Tabular Methods - Policy Optimization with On-Policy First-Visit Monte Carlo Control, On-Policy SARSA, and Off-Policy Q-Learning

Ishaan Kandamuri

I. Introduction

For this assignment, three different model-free RL algorithms: on-policy first visit MC control, SARSA, and Q-learning were implemented to optimize a policy π for the Gymnasium FrozenLake-v1 environment. The task centered around navigating a 4x4 grid, where each tile can be either solid ice, a hazardous hole, the starting point, or the goal. The agent always begins at the top-left corner (0,0) and aims to reach the bottom-right corner (3,3) by moving either left, right, up, or down across 16 possible tiles. This environment is modeled as a Markov Decision Process (MDP). When `is_slippery` is set to true, the intended direction of motion occurs with probability $\frac{1}{3}$, while the other $\frac{2}{3}$ of the time the agent slides into one of the perpendicular directions, each with probability $\frac{1}{3}$. If the agent tries to move outside the grid or steps into a hole, the transition function renders it stationary (no change in state). In this case, the reward remains 0 unless the movement ends in the goal state, for which the agent receives a reward of 1. Paths blocked by holes or grid boundaries ensure that invalid transitions do not reward the agent and do not result in movement, maintaining the dynamics of the episode. The following sections detail the process behind optimizing a policy π for a given MDP and how this can be affected by a slippery or non-slippery Frozen Lake.

II. Markov Decision Process Problem Definition

The Frozen Lake map used for this experiment is shown below. For this experiment, let us also state that `slippery = True` for completeness.

$$\begin{bmatrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & G \end{bmatrix}$$

A. State Space

The state space S consists of all the positions in the Frozen Lake map grid. This is shown in row-wise format below:

$$S = \{0, 1, 2, \ldots, 15\}$$

These states correspond to a grid index on the map, where each state (0-indexed) corresponds to a Start, Frozen, Hole, or Goal tile in a left-to-right, top-to-bottom order. Thus, the array `["SFFF", "FHFH", "FFFH", "HFFG"]` maps exactly to the Frozen Lake configuration above.

B. Action Space

The action space A is defined as:

$$A = \{0, 1, 2, 3\}$$

where `0 = Left`, `1 = Down`, `2 = Right`, and `3 = Up`.

C. Transition Function of Probabilities

Let $P(s' \mid s, a)$ denote the probability of transitioning from state s to state s' using action a. For terminal states (holes H or goal G):

$$P(s' = s \mid s, a) = 1, \quad \forall a \in A$$

For non-terminal states ($s \in \{0, 1, 2, 3, 4, 6, 8, 9, 10, 13, 14\}$):

$$\begin{cases} P(s' = \text{Intended Direction}) = \frac{1}{3} \\ P(s' = \text{Move Up}) = \frac{1}{3} \\ P(s' = \text{Move Down}) = \frac{1}{3} \end{cases}$$

D. Discount Factor

The MDP has a constant discount factor:

$$\gamma = 0.95$$

E. Reward Function

The reward function is:

$$R(s, a, s') = \begin{cases} 1 & \text{if } s' = 15 \text{ (goal)} \\ 0 & \text{otherwise} \end{cases}$$

III. Methods

The main objective is to obtain the optimal value function $V^*()$ and the optimal policy $\pi^*(|)$ using three distinct model-free approaches: on-policy MC control, SARSA, and Q-learning. These model-free methods resolve the issues caused by continuous state and action spaces. Even when the number of states and actions is very large, the optimal value function and policy can be estimated through a data-driven approach. In the following discussions, each of the three model-free approaches is outlined.

A. Parameters

- ϵ (Epsilon-Greedy Parameter): Determines how often the agent explores by picking a random action instead of the current estimated best action.
- γ (Discount Factor): Used to discount future expected returns (set to 0.95).
- `slippery`: If `True`, follow the defined MDP from Section II; otherwise, transitions are deterministic.

B. Pseudocode - First-Visit Monte Carlo Control: On-Policy

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Fig. 1 Sutton and Barto 5.4: On-Policy First-Visit MC Control.

C. Pseudocode - SARSA: On-Policy, TD(0)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Fig. 2 Sutton and Barto 6.4: On-Policy SARSA

D. Pseudocode - Q-Learning: Off-Policy, TD(0)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Fig. 3 Sutton and Barto 6.5: Off-Policy Q-Learning

E. Code Implementation

The previously shown pseudocode algorithms were implemented using Python, the gymnasium library, the numpy library, and the matplotlib library for plotting. Additionally, the Frozen Lake environment (slippery or not) had to be unwrapped to access the transition and reward function. The policies that correspond to each agent are outputted as lists. The value functions are outputted as 4x16 matrices.

IV. Results

A. Mean Evaluation Return for Each Trained Agent

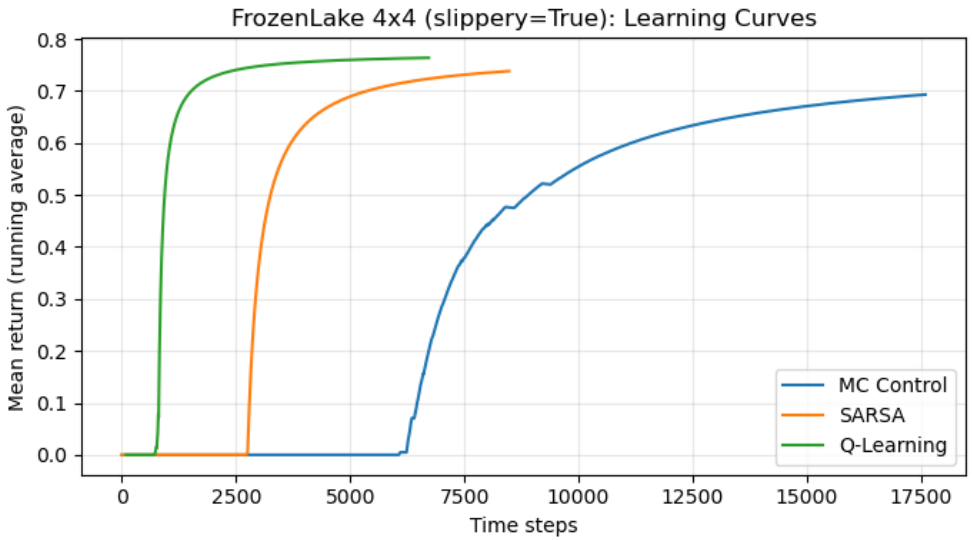

Fig. 4 Mean Evaluation Return - Slippery

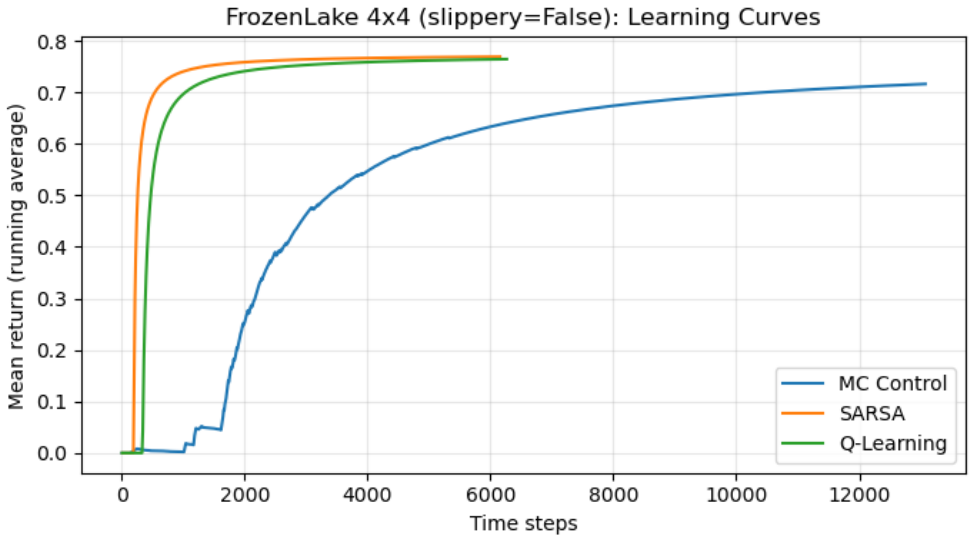

Fig. 5 Mean Evaluation Return - Not Slippery

In Figure 3, it can be observed that all three algorithms gradually improve their evaluation returns over time despite the inherent stochasticity of state transitions. Q-learning demonstrates faster learning and higher asymptotic performance than SARSA (orange) and Monte Carlo control (blue). This is because Q-learning is off-policy, and it can aggressively exploit optimal actions via the argmax function for the action-value function Q for its target. This leads to higher variance but faster convergence toward the optimal value. SARSA, being on-policy, updates using the next action actually taken under the ϵ-greedy policy, which makes it more conservative and results in a smoother but slower learning. Monte Carlo control also exhibits the slower early learning due to its reliance on complete episodes before updates,

though it eventually approaches near optimal returns with many more time steps. Overall, Q-learning achieves the best balance between sample efficiency and asymptotic performance under the Frozen Lake's slippery stochastic dynamics.

In Figure 4, all three algorithms exhibit faster and more stable learning compared to the slippery case. The deterministic transitions allow Q-learning and SARSA to converge rapidly to near-optimal performance, with Q-learning maintaining a slight edge in both speed and final return due to its efficient off-policy updates. Monte Carlo Control again demonstrates slower early learning, as it requires complete episodes to update its values, but eventually approaches optimality with sufficient time steps. Overall, the deterministic characteristics of the non-slippery environment enables all methods to learn more efficiently, with Q-learning achieving the best combination of sample efficiency and asymptotic performance, while SARSA closely follows.

B. Policy and State Value Function Output for Each Trained Agent

Below are the policy and value function outputs for each trained agent.

```
SLIPPERY: True
MC Policy: [1 0 0 2 1 0 3 0 2 1 0 1 1 2 2 0]

Q Policy: [1 3 0 3 1 0 2 3 2 2 1 1 1 0 2 0]

SARSA Policy: [2 2 1 1 1 1 1 2 2 2 1 1 2 1 2 1]
```

Fig. 6 Policy Output for All Three Methods - Slippery

```
SLIPPERY: False
MC Policy: [1 0 0 3 1 0 3 1 2 1 0 2 0 2 2 2]

Q Policy: [1 0 3 3 1 0 2 1 2 2 1 2 0 2 2 1]

SARSA Policy: [1 1 0 0 1 0 1 2 2 2 1 1 0 3 2 0]
```

Fig. 7 Policy Output for All Three Methods - Not Slippery

```
MC Value Function:
  [[4.57541906e-01 7.73780937e-01 4.37868249e-01 2.39979576e-01]
  [4.61274982e-01 9.58629311e-04 1.00680137e-04 9.38119569e-04]
  [8.35199220e-02 7.87826855e-02 3.41246076e-02 8.09989136e-02]
  [4.36311259e-01 2.86017964e-01 4.90964449e-01 3.04856388e-01]
  [2.71440821e-01 8.14506250e-01 1.02028781e-03 2.66968132e-01]
  [6.95414729e-01 5.28541848e-01 4.20386161e-01 4.96472024e-01]
  [1.60262555e-01 1.04215812e-01 1.61019719e-01 1.73917513e-01]
  [7.59026444e-01 2.80460162e-01 5.17256934e-01 3.25052765e-01]
  [3.87753138e-01 5.60679834e-02 8.57375000e-01 3.15997384e-01]
  [6.18830833e-01 9.02500000e-01 5.03618173e-01 5.58091303e-02]
  [7.69452874e-01 4.13155429e-01 5.57797821e-01 4.72557611e-01]
  [2.33318378e-01 6.61859245e-01 6.31662438e-01 6.09989098e-01]
  [6.65324257e-01 9.81230796e-01 5.62939929e-01 6.37806262e-01]
  [1.68449835e-01 8.30450504e-01 9.50000000e-01 7.57289040e-01]
  [6.32467332e-01 7.42480427e-01 1.00000000e+00 5.34761883e-01]
  [7.27604552e-01 4.89471811e-01 3.13655343e-01 5.08926837e-01]]

SARSA Value Function:
  [[0.2574142  0.26141041 0.77378094 0.04752653]
  [0.14117961 0.24556421 0.81450625 0.26290325]
  [0.27378605 0.857375   0.15068051 0.27251922]
  [0.37852667 0.78051457 0.6351293  0.23245277]
  [0.26759763 0.28158201 0.26134288 0.26683724]
  [0.48973187 0.71251138 0.04388428 0.43693061]
  [0.27066963 0.9025     0.23623498 0.27997387]
  [0.58872313 0.8018518  0.81604187 0.25539744]
  [0.27115436 0.23324866 0.36055599 0.27406944]
  [0.27749857 0.26711253 0.56453654 0.0738678 ]
  [0.27769298 0.95       0.09711504 0.09312619]
  [0.21503226 0.67203031 0.49536442 0.03008243]
  [0.29633901 0.28852884 0.61999331 0.12023666]
  [0.25541001 0.2799047  0.27748615 0.1680127 ]
  [0.27835768 0.27762442 1.         0.2803508 ]
  [0.18132491 0.89614504 0.06673286 0.30153324]]

Q Value Function:
  [[0.38053381 0.77378094 0.37598273 0.07846903]
  [0.10254746 0.22794578 0.27313528 0.27536348]
  [0.27407991 0.24661566 0.27271058 0.27291399]
  [0.27228378 0.02449091 0.27024006 0.27393802]
  [0.00485168 0.81450625 0.32030511 0.00192469]
  [0.94657747 0.31157645 0.6827014  0.79710261]
  [0.1106717  0.02507835 0.12029759 0.05776331]
  [0.80806484 0.48832833 0.36518589 0.9931468 ]
  [0.23275139 0.19320786 0.857375   0.22514943]
  [0.16230534 0.18834783 0.9025     0.69349208]
  [0.70916108 0.95       0.33367015 0.26584932]
  [0.59031781 0.79342798 0.23071814 0.65191876]
  [0.35672136 0.96425466 0.5584514  0.8509143 ]
  [0.94588066 0.57155287 0.05544027 0.60031507]
  [0.44150515 0.32253215 1.         0.64994906]
  [0.9892063  0.67036846 0.1994473  0.84442758]]
```

Fig. 8 Value Function Output for All Three Methods - Slippery

```
MC Value Function:
  [[0.35004932 0.77378094 0.21029977 0.02680741]
  [0.27299836 0.01186753 0.01342173 0.15073524]
  [0.07749236 0.07304334 0.05439265 0.07141143]
  [0.15660394 0.01991918 0.06475944 0.18262081]
  [0.4082301  0.81450625 0.00629347 0.37332255]
  [0.75676041 0.30417361 0.05943668 0.19485135]
  [0.29404478 0.29717009 0.34548769 0.39483153]
  [0.1386962  0.57656071 0.41103026 0.46612156]
  [0.64353057 0.03381516 0.857375   0.31341357]
  [0.63516849 0.9025     0.57794114 0.0597608 ]
  [0.74125523 0.18814442 0.22247129 0.31191087]
  [0.69855576 0.7134458  0.71849847 0.16472962]
  [0.93999122 0.93250617 0.68790474 0.52755461]
  [0.20431708 0.61219945 0.95       0.73822055]
  [0.79519975 0.78129839 1.         0.64965498]
  [0.10190095 0.59756375 0.71325769 0.092845841]

SARSA Value Function:
  [[0.17765383 0.77378094 0.08136185 0.08361533]
  [0.39475362 0.80912682 0.56288499 0.52460364]
  [0.8654551  0.1717157  0.22059699 0.05593376]
  [0.54286087 0.12216638 0.42075775 0.09795267]
  [0.18151564 0.81450625 0.1563057  0.1841025 ]
  [0.9712751 0.64327216 0.87556331 0.67926308]
  [0.1336055  0.16689665 0.12266908 0.1156938 ]
  [0.28725905 0.85905602 0.89613207 0.47548779]
  [0.185978   0.16230132 0.857375   0.18762621]
  [0.15954135 0.12095294 0.9025     0.0217058 ]
  [0.16576611 0.95       0.13276611 0.16542234]
  [0.25712152 0.74702371 0.1823597  0.53602116]
  [0.89504939 0.84580968 0.48686677 0.36396619]
  [0.13782598 0.04063652 0.09669898 0.16864947]
  [0.17037718 0.16886752 1.         0.16901972]
  [0.75596583 0.4130171  0.04291126 0.21938208]]

Q Value Function:
  [[2.09409717e-01 7.73780937e-01 1.88033226e-01 2.10277248e-01]
  [7.50812632e-01 2.55542005e-01 5.02243229e-01 4.90107384e-01]
  [3.35325614e-01 4.29986501e-01 8.77651710e-01 9.92461482e-01]
  [6.25265077e-01 5.56496146e-01 1.45144444e-01 8.54117365e-01]
  [1.09977816e-01 8.14506250e-01 9.06315506e-02 7.07362017e-02]
  [2.88112709e-01 2.51823967e-01 3.75675196e-02 3.09860725e-02]
  [8.99646102e-01 3.02090866e-01 9.30414622e-01 3.42135591e-02]
  [1.12976627e-01 8.23566268e-01 1.56866270e-01 5.12265877e-01]
  [6.56220441e-02 1.23298274e-01 8.57375000e-01 1.39732104e-01]
  [6.85697165e-01 7.34640747e-01 9.02500000e-01 1.44158219e-01]
  [3.46728830e-01 9.50000000e-01 5.84721854e-01 1.63177328e-01]
  [2.41122252e-01 4.87020858e-01 9.25977328e-01 2.46354878e-02]
  [7.69636990e-01 1.37958068e-01 7.02760034e-01 2.47237885e-01]
  [2.77096888e-01 5.80264025e-01 7.14282615e-01 8.01180822e-04]
  [9.68174444e-01 4.69927644e-01 1.00000000e+00 4.87286029e-01]
  [8.79287761e-01 9.51330164e-01 1.05610232e-01 8.84023949e-02]]
```

Fig. 9 Value Function Output for All Three Methods - Not Slippery

V. Conclusions

This report compared three RL methods: Monte Carlo, SARSA, and Q-learning algorithms in the 4×4 FrozenLake tabular environment under both stochastic (slippery) and deterministic (non-slippery) environments. Q-learning proved to be the most sample-efficient and consistently reached near-optimal performance fastest, especially when transitions were deterministic. SARSA showed smoother, more stable learning but converged more slowly—particularly in the presence of random transitions, which demonstrates its cautious, on-policy nature. Monte Carlo control improved steadily but was initially slower since it updates at the end of each episode, and with sufficient experience, it achieved returns close to the optimum. All algorithms performed better and learned faster in the deterministic (non-slippery) setting, which is expected. However, for small, discrete tasks like FrozenLake, Q-learning is best when prioritizing fast, optimal learning. Regarding the other two methods, SARSA is preferable when stability during training matters, and MC control is good for establishing a simple and reliable baseline given enough training data.

References

[1] Richard S. Sutton and Andrew G. Barto (2018). Reinforcement Learning: An Introduction, MIT Press