Reinforcement Learning : Dynamic Programming Technical Report

Riashat Islam
McGill University
Reasoning and Learning Lab
riashat.islam@cs.mcgill.ca

January 26, 2017

1 Introduction

In this work, we consider analysis of the basic reinforcement learning algorithms such as value iteration and policy iteration on three different models, namely smallworld, gridworld and cliffworld MDPs. We consider value and policy iteration and discuss the proof of convergence for these algorithms. We implemented value and policy iteration algorithms in matlab, and demonstrated its effectiveness in converging to an optimal value function and policy. We also include an interpretation of how policy iteration can be considered to be similar to Newton's method.

1.1 Experimental Setup

Models/Environments: We consider experiments over three different benchmark RL tasks. For all the environments, the agent is allowed 4 discrete actions in a discrete state space. In the Small World MDP (model), the goal states (state with maximal reward) is at the coordinates $(4, 4)$. We consider a constant reward of -1 in all the states, reward of 10 in the goal state and a negative reward (cost) of -6 in the bad state. The smallworld MDP has 17 different states. Similarly, the gridworld MDP has 109 different states, with the goal state at 93 and pre-defined start state. We also consider a cliffworld MDP with the usual actions of up, down, left and right, with a reward of -1 for all transitions, except for a reward of -100 if the agent enters the cliff, and a reward of 10 in the goal state with co-ordinates $(5, 9)$. We always consider discounted MDPs with $\gamma = 0.9$. Our work also considers careful fine-tuning of the ϵ parameter for greedy action selection as discussed later. We always consider stochastic discrete policies in a model-based environment with given transition dynamics.

2 Value Iteration

In reinforcement learning, value iteration concerns with finding the optimal policy π using an iterative application of the Bellman optimality backup. At each iteration, value iteration uses synchronous updates for all the states to update the value function, ie, update $V_{k+1}(s)$ from $V_k(s')$. In our work, we consider using value iteration over the gridworld model. In order for value iteration to converge, it requires an infinite number of iterations to converge to $V*$. We use the stopping criterion for value iteration that when there are no further improvements to the value function, or when the change in value function is less than a very small positive number in a given sweep, we terminate the algorithm. Value iteration is guaranteed to converge to an optimal policy for finite MDPs with a discounted reward. Unlike policy iteration, there is no explicit policy in value iteration. In our algorithm, we evaluate the policy at each step of the episode as shown by the code below.

$$V_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma(\sum_{s' \in S} P_{ss'}^a V_k(s')) \tag{1}$$

We include a brief snippet of our value iteration algorithm implementation as below:

```
% run VI on GridWorld
gridworld;
[v, pi] = valueIteration(model, 1000)
plotVP(v,pi, paramSet)

%value iteration algorithm
function [v, pi] = valueIteration(model, maxit)
% initialize the value function
v = zeros(model.stateCount, 1);
pi = ones(model.stateCount, 1);
old_v = zeros(model.stateCount, 1);
threshold = 1.0000e-22;

for iterations = 1:maxit,
    % initialize the policy and the new value function
    policy = ones(model.stateCount, 1);
    v_ = zeros(model.stateCount, 1);
    % perform the Bellman update for each state
    for s = 1:model.stateCount,
    %compute transition probability
    P = reshape(model.P(s,:,:), model.stateCount, 4);
    %update value function
    [v_(s,:), action] = max(model.R(s,:) +
        (model.gamma * P' * v)');
    %policy evaluated every step
    policy(s,:) = action;
    end
old_v = v;
v = v_;
pi = policy;
%break condition
%to check convergence of VI algorithm
if v - old_v <= threshold
    fprintf('Value function converged
    after \%d iterations\n',iterations);
        break;
end

end
end
```

Listing 1: Value Iteration Algorithm

2.1 Experimental Results

We demonstrate the effectiveness of value iteration algorithm on three model environments, namely smallworld, gridworld and cliffworld. First we show the performance of our value iteration algorithm on a 2D grid, where lighter region indicates a better learnt policy at that state, compared to the darker regions. The cumulative reward of 0 indicates the maximum reward that can be achieved. We basically used negative rewards, where a large negative cumulative reward would indicate the agent performing poorly.

Value iteration algorithm on small world MDP with 100 and 1000 iterations as shown in figure 1 and 2 below:

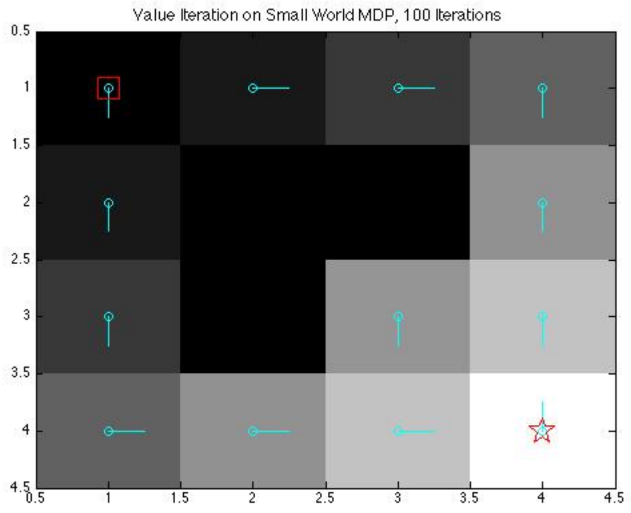

Figure 1: Value iteration on Small World MDP with 100 iterations

The results above indicate that the learnt value function is the same for both 100 and 1000 iterations. This means that for the smallworld MDP case, the value iteration algorithm converges really fast, especially since it is a small environment we are considering here. This can be further demonstrated by plotting the cumulative reward plot for both 100 and 1000 iterations. Results show that the maximum cumulative reward achieved, or in other words the value function converges within 50 iterations in the smallworld MDP case.

In a similar way, we can evaluate the performance of our value iteration algorithm on the gridworld and cliffworld MDPs, which are much larger environments, as shown in figures 5 and figure 6 below. We ran our value iteration algorithm for 1000 episodes for both MDPs.

Testing for convergence of Value Iteration on different environments : In order to check for convergence of value iteration and how it depends on the state space of the environment, we ran the algorithm for same number of iterations on the different environments. Figure 7 summarizes how the convergence of value iteration algorithm depends on the size of the environment. For convenice, we only show the plot for upto 500 iterations since the algorithm converges within that number of

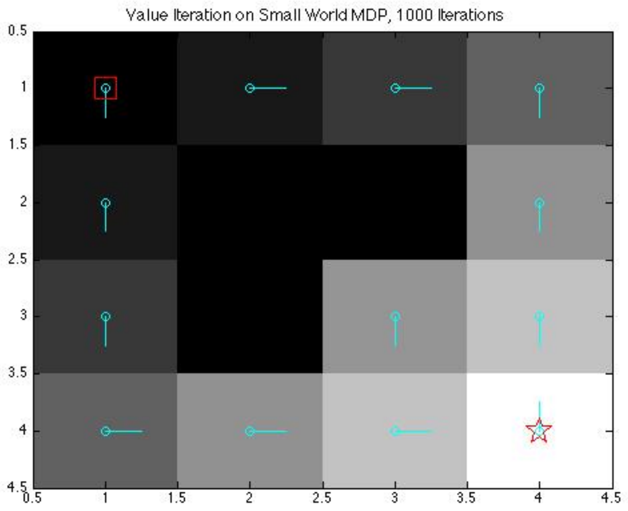

Figure 2: Value iteration on Small World MDP with 1000 iterations

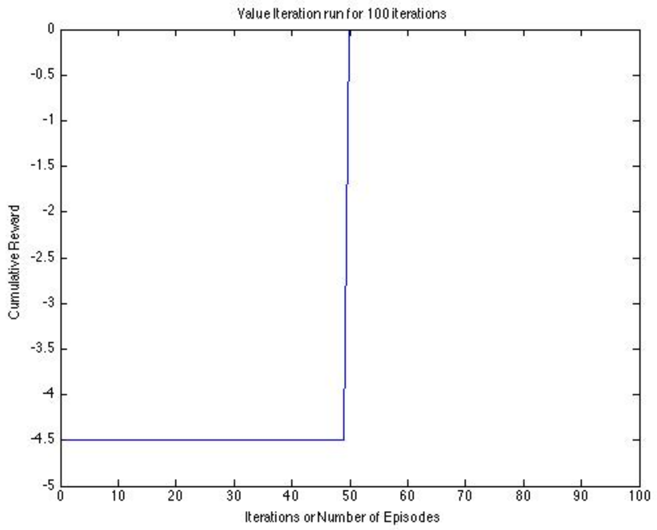

Figure 3: Cumulative reward of agent following value iteration on Small World MDP with 100 iterations

iterations.

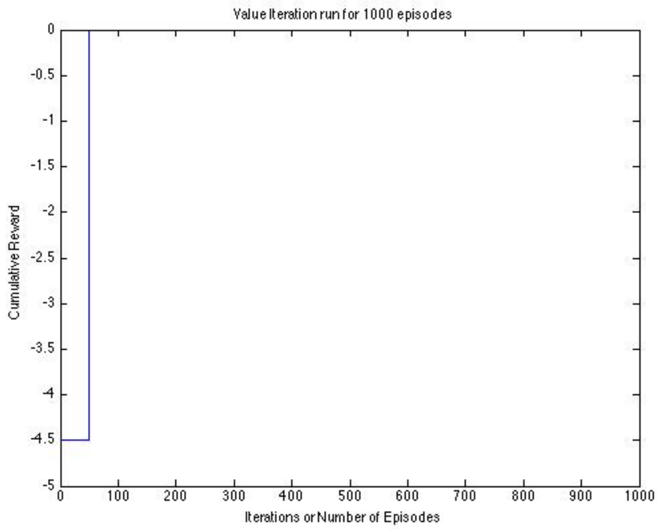

Figure 4: Cumulative reward of agent following value iteration on Small World MDP with 1000 iterations

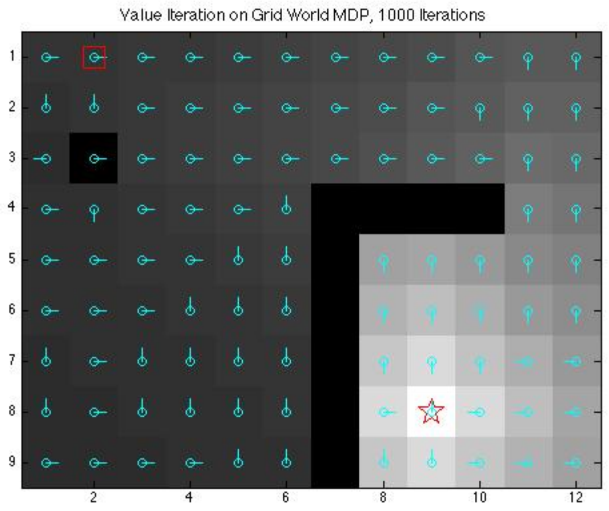

Figure 5: Value Iteration on Grid World MDP with 1000 iterations

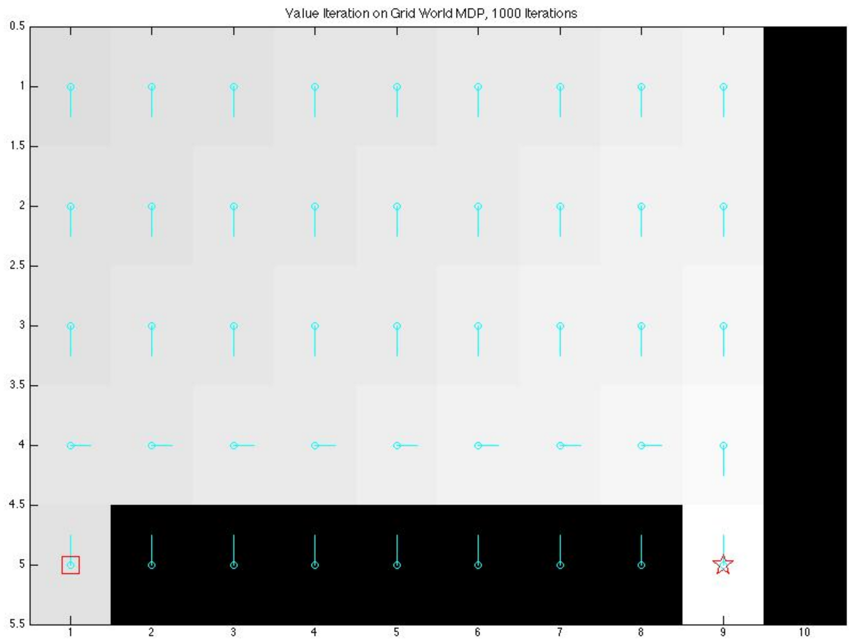

Figure 6: Value Iteration on Cliff World MDP with 1000 iterations

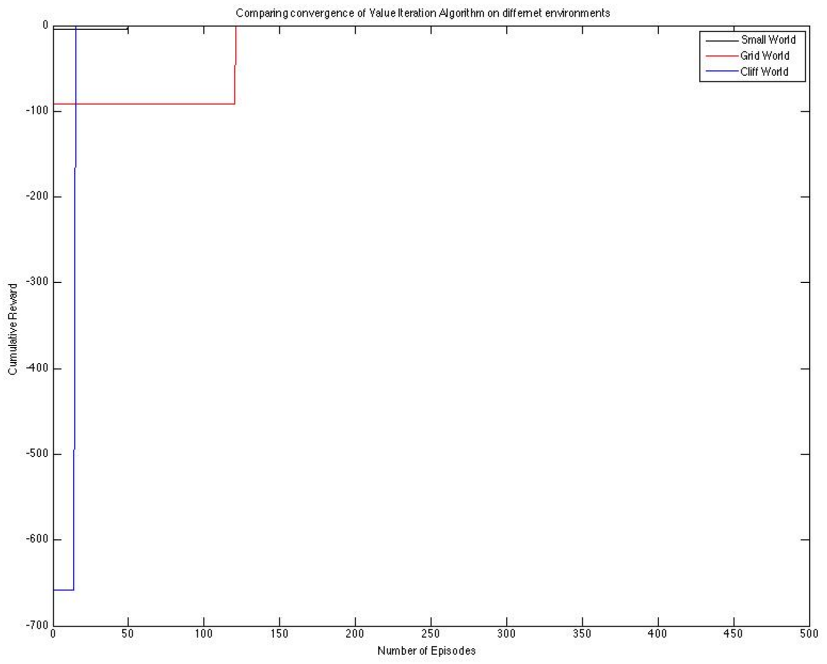

Figure 7: Comparison of convergence of value iteration on different environments

3 Policy Iteration

We then consider policy iteration instead of value iteration on the gridworld model. Policy iteration uses an iterative policy evaluation step to estimate V^{π} and a policy improvement step to generate $\pi' > \pi$, where π' is obtained by a greedy policy improvement step. The policy evaluation step evaluates the value function for a given policy π using an iterative Bellman expectation backup. The policy improvement step is then the action that maximizes the value function, using a greedy policy improvement step. In other words, policy iteration obtains a sequence of continually improving policies and value functions, where we do policy evaluation and improvement separately, while every policy improvement is guaranteed to be an improvement.

3.1 Experimental Results

We then implemented the policy iteration algorithms on the same environments, and evaluated the convergence of this algorithm. Figures 8, 9 and 10 shows the policy iteration algorithm on small-world, gridworld and cliffworld MDPs. Similar to above, we again ran the policy iteration algorithm with 1000 iterations. The results show that similar plots are obtained for value iteration as for policy iteration, suggesting that the algorithm converges similar to value iteration.

Below we include a brief code snippet of out implementation of the policy iteration algorithm.

```
function [v, pi] = policyIteration(model, maxit)

% initialize the value function
v = zeros(model.stateCount, 1);
pi = ones(model.stateCount, 1);
% old_v = zeros(model.stateCount, 1);
policy = ones(model.stateCount, 1);
tol = 0.0000000000000000000001;

%run this extra loop
% to check for convergence
%in policy evaluation and
%policy improvement step
for iterations = 1:maxit,

    % Policy Evaluation Step
    %with same number of episodes
    for i = 1:maxit,
    v_ = zeros(model.stateCount, 1);
    % perform the Bellman update for each state
    for s = 1:model.stateCount
    v_(s) = model.R(s, policy(s))
    + (model.gamma*model.P(s,:,policy(s))*v)';
    end
    delta = norm(v - v_);
    v = v_;

%check for convergence
if delta <= tol
fprintf('Value function
converged after $\%$ d iterations\n',i);
break;
end
end

```

```
for s = 1:model.stateCount
    P =reshape(model.P(s,:,:),model.stateCount,4);
    [~, action] =
    max(model.R(s,:) + (model.gamma *P'*v)');

    policy(s) = action;
end
end
pi = policy;
v = v_;

end
```

Listing 2: Policy Iteration Algorithm

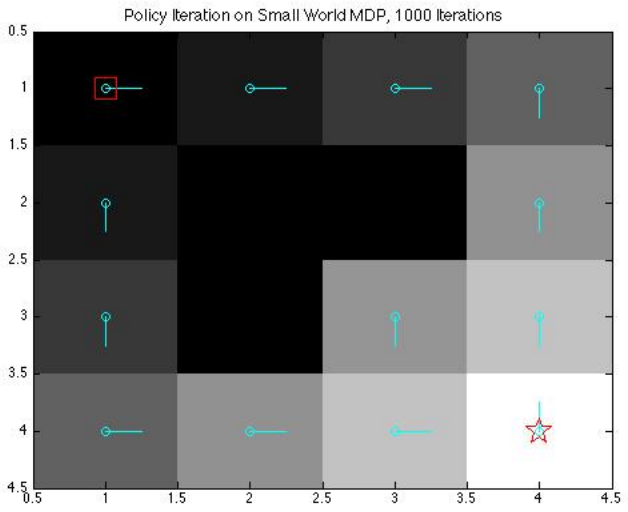

Figure 8: Policy iteration on small world MDP with 1000 iterations

However, we note that the policy iteration algorithm takes longer to run than the value iteration algorithm. This is mainly because we first test for convergence of the value function, after which there is a greedy maximisation step to improve the policy. Due to two loops in the algorithm, one for converging value function - the policy evaluation step, and one for improving the policy - the policy improvement step - the policy iteration algorithm takes longer to converge.

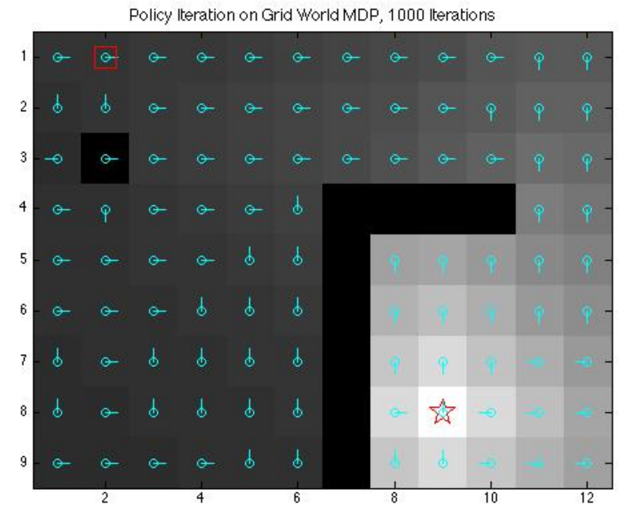

Figure 9: Policy iteration on grid world MDP with 1000 iterations

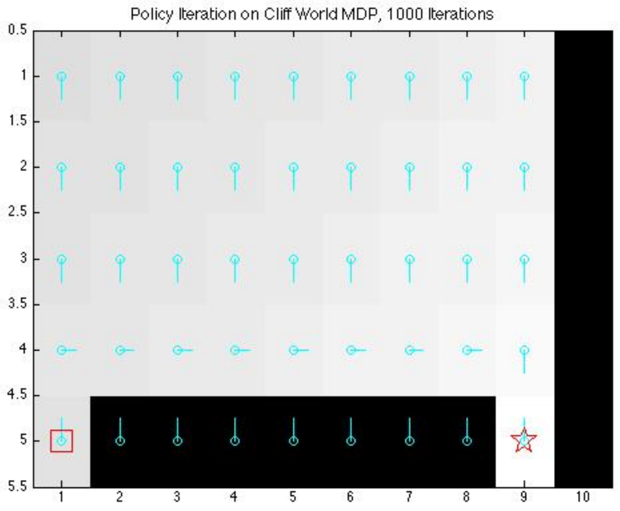

Figure 10: Policy iteration on cliff world MDP with 1000 iterations

4 Convergence of Value and Policy Iteration

Our insight is that, even though the policy iteration algorithm takes longer to run, it is more guaranteed to converge experimentally than the value iteration algorithm. The speed of convergence is faster for policy iteration compared to value iteration, even though the entire policy iteration algorithm may take longer to compute. Here are some reasons why

The value iteration algorithm converges after 120 iterations. Compared to that, the policy iteration algorithm converges after 88 iterations. This further validates that using policy iteration, we can converge faster to the optimal policy π^*. The reason is further explained below: A drawback of using value iteration is that it can take longer for value iteration algorithms to converge in some state spaces. This is because the iterations of value iteration is independent of the actual policy, and so the algorithm runs even when the policy is not changing. Since in RL, the goal is to find the optimal policy, and value functions at each state provides a tool to find the optimal policy, it is indeed better to evaluate the policy directly and check for convergence based directly computing the policy. Policy iteration algorithms therefore provide a good measure of when value functions have converged, since we can directly compute the policy. When there are no more improvements in the actual policy, the value function is guaranteed to converge. Compared to this, value iteration uses no measure of the actual policy and hence finding the optimal value function takes larger number of iterations than policy iteration. Policy iteration algorithms converges faster.

5 Theoretic Analysis of Convergence of Policy Iteration

In this section, we include a brief theoretical analysis of the proof of convergence of the policy iteration algorithm.

This section considers the proof of convergence of the policy iteration algorithm. We denote our policy as $\pi(s)$ and the Bellman operator as T. As discussed earlier, policy iteration algorithm involves two steps: policy improvement followed by policy evaluation.

First we show that by acting greedily means policy improvement $\pi'(s) > \pi(s)$ for every greedy action. Since the value function is given as $V_{\pi(s)} = R_s + \gamma P^{\pi} V$, greedy action means:

$$\pi'(s) = \arg\max_a [R_s + P^{\pi} V] \tag{2}$$

therefore, a greedy action leads to an improvement in the value function:

$$V_{\pi'(s)} \geq V_{\pi(s)} \tag{3}$$

In other words, since $Q_\pi(s, \pi(s)) = V_\pi(s)$, this means

$$Q_\pi(s, \pi'(s)) \geq Q_\pi(s, \pi(s)) \tag{4}$$

We can therefore prove that a policy improvement step in the policy iteration algorithm leads to $V_{\pi'}(s) \geq V_\pi(s)$ as follows:

V$_\pi(s) =\leq Q_\pi(s, \pi'(s))$
= E$_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | s_t]$
= $\leq E_{\pi'}[R_{t+1} + \gamma Q_\pi(s_{t+1}, \pi'(s_{t+1})) | s_t]$
= $\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(s_{t+2}, \pi'(s_{t+2})) | s_t$
= $\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + ... | s_t]$
= V$_{\pi'}(s)$
(5)

The policy improvement step in policy iteration therefore guarantees that:

$$V^{\pi_{k+1}} \geq V^{\pi_k} \tag{6}$$

Using the Bellman operator this therefore means:

V$^{\pi_k} = T^{\pi_k} V^{\pi_k}$
V$^{\pi_k} =\leq TV^{\pi_k}$
V$^{\pi_k} = T^{\pi_{k+1}} V^{\pi_k}$
(7)

where π_{k+1} is an improved policy following greedy policy improvement. We want to show that following each step of policy improvement, there is an improvement in the value function given by the Bellman operator as follows:

$$V^{\pi_k} \leq T^{\pi_{k+1}} V^{\pi_k} \tag{8}$$

Hence, following the Bellman operator, the improvement of policy from π_k to π_{k+1} shows an improvement in the value function as given by:

$$V^{\pi_k} \leq T^{\pi_{k+1}} V^{\pi_k} \leq (T^{\pi_{k+1}})^2 V^{\pi_k} \leq ... \tag{9}$$

Hence, we can show that:

$$V^{\pi_k} \leq lim_{n->\inf} (T^{\pi_{k+1}})^n V^{\pi+k} = V^{\pi_{k+1}} \tag{10}$$

Following iterations in policy iteration algorithm, when the policy improvement stops, ie $\pi'(s) = \pi(s)$, or alternatively we can write $Q_\pi(s, \pi'(s)) = Q_\pi(s, \pi(s)) = V_\pi(s)$. When an optimal policy is reached following greedy policy improvement steps, it will satisfy the Bellman optimality equation:

$$V^{\pi_k} = V^* \tag{11}$$

So the algorithm stops after finite steps k, when there is no more improvement in policy improvement step, and hence the policy evaluation step. **This means convergence of the policy iteration algorithm is guaranteed**. Policy iteration stops when there are no more improvements by taking a greedy action, and the Bellman optimality equation is satisfied as given by equation 11.

There exists only a finite number of policies in a given MDP. However, there exists only one optimal deterministic policy for any setting. Since the number of policies is finite, so the policy iteration algorithm steps must also converge after a finite k steps as:

$$V^{\pi_q} = V^{\pi_{q+1}} \tag{12}$$

V^{π_q} is a fixed point of T, and since T has a unique fixed point, we can therefore also deduce that:

$$V^{\pi_q} = V^* \tag{13}$$

and hence π_q is an optimal policy.

6 Policy Iteration as Newton's Method

We will include a brief intuitive understanding of how policy iteration can be considered similar to Newton's method. Similar to Newton's method, the policy evaluation step can be considered as deriving the value of $f(x)$ for a given x, just as we would derive the value function as an expected sum of rewards. Thereafter, since the policy iteration has a step involving maximisation of the value function to find an action that would maximise the value - this can be considered the same as finding the gradient and updating x in $f(x)$ in the direction of the gradient. The update in the point x in $f(x)$ by finding the gradient of the function is similar to the policy improvement step where the maximisation essentially chooses an action that would increase the value function.

7 Conclusion

In this report, we included basic implementations of the value and policy iteration algorithms, and demonstrated their performances on three different environments. For all the environments, we showed the convergence of the policy and value iteration algorithms to the optimal policy. We also discussed that even though the policy iteration algorithm may take longer time to compute, but it has a higher speed of convergence mainly because the policy improvement step chooses actions which would maximise the value function. Hence, even if considering the contraction mapping theorem, the value function would converge to an optimal value function much faster. We then briefly demonstrated theoretical guarantees of the convergence of policy iteration, which is in line with what has been shown in the demonstrated experiments. Finally, we included an intuitive interpretation of how the policy iteration algorithm can be considered to be similar to Newton's method.