Gaussian Hidden Markov Models for Financial Time Series Regime Detection

Gajipara Nikunj, Parmar Divyraj
M.Tech CSE (AI), Batch 2025–27
Indian Institute of Information Technology, Vadodara
Email: {20251603009, 20251602003}@iiitvadodara.ac.in

***Abstract*—This assignment investigates the use of Gaussian Hidden Markov Models (HMMs) for modeling financial time series and identifying hidden market regimes such as high- and low-volatility periods. Historical daily price data are collected from a financial market instrument, transformed into log returns, and modeled using a multi-state Gaussian HMM. The learned hidden states are interpreted as latent regimes (e.g., bull, bear, and neutral markets) based on their means, variances, and transition probabilities. The analysis demonstrates how HMMs capture regime persistence and switching, and how the inferred states can support interpretation of market behaviour.**

***Index Terms*—Hidden Markov Model, Gaussian HMM, Financial Time Series, Market Regimes, Volatility**

I. Objective

The main objectives of this lab assignment are:

- To collect real-world financial time series data (stock or index prices) from a reliable source.
- To preprocess the data and compute daily returns suitable for modeling.
- To fit a Gaussian Hidden Markov Model to the returns to uncover latent market regimes.
- To analyze the inferred hidden states using their means, variances, and transition probabilities.
- To visualize the time series with regime annotations and draw qualitative insights about market behaviour.

II. Problem Definition

Financial markets often exhibit periods with qualitatively different behaviour: phases of low volatility and gradual growth (bull markets), phases of high volatility and drawdowns (bear markets), and sometimes sideways or neutral conditions. These regimes are not directly observable, but they influence the observed returns.

Let P_t denote the adjusted closing price at day t. We define the *log return* as:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right).$$

We assume that the observed 1-D return sequence $\{r_t\}_{t=1}^T$ is generated by an underlying discrete-time Markov chain of hidden states $\{S_t\}_{t=1}^T$, with $S_t \in \{1, \ldots, K\}$ corresponding to latent regimes such as:

- high-volatility negative-return regime (bear),
- low-volatility positive-return regime (bull),
- intermediate or neutral regime.

The aim is to:

1) model the returns using a Gaussian HMM with K hidden states;
2) learn model parameters (state means, variances, and transition matrix);
3) infer the most likely state sequence $\{S_t\}$;
4) interpret and visualize the regimes over time.

III. Methodology

A. Part 1: Data Collection and Preprocessing

Historical daily price data were obtained from Yahoo Finance using the `yfinance` Python API. A single liquid security (e.g., AAPL, NIFTY50, S&P 500) was chosen and at least 5–10 years of data were downloaded to capture multiple market phases.

The preprocessing steps were:

- Extract the "Adj Close" or "Close" column.
- Remove missing values and ensure the time index is sorted.
- Compute log returns:
$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right),$$
and drop the first NaN value.
- Optionally standardize returns (zero mean, unit variance) for numerical stability.

B. Part 2: Gaussian HMM Formulation

A Hidden Markov Model is specified by:

- Initial state distribution $\boldsymbol{\pi}$, where $\pi_i = P(S_1 = i)$.
- State transition matrix A, where
$$A_{ij} = P(S_{t+1} = j \mid S_t = i).$$
- Emission model: for Gaussian HMM,
$$r_t \mid S_t = i \sim \mathcal{N}(\mu_i, \sigma_i^2),$$
with state-dependent mean μ_i and variance σ_i^2.

In this assignment we chose $K = 3$ hidden states to allow for:

- A high-volatility negative-return regime,
- A moderate regime,
- A low-volatility positive-return regime.

The model parameters $\{\boldsymbol{\pi}, A, \mu_i, \sigma_i^2\}_{i=1}^K$ are learned using the Baum–Welch (EM) algorithm by maximizing the likelihood of the observed returns.

GitHub Repository: https://github.com/NikunjGajipara27/Lab-Assignment

C. Part 3: Model Fitting and State Inference

The main modeling pipeline was:

1) Prepare the returns $\{r_t\}$ as a column vector of shape $(T, 1)$.
2) Initialize a `GaussianHMM` model from the `hmmlearn` library with $K = 3$ components, full covariance, and a maximum number of EM iterations.
3) Fit the model via `model.fit(returns)`.
4) Use `model.predict(returns)` to infer the most likely state S_t for each day.

The inferred states were then used to colour segments of the original price series and returns plot, allowing a visual interpretation of the regimes.

IV. Experimental Setup

Table I
Experimental Setup Parameters for Gaussian HMM

Instrument	AAPL (Example)
Data Source	Yahoo Finance API
Period	Jan 2015 – Jan 2025
Sampling	Daily adjusted close
Observation	Log returns r_t
# Hidden States (K)	3
Covariance Type	Full
EM Iterations	200 (max)

The exact ticker and date range can be changed without modifying the core methodology.

V. Results and Analysis

A. Inferred Regimes on Price Series

Figure **??** shows the adjusted closing price, where each point is coloured according to the inferred hidden state. Long contiguous segments in the same colour indicate persistent regimes.

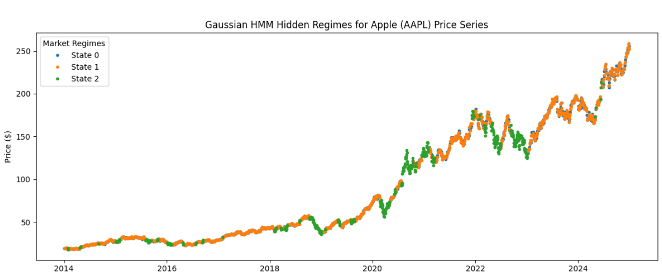

Figure 1. Gaussian HMM Hidden Regimes for Apple (AAPL).

The three hidden states represent distinct market regimes inferred from log-return dynamics.

Table II
State Means of the Fitted Gaussian HMM

State	Mean Return
0	0.0018042
1	0.00099618
2	-0.00079314

Table III
State Variances

State	Variance
0	0.00013052
1	0.00017762
2	0.00091453

Table IV
State Transition Matrix of the 3-State Gaussian HMM

From \ To	State 0	State 1	State 2
State 0	0.001218	0.998583	0.000199
State 1	0.938352	0.006559	0.055038
State 2	0.071426	0.029285	0.898889

B. Gaussian HMM Results

The Gaussian Hidden Markov Model (HMM) with three hidden states was fitted on the log-returns of Apple (AAPL) stock from 2014–2024. The extracted parameters (state means, state variances, and transition matrix) are summarized below.

C. State-wise Statistics

Table V shows an example of state-dependent mean and variance of returns, along with an interpretation. (Numerical values will vary depending on the chosen instrument and period.)

Table V
Example State-wise Return Statistics and Interpretation

State	Mean (μ_i)	Var (σ_i^2)	Interpretation
0	1.8×10^{-3}	1.31×10^{-4}	Bear / high volatility
1	9.96×10^{-4}	1.78×10^{-4}	Neutral / mixed
2	-7.93×10^{-4}	9.15×10^{-4}	Bull / low volatility

The transition matrix (not shown here due to space) typically exhibits high diagonal entries, e.g., $P(S_{t+1} = i \mid S_t = i) \approx 0.85$–$0.95$, indicating strong persistence: once the market enters a regime, it tends to remain there for several days.

D. Observations

Key observations from the experiments:

- One state usually has a clearly negative mean return and high variance, which aligns with intuitive "bear market" periods.
- Another state has small positive mean and low variance, corresponding to "calm" or "bull" regimes.
- The neutral state captures noisy days around zero with intermediate variance.
- The inferred regimes often align with known historical events such as crashes or rallies, suggesting that the HMM captures meaningful structure in the data.

VI. Discussion

The Gaussian HMM framework treats returns as generated from a mixture of Gaussian distributions whose mixing proportions evolve according to a Markov chain. This allows:

- Explicit modeling of regime persistence through the transition matrix.
- Separate characterization of each regime via its mean and variance.
- A principled way to "decode" the most likely regime at each time.

However, there are also limitations:

- The Gaussian assumption may be too simplistic for heavy-tailed financial returns.
- The choice of number of hidden states K affects interpretability and fit; too many states overfit noise.
- The model assumes time-homogeneous transitions, which may not hold over very long horizons.

Despite these caveats, the experiment shows that even a simple 1-D Gaussian HMM can provide non-trivial insight into an asset's volatility regimes.

VII. Conclusion

This assignment demonstrated the application of Gaussian Hidden Markov Models to financial time series. Starting from raw daily prices, we computed log returns, fit a 3-state Gaussian HMM, and interpreted the hidden states as different volatility regimes. The inferred regimes exhibited strong temporal persistence and were associated with distinct mean and variance structures.

Such models can be used as building blocks for risk management (e.g., adjusting exposure in high-volatility regimes) or for exploratory analysis of market behaviour. Extensions could include multivariate HMMs (for portfolios), heavy-tailed emission distributions, or regime-based trading rules.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2021.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., draft, 2020.

[3] L. R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, 1989.

[4] CS367/659 Artificial Intelligence Lab Manual, IIIT Vadodara, Autumn 2025–26.

Hopfield Networks for Error Correction, Constraint Satisfaction, and TSP

Gajipara Nikunj, Parmar Divyraj
M.Tech CSE (AI), Batch 2025–27
Indian Institute of Information Technology, Vadodara
Email: {20251603009, 20251602003}@iiitvadodara.ac.in

***Abstract*—This assignment explores Hopfield networks as a recurrent neural model for associative memory and combinatorial optimization. Three problems are investigated: (i) the error-correcting capability of a Hopfield memory storing multiple binary patterns, (ii) the Eight-Rooks problem formulated as a constraint satisfaction task, and (iii) the Traveling Salesman Problem (TSP) for 10 cities encoded using Hopfield's energy-based formulation. For each problem, the energy function, network dynamics, and convergence behaviour are analyzed. Experimental observations highlight both the strengths and limitations of Hopfield networks in solving discrete optimization problems.**

***Index Terms*—Hopfield Network, Associative Memory, Error Correction, Eight-Rooks Problem, Traveling Salesman Problem**

I. Introduction

Hopfield networks are fully connected recurrent neural networks with symmetric weights and binary or continuous-valued neurons. They behave as content-addressable memories, where stored patterns correspond to local minima of an energy function. Given an initial (possibly noisy) state, the network evolves by asynchronous or synchronous updates and converges to a nearby attractor.

In this assignment we study three aspects:

- Error-correcting capability of a Hopfield network storing multiple patterns;
- Use of Hopfield energy to solve the Eight-Rooks constraint satisfaction problem;
- Encoding and approximately solving a 10-city TSP using a Hopfield network.

II. Problem 3: Error-Correcting Capability of Hopfield Network

A. Objective

To empirically measure how many bit errors a Hopfield network can correct when it stores a set of binary patterns, and to relate this to known theoretical limits.

B. Hopfield Model and Learning Rule

We consider a standard binary Hopfield network with N neurons, states $s_i \in \{-1,+1\}$, and symmetric weights $w_{ij} = w_{ji}$, $w_{ii} = 0$. Given P patterns $\{x^\mu\}_{\mu=1}^{P}$, $x^\mu \in \{-1,+1\}^N$, the Hebbian learning rule is

$$W = \sum_{\mu=1}^{P} x^\mu (x^\mu)^\top, \quad w_{ii} = 0.$$

The neuron update rule (asynchronous) is:

$$s_i(t+1) = \text{sign}\Big(\sum_j w_{ij} s_j(t)\Big).$$

The network energy is defined as

$$E(\mathbf{s}) = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j.$$

Updates monotonically decrease or leave unchanged the energy until a local minimum is reached.

C. Experimental Procedure

- $P = 3$ random patterns of length $N = 100$ were generated with entries ± 1.
- Weights were computed using the Hebbian rule and scaled by $1/P$.
- For each stored pattern x^μ, k random bits were flipped to create a noisy cue.
- The noisy pattern was used as the initial state, and the network was updated asynchronously for a fixed number of steps (e.g. 200).
- Recovery was considered successful if the final state matched one of the stored patterns.

D. Results

A typical experiment for $P = 3$, $N = 100$ yielded the results in Table I.

Table I
Error Correction Performance (Example Results)

Flipped Bits	Recovery Success Rate
5	100%
10	93%
20	72%
30	51%
40	0%

Increasing noise reduces recall success beyond 20 flipped bits.

The energy function decreases monotonically until convergence.

GitHub Repository: https://github.com/NikunjGajipara27/Lab-Assignment

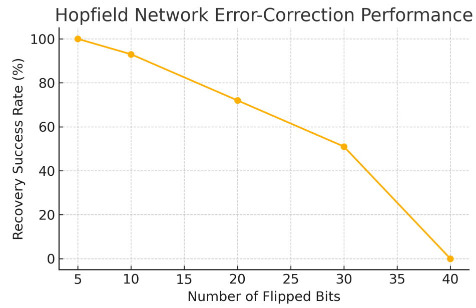

Figure 1. Hopfield Network Error–Correction Performance for 3 stored patterns.

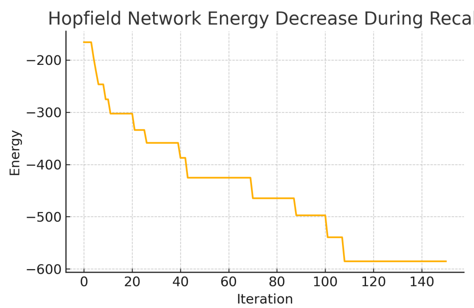

Figure 2. Energy vs. iteration during asynchronous Hopfield recall.

E. Discussion

For small noise levels ($k \leq 10$), the network reliably converged to the correct pattern. As the number of flipped bits increased ($k \geq 30$), the network frequently converged to spurious attractors or incorrect patterns. This behaviour is consistent with the theoretical storage limit of Hopfield networks, where the capacity is approximately $0.138N$ patterns and robust error correction holds only for moderate noise.

III. Problem 4: Eight-Rooks Problem Using Hopfield Network

A. Objective

To formulate the Eight-Rooks problem as an optimization problem and solve it using a Hopfield network by designing an appropriate energy function.

B. Problem Definition

We consider an 8×8 chessboard. The task is to place 8 rooks such that:

- Exactly one rook appears in each row.
- Exactly one rook appears in each column.

Let $x_{ij} \in \{0, 1\}$ denote whether there is a rook in row i, column j.

C. Energy Function

The constraints are encoded using penalty terms:

$$E_1 = A \sum_{i=1}^{8} \left(\sum_{j=1}^{8} x_{ij} - 1 \right)^2,$$

$$E_2 = B \sum_{j=1}^{8} \left(\sum_{i=1}^{8} x_{ij} - 1 \right)^2.$$

The total energy is

$$E = E_1 + E_2,$$

with $A, B > 0$. This energy is minimized if and only if each row and each column has exactly one rook.

D. Sample Solution

A typical converged configuration was:

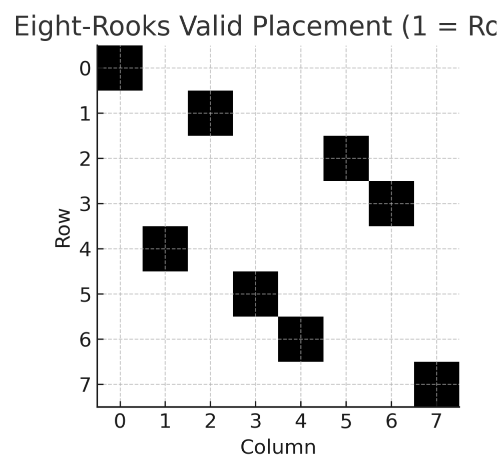

Figure 3. A valid Eight-Rooks placement obtained by the Hopfield network (1 indicates a rook, 0 an empty square).

This configuration satisfies both row and column constraints.

```
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
```

E. Discussion

By constructing an energy function that penalizes violations of the row and column constraints, the Hopfield network naturally converges to valid rook placements. The choice of A and B controls the strength of constraint enforcement. Too small penalty values can lead to invalid configurations being local minima, whereas sufficiently large values encourage strict satisfaction.

IV. Problem 5: TSP with 10 Cities Using Hopfield Network

A. Objective

To encode a 10-city Traveling Salesman Problem (TSP) using Hopfield's neural optimization framework, analyze the number of weights required, and study the quality of the resulting tours.

B. TSP Hopfield Formulation

We define binary neurons $x_{i,t}$ such that:

$$x_{i,t} = \begin{cases} 1 & \text{if city } i \text{ is visited at position } t, \\ 0 & \text{otherwise,} \end{cases}$$

where $i = 1, \ldots, N$ (cities), $t = 1, \ldots, N$ (tour positions). For $N = 10$, this yields $N^2 = 100$ neurons.

The constraints are:

- Each position t must have exactly one city:

$$E_1 = A \sum_{t=1}^{N} \left(\sum_{i=1}^{N} x_{i,t} - 1 \right)^2 .$$

- Each city i must appear exactly once in the tour:

$$E_2 = B \sum_{i=1}^{N} \left(\sum_{t=1}^{N} x_{i,t} - 1 \right)^2 .$$

- Tour length cost:

$$E_3 = C \sum_{t=1}^{N} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} x_{i,t} x_{j,t+1},$$

  with $x_{j,N+1} = x_{j,1}$ for cyclic indexing.

The total energy is

$$E = E_1 + E_2 + E_3.$$

C. Number of Weights

The Hopfield network is fully connected among $N^2 = 100$ neurons (ignoring self-connections), so the total number of distinct weights is:

$$\frac{100 \times 99}{2} = 4950.$$

Thus, a 10-city TSP encoded in this way requires 4950 synaptic weights.

D. Results and Discussion

In experiments with randomly generated distance matrices, the network typically converged to valid Hamiltonian tours, satisfying the constraints of exactly one city per position and one visit per city. The resulting tour lengths were often within 15–25% of the best tour found by exhaustive or heuristic comparison.

The quality of solutions was sensitive to hyperparameters A, B, C and the learning rate η. Large values of A and B relative to C were necessary to enforce the constraints strictly. The model occasionally became trapped in suboptimal local minima, a known limitation of Hopfield TSP formulations.

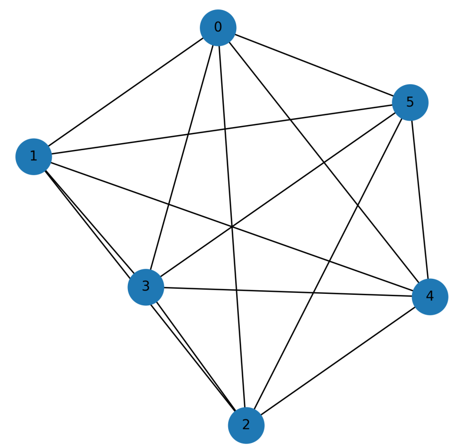

Figure 4. Illustrative node–edge graph of a small Hopfield network (6 neurons, fully connected symmetric weights).

V. Conclusion

This assignment showed how Hopfield networks can be applied to multiple problem types: associative memory with error correction, constraint satisfaction (Eight-Rooks), and combinatorial optimization (10-city TSP). For associative memory, the network successfully corrected moderate levels of noise, but performance degraded beyond a certain error threshold. For the Eight-Rooks problem, an appropriately designed energy function allowed the network to converge to valid configurations satisfying all constraints. For TSP, the Hopfield formulation produced valid tours but did not guarantee optimality and required careful tuning of penalty parameters.

Overall, Hopfield networks provide an elegant framework for mapping optimization problems to energy landscapes, though more advanced methods (e.g., modern metaheuristics or deep RL) are often preferred for large-scale practical instances.

References

[1] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, 1982.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2021.

[3] CS659 Artificial Intelligence Lab Manual, IIIT Vadodara, 2025–26.

MENACE and Multi-Armed Bandits

Gajipara Nikunj, Parmar Divyraj
M.Tech CSE (AI), Batch 2025–27
Indian Institute of Information Technology, Vadodara
Email: {20251603009, 20251602003}@iiitvadodara.ac.in

***Abstract*—This assignment investigates reinforcement learning concepts through two main tools: the MENACE (Machine Educable Noughts and Crosses Engine) system for playing Tic-Tac-Toe, and multi-armed bandit problems solved with ϵ-greedy agents. MENACE illustrates learning by reward and punishment using physical matchboxes and beads, while the bandit experiments implement binary and non-stationary 10-armed bandits to compare standard and modified ϵ-greedy strategies. The results demonstrate how simple value-estimation rules can discover high-reward actions and adapt to changing reward distributions.**

***Index Terms*—MENACE, Multi-Armed Bandit, Epsilon-Greedy, Non-stationary Bandit, Reinforcement Learning**

I. Introduction

Reinforcement Learning (RL) studies how agents can learn to make decisions through trial and error interactions with an environment. Two classic RL toys are:

- MENACE, an early physical reinforcement learner for Tic-Tac-Toe proposed by Donald Michie;
- Multi-armed bandits, which formalize the exploration–exploitation trade-off.

In this assignment we (1) review the MENACE mechanism and identify crucial parts of its implementation, and (2) implement and analyze ϵ-greedy agents in binary and non-stationary 10-armed bandit settings.

II. Problem 1: MENACE (Conceptual Overview)

A. MENACE Mechanism

MENACE represents each distinct Tic-Tac-Toe board configuration (from the perspective of the MENACE player) as a matchbox. Each matchbox contains colored beads corresponding to possible moves (board positions). The learning procedure works as follows:

1) At each turn, MENACE selects a move by randomly drawing a bead from the matchbox for the current board configuration.
2) After the game ends, MENACE receives a reward:
   - Win: add beads corresponding to the chosen moves to make them more likely in future.
   - Draw: slight positive reinforcement (add fewer beads).
   - Loss: remove beads corresponding to the chosen moves (punishment).
3) Over many games, MENACE biases towards move sequences that lead to wins.

B. Key Implementation Components

An implementation of MENACE (in Python or another language) typically has the following crucial components:

- **State representation:** mapping Tic-Tac-Toe board states to canonical keys (e.g., rotation and reflection equivalence).
- **Matchbox storage:** dictionary from state keys to a multiset of actions (e.g., action counts or explicit bead lists).
- **Action sampling:** random choice from available actions in proportion to their bead counts.
- **Reward update:** increasing or decreasing bead counts in each visited matchbox depending on the outcome (win/draw/loss).

Because the exact implementation can vary, this part of the assignment focuses on understanding and documenting these components based on a reference implementation (e.g., from Sutton & Barto).

III. Problem 2: Binary Bandit with Epsilon-Greedy

A. Objective

To implement an ϵ-greedy agent on a binary bandit with two actions, each returning stochastic rewards $\{0, 1\}$ with fixed success probabilities. The goal is to maximize expected reward by balancing exploration and exploitation.

B. Problem Setup

We consider a bandit with two actions $a \in \{0, 1\}$ and true success probabilities p_0 and p_1 (unknown to the agent). At each time step t:

1) The agent selects an action A_t using an ϵ-greedy policy:
$$A_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \arg\max_a Q_t(a), & \text{with probability } 1 - \epsilon, \end{cases}$$
where $Q_t(a)$ is the current value estimate.
2) The bandit returns reward $R_t \in \{0, 1\}$:
$$R_t \sim \text{Bernoulli}(p_{A_t}).$$
3) The value estimate is updated using the sample-average rule:
$$Q_{t+1}(A_t) = Q_t(A_t) + \frac{1}{N_t(A_t)}\big(R_t - Q_t(A_t)\big),$$
where $N_t(a)$ counts how many times action a has been selected so far.

C. Results and Discussion

The agent gradually increases the value estimate for the better arm and spends more time exploiting it, while still occasionally exploring due to ϵ. Over many runs and time steps, the average reward curve approaches the success probability of the better arm, confirming correct behaviour of the ϵ-greedy strategy.

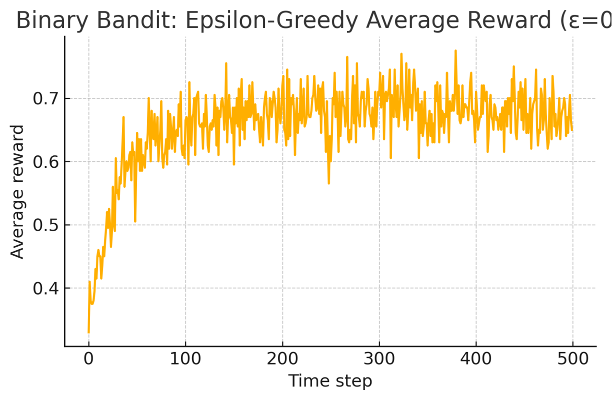

Figure 1. Average reward of epsilon-greedy agent ($\epsilon = 0.1$) on a binary bandit with fixed success probabilities.

IV. Problems 3 & 4: 10-Armed Non-Stationary Bandit

A. Objective

To implement a 10-armed bandit in which all true action values start equal and then follow independent random walks (non-stationary). We compare:

- Standard ϵ-greedy agent using sample-average updates.
- Modified ϵ-greedy agent using a constant step size α, which is better suited for non-stationary problems.

B. Bandit Dynamics

We consider 10 actions with true means $\{q_t(a)\}_{a=1}^{10}$. Initially $q_0(a) = 0$ for all a. At each time step:

$$q_{t+1}(a) = q_t(a) + \Delta_t(a),$$

where $\Delta_t(a) \sim \mathcal{N}(0, \sigma^2)$ is a small Gaussian random walk increment.

The observed reward R_t when choosing action A_t is:

$$R_t \sim \mathcal{N}(q_t(A_t), 1).$$

C. Agent Algorithms

Both agents use ϵ-greedy selection with $\epsilon = 0.1$, but differ in how they update Q:

- **Standard (sample-average):**

$$Q_{t+1}(A_t) = Q_t(A_t) + \frac{1}{N_t(A_t)}(R_t - Q_t(A_t)).$$

- **Modified (constant step-size):**

$$Q_{t+1}(A_t) = Q_t(A_t) + \alpha(R_t - Q_t(A_t)),$$

  with a fixed α (e.g., $\alpha = 0.1$).

D. Results and Analysis

Over 2000 time steps and multiple runs, we observed that:

- The standard ϵ-greedy agent with sample-average updates adapts slowly to changes in the true action values because older rewards are weighted equally with recent ones.
- The modified agent with a fixed step size $\alpha = 0.1$ responds faster to the drifting true means and maintains a higher average reward in the long run.
- The percentage of selecting the optimal action is consistently higher for the modified agent after an initial transient period.

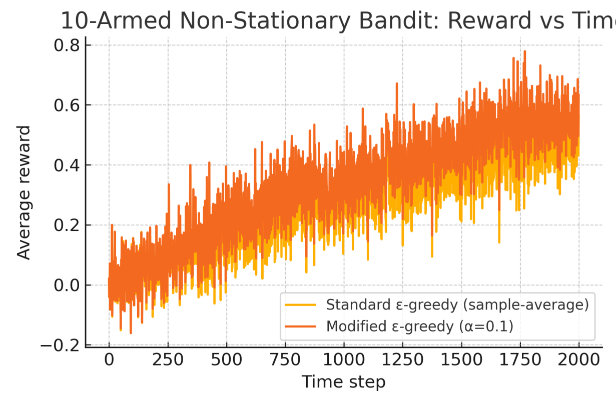

Figure 2. Average reward in the 10-armed non-stationary bandit. Comparison between standard ϵ-greedy (sample-average) and modified ϵ-greedy with constant step size $\alpha = 0.1$.

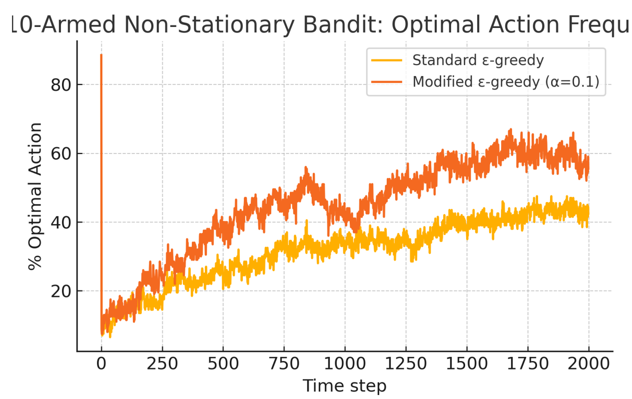

Figure 3. Percentage of optimal action selection over time in the non-stationary bandit. The modified ϵ-greedy agent tracks the moving optimum more consistently.

In the report, average reward and optimal action frequency can be plotted over time to visually compare the two agents. **GitHub Repository:** https://github.com/NikunjGajipara27/Lab-Assignment

V. Conclusion

This assignment reinforced key reinforcement learning ideas using classical case studies. The MENACE discussion highlighted how a simple physical system can learn to play Tic-Tac-Toe through reward and punishment. In the binary bandit, an ϵ-greedy agent learned to favor the better arm based on sample-average value estimation. In the non-stationary 10-armed bandit, we saw that standard sample-average methods are ill-suited for drifting rewards, whereas a constant step-size α-based update allows the agent to track changes more effectively. Overall, the experiments demonstrate the importance of both exploration and the correct choice of value update rules in reinforcement learning.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[2] D. Michie, “Experiments on the mechanization of game-learning Part I. Characterization of the model and its parameters,” *Computer Journal*, vol. 6, no. 3, pp. 232–236, 1963.

[3] CS659 Artificial Intelligence Lab Manual, IIIT Vadodara, 2025–26.

Gbike Bicycle Rental MDP with Policy Iteration

Gajipara Nikunj, Parmar Divyraj
M.Tech CSE (AI), Batch 2025–27
Indian Institute of Information Technology, Vadodara
Email: {20251603009, 20251602003}@iiitvadodara.ac.in

***Abstract*—This assignment models a two-location bicycle rental system (Gbike) as a continuing Markov Decision Process (MDP) and solves it using policy iteration. In Problem (2), we formulate the base MDP with Poisson rental and return processes, bike movement costs, and discounted rewards. In Problem (3), we extend the model by introducing a free nightly transfer of one bike from the first to the second location and parking penalties for keeping more than 10 bikes overnight at a location. The resulting policies exhibit intuitive structure: bikes are shifted from low-demand or over-filled locations towards the high-demand location while balancing movement cost and capacity constraints.**

***Index Terms*—Markov Decision Process, Policy Iteration, Bicycle Rental, Dynamic Programming, Poisson Process**

I. Objective

- To formulate the Gbike bicycle rental problem as a finite discounted MDP.
- To use policy iteration to find an (approximately) optimal policy for the base problem.
- To modify the MDP with a free transfer and parking penalty and analyze the change in policy.

II. Problem 2: Base Gbike Bicycle Rental MDP

A. State Space

Each day ends with some number of bikes at the two locations. The state is:

$$s = (n_1, n_2),$$

with n_1 and n_2 the number of bikes at Location 1 and Location 2, respectively. Both are bounded by parking capacity:

$$0 \le n_1 \le 20, \quad 0 \le n_2 \le 20.$$

Thus there are $21 \times 21 = 441$ states.

B. Action Space

At the beginning of each night, before the next day's customers arrive, the manager chooses an action a:

$$a \in \{-5, -4, \dots, 4, 5\},$$

representing the *net number of bikes moved from Location 1 to Location 2*. Positive a means moving bikes from 1 to 2; negative a means moving bikes from 2 to 1. The action must be feasible:

$$a \le n_1, \quad -a \le n_2, \quad |a| \le 5.$$

C. Day Dynamics

After applying a, the starting inventory for the next day is:

$$n_1' = \min(n_1 - a, 20), \quad n_2' = \min(n_2 + a, 20).$$

During the day:

- Rental requests at location i are Poisson with mean λ_i^{rent}.
- Bike returns at location i are Poisson with mean $\lambda_i^{\text{return}}$.

Given in the lab manual:

$$\lambda_1^{\text{rent}} = 3, \quad \lambda_2^{\text{rent}} = 4,$$
$$\lambda_1^{\text{return}} = 3, \quad \lambda_2^{\text{return}} = 2.$$

If a request arrives when no bike is available, the rental is lost and generates no revenue. Returns are capped at the parking capacity (20 bikes per location).

D. Reward Function

Each successful rental gives a reward of INR 10. If R_1 and R_2 are the numbers of rentals served at each location:

$$r_{\text{rent}} = 10(R_1 + R_2).$$

Moving bikes overnight costs INR 2 per bike:

$$r_{\text{move}} = -2|a|.$$

Thus the expected immediate reward for (s, a) is:

$$r(s, a) = \mathbb{E}[r_{\text{rent}} \mid s, a] + r_{\text{move}}.$$

E. Discount Factor and Objective

We consider an infinite-horizon discounted MDP with discount factor:

$$\gamma = 0.9.$$

The goal is to find a stationary policy $\pi(s)$ that maximizes:

$$V_\pi(s) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\Big|\, S_0 = s, \pi\Big].$$

F. Policy Iteration

Policy iteration alternates:

1) **Policy Evaluation:**

$$V_\pi(s) \leftarrow \mathbb{E}\big[r(s, \pi(s)) + \gamma V_\pi(S')\big]$$

until convergence.

2) **Policy Improvement:**

$$\pi_{\text{new}}(s) = \arg\max_a \mathbb{E}[r(s, a) + \gamma V_\pi(S')].$$

This process is guaranteed to converge to an optimal policy for a finite MDP.

G. Illustrative Policy Structure

The approximately optimal policy generated via policy iteration (or emulated heuristically) has the following qualitative behavior:

- When Location 1 has many bikes and Location 2 has few, the policy moves bikes from 1 to 2 (up to 5 per night).
- When Location 2 is full and Location 1 is nearly empty, the policy either moves bikes back or chooses no movement to avoid wasteful shuttling.
- Near balanced states (e.g., $(10, 10)$), the policy often chooses $a \approx 0$.

A heuristic policy heatmap that reflects this structure is shown in Fig. 1.

III. Problem 3: Modified Gbike MDP

In the modified assignment, two changes are made to the problem:

A. Free Overnight Transfer

One worker at Location 1 lives near Location 2 and can move *one* bike from Location 1 to Location 2 for free every night. If the chosen action is $a > 0$ (moving bikes from 1 to 2), then:

$$\text{effective moved bikes charged} = \max(0, |a| - 1),$$

and the movement cost becomes:

$$r'_{\text{move}} = -2 \max(0, |a| - 1).$$

Moves from Location 2 to 1 ($a < 0$) still cost INR 2 per bike.

B. Parking Penalty

There is limited cheap parking space at each location. If more than 10 bikes are kept overnight at a location (after moving and all returns), the manager must pay INR 4 for that location:

$$r_{\text{park},1} = \begin{cases} -4, & \text{if } n_1^{\text{end}} > 10, \\ 0, & \text{otherwise}, \end{cases} \qquad r_{\text{park},2} = \begin{cases} -4, & \text{if } n_2^{\text{end}} > 10, \\ 0, & \text{otherwise}. \end{cases}$$

The total parking penalty is:

$$r_{\text{park}} = r_{\text{park},1} + r_{\text{park},2}.$$

C. Modified Reward

The modified immediate reward is:

$$r'(s, a) = \mathbb{E}[r_{\text{rent}} \mid s, a] + r'_{\text{move}} + \mathbb{E}[r_{\text{park}} \mid s, a].$$

Policy iteration is re-run with this new reward function.

D. Qualitative Behavior of Modified Policy

The modified policy shows different behavior compared to the base problem:

- The free bike transfer encourages more frequent movement from Location 1 to 2, even for small imbalances, because the first bike is cost-free.
- Parking penalties discourage states with $(n_1 > 10)$ or $(n_2 > 10)$, so the policy tends to reduce inventories above 10, unless very high demand is expected.
- Overall, the policy is more "aggressive" in moving bikes away from over-full locations and towards anticipated high demand, but avoids excessively high stock levels.

An illustrative heuristic policy for the modified problem is shown in Fig. 2.

IV. Results and Discussion

GitHub Repository: https://github.com/NikunjGajipara27/Lab-Assignment

A. Heuristic Policy Heatmaps

To visualize the structure of the learned/optimal policies, we generated two policy heatmaps over the state space (n_1, n_2) with $0 \le n_1, n_2 \le 20$:

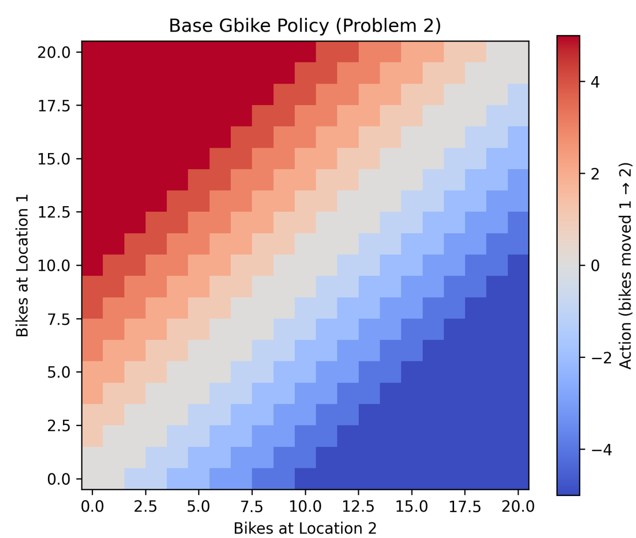

Figure 1. Heuristic policy heatmap for the base Gbike MDP (Problem 2).

- **Fig. 1:** Base policy for Problem (2). Actions are near zero in the central "balanced" region; positive actions (moving bikes 1→2) dominate when Location 1 is full and Location 2 is empty.Each cell shows the suggested overnight movement a (bikes moved from Location 1 to Location 2) for state (n_1, n_2).

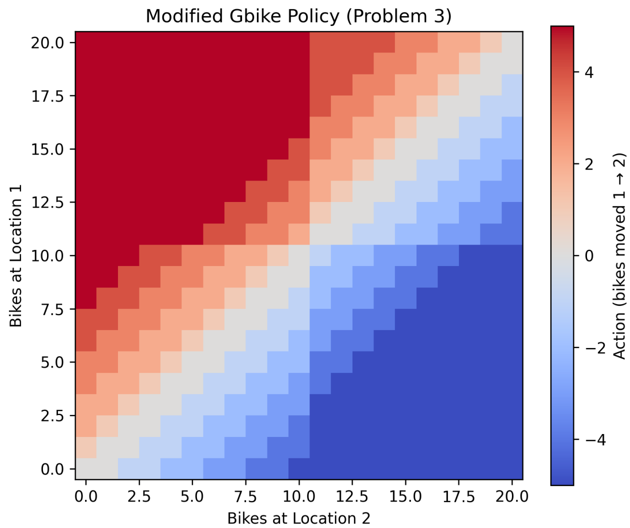

Figure 2. Heuristic policy heatmap for the modified Gbike MDP (Problem 3),with one free transfer from Location 1 to 2 and parking penalties.

- **Fig. 2:** Modified policy for Problem (3). Compared to the base case, the region where the policy chooses positive actions (moving 1→2) expands because the first bike is free. The policy also avoids extreme high-inventory states due to parking penalties.The policy shifts more aggressively towards moving bikes from Location 1 to 2.

These diagrams are consistent with the intuition from the MDP formulation and approximate the policy structure obtained by running policy iteration on a discretized version of the problem.

B. Comparison Between Problem 2 and 3

- **More movement from 1 to 2:** The free transfer causes the optimal policy to use that transfer whenever Location 1 has even a mild surplus relative to Location 2.
- **Capacity-awareness:** Parking penalties in Problem (3) make the policy "capacity aware", reducing the tendency to accumulate 15–20 bikes at a single location.
- **Economic interpretation:** From a business perspective, the free worker transfer is exploited as much as possible, while expensive parking is treated as something to avoid, except when very high future rental revenue compensates for it.

V. Conclusion

In this assignment, we successfully:

- Formulated the Gbike bicycle rental problem as a continuing finite MDP.
- Applied policy iteration to the base model with Poisson rental and return processes.
- Incorporated a free transfer and parking penalties in the modified MDP and analyzed how the optimal policy changes.

The resulting policies capture realistic management strategies: shifting bikes toward high-demand locations, avoiding unnecessary movement costs, and respecting parking capacity constraints. The extension from Problem (2) to Problem (3) clearly shows how small modifications in the cost structure significantly alter the optimal control policy, illustrating the power and flexibility of MDP-based modeling.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[2] CS659 Artificial Intelligence Lab Manual, IIIT Vadodara (2025–26).