****

**Московский Государственный Технический Университет имени Н.Э.Баумана**

**Факультет Информатика и системы управления**

**Кафедра ИУ-5**

**«Системы обработки информации и управления»**

**Отчёт по Рубежному Контролю No 2**

**Методы обработки данных**

Выполнили стуленты группы ИУ5И

Шэнь Цюцзе 22M

**Москва 2022г.**

### Цель лабораторной работы

 ознакомление с базовыми методами обучения с подкреплением на основе временных различий.

### Задание

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

* SARSA
* Q-обучение
* Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки [Gym](https://www.gymlibrary.dev/) (или аналогичной библиотеки).

### Программа

import numpy as np

import gym

import random

import matplotlib.pyplot as plt

env = gym.make("Taxi-v3")

env.reset()

env.render()

action\_size = env.action\_space.n

print("Action size ", action\_size)

state\_size = env.observation\_space.n

print("State size ", state\_size)

qtable = np.zeros((state\_size, action\_size))

print(qtable)

total\_episodes = 5000  # Total episodes

total\_test\_episodes = 100  # Total test episodes

max\_steps = 99  # Max steps per episode

learning\_rate = 0.7  # Learning rate

gamma = 0.618  # Discounting rate

# Exploration parameters

epsilon = 1.0  # Exploration rate

max\_epsilon = 1.0  # Exploration probability at start

min\_epsilon = 0.01  # Minimum exploration probability

decay\_rate = 0.01  # Exponential decay rate for exploration prob

# Tracking metrics

rewards\_per\_episode = []  # List to store rewards per episode

# 2 For life or until learning is stopped

for episode in range(total\_episodes):

    # Reset the environment

    state = env.reset()

    step = 0

    done = False

    total\_rewards = 0  # Total rewards accumulated in the episode

    for step in range(max\_steps):

        # 3. Choose an action a in the current world state (s)

        ## First we randomize a number

        exp\_exp\_tradeoff = random.uniform(0, 1)

        ## If this number > greater than epsilon --> exploitation (taking the biggest Q value for this state)

        if exp\_exp\_tradeoff > epsilon:

            action = np.argmax(qtable[state, :])

        # Else doing a random choice --> exploration

        else:

            action = env.action\_space.sample()

        # Take the action (a) and observe the outcome state(s') and reward (r)

        new\_state, reward, done, info = env.step(action)

        # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma \* max Q(s',a') - Q(s,a)]

        qtable[state, action] = qtable[state, action] + learning\_rate \* (

                    reward + gamma \* np.max(qtable[new\_state, :]) - qtable[state, action])

        total\_rewards += reward  # Accumulate the rewards

        # Our new state is state

        state = new\_state

        # If done: finish episode

        if done == True:

            break

    # Reduce epsilon (because we need less and less exploration)

    epsilon = min\_epsilon + (max\_epsilon - min\_epsilon) \* np.exp(-decay\_rate \* episode)

    rewards\_per\_episode.append(total\_rewards)  # Append the total rewards for this episode to the list

    # Print progress

    if (episode + 1) % 1000 == 0:

        print(f"Episode: {episode+1}/{total\_episodes}")

env.reset()

test\_rewards = []

for episode in range(total\_test\_episodes):

    state = env.reset()

    step = 0

    done = False

    total\_rewards = 0

    for step in range(max\_steps):

        # Take the action (index) that has the maximum expected future reward given that state

        action = np.argmax(qtable[state, :])

        new\_state, reward, done, info = env.step(action)

        total\_rewards += reward

        if done:

            test\_rewards.append(total\_rewards)

            break

        state = new\_state

# Calculate and print the average reward per test episode

avg\_test\_reward = np.mean(test\_rewards)

print("Average test reward:", avg\_test\_reward)

# Plot rewards per episode

plt.plot(rewards\_per\_episode)

plt.xlabel("Episode")

plt.ylabel("Total Reward")

plt.title("Total Reward per Episode")

plt.show()

### результат

