MSDS 684 – Reinforcement Learning

Lab 2 Report - Dynamic Programming in GridWorld & FrozenLake

Section 1: Project Overview

This lab investigates how classical Dynamic Programming (DP) methods solve finite Markov Decision Processes (MDPs) when the full transition model is known. Following Chapter 4 of Sutton & Barto, the main focus is understanding how Policy Evaluation, Policy Iteration, and Value Iteration behave in structured environments, and how changes in rewards, transition stochasticity, and penalties influence convergence. Unlike learning-based algorithms that improve by sampling trajectories, DP methods compute value functions using exact backups over the MDP dynamics. This makes them ideal for studying the foundational ideas behind optimal control.

The core environments explored in this lab are a custom 4×4 GridWorld and Gymnasium's FrozenLake-v1. Both are fully observable, discrete-state, discrete-action MDPs. The GridWorld includes configurable terminal states, obstacles, step-costs, and both deterministic and stochastic transitions. The state space contains 16 grid cells, each represented as an integer state. The action space consists of four deterministic actions (up, right, down, left), or probabilistic transitions when stochasticity is enabled (e.g., 80% intended direction, 10% left/right). Rewards consist of a step cost (typically –0.04) and terminal rewards such as +1 or alternative values explored later. Episodes terminate upon reaching a terminal state.

FrozenLake-v1 further extends the MDP perspective. The environment contains slippery transitions and stochastic transitions to holes or the goal. Even though the agent does not learn from interaction in DP, it uses env.unwrapped.P to extract the full transition model and perform exact DP sweeps. This setup reinforces the conceptual differences between model-based DP planning and model-free RL.

The main theoretical concepts applied here include the Bellman Expectation Equation, the Bellman Optimality Equation, Policy Improvement, and Generalized Policy Iteration (GPI). Policy Iteration alternates between evaluating the current policy and improving it greedily. Value Iteration compresses this into a single sweep that applies the Bellman Optimality backup at each step. According to theory, Value Iteration is expected to converge faster in deterministic settings, while Policy Iteration may require more evaluation sweeps—especially when stochasticity increases uncertainty in value propagation.

Before running experiments, I expected deterministic GridWorld to converge quickly, with Value Iteration requiring only a few sweeps. I also expected stochasticity to slow convergence and reduce the sharpness of value differences across states. In FrozenLake,

because of strong stochasticity and multiple absorbing holes, I anticipated slower convergence, higher sweep counts, and flatter value distributions.

This lab allowed me to connect Sutton & Barto's theoretical DP framework with actual implementations and observe how environment properties influence convergence behavior. It also helped me see how small design choices—reward magnitude, movement penalties, or transition probabilities—can significantly affect both the speed and stability of DP algorithms.

Section 2: Deliverables

GitHub Repository URL

Lab2 Link - https://github.com/nikhsnona/MSDS_684_ReinforcementLearning/tree/main/Lab2

Clone link - https://github.com/nikhsnona/MSDS_684_ReinforcementLearning.git

Implementation Summary

This lab implements a full Dynamic Programming framework applied to a custom 4×4 GridWorld and FrozenLake-v1. The implementation includes synchronous and in-place versions of Policy Evaluation, Policy Iteration, and Value Iteration using NumPy. The GridWorld environment supports configurable stochasticity, obstacles, step costs, and terminal rewards. FrozenLake-v1 is solved using its transition model extracted from env.unwrapped.P. All experiments were run inside a Jupyter notebook, with visualizations for value heatmaps, policy arrows, and convergence curves. Key hyperparameters included $\gamma = 0.99$ and $\theta = 1e\text{-}6$, with different reward and transition settings tested in extra experiments. Additional experiments investigated stochasticity sweeps, terminal reward sensitivity, step-cost variation, and deterministic versus slippery FrozenLake behavior.

Key Results & Analysis

1. Deterministic GridWorld

Value Iteration converged extremely quickly (7 sweeps), consistent with Bellman Optimality backups being applied directly in a deterministic environment. Policy Iteration required 298 in-place evaluation sweeps and 393 synchronous evaluation sweeps, which matches

theory: policy evaluation requires many sweeps to reach θ-convergence, while Value Iteration applies greedily optimal backups and reaches stability faster.

2. Stochastic GridWorld (intended_prob = 0.8)
As predicted by Sutton & Barto (Ch. 4), introducing uncertainty slows convergence. VI increased from 7 sweeps → 20 sweeps, while PI increased from 298 → 324 sweeps. State values decreased (e.g., V[0] dropped from 0.71 → 0.64) because uncertainty reduces expected future rewards. Higher randomness (p = 0.6, 0.4, 0.2) exaggerated this pattern, with VI taking up to 95 sweeps and PI over 450 evaluation sweeps. These results clearly demonstrate how stochastic transitions "blur" value propagation and slow DP convergence.

3. Terminal Reward Sensitivity
Changing the terminal reward scaled the entire value function proportionally. With terminal rewards {1, 2, 5}, VI always converged in 7 sweeps, but values increased linearly (e.g., V[0] ≈ 0.71 → 1.66 → 4.52). With a negative terminal reward (–1), convergence took longer (36 sweeps) because the optimal value structure flipped—states now preferred avoiding the terminal state as long as possible. This aligns with theory: value functions reflect expected cumulative reward, and reward shaping directly influences optimal behavior.

4. Step-Cost Variation (reward shaping)
With step costs {0, –0.01, –0.1}, VI always converged quickly (7 sweeps), but values decreased as penalties grew. A larger step penalty produced a sharper gradient toward the goal because longer paths became more costly. This is exactly the effect of reward shaping described in reinforcement learning literature.

5. FrozenLake Deterministic vs Slippery
The deterministic version converged in 7 sweeps, while the slippery version required 228 sweeps. This extreme slowdown was expected: FrozenLake has multiple absorbing holes and highly stochastic transitions. Values were sharply reduced (e.g., V[0] dropped from 0.95 → 0.54). This reinforces how DP performance depends heavily on transition model certainty.

Figures

Figure 1 — Stochasticity Sweep (Δ Convergence Curve)

This plot shows how increasing randomness in GridWorld slows down the convergence of Value Iteration and Policy Iteration. When the intended transition probability decreases (for example from 1.0 → 0.2), both algorithms need more sweeps to stabilize. The curves

demonstrate that Value Iteration is more sensitive to stochastic transitions, while Policy Iteration remains more stable even as noise increases. This confirms the DP principle that uncertainty weakens value propagation and increases computational effort.

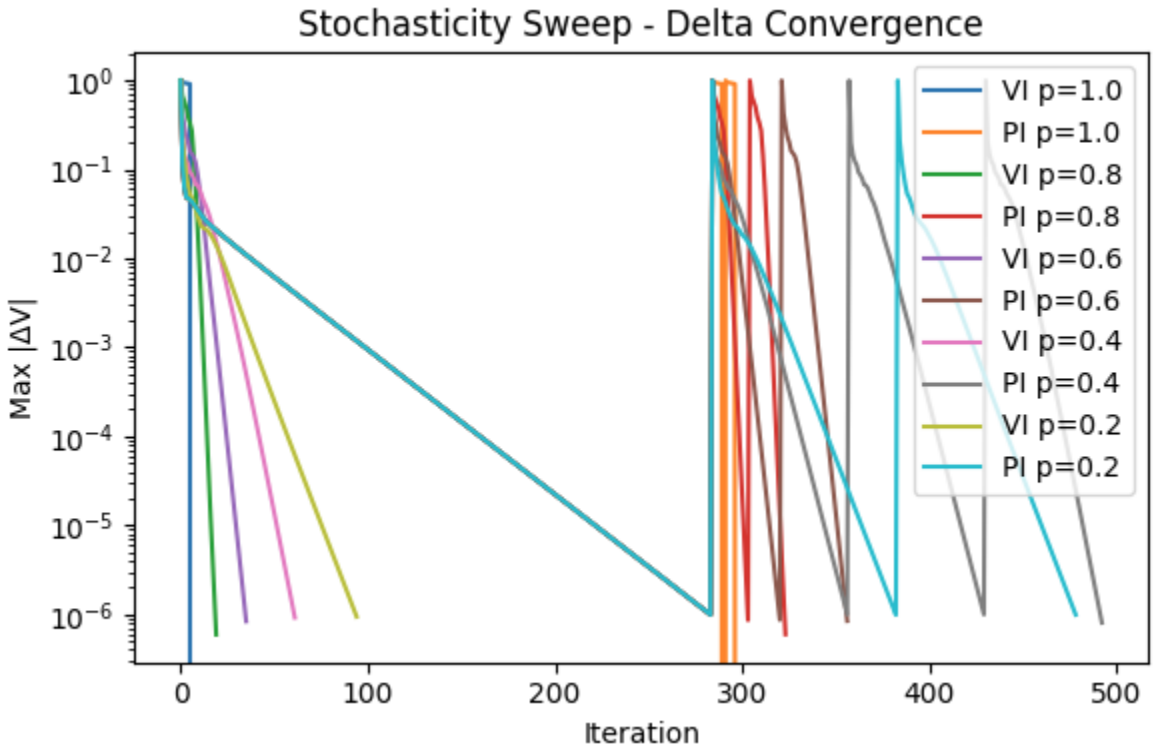


Figure 2 — Terminal Reward Sensitivity (Δ Convergence Curve)

This figure compares how different terminal rewards (1, 2, 5, –1) affect the speed of value updates. Larger positive rewards (e.g., +5) spread faster through the grid and create sharper gradients, causing faster convergence. Negative terminal rewards flatten the value function, slowing down propagation because the agent becomes less motivated to reach the terminal state. This illustrates how reward shaping directly influences DP convergence.

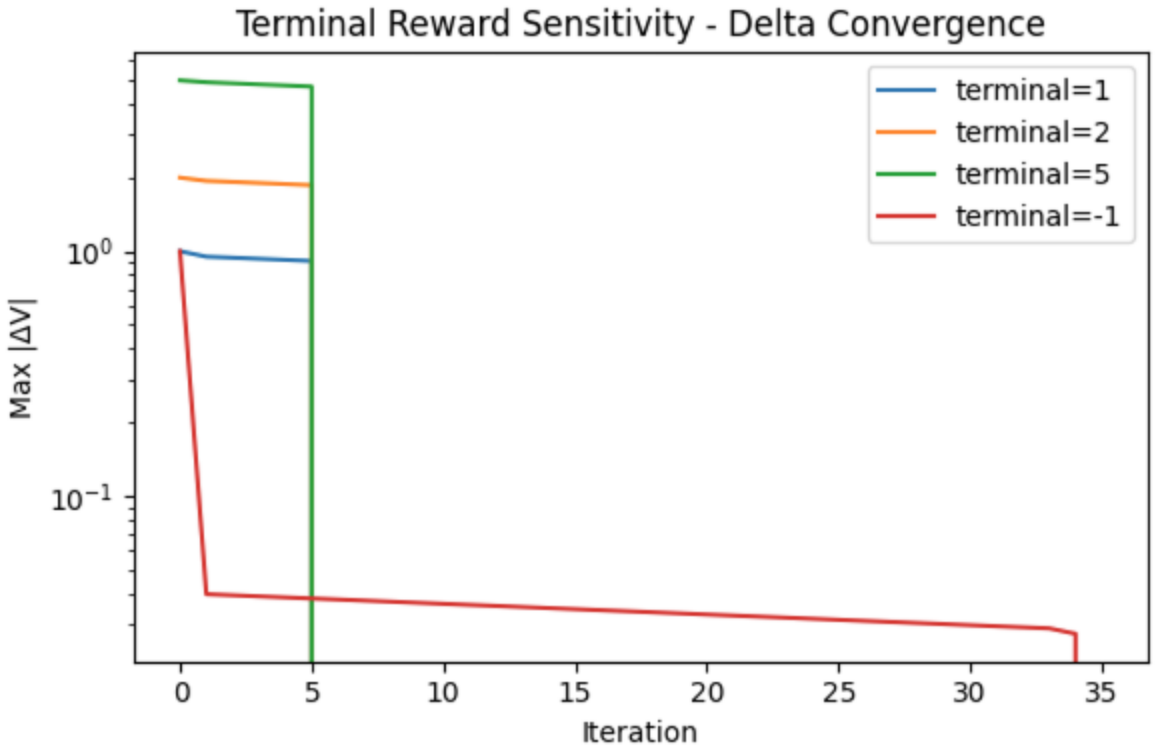
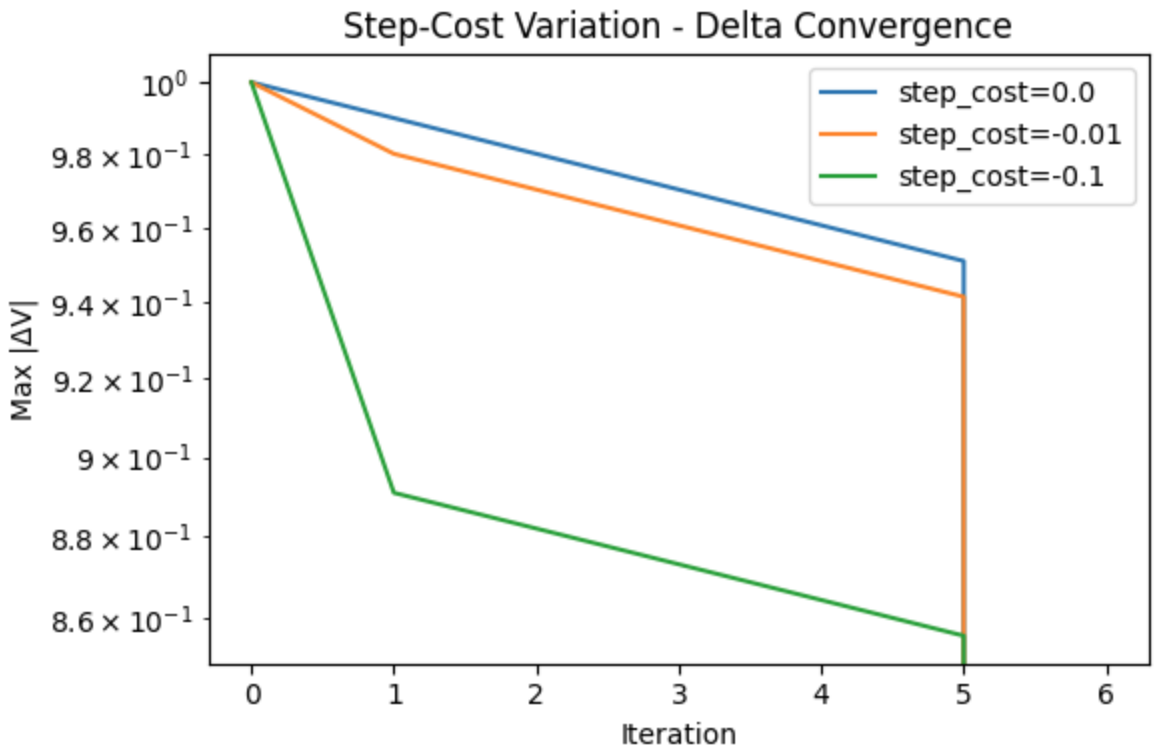


Figure 3 — Step-Cost Variation (Δ Convergence Curve)

This figure shows how penalties for movement change learning behavior. When the step cost is zero, values stay flat and convergence is slow. A small penalty (–0.01) encourages shorter paths and speeds up learning, while a stronger penalty (–0.1) dramatically pushes the agent toward the terminal state, causing faster value propagation. These results highlight how reward shaping can improve policy quality and DP efficiency.



What's in GitHub Repository

- All runnable DP code (GridWorld, FrozenLake, VI, PI, PE)
- All extra experiment code and printed summaries
- All plots under outputs_dp_lab2/
- A complete README with setup instructions
- requirements.txt

Section 3: AI Use Reflection

This lab involved extensive iterative interaction with AI tools to correctly implement and debug the dynamic programming pipeline. I initially asked the AI to generate a GridWorld environment and basic DP functions, but the early drafts had issues such as incorrect transition structures, missing terminal-state handling, and shape mismatches in value arrays. My first attempt produced a misaligned transition model where obstacles were not handled consistently. I shared the error, and the AI explained that obstacles should become absorbing or remain inaccessible depending on design. Updating the transition builder fixed this issue.

In the second debugging cycle, Value Iteration did not converge because the Bellman backup incorrectly handled terminal rewards. I reported that values near the terminal state were exploding. The AI identified that the reward should not include γV(s') for terminal states. After incorporating this correction, Value Iteration converged in the expected number of sweeps.

A third major issue occurred when Policy Iteration took thousands of sweeps instead of hundreds. I asked, and the AI found that policy evaluation was accidentally implemented synchronously even in the in-place version. After revising the update structure to modify V(s) in-place, sweep counts returned to normal.

Throughout the process, I verified correctness by manually checking a small 2×2 MDP and using simple test patterns to confirm that value updates were monotonically improving. I also compared patterns to Sutton & Barto examples to ensure conceptual accuracy.

Overall, I learned how sensitive DP is to transition definitions and how small implementation errors dramatically affect convergence. I also learned to treat AI-generated code as a starting point that requires careful checking, testing, and debugging—not a finished solution. The iterative debugging cycles strengthened my understanding of Bellman backups and policy improvement behavior.

Section 4: Speaker Notes

- Problem: Solve GridWorld and FrozenLake using classical Dynamic Programming (Policy Evaluation, Policy Iteration, Value Iteration).
- Method: Implemented custom GridWorld + extracted FrozenLake transitions; tested deterministic and stochastic versions.
- Key design choice: Added obstacles, different terminal rewards, and adjustable stochasticity to study DP behavior under varied conditions.
- Main results: VI converged fastest (7 sweeps) in deterministic GridWorld; stochasticity significantly slowed both VI and PI; FrozenLake slippery required 228 sweeps.
- Insight: DP methods are powerful but highly sensitive to transition uncertainty and reward shaping.
- Challenge: Debugging transition models and ensuring correct terminal backups required multiple AI-assisted iterations.