

|  |  |  |
| --- | --- | --- |
| **Reinforcement Learning** | | |
| Lab Manual | | |
| **Department of Computer Science and Engineering**  **The NorthCap University, Gurugram** | | |
|  | |
|  | |
|  | |
|  | |
|  |  | | |

**Reinforcement Learning**

**Laboratory Manual**

**CSL348**

**Ms. Neetu Singla**



Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2023-24

*Published by:*

**Department of Computer Science & Engineering**

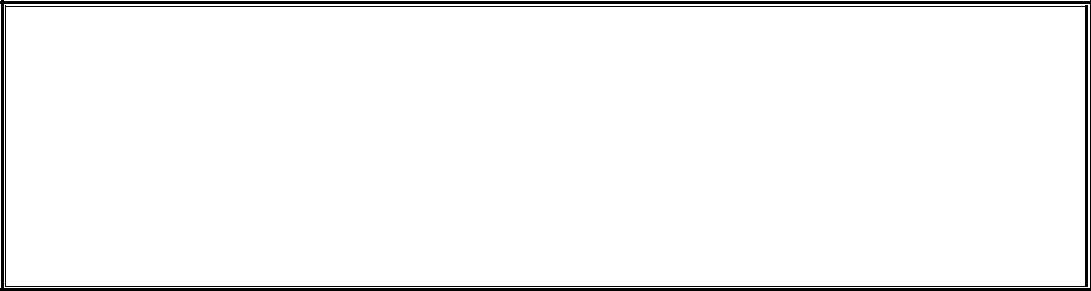
**School of Engineering and Technology**

**The NorthCap University Gurugram**

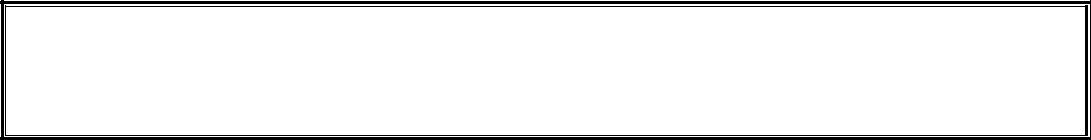
**• Laboratory Manual is for Internal Circulation only**

**© Authors**

*No part of this Practical Record Book may be reproduced, used, stored without prior permission of The NorthCap University*



Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary action. Attendance in lab classes is mandatory.



Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

**PREFACE**

Applied Computational Statistics Laboratory Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech. fourth semester students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of “teach yourself pattern” and expected that students who come with proper preparation should be able to perform the experiments without any difficulty. A brief introduction to each experiment with information about self-study material is provided. The laboratory exercises will help students to provide a hands-on each exercise that will help them to understand thoroughly. The students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for the The NorthCap University, Gurugram. Teacher’s copy of the experimental results and answer for the questions are available as sample guidelines.

We hope that lab manual would be useful to students of CSE branch and author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

**Authors**

**The NorthCap University**

**Gurugram, India**

|  |  |  |
| --- | --- | --- |
| **CONTENTS** | | |
| S.No. | **Details** | Page No. |
|  |  |  |
| 1 | **Introduction** | vi |
| 2 | **Lab Requirement** | vii |
| 3 | **General Instructions** | viii |
| 4 | **List of Experiments** | x |
| 5 | **List of Flip Experiments** | xi |
| 6 | **List of Projects** | xii |
| 7 | **Rubrics** | xiv |
| 8 | **Annexure 1 (Format of Lab Report)** | xv |
| 9 | **Annexure 2 (Format of Project Report)** | xlviii |

1. **INTRODUCTION**



That ‘learning is a continuous process’ cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus, practical makes an integral part of a learning process. ­­­­­­­­­­­­­­­­­­­­­

**COURSE OBJECTIVES:**

1. **Understand the basics of descriptive and inferential statistics and be able to apply appropriate descriptive statistical and exploratory methods to analyze datasets.**
2. **Recognize the concept & need of probability in real world. Students will understand the basics of probability, sample space, events, statistics and apply them to real life problems to determine marginal, conditional and joint probabilities.**
3. **Understand the probability mass function and distinguish between the different discrete distributions through application on real-world examples.**
4. **Understand the probability density function and distinguish between the different continuous distributions through application on real-world examples.**
5. **Identify the need for statistical hypothesis testing. Apply the appropriate hypothesis test, interpret the results and devise appropriate strategies.**
6. **Translate real world problems into probability models using Bayesian statistics.**
7. **LAB REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Requirements** | **Details** |
| **1** | **Software Requirements** | Python 3.x, Numpy, Pandas, Matplotlib, Seaborn, statistics, sci-kit learn |
| **2** | **Operating System** | Windows 7 onwards or Linux (32 or 64 bit) |
| **3** | **Hardware Requirements** | 4 GB RAM (Recommended)  2.60 GHz (Recommended) |
| **4** | **Required Bandwidth** | NA |

1. **GENERAL INSTRUCTIONS** 
   1. **General discipline in the lab**
   * Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
   * Students will not be allowed to enter late in the lab.
   * Students will not leave the class till the period is over.
   * Students should come prepared for their experiment.
   * Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
   * Students must get the connection of the hardware setup verified before switching on the power supply.
   * Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
   * Violating the above code of conduct may attract disciplinary action.
   * Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.
   1. **Attendance**

* Attendance in the lab class is compulsory.
* Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
* On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**
  1. **Preparation and Performance**
* Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self -study reference is provided on LMS.
* Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
* Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
* Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
* Refer **Annexure 1** for Lab Report Format

1. **LIST OF EXPERIMENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Title of the Experiment** | **Software**  **used** | **Unit**  **Covered** | **CO**  **Covered** | **Time**  **Required** |
|  | Implement Probability using Python | Python | 1 | CO1 | 2 Hours |
|  | Write a Python Program to compute Karl Pearson and Spearman’s Rank Correlation Coefficient | Python  (Jupyter) | 1 | CO1 | 2 hours |
|  | Write a Python program to solve the Multi-Armed Bandit problem using the Upper Confidence Bound Algorithm. Compare the reward obtained with random sampling. | Python  (Jupyter) | 2 | CO2 | 2 hours |
|  | Write a Python program to solve Multi-Armed Bandit problem using Thompson sampling. | Python  (Jupyter) | 2 | CO2 | 2 hours |
|  | Write a program to implement Q-Learning in Python. | Python  (Jupyter) | 3 | CO3 | 2 hours |
|  | Write python program to implement Markov Process. | Python  (Jupyter) | 3 | CO3 | 2 hours |
|  | Write a python program to implement policy iteration in Dynamic programming. | Python  (Jupyter) | 4 | CO4 | 2 hours |
|  | Write a python program to implement value iteration in Dynamic programming. | Python  (Jupyter) | 4 | CO4 | 2 hours |
|  | Write a Python Program to implement Monte Carlo method | Python  (Jupyter) | 5 | CO5 | 2 hours |
|  | Write a Python Program to implement TD in Reinforcement Learning | Python  (Jupyter) | 5 | CO5 | 2 hours |
|  | Implement function approximation methods | Python  (Jupyter) | 6 | CO6 | 2 hours |
|  | Implement function approximation methods. | Python  (Jupyter) | 6 | CO6 | 2 hours |
| **Value Added Experiments** | | | | | |
|  | Use RL algorithms to solve CartPole Balancing | Python  (Jupyter) | 5 | CO2,3,4,5,6 | 2 hours |
|  | Create Deep Reinforcement Learning Algorithms to play Atari games. | Python  (Jupyter) | 5 | CO2,3,4,5,6 | 2 hours |
|  | Implement Q-Learning and Markov Algorithms with Python and OpenAI | Python  (Jupyter) | 5 | CO3 | 2 hours |

1. **LIST OF FLIP EXPERIMENTS**

|  |  |  |
| --- | --- | --- |
| **Exp. No.** | **Title of the Experiment** | **Mapped CO** |
|  | Apply advanced deep RL algorithms to games  such as Minecraft | CO 1, 2,3,4,5,6 |
|  | Deploy RL algorithms using OpenAI Universe | CO1,2,3,4,5,6 |
|  | Implement basic actor-critic algorithms for  continuous control | CO1,2,3,4,5, 6 |

1. **LIST OF PROJECTS**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Project Title** | **Mapped CO** |
|  | Traffic Light Control | CO 1,2,3,4,5,6 |
|  | Robotics | CO1,2,3,4,5,6 |
|  | News Recommendation System. | CO1,2,3,4,5,6 |

1. **RUBRICS (Only for Lab components)**

|  |  |
| --- | --- |
| **Marks Distribution** | |
| **Continuous Evaluation (25 Marks)** | **Project Evaluations (20 Marks)** |
| Each experiment shall be evaluated for 5 marks and at the end of the semester proportional marks shall be awarded out of total 25. | Project shall be evaluated for 20 marks and at the end of the semester viva will be conducted related to the project. |
| **Viva and Reporting (25 Marks)**  Following is the breakup of 25 marks for each  **10 Marks**: Observation & conduct of experiment. Teacher may ask questions about experiment in mid-term viva.  **10 Marks:** Observation & conduct of experiment.  **5 Marks:** For report writing |

**Annexure 1**

**CSL348**

Lab Practical Report



Faculty name: Prof. Neetu Singla Student name: Keerti Kohli

Roll No.: 21csu260

Semester: 5th

Group: AI-3

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2022-2023

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number: KEERTI KOHLI and 21csu260** |
| **Semester /Section: 5th/AIML-B** |
| **Link to Code:** |
| **Date: 16/08/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):**  Compute correlation for two given series |
| **Outcome:** Understanding the meaning of correlation |
| **Problem Statement:** Compute Karl Pearson’s and Spearman’s Rank Correlation |
| **Background Study:** In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data. Although in the broadest sense, "correlation" may indicate any type of association, in statistics it normally refers to the degree to which a pair of variables are linearly related. |
| **Question Bank:**  1.Differentiate between correlation and causation.  Correlation describes the "association" or "relationship" between variables, while causation explains the "why" or "reason" behind that relationship. Just because two variables are correlated does not mean that one causes the other.  2. How to compute Spearman’s rank correlation coefficient for repeated ranks.  When dealing with repeated ranks, a slight modification is required to compute Spearman's rank correlation coefficient. The following steps outline the calculation:  Assign ranks to the observations of each variable. If there are ties, assign the average rank to the tied observations.  Calculate the differences between the ranks of the two variables for each pair of observations.  Square the differences obtained in step 2.  Sum the squared differences.  Calculate the number of pairs of observations (N).  If there are ties, calculate the correction factor (Σd^2t), where d^2t is the sum of the squared deviations from the mean of the tied ranks for each variable.  Calculate the Spearman's rank correlation coefficient (ρ) using the formula:  ρ = 1 - (6Σd^2 / (N(N^2 - 1)))  where Σd^2 is the sum of the squared differences from step 4 and N is the number of pairs of observations.  3. Elucidate on the graphical method for estimating correlation.  a summary of the graphical method for estimating correlation:  Create a scatterplot by plotting the data points on a two-dimensional graph.  Observe the general trend of the points. If the points tend to move in the same direction, there is a positive correlation. If they tend to move in opposite directions, there is a negative correlation.  Assess the strength of the correlation by considering how tightly clustered the points are around the trend line. The more clustered the points, the stronger the correlation.  Remember that correlation does not imply causation. Additional analysis is required to establish a causal relationship. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA white rectangular object with numbers

Description automatically generated**

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli and 21csu260** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):**  Solve the Multi-Armed Bandit Problem |
| **Outcome:** Understanding and comparing bandit strategies. |
| **Problem Statement:** Solve the muti-armed bandit problem using the Upper Confidence Bound Algorithm. Compare the reward obtained with random sampling. |
| **Background Study:** In probability theory and machine learning, the **multi-armed bandit problem** (sometimes called the ***K* or *N*-armed bandit problem** is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice. This is a classic reinforcement learning problem that exemplifies the exploration–exploitation tradeoff dilemma. |
| **Question Bank:**   1. Differentiate between exploration and exploitation.   Exploration refers to the act of trying out new actions or strategies to gather more information about the environment and learn which actions lead to the best rewards. It involves sacrificing some immediate rewards in favor of long-term gains by exploring new possibilities.  Exploitation, on the other hand, refers to the act of maximizing the expected reward by selecting the actions that have been shown to be most rewarding based on the current knowledge about the environment. It focuses on optimizing the current understanding to maximize immediate rewards.   1. Differentiate between greedy and epsilon greedy strategies for solving Multi-armed bandit problem.   Greedy strategy always selects the action that has the highest estimated average reward based on the current knowledge. It focuses on maximizing immediate rewards and does not explore new actions.  Epsilon-greedy strategy balances exploration and exploitation by introducing a probability ε (epsilon) of randomly selecting an action, even if it is not the current best estimate. This ensures that some exploration occurs, even if the focus is primarily on exploitation.  3. Explain the Upper Confidence Bound Algorithm for solving Multi-armed bandit problem.  The Upper Confidence Bound (UCB) algorithm is another popular approach to solving the multi-armed bandit problem. It balances exploration and exploitation by considering not only the average reward of each action but also the uncertainty associated with that average reward.  The UCB algorithm maintains an upper confidence bound (UCB) for each action, which is an estimate of the maximum possible average reward for that action. The UCB is calculated using the average reward and the uncertainty about the average reward, typically represented by the standard deviation. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated**

**Greedy**

**A screenshot of a computer program

Description automatically generated**

**Greedy epsilon**

**A screenshot of a computer code

Description automatically generatedA screenshot of a computer program

Description automatically generated**

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli 21csu260** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:20/9/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** Solve the Multi-Armed Bandit Problem. |
| **Outcome:** Understand Thompson sampling as a solution to the Multi-Armed Bandit Problem. |
| **Problem Statement:** Write a python program to solve the Multi-Armed Bandit Problem using Thompson Sampling. |
| **Background Study:** Thompson sampling, named after William R. Thompson, is a heuristic for choosing actions that addresses the exploration-exploitation dilemma in the multi-armed bandit problem. It consists of choosing the action that maximizes the expected reward with respect to a randomly drawn belief. |
| **Question Bank:**   1. What are beta distributions and why are they used for Thompson sampling?   1. Beta Distributions and Thompson Sampling  Beta distributions are a family of probability distributions that are commonly used to model the probability of a success or failure in a Bernoulli experiment. A Bernoulli experiment is a simple experiment with two possible outcomes, such as a coin toss (heads or tails) or a customer click (click or no click).  In the context of Thompson sampling, beta distributions are used to represent the uncertainty about the true mean reward of each action in a multi-armed bandit problem. The beta distribution has two parameters, α and β, which represent the number of successes and failures observed for each action, respectively.  2. Compare and contrast Thompson sampling with other bandit strategies.  Feature Thompson Sampling ε-greedy UCB  Exploration Probabilistic Epsilon parameter Upper confidence bound  Exploitation Samples from beta distribution Selects highest estimated mean Selects highest UCB  Performance Generally performs well in non-stationary environments Can be sensitive to initial estimates Can be conservative in exploration  3. Why is Thompson sampling referred to as Bayesian Bandits?  Thompson sampling is referred to as Bayesian Bandits because it explicitly incorporates Bayesian probability theory into its decision-making process. Bayesian probability allows for the incorporation of prior beliefs about the problem, which can improve performance in situations where there is limited data.  In Thompson sampling, the beta distribution represents the prior belief about the true mean reward for each action. As more data is collected, the beta distribution is updated to reflect the observed rewards, effectively incorporating the new information into the algorithm's belief system. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer

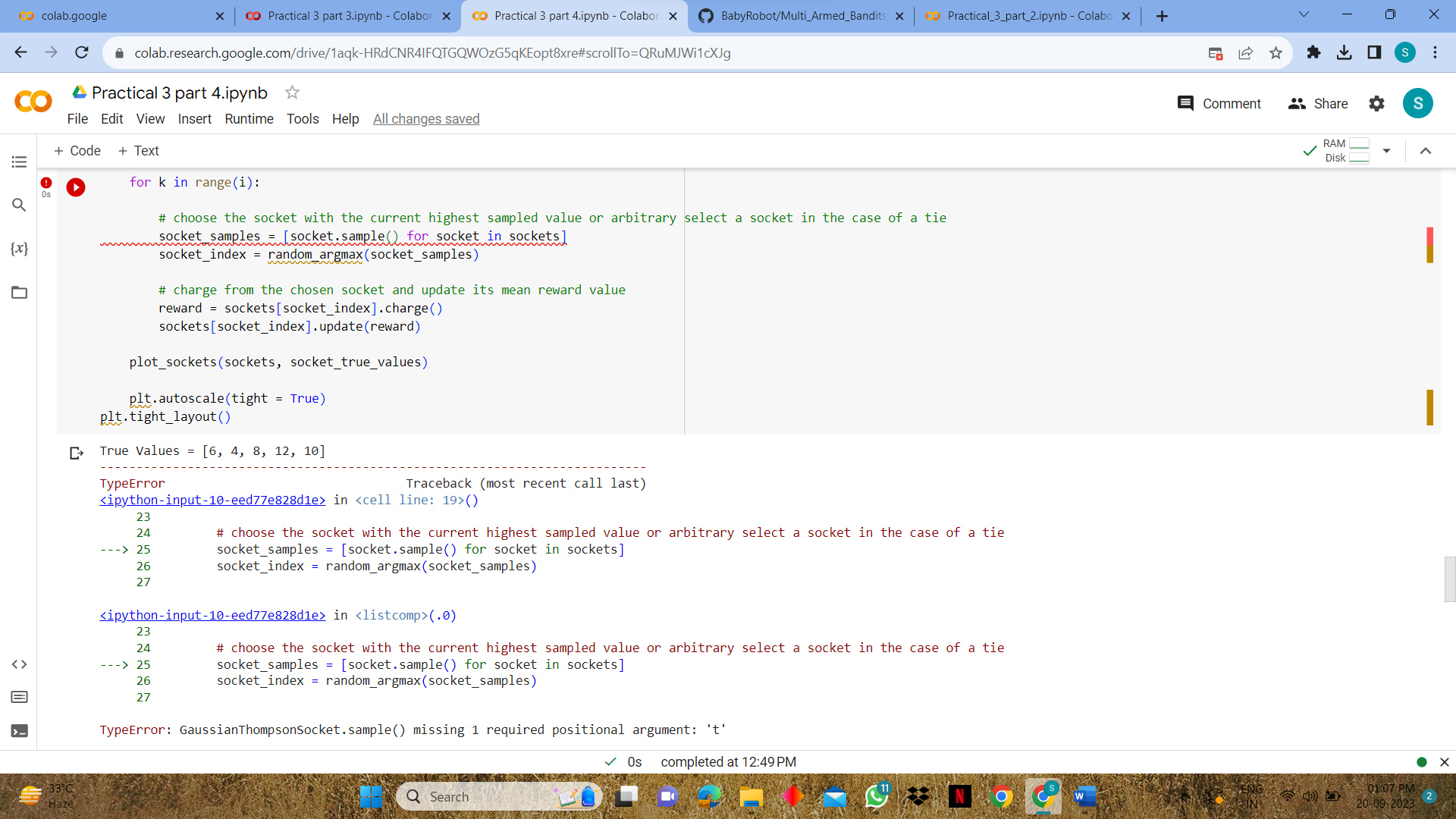
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**EXPERIMENT NO. 5**

|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli 21csu260** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** Write python program to implement Q-Learning |
| **Outcome(s):** To understand Q-Learning |
| **Problem Statement:** Implement Q-Learning using Python |
| **Background Study:** Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.  For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state. |
| **Question Bank:**   1. Differentiate between policy based and value-based reinforcement learning.   In general, policy-based algorithms are more directly focused on optimizing the agent's behavior, while value-based algorithms provide additional information about the value of different states and actions. The choice between policy-based and value-based algorithms depends on the specific problem and the desired trade-off between computational efficiency and interpretability.   1. What are off-policy and on-policy learners?   **On-policy learners only use data generated from the current policy being evaluated. This means that the algorithm learns from the actions it actually takes, which can be inefficient, especially in complex environments where the optimal policy may be difficult to explore.**  **Off-policy learners, on the other hand, can use data generated from any policy, including those different from the current policy being evaluated. This allows them to learn from a broader range of data and potentially converge to the optimal policy more efficiently.**   1. What is the Bellman equation?   **The Bellman equation is a fundamental concept in reinforcement learning that describes the relationship between the value of a state and the value of its subsequent states. It is a recursive equation that can be used to compute the optimal value function.**  **The basic form of the Bellman equation for value functions is:**  **V\*(s) = max\_a [R(s, a) + γ Σ\_s' P(s'|s, a) \* V\*(s')]**   1. What will be the effect(s) of changing the learning rate in Q-Learning?   Changing the learning rate can have several effects on Q-learning performance:  High learning rate:  Faster convergence to an approximate solution  More sensitive to noisy or inaccurate data  Potential for instability and oscillations  Low learning rate:  Slower convergence to an approximate solution  More stable and less sensitive to noisy data  May require more training iterations to reach a satisfactory solution |

**Student Work Area**

Algorithm/Flowchart/Code/Sample Outputs

**A screenshot of a computer code

Description automatically generated**

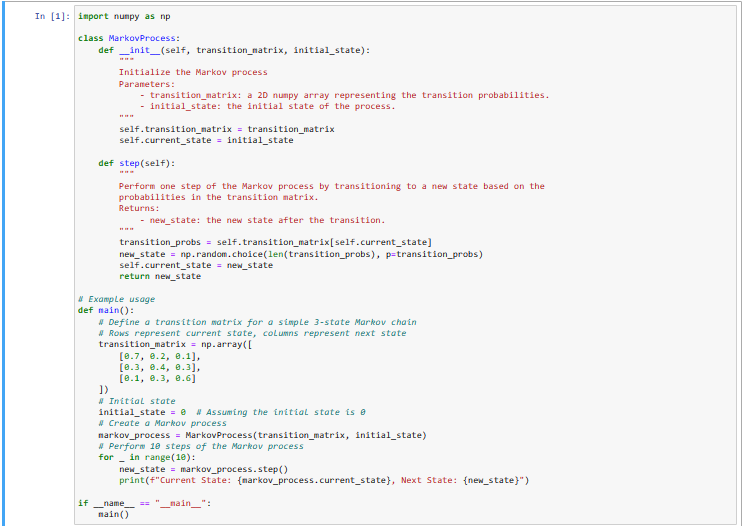
**EXPERIMENT NO. 6**

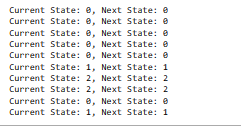
|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli 21csu260** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** Understand the Markov Process and Transition Probability Matrix |
| **Outcome(s):** Apply markov process |
| **Problem Statement:** To implement Markov Process in python |
| **Background Study:** A Markov process is a**stochastic process that satisfies the Markov property**  of memorylessness. |
| **Question Bank:**   1. What is the Markov property?   The Markov property is a fundamental concept in probability theory that states that the probability of a future event depends only on the present state, not on the events that led to that state. In simpler terms, the Markov property implies that the past is irrelevant once we know the present.   1. What are Markov chains?   **A Markov chain is a mathematical model that describes a sequence of events where the probability of each event depends only on the state of the system at the previous event. In other words, the future of the system depends only on its current state, not on its past history.**   1. What are transient, recurring and absorbing states?   **Transient states: States that the system is only likely to visit for a finite number of times before eventually moving to another state.**  **Recurring states: States that the system is likely to visit infinitely often. Recurring states can be further divided into:**  **a. Positive recurrent states: States to which the system will eventually return with probability 1.**  **b. Null recurrent states: States to which the system will eventually return with probability less than 1.**  **Absorbing states: States from which the system cannot escape once it enters them. Once the system reaches an absorbing state, it will remain there forever.** |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

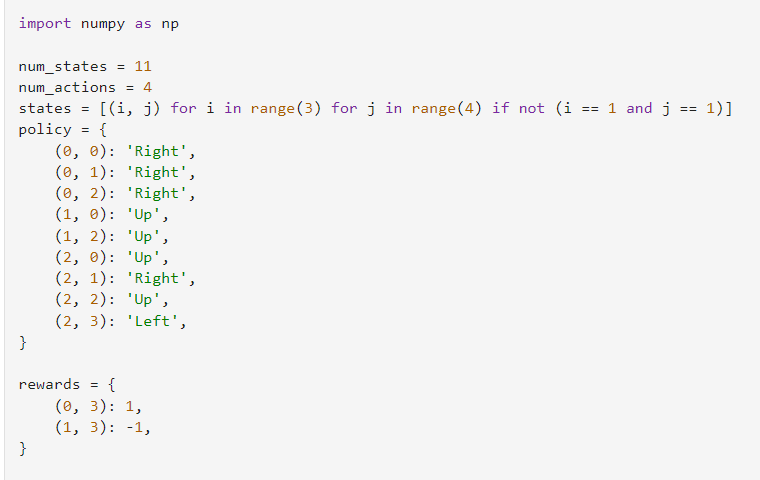
**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli and 21csu260** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:**  **4/10/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** To understand the concept of dynamic programming and policy iteration in RL. |
| **Outcome:** Understand the policy iteration algorithm. |
| **Problem Statement:** Implementation of policy iteration algorithm in dynamic programming. |
| **Background Study:** Policy Iteration is a way to find the optimal policy for given states and actions. **Policy Iteration** takes an initial **policy**, evaluates it, and then uses those values to create an improved **policy**. These steps of evaluation and improvement are then repeated on till convergence. |
| **Question Bank:**   1. What are Bellman expectation and optimality equations?   The Bellman Expectation Equation and the Bellman Optimality Equation are fundamental concepts in reinforcement learning. They describe how the value of a state or state-action pair can be decomposed into immediate rewards and the expected values of future states under a given policy or an optimal policy, respectively.  1. Bellman Expectation Equation (also known as the Bellman Expectation Backup):  The Bellman Expectation Equation expresses the value of a state under a policy as the sum of the immediate reward and the expected value of the next state, considering the policy's action selection.  For a state `s` under a policy `π`:  - Value of state `s` (Vπ(s)) is the expected return (cumulative reward) when starting from state `s` and following policy `π`.  - The Bellman Expectation Equation for state `s` is given by:    Vπ(s) = Σ [π(a|s) \* Σ [P(s' | s, a) \* [R(s, a, s') + γ \* Vπ(s')]]]  - `π(a|s)` is the probability of taking action `a` in state `s` under policy `π`.  - `P(s' | s, a)` is the probability of transitioning from state `s` to state `s'` when taking action `a`.  - `R(s, a, s')` is the immediate reward obtained when transitioning from state `s` to state `s'` by taking action `a`.  - `γ` is the discount factor, which represents the importance of future rewards.  This equation provides a way to iteratively update the value of each state under a given policy. It's used in policy evaluation algorithms like the iterative methods (e.g., Policy Iteration, Value Iteration) to estimate the values of states.  Bellman Optimality Equation:  The Bellman Optimality Equation expresses the value of a state or state-action pair under the optimal policy as the maximum expected return. It defines what it means for a policy to be optimal.  For a state `s`:  - Value of state `s` under the optimal policy (V\*(s)) is the maximum expected return when starting from state `s` and following an optimal policy.  - The Bellman Optimality Equation for state `s` is given by:    V\*(s) = max [Σ [π(a|s) \* Σ [P(s' | s, a) \* [R(s, a, s') + γ \* V\*(s')]]]]  - `max` is taken over all possible policies `π`.  - The rest of the terms have the same meaning as in the Bellman Expectation Equation.  Similarly, for a state-action pair `(s, a)`:  - Value of taking action `a` in state `s` under the optimal policy (Q\*(s, a)) is the maximum expected return when taking action `a` in state `s` and then following an optimal policy.  - The Bellman Optimality Equation for state-action pair `(s, a)` is given by:    Q\*(s, a) = Σ [P(s' | s, a) \* [R(s, a, s') + γ \* max(Q\*(s', a'))]]  - `max` is taken over all possible actions `a'` in state `s'` (the next state).  The Bellman Optimality Equation is used to find the optimal policy and the corresponding optimal value function in reinforcement learning. Solving this equation helps identify the best actions to take in each state to maximize the expected cumulative reward.   1. What is dynamic programming?   Dynamic programming is an optimization technique used to solve problems by breaking them down into smaller overlapping subproblems and efficiently reusing previously computed solutions to those subproblems. It's commonly applied to optimization problems with recursive structures, making it a powerful tool in computer science, algorithms, and operations research.   1. What is a policy?   A policy in reinforcement learning is a strategy that an agent uses to decide its actions in an environment, defining "what to do" based on the current state or state-action pair. It can be deterministic (always chooses the same action) or stochastic (chooses actions with probabilities).   1. Explain the policy evaluation and policy improvement steps in policy iteration.   1. Policy Evaluation:  - Goal: Estimate the value function for the current policy.  - Process: Repeatedly update the value function using the Bellman Expectation Equation until it stabilizes.  2. Policy Improvement:  - Goal: Improve the current policy.  - Process: For each state, select actions that maximize expected returns based on the estimated value function, making the policy "greedy."  Policy iteration alternates between these two steps until the policy becomes optimal, meaning it no longer changes for any state.   1. What do you mean by optimal policy? When is a policy optimal?   An optimal policy in the context of reinforcement learning and Markov Decision Processes (MDPs) is a policy that, when followed by an agent, maximizes the expected cumulative reward over time in the given environment. In other words, it's the best strategy or set of actions an agent can take to achieve the highest possible long-term reward.  A policy is considered optimal under the following conditions:  1. Maximizes Expected Cumulative Reward: An optimal policy ensures that, when the agent follows it, the expected total reward obtained over time is greater than or equal to the expected total reward achievable with any other policy in the same environment.  2. Satisfies the Bellman Optimality Equation: An optimal policy satisfies the Bellman Optimality Equation, which describes the optimal value of states (or state-action pairs) in terms of the maximum expected return achievable under the policy. The Bellman Optimality Equation helps identify the values of states and actions that lead to the best outcomes.  6. What is the convergence condition for the policy iteration algorithm?  In Policy Iteration, the convergence condition is met when the current policy remains unchanged during the policy improvement step. This means that for all states, the new policy generated in the iteration is identical to the current policy. When this condition is satisfied, it indicates that the algorithm has found the optimal policy because further iterations would not lead to any policy improvement. In essence, the algorithm has converged to the best possible policy given the current environment and value estimates. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screenshot of a computer code

Description automatically generatedA screen shot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generated**

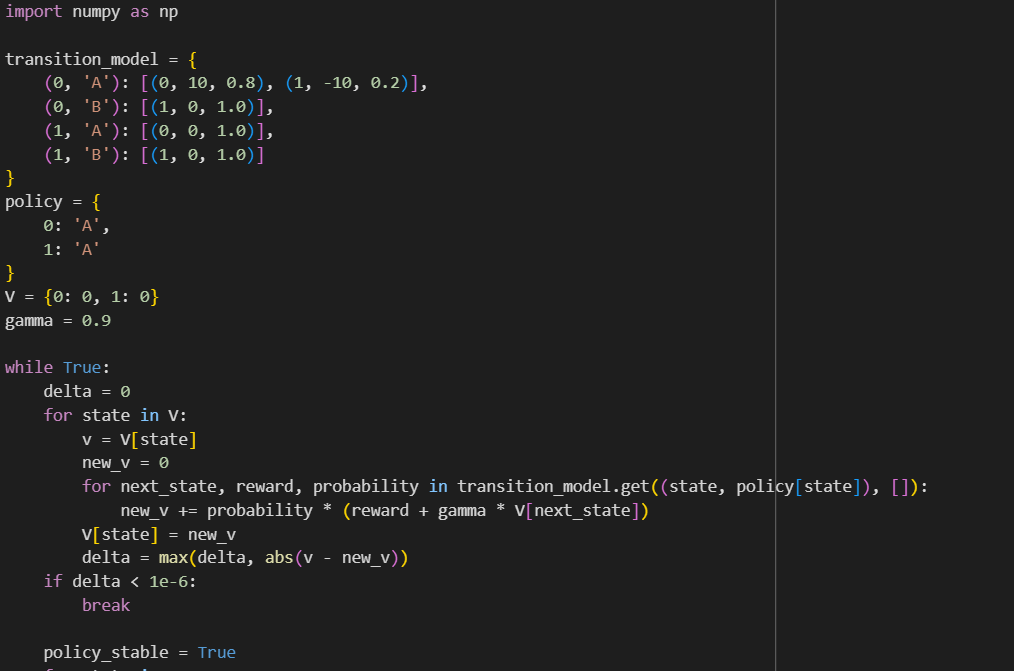
**EXPERIMENT NO. 8**

|  |
| --- |
| **Student Name and Roll Number: Keerti Kohli and 21csu260** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** To understand the concepts of dynamic programming and value iteration in RL. |
| **Outcome:** Understand value iteration algorithm. |
| **Problem Statement:** Write a python program to implement value iteration in dynamic programming. |
| **Background Study:** One of the challenges of RL is to find an optimal policy to solve our task. **Value iteration** is a method of computing an optimal policy for an MDP and its**value. In value iteration, we compute the optimal state value function by iteratively updating the state value estimate.** |
| **Question Bank:**  1. What is a Markov Decision Process.  A Markov Decision Process (MDP) is a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. It is a discrete-time stochastic control process that helps an agent make optimal choices in sequential environments.  MDPs are characterized by five elements:  States: The possible configurations of the environment at a given time step.  Actions: The available choices that the agent can take in each state.  Transition probabilities: The probabilities of transitioning from one state to another after taking a specific action.  Rewards: The immediate rewards associated with taking actions in each state.  Policy: A mapping from states to actions that defines the agent's behavior.  2. Can we obtain the optimal policy using value iteration algorithm?  Value iteration is a dynamic programming algorithm that can be used to find the optimal policy for an MDP. It iteratively updates the value function, which represents the expected cumulative reward for being in a given state and following the optimal policy from that state.  The value iteration algorithm consists of the following steps:  Initialize the value function for all states.  Repeat until convergence:  a. For each state, calculate the updated value function using the Bellman equation:  V\*(s) = max\_a [R(s, a) + γ Σ\_s' P(s'|s, a) \* V\*(s')]  where V\*(s) is the updated value function, R(s, a) is the immediate reward for taking action a in state s, γ is the discount factor, P(s'|s, a) is the transition probability of reaching state s' from state s after taking action a, and the max is over all possible actions a.  Find the optimal policy by selecting the action that maximizes the expected cumulative reward for each state:  π\*(s) = argmax\_a [R(s, a) + γ Σ\_s' P(s'|s, a) \* V\*(s')]  3. Compare and contrast policy and value iteration algorithms.  Feature Policy Iteration Value Iteration  Focus Policy improvement Value function computation  Iterative updates Policy Value function  Efficiency Small state-action spaces Large state-action spaces |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screen shot of a computer program

Description automatically generated**

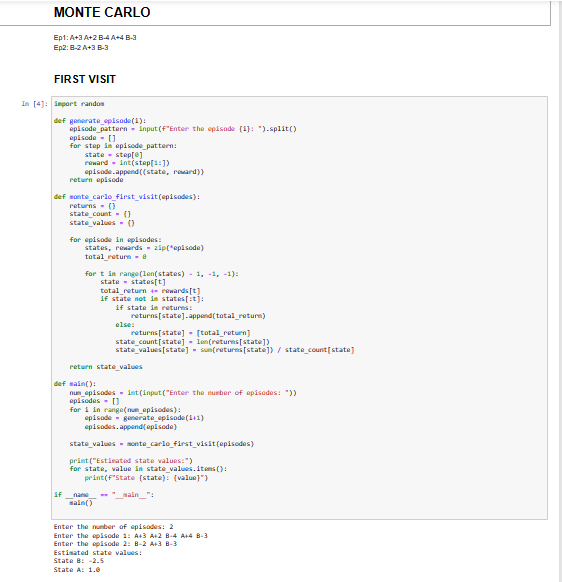
**EXPERIMENT NO. 9**

|  |
| --- |
| **Student Name and Roll Number: KEERTI KOHLI and 21csu260** |
| **Semester /Section: 5/AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** To understand Monte Carlo methods and apply them in Reinforcement Learning scenarios. |
| **Outcome:** Students will be familiarized with Monte Carlo methods. |
| **Problem Statement:** Write Python Program to implement Monte Carlo method to solve the Blackjack problem. |
| **Background Study:** Monte Carlo (MC) methods are a subset of computational algorithms that use the process of repeated random sampling to make numerical estimations of unknown parameters. They allow for the modeling of complex situations where many random variables are involved, and assessing the impact of risk. The uses of MC are incredibly wide-ranging, and have led to a number of ground-breaking discoveries in the fields of physics, game theory, and finance. There are a broad spectrum of Monte Carlo methods, but they all share the commonality that they rely on random number generation to solve deterministic problems.  The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward. One caveat is that it can only be applied to episodic MDPs. |
| **Question Bank:**  1. What are episodic MDPs?  Episodic Markov decision processes (MDPs) are a type of MDP where each episode has a definite start and end, and the agent's goal is to maximize the expected cumulative reward over the course of the episode. In episodic MDPs, the reward function is only defined for transitions between states, and the agent's state is reset to the initial state at the beginning of each episode.  2. What are model-free and model-based methods in RL?  Model-free methods do not explicitly build a model of the environment. Instead, they learn the optimal policy directly from experience, by interacting with the environment and observing the rewards received. Examples of model-free methods include Q-learning and SARSA.  Model-based methods first build a model of the environment, which can be used to simulate the effects of different actions. The agent then uses this model to plan the optimal sequence of actions to take in order to maximize its expected reward. Examples of model-based methods include policy iteration and value iteration.  3. Differentiate between on-policy and off-policy learning in RL.  On-policy methods evaluate and improve the policy that the agent is currently using. They do this by sampling episodes from the current policy and updating the policy based on the observed rewards. Examples of on-policy methods include Sarsa and policy gradient methods.  Off-policy methods evaluate or improve a policy that is different from the policy that is being used to generate the data. They do this by using the observed rewards to estimate the value of different state-action pairs, and then using this information to improve the policy. Examples of off-policy methods include Q-learning and Monte Carlo methods.  4. What are exploring starts in Monte Carlo?  Exploring starts are a technique used in Monte Carlo methods to ensure that the agent explores all parts of the state space. In Monte Carlo methods, the agent learns by sampling episodes from the current policy and updating the value function based on the observed rewards. However, this can lead to problems if the policy is too deterministic and does not explore all parts of the state space. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

**EXPERIMENT NO. 10**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):**   * To Understand Temporal Difference Learning. |
| **Outcome:** Students will understand Model-Free Temporal Difference Learning. |
| **Problem Statement:** Write python code to implement the frozen-lake problem using TD(0). |
| **Background Study:** Temporal difference (TD) learning refers to a class of**model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function**. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods. |
| **Question Bank:**  1.What is bootstrapping?  2. TD methods combine the advantages of both MC and Dynamic programming. Explain.  3. Differentiate between TD(0) and TD(lambda). |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**EXPERIMENT NO. 11**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** To understand the mathematical concept of function approximation. |
| **Outcome:** Students will be able to apply function approximation to solve RL problems. |
| **Problem Statement:** Write a python program to Implement function approximation for RL. |
| **Background Study:** In the domain of reinforcement learning, function approximation helps**finding the value of a state or an action when similar circumstances occur, whereas in computing the real values of V and Q requires a full computation and does not learn from past experience.** Furthermore function approximation saves computation time and memory space. |
| **Question Bank:**  1. Explain in detail any one function approximation method.  2. State the advantages of function approximation. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**EXPERIMENT NO. 12**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** To understand the mathematical concept of function approximation. |
| **Outcome:** To be able to implement function approximation methods to solve RL problems. |
| **Problem Statement:** Write a Python program to implement function approximation for RL. |
| **Background Study:** F**unction approximation helps finding the value of a state or an action when similar circumstances occur, whereas in computing the real values of V and Q requires a full computation and does not learn from past**experience. Furthermore, function approximation saves computation time and memory. |
| **Question Bank:**   1. Why do we need function approximation for RL? 2. What is batch reinforcement learning? 3. What are non-stationary target functions? |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**Annexure 2**

**Reinforcement Learning**

**Project Report**



Faculty name Prof. NEETU SINGLA Student name

Roll No.:

Semester:

Group:

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2021-2022

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.No** |  | **Page No.** |
| **1.** | **Project Description** |  |
| **2.** | **Problem Statement** |  |
| **3.** | **Analysis**  **3.1 Hardware Requirements**  **3.2 Software Requirements** |  |
| **4.** | **Design**  **4.1 Data/Input Output Description:**  **4.2 Algorithmic Approach / Algorithm / DFD / ER diagram/Program Steps** |  |
| **5.** | **Implementation and Testing (stage/module wise)** |  |
| **6.** | **Output (Screenshots)** |  |
| **7.** | **Conclusion and Future Scope** |  |