CMPUT 655
Assignment 3

September 15, 2024

Dikshant
1877098

---

1. *Write the iterative policy evaluation update*
   (a) *Discuss in one sentence why the update is "expected" (hint: consider the backup diagram we discussed in-class).*
   (b) *Assume you have one sample per transition (i.e., tuple). Can you still do the expected update? Explain your answer in one sentence.*
   (c) *What assumptions do we need to guarantee its convergence? (hint: in-class we discussed that Puterman's "Markov Decision Processes: Discrete Stochastic Dynamic Programming" says that states and actions must be ..., rewards must be ..., transition and reward functions must be ..., discount factor must be ...).*
   (d) *Policy evaluation can be solved in a way that is not iterative. Describe it in one sentence. (hint: check the Sutton's book, Chapter 4.1).*
   (e) *What is one advantage of the iterative method? What is one disadvantage? (hint: consider time and space complexity)*

Ans: The iterative policy evaluation update is defined as:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right]$$

(a) The iterative policy evaluation update is "expected" because it averages over all possible next states based on the policy's probability distribution, rather than on a sample next state.

(b) If we have only one sample per transition, we cannot compute the expected update exactly because the sample might not represent the true average over all possible transitions.

(c) To guarantee convergence, the state and action spaces must be finite, rewards must be bounded, the transition and reward functions must be stationary (i.e., they don't change over time), and the discount factor must satisfy $0 \leq \gamma < 1$.

(d) Policy evaluation can be solved non-iteratively by solving the bellman equation as a system of linear equations, where the value function is computed directly. We can get the value function directly by using the following update rule: $\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$

(e) The iterative method is more memory efficient and scalable, as it doesn't require storing or solving a large system of equations, making it feasible for large state spaces. A disadvantage is that it can be computationally expensive and slow to converge, especially when the discount factor γ is close to 1, requiring many iterations to achieve accurate results.

2. *What are the equations analogous to (4.3), (4.4), and (4.5), but for action-value functions instead of state-value functions?*

Ans:

$$Q_\pi(s,a) = \mathbb{E}_\pi\{r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\}$$

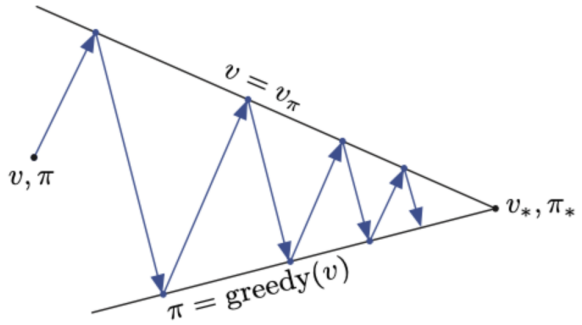
$$Q_\pi(s,a) = \sum_{s',a} p\left(s', r | s, a\right) \left[r + \gamma \sum_{a'} \pi\left(a' | s'\right) Q_\pi(s', a') \right]$$

$$q_{k+1}(s,a) = \sum_{s',a} p\left(s', r | s, a\right) \left[r + \gamma \sum_{a'} \pi\left(a' | s'\right) q_k(s', a') \right]$$

3. *Generalized Policy Iteration*

   (a) *In two sentences, explain what GPI is. (hint: think about "early stopping" of the policy evaluation and policy improvement steps)*

   (b) *The diagram below shows the policy iteration algorithm. Draw a similar diagram conveying GPI. (hint: we did it in-class)*



Ans:

(a) Generalized Policy Iteration (GPI) alternates between policy evaluation and policy improvement, stopping the evaluation before full convergence. This process allows for faster convergence by continuously updating both the value function and the policy.

(b) Diagram for GPI:

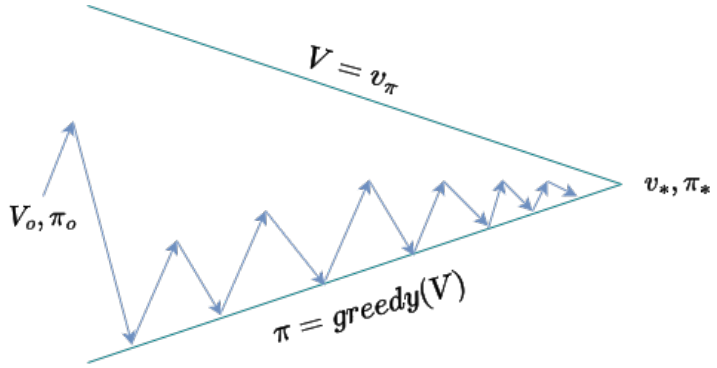


Figure 1: Generalized Policy Iteration

4. *(a)* *Part 1*

- *Implement policy iteration (PI), value iteration (VI), and generalized policy iteration (GPI) as in the book (i.e., learn $V_\pi(s)$). For GPI, have the algorithm alternate between 5 policy evaluation steps and one policy improvement. That is, don't wait until convergence of policy evaluation to do policy improvement.*
- *Learn the optimal policy when $\gamma = 0.99$. For PI, start from a random policy (i.e., all actions have the same probability). To be sure it worked, manually write the optimal policy yourself (you should have done it for Assignment 2) and check it with assert* `np.allclose(`$\pi_{learned}, \pi_{opt}$`)` *after every algorithm run.*
  - *For example, say in one run PI does 10 policy evaluation iterations, 1 policy improvement, 5 policy evaluation iterations, 1 policy improvement, then stops (converges). The total number of iterations is 15 and you should have logged all 15 Bellman errors.*
- *Repeat the learning with 7 different initializations of $V_o(s)$ (all values initialized to: -100, -10, -5, 0, 5, 10, 100). At every policy evaluation step, log the total absolute Bellman error (see Assignment 2). Also log the total number of policy evaluations needed.*
  - *Report the total number of policy evaluations in a table with mean and standard deviation (rows: algorithm; column: mean std).*

        - *Which algorithm learn the optimal policy faster (i.e., fewer policy evaluations) on average according to your table? PI, VI, GPI?*
    - *Plot the Bellman error trend (x-axis: policy evaluation iteration; y-axis: Bellman error). For each of the 3 algorithms you have to show 7 plots (one for each initialization).*
        - *Comment on the trend of the BE. Does it decrease steadily or does it suddenly increase sometime? If there is any spike in the BE trend, explain why that's the case.*
    - *Pay attention to GPI: is it better or worse than PI and VI in terms of policy iterations needed to converge?*
        - *Maybe it is neither better nor worse. Or maybe it's better with some initializations but worse with others. Discuss if that's the case.*
        - *Is this gridworld the right example to make the best use of the benefits of GPI? If not, what would be a better setting? (hint: think about the policy improvement and the max operator)*

(b) *Part 2*

Now implement the same 3 algorithms, but this time learn (you'll need the equations we ask for in Exercise 4.3). Hint: you can just copy-paste your code from Part 1 and replace with in value function updates.

- *Like in Part 1, report the mean std for the total iterations of policy evaluation and plot the Bellman error trend.*
    - *Is the overall efficiency (i.e., number of policy evaluations) better or worse than Part 1?*
- *Compare the new plots and table to the ones from Part 1 and discuss differences/similarities.*
    - *Is the trend across initializations the same as in Part 1?*

Draw your conclusion:

- *Consider the sensitivity to initialization.*
    - *Which algorithm is preferrable (PI, VI, or GPI) in terms of number of policy evaluations?*
- *What do the results tell us about learning vs ?*
    - *Is there an algorithm that works better when learning or ?*
- *Does initialization matter?*
    - *Are there initializations (among -100, -10, -5, 0, 5, 10, 100) that always take more iterations to learn, regardless of the algorithm and which value function we learn?*
    - *Is one algorithm (PI, VI, GPI) more sensitive to the initialization than the others?*

- *What about value function, i.e., is more sensitive than ? Justify your answers by looking at the trend of the BE, and saying if with some initialization the plot is very different from the plots of other initializations.*

Submit your code and plots.

Ans: We will examine Policy Iteration (PI), Value Iteration (VI), and Generalized Policy Iteration (GPI) with $\gamma = 0.99$. We will analyze their performance with various initializations of the value function and track the total Bellman error and number of policy evaluations steps required for convergence.

PART 1

(a) **Policy comparison**

Table 1 summarizes the total number of policy evaluation steps required for each algorithm—Policy Iteration (PI), Value Iteration (VI), and Generalized Policy Iteration (GPI)—across different initializations of the value function. The results are tabulated (Table 2) to show the mean and standard deviation of policy evaluations steps for each algorithm. In Figure 2, we have the data from Table 2 plotted to compare the number of policy evaluation steps for each algorithm.

Initial Value (V_0)	**PI**	**VI**	**GPI**
-100	3376	8	25
-10	3464	8	35
-5	3467	8	40
0	3470	8	30
5	3472	168	70
10	3474	237	225
100	3500	466	480

Table 1: Steps to Bellman Error Convergence for PI, VI, and GPI

Algorithm	**Mean**	**Standard Deviation**
Policy Iteration	3460.43	36.17
Value Iteration	129.0	162.71
Generalized Policy Iteration	126.43	153.29

Table 2: Total number of policy evaluations

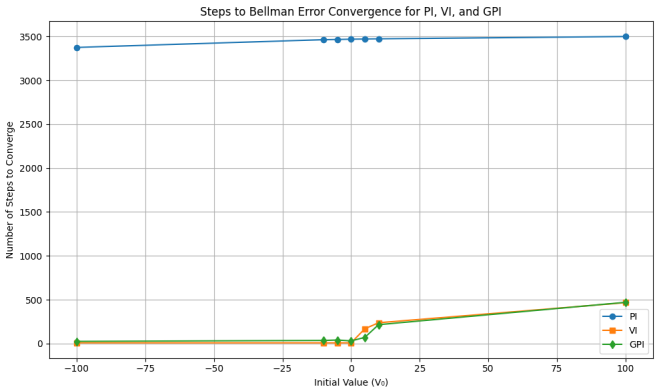
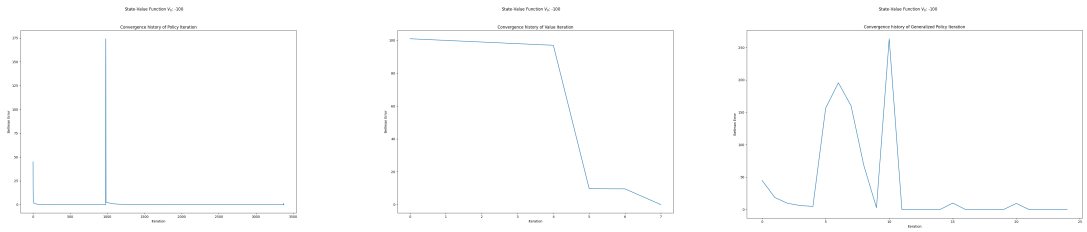


Figure 2: Policy Comparison

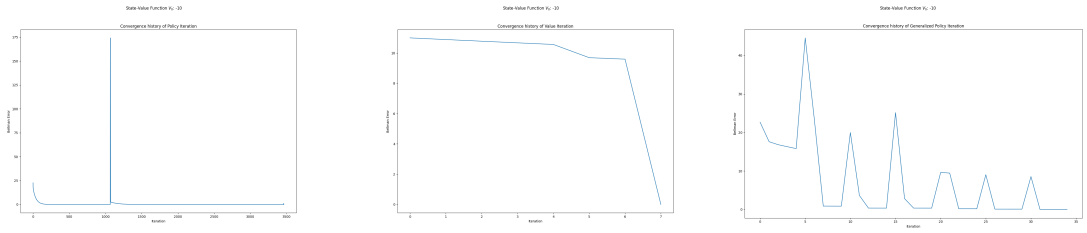
Algorithm that learn the optimal policy faster: According to the table, GPI requires the fewest policy evaluation steps compared to VI and PI. However, while GPI generally performs significantly better than PI, the results show that GPI's average performance is quite similar to VI and with a lower initial value, VI can outperform GPI.

(b) **Bellman Error Convergence Plots for Different Initializations of V_o**



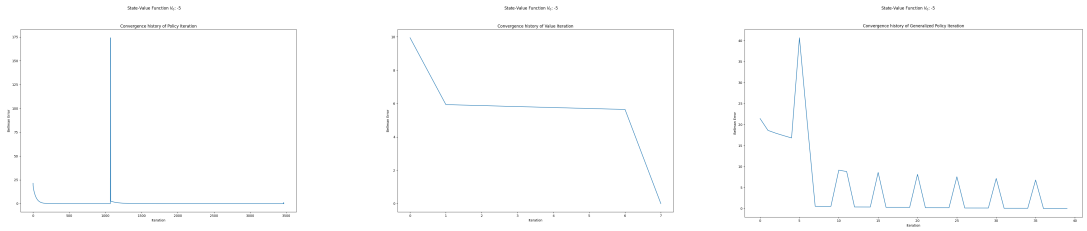
(a) PI, $V_0 = -100$ (b) VI, $V_0 = -100$ (c) GPI, $V_0 = -100$

Figure 3: Bellman Error Convergence for $V_0 = -100$



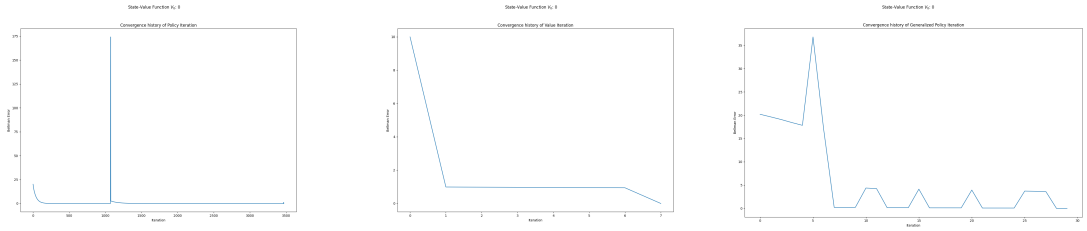
(a) PI, $V_0 = -10$ (b) VI, $V_0 = -10$ (c) GPI, $V_0 = -10$

Figure 4: Bellman Error Convergence for $V_0 = -10$



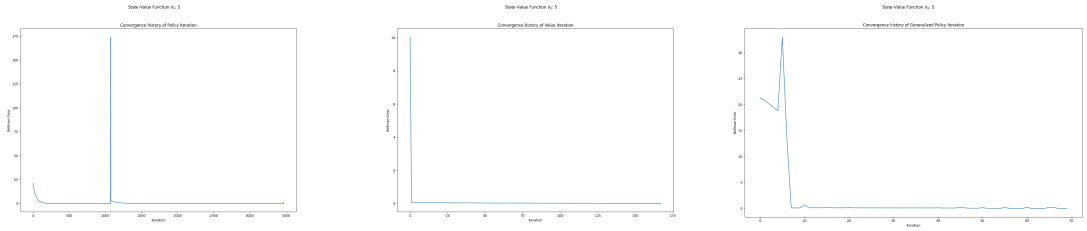
(a) PI, $V_0 = -5$ (b) VI, $V_0 = -5$ (c) GPI, $V_0 = -5$

Figure 5: Bellman Error Convergence for $V_0 = -5$



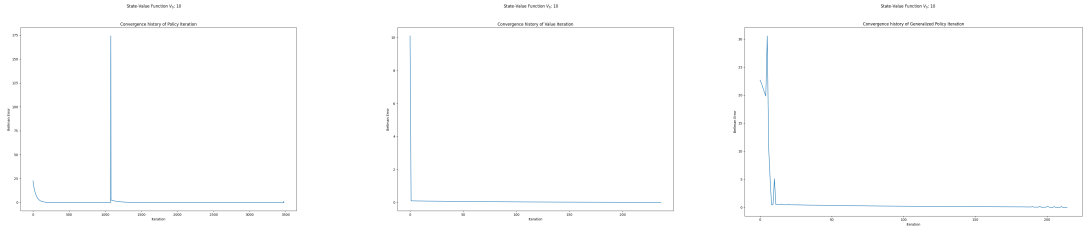
(a) PI, $V_0 = 0$ (b) VI, $V_0 = 0$ (c) GPI, $V_0 = 0$

Figure 6: Bellman Error Convergence for $V_0 = 0$



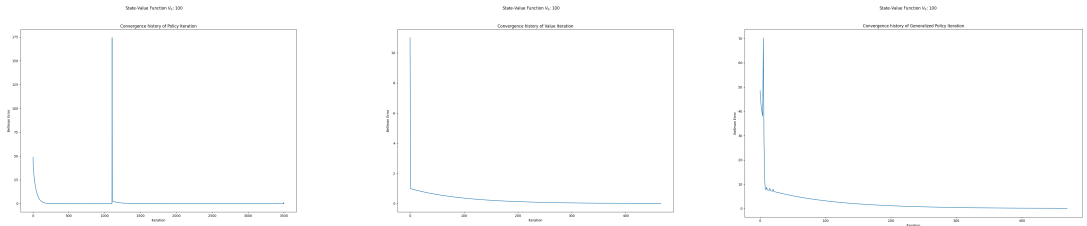
(a) PI, $V_0 = 5$ (b) VI, $V_0 = 5$ (c) GPI, $V_0 = 5$

Figure 7: Bellman Error Convergence for $V_0 = 5$



(a) PI, $V_0 = 10$ (b) VI, $V_0 = 10$ (c) GPI, $V_0 = 10$

Figure 8: Bellman Error Convergence for $V_0 = 10$



(a) PI, $V_0 = 100$ (b) VI, $V_0 = 100$ (c) GPI, $V_0 = 100$

Figure 9: Bellman Error Convergence for $V_0 = 100$

Trend of Bellman Error:
In Policy Iteration, starting with a random policy, the Bellman error decreases steadily during the policy evaluation phase as the algorithm works to minimize the error. However, after the first policy improvement step, the newly updated greedy policy significantly alters the value function estimates, resulting in a large spike in the Bellman error. This spike occurs because the new policy leads to values that deviate significantly from the previous ones. After this initial jump, subsequent updates exhibit smaller spikes as the policy continues to improve and the value function stabilizes.

For Value Iteration, the error remains more controlled, with no significant spikes, as policy improvement occurs after each policy evaluation step. This approach gradually moves the policy towards optimality without drastic changes. After a few iterations, the process converges to a stable policy, maintaining smooth Bellman error trends throughout.

In Generalized Policy Iteration, there is an initial large spike due to the starting random policy, similar to Policy Iteration. However, in GPI, the policy is evaluated over multiple steps, leading to periodic smaller spikes after each policy improvement. These smaller spikes arise because the policy is refined in stages, bringing the value estimates closer to the optimal values. The overall magnitude of spikes is generally smaller compared to Policy Iteration since GPI doesn't fully minimize the Bellman error before each policy improvement step, resulting in faster convergence with fewer drastic changes.

(c) **GPI Comparison with VI and PI**
GPI performs significantly better than PI, as seen in the plots where it converges to a good policy with fewer iterations. Compared to VI, the performance of GPI depends on the initialization. For lower initial values, VI converges faster, but as the initial values increase to larger positive numbers, GPI starts to outperform VI.

Is GPI better than VI/PI: This difference arises because GPI balances policy evaluation and improvement more effectively, especially with higher initial values, where VI's one-step evaluations may take longer to stabilize. GPI's approach of performing multiple evaluations before each policy improvement helps it adapt faster in such cases.

Is gridworld the right example to make the best use of the benefits of GPI?: The current gridworld example may not fully showcase the benefits of GPI, as it is relatively simple with straightforward state transitions and rewards. Also, if number of policy evaluation steps are low, then it may try a suboptimal action as well. In our case, when using only 5 evaluations, selecting the optimal action with np.argmax can cause issues. Specifically, if two or more actions have the same value, the agent may repeatedly choose a suboptimal action, preventing the policy from converging. To address this, we can modify the approach by randomly selecting one of the best actions instead of relying on np.argmax. While this adjustment allows convergence, it introduces the risk of incorporating suboptimal values into the policy. This occurs because the policy improvement step does not sufficiently differentiate between what is truly good and bad for the agent.

What can be a better setting: A better setting for GPI would involve larger state spaces, stochastic environments, or tasks with long-term dependencies, where GPI's periodic policy evaluation and improvement steps offer advantages. In such environments, GPI can better handle complexity and uncertainty, allowing it to converge more effectively compared to PI or VI, especially when more accurate value estimates are crucial before making policy improvements.

PART 2

(d) **Policy comparison**

Table 3 summarizes the total number of policy evaluation steps required for each algorithm—Policy Iteration (PI), Value Iteration (VI), and Generalized Policy Iteration (GPI)—across different initializations of the value function. The results are tabulated (Table 4) to show the mean and standard deviation of policy evaluations steps for each algorithm.

Initial Value (Q_0)	PI	VI	GPI
-100	3603	9	25
-10	3691	9	30
-5	3694	9	35
0	3696	9	25
5	3698	169	135
10	3699	238	240
100	3725	467	495

Table 3: Steps to Bellman Error Convergence for PI, VI, and GPI

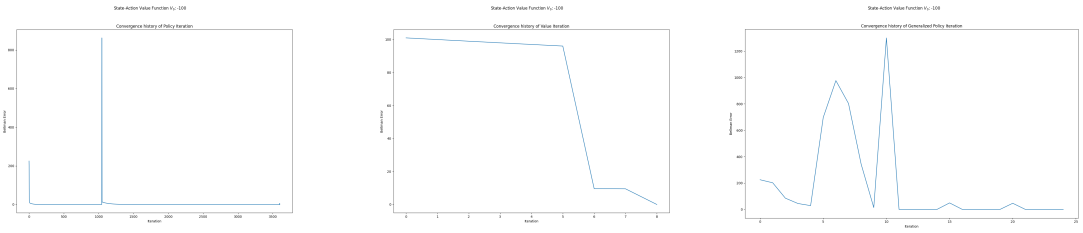
Algorithm	Mean	Standard Deviation
Policy Iteration	3686.57	35.68
Value Iteration	130.0	162.71
Generalized Policy Iteration	140.71	162.84

Table 4: Total number of policy evaluations

Overall efficiency: Number of evaluation steps for each algorithm increased slightly compared to part 1 and hence slighty worser for Q_o initializations.

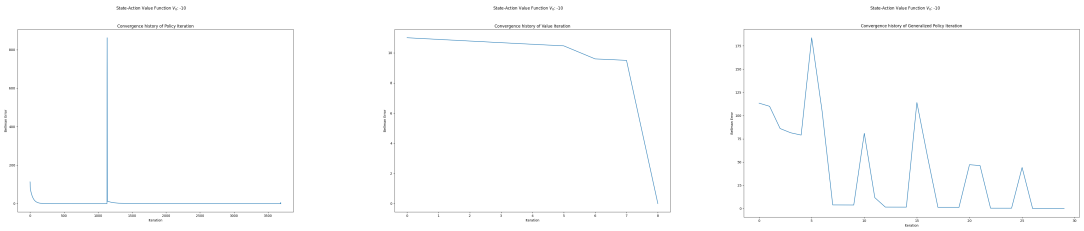
Trend: The trend across initializations is largely similar to what was observed in Part 1, but with more iterations required in the new plots. The overall pattern of convergence remains the same, with the algorithms gradually improving the policy and reducing the Bellman error.

(e) **Bellman Error Convergence Plots for Different Initializations of Q_o**



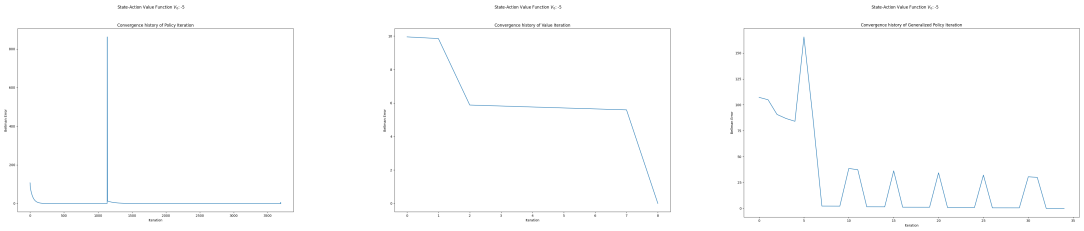
(a) PI, $Q_0 = -100$ (b) VI, $Q_0 = -100$ (c) GPI, $Q_0 = -100$

Figure 10: Bellman Error Convergence for $Q_0 = -100$



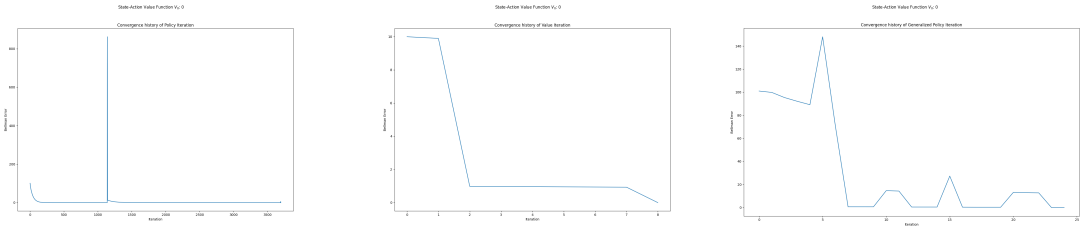
(a) PI, $Q_0 = -10$ (b) VI, $Q_0 = -10$ (c) GPI, $Q_0 = -10$

Figure 11: Bellman Error Convergence for $Q_0 = -10$



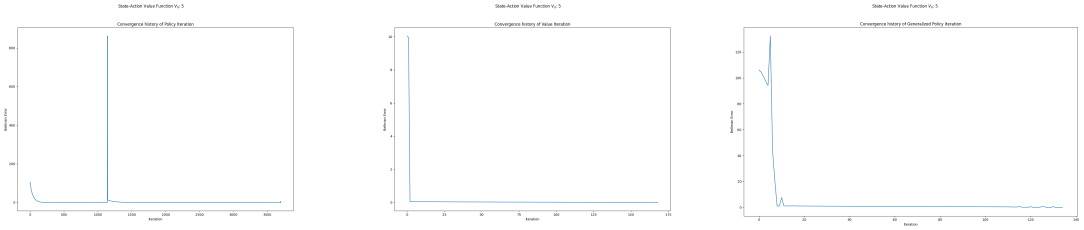
(a) PI, $Q_0 = -5$ (b) VI, $Q_0 = -5$ (c) GPI, $Q_0 = -5$

Figure 12: Bellman Error Convergence for $Q_0 = -5$



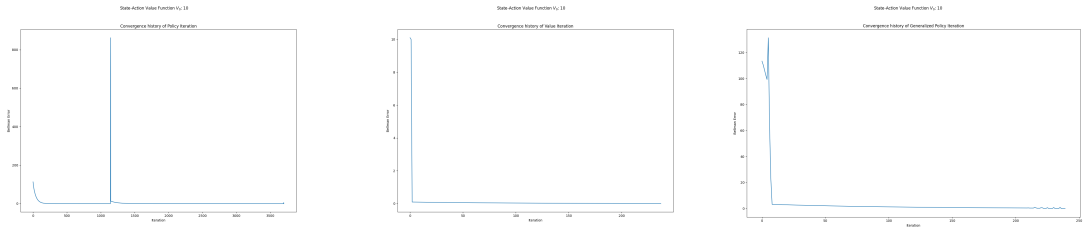
(a) PI, $Q_0 = 0$ (b) VI, $Q_0 = 0$ (c) GPI, $Q_0 = 0$

Figure 13: Bellman Error Convergence for $Q_0 = 0$



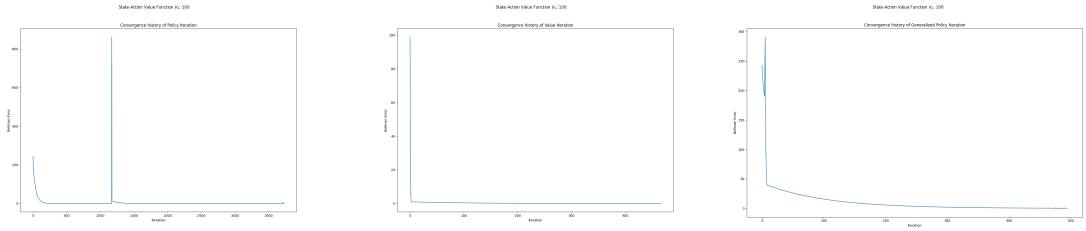
(a) PI, $Q_0 = 5$ (b) VI, $Q_0 = 5$ (c) GPI, $Q_0 = 5$

Figure 14: Bellman Error Convergence for $Q_0 = 5$



(a) PI, $Q_0 = 10$ (b) VI, $Q_0 = 10$ (c) GPI, $Q_0 = 10$

Figure 15: Bellman Error Convergence for $Q_0 = 10$



(a) PI, $Q_0 = 100$ (b) VI, $Q_0 = 100$ (c) GPI, $Q_0 = 100$

Figure 16: Bellman Error Convergence for $Q_0 = 100$

(f) **Conclusion**

Preferred Algorithm for Number of Policy Evaluations: GPI and VI tends to require fewer number of policy evaluations steps. PI requires more policy evaluations because it performs a full policy evaluation before each improvement. VI performs well with lower initializations but can take longer to converge with higher initializations. In this case, GPI is not able to outperform VI but if we have more stochasticity then GPI may outperform VI and thus GPI is more suitable.

Learning V vs Q: The results suggest that learning Q is more sensitive to initialization compared to learning V. This is because Q values involve evaluating state-action pairs, making them more prone to fluctuations during early stages of learning, especially with extreme initial values. Hence, the initial bellman error is higher in case of very high or very low value Q_o. Learning V tends to be smoother because it aggregates over all actions.

Impact of Initialization: Initialization does matter and cause both V and Q to take more iterations to converge. These extreme initializations cause large Bellman errors early on, requiring more iterations to correct the values.

Initializations that Always Take Longer Big positive initializations 10/100, consistently require more iterations to converge. This is due to the large initial gap between the initialized values and the true optimal values, leading to larger errors that take longer to correct.

Algorithm Sensitivity to Initialization: Among the three algorithms, VI appears to be the most sensitive to initialization. With lower initial values, VI performs well and converges quickly, but as the initialization increases, it takes longer to stabilize. PI is less sensitive than VI but still more prone to spikes with larger initializations compared to GPI.

Sensitivity of V vs Q: Learning Q tends to be more sensitive to initialization than learning V. This is reflected in the Bellman error trends: with extreme initializations, the Bellman error for Q tends to exhibit larger spikes in the starting compared to Q. This difference arises because learning Q requires evaluating both states and actions, making it more prone to fluctuations, while learning V aggregates across all actions, providing more stability.

Code can be accessed on github from here: Policy Comparison