Assignment-1

Kunal Kumar Sahoo (2025AIZ8459)
AIL7022

September 4, 2025

1 Policy Iteration and Value Iteration

1.1 Stationary Environment Analysis

1. Policy Iteration (PI) and Value Iteration (VI) algorithms were implemented for a discount factor of $\gamma = 0.95$ and $\theta = 10^{-6}$.

2. The total number of calls made to the environment's transition function, `env.get_transitions_at_time(state,action)`, is recorded in Table 1. It can be observed that PI makes more number of calls than VI. This is justified the way both the algorithms work. In one epoch, VI just performs the Bellman backup only once and then constructs a greedy policy out of it, whereas PI performs the Bellman backup until the value function converges and then constructs an greedy optimal policy. Hence, PI makes more calls to the `env.get_transitions_at_time(state,action)` than VI.

Table 1: Number of transition calls to the stationary environment.

Algorithm	**# calls**
Value Iteration	168000
Policy Iteration	411200

3. The final policies obtained from both algorithms are optimal and identical. Both VI and PI are guaranteed to converge to an optimal policy as they solve the Bellman Optimality Equation:

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right)$$

This equation signifies that the value of a state under an optimal policy is the maximum expected return, consisting of the immediate reward plus the discounted value of the subsequent state. This equation has a unique solution for $V^*(s)$, and any policy that is greedy with respect to $V^*(s)$ is an optimal policy.

VI's convergence is guaranteed by the contraction mapping theorem, where each iteration reduces the error between the current value estimate V_k and the true optimal value V^*. PI's convergence is guaranteed by the Policy Improvement Theorem, which ensures that each new policy is a strict improvement upon (or equal to) the old one. Given a finite number of policies in a finite MDP, this iterative improvement must terminate at the optimal policy.

4. The performance of each policy was evaluated over 20 episodes using different seeds ($\{0, 1, \ldots, 19\}$). The mean and standard deviation of the rewards are reported in Table 2. It can be observed that since both the algorithms converge to the same true optimal policy, hence their rewards are also same.

Table 2: Reward statistics for Value and Policy Iterations.

Algorithm	**Mean Reward**	**Std Dev of Reward**
Value Iteration	47	21.79
Policy Iteration	47	21.79

5. An analysis of policy performance for different values of γ is presented in Table 3. It can be observed that TDVI has better rewards against standard VI because the policy learned in TDVI is non-stationary which is ideal for the non-stationary environment.

Table 3: Comparison for different values of γ.

γ	**# calls in VI**	**# calls in PI**	**PI Reward**	**VI Reward**	**Identical Policies**
0.3	89600	189600	32 ± 57.24	32 ± 57.24	False
0.5	145600	204800	32 ± 57.24	32 ± 57.24	True
0.95	168000	411200	47 ± 21.79	47 ± 21.79	True

- The discount factor γ dictates the agent's foresight. A high γ values future rewards, requiring more iterations for value propagation and thus more computation. A low γ makes the agent myopic, leading to faster convergence based on immediate rewards.
- PI generally requires more transition calls due to its expensive policy evaluation step, which must converge before each policy improvement step can occur. This contrasts with VI's simpler, single Bellman update per state per iteration.
- A myopic agent (low γ) might prefer a safe, immediately rewarding path, while a far-sighted agent (high γ) may choose a more complex path for a higher cumulative reward.
- For very low γ, long-term consequences are heavily discounted, which can make multiple actions appear equally optimal. Since VI and PI explore the policy space differently, they may converge to different, but equally optimal, policies for that specific γ.

6. Links to generated GIFs for one episode run with the policy derived using $\gamma = 0.95$:

- Value Iteration: https://drive.google.com/file/d/1kDBDHC2tLG-G-O2Svy5jM4UQYQU86ZIq/view?usp=sharing
- Policy Iteration: https://drive.google.com/file/d/10dErlVWTYFTPp8o6exdlyz94T3si4SLO/view?usp=sharing

1.2 Non-Stationary Environment Analysis

1. For the non-stationary environment, a time-dependent version of Value Iteration was implemented.

2. In the time-dependent VI, the number of transition calls to `env_degraded.get_transitions_at_time(state,action,time_step)` was $9,184,000$.

3. For the standard VI (ignoring time), the number of transition calls was $341,600$. The significant difference arises because the state space for Time-Dependent Value Iteration (TDVI) is much larger, defined by `(x, y, has_shot, time_step)`, compared to standard VI's state space of `(x, y, has_shot)`. TDVI must therefore explore a larger state-time space, resulting in more calls.

4. Performance was evaluated over 20 seeds. The reward statistics are in Table 4. TDVI results to a non-stationary policy which tends to perform better than VI's stationary policy on the non-stationary environment.

Table 4: Reward Statistics for TDVI and Standard VI.

Algorithm	**Mean Reward**	**Std Dev of Reward**
Time-Dependent VI	37.7	39.62
Standard VI	23.65	44.71

5. Links to generated GIFs for one episode:

   - Time-Dependent Value Iteration: https://drive.google.com/file/d/1LgXs25xnh4vwSqKjEZiLr4dkfC364P-j/view?usp=sharing
   - Standard Value Iteration: https://drive.google.com/file/d/1DyHsKKO6lJPxLh1FqmwOJNHPGejBH0tA/view?usp=sharing

1.3 Can we do something better?

1. Standard VI is synchronous, performing a full sweep of all states in each iteration. A more efficient, asynchronous approach called Prioritized Sweeping was implemented. This modified VI uses a priority queue to manage states based on their Bellman error.

   - The algorithm prioritizes states with the highest Bellman error.
   - In each step, it performs a Bellman backup only for the single state with the highest priority.

- Following the backup, the Bellman error for the affected predecessor states is recalculated, and their priorities are updated.
- This process continues until the maximum priority in the queue falls below the convergence threshold, focusing computation where it is most needed.

2. The total number of transition calls is compared in Table 5. The modified VI significantly reduces the number of calls.

Table 5: Number of transition calls for different algorithms.

Algorithm	**# calls**
Value Iteration	168,000
Policy Iteration	411,200
Modified Value Iteration	8,505

2 Dynamic Programming

2.1 Online Knapsack

1. For the Online Knapsack problem, the value function was implemented across the state space defined by `(time_step, current_knapsack_weight, current_item)`. Due to the large state space, training was restricted to 1000 iterations. Figure 1 shows the knapsack value evolution across five episodes for Value Iteration.

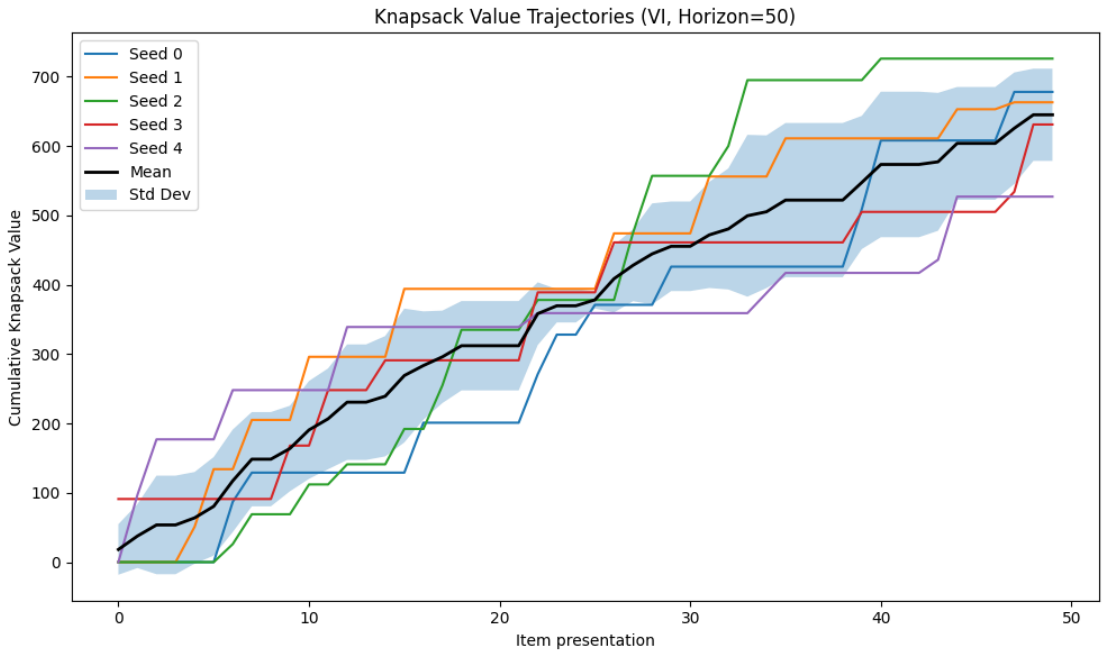


Figure 1: Evolution of knapsack value for Value Iteration (Horizon=50).

2. Similarly, Policy Iteration was implemented. Figure 2 shows the knapsack value evolution for Policy Iteration.

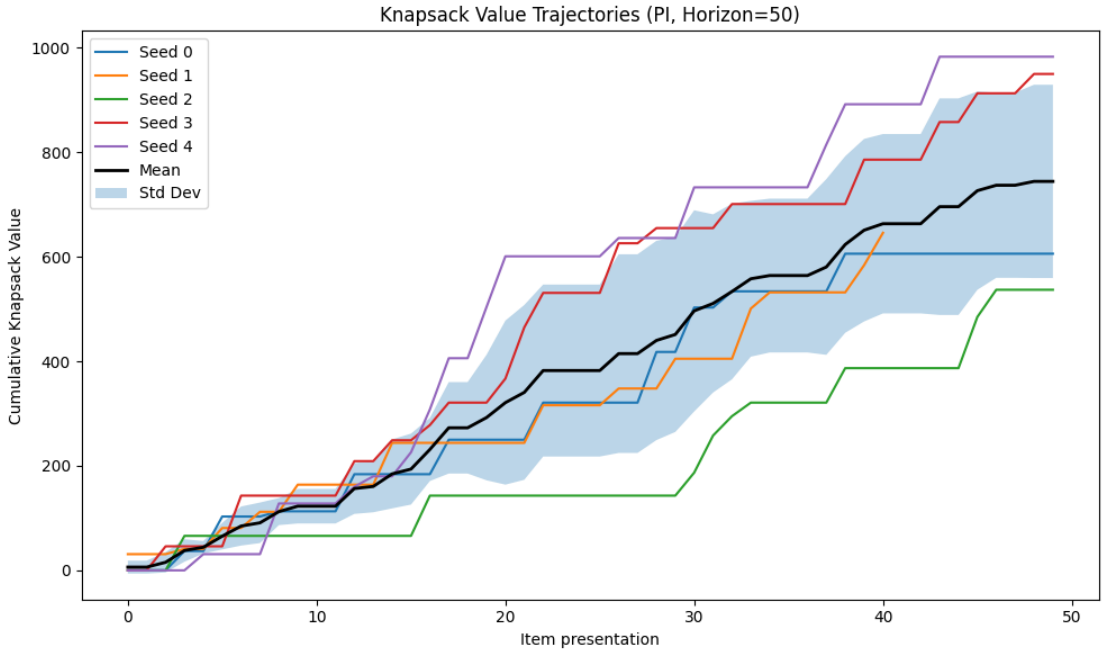


Figure 2: Evolution of knapsack value for Policy Iteration (Horizon=50).

3. Links to GIFs illustrating the knapsack filling process for VI with different time hori-

zons:

- Horizon 10: `https://drive.google.com/drive/folders/1snlgEJPCp0Qo30OcqxbI_supchvqQ7DH?usp=sharing`
- Horizon 50: `https://drive.google.com/drive/folders/1Tt-r2DojR5t5Dd_ClENZ80uekjPIkiES?usp=sharing`
- Horizon 500: `https://drive.google.com/drive/folders/1ef0ldQlqESZvMSoSQbb4wJyeuV1trkBg?usp=sharing`

2.2 Portfolio Optimization

1. **Value Iteration Results:** The execution times for VI are in Table 6. The training curves and post-training trajectories are shown in Figures 3 through 6. One of the key observations while training the Value Iteration algorithm is that for a finite-horizon MDP problem, irrespective of any value for γ (convergence value differs), the value iteration converges in one epoch. This means that instead of iterating over the value iteration value function approximation and greedy policy extraction forever until convergence, we can just do that step only once. That implies that Value Iteration for finite-horizon MDPs is equivalent to the Finite-Horizon Dynamic Programming algorithm.

Table 6: Execution time of VI for different price sequences.

Sequence	γ	**Execution Time (s)**
$[1, 3, 5, 5, 4, 3, 2, 3, 5, 8]$	0.999	0.0837
$[1, 3, 5, 5, 4, 3, 2, 3, 5, 8]$	1.0	0.0839
$[2, 2, 2, 4, 2, 2, 4, 2, 2, 2]$	0.999	0.0534
$[2, 2, 2, 4, 2, 2, 4, 2, 2, 2]$	1.0	0.0524
$[4, 1, 4, 1, 4, 4, 4, 1, 1, 4]$	0.999	0.0528
$[4, 1, 4, 1, 4, 4, 4, 1, 1, 4]$	1.0	0.0525

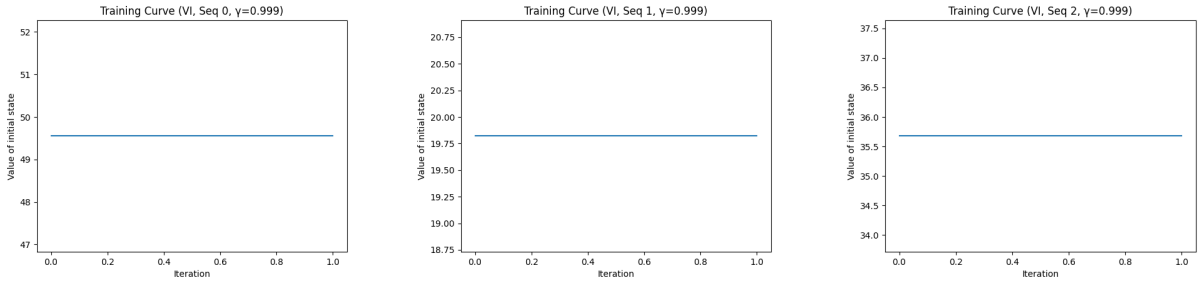


Figure 3: VI Training Curves for three sequences ($\gamma = 0.999$).

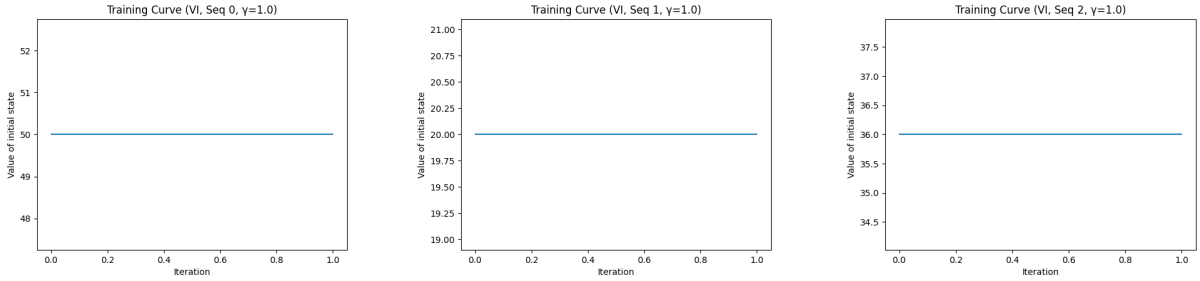


Figure 4: VI Training Curves for three sequences ($\gamma = 1.0$).

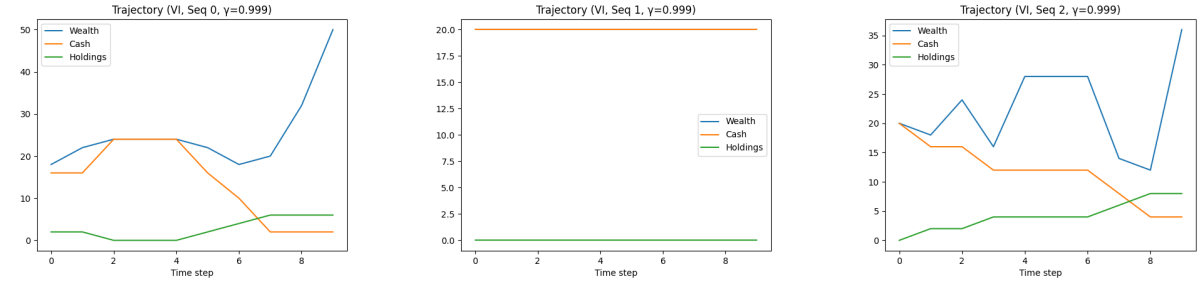


Figure 5: VI Post-Training Trajectories for three sequences ($\gamma = 0.999$).

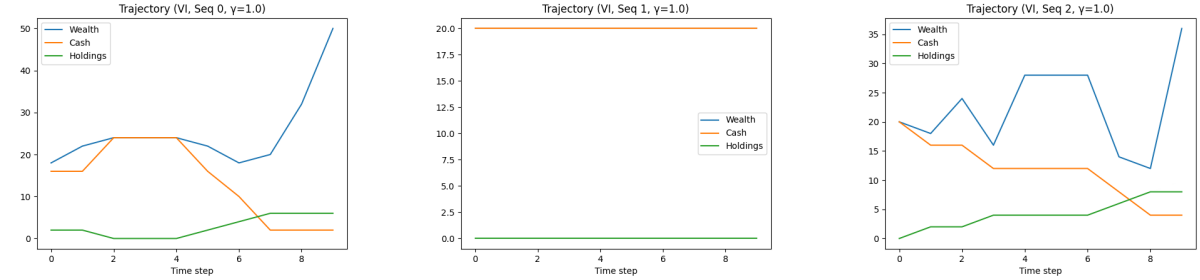


Figure 6: VI Post-Training Trajectories for three sequences ($\gamma = 1.0$).

2. **Policy Iteration Results:** The execution times for PI are in Table 7. The training curves and post-training trajectories are shown in Figures 7 through 10. One key takeaway from here is that, unlike VI, we cannot optimize Policy Iteration algorithm as a variant of finite-horizon DP, and we have to perform it as it is.

Table 7: Execution time of PI for different price sequences.

Sequence	γ	**Execution Time (s)**
$[1, 3, 5, 5, 4, 3, 2, 3, 5, 8]$	0.999	0.4153
$[1, 3, 5, 5, 4, 3, 2, 3, 5, 8]$	1.0	0.4087
$[2, 2, 2, 4, 2, 2, 4, 2, 2, 2]$	0.999	0.0746
$[2, 2, 2, 4, 2, 2, 4, 2, 2, 2]$	1.0	0.0741
$[4, 1, 4, 1, 4, 4, 4, 1, 1, 4]$	0.999	0.2595
$[4, 1, 4, 1, 4, 4, 4, 1, 1, 4]$	1.0	0.2618

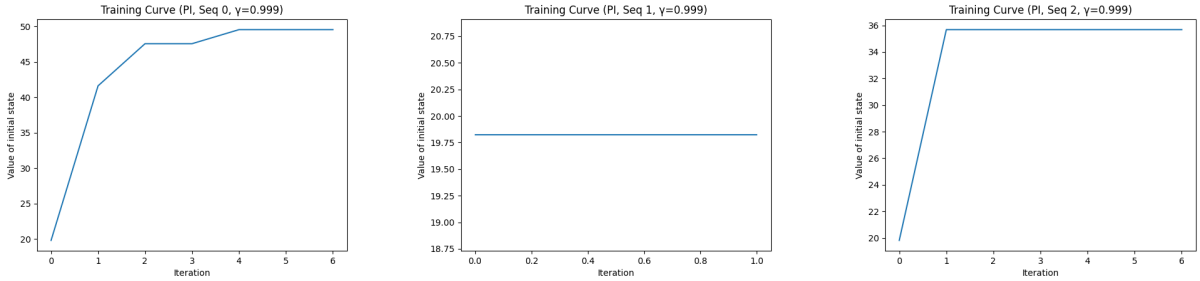


Figure 7: PI Training Curves for three sequences ($\gamma = 0.999$).

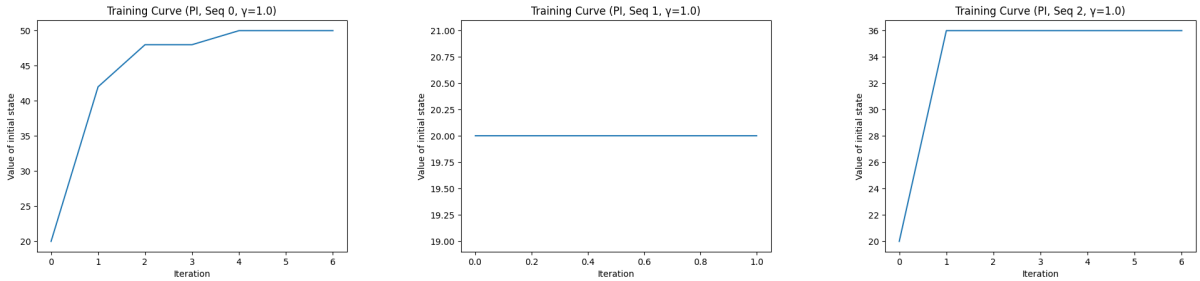


Figure 8: PI Training Curves for three sequences ($\gamma = 1.0$).

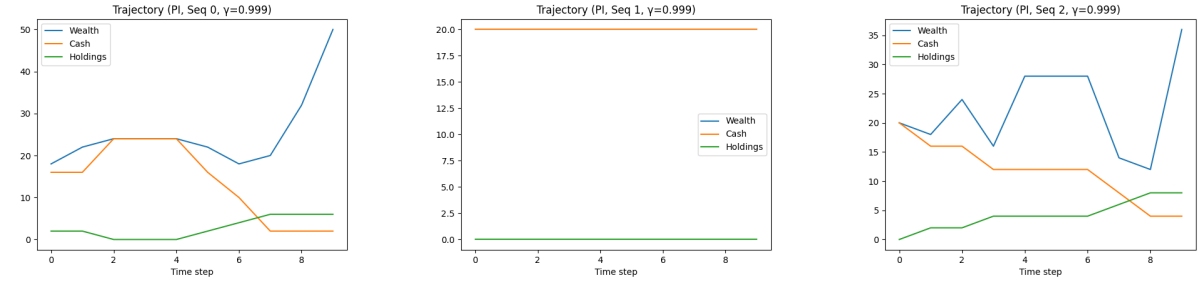


Figure 9: PI Post-Training Trajectories for three sequences ($\gamma = 0.999$).

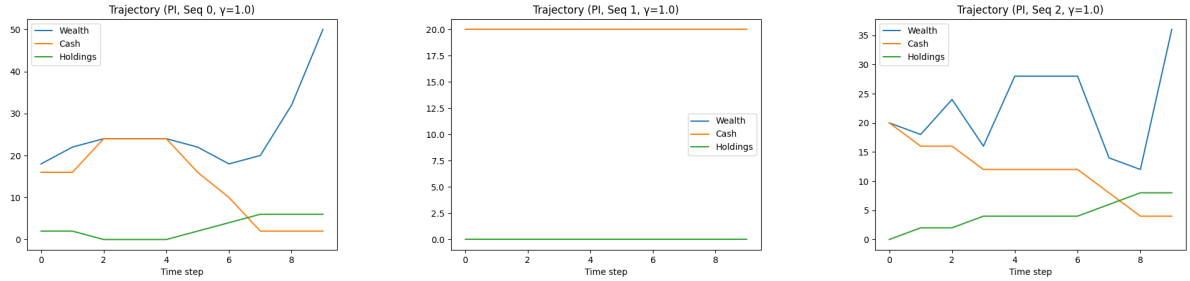
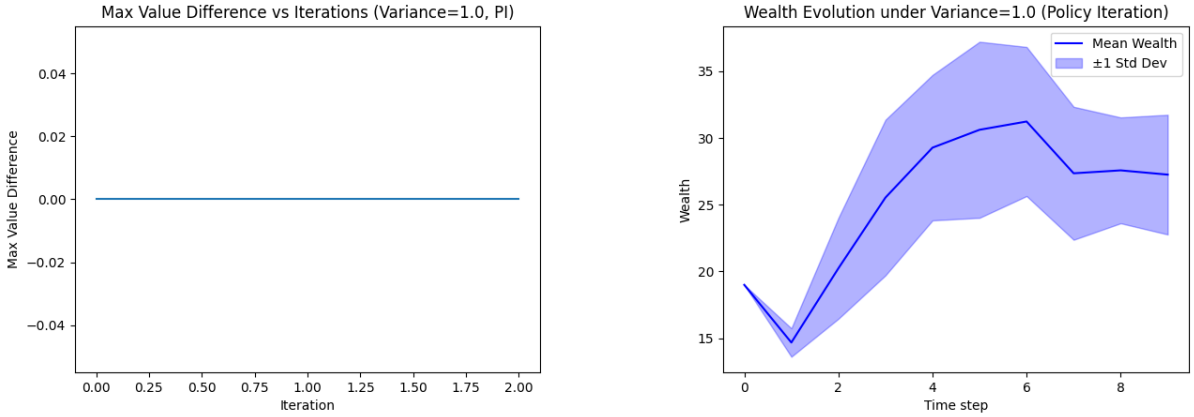


Figure 10: PI Post-Training Trajectories for three sequences ($\gamma = 1.0$).

3. **High-Variance Environment:** Yes, the Policy Iteration algorithm converged to the optimal solution in the high-variance setting (variance = 1.0). The convergence plot and resulting wealth trajectory are shown in Figure 11.



(a) Convergence of PI (Max Value Difference). (b) Wealth trajectory of PI.

Figure 11: PI Performance in the high-variance setting.