MEMORIAL UNIVERSITY OF NEWFOUNDLAND

A red logo with white text

Description automatically generated

ASSIGNMENT 2

Experimentation with learning policies on

simple reinforcement algorithms

Submitted by:

Oviemuno Peter Utomakili (201477924)

Dhiraj Rajkarnikar (202382307)

Table of Contents

[INTRODUCTION 3](#_Toc172239233)

[OBJECTIVES 4](#_Toc172239234)

[EXPERIMENTS 4](#_Toc172239235)

[Scenario 1: 4](#_Toc172239236)

[Part 1: 5](#_Toc172239237)

[Scenario 2: 8](#_Toc172239238)

[Part 2 8](#_Toc172239239)

[CONCLUSION 15](#_Toc172239240)

**Table of Figures**

[Figure 1: Explicitly solving Bellman Equation 5](#_Toc172239247)

[Figure 2: Using iterative policy evaluation for Bellman equation 5](#_Toc172239248)

[Figure 3: Using value iteration for Bellman equation 6](#_Toc172239249)

[Figure 4: Optimal policy determination for Bellman equation solving explicitly 7](#_Toc172239250)

[Figure 5: Optimal policy determination using iterative policy evaluation 7](#_Toc172239251)

[Figure 6: Optimal policy determination using value iteration 8](#_Toc172239252)

[Figure 7: Monte Carlo with exploring starts (Optimal Policy) 9](#_Toc172239253)

[Figure 8: Monte Carlo with exploring starts (Value function) 9](#_Toc172239254)

[Figure 9: Monte Carlo with epsilon-soft approach 10](#_Toc172239255)

[Figure 10: Monte Carlo Importance Sampling 11](#_Toc172239256)

[Figure 11: Policy Iteration for dynamic gridworld with termination states (Optimal Policy) 12](#_Toc172239257)

[Figure 12: Policy Iteration for dynamic gridworld with termination states (Value function) 13](#_Toc172239258)

[Figure 13: Policy Iteration for static gridworld with termination states (Optimal Policy) 14](#_Toc172239259)

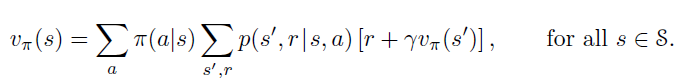
[Figure 14: Policy Iteration for static gridworld with termination states (Value function) 14](#_Toc172239260)

# INTRODUCTION

Reinforcement learning (RL) is a domain of machine learning where an agent learns to make decisions by interacting with an environment where the agent aims to maximize some notion of cumulative rewards through its actions. These actions are guided by policies which basically are mappings from states of the environment to actions.

A concept in this field that has been discussed in class is the **Bellman Equation**. It serves as the basis for determining the optimal policy and value functions. This is one of the fundamental reinforcements learning policy which provides a recursive decomposition of the value function. A value function that assigns to each state (or state-action pair) a value representing the expected cumulative reward starting from that state (or state-action pair). Bellman expectation equation is used for evaluation of the value of a policy. Considering a policy’s actions, the Bellman expectation equation defines the value of state under a policy as the expected return from that state.

Mathematically, the Bellman expectation equation for value function for a policy π can be expressed as:



* is the value function for policy π and represents the expected cumulative reward.
* π(a∣s) is the policy which gives the probability of taking action *a* when in state *s*.

In general, the Bellman equation provides a value function for a learning policy as an output and policy evaluation methods can be implemented to evaluate the given policy. One of those discussed is Iterative policy evaluation. The idea in **Iterative policy evaluation** is to start with an arbitrary value function and iteratively apply the Bellman equation for the policy to update the value function until it converges to the true value function for that policy. The iterative process involves following steps:

1. Initialize *V(s)* arbitrarily (usually to 0 for all states).
2. Update *V(s)* for all states *s* based on the Bellman expectation equation.
3. Repeat step 2 until *V(s)* converges to a stable value.

Additionally, we also discussed about **Value Iteration**, an algorithm used to find the optimal policy by iteratively improving the value function. This algorithm combines the principles of policy evaluation and policy improvement into a single step. The value iteration process involves the following steps.

1. Initialize *V(s)* arbitrarily.
2. Update *V(s)* using the Bellman optimality equation.
3. Repeat step 2 until *V(s)* converges.

Understanding these processes and implementing them are essential for developing effective reinforcement learning policies.

**Monte-Carlo methods** were also a topic of discussion in the lectures. These methods are used to evaluate and improve policies by relying on direct sampling of experiences to estimate value functions. This is contrary to the approach from iterative policy evaluation and value iteration which rely on the model of the environment. Simply, we can say that the Monte Carlo methods are used to estimate the value of states or state-action pairs based on average returns observed in actual or simulated episodes. By leveraging the observed experiences and returns, Monte-Carlo methods provide a practical way to estimate value function and derive optimal policies specifically in environments where a model is not available or feasible to construct.

# OBJECTIVES

The project aims to experiment with learning policies using the methods discussed above. We experiment with a 5 \* 5 gridworld problem and benchmark and compare learning algorithms. The objective of this project is to estimate the value function for each of the states in the gridworld with the provided scenarios, actions and rewards. We will try to achieve this by implementing the system of Bellman equations, utilizing the iterative policy evaluation for choosing better policy and executing value iteration for determining the optimal policy for the given conditions.

We also change the environment conditions adding some terminal states in our and using Monte Carlo method to learn an optimal policy for this modified gridworld problem. Using the Monte Carlo method, we implement it with exploring starts and without exploring starts but with epsilon-soft approach.

# EXPERIMENTS

Scenario 1:We consider a simple 5 \* 5 gridworld problem with each of the 25 cells representing a possible state of the world. The possible action for an agent is up, down left or right and if the agent attempts to step off the grid, the location of the agent remains unchanged. The gridworld has blue, green, red and yellow squares representing special states with various behaviours listed below:

* At the blue square, any action yields a reward of 5 and causes the agent to jump to the red square.
* At the green square, any action yields a reward of 2.5 and causes the agent to jump to either the yellow square or the red square with probability 0.5.
* Attempt to step off the grid yields a reward of −0.5 and any move from a white square to another white square yields a reward of 0.

From the scenario above, an agent with good policy should try to find the states with high value and exploit rewards at those states.

### Part 1:

#### 1.1. Consider a reward discount of γ = 0.95 and a policy which simply moves to one of the four directions with equal probability of 0.25. Estimate the value function for each of the states using:

#### Explicitly solving the system of Bellman equations:

To solve the Bellman equations explicitly, we can utilize the system of linear equations specifically for small, finite state spaces such as our scenario. This can be solved using matrix operations. We initialize ‘V’, ‘A’ and ‘b’ matrices which are 5\*5 matrix initialized to zeros representing value function, coefficient matrix of size 25\*25 and a constant vector of size 25. Then we construct coefficient matrix ‘A’ and constant vector ‘b’ based on Bellman equation for each state and the system of linear equations ‘Ax=b’ is solved to find the value function ‘V’ and finally reshape back to a 5\*5 grid format.

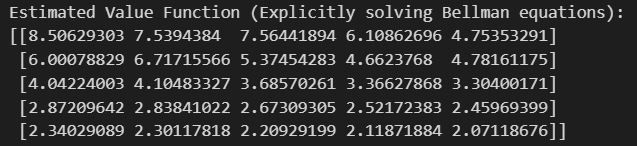


Figure 1: Explicitly solving Bellman Equation

The output of the Bellman equation solved by system of linear equations is the estimated value function for each state in the gridworld.

#### Using Iterative policy evaluation

To solve the Bellman equation using iterative policy evaluation, we utilize the ‘V’ matrix and define a function to calculate the next state and the associated reward given a state. We handle the boundary conditions, special state conditions and regular transitions. Then we execute a function with ‘V’ as zeros that performs iterative policy evaluation to update the value function V until it converges.

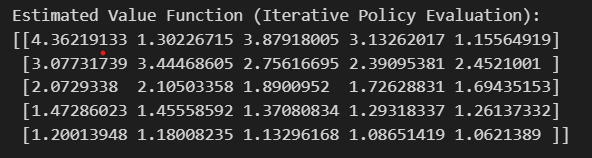


Figure 2: Using iterative policy evaluation for Bellman equation

The value function in the figure above is the output of solving the Bellman equation using iterative policy evaluation.

#### Using Value iteration

To solve Bellman equation using value iteration method, we assign a small threshold determining the convergence criterion represented by epsilon. We loop through states and make updates depending on whether the state has special transition rules or regular transition rules.

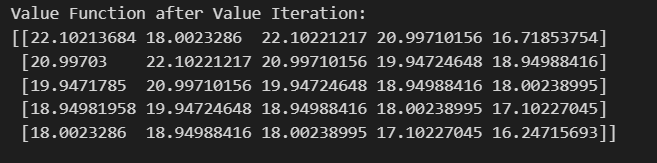


Figure 3: Using value iteration for Bellman equation

The value function in the figure above is the output of solving the Bellman equation using value iteration.

**Conclusion (Part 1): Which states have the highest value? Does this surprise you?**

All the output has relatively moderate values across the grid with maximum value at top-left corner. This value function represents the expected cumulative reward starting from each state, given the policy and the environment’s dynamics. Considering the stochastic nature of transitions and rewards, higher values indicate states from which the agent can expect higher future rewards. The values seem to be higher near the special states – blue and green squares and seem to be decreasing as it moves away from the high-reward states, showing the impact of distance and discounting affect of rewards.

Comparing the output of the three methods, we conclude that solving Bellman equation explicitly results in relatively moderate values across the grid while iterative policy has the lowest values throughout the grid and value iteration has significantly higher values across the grid.

The highest values always seem to occur at [0,0] hinting that the state benefits significantly from nearby rewarding states or favorable transitions. Additionally, for value iteration, the highest value also occurs at [1,1] indicating these states are particularly strategic, offering best long-terms rewards under optimal policy.

#### 1.2. Determine the optimal policy for the gridworld problem by:

#### Explicitly solving the Bellman optimality equation

#### Using policy iteration with iterative policy evaluation

#### Policy improvement with value iteration

To find the optimal policy for the gridworld problem by solving the Bellman equation in three given methods, we need to determine the actions that maximize the expected return from any state. And the resulting value functions from this process reflects the highest possible rewards from each state under the best possible actions. The explicit solving of the Bellman equation iteratively updates the value function to reflect the highest possible return by handing special cases and general cases, the policy iteration alternates between two phases – policy evaluation and policy improvement while the value iteration combines policy evaluation and improvement into a single step. The following output shows the optimal policy for each of the methods solving the Bellman equation.

A screenshot of a graph

Description automatically generated

Figure 4: Optimal policy determination for Bellman equation solving explicitly

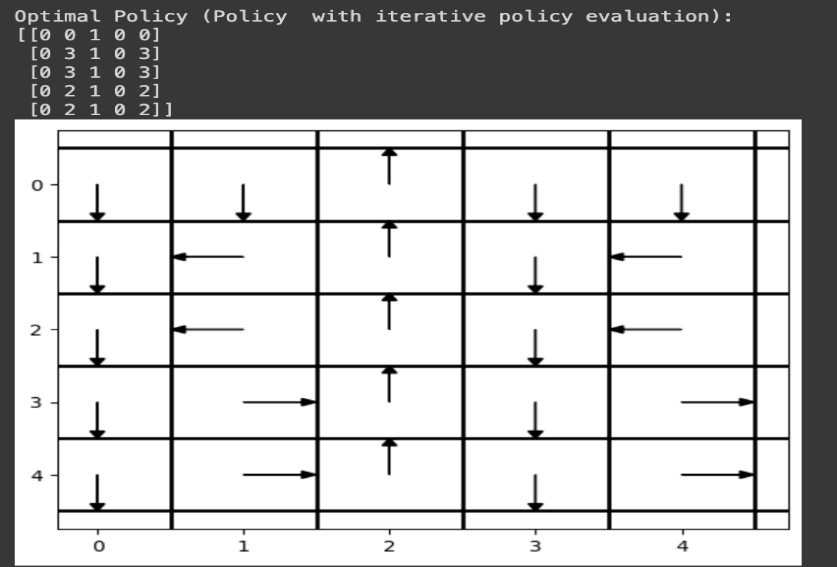


Figure 5: Optimal policy determination using iterative policy evaluation

A graph with arrows and numbers

Description automatically generated

Figure 6: Optimal policy determination using value iteration

The matrices above show the optimal policy for each of the three methods. Each entry indicates the best action to take from that particular state to maximize the agent’s expected reward. Bellman optimality and value iteration provide similar results since the aim of these methods is to maximize the utility while policy iteration shows a more conservative improvement patterns as there is refinement of policy based on evaluations.

Scenario 2:Change the environment a bit by adding some terminal states represented as the black squares. This gives rise to episodes where termination occurs once the agent hits one of the black squares. We will also assume, unlike in Part 1, that any move from a white square to a white square yields a reward of -0.2.

### Part 2

#### 2.1. Use the Monte Carlo method with (1) exploring starts and (2) without exploring starts but the ϵ-soft approach to learn an optimal policy for this modified gridworld problem. Use the same discount factor of γ = 0.95 as you have in the Part 1 above. You can start with a policy with equiprobable moves.

#### Monte Carlo with exploring starts:

Implementation of Monte Carlo with exploring starts ensures that all state-action pairs are visited sufficiently often as this method starts episodes from every state-action pair with non-zero probability. Monte Carlo with exploring starts choses the starting state and action randomly (sometimes deterministically cycling) which helps this method to overcome the exploration-exploitation dilemma during learning.

A screenshot of a computer

Description automatically generated

Figure 7: Monte Carlo with exploring starts (Optimal Policy)

A screenshot of a graph

Description automatically generated

Figure 8: Monte Carlo with exploring starts (Value function)

The above figures show the optimal policy and the associated value function from Monte Carlo method with exploring starts. The optimal policy matrix indicates the direction agent should move when it is in a particular state to maximize its long-term rewards. The value function output shows the maximum expected return value from each state under the optimal policy. The highest values are observed near the top of the grid, specifically near blue and green. The zero values represent the terminal state as no future rewards can be accrued once these states are reached, and the negative values likely represent the states when exited lead to less advantageous states.

#### Monte Carlo without exploring starts but with epsilon-soft approach:

Monte Carlo with epsilon-soft policy is designed to find optimal policy by ensuring continuous exploration by using an epsilon-soft policy, where ethe agent has a non-zero probability of choosing any action from each state. This avoids the potential pitfalls of limited exploration in Monte Carlo with exploring starts. This method ensures that all actions are chosen with a non-zero probability where epsilon is the probability of choosing a non-greedy action randomly which balances the exploration and exploitation of the model.

A computer screen with numbers and symbols

Description automatically generated

Figure 9: Monte Carlo with epsilon-soft approach

The above figure shows the optimal policy and the associated value function from Monte Carlo method with exploring starts. The optimal policy matrix suggests the best-known way to navigate the gridworld to maximize the rewards based on the learning from Q-values. The value function shows the maximum Q-value for each state and higher values indicate the states from which the expected return, starting from that state following the optimal policy, is higher. The higher values are generally observed away from the terminal states, which reflects the agent trying to stay away from terminal states. The zero or low value indicates the end of episodes with no further rewards.

Between the two approaches, Monte Carlo with epsilon-soft approach shows significantly more variation in policy with predominant directions and also shows significantly higher value estimates indicating better long-term outcomes under the Monte Carlo Epsilon-soft policy.

#### 2.2. Now use a behaviour policy with equiprobable moves to learn an optimal policy. Note here the dynamics of the world are known exactly, so you can compute the importance weights needed for this.

Implementation of Monte Carlo with behaviour policy with equiprobable moves allows the agent to choose all available actions with equal probability in each state. This uniform choice facilitates exploration of the state space without initial bias. For this method, we generate episodes using behaviour policy, update estimates and then update the policy using epsilon-greedy approach. The following image provides the output for Monte Carlo Importance Sampling method.

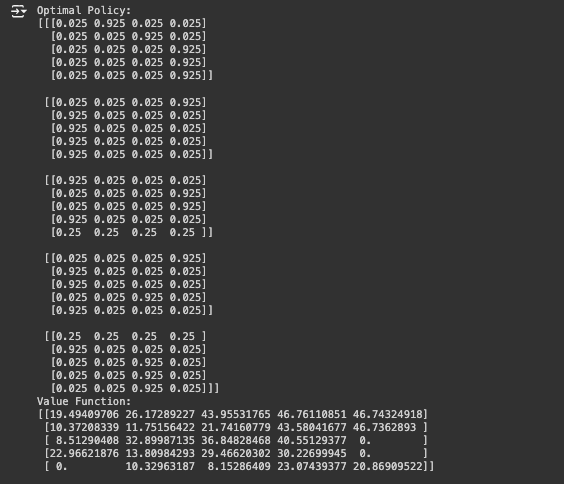
****

Figure 10: Monte Carlo Importance Sampling

The output above shows four different optimal policies and max value functions of those optimal policies. As seen, we have the optimal policy divided into sub-matrices such as [0.025, 0.925, 0.025, 0.025], indicates the probabilities of choosing each respective action from that state under the policy. This provides us the distribution over actions that can be taken from a particular state.

The value function is the max of all the value function matrices from the optimal policies. The higher values occur in squares adjacent to blue and green and the lowest values which is 0 occurs in terminal states and states adjacent to terminal states.

Comparison between all three approaches for Monte Carlo method, the values from Importance sampling show more varied than the previous methods reflecting a balance between long-term gains and risk. But overall, Monte Carlo Epsilon soft appears to offer the best overall strategy based on the results in terms of consistency and effectiveness in navigating the gridworld.

#### 2.3. Finally, let’s suppose that at every step, we permute the locations of the green and blue squares with probability 0.1, while preserving the rewards and transition structure as before. Use policy iteration to determine a suitable policy for this environment. How does it differ from the case where the squares stay where they are? This is a bit more of an open-ended question for you to think about how to address such problems.

In this scenario where the locations of green and blue squares permute with a probability of 0.1 at each step, traditional static policy solutions become less effective because the agent must adapt its strategy also to the expected changes in the environment and not only the immediate rewards hence adding a layer of complexity. Implementing policy iteration, we calculate value function for a given policy until it reaches a steady state and then update the policy at each state by choosing the action that maximizes the expected value based on the current estimate of the value function. The following provides the output for the given scenario.

A screenshot of a graph

Description automatically generated

Figure 11: Policy Iteration for dynamic gridworld with termination states (Optimal Policy)

A screenshot of a graph

Description automatically generated

Figure 12: Policy Iteration for dynamic gridworld with termination states (Value function)

From the output above, we can see that there a strategic diversity in the action selection. And the value function matrix provides high values all over the grid indicating highly advantageous states with 0 at the terminal states as no further rewards can be accrued. This indicates an accurate recognition of terminal states enhancing the realism and applicability of the policy.

We also experimented with some hyper parameters and received the following results.

Tested num\_episodes=1000, score=1055.2874999864196  
Tested num\_episodes=5000, score=1057.6624999877436  
Tested num\_episodes=10000, score=1057.6624999906326  
Tested num\_episodes=20000, score=1057.6624999933765  
Tested num\_episodes=50000, score=1057.6624999877436  
Best Number of Episodes: 20000

In the output above, num\_episodes define how many episodes or trails the learning algorithm uses to train the policy and the score represents the cumulative reward obtained by the agent over the course of the episodes. We can see that by 5000 episodes, the score stabilizes around 1057 indicating a point of diminishing returns. Hence, choosing 20000 seems the best number of episodes that is likely to avoid overfitting and has ample learning experiences.

We also ran the same scenario with the probability of 0.5 for the permutation of the green and blue squares and we did not find any difference between the output from the two different probability values.

In a scenario where the squares stay where they are (in a static environment), we concluded the following result.

A screenshot of a graph

Description automatically generated

Figure 13: Policy Iteration for static gridworld with termination states (Optimal Policy)

A screenshot of a graph

Description automatically generated

Figure 14: Policy Iteration for static gridworld with termination states (Value function)

**Conclusion:** The results above indicate there is a slight difference in the optimal policy between dynamic and static environments where dynamic environment showcases a varied policy suggesting more adaptability whereas static environment’s policy is consistent over multiple runs. The value functions in both environments reflect similar values indicating that despite the change, the overall potential rewards or costs for each remain the same. And finally, both environments correctly reflect zero values in terminal states. Hence

# CONCLUSION

The project utilizes different reinforcement learning methods in a 5\*5 gridworld environment and provides valuable insights to the dynamics of policy learning under various conditions. Implementation of Bellman equations and Monte Carlo methods followed by various evaluation and optimization methods has given insights to unique strengths and adaptation strategies according to specified changes in the environment with terminal states and dynamic changes.

The Bellman equation through explicit and iterative methods has shown that while explicit methods provide quick convergence, iterative methods offer flexibility in terms of policy adjustment over time. Monte Carlo methods with exploring starts and epsilon-soft methods have shown the affect of different exploration strategies on the outcome and robustness of the policies. Also, the adaptation to dynamic environments of Monte Carlo and policy iteration approach adapted well to the changes.

Therefore, we can say that the project has successfully demonstrated the applicability of foundational reinforcement learning techniques in a controlled but varied setting.