ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Νευροασαφής Έλεγχος και Εφαρμογές

Άσκηση 5η

Αναστάσιος Στέφανος Αναγνώστου
03119051

11 Ιουνίου 2024

Περιεχόμενα

Θέμα 1

Δεν απαντήθηκε.

Θέμα 2

Η προσομοίωση Monte Carlo για τον υπολογισμό του π γίνεται παράγοντας τυχαία ένα πλήθος σημειών με συντεταγμένες $(x, y) \in \{0, 1\}^2$ και μετρώντας το πλήθος αυτών που βρίσκονται εντός του τεταρτοκύκλιου κύκλου κέντρου $(0, 0)$ και ακτίνας 1. Δεδομένου ότι το εμβαδόν του κύκλου είναι π και το εμβαδόν του τετραγώνου είναι 1, η τιμή του π μπορεί να υπολογιστεί ως

$$
\begin{aligned}
P(\text{σημείο εντός κύκλου}) = \frac{\text{εμβαδόν τεταρτοκυκλίου}}{\text{εμβαδόν τετραγώνου}} = \frac{\frac{\pi}{4}}{1} \implies \\
\pi = 4 \cdot P(\text{σημείο εντός κύκλου}) \implies \\
\pi \approx 4 \cdot \frac{\text{πλήθος σημείων εντός κύκλου}}{\text{πλήθος σημείων}}
\end{aligned} \tag{1}
$$

Παρατίθεται κώδικας Python που υλοποιεί την προσομοίωση.

```python
import numpy as np
import random

def monte_carlo(n):
    count = 0
    for _ in range(n):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)
        if x*x + y*y <= 1:
            count += 1
    return 4 * count / n


if __name__ == '__main__':
    iterations = 10000
    print(monte_carlo(iterations))
```

Η αντίστοιχη έξοδος της προσομοίωσης για 10^4 σημεία είναι 3.1484 και για 10^6 σημεία είναι 3.141075, πράγματι πολύ κοντά στην τιμή του π.

Θέμα 3

Ερώτημα 3.1

Παρατίθεται κώδικας Python που υλοποιεί την προσομοίωση.

```python
import numpy as np
import random

def simulate_trajectory(initial_state, cost, controller, alpha=0.9, steps=10):
    """
    Simulate a trajectory of the controlled Markov chain.

    Parameters:
    - initial_state: The starting state of the system.
    - controller: The controller.
    - alpha: Discount factor for the cost.
    - max_steps: Maximum number of steps to simulate.
```

```python
    Returns:
    - states: List of states visited during the trajectory.
    - cost: Total discounted cost of the trajectory.
    """
    states = [initial_state]
    current_state = initial_state
    total_cost = 0
    for k in range(steps):
        control = controller[current_state-1]
        if control == +1:
            next_state = current_state + 1 if np.random.rand() < 0.5 else current_state
        elif control == -1:
            next_state = current_state - 1 if np.random.rand() < 0.5 else current_state
        else:
            raise ValueError("Control must be +1 or -1")

        next_state = max(1, min(10, next_state))  # Ensure the state is within bounds

        # Cost function g(x)
        g_x = cost[next_state - 1]

        total_cost += (alpha ** k) * g_x
        states.append(next_state)
        current_state = next_state

    return states, total_cost
```

Η συνάρτηση τρέχει μία προσομοίωση για K βήματα και για έναν δεδομένο ελεγκτή. Σε κάθε βήμα επιχειρεί να εφαρμόσει τον έλεγχο με πιθανότητα 1/2 να αποτύχει και να μείνει στην ίδια κατάσταση. Τέλος, επιστρέφει τις καταστάσεις από τις οποίες διήλθε και το συνολικό κόστος, βάσει ενός discount factor.

Ερώτημα 3.2

```python
def q_learning(cost, num_episodes=1000000, alpha=0.1,
               gamma=0.9, epsilon=0.05, max_iter_episode=20):
    """
    Perform Q-learning on the stochastic system.

    Parameters:
    - num_episodes: Number of episodes to run.
    - alpha: Learning rate.
    - gamma: Discount factor.
    - epsilon: Exploration rate.

    Returns:
    - Q: Learned Q-values.
    """
    Q = np.zeros((10, 2))  # 10 states and 2 actions (+1, -1)

    for _ in range(num_episodes):
        state = random.randint(0, 9)  # Start from a random state
```

```
    for _ in range(max_iter_episode):
        if random.uniform(0, 1) < epsilon:
            action = random.choice([0, 1])  # Explore
        else:
            action = np.argmin(Q[state])  # Exploit
        flag = random.uniform(0, 1) < 0.5
        next_state = state + 1 if action == 0 and flag else state
        next_state = state - 1 if action == 1 and flag else state
        next_state = max(0, min(9, next_state))

        reward = cost[next_state]

        best_next_action = np.argmax(Q[next_state])
        td_target = reward + gamma * Q[next_state, best_next_action]
        td_delta = td_target - Q[state, action]
        Q[state, action] += alpha * td_delta
        state = next_state
return Q
```