Solving Markov Decison Processes with Value Iteration, Policy Iteration and Reinforcement Learning

Damian Abasto. damian.f.abasto@gmail.com

***Abstract*—Two Markov Decision Processes (MDPs) are solved via Value Iteration and Policy Iteration, as well as reinforcement techniques via Q-learning, where the model for the environment (transition probabilities and reward function) are considered unknown. The first MDP is called "Frozen Lake" from OpenAI `gym` environment [?], while the second consists of a stock trading environment where an agent tries to invest optimally on a single stock that exhibits mean reversion (custom environment created by inheriting from the `gym` environment).**

I. Markov Decision Processes

We proceed to describe briefly the two environments used throughout this article.

A. Frozen Lake

We solved two Frozen Lake environments, a large 8x8 grid, and a smaller 4x4 grid. Frozen Lake consists of an 8x8 grid where the agent tries to travel from the start position in the (0,0) coordinate to the end goal at (7,7). The agent actions are to move up, down, right or left, and as he moves through the grid he receives a -1 reward for each step. The agent receives a reward of +10 if he reaches the end goal. This incentives the agent to minimize the distance traveled from the start cell to the end goal. The cells marked by "X" represents holes and the agent gets penalized by a reward of -10 should he land in one of those cells.

The environment is stochastic, in that the agent has only a 80% chance of moving in the direction he desires, with a 10% chance of moving on either direction perpendicular to the intended direction of movement. This probability structure and rewards encourages the agent to try to avoid as much as possible going near the holes, as it has a non-zero probability of falling into them. Figure 1 shows as example of the agent after taking the steps down and right. In the first step down the agent actually went left, bumping itself against the wall. The move did not take place, and only the right step was successful.

While the customized implementation and visualization tools for the Frozen Lake environment are from [?], we have implemented Value Iteration, Policy Iteration, Q-Learning as well as the customized stock environment with mean reversion.

B. Stock trading environment

We implemented a stock trading environment with mean reversion inspired by a Geometric Brownian Motion. In this environment we have a stock S_t that evolves in time t by either going up by a factor of u or down by a factor d as follows:

$$S_{t+1} = \begin{cases} u \times S_t, & \text{with prob } p \\ d \times S_t, & \text{with prob } 1-p \end{cases} \tag{1}$$

where $u = e^{-\sqrt{\sigma \delta t}}$, $d = e^{\sqrt{\sigma \delta t}}$, σ is the volatility of the underlying stock, r is the risk-free interest rate, and the probability of going up $p = \frac{e^{r\delta t} - d}{u-d}$, where $\delta t = T/N$ is the time step size, T is the total time of the simulation, and N the total number of time steps. Notice that because $d \times u = 1$, the possible array of steps that the stock can make forms a binary tree, with the origin node at S_0, and bifurcating from there on as time goes from $t = 0, .., T$. The choices for u, d and p are from the risk neutral approach to value options, whereby an agent can replicate the payoff of a call option of strike K, given by $\Pi(S_T) = max(S_T - K, 0)$ by forming an instantaneous risk-less portfolio made of stocks and money market account $B(t) = B_0 e^{-r \times t}$. For more details see [**?**].

As we increase (decrease) the number of steps N (step size) of the simulation, the number of states grows as $O(N^2)$. In the limit of $N \to \infty$, the price dynamics of S_t follows a

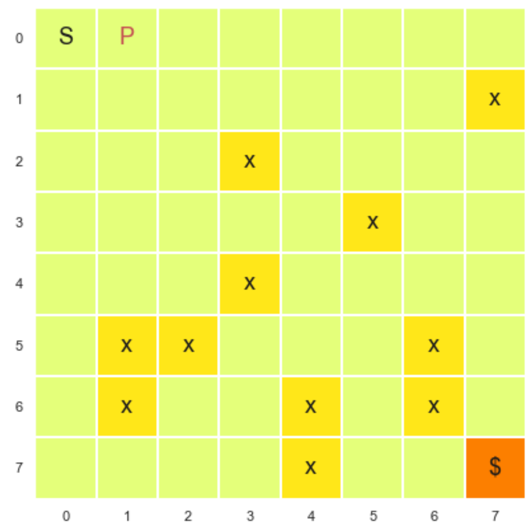


Fig. 1: Frozen Lake Environment we used for Planing and Reinforcement Learning. Agent starts at (0,) and tries to move to (7,7) as quickly as possible, while avoiding the "holes" marked with "X".

geometric Brownian motion, which is used in the context of the Black-Scholes setting to pricing options.

In terms of actions, the agent can take three actions: it can buy the stock at the price S_t, with corresponding reward $-S_t$, it can sell the stock at time t for a reward of $+S_t$, or it can hold the stock, with zero reward. To make the system Markovian and keep it simple we assume the agent can only hold one stock at a time, and it cannot short sell or borrow. The agent then will try to maximize the total discounted reward by buying and selling the stock, potentially many times across the evolution of S_t.

The total number of states is therefore $N(N+1)$, since it is the number of notes in a binary tree, given by $1/2N(N+1)$, times 2, because the agent either does not have or has at most one stock. The state the agent is at any time is described by the tuple (time, stock evolution, purchased stock), where the last variable is a boolean indicating whether or not the agent has purchased the stock.

Figure 2 shows a random stock path for $N = 100, S0 = 10, T = 1, r = 2\%, \sigma = 0.2$, with an agent that buys and sells the stock. Buys are marked in green, sells in red. This environment has $N(N+1) = 10,302$ total states.

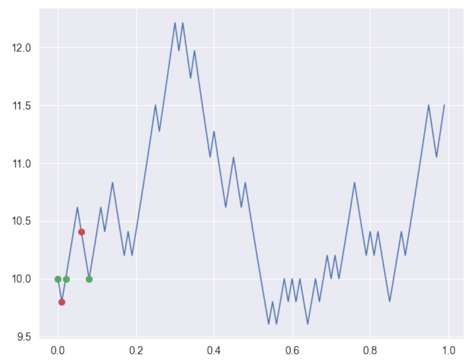


Fig. 2: Stock trading environment, which we implemented via subclassing the gym package. Random stock path for $N = 100, S0 = 10, T = 1, r = 5\%, \sigma = 0.2$, with an agent that buys and sells the stock. Buys are marked in green, sells in red.

II. Solving the MPDs

A. Methods

We applied value iteration, policy iteration and Q-learning to the two MPDs we considered. While policy iteration and value iteration are planning methods that rely on a complete specification of the environment dynamics, given by the transition probabilities $p(s', r|s, a)$ and the reward function $r(s, a)$, Q-learning is a reinforcement learning algorithm that does not depend on the model of the environment being known, and instead, relies on experience to learn the optimal value function for each possible action, as the agent interacts with the environment. As we will see in our experiments, Q-learning tends to require longer times for convergence and requires more hyperparameter tuning, as it lacks the information of the environment encapsulated in $p(s', r|s, a)$ and the reward function $r(s, a)$, which value iteration and policy iteration have.

Each algorithm needs a criteria for convergence when running. For value iteration the convergence criteria is given in terms of the maximum difference Δ, across all states s, between the value function $V(s)$ before and after the one-step look ahead, across all states s:

For Value iteration (see [?], section 4.4):

$$v \leftarrow V(s) \tag{2}$$
$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r, |s, a)[r + \gamma V(s')] \tag{3}$$
$$\Delta \leftarrow max(\Delta, |v - V(s)|, \forall s \tag{4}$$

Therefore, value iteration focuses on estating the value function at every state and stopping when the change in value over each iteration dropped below the threshold θ, that is, $\Delta < \theta$.

The convergence criteria for policy iteration is different. Policy iteration consists of a loop between policy evaluation and policy improvement, where both value function $V(s)$ and policies $\pi(s)$ are being estimated in a single algorithm. In Value iteration, the policy $\pi(s)$ is computed after the value function is outputed by Value Iteration. For the policy evaluation step, the stopping criteria is somewhat similar than value iteration.

For Policy evaluation (see [?], section 4.3):

$$v \leftarrow V(s) \tag{5}$$
$$V(s) \leftarrow \sum_{s',r} p(s', r, |s, \pi(s))[r + \gamma V(s')] \tag{6}$$
$$\Delta \leftarrow max(\Delta, |v - V(s)|, \forall s \tag{7}$$

and we stop when $\Delta < \theta$. We took $\theta = 0.0001$ to be our criteria for convergence, and $\gamma = 1.0$, so no discounting. Notice that for Value Iteration you take the maximum across all possible actions a in the one step look-ahead, while for policy evaluation the action you take comes from the policy $\pi(s)$ that the policy iteration algorithm is trying to estimate concurrently. We initialized the value function to be all zeroes, while the policy was initialized to random values, normalized so that the sum across all actions equals one, and we fixed the random seed to ensure reproducible results.

After policy evaluation comes policy improvement, where the policy is updated according to the value estimated in the policy evaluation step. The loop between policy evaluation and policy improvement stops whenever the optimal action according to the policy $\pi(s)$ being estimated in the policy improvement stops changing.

Q-learning is an off-policy reinforcement learning approach that does not rely on the specification of a model, contrary to planning (value iteration or policy iteration). Instead, it performs updates to the $Q(S, A)$ matrix as the agent explores the environment and receives new information in the form of tuples (s, a, r, s'). Since Q-learning does not have direct access to the model or the rewards function, it takes much longer to converge. We have run Q-earning over 50,00 episodes, where one episode is defined as a full round of interactions with the environment until the interaction is done (either because we have landed in the goal or fell into a hole (Frozen Lake), or we have finished the full evolution of the stock price).

The main update in Q-learning is given by (see [?], page 131):

$$Q(S, A) \to Q(S, A) + \alpha_t[R + \gamma \max_a (Q(S', a) - Q(S, A))] \tag{8}$$

, where α_t is the adaptive learning rate that decreases with the episode index t. We have implemented the Q-learning algorithm with various schemes for the ϵ greedy search. ϵ greedy encourages the agent to explore the environment by taking random actions that are not in accordance with the $Q(S, A)$ matrix (that is $a = argmaxQ(s, a)$ when the agent is at state s). The action selected is as follows:

$$a = \begin{cases} \text{argmax}_a Q(s, a), & \text{with prob } 1 - \epsilon \\ \text{random a}, & \text{with prob } \epsilon \end{cases} \tag{9}$$

where $\epsilon \in [0, 1]$. This is a way to balance the exploration-exploitation dilemma of Reinforcement Learning: the agent needs to exploit the Q values as it is learning about the environment, while at the same time it needs to explore actions and other parts of the state space that may not be able to reach otherwise, have it only follows the greedy actions $a = \text{argmax}_s Q(s, a)$.

Greedy in the limit of infinite exploration (GLIE) means that if all the state-action pairs are explored infinitely many times, then the policy that we find via the $Q(s, a)$ matrix is the greedy policy in the limit. The ϵ-greedy scheme in Eq.9 is GLIE if ϵ reduces to zero in the limit of infinite episodes.

We have explored two ways of adjusting ϵ:

- Constant scale ramp-down: $\epsilon_k = min(\epsilon_0 \times \delta\epsilon^k, \epsilon_{min})$, where $k \in 0, 1, 2, ...$ is the episode number, ϵ_0 the initial exploration rate (which we always took equal to 1.0), $\delta\epsilon$ the rate of decay, and ϵ_{min} a minimum exploration rate (which could be zero).
- Hyperbolic: $\epsilon_k = 1/(1 + k)$, where $k \in 0, 1, 2, ...$, with k indicating the episode.

According to the theory, only if $\alpha_t \to 0$ for $t \to infty$ we get a true expected value and convergence of Q-Learning to a solution of the Bellman equation. As we will show, in practice, we have found better results (in terms of rolling averages of rewards), if we kept α constant.

We have run the Q-leaner across the following combination of hyperparameters:

- $\alpha = 0.01, 0.05, 05$
- $\gamma = 0.99, 0.95$
- $\delta\epsilon = 0.9995, 0.999, 0.99, 0.9$
- $\epsilon_{min} = 0.0, 0.05$
- Epsilon Schedulers: Constant scale ramp-down, Hyperbolic

Across all 48 possible combinations of the above hyperparameters, we noticed across all the MDPs run that only a small subset of these hyperparameters resulted in good performance.

Performance for Q-learning was measured primarily in terms of the *rolling average* reward of the agent during the last 100 episodes, as a function of the number of episodes. We executed the algorithm for a fixed 50,000 episodes across all MDPs. The reason to take rolling averages is because each episode is stochastic and can result in large variations in the rewards per epsiode that the agent achieves. By focusing on the rolling average during the last 100 episodes we can more clearly see the trend of the rewards. If the agent is learning successfully, we should see an increase in the rolling average with the number of episodes. Notice that because the environments are stochastic and the agent only success taking a given action with probability less than 1, an agent can still fall on a hole for the Frozen Lake environment even if it is following the optimal policy, due to the stochasticity of the environment, and therefore, don't achieve the maximum reward possible. So even with a "test" run, whereby we turn off the ϵ-greedy exploration and act fully greedy with respect to $Q(s, a)$ we still expect to see quite a bit of variation in the rewards that the agent obtains.

B. *Frozen Lake*

We applied value iteration, policy iteration and Q-learning to the Frozen Lake Environment. The evolution of the Value Function $V(s)$ by the number of iterations and across each of the cells of the Frozen Lake environment can be seen in Figure 3. The cells near the end goal have clearly higher values, which have propagated from the reward of +10 in the end goal, while as you go further out or near the holes, the values are clearly negative, reflecting the -10 reward for landing into the holes, or the -1 reward for each successive step. Value Iteration converges after 27 steps. Figure 4 shows the corresponding policy derived from the value function.

Figure 5 compares the value functions between Value Iteration (left) and Policy Iteration (right). While the value functions coincide within the two significant figures, the number of iterations to converge was larger for policy iteration compared to value iteration. Since both value iteration and policy iteration rely on having a complete specification of the model, including transition probability matrix and reward function, they both arrive at the same answer, however, because of the reasons mentioned, policy iteration takes longer, and both take longer to converge with the higher number of states (4x4 vs 8x8 Frozen Lake Environments), given that there are higher number of states to evolve through.

Table I shows the number of iterations and time to convergence for both policy and value iteration, for two Frozen Lake environments, 4x4 amd 8x8. To ensure consistency of the number of iterations for both value iteration and policy iteration, the number of iterations was the total number of times the algorithms went through the loop over all states s

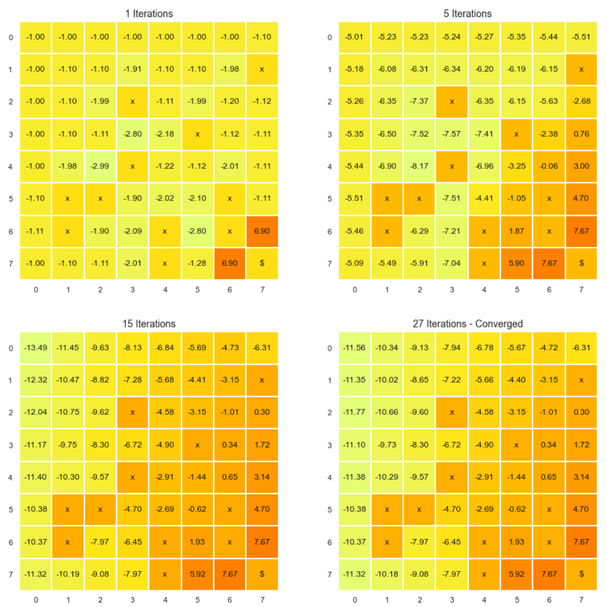


Fig. 3: Value function on each cell of the Frozen 8x8 environment, for 1, 5 and 15 iterations, and after convergence at 27 iterations. The Cells near the end goal have clearly higher values, which have propagated from the reward of +10 in the end goal, while as you go father out or near the holes, the values are clearly negative, reflecting the -10 reward for landing into the holes, or the -1 reward for each successive step.

and updating Δ, before stopping when $\Delta < \theta$. Value iteration converges faster than policy iteration, and the larger problem size of 8x8 does take more iterations, and therefore more time to convergence, for both value and policy iteration.

TABLE I: `f1` Number of iterations and running time for Value Iteration and Policy Iteration.

Problem Size	Method	Number of Iterations	Time to converge (ms)
4x4	Value Iteration	17	5.00
4x4	Policy Iteration	145	40.00
8x8	Value Iteration	27	39.01
8x8	Policy Iteration	1920	1767.61

Policy iteration clearly takes longer to converge than value iteration in this case because the algorithm is trying to evolve iteratively between the policy being estimated and the value function according to that policy. Since the policy itself is evolving, it takes longer for the algorithm to converge both the value function and the policy. Value iteration instead is *greedy* with respect to the actions in the one stop look-ahead, and instead of evolving with respect to a policy, it takes the best action across all actions possible. This is the difference between $V(s) \leftarrow \sum_{s',r} p(s', r, |s, \boldsymbol{\pi(s)})[r + \gamma V(s')]$ for policy iteration, versus $V(s) \leftarrow \mathbf{max_a} \sum_{s',r} p(s', r, |s, a)[r + \gamma V(s')]$ for value iteration.

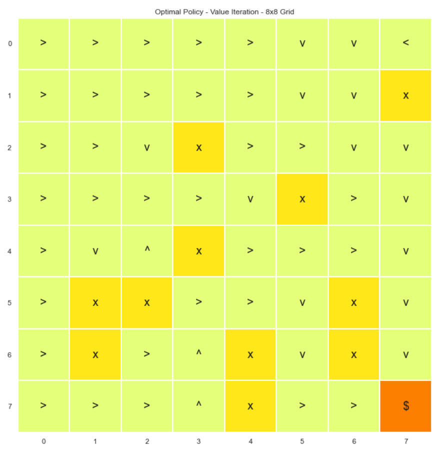
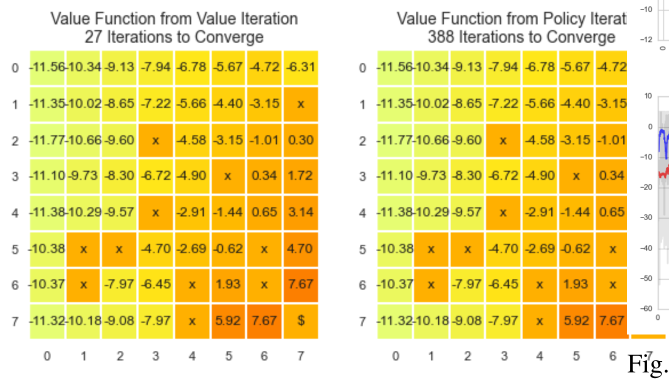


Fig. 4: Policy derived from the Value Function, shown in Figure 3

The average reward during the last 100 iterations for Q-learning across all for the 48 combination of hyperparameters is shown in Figure 6 (top), sorted in decreasing order. Each bar represents a unique combination of across all the hyper-parameters and ways to decrease epsilon. The large variation in rewards highights the sensitivity of the algorithm to the choice of those hyperparameters. Figure 6 (bottom), shows the evolution of the rewards and the rolling average reward for the best performing combination of hyperparameters for both Constant scale ramp-down, Hyperbolic. Notice there is much variability in the rewards per episode (grayed lines in the plot), given the randomness in the environment. There is an initial transient where the agent does seem to improve, as the rolling moving average of rewards increases initially with the number of episodes, but around 10,000 episodes the agent plateaus and does not seem to make any further improvements. Notice that by 10,000 iterations, ϵ 0 for both ϵ schedules, so at this stage the agent is only exploring at the rate ϵ_{min}.

If we rank all the different hyperpameter combinations in terms of the largest average rewards over the last 100 episodes, the best hyperparameters combinations were as follows:

- Constant scale ramp-down:
  - $\alpha = 0.05$
  - $\gamma = 0.95$
  - $\delta\epsilon = 0.9995$
  - $\epsilon_{min} = 0.0$
- Hyperbolic:
  - $\alpha = 0.5$
  - $\gamma = 0.95$
  - $\delta\epsilon = 0.9$
  - $\epsilon_{min} = 0.05$



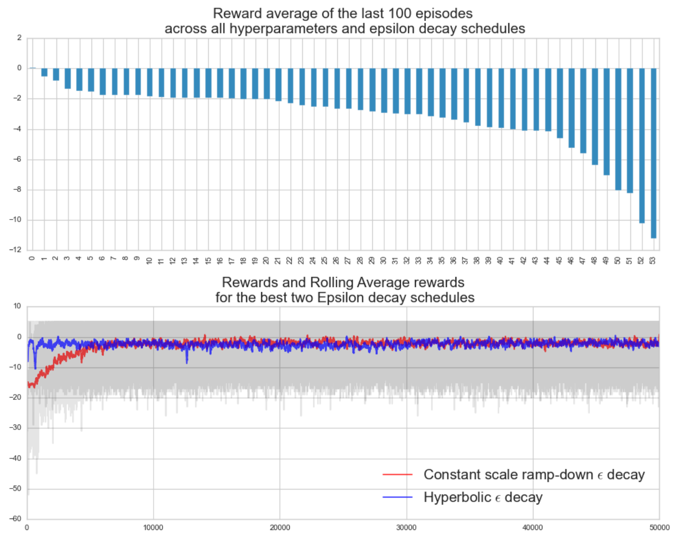


Fig. 6: Q-Learning for Frozen Lake 4x4. Top: average reward during the last 100 iterations for Q-learning across all for the 48 combination of hyperparameters and ϵ schedules. Bottom: evolution of the rewards and the rolling average reward over a window of 100 episodes for the best performing combination of hyperparameters for both Constant scale ramp-down and Hyperbolic.

Fig. 5: Comparison between the value functions from value iteration vs policy iteration. While the value functions coincide within the two significant figures, the number of iterations to converge was larger for policy iteration compared to value iteration

Figure 7 displays similar results than Figure 6 for Frozen Lake 4x4 when the learning rate α was adjusted as $\alpha_t = \min(\alpha_{\text{min}}, 1/t^{\omega})$. We explored across various possible values for min, and displayed in Figure 7 is the best combination we found, suing 10,000 episodes. However, notice that the rolling average of rewards is worse compared to keeping α fixed (Figure 6). Therefore, we decided to keep α constant for all our experiments. The agent seems to "forget" also the behavior it has learned, as we see a sudden slump in the performance, as measured by rolling averages of rewards across episodes for the hyperbolic ϵ scheme, around 4,000 episodes.

Figure 8 shows a comparison of run time between Q-learning and the planning algorithms value iteration and policy iteration. As we mentioned before, because the model is known for value iteration and policy iteration, they take much fewer iterations, and therefore less time to execute, than Q-learning.

Figures 9 and 10 compares the value function and corresponding policies from the Policy Iteration algorithm, versus the approximate solutions found by the best Q-learners across the two different ϵ schedules. Even though the value functions from the Q-learner are very different from the one found by

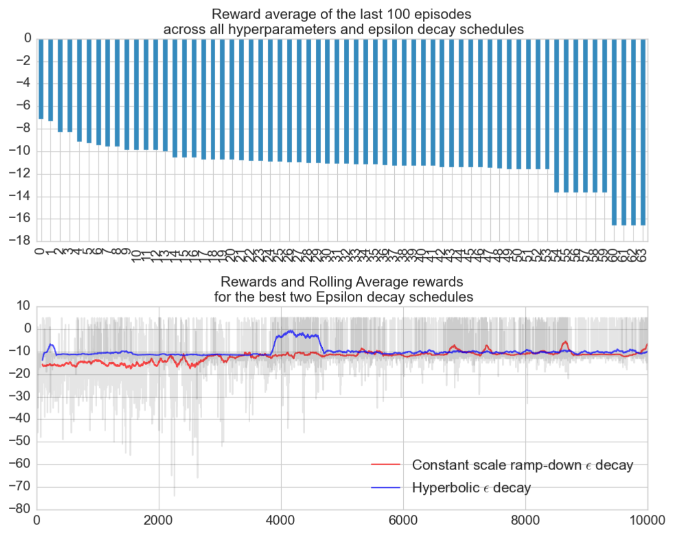


Fig. 7: Q-Learning for Frozen Lake 4x4 when the learning rate α was adjusted as $\alpha_t = \min(\alpha_{\text{min}}, 1/t^{\omega})$.

policy iteration, the *relative* values are still correct, producing the same final policy. If we take the value function from Policy Iteration as the benchmark (given that it has access to the environment model, while Q-learning does not and tries to estimate it), the average of the absolute differences across all cells in 1.12 for the Q-learner using the ramp down schedule, while the error increases to 1.79 for the Q-learning using the hyperbolic schedule. Given this criteria, we can

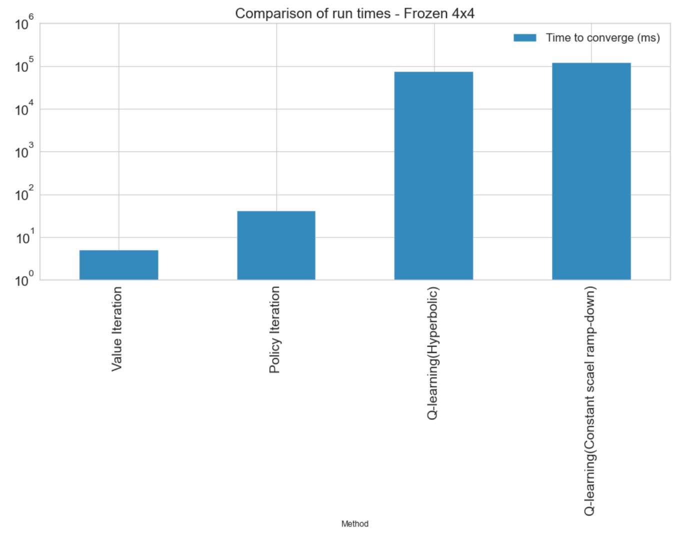


Fig. 8: Run times for Frozen Lake 4x4 across value iteration, policy iteration and Q-learning, for two different ϵ schedules.

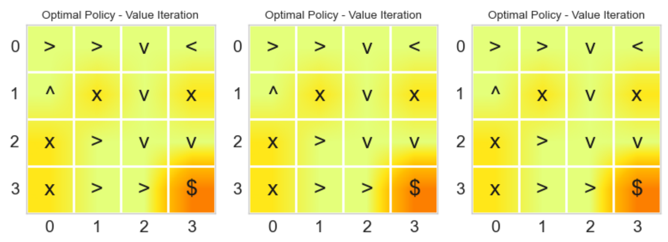

Optimal Policy - Value Iteration
Optimal Policy - Value Iteration
Optimal Policy - Value Iteration
> > v <
^ x v x
x > v v
x > > $

Fig. 10: Policy comparison from policy iteration and Q-learning, for two different ϵ schedules.

claim that the ramp down schedule was better in achieving a more accurate value function compared to the best hyperbolic schedule, across all the other hyperparameters tested.

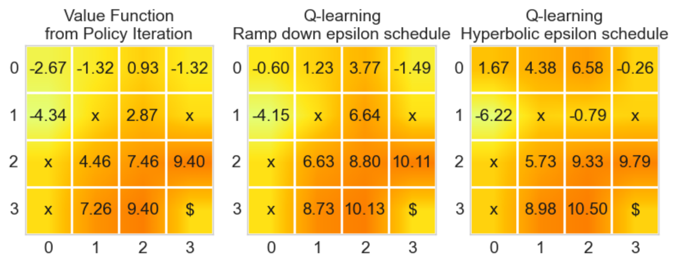


Fig. 9: Value function comparison from policy iteration and Q-learning, for two different ϵ schedules.

Figure 11 shows similar results for the Q-learner for the larger Frozen 8x8 environment. Notice again the large variability is the average rewards across different combinations of hyperparameters. Interestingly, the initial transient behavior is now different between the two ϵ schedules in Figure 11(bottom), compared with Figure 6, with now both schedules showing an initial transient and plateauing around 10,000 iterations. Since this is a larger problem, the Q-learning agent has taken more episodes to converge, compared to the smaller Frozen 4x4 environment.

Figure 12 shows the run time across planning and reinforcement learning for Frozen Lake 4x4 and Frozen Lake 8x8. Interestingly, Value Iteration runtime increased by 7.8 times, Policy Iteration by 44.2 times, while Q-learning running time increased only by a factor of 2.0. Both panning algorithms are more directly impacted by the number of states, as they have to loop over a larger set of states to perform the updates, while Q-Learning is only affected by the extent to which each episode takes longer to finish.

Finally, Figure 13 and 14 show the corresponding value functions and policies as obtained via Policy Iteration and Q-Learning for Frozen 8x8. Interestingly, we see this time larger deviations of Q-learning solutions with respect to the benchmark found by Policy iteration, with the resulting policies showing now differences across the algorithms, in contrast with Frozen 4x4. Given that this is a larger environment, the algorithms are not only converging to different solutions for the value iteration, but also for the policy. Therefore, the relative values between the value function are incorrect, as found by the Q-learners.

For the Frozen Lake 8x8 environment, if we rank all the different hyperpameter combinations in terms of the largest average rewards over the last 100 episodes, the best hyperparameters were as follows:

- Constant scale ramp-down:

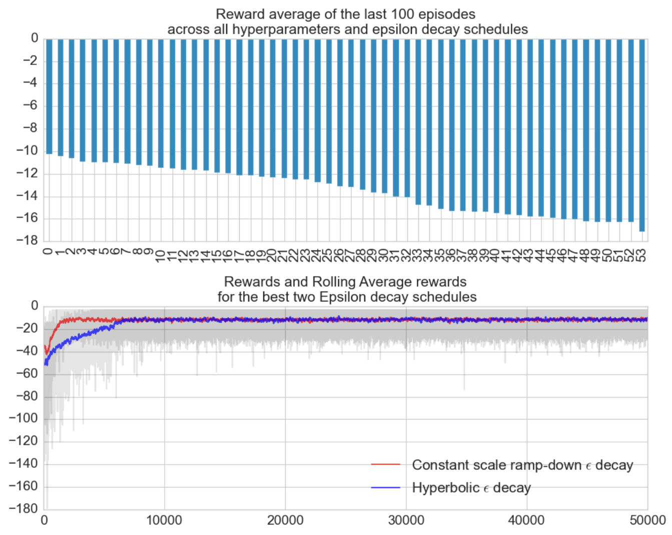


Fig. 11: Q-Learning for Frozen Lake 8x8. Top: average reward during the last 100 iterations for Q-learning across all for the 48 combination of hyperparameters and ϵ schedules. Bottom: evolution of the rewards and the rolling average reward over a window of 100 episodes for the best performing combination of hyperparameters for both Constant scale ramp-down and Hyperbolic.

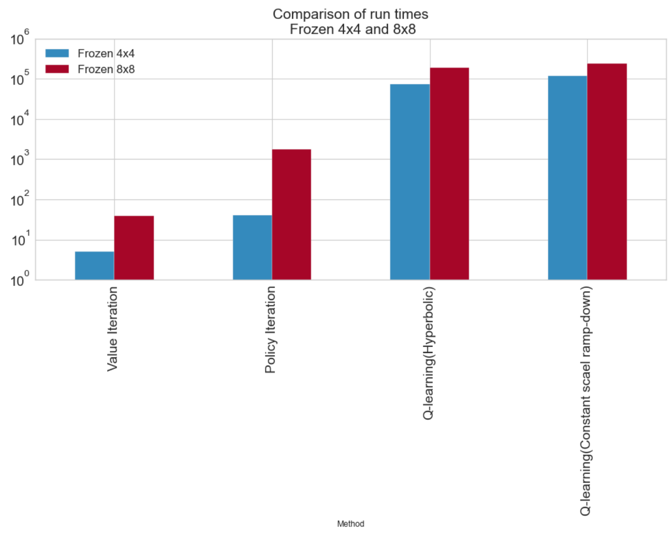


Fig. 12: Run times for Frozen Lake 4x4 and 8x8 across value iteration, policy iteration and Q-learning, for two different ϵ schedules.

  - $\alpha = 0.05$
  - $\gamma = 0.99$
  - $\delta\epsilon = 0.99$
  - $\epsilon_{min} = 0.0$
- Hyperbolic:
  - $\alpha = 0.01$
  - $\gamma = 0.99$

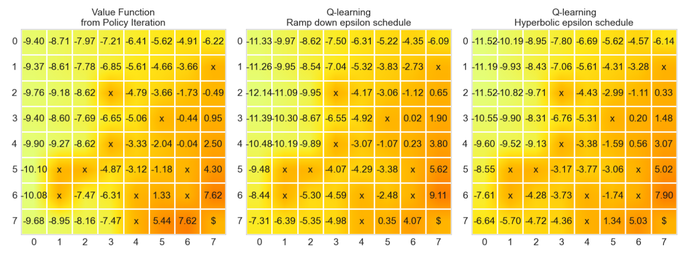


Fig. 13: Frozen Lake 8x8. Value function comparison from policy iteration and Q-learning, for two different ϵ schedules.

  - $\delta\epsilon = 0.9$
  - $\epsilon_{min} = 0.05$

These parameters are somewhat different from Frozen 4x4. However, for both environments cases, the hyperbolic decay does seem to require the same non-zero minimum exploration rate $\epsilon_{min} = 0.05$.

If we take the value function from Policy Iteration as the benchmark (given that it has access to the environment model, while Q-learning does not and tries to estimate it), the average of the absolute differences across all cells in 1.06 for the Q-learner using the ramp down schedule, while the error decreased to 0.98 for the Q-learning using the hyperbolic schedule. Given this criteria, and contrary to what we found in Frozen 4x4, the hyperbolic ϵ schedule was the best configuration this time for the larger environment.

C. Stock trading

We have implemented this environment by sub-classing the `gym` environment. To test the implementation, we compared successfully the price of a call option using the Black-Scholes closed-form solution (see [?]) against the value function of a Markov chain where, regardless of the action taken by the agent, the reward is only given at the termination of the episode at time $t = T$, and equal to the call option payoff $max(S_T - K, 0)$. The value of the option at time $t = 0$ is compared in Figure 15. Since Value Iteration agrees with the closed-form solution provided by Black-Scholes mdodel (see [?]), we conclude the implementation of the stock environment is correct.

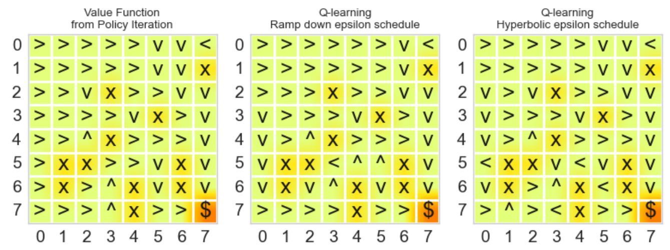


Fig. 14: Frozen Lake 8x8. Policy comparison from policy iteration and Q-learning, for two different ϵ schedules.

We further modified the environment so that the stock price displays *mean-reversion* around S_0: basically, when prices are either higher or lower than S_0, the process will tend to revert back to S_0. This creates a statistical arbitrage opportunity, whereby the agent could profit by buying the stock when prices are low, and sell when prices are high, knowing that the process is mean reverting. Mean reversion was implemented by modifying the up and down probabilities, and making them depend on the moving average of the stock in relation to S_0. Across all our numerical experiments we have set $S0 = 10, T = 1, r = 2\%, \sigma = 0.2$, and the rewards are either $-S_t$ when the agent decides to buy the stock at time t, or $+S_t$ when selling at time t. Therefore, maximizing the discounted expected sum of future rewards means making a profit by trading on the underlying stock.

Figure 16 shows the agreement between value iteration and policy iteration across the different states for the stock environment in time and space represented as a binary tree (time in the horizontal axis, possible steps in the stock in the vertical axis, asn the initial state $S_0 = S0$ is at the center of the vertical axis), for those states where the agent has 1 stock. As we have found before for the Frozen Lake environment, both algorithms converge to the same answer. These results are intuitive, as the value of holding one stock should be higher as the stock moves up, and viceversa when the stock moves down.

Figure 18 shows the behavior of Q-learning for the stock environment for N = 40, which has 1,640 states. The top plot shows the variability of the results across various combinations of hyperparameters, sorted by the magnitude of the rolling

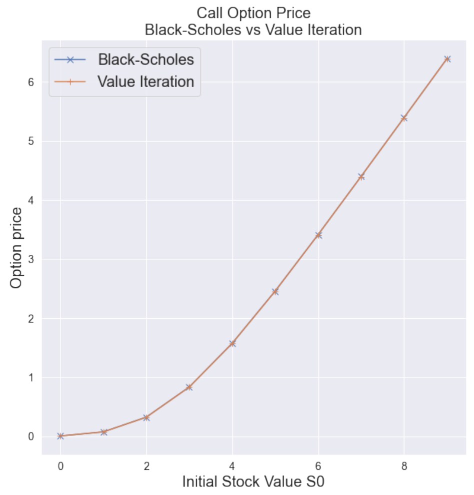
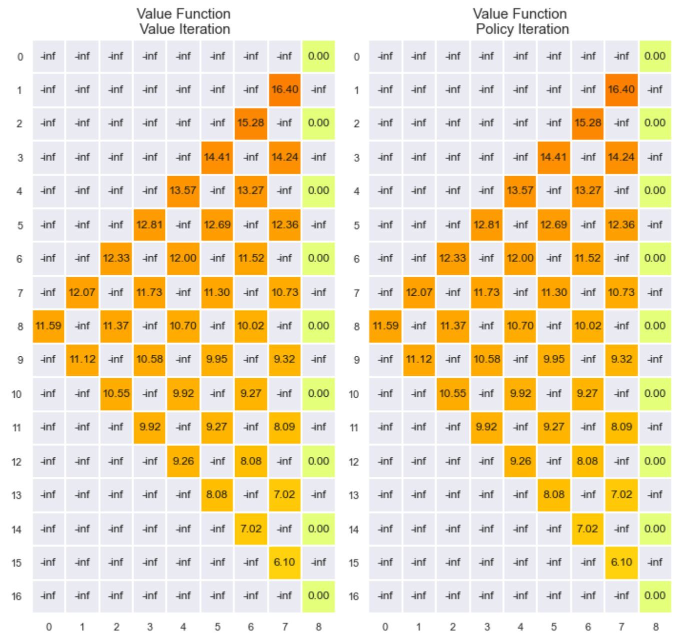


Fig. 15: Checking our implementation of the stock environment: Value Iteration on a call option matches Balck-Scholes formula, indicating the implementation fo the stock trading environment is correct.

reward over the last 100 episodes. This again highlights the sensitivity of Q-learning to the choice of hyperparameters and ϵ schedules. The bottom plot shows the evolution of the rewards and the rolling average reward over a window of 100 episodes for the best performing combination of hyperparameters for both Constant scale ramp-down and Hyperbolic. We clearly see a much better performance for Constant scale ramp-down, compared to hyperbolic decay for ϵ. Aslo, it is clear from the plot that the first ϵ schedule correctly exploites the mean reversion of the stock dynamics and exploits the statistical arbitrage quite successfully, while the second one never actuall buys the stock, holding all the time, and therefore, does not loose money, nor gain any either. Please note that for making the plot shading easier to read we have made all other cell values to equal $-\infty$, so that they appeared dark gray in color. However, those cells are not part of the state space, and therefore, are not part of the $Q-learning$ algorithm (I could not just get rid of them in the plot, so I initialized them to $-\infty$).

We also notice the drastic reduction in standard deviation of the rewards compared with Frozen Lake. The stock environment is stochastic because the transitions from the stock are random. However, the action of buying or selling a stock gets executed 100% of the time should the agent decide (which is what happens in reality). However, the Frozen environment has randomness in the actions the agent decides to take, and therefore, should an agent decide to move, the movement gets executed only 80% of the time successfully. This different nature of the stochastic behavior results in very different profiles for the rewards.

Figure 18 shows a comparison between the Value function from value iteration, versus Q-Learning using the best combi-


Value Function
Value Iteration

	0	1	2	3	4	5	6	7	8
0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	0.00
1	-inf	-inf	-inf	-inf	-inf	-inf	-inf	16.40	-inf
2	-inf	-inf	-inf	-inf	-inf	-inf	15.28	-inf	0.00
3	-inf	-inf	-inf	-inf	-inf	14.41	-inf	14.24	-inf
4	-inf	-inf	-inf	-inf	13.57	-inf	13.27	-inf	0.00
5	-inf	-inf	-inf	12.81	-inf	12.69	-inf	12.36	-inf
6	-inf	-inf	12.33	-inf	12.00	-inf	11.52	-inf	0.00
7	-inf	12.07	-inf	11.73	-inf	11.30	-inf	10.73	-inf
8	11.59	-inf	11.37	-inf	10.70	-inf	10.02	-inf	0.00
9	-inf	11.12	-inf	10.58	-inf	9.95	-inf	9.32	-inf
10	-inf	-inf	10.55	-inf	9.92	-inf	9.27	-inf	0.00
11	-inf	-inf	-inf	9.92	-inf	9.27	-inf	8.09	-inf
12	-inf	-inf	-inf	-inf	9.26	-inf	8.08	-inf	0.00
13	-inf	-inf	-inf	-inf	-inf	8.08	-inf	7.02	-inf
14	-inf	-inf	-inf	-inf	-inf	-inf	7.02	-inf	0.00
15	-inf	-inf	-inf	-inf	-inf	-inf	-inf	6.10	-inf
16	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	0.00

Value Function
Policy Iteration

	0	1	2	3	4	5	6	7	8
0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	0.00
1	-inf	-inf	-inf	-inf	-inf	-inf	-inf	16.40	-inf
2	-inf	-inf	-inf	-inf	-inf	-inf	15.28	-inf	0.00
3	-inf	-inf	-inf	-inf	-inf	14.41	-inf	14.24	-inf
4	-inf	-inf	-inf	-inf	13.57	-inf	13.27	-inf	0.00
5	-inf	-inf	-inf	12.81	-inf	12.69	-inf	12.36	-inf
6	-inf	-inf	12.33	-inf	12.00	-inf	11.52	-inf	0.00
7	-inf	12.07	-inf	11.73	-inf	11.30	-inf	10.73	-inf
8	11.59	-inf	11.37	-inf	10.70	-inf	10.02	-inf	0.00
9	-inf	11.12	-inf	10.58	-inf	9.95	-inf	9.32	-inf
10	-inf	-inf	10.55	-inf	9.92	-inf	9.27	-inf	0.00
11	-inf	-inf	-inf	9.92	-inf	9.27	-inf	8.09	-inf
12	-inf	-inf	-inf	-inf	9.26	-inf	8.08	-inf	0.00
13	-inf	-inf	-inf	-inf	-inf	8.08	-inf	7.02	-inf
14	-inf	-inf	-inf	-inf	-inf	-inf	7.02	-inf	0.00
15	-inf	-inf	-inf	-inf	-inf	-inf	-inf	6.10	-inf
16	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	0.00

Fig. 16: Value iteration and policy iteration converge to the same solution for the stock environment. $N = 8$, which has 90 states.

nation of hyperparameters and the constant ramp down, for N = 10 (which has 132 states). As can be seen, Q-learning Value function across the states differ from the more exact value provided by Value Iteration. Moreover, Q-learning remains at zero value to several states (Q-learning matrix was initialized to zero across all states), probably from the fact that those states where never visited during the episodes due to the mean-reverting nature of the stock, as only the cells around $S_0 = 10$ get assigned non-zero values. Cells far apart from $S_0 = 10$ have zero chance of getting visited due to the mean-reversion effect. There is also a slightly shift upwards in the states that are non-zero, due to the upward drift provided by the non-zero risk free rate $r = 0.02$.

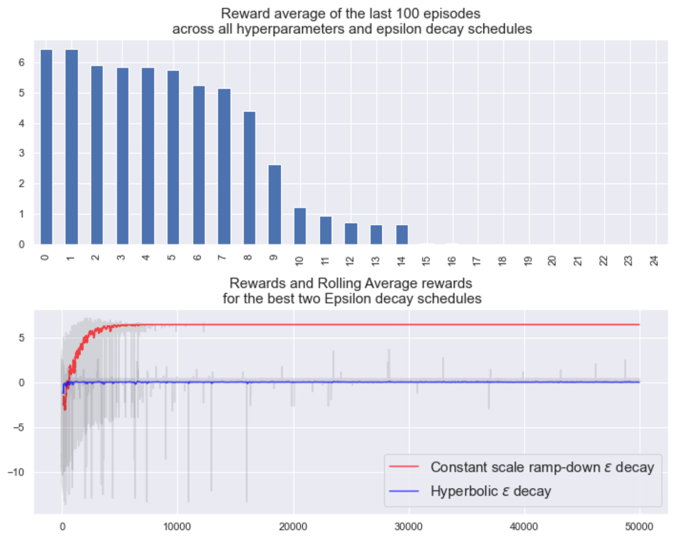


Fig. 17: Q-Learning for the stock trading environment for N = 40, which has 1,722 states. Top: average reward during the last 100 iterations for Q-learning across all for the 48 combination of hyperparameters and ϵ schedules. Bottom: evolution of the rewards and the rolling average reward over a window of 100 episodes for the best performing combination of hyperparameters for both Constant scale ramp-down and Hyperbolic.

Figure 8 shows a comparison of the running times between Value iteration, Policy Iteration and Q-learning for the stock trading environment, as a function of state size. Across all state sizes Value Iteration takes much less time to converge than policy Iteration. As we mentioned before, value iteration takes less time to convergence because it is a *greedy* with respect to the actions in the one stop look-ahead, and instead of evolving with respect to a policy, it takes the best action across all actions possible. Q-learning takes the most to run, as it does not have access to the full model as policy and value iteration do.

III. Conclusions

We have solved two Markov Decision Processes, Frozen Lake and a stock trading environment, using Value Iteration and Policy Iteration (planing), and Q-learning (reinforcement-learning) with different schedules for α and ϵ, and analyzed and contrasted their behavior as a function of the hyperparameters and problem sizes. Value iteration and policy iteration have access to the model of the environment, given by the transition probabilities and rewards function, and therefore, converge much faster than Q-learning, which does not have

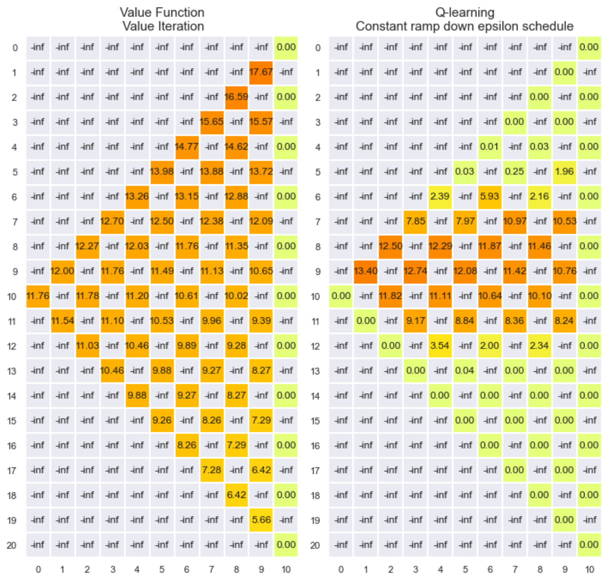


Fig. 18: Comparison between the Value function from value iteration, versus Q-Learning using the best combination of hyperparameters and the constant ramp down, for N = 10. As in the case of Frozen lake, the approximate nature of the Q-Larning is manifested by its different value function for the state, compared to Value Iteration.

access to the a model of the environment, and only learns from experiences (s, a, r, s').

Among the planning methods, Value iteration tends to converge faster, and take fewer iterations to converge, than policy iteration. This is because value iteration is *greedy* with respect to the actions in the one stop look-ahead, and instead of evolving with respect to a policy, it takes the best action across all actions possible. Policy iteration instead tries to

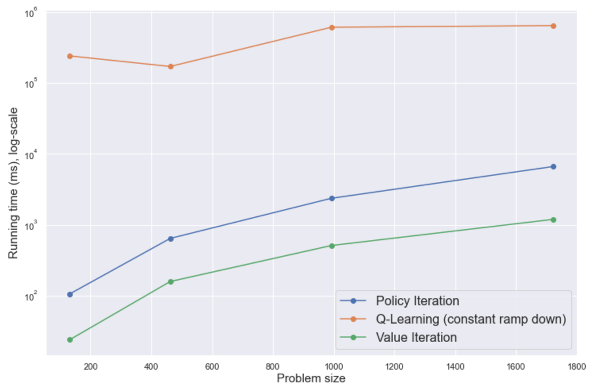


Fig. 19: Comparison of the running times between Value iteration, Policy Iteration and Q-learning for the stock trading environment, as a function of state size.

learn both the value function and the policy at the same time, and in particular, the value function is estimated from an approximate policy solution. This is the difference between $V(s) \leftarrow \sum_{s',r} p(s', r, |s, \boldsymbol{\pi(s)})[r + \gamma V(s')]$ for policy iteration, versus $V(s) \leftarrow \mathbf{max}_{\boldsymbol{a}} \sum_{s',r} p(s', r, |s, a)[r + \gamma V(s')]$ for value iteration. However, both Value Iteartion and Policy Iteration converged to the same answer, as they both are algorithms to solve the Bellman Equation for the same model of the environment.

For the smaller Frozen 4x4 environment, even though Q-learning did not arrive at the same value function as Value Iteration or Policy Iteration, the *relative* values were still correct, producing the same policy than the planning methods. However, for the larger environments, Frozen 8x8 or the stock environment, Q-learning did not produce the same policies.

Q-learning is a reinforcement learning algorithm that does not rely on a model for the environment, and instead learns from experiences (s, a, r, s'), therefore, taking much longer to converge and only arriving at approximate solutions. We analyzed two exploration strategies for ϵ greedy algorithm, as well as a wide array of hyperparameters, and found that Q-learning is very sensitive to a) the choice of hyperparameters (α, γ, etc) b) exploration strategy choosing different ϵ greedy algorithm c) it displays wide variability of rewards per episode and d) we found cases where it tends to "forget" the values, as we saw sudden slumps in the performance, as measured by rolling averages of rewards across episodes, specially when α decreased with episode t.

References

[1] Custom frozen lake gym implementation. https://github.com/wesley-smith/CS7641-assignment-4.git.
[2] Openai gym. https://gym.openai.com/.
[3] V. P. Leif B.G. Andersen. *Interest Rate Modelling*. 2010.
[4] R. Sutton and A. Barto. *Reinforcement Learning*. 2000.