
1877098

---

1. *Write the Bellman Equation for the V-function and the Q-function*
   (a) *Prove the Bellman Equation for the V-function (hint: the proof it's already in the book, you need to explain its step)*
   (b) *Are the Bellman Equations easy or hard to estimate? Why?*
   (c) *What do we need to know about the MDP to use them?*
   (d) *What assumptions do we need on the computational resources our agent might have access too?*

Ans: The Bellman equation for the state-value function $V_\pi(s)$ under a policy π is given by:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right]$$

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r|s, a) \left[r + \gamma V_\pi(s')\right]$$

The Bellman equation for the action-value function $Q_\pi(s, a)$ under a policy π is given by:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right]$$

$$Q_\pi(s, a) = \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r|s, a) \left[r + \gamma V_\pi(s')\right]$$

where,

$\mathcal{A}$ is the set of possible actions
$\mathcal{S}$ is the set of possible states
$\pi(a|s)$ is the policy, i.e., the probability of taking action a in state s
$p(s', r|s, a)$ is the transition probability from state s to state s' on taking action a
r is the reward after transitioning from state s to state s' by taking action a
γ is the discount factor

(a) The value function $V_\pi(s)$ is defined as:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s\right]$$

where G_t is the return starting from time step t :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

substituting the above expression for G_t into the definition of $V_\pi(s)$:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right]$$

The expectation $\mathbb{E}_\pi \left[\cdot \mid S_t = s\right]$ can be expanded using the linearity of expectation:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_t = s\right]$$

Let's solve both parts separately now.

Given that the policy $(\pi(a|s))$ defines the probability of taking action (a) in state (s), the expectation over (R_{t+1}) can be expanded as:

$$\mathbb{E}_\pi \left[R_{t+1} \mid S_t = s\right] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r * p(s', r|s, a)$$

Now, G_{t+1} is the sum of all the future (discounted) rewards that the agent receives after state S_{t+1}. The Markovian property is that the process is memory-less with regards to previous states, actions and rewards. Future actions depend only on the state in which the action is taken and hence we can write the second part as:

$$\mathbb{E}_\pi \left[G_{t+1} \mid S_t = s\right] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s\prime \in \mathcal{S}} \sum_{r \in \mathcal{R}} V_\pi(s') * p(s', r|s, a)$$

Thus, combining the above two expressions:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r{*}p(s', r|s, a) + \gamma \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s\prime \in \mathcal{S}} \sum_{r \in \mathcal{R}} V_\pi(s'){*}p(s', r|s, a)$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left[r + \gamma V_\pi(s')\right]$$

(b) Whether the Bellman equations are easy or hard to estimate depends really upon how we can model the environment. In small, well-defined problem with known dynamics, the bellman equations can be relatively easy to estimate. However, in large and complex environments, bellman equations are not the best shot because it suffers from the curse of dimensionality and computation complexity. Bellman equations are quite difficult to estimate when state spaces and action spaces size are large because the computation of value functions requires summing over all possible states and actions, which becomes infeasible as the number of states and actions increases. Moreover, if the transition probabilities $p(s'|s,a)$ is unknown or difficult to model accurately, estimating the Bellman equations becomes even more challenging. This situation often occurs in real-world problems where the environment is only partially observable or stochastic, and where gathering sufficient data to accurately estimate these probabilities is difficult.

(c) To use the Bellman optimality equation, we need to have information about the MDP's state space, action space, transition probabilities, reward function, and discount factor.

(d) The agent has enough computational power to perform the necessary calculations of iterating over states and actions, updating value functions, and maximizing over actions. It should also have sufficient memory to store necessary information such as value functions, policies, transition probabilities, and rewards for all states and actions.

2. *If the current state is S_t, and actions are selected according to a stochastic policy π, then what is the expectation of R_{t+1} in terms of π and the four-argument function p (3.2)?*

Ans:

$$\mathbb{E}_\pi\left[R_{t+1} \mid S_t = s\right] = \sum_{a\in\mathcal{A}} \pi\left(a|s\right) * r(s,a) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} r * p(s', r|s,a)$$

3. *Give an equation for v_π in terms of q_π and π*

Ans:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi\left(a|s\right) q_\pi(s,a)$$

4. *Give an equation for q_π in terms of v_π and the four-argument p.*

Ans:

$$q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s,a)[r + \gamma v_\pi(s')]$$

5. *Now consider adding a constant c to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example.*

Ans: Adding a constant c to all rewards in an episodic task does change the task, unlike in a continuing task, due to the nature of how cumulative rewards are calculated over the finite horizon of the episode.

The value function for a state s in an episodic task is given by:

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

If a constant c is added to all rewards, the updated reward for any state-action pair becomes $R(s,a) + c$, and the value function becomes:

$$V'(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t (R(s_t, a_t) + c)\right]$$

$$V'(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] + \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t c\right]$$

$$V'(s) = V(s) + c \cdot \frac{1 - \gamma^{T+1}}{1 - \gamma}$$

Second term, that we got here is from adding the constant c in all our rewards. Thus, adding a constant c shifts the value of every state by a factor dependent on the number of steps T and the discount factor γ. This affects the policy, since the value function is now offset, and agents might choose different actions based on this new value function.

However in case of continuing tasks, adding a constant c to all rewards does not change the optimal policy because the constant shifts every value function uniformly:

$$V'(s) = V(s) + \frac{c}{1-\gamma}$$

This constant shift affects all states equally, leaving relative differences between states the same, meaning the policy remains unchanged.

Example:

Let's consider the following episodic MDP which has two states, S_0 (non-terminal) and S_T (terminal), where there are two possible actions: - a_1: Looping at S_0 with reward 0. - a_2: Moving to S_T with reward 1.

The discount factor is $\gamma = 0.5$, and there is an equal probability of choosing both actions.

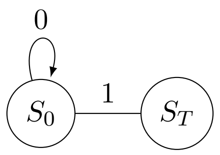

Figure 1: Original MDP

$$V(S_0) = 0.5\,(V(S_0, a_1)) + 0.5\,(V(S_0, a_2))$$
$$V(S_T) = 0$$

$$V(S_0, a_1) = 0 + \gamma V(S_0) = 0.5 \cdot V(S_0)$$

$$V(S_0, a_2) = 1 + \gamma V(S_T) = 1$$

$$V(S_0) = 0.5 \cdot (0.5 \cdot V(S_0)) + 0.5 \cdot 1$$
$$V(S_0) = 0.25 \cdot V(S_0) + 0.5$$
$$0.75 \cdot V(S_0) = 0.5$$
$$V(S_0) = \frac{0.5}{0.75} = 0.67$$

Thus, $V(S_0) = 0.67$ and $V(S_T) = 0$.

The action values are:

$$V(S_0, a_1) = 0.5 \cdot 0.67 = 0.335$$

$$V(S_0, a_2) = 1$$

Therefore, the optimal policy is to take action a_2, i.e., $\pi(S_0) = a_2$.

Now, we add a constant c to both rewards: - For a_1, the reward becomes c. - For a_2, the reward becomes $1 + c$.

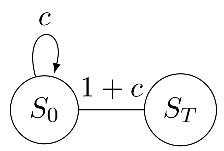

Figure 2: New MDP

$$V(S_0) = 0.5 \cdot (V(S_0, a_1)) + 0.5 \cdot (V(S_0, a_2))$$
$$V(S_T) = 0$$

$$V(S_0, a_1) = c + \gamma V(S_0) = c + 0.5V(S_0)$$

$$V(S_0, a_2) = (1 + c) + \gamma V(S_T) = 1 + c$$

Now, we compare $V(S_0, a_1)$ and $V(S_0, a_2)$ to find the optimal policy. The value function is:

$$V(S_0) = \max\left(c + 0.5V(S_0), 1 + c\right)$$

If we solve $V(S_0) = c + 0.5V(S_0)$:

$$V(S_0) - 0.5V(S_0) = c$$

$$0.5V(S_0) = c$$
$$V(S_0) = 2c$$

Now, we compare $2c$ and $1 + c$:

- If $2c \geq 1 + c$, the optimal action is a_1 (loop at S_0)
- $2c < 1 + c$, the optimal action is a_2 (move to S_T)

Thus:

- For $c = 1$, both actions are optimal
- For $c < 1$, the optimal policy is $\pi(S_0) = a_2$
- For $c > 1$, the optimal policy is $\pi(S_0) = a_1$

Thus, adding a constant to all the rewards in an episodic task can possibly change the optimal policy.

6. *Implement two functions that compute $V_\pi(s)$ and $Q_\pi(s,a)$ of any policy using policy evaluation. Additionally, have these functions log the total absolute Bellman error (i.e., over all states and actions) for every value update. For example, for $V_\pi(s)$, you will need to log: $\sum_s |V_{k+1}(s) - V_k(s)|$for each iteration k.*

   Manually define the optimal policy and learn $V_\pi(s)$ and $Q_\pi(s,a)$, plot the value functions with heatmaps (one for $V_\pi(s)$, five for $Q_\pi(s,a)$ and the trend of the Bellman error over the iterations.

   Do it for $\gamma = 0.01$, $\gamma = 0.5$, and $\gamma = 0.99$, and for three different initializations of $V_\pi(s)$ and $Q_\pi(s,a)$: all values initialized to -10, all values initialized to 0, all values initialized to 10.

   Discuss your findings. How does the initialization affect the speed of learning (i.e., the rate of convergence of the Bellman error)? How does γ affect the final value functions? Does the algorithm converge faster or slower with different values of γ? Do you think that γ would matter more for larger gridworlds?

   Submit your code, heatmaps, and plots. Have clear heatmaps and plots with labels.

Ans: We are provided with the following gridworld, where the blue block represents the starting point and the green block represents the goal location.

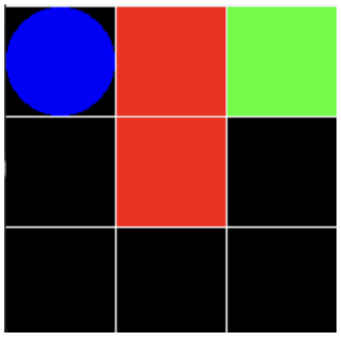

Figure 3: Gridworld

1 Implementation

We need to compute V_π and Q_π using policy evaluation. The optimal policy is defined as:

$\downarrow$	$\rightarrow$	o
$\downarrow$	$\rightarrow$	$\uparrow$
$\rightarrow$	$\rightarrow$	$\uparrow$

1.1 Computing value function

We can begin by representing the value function in vectorized form and then find this value function by solving a linear system of equations.

$$\begin{aligned}
v_\pi(s) &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma V_\pi(s')\right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r \left[p(s', r \mid s, a) r + \gamma p(s', r \mid s, a) v_\pi(s')\right] \\
&= \sum_a \pi(a \mid s) \left[\sum_r \underbrace{\sum_{s'} p(s', r \mid s, a)}_{\text{Marginalization of } p \text{ over } s'} r + \gamma \sum_{s'} \underbrace{\sum_r p(s', r \mid s, a)}_{\text{Marginalization of } p \text{ over } r} v_\pi(s')\right] \\
&= \sum_a \pi(a \mid s) \left[\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) v_\pi(s')\right] \\
&= \sum_a \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) v_\pi(s')\right], \ \forall s \in \mathcal{S} \\
&= \sum_a \pi(a|s) \left[R_{as} + \gamma \sum_{s'} P_{as,s'} v^\pi(s')\right], \quad \forall s \in S \qquad (1)
\end{aligned}$$

We can further solve it to get the matrix form, $\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}$. I have used equation (1) in my code to compute the state value function.

1.2 Equation

In this MDP, **T** stores terminal states. We don't want to update the value function of our terminal state so we can avoid it by using **1-T** in our update equation.

Therefore, value function update equation can be expressed as:

```
V = np.sum(pi * (R + gamma * np.multiply(np.dot(P, V), (1 - T))),
axis=1)
```

2 Plotting

2.1 Heatmaps for $V_\pi(s)$

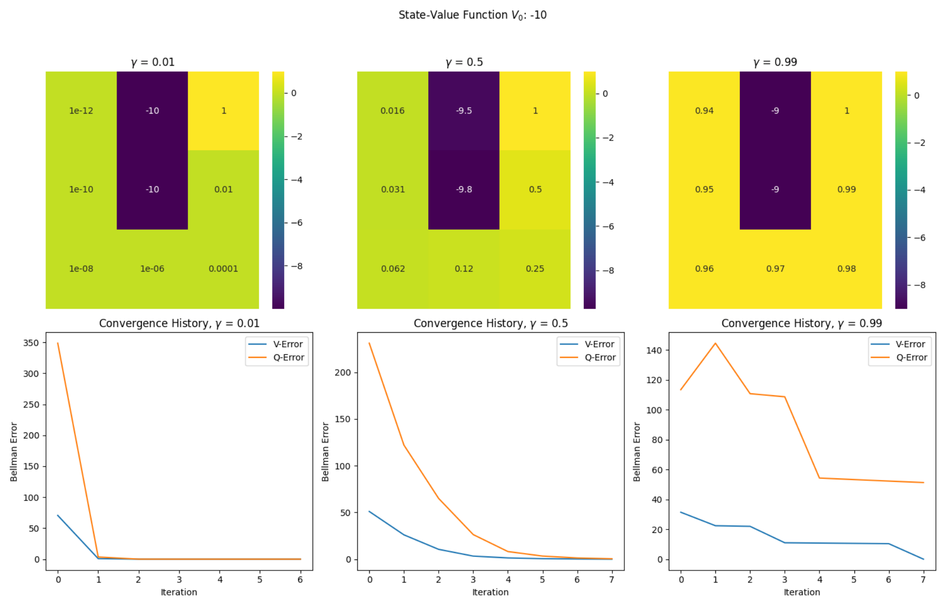

Figure 4: V_o/Q_o = -10

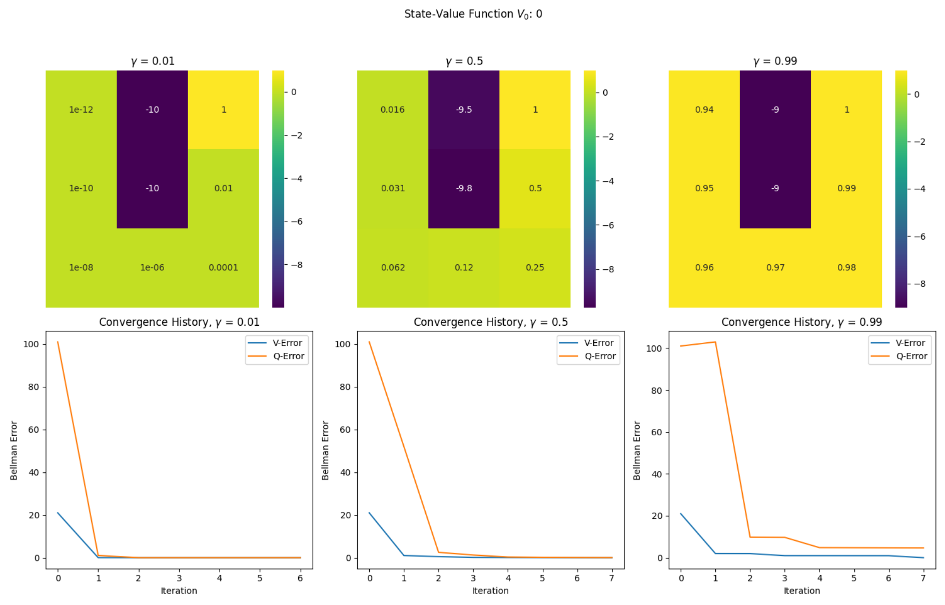

Figure 5: $V_o/Q_o = 0$

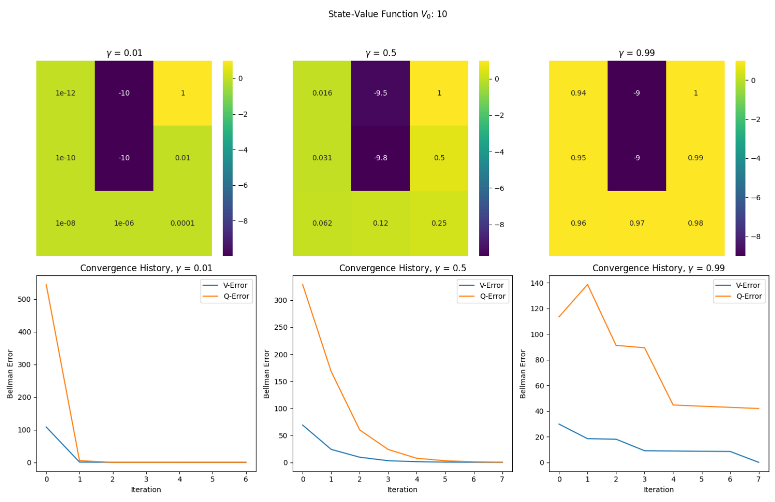

Figure 6: $V_o/Q_o = 10$

Heatmaps of the value function $V_\pi(s)$ for different initializations. The plots include the Bellman error at each iteration to indicate convergence behavior.

2.2 Heatmaps for $Q_\pi(s, a)$

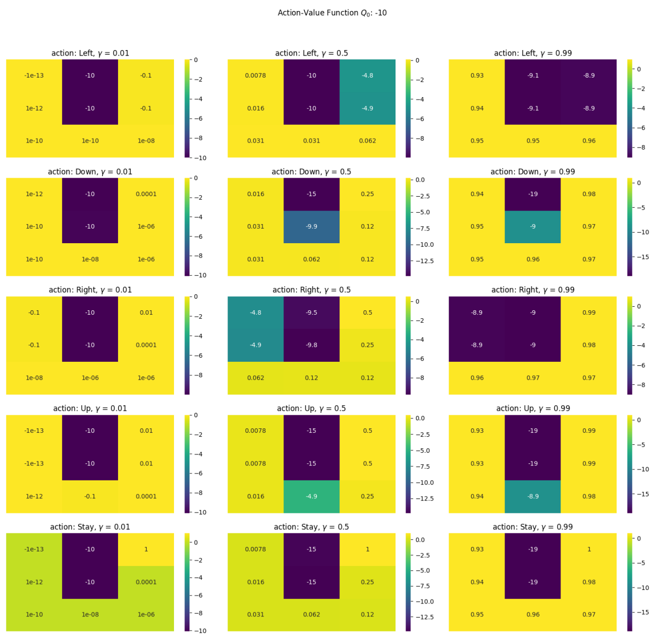

Figure 7: V_o/Q_o = -10

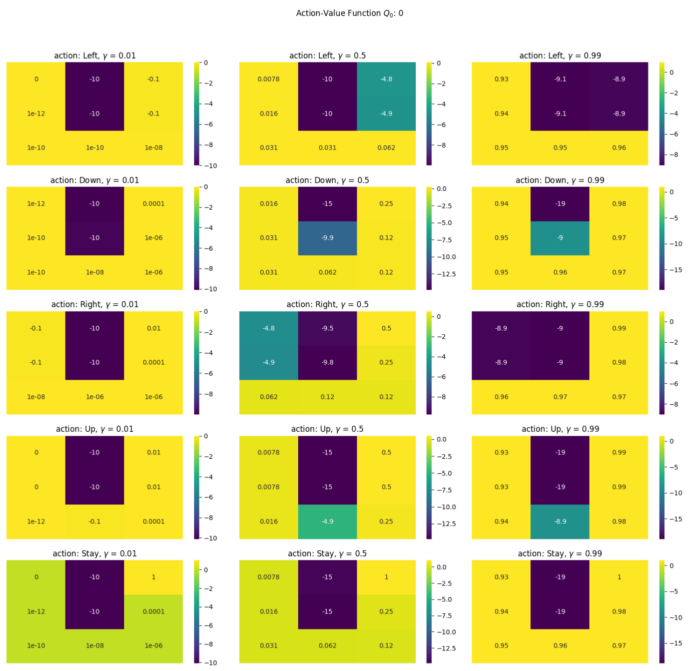

Figure 8: $V_o/Q_o = 0$

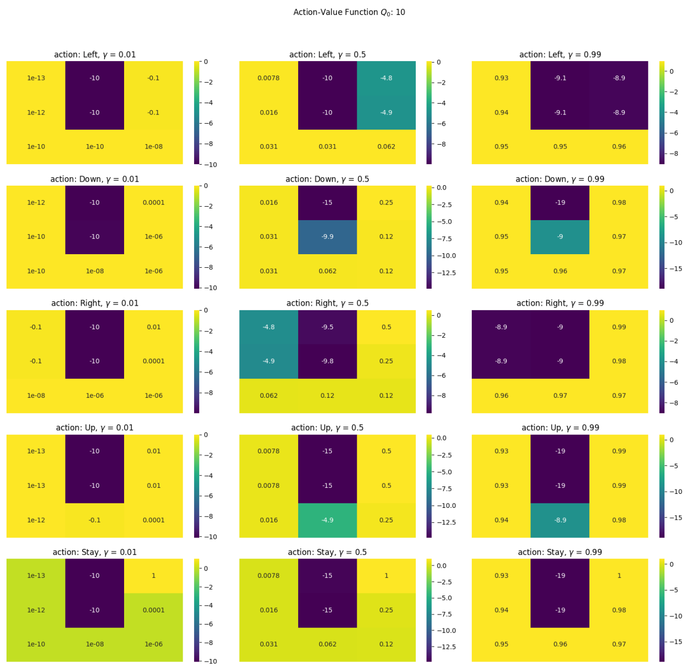

Figure 9: $V_o/Q_o = 10$

3 Discussion

3.1 Initialization factor

High initial values (either large positive or large negative) can lead to higher bellman errors initially as the algorithm tries to correct the overly optimistic estimates. This could result in a slower reduction in bellman error over time. However, the number of iterations it will take to converge will be almost similar, given same γ value but the slope of convergence will be a bit different. For high initial values, it will take some time to reach a near good estimate of value function and convergence will be a bit unstable. While for near optimal initial values, i.e. true values are around the neutral starting point, it will go near good estimate of value function much faster and hence a more stable convergence. This can be seen when the $\gamma = 0.99$, where the $V_o = 0$ reaches near 0 bellman error only after 1 iteration and stable thereafter while $V_o = 10, V_o = -10$ reaches near 0 bellman error after 7 steps in a shaky manner.

3.2 Effect of γ on final value function of the states

The discount factor γ determines how the agent balances immediate and future rewards. Specifically, For **lower values of** γ, the agent focuses primarily on immediate rewards. Future rewards are heavily discounted, **resulting in lower overall value functions**. The policy will prioritize quickly reaching positive reward states while avoiding negative ones. For **higher values of** γ, the agent places significant emphasis on future rewards, **leading to higher value functions**, especially for states leading to the goal.

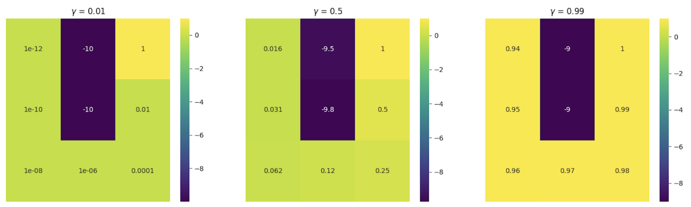

Figure 10: V_o/Q_o initialized at 10 under an optimal policy

Therefore, a lower γ results in a value function that primarily reflects immediate rewards, while a higher γ leads to higher overall values, as it accounts for future rewards over a longer horizon.

3.3 Effect of γ on convergence rate

The discount factor γ significantly influences the convergence rate of the bellman error. A **high discount factor can lead to slower convergence** because the agent evaluates long sequences of actions, which increases the complexity of the value function. On the other hand, **a low discount factor can speed up convergence** but may result in suboptimal policies that do not adequately account for future rewards. If we start with a random policy instead of an optimal policy, the effect of γ on the convergence rate becomes clearer. In the attached plot, where a random policy is used, it is evident that the value of γ affects the rate of convergence, with higher values of γ requiring more iterations to converge. Note that, it's only first step of convergence without any policy improvement.

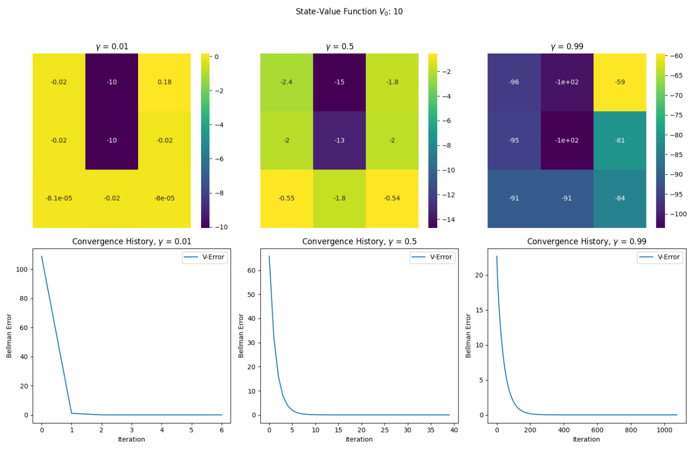

Figure 11: V_o/Q_o initialized at 10 under a random policy

Thus, while a lower γ allows for faster learning, a higher γ produces a more accurate policy by considering long-term rewards, albeit at the cost of slower convergence.

3.4 γ for larger gridworlds

In larger gridworlds, the effect of γ becomes more pronounced. A higher γ (e.g., 0.99) leads to slower learning, as the agent must evaluate long-term consequences across a larger state space. This requires more extensive planning and slows the convergence of bellman updates. On the other hand, a lower γ accelerates convergence but may result in policies that are not robust over large state spaces due to insufficient consideration of future rewards.

3.5 Extra findings

3.5.1 Effect of error threshold value, θ on convergence

A low θ results in more precise value function estimates but at the cost of slower convergence and higher computational expense. A high θ leads to faster convergence but may compromise the accuracy of the value function estimates, potentially resulting in suboptimal policies.

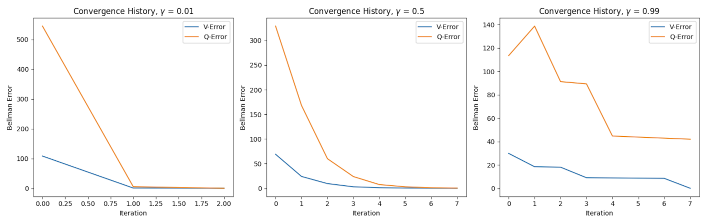

Figure 12: $V_o/Q_o = 10$ with θ = 1e-10

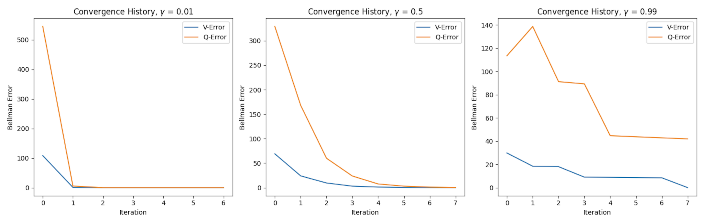

Figure 13: $V_o/Q_o = 10$ with θ = 1e-1

3.5.2 Initial spike in the Q-value error for higher values of γ

If the initial Q-values are not well-aligned with the expected values (considering long-term rewards), the bellman updates will be more drastic. Since γ is high, even small discrepancies in initial values lead to significant updates in future rewards, leading to a higher initial bellman error.

Code can be accessed on github from here: Bellman Error evaluation

References

[1] Deriving the matrix form: `https://ai.stackexchange.com/questions/28560/how-can-we-find-the-value-function-by-solving-a-system-of-linear-equations`