MVA: Reinforcement Learning (2020/2021) **Homework 1**

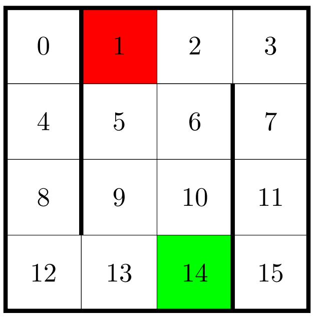
Dynamic Programming

Lecturers: *A. Lazaric, M. Pirotta* *(November 8, 2020)*

Solution by Samuel Diai

1 Question

Consider the following grid environment. The agent can move up, down, left and right. Transitions are deterministic. Attempts to move in the direction of the wall will result in staying in the same position. There are two absorbing states: 1 and 14. Taking any action in 1 (resp 14) leads to a reward r_r (resp. r_g) ($r(1,a) = r_r, r(14,a) = r_g, \forall a$) and *ends the episode.* Everywhere else the reward is r_s. Assume discount factor $\gamma = 1$, $r_g = 10$ and $r_r = -10$, unless otherwise specified.



Question 1.1

Define r_s such that the optimal policy is the shortest path to state 14. Using the chosen r_s, report the value function of the optimal policy for each state.
There is a simple solution that doesn't require complex computation. You can copy the image an replace the id of the state with the value function.

Solution 1.1

In the infinite time horizon framework, with a discount factor $\gamma = 1$, the state value function writes :

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, \pi(s_t))|s_0 = s; \pi\right]$$

Thus, each reward has the same importance in the computation of the state value function. In order to force the optimal policy to be the shortest path to state 14, we need to set a negative reward r_s.
Otherwise, when maximising the state value function, the optimal policy will force the agent to do as many transition as possible, to get an infinite reward. Moreover, this condition is not sufficient, we need to make sure that for any starting state (chosen randomly), the agent will prefer to go to the state 14 rather than the state 1. Indeed, if the agent goes to the state 1, this ends the episode and the agent will never see the state 14. The worst case scenario is when the agent spawns closer to the state 1 than the state 14 (in Manhattan distance). The worst states are the states 5 and 2, and we need to make sure that transitionning from state 2 or 5 to the state 1 gives less reward than transitionning to the state 14. This writes :

$$r_s - 10 < 3r_s + 10 \iff rs > -10$$

We verify that for the other starting states, this condition also implies that the agent will go to the state 14 with the shortest path. Defining $r_s = -1$ obsiously met the previous conditions. We can compute the state value function for this reward :

5	-10	7	6
6	7	8	5
7	8	9	4
8	9	10	3

Question 1.2

Consider a general MDP with rewards, and transitions. Consider a discount factor of $\gamma < 1$. For this case assume that the horizon is infinite (so there is no termination). A policy π in this MDP induces a value function V^π. Suppose an affine transformation is applied to the reward, what is the new value function? Is the optimal policy preserved?

Solution 1.2

In this setting, we have :

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 = s; \pi\right]$$

If we set $r'(s_t, \pi(s_t)) = \alpha r(s_t, \pi(s_t)) + \beta$, the new state value function V' becomes :

$$V'^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t (\alpha r(s_t, \pi(s_t)) + \beta) | s_0 = s; \pi\right]$$

With the linearity of the expectation

$$V'^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \alpha r(s_t, \pi(s_t)) | s_0 = s; \pi\right] + \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \beta | s_0 = s; \pi\right]$$

Which gives by recognizing a geometric sum :

$$V'^\pi(s) = \alpha V^\pi(s) + \sum_{t=0}^{\infty} \gamma^t \beta \mathbb{E}\left[1 | s_0 = s; \pi\right] = \alpha V^\pi(s) + \frac{\beta}{1-\gamma}$$

Thus, if $\alpha > 0$, maximizing the orginal state value function and the translated one is equivalent, but if $\alpha < 0$, we have a totally different optimal policy, as the sign of the state value function changes.

Question 1.3

Consider the same setting as in question 1. Assume we modify the reward function with an additive term $c = 5$ (i.e., $r_s = r_s + c$). How does the optimal policy change (just give a one or two sentence description)? What is the new value function?

Solution 1.3

In this setting, we have $\gamma < 1$, and the results from the previous question do not remain valid. As we have seen in the first question, the condition on r_s was : $-10 < r_s < 0$.
Thus, if $-10 < r_s < -5$, the modified reward function $r'_s = r_s + 5$ verifies : $-5 < r'_s < 0$, thus the optimal policy does not change. The new value fonction V'^{π} simply becomes $V'^{\pi}(s) = V^{\pi}(s) + c$.
However, if $-5 < r_s < 0$, then, $0 < r'_s < 5$, the reward function becomes positive, and the optimal policy will consist of making an infinite number of transitions (so as to get the maximum final reward possible). In that case, the value fonction is not defined or becomes $+\infty$ outside of the states 1 and 14.

2 Question

Consider infinite-horizon γ-discounted Markov Decision Processes with S states and A actions. Denote by $Q^\star$ the Q-function of the optimal policy $\pi^\star$. Prove that, for any function $Q(s,a)$, the following inequality holds for any s

$$V^{\pi_Q}(s) \geq V^\star(s) - \frac{2\|Q^\star - Q\|_\infty}{1-\gamma}$$

where $e = (1, \dots, 1)$, $\|Q^\star - Q\|_\infty = \max_{s,a} |Q^\star(s,a) - Q(s,a)|$ and $\pi_Q(s) = \arg\max_a Q(s,a)$. Thus $\pi^\star(s) = \arg\max_a Q^\star(s,a)$.

Solution 2.1

We start by observing that by definition of the greedy policy π_Q :

$$Q(s, \pi^\star(s)) \leq Q(s, \pi_Q(s))$$

And for any $s \in S$ and $a \in A$:

$$|Q^\star(s,a) - Q(s,a)| \leq \|Q^\star - Q\|_\infty$$

In particular for $a = \pi^\star(s)$ and $a = \pi_Q(s)$:

$$|Q^\star(s,\pi^\star(s)) - Q(s,\pi^\star(s))| \leq \|Q^\star - Q\|_\infty \text{ and } |Q^\star(s,\pi_Q(s)) - Q(s,\pi_Q(s))| \leq \|Q^\star - Q\|_\infty$$

Which leads to :

$$Q(s,\pi_Q(s)) \leq Q^\star(s,\pi_Q(s)) + \|Q^\star - Q\|_\infty \text{ and } Q(s,\pi^\star(s)) \geq Q^\star(s,\pi^\star(s)) - \|Q^\star - Q\|_\infty$$

Which can be summarized as :

$$Q^\star(s,\pi^\star(s)) - Q^\star(s,\pi_Q(s)) \leq 2\|Q^\star - Q\|_\infty$$

By using the link between $Q^\star$ and $V^\star$:

$$r(s,\pi^\star(s)) + \gamma \sum_{s'} p(s'|s,\pi^\star(s)) V^\star(s') - r(s,\pi_Q(s)) - \gamma \sum_{s'} p(s'|s,\pi_Q(s)) V^\star(s') \leq 2\|Q^\star - Q\|_\infty$$

On the other side, using bellman identities:

$$V^\star(s) - V^{\pi_Q}(s) = r(s,\pi^\star(s)) + \gamma \sum_{s'} p(s'|s,\pi^\star(s)) V^\star(s') - r(s,\pi_Q(s)) - \gamma \sum_{s'} p(s'|s,\pi_Q(s)) V^{\pi_Q}(s')$$

By combining the two previous equations, we have :

$$V^\star(s) - V^{\pi_Q}(s) - \gamma \sum_{s'} p(s'|s,\pi_Q(s)) \left[V^\star(s') - V^{\pi_Q}(s')\right] \leq 2\|Q^\star - Q\|_\infty$$

By using that $\forall s' \in S, V^\star(s') - V^{\pi_Q}(s') \leq \|V^\star - V^{\pi_Q}\|_\infty$:

$$\gamma \sum_{s'} p(s'|s, \pi_Q(s)) \left[V^\star(s') - V^{\pi_Q}(s')\right] \leq \gamma \sum_{s'} p(s'|s, \pi_Q(s)) \|V^\star - V^{\pi_Q}\|_\infty \leq \gamma \|V^\star - V^{\pi_Q}\|_\infty$$

We end up with :

$$V^\star(s) - V^{\pi_Q}(s) \leq 2\|Q^\star - Q\|_\infty + \gamma \|V^\star - V^{\pi_Q}\|_\infty$$

As it is true for every $s \in S$, it is true for $s^\star$ such that $V^\star(s^\star) - V^{\pi_Q}(s^\star) = \|V^\star - V^{\pi_Q}\|_\infty$:

$$\|V^\star - V^{\pi_Q}\|_\infty \leq 2\|Q^\star - Q\|_\infty + \gamma \|V^\star - V^{\pi_Q}\|_\infty$$

Which gives :

$$\|V^\star - V^{\pi_Q}\|_\infty \leq \frac{2}{1-\gamma} \|Q^\star - Q\|_\infty$$

Finally, we have :

$$\forall s \in S, V^\star(s) - V^{\pi_Q}(s) \leq \frac{2}{1-\gamma} \|Q^\star - Q\|_\infty$$

3 Question

Consider the average reward setting ($\gamma = 1$) and a Markov Decision Process with S states and A actions. Prove that

$$g^{\pi'} - g^\pi = \sum_s \mu^{\pi'}(s) \sum_a (\pi'(a|s) - \pi(a|s)) Q^\pi(s, a) \tag{1}$$

using the fact that in average reward the Bellman equation is

$$Q^\pi(s, a) = r(s, a) - g^\pi + \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a'), \quad \forall s, a, \pi$$

and μ^π is the **stationary distribution** of policy π. Note also that $g^\pi = \sum_x \mu^\pi(s) \sum_a \pi(a|s) r(s, a)$.

Note: All the information provided to prove Eq. 1 are mentioned in the question. Start from the definition of Q and use the property of stationary distribution.

Solution 3.1

We have by definition, $g^{\pi'} = \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) r(s, a)$. And using the average reward the Bellman equation:

$$r(s, a) = Q^\pi(s, a) + g^\pi - \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a'), \quad \forall s, a, \pi$$

We then replace $r(s, a)$ in the definition of $g^{\pi'}$:

$$g^{\pi'} = \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \left[Q^\pi(s, a) + g^\pi - \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right]$$

Moreover, for any t_∞ such that we have a stationary distribution,

$$\sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) = \sum_a \sum_s \mathbb{P}(S_{t_\infty} = s) \mathbb{P}(A_{t_\infty} = a | S_{t_\infty} = s) = \sum_a \sum_s (A_{t_\infty} = a, S_{t_\infty} = s) = 1$$

We then have :

$$g^{\pi'} = \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^\pi(s, a) + g^\pi - \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a')$$

As we have stochastic policies, we now write the stationary property for the matrix $P^{\pi'}$, where $P^{\pi'}_{s,s'} = \mathbb{E}_{a\sim\pi'(s)}\left[P(s,a,s')\right]$:

$$\mu^{\pi'} P^{\pi'} = \mu^{\pi'}$$

We have :

$$\mathbb{E}_{a\sim\pi'(s)}\left[P(s,a,s')\right] = \sum_a p(s'|s,a)\pi'(a|s)$$

Then if we develop the matrix product given by the stationary property, we have :

$$\forall s' \in S, \mu^{\pi'}(s') = \sum_s \mu^{\pi'}(s) \sum_a p(s'|s,a)\pi'(a|s)$$

By substituting, we have :

$$g^{\pi'} = \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^{\pi}(s,a) + g^{\pi} - \sum_{s'} \mu^{\pi'}(s') \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')$$

Which gives, by changing the index names in the sums :

$$g^{\pi'} = g^{\pi} + \sum_s \mu^{\pi'}(s) \sum_a \left[\pi'(a|s) - \pi(a|s)\right] Q^{\pi}(s,a)$$

4 Question

Provide an MDP modeling, specifying all its defining elements, of the following process:

- Elevator dispatching. The elevator controllers assign elevators to service passenger requests in real-time while optimizing the overall service quality, e.g. by minimizing the waiting time and/or the energy consumption. The agent can simultaneously control all the elevators. In order to model this problem, consider a 6-story building with 2 elevators. Explain the choices made for modeling this problem.

Question 4.1

In order to model the problem, I introduced several variables that will be uselfull for the state definitions, transition etc..

- $c_i, i = 0,1,2,3,4,5,6$, binary values representing call requests on each floor i. We suppose the two elevator share the same call requests, as we usually have only one button for multiple elevators in real life applications.
- $d_{i,j}, i = 0,1,2,3,4,5,6, j = 1,2$, binary variables representing call requests inside each elevator j for each floor i. We have 7 variables per elevator which makes 14 in total.
- p_1 and p_2 The discrete elevator positions. p_1 denotes the position for the first elevator and p_2 denotes the position for the second elevator. $(p_j)_{j\in\{1,2\}}$ take values in $\{0,1,2,3,4,5,6\}$
- N is the maximum occupancy of each elevator. Usually around 6.

I made the following assumptions :

- I model the elevator system as a discrete-time system with a given sample time. Indeed I assume the time required for passengers to enter and exit the elevator and the transition time between two floors is constant. I think it is a good heuristic of the real functioning of elevators and in practice we can change this setting to adapt as close as possible to the history data.
- It is hard to modelise the occupancy of the elevators since multiple passengers can be waiting at a given floor and in practice I don't think elevators have an occupancy system to count the number of passengers. I chose to introduce the variables $d_{i,j}$ which are an indirect measure of the occupancy, but I could have deleted these variables and introduced in the agent's state the number of passengers at the time t.

The state space of the elevator system is discrete and has 28 dimensions :

$$s = [(c_i)_i, (d_{i,j})_{i,j}, p_0, p_1]$$

The elevator controller can chose among 6 discrete actions $a = (a_1, a_2)$ where $a_1 \in \{-1, 0, 1\}$ and $a_2 \in \{-1, 0, 1\}$. a_1 denotes the action imposed on the first elevator and a_2 the action imposed on the second elevator. The -1 action accelerates the elevator downwards, 1 accelerates it upwards, and 0 stops the elevator.
If the j-th elevator is at the 6th floor, $a_j \in \{-1, 0\}$ as it cannot go upwards. Similarly, If the j-th elevator is at the ground floor (0) , $a_j \in \{0, 1\}$.
The reward function is :

$$r_t(s,a) = -\sum_{i \in \{0,1,\ldots,6\}} \left[c_i + \sum_{j \in \{1,2\}} d_{ij} \right] - \sum_{j \in \{1,2\}} |a_j|$$

- The first term : $-\sum_{i \in \{0,1,\ldots,6\}} c_i$ is important so as to minimize the number of calls at each floor. As c_i is binary, the elevator controller will try to reduce the number of calls c_i so that $c_i = 0$ (no elevator calls).
- The second term : $-\sum_{i \in \{0,1,\ldots,6\}} \sum_{j \in \{1,2\}} d_{ij}$ is important so as to minimize the total number of calls inside the elevators. As d_{ij} is binary, the elevator controller will try to reduce the number of calls d_{ij} so that $d_{ij} = 0$ (each passenger was deposited on his floor).
- The last term $-\sum_{j \in \{1,2\}} |a_j|$ is very important because it forces the elevator system to stand still if there is no outside or inside calls. Indeed, if one elevator j moves upwards or downwards $a_j = -1$ or $a_j = 1$, thus $|a_j| = 1$

Finally I modeled the transition as follows :
As it is impossible to know how many people are waiting at each floor, and as the maximum occupancy of each elevator is limited, I introduced some variables e which are used to limit the number of passenger entering the elavator. The transitions are defined such that : e is a Bernouilli variable with parameter $1/N$ (I remind that N is the maximum occupancy of the elevator), or any function such that the greater the N, the lower the probability of rejection.

$$c_i \leftarrow \begin{cases} 1 & \text{if } e = 1 \\ c_i & \text{otherwise} \end{cases}$$

That is to say, the button can remain in the same state after a timestep, and the probability of this event happening depends on the maximum occupancy N of the elevator. (N is a parameter known in advance). This allows the model to take into consideration that some passengers can wait several timesteps, the time for the elevator to go back and forth if the elevator is full.
And obvioulsy we can update the position of the elevators given an action a between two timesteps :

$$p_0 \leftarrow p_0 + a_0 \text{ and } p_1 \leftarrow p_1 + a_1$$

5 Question

Implement value iteration and policy iteration. We have provided a custom environment in the starter code.

Question 5.1

(coding) Consider the provided code `vipi.py` and implement `policy_iteration`. Use γ as provided by `env.gamma` and the terminal condition seen in class. Return the optimal value function and the optimal policy.

Solution 5.1

We write the policy iteration algorithm. The first step is the policy evaluation, given π_k, we compute V^{π_k}, to do that we use the result :

$$V^{\pi_k} = (I - \gamma P^{\pi_k})^{-1} r^{\pi_k}, \text{ Where } r^{\pi_k}(s) = r(s, \pi_k(s)) \text{ And } P^{\pi_k}_{s,s'} = p(s'|s, \pi_k(s)) = p(s, \pi_k(s), s')$$

```
def policy_evaluation(P, R, policy, gamma=0.9, tol=1e-2):
    """
    Args:
        P: np.array
            transition matrix (NsxNaxNs)
        R: np.array
            reward matrix (NsxNa)
        policy: np.array
            matrix mapping states to action (Ns)
        gamma: float
            discount factor
        tol: float
            precision of the solution
    Return:
        value_function: np.array
            The value function of the given policy
    """
    Ns, Na = R.shape
    # ====================================================
    ## Compute matrix P_pi with policy pi ie P_pi(s', s) = P(s, pi(s), s')
    P_pi = P[range(Ns), policy, :]
    R_pi = R[range(Ns), policy]
    ## We use solve to resolve the linear system instead of inverting the matrix
    value_function = np.linalg.solve(np.eye(Ns) - gamma*P_pi, R_pi)
    # ====================================================
    return value_function
```

The second step is the policy improvement, given V^{π_k}, compute π_{k+1}:

$$\pi_{k+1}(s) \in \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^{\pi_k}(s') \right]$$

We stop if $\|\pi_{k+1} - \pi_k\|_\infty <$ tol

```
def policy_iteration(P, R, gamma=0.9, tol=1e-3):
    """
    Args:
        P: np.array
            transition matrix (NsxNaxNs)
        R: np.array
            reward matrix (NsxNa)
        gamma: float
            discount factor
        tol: float
            precision of the solution
    Return:
        policy: np.array
```

```
            the final policy
        V: np.array
            the value function associated to the final policy
    """
    Ns, Na = R.shape
    V = np.zeros(Ns)
    policy = np.zeros(Ns, dtype=np.int)
    # ====================================================
    iterating = True
    while iterating :
        V = policy_evaluation(P, R, policy, gamma)
        before_max = R + gamma*P.dot(V) # Shape (NsxNa)
        new_policy = np.argmax(before_max, axis = 1)
        if np.allclose(new_policy, policy, atol = tol, rtol = 0):
            iterating = False
        else :
            policy = new_policy
    # ====================================================
    return policy, V
```

Question 5.2

(coding) Implement `value_iteration` in `vipi.py`. The terminal condition is based on $\|V_{new} - V_{old}\|_\infty$ and the tolerance is 10^{-5}. Return the optimal value function and the optimal policy.

Solution 5.2

For the value iteration algorithm, we iteratively update V_k : $V_{k+1} = \mathcal{T}V_k$, we return the greedy policy $\pi_K(s) \in \arg\max_{a\in A}\left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V^{\pi_K}(s')\right]$, once $\|V_{new} - V_{old}\|_\infty$ is inferior to a tolerance parameter.

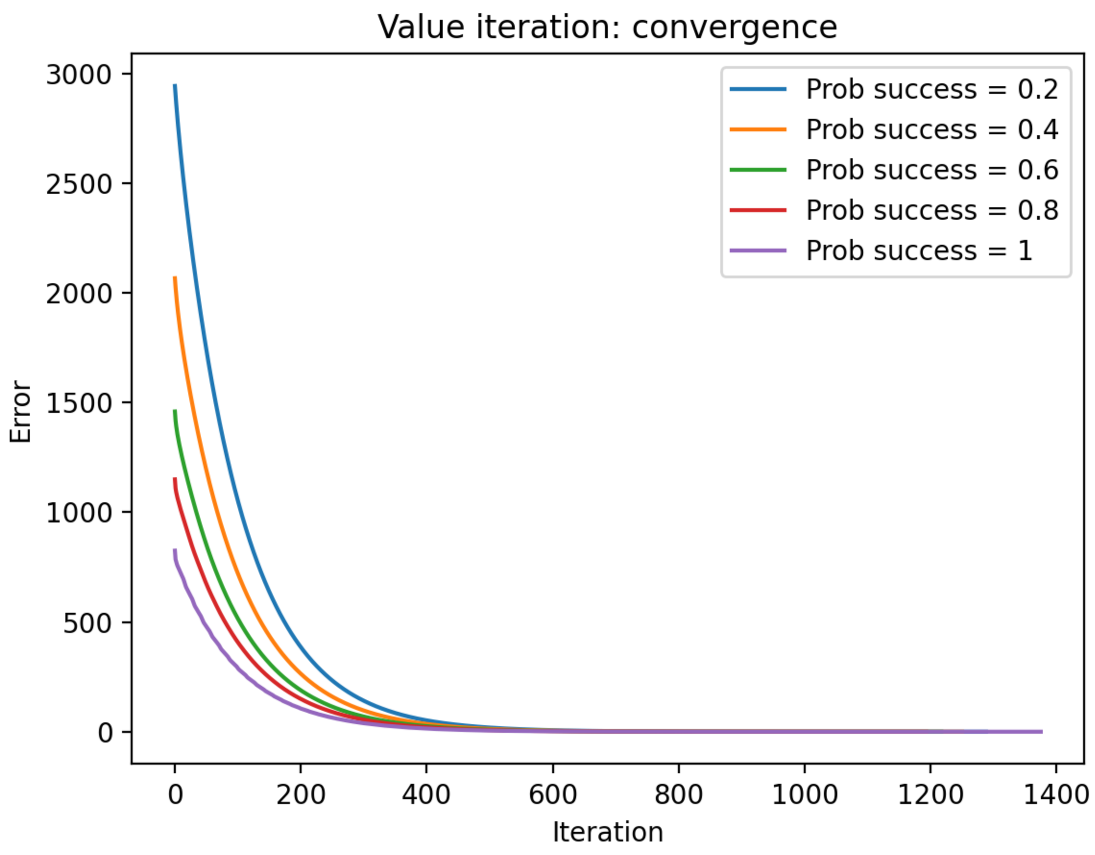
```
def value_iteration(P, R, gamma=0.9, tol=1e-3):
    """
    Args:
        P: np.array
            transition matrix (NsxNaxNs)
        R: np.array
            reward matrix (NsxNa)
        gamma: float
            discount factor
        tol: float
            precision of the solution
    Return:
        Q: final Q-function (at iteration n)
        greedy_policy: greedy policy wrt Qn
        Qfs: all Q-functions generated by the algorithm (for visualization)
    """
    Ns, Na = R.shape
    Q = np.zeros((Ns, Na))
    Qfs = [Q]
    # ====================================================
# YOUR IMPLEMENTATION HERE
    iterating = True
    while iterating :
        V = np.max(Q, axis = 1)
        Q_new = R + gamma * P.dot(V)
        V_new =  np.max(Q_new, axis = 1)
        if np.linalg.norm(V_new - V, ord = np.inf) < tol :
            iterating = False
        else :
            Qfs.append(Q_new)
            Q = Q_new
    greedy_policy = np.argmax(Q_new, axis = 1)
    # ======================================================
    return Q, greedy_policy, Qfs
```

Question 5.3

(written) Run the methods on the deterministic and stochastic version of the environment. How does stochasticity affect the number of iterations required, and the resulting policy?
You can render the policy using `env.render_policy(policy)`

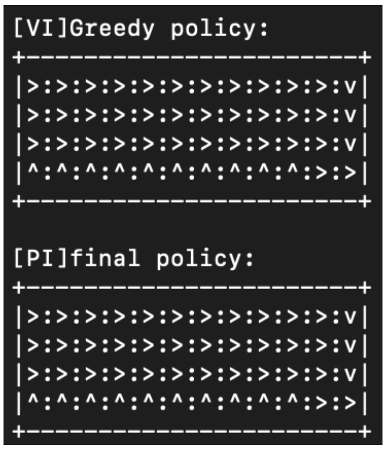
Solution 5.3

I plotted the number of iterations required to get convergence for the value iteration method, for different probability of success parameters : $[0.2, 0.4, 0.6, 0.8, 1]$.



As we can see, the number of iterations required to get below a certain error threshold increases as the stochasticity parameter is important. The stochasticity parameter defines the probability of success when the agent chose to do a certain action. The higher this parameter is, the more chance there is

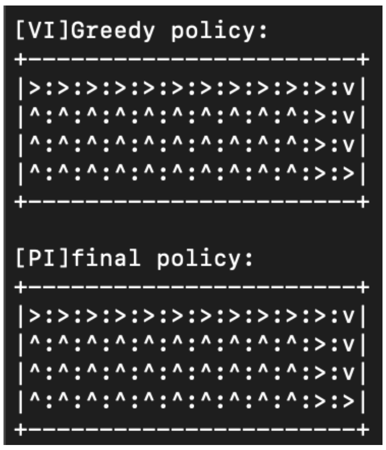
for the agent to move into an unwanted state. Thus it seems completely normal that the higher the stochasticity parameter, the more the number of iterations increases as it is more difficult to tell the agent to move to a certain direction.
We can also observe the optimal policies for different values of p_{succ}.



```
[VI]Greedy policy:
+-----------------------+
|>:>:>:>:>:>:>:>:>:>:>:v|
|>:>:>:>:>:>:>:>:>:>:>:v|
|>:>:>:>:>:>:>:>:>:>:>:v|
|^:^:^:^:^:^:^:^:^:^:>:>|
+-----------------------+

[PI]final policy:
+-----------------------+
|>:>:>:>:>:>:>:>:>:>:>:v|
|>:>:>:>:>:>:>:>:>:>:>:v|
|>:>:>:>:>:>:>:>:>:>:>:v|
|^:^:^:^:^:^:^:^:^:^:>:>|
+-----------------------+
```

(a) Optimal policies with $p_{succ} = 1$



```
[VI]Greedy policy:
+-----------------------+
|>:>:>:>:>:>:>:>:>:>:>:v|
|^:^:^:^:^:^:^:^:^:^:>:v|
|^:^:^:^:^:^:^:^:^:^:>:v|
|^:^:^:^:^:^:^:^:^:^:>:>|
+-----------------------+

[PI]final policy:
+-----------------------+
|>:>:>:>:>:>:>:>:>:>:>:v|
|^:^:^:^:^:^:^:^:^:^:>:v|
|^:^:^:^:^:^:^:^:^:^:>:v|
|^:^:^:^:^:^:^:^:^:^:>:>|
+-----------------------+
```

(b) Optimal policies with $p_{succ} = 0.6$

Figure 1: 2 Figures side by side

As we can see, for $p_{succ} = 1$ the optimal policies for both the algorithms are identical and very simple : the agent needs to go to the right until it reaches the wall and then go down. Howerver, for $p_{succ} = 0.6$, the optimal policies are still identical but less straightforward. If the agent is not located next to the top wall, he first needs to reach it and then go to the right. This seems intuitive, when the agent is close to the top wall, he has only 3 possibilities of movement instead of 4 if he is in the middle of the grid. Thus is has less movement uncertainties, and he will reach the final state faster.

Question 5.4

(written) Compare value iteration and policy iteration. Highlight pros and cons of each method.

Solution 5.4

The main difference between Policy iteration and Value Iteration is that :

- Policy iteration includes: policy evaluation and policy improvement, and the two are repeated iteratively until policy converges.
- Value iteration includes: finding optimal value function and only one one policy extraction. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).

Thus policy iteration requires to do two computations at each step, while value iteration only does one and extract the optimal policy at the end. Howerver as we have seen in class, policy iteration converges in a finite number of iterations and usually faster than value iteration.
I've run several benchmark tests for the grid example. I've found that policy iteration's computing time is significantly higher than the value iteration one: on average, value iteration is done in 0.125 seconds while policy iteration is done in 0.002 seconds. Policy Iteration is 60 times faster on average !