دانشکده مهندسی برق

یادگیری تقویتی در کنترل

تمرین جامع

استاد: دکتر سعید شمقدری

دانشجو: سیده ستاره خسروی

زمستان ۱۴۰۳

چکیده

در این تمرین به مباحث یادگیری تقویتی با رویکرد کامپیوتری و پیاده سازی در محیط Gymnasium پرداخته خواهد شد، بدین منظور ۳ الگوریتم در محیط پاندول معکوس آن توسعه داده می‌شود و فرایند تست و آموزش نیز در قالب یک اپلیکیشن توسعه داده می‌شود.

در بخش دوم به کاربرد یادگیری تقویتی در کنترل بهینه پرداخته می‌شود، و توسعه فیدبک حالت بهینه برای پایدارسازی و تنظیم یک سیستم ناپایدار پیوسته پیگیری می‌شود. همچنین برای این سیستم ردیاب خطی کوادراتیک توسعه داده می‌شود.

واژه‌های کلیدی: یادگیری تقویتی، Gymnasium، LQR، LQT، Sarsa، Q Learning، Monte Carlo

فهرست مطالب

عنوان صفحه

فهرست تصاویر و نمودارها

عنوان صفحه

فصل ۱: یادگیری تقویتی در Gymnasium

۱.۱ مقدمه

در این بخش به سوال اول تمرین جامع به طور کامل پاسخ داده می‌شود، عملکرد مدل‌ها تحلیل می‌گردد، همچنین عملکرد مدل‌ها در فایل‌های ویدیویی پیوست برای مشاهده قرار داده شده است.

توجه!

برای مشاهده بدون مشکل ویدیوها، از Player هایی مانند KMPlayer، PotPlayer و ... استفاده شود. از Windows Media Player استفاده نکنید.

صورت سوال:

۱- در این سوال می‌خواهیم مسئله کنترل پاندول معکوس را با استفاده از الگوریتم‌های RL حل کنیم. به‌منظور شبیه‌سازی محیط، از کتابخانه Gymnasium و محیط cartpole-v1 استفاده می‌کنیم. هدف این است که عاملی را آموزش دهیم تا بتواند پاندول را به‌طور عمودی نگه دارد. در نهایت، با مقایسه عملکرد روش‌های زیر از نظر زمان همگرایی، دقت همگرایی، پایداری و...،تحلیل خود را نسبت به آن‌ها ارائه دهید.

الف) عامل را با استفاده از روش Monte-Carlo آموزش دهید.
ب) عامل را با استفاده از روش Q-Learning آموزش دهید.
پ) عامل را با روش SARSA آموزش دهید.

*توجه کنید که استفاده از فریمورک‌های آماده RL برای اجرای الگوریتم‌ها مجاز نبوده و شما باید الگوریتم را به‌طور کامل پیاده‌سازی نمایید. به‌منظور راحتی در مقایسه خروجی الگوریتم‌های مختلف، می‌توانید از فضای Jupyter Notebook برای نوشتن و اجرای کدهای خود استفاده کنید. در گزارش نهایی لازم است عملکرد هر کد به‌طور کامل توضیح داده شده و با کامنت‌گذاری در داخل کد، عملکرد بخش‌های مختلف آن مشخص شود. (عدم رعایت نکات گفته شده موجب کسر نمره خواهد شد).

۱.۲ روش مونت کارلو

در این بخش به قسمت الف سوال اول پاسخ داده می‌شود.

به منظور پیاده سازی این بخش، کدی به نام CartePoleOnMonteCarlo.py توسعه داده شد. در این کد روش On Policy Monte Carlo پیاده سازی شده است. قدم به قدم اجزای کد را توضیح می‌دهیم.

ابتدا مطابق تصویر زیر کتابخانه‌های مورد نیاز برای پیاده سازی الگوریتم را فراخوانی می‌کنیم.

```python
# import libraries
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import argparse
import pickle
import time
```

سپس با استفاده از کتابخانه argparse، تابعی نوشته شد، که کاربر بتواند کد را با پاس دادن پارامترهای مهم آن در ترمینال سیستم خود اجرا کند.

```python
def parse_args():
    parser = argparse.ArgumentParser(description="CartePole Training")
    parser.add_argument('--train', type=str, required=True, help="Train or test")
    parser.add_argument('--render', type=str, required=True, help="yes or no")
    parser.add_argument('--episodes', type=int, required=True, default=15000, help="Number of episodes")
    parser.add_argument('--qtable', type=str, required=False, help="If you want to test, parse the qtable path")
    parser.add_argument('--learning_rate', type=float, required=False, default=0.01, help="Learning rate for training process")
    parser.add_argument('--discount_factor', type=float, required=False, default=0.8, help="Discount factor for training process")
    parser.add_argument('--epsilon', type=float, required=False, default=0.7, help="Epsilon factor")
    parser.add_argument('--epsilon_decay', type=float, required=False, default=0.00001, help="Epsilon decay factor")
    return parser.parse_args()
```

جزئیات پارامترها به شرح زیر است:

۱. Train: انتخاب می‌کنید که فرآیند تست انجام شود یا آموزش.

۲. Render: با استفاده از این پارامتر مشخص می‌کنید که در هنگام آموزش و یا تست رندر انجام شود یا خیر.

۳. Episodes: با استفاده از این پارامتر تعداد اپیزودهای لازم توسط کاربر مشخص می‌شود.

۴. Qtable: با استفاده از این پارامتر، مسیر Q Table ذخیره شده برای الگوریتم Monte Carlo به کد پاس داده می‌شود که در هنگام تست از آن استفاده می‌گردد.

۵. Learning_rate: با استفاده از این پارامتر نرخ یادگیری مشخص می‌گردد، که مقدار پیش‌فرض آن ۰.۰۱ لحاظ شده است.

۶. با استفاده از پارامتر discount_factor نرخ تخفیف تنظیم می‌شود. مقدار پیش‌فرض آن برابر با ۰.۸ است.

۷. با دو پارامتر نهایی نیز مقدار epsilon و نرخ کاهش آن تنظیم می‌شود. که مقادیر پیش‌فرش آن‌ها به ترتیب ۱ و ۰.۰۰۰۰۱ است.

سپس تابع OnMonteCarlo برای کدنویسی فرایند آموزش و تست توسعه داده شد. مطابق کد زیر، ابتدا با استفاده از آرگومان‌هایی که کاربر به سیستم می‌دهد، فرایند اصلی که آموزش باشد یا تست مشخص می‌شود، سپس مشخص می‌گردد که رندر صورت پذیرد یا خیر.

در نهایت محیط پاندول معکوس ساخته می‌شود.

```
# def main function
def OnMonteCarlo(args):
    if args.train.lower() == 'train':
        is_training = True
        print("Training Started...")
    else:
        is_training = False
        print("Testing Started...")

    if args.render.lower() == 'yes':
        render = True
    else:
        render = False
    # creating the environment
    env = gym.make('CartPole-v1', render_mode='human' if render else None)
```

محیط CartePole محیطی اســـت با فضـــای حالت (state یا Observation) پیوســـته با که دارای ۴ پارامتر است. این فضا شامل اطلاعاتی درخصوص پوزیشن و سرعت ارابه، زاویه میله و سرعت زاویه‌ای آن است. با توجه به اینکه این فضا پیوسته است لازم است برای پیاده سازی الگوریتم‌ها گسسته شود. با استفاده از کد زیر فضای حالت سیستم را گسسته می‌کنیم.

```
# convert the continuous state space to discrete state space
pos_space = np.linspace(-2.4, 2.4, 10)
vel_space = np.linspace(-4, 4, 10)
ang_space = np.linspace(-.2095, .2095, 10)
ang_vel_space = np.linspace(-4, 4, 10)
```

در ادامه براســـاس اینکه در فرایند آموزش هســـتیم Qtable ابتدایی را تشـــکیل می‌دهیم. ابعاد آن متناسب با فضای حالت است، که برابر است با ۱۱*۱۱*۱۱*۱۱*۲. این مقدار ۲ درواقع نشانگر فضای عمل است که گسسته است. اگر درفرایند تست باشیم، Qtable ذخیره شده را برای اجرا فرامی‌خوانیم.

```
# Initialize Q-table
if is_training:
    q = np.zeros((len(pos_space)+1, len(vel_space)+1, len(ang_space)+1, len(ang_vel_space)+1, env.action_space.n))
else:
    with open(args.qtable, 'rb') as f:
        q = pickle.load(f)
```

با اســـتفاده از کد زیر تنظیماتی که کاربر وارد کرده، با اســـتفاده از آرگومان‌های مربوطه، در متغیرهای محلی متناظر ذخیره می‌کنیم.

```
# setting the parameters for training
learning_rate_a = args.learning_rate # alpha or learning rate
discount_factor_g = args.discount_factor # gamma or discount factor.
episodes = args.episodes
epsilon = args.epsilon          # 1 = 100% random actions
epsilon_decay_rate = args.epsilon_decay # epsilon decay rate
rng = np.random.default_rng()   # random number generator

rewards_per_episode = []

i = 0
```

در حلقه آموزش، ابتدای هر اپیزود محیط reset می‌شـــود، حالت نیز در متغیرهای محلی ذخیره می‌شود، تا برای بروزرسانی تابع ارزش استفاده شوند. یک متغیر دیگر به نام trajectory تعریف می‌شود، زیرا لازم است یک مسیر کامل تا انتها تولید شود.

```
def OnMonteCarlo(args):
    for i in range(episodes):
        state = env.reset()[0]  # Initial state
        state_p = np.digitize(state[0], pos_space)
        state_v = np.digitize(state[1], vel_space)
        state_a = np.digitize(state[2], ang_space)
        state_av = np.digitize(state[3], ang_vel_space)

        trajectory = []  # Store (state, action, reward) tuples
        terminated = False
        rewards = 0
```

همانطور که گفتیم برای مونت کارلو لازم است یک اپیزود تا انتها تولید شود، به همین منظور از حلقه درونی اســـتفاده می‌کنیم. در این حلقه ابتدا یک عدد تصـــادفی تولید می‌گردد، اگر این عدد از مقدار epsilon کمتر باشد، عملی تصادفی انتخاب می‌شود. در غیر این صورت ویا اگر در فرایند تست باشیم، به صورت greedy عمل خواهیم کرد.

```
# Generate an episode
while not terminated and rewards < 10000:
    if is_training and rng.random() < epsilon:
        # Choose random action (exploration)
        action = env.action_space.sample()
    else:
        # Choose greedy action (exploitation)
        action = np.argmax(q[state_p, state_v, state_a, state_av, :])
```

پس از اعمال این عمل به محیط، حالت جدید دریافت می‌شود، گسسته سازی شده و در متغیر محلی مربوطه ذخیره می‌شود.

سپس این حالت جدید، به همراه عمل انجام شده و پاداش دریافتی به متغیر trajectory اضافه می‌شود.

```
# Append the experience to the trajectory
trajectory.append((state_p, state_v, state_a, state_av, action, reward))

# Update state variables
state = new_state
state_p = new_state_p
state_v = new_state_v
state_a = new_state_a
state_av = new_state_av

rewards += reward
```

پس از تولید trajectory تا terminal، از حلقه درونی خارج می‌شویم. اگر در فرایند آموزش باشیم، Qtable به صورت زیر به روز می‌شود. ابتدا برای هر تک المان درون trajectory، که شامل ۴ حالت، یک عمل و reward مربوطه است، میزان return محاسبه می‌شود. سپس Qtable به صورت بازگشتی به روز می‌گردد.

```
def OnMonteCarlo(args):
    # Monte Carlo updates after the episode
    if is_training:
        G = 0  # Initialize return
        for state_p, state_v, state_a, state_av, action, reward in reversed(trajectory):
            G = reward + discount_factor_g * G  # Calculate return
            # Update Q-value using incremental mean
            q[state_p, state_v, state_a, state_av, action] = q[state_p, state_v, state_a, state_av, action] + learning_rate_a * (
                G - q[state_p, state_v, state_a, state_av, action]
            )
```

نحوه به روز رسانی تابع ارزش حالت و عمل مطابق معادله زیر است:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

این معادله مربوط به GLIE Monte Carlo Control[1] است که روشی On Policy است.

[1] از دوره‌ی یادگیری تقویتی David Silver

سپس میانگین پاداش‌ها بدست می‌آید، و مقادیر در ترمینال چاپ می‌شود. اگر میزان پاداش در یک اپیزود بیش از ۱۰۰۰ شود، این اپیزود قطع می‌شود.

```
def OnMonteCarlo(args):
        # Record rewards and print progress
        rewards_per_episode.append(rewards)
        mean_rewards = np.mean(rewards_per_episode[-100:])

        if not is_training and rewards % 100 == 0:
            print(f'Episode: {i}  Rewards: {rewards}')

        if is_training and i % 100 == 0:
            print(f'Episode: {i} {rewards}  Epsilon: {epsilon:0.2f}  Mean Rewards {mean_rewards:0.1f}')

        if mean_rewards > 1000:
            break
```

سپس اپسیلون در انتهای اپیزود کاهش می‌یابد. پس از اتمام آموزش Qtable با استفاده از کتابخانه pickle ذخیره می‌گردد. نمودار تغییرات میانگین پاداش نیز رسم می‌گردد.

```
def OnMonteCarlo(args):
            epsilon = max(epsilon - epsilon_decay_rate, 0)

        env.close()

        # Save Q table to file
        if is_training:
            with open(f'cartpole_OnMC_{episodes}.pkl', 'wb') as f:
                pickle.dump(q, f)

        # plot rewards convergence graph
        mean_rewards = []
        for t in range(i):
            mean_rewards.append(np.mean(rewards_per_episode[max(0, t-100):(t+1)]))
        plt.plot(mean_rewards)
        plt.savefig(f'cartpole_On-Policy_MonteCarlo.png')
```

در نهایت تابع با پاس دادن آرگومان‌ها و تنظمیات اجرا، اجرا می‌گردد، زمان اجرای فرایند آموزش و یا تست نیز اندازه گیری شده جهت مقایسه در ترمینال چاپ می‌شود.

```python
if __name__ == '__main__':
    args = parse_args()
    t1 = time.time()
    OnMonteCarlo(args)
    t2 = time.time()

    processing_time = t2 - t1
    print(f"Training Time: {processing_time}")
```

توجه! برای آموزش هر سه روش از پارامترهای پیش‌فرضی که در توضیحات ابتدایی ذکر شد استفاده شده است و تعداد اپیزودهای تمام آموزش‌ها ۱۵۰۰۰۰ است.

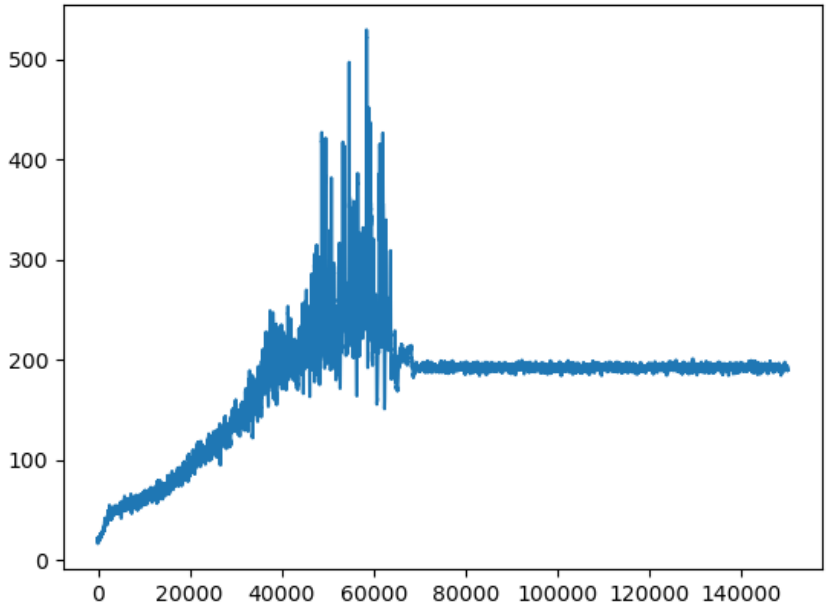
```
Episode: 2  Rewards: 1300.0
Episode: 2  Rewards: 1400.0
Episode: 2  Rewards: 1500.0
Episode: 2  Rewards: 1600.0
Process Time: 65.39161944389343
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Reinforcement Learning/Comperehensive Project/Code$ python CartePoleOnMonteCarlo.py --train train --render no --epis
000
Training Started...
Episode: 0 19.0  Epsilon: 0.70  Mean Rewards 19.0
Episode: 100 62.0  Epsilon: 0.70  Mean Rewards 22.3
Episode: 200 20.0  Epsilon: 0.70  Mean Rewards 20.5
```

پس از اتمام اجرا، زمان اجرا نیز به صورت زیر در ترمینال چاپ شد.

```
Episode: 149300 195.0  Epsilon: 0.00  Mean Rewards 194.0
Episode: 149400 190.0  Epsilon: 0.00  Mean Rewards 196.1
Episode: 149500 187.0  Epsilon: 0.00  Mean Rewards 192.3
Episode: 149600 198.0  Epsilon: 0.00  Mean Rewards 193.4
Episode: 149700 195.0  Epsilon: 0.00  Mean Rewards 191.2
Episode: 149800 207.0  Epsilon: 0.00  Mean Rewards 193.2
Episode: 149900 197.0  Epsilon: 0.00  Mean Rewards 193.5
Training Time: 474.6267092227936
```

همانطور که مشاهده می‌شود، زمان اجرای فرایند آموزش ۴۷۴ ثانیه است.

نمودار میانگین پاداش در شکل ۱ رسم شده است. به لحاظ همگرایی ظاهرا عملکرد حین آموزش خوب بوده است. اما لازم است Q Table نهایی آن با اجرا ارزیابی شود.



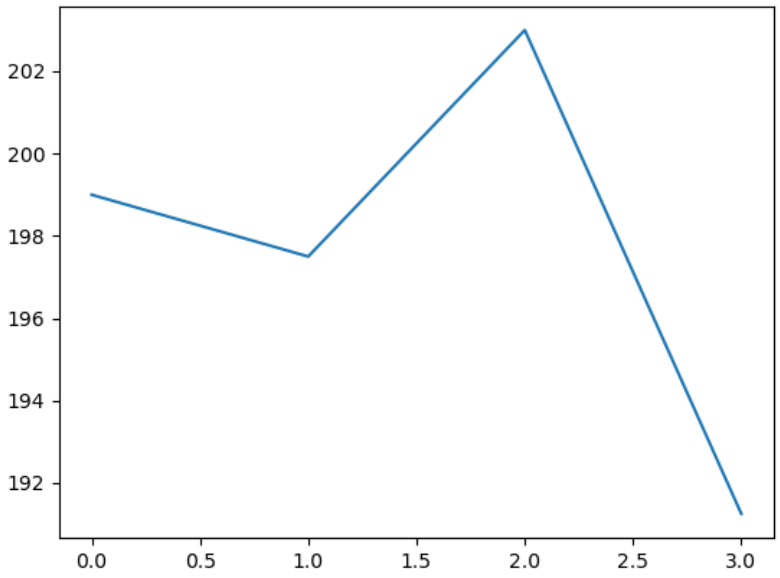
شکل ۱: میانگین پاداش مونت کارلو در طی اپیزودهای مختلف آموزش

با استفاده از کد زیر نیز، فرایند تست با فعالسازی رندر اجرا می‌گردد.

```
setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV: ~/Reinforcement Learning/Comperehensive Project/Code
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Reinforcement Learning/Comperehensive Project/Code$ python CartePoleOnMonteCarlo.py --train test --render yes --epis
Testing Started...
Training Time: 19.49344801902771
```

زمان اجرا ۲۰ ثانیه است، و نمودار میانگین پاداش در طی اپیزودهای تست به صورت شکل ۲ است (۵ اپیزود)

باتوجه به زمان اجرای تست و میانگین پاداش‌های آن، عملکرد مدل ضعیف است. ویدیوی عملکرد آن نیز در پیوست موجود است.



شکل ۲: میانگین پاداش‌های مونت کارلو در طی اپیزودهای تست

در قسمت جمع‌بندی به مقایسه روش‌های مختلف نیز می‌پردازیم.

۱.۳ روش Q Learning

در این قسمت به منظور پیاده سازی روش Q Learning از کدی مشابه با کد مورد استفاده در قسمت قبل استفاده شد. پارامترها مشابه قبل است، کد این بخش تا خطوط حدود ۸۰ مشابه کد مونت کارلو است. در این بخش فقط متغیر trajectory تعریف نمی‌گردد زیرا نیازی به ذخریه سازی کامل یک اپیزود به آن شیوه نیست.

در حلقه درونی، ابتدا بر اساس عدد تصادفی تولید شده، تصمیم گیری می‌شود که عمل به صورت تصادفی و یا به صورت حریصانه انتخاب شود. سپس این عمل به محیط اعمال می‌شود، حالت جدید دریافت، گسسته ودر متغیر محلی متناظر مانند مونت کارلو ذخیره می‌شود.

```
def Qlearning(args):
        while(not terminated and rewards < 10000):

            if is_training and rng.random() < epsilon:
                # Choose random action  (0=go left, 1=go right)
                # NOTE: the action space is discrete
                # if the random number generated is lower than
                # epsilon we should choose random action
                action = env.action_space.sample()
            else:
                # if it is greater than epsilon the policy is greedy
                action = np.argmax(q[state_p, state_v, state_a, state_av, :])

            # apply the action to environment and get state
            new_state,reward,terminated,_,_ = env.step(action)
            new_state_p = np.digitize(new_state[0], pos_space)
            new_state_v = np.digitize(new_state[1], vel_space)
            new_state_a = np.digitize(new_state[2], ang_space)
            new_state_av= np.digitize(new_state[3], ang_vel_space)
```

تفاوت اصلی این کد با کد قبل در نحوه بروز رسانی تابع ارزش حالت عمل است. با الگو گیری از رابطه زیر:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

بروز رسانی را به صورت زیر پیاده سازی کردیم، سپس حالات جدید جایگزین حالات قبلی می‌شوند.

```
# now update the q function with qlearning (off-policy strategy)
if is_training:
    q[state_p, state_v, state_a, state_av, action] = q[state_p, state_v, state_a, state_av, action] + learning_rate_a * (
        reward + discount_factor_g*np.max(q[new_state_p, new_state_v, new_state_a, new_state_av,:]) - q[state_p, state_v, state_a, state_av, action]
    )

# store new states into previous variables
state = new_state
state_p = new_state_p
state_v = new_state_v
state_a = new_state_a
state_av= new_state_av

# calculate reward
rewards+=reward
```

سایر قسمت‌های کد نیز مشابه کد مونت کارلو است. برای آموزش از دستور زیر در ترمینال استفاده می‌کنیم.

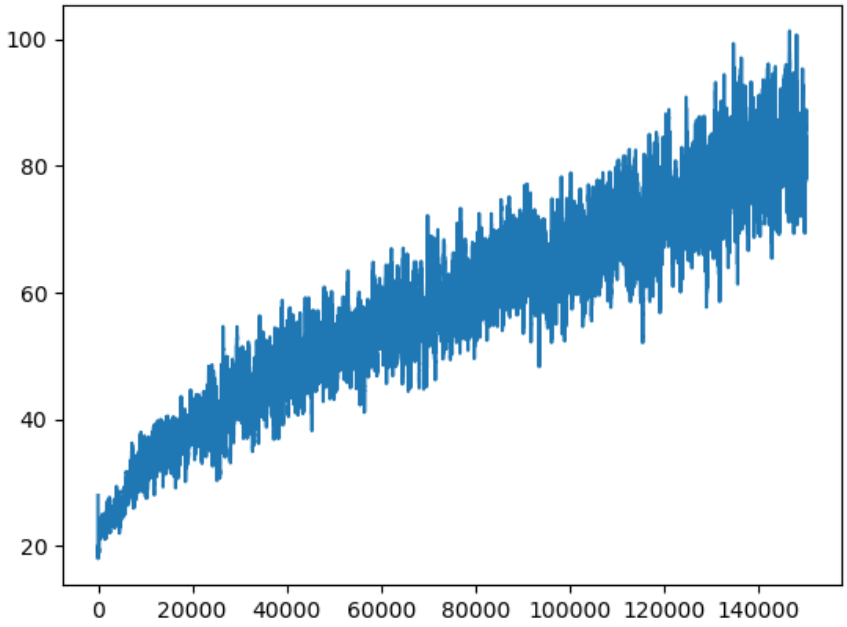
```
python CartePoleQlearning.py --train train --render no --episodes 150000
```

در آموزش با Q Learning نرخ کاهش epsilon را نیز کاهش دادیم تا epsilon به صفر نرسد.

```
Episode: 148500 23.0  Epsilon: 0.55  Mean Rewards 75.6
Episode: 148600 170.0  Epsilon: 0.55  Mean Rewards 86.3
Episode: 148700 26.0  Epsilon: 0.55  Mean Rewards 83.0
Episode: 148800 40.0  Epsilon: 0.55  Mean Rewards 77.9
Episode: 148900 81.0  Epsilon: 0.55  Mean Rewards 82.1
Episode: 149000 14.0  Epsilon: 0.55  Mean Rewards 79.2
Episode: 149100 101.0  Epsilon: 0.55  Mean Rewards 80.1
Episode: 149200 204.0  Epsilon: 0.55  Mean Rewards 90.7
Episode: 149300 38.0  Epsilon: 0.55  Mean Rewards 85.2
Episode: 149400 147.0  Epsilon: 0.55  Mean Rewards 80.2
Episode: 149500 106.0  Epsilon: 0.55  Mean Rewards 82.8
Episode: 149600 42.0  Epsilon: 0.55  Mean Rewards 73.3
Episode: 149700 89.0  Epsilon: 0.55  Mean Rewards 73.9
Episode: 149800 91.0  Epsilon: 0.55  Mean Rewards 83.5
Episode: 149900 233.0  Epsilon: 0.55  Mean Rewards 81.0
Training Time: 187.48808765411377
```

همانطور که مشاهده می‌شود، فرایند آموزش بیش از ۲ برابر سریعتر از مونت کارلو اجرا می‌گردد.

در شکل ۳، نمودار میانگین پاداش در طول اپیزودهای آموزش رسم شده است. همانطور که مشاهده می‌شود روندی صعودی دارد، دلیل اینکه به مقدار بیشینه همگرا نشده است و همچنان در حال افزایش است این است که مقدار epsilon در انتها به حددود ۰.۵ می‌رسد و همچنان از جست و جو نیز در کنار انتخاب حریصانه استفاده می‌گردد. در مونت کارلو چون نرخ کاهش epsilon کمتر لحاظ شده بوده است، به مقدار بیشینه‌ای که می‌توانسته برسد، همگرا می‌شود.

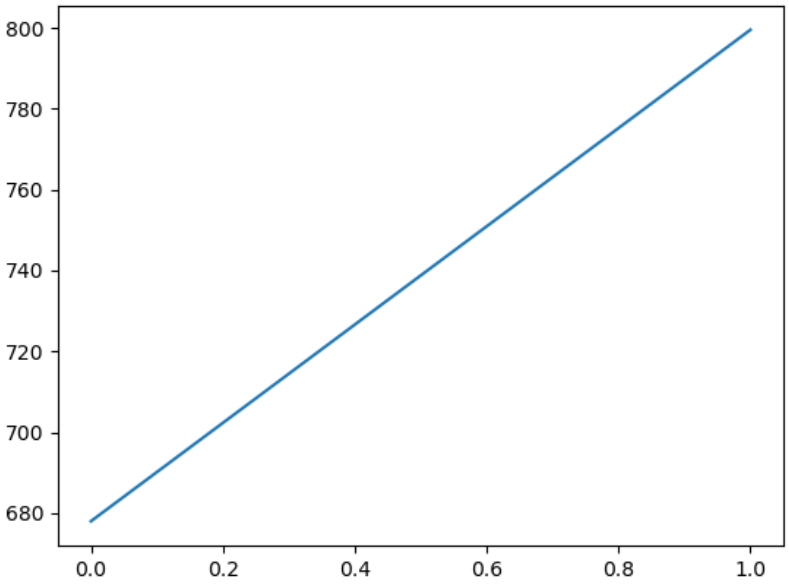


شکل ۳: میانگین پاداش روش q learning در طول آموزش

مانند بخش قبل، عملکرد این روش ارزیابی گردید. خروجی تست آن به صورت زیر است:

```
Episode: 2  Rewards: 600.0
Episode: 2  Rewards: 700.0
Episode: 2  Rewards: 800.0
Episode: 2  Rewards: 900.0
Episode: 2  Rewards: 1000.0
Episode: 2  Rewards: 1100.0
Episode: 2  Rewards: 1200.0
Episode: 2  Rewards: 1300.0
Episode: 2  Rewards: 1400.0
Episode: 2  Rewards: 1500.0
Episode: 2  Rewards: 1600.0
Process Time: 65.39161944389343
```

همانطور که مشاهده می‌شود، زمان اجرای فقط ۲ اپیزود آن ۶۵ ثانیه است و پاداش در انتها به بیش از ۱۰۰۰ رسیده است. نمودار مربوطه در شکل ۴ موجود است.



شکل ۴: نمودار پاداش در ۲ اپیزود q learning فرایند تست

ویدیوی عملکرد آن نیز در پیوست موجود است، در مجموع عمکلرد این مدل مناسب است.

۱.۴ روش SARSA

در این بخش نیز کد مربوط به SARSA پیاده سازی گردید.

شـرایط آزمایش (پارامترها و تعداد اپیزود) در این بخش کاملا مشـابه قسـمت قبلی، Q Learning است.

کد این بخش نیز با اسـتفاده از کد Q Learning نوشـته شـد، تنها تفاوت پیاده سـازی SARSA در نحوه‌ی پیاده سازی بروزرسانی تابع ارزش حالت و عمل است.

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$$

با استفاده از رابطه بالا، بروز رسانی را تغییر دادیم.

مطابق الگوریتم، در ابتدا، عمل انتخاب می‌شـود، پس از اعمال آن به محیط، عمل بعدی نیز براسـاس سـیاسـت e-greedy صـرفا انتخاب می‌شـود ولی اعمال نمی‌شـود، صـرفا از آن برای بروز رسـانی تابع ارزش حالت و عمل استفاده می‌شود.

```
# now update the q function with SARSA (on-policy strategy)
if is_training:
    # Choose next action based on current policy (ε-greedy)
    next_action = None
    if rng.random() < epsilon:
        next_action = env.action_space.sample()  # Random action
    else:
        next_action = np.argmax(q[new_state_p, new_state_v, new_state_a, new_state_av, :])  # Greedy action

    q[state_p, state_v, state_a, state_av, action] = q[state_p, state_v, state_a, state_av, action] + learning_rate_a * (
        reward + discount_factor_g * q[new_state_p, new_state_v, new_state_a, new_state_av, next_action] - q[state_p, state_v, state_a, state_av, action]
    )
```

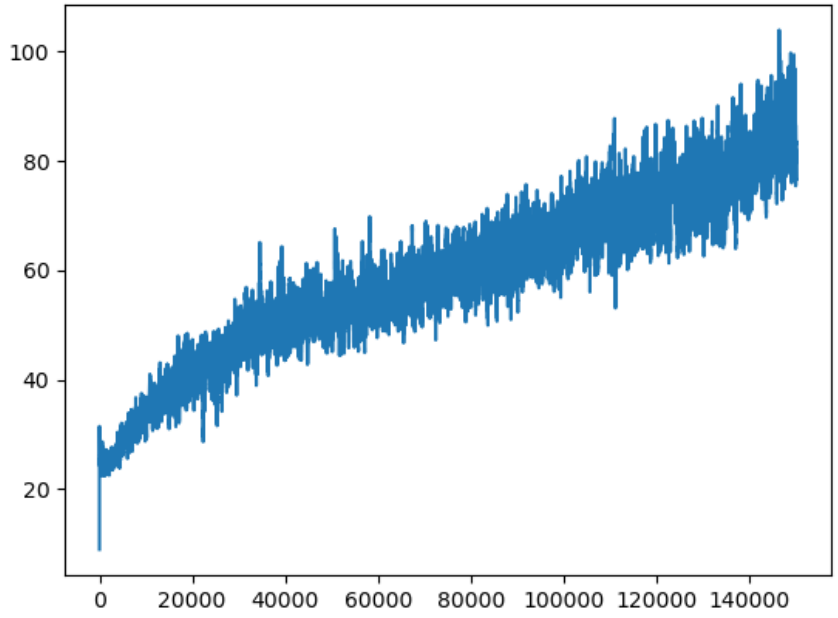
سـایر قسـمت‌های کد مشـابه قبل اسـت. از دسـتور زیر برای اجرای فرایند آموزش در ترمینال اسـتفاده کردیم.

```
python CartePoleSARSA.py --train train --render no --episodes 150000
```

فرایند آموزش آن حدود ۹ ثانیه سریعتر از Q learning اجرا گردید.

```
Episode: 148800 139.0  Epsilon: 0.55  Mean Rewards 97.5
Episode: 148900 71.0  Epsilon: 0.55  Mean Rewards 82.5
Episode: 149000 133.0  Epsilon: 0.55  Mean Rewards 78.8
Episode: 149100 161.0  Epsilon: 0.55  Mean Rewards 94.2
Episode: 149200 20.0  Epsilon: 0.55  Mean Rewards 80.8
Episode: 149300 118.0  Epsilon: 0.55  Mean Rewards 83.6
Episode: 149400 82.0  Epsilon: 0.55  Mean Rewards 82.4
Episode: 149500 139.0  Epsilon: 0.55  Mean Rewards 97.3
Episode: 149600 38.0  Epsilon: 0.55  Mean Rewards 82.0
Episode: 149700 114.0  Epsilon: 0.55  Mean Rewards 91.2
Episode: 149800 117.0  Epsilon: 0.55  Mean Rewards 89.2
Episode: 149900 11.0  Epsilon: 0.55  Mean Rewards 77.8
Process Time: 179.68412923812866
```

نمودار میانگین پاداش آن در طول اپیزودهای آموزش، روندی مشابه Q learning دارد زیرا نرخ کاهش epsilon را به گونه‌ای تنظیم کردیم که در انتها به حدود ۰.۵ و نه به صفر نرسد.



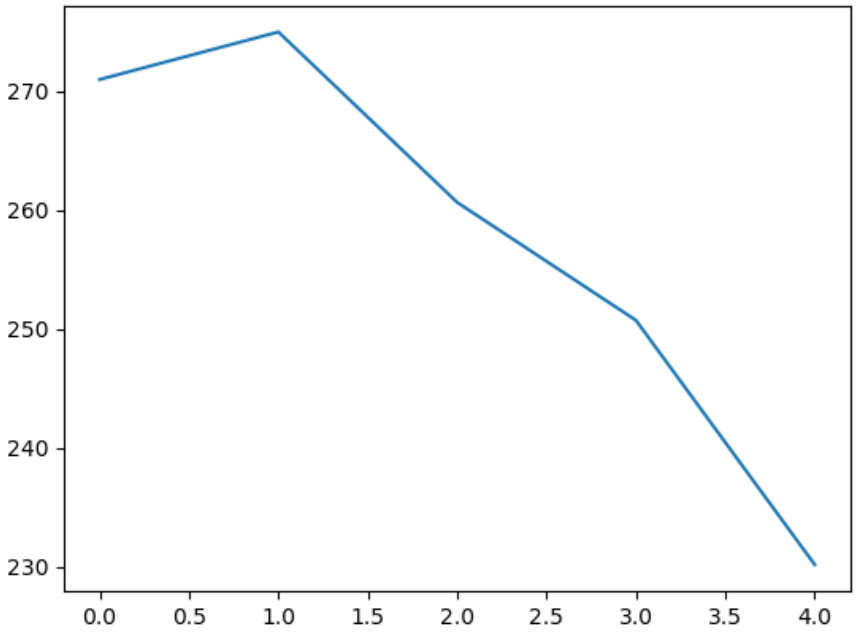
شکل ۵: نمودار میانگین پاداش sarsa در طول اپیزودهای آموزش

تست آن نیز به صورت زیر انجام شد:

```
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Reinforcement Learning/Comperehensive Project/Code$ python CartePoleSARSA.py --train test --render yes --episodes 5
Testing Started...
Episode: 0  Rewards: 100.0
Episode: 0  Rewards: 200.0
Episode: 1  Rewards: 100.0
Episode: 1  Rewards: 200.0
Episode: 2  Rewards: 100.0
Episode: 2  Rewards: 200.0
Episode: 3  Rewards: 100.0
Episode: 3  Rewards: 200.0
Episode: 4  Rewards: 100.0
Process Time: 23.556307315826416
```

از روی مدت زمان اجرای ۵ اپیزود و مقادیر پاداش‌های دریافتی مشـــخص اســـت که عملکرد این مدل وضعیت مناسبی ندارد. (ویدیوی پیوست مشاهده شود)

نمودار پاداش دریافتی آن در طول تست به شکل زیر است:



شکل ۶: پاداش دریافتی sarsa در طول ۵ اپیزود تست

شکل۶ نیز مطابق توضیحات قبلی است که در خصوص عملکرد این مدل بیان گردید.

۱.۵ مقایسه روش‌ها

در این بخش روش‌ها را به لحاظ پارامترهای مختلف مقایسه می‌کنیم.

۱. زمان همگرایی: با وجود تفاوت در نرخ کاهش epsilon، سرعت همگرایی مونت کارلو بسیار کندتر از دو روش دیگر است.

۲. دقت همگرایی: باتوجه به عملکرد مدل‌ها و نمودارهای رسم شده، دقت همگرایی Qlearning از دو روش دیگر بیشتر است، مطابق تئوری درس انتظار می‌رود Qlearning به راه حل بهینه، بهتر از دو روش دیگر، برسد زیرا ماهیتی Off Policy دارد.

۳. پایداری: به لحاظ پایداری، Q learning به دلیل رسیدن به جوابی بهینه‌تر نسبت به سایر روش‌ها باعث پایدارسازی بهتر سیستم می‌گردد.

۱.۶ اجرای برنامه

برای اجرای ساده تر کدها و مشاهده عملکرد آن‌ها، دو کد تحت نام‌های main.py و app.py توسعه داده شده است. کد اول لازم است با وارد کردن آرگومان‌ها در ترمینال اجرا شود (مانند سه کد قبلی) اما با اجرای دستور ساده‌ه زیر در ترمینال:

```
python main.py
```

پنجره GUI ساده‌ای که توسعه داده شده است، باز میگردد.

در این پنجره، می‌توانید انتخاب کنید که چه فرایندی انجام شود، اگر آموزش را انتخاب کنید، لازم است پارامترهای موثر در آموزش را نیز تنظیم کنید. اگر تست را انتخاب کنید صرفا لازم است تعداد اپیزودها را تنظیم کنید و مسیر Qtable را نیز انتخاب کنید. با انتخاب هر یک از ۳ روش، می‌توانید عملکرد روش‌ها را مشاهده کنید.

فصل ۲: یادگیری تقویتی در کنترل بهینه

۲.۱ مقدمه

در این بخش با یک سیستم خطی پیوسته اما ناپایدار روبرو هستیم. لازم است برای آن رگولاتور درجه دوم خطی و ردیاب درجه دوم خطی بهینه طراحی شود.

$$\dot{x}(t)=\begin{bmatrix}0.5 & 1.5\\ 2 & -2\end{bmatrix}x(t)+\begin{bmatrix}1\\ 4\end{bmatrix}u(t)$$
$$y(t)=\begin{bmatrix}1 & 0\end{bmatrix}x(t)$$
$$x(0)=\begin{bmatrix}5\\ -5\end{bmatrix}$$

۲.۲ رگولاتور خطی بهینه

در بخش ۲.۲ به سه قسمت ابتدایی سوال دوم که مربوط به طراحی کنترل کننده برای مسئله LQR است پاسخ می‌دهیم.

صورت سوال بخش الف:

الف) برای سیستم بالا بر اساس الگوریتم IRL زیر یک کنترل کننده طراحی کنید و با شبیه سازی، سیاست بهینه به‌دست آمده در هر مرحله را ثبت کنید. فرض کنید $Q(x)=x^TQx$ و مقادیر Q و R را به‌ترتیب برابر I_2 و ۱ در نظر بگیرید و سیاست اولیه u_0 را پایدارساز انتخاب کنید. Probing Noise را به‌صورت تصادفی بین $[0,0.1]$ در نظر بگیرید. پس از همگرایی نمودار پاسخ زمانی حالت‌ها را رسم کنید. (راهنمایی: از آنجا که با انتخاب Q به‌فرم گفته شده تابع هزینه به‌صورت کوادراتیک در می‌آید، تابع ارزش نیز می‌تواند به‌فرم کوادراتیک نوشته شود. شرط همگرایی را به‌صورت $\left\|P^{j+1}-P^{j}\right\|\le 10^{-4}$ در نظر بگیرید.)

پاسخ بخش الف:

ابتدا برای حل سوال دینامیک سیستم را بررسی می‌کنیم.

با استفاده از کد زیر، ابتدا دینامیک سیستم را تعریف می‌کنیم، سپس مقادیر ویژه ماتریس A را برای تحلیل پایداری سیستم نمایش می‌دهیم، سپس به لحاظ کنترل پذیری، با استفاده از دستور ctrb سیستم را بررسی می‌کنیم، علیرغم اینکه سیستم دارای قطب ناپایدار است اما ماتریس کنترل پذیری آن رنک کامل است، بنابراین می‌توان بدون نگرانی برای این سیستم کنترلر طراحی نمود.

```
%% Define Dynamics
A = [0.5 1.5; 2 -2];
B = [1 4]';
C = [1 0]; % doesn't need
x0 = [5 -5]';

disp("Eigen Values of A:")
disp(eig(A))

disp("A rank:")
disp(rank(ctrb(A,B)))
```

خروجی کد بالا به صورت زیر است:

```
Eigen Values of A:
    1.3860
   -2.8860

A rank:
     2
```

سپس برای اینکه بتوانیم با روش IRL، برای سیستم کنترل کننده طراحی کنیم، نیاز داریم یک سیاست پایدار ساز اولیه نیز بدست آوریم. در اینجا محل قطب‌های اولیه مطلوب را در ۲- و ۴-، انتخاب می‌کنیم، سپس با دستور place بهره فیدبک حالتی که شرایط را براورده می‌کند بدست می‌آوریم.

بدین منظور از کد زیر استفاده می‌کنیم.

```
%% Design admissible policy
desired_poles = [-2 -4];
K = place(A,B,desired_poles);

disp("Stabilizing K: ")
disp(K)
```

بهره فیدبک حالت به صورت زیر بدست می‌آید:

```
Stabilizing K:
    1.5000    0.7500
```

که از آن برای پیگیری فرایند طراحی کنترل کننده بهینه با رویکرد IRL بهره خواهیم برد.

با استفاده از کد زیر دینامیک سیستم و تابع هزینه را تعریف می‌کنیم.

```
%% Define Dynamics
A = [0.5 1.5; 2 -2];
B = [1 4]';
C = [1 0]; % doesn't need
x0 = [5 -5]';

K0 = [1.5 0.75];
n = size(A,1);

Q = eye(n);
R = 1;
```

سپس با استفاده از دستور lqr مقادیر k و p بهینه را بدست می‌آوریم.

```
%% Calculate optimal K and P with LQR command
[K_lqr, P_lqr] = lqr(A,B,Q,R);
disp("K derived from LQR: ")
disp(K_lqr)

disp("P derived from LQR: ")
disp(P lqr)
```

که مقادیر آن برابر است با:

```
K derived from LQR:
    1.6180    0.9001

P derived from LQR:
    0.9077    0.1776
    0.1776    0.1806
```

حال لازم است برای انجام فرایند آموزش، تنظیمات آن را تعیین کنیم. تعداد تکرارها را برابر با ۵۰ لحاظ می‌کنیم، باتوجه به اینکه ابعاد ماتریس P ۲*۲ است، باید ۳ داده برداری صورت بگیرد که جهت اطمینان آن را مساوی ۸ قرار می‌دهیم. برای شبیه سازی قسمت انتگرالی نیز باید از روش‌های عددی و تقریب اویلر استفاده کنیم. بدین منظور نرخ نمونه برداری را برابر با ۰.۰۰۱ انتخاب می‌کنیم. سپس سایر پارامترها جهت بروز رسانی ارزش و سیاست را تنظیم می‌کنیم، اولین المان سیاست را نیز، که سیاست اولیه می‌باشد، برابر با بهره k ای قرار می‌دهیم که با دستور place بدست آوردیم.

سپس برای مقادیر مختلف ماتریس‌های P نیز، متغیری را به صورت سلولی تعریف کردیم که بتوانیم مقادیر مختلف این ماتریس را در سلول‌های آن ذخیره و برای شرط توقف استفاده کنیم.

```
%% LQR using IRL
nP = 50;
M = 8;
Ts = 0.001;
T = 1;
time = 0:Ts:T; Nt = numel(time);
K = zeros(nP, n); K(1,:) = K0;
P_cell = cell(nP , 1) ; P_cell{1} = zeros(n);
```

حال حلقه‌های برنامه را می‌نویسیم، حلقه بیرونی به تعداد کل تکرارها یعنی ۵۰ بار قرار است اجرا شود، ابتدا ماتریس‌های SAI و PHI را تعریف می‌کنیم، از آن‌ها برای اجرای LS قرار است استفاده گردد.

سپس در حلقه دوم داده برداری قرار است صورت پذیرد که درون آن به وسیله حلقه سوم دینامیک سیستم و ورودی کنترلی به علاوه نویز گفته شده در صورت سوال شبیه سازی و مقادیر پاداش با تقریب اویلر محاسبه می‌شود.

پس از اتمام حلقه سوم، درون حلقه دوم مقادیر ماتریس‌های SAI و PHI محاسبه می‌گردد. برای محاسبه انتگرال پاداش، از روش ذوزنقه و دستور trapz استفاده می‌کنیم. باید برای محاسبه و استفاده از ls بردار x به فرم xbar یا همان ضرب کرونکر خودش در خودش تبدیل شود (توسط تابعی که در انتهای کد وجود دارد این تبدلی صورت می‌گیرد)، پارامتر Pbar که فرم برداری P با پارامترهای مستقل است (یعنی ۳ درایه دارد) از تقسیم PHI و SAI بدست می‌آید که از روی آن باید ماتریس P را محاسبه کنیم. توضیحات داده شده به صورت زیر پیاده سازی گردید:

```
for j = 1:nP

    PHI = [];
    SAI = [] ;

    for i = 1:M
        x = zeros(n , Nt) ; x(: , 1) = randn(n , 1) ;
        u = zeros(1 , Nt) ;
        r = zeros(1 , Nt) ;

        for k = 1:Nt-1
            u(k) = -K(j , :)*x(: , k)+0.1*rand ;
            x(: , k+1) = x(: , k) + Ts*(A*x(: , k)+B*u(k));
            r(k) = x(: , k)'*Q*x(: , k)+u(k)'*R*u(k);
        end
        SAI = [SAI ; trapz(time , r)];
        PHI = [PHI ; ComputeXbar(x(: , 1))-ComputeXbar(x(: , k+1))] ;
    end

    Pbar = PHI\SAI ;
    P = ConvertPbarToP(Pbar) ;
    P_cell{j+1} = P;
```

پس از آن سیاست به صورت زیر بروزرسانی می‌گردد.

```
K(j+1 , :) = inv(R)*B'*P ;
```

شرط توقف مطابق صورت سوال به صورت زیر پیاده سازی گردید.

```
if norm(P_cell{j+1}-P_cell{j}, 'fro') < 1e-4
    break;
end
```

در ادامه مقادیر K بهینه حاصل از IRL، چاپ و روند همگرایی آن رسم می‌گردد.

برای محاسبه حاصل ضرب کرونکر x در خودش از تابع زیر استفاده کردیم.

```
function Xbar = ComputeXbar(X)
    X = X(:)';
    Xbar = [] ;

    for i = 1:numel(X)
        Xbar = [Xbar X(i)*X(i:end)];
    end
end
```

و برای تبدیل P به فرم ماتریسی آن، از تابع زیر استفاده گردید.

```
function P = ConvertPbarToP(Pbar)

    P = [Pbar(1)   Pbar(2)/2
         Pbar(2)/2 Pbar(3)];
end
```

حال کد را به طور کامل اجرا می‌کنیم.

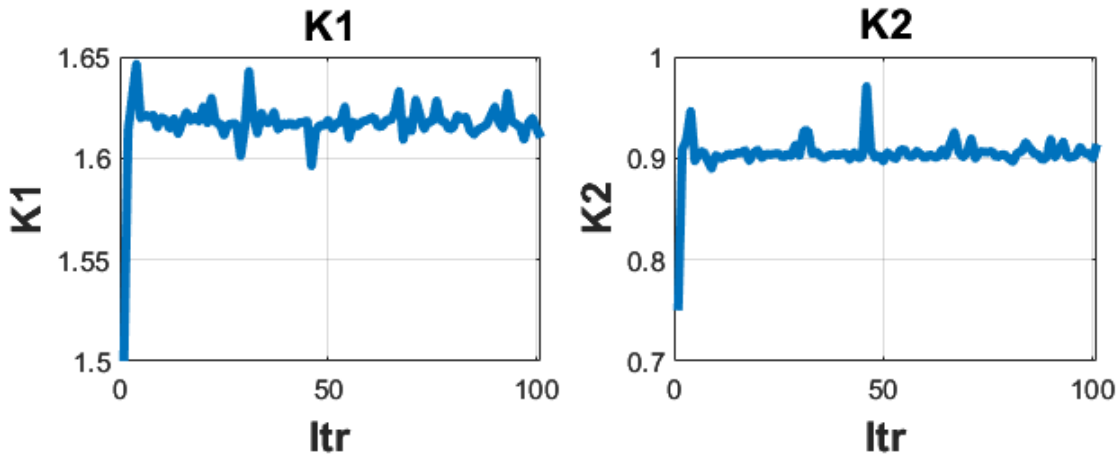
شرط توقف با اجرای کد برقرار نگردید، مقادیر سیاست بهینه در مقایسه با LQR به صورت زیر بدست آمد:

```
K LQR = 1.618      0.90012
K IRL = 1.6078      0.90655
```

مقدار بدست آمده بسیار نزدیک حالت بهینه است. می‌توانیم تعداد تکرارها را بیشتر کنیم. پس از صد تکرار داریم:

```
K LQR = 1.618      0.90012
K IRL = 1.6165      0.90162
```

نمودار همگرایی نیز به صورت زیر بدست می‌آید:



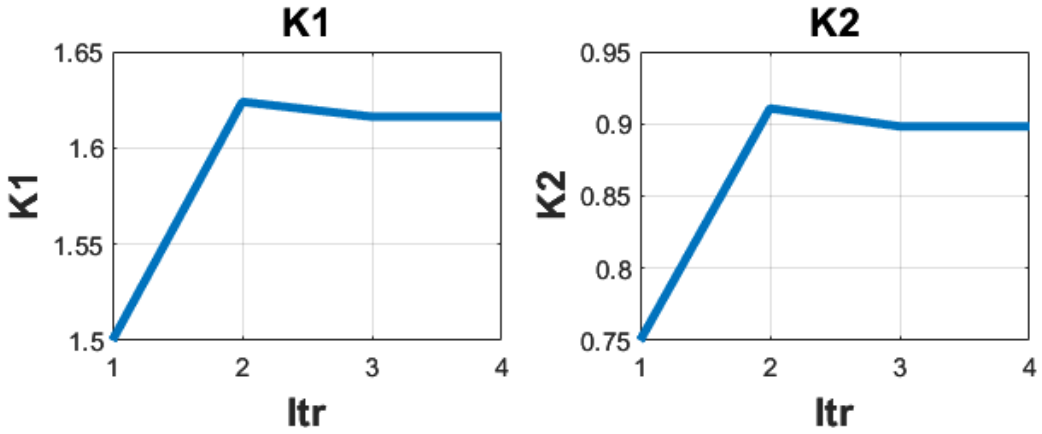
شکل ۷: نمودار همگرایی مقادیر k در on policy IRL

می‌توانیم نویز اضافه شده را کوچکتر کنیم، ضریب آن را بجای ۰.۱، ۰.۰۱ قرار می‌دهیم.

در صورت کاهش بازه‌ی تغییرات نویز، در ۳ تکرار شرط توقف اغنا می‌گردد و نتایج به صورت زیر خواهد بود که بسیار به نتیجه LQR نزدیک است:

```
Iteration(1)
Iteration(2)
Iteration(3)
K LQR = 1.618      0.90012
K IRL = 1.6162      0.89818
```

و همگرایی K در ۳ تکرار به صورت زیر خواهد بود:

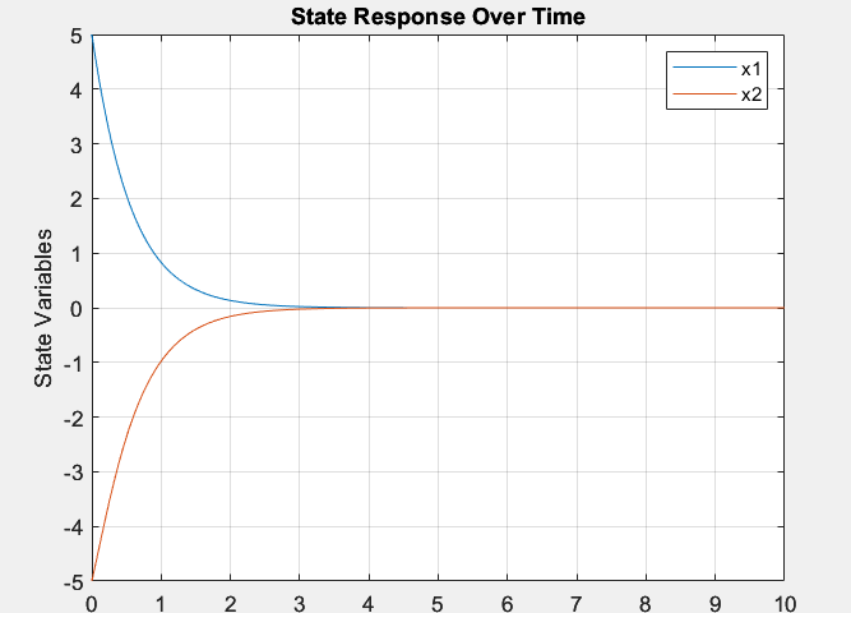


شکل ۸: نمودار همگرایی k پس از کاهش بازه نویز در on policy IRL

با اعمال K حاصل از IRL رفتار متغیرهای حالت را با کد زیر رسم می‌کنیم. (نیاز به توضیح اضافه نیست صرفا شبیه سازی سیستم است)

```
%% Plot System Variables
A = [0.5 1.5; 2 -2];
B = [1 4]';
n = size(A, 1);

K_IRL = [1.6162     0.89818];


t = 0:0.01:10;

x0 = [5 -5]';

% Function to represent the system dynamics (x'(t) = Ax(t) + Bu(t))
system = @(t, x) (A - B * K_IRL) * x;

% Solve the system of ODEs using ode45
[t, x] = ode45(system, t, x0);
```

خروجی به صورت زیر است:



شکل ۹: متغیرهای حالت پس از اعمال سیاست کنترلی بهینه

صورت سوال بخش ب:

Algorithm 6 On-Policy IRL Algorithm to Find the Solution of HJB

1: **procedure**
2: Given admissible policy u_0
3: for $j = 0, 1, \ldots$ given u_j, solve for the value $V_{j+1}(x)$ using Bellman equation

$$V_{j+1}(x(t)) = \int_t^{t+T} \left(Q(x) + u_j^T R u_j\right) d\tau + V_{j+1}(x(t+T)),$$

on convergence, set $V_{j+1}(x) = V_j(x)$.
4: Update the control policy $u_{j+1}(k)$ using

$$u_{j+1}(t) = -\frac{1}{2} R^{-1} g^T(x) \left(\frac{\partial V_{j+1}(x)}{\partial x}\right).$$

5: Go to 3.
6: **end procedure**

ب) از الگوریتم on-policy فوق الگوریتم off-policy زیر را به‌دست آورده و مراحل آن را گزارش کنید.

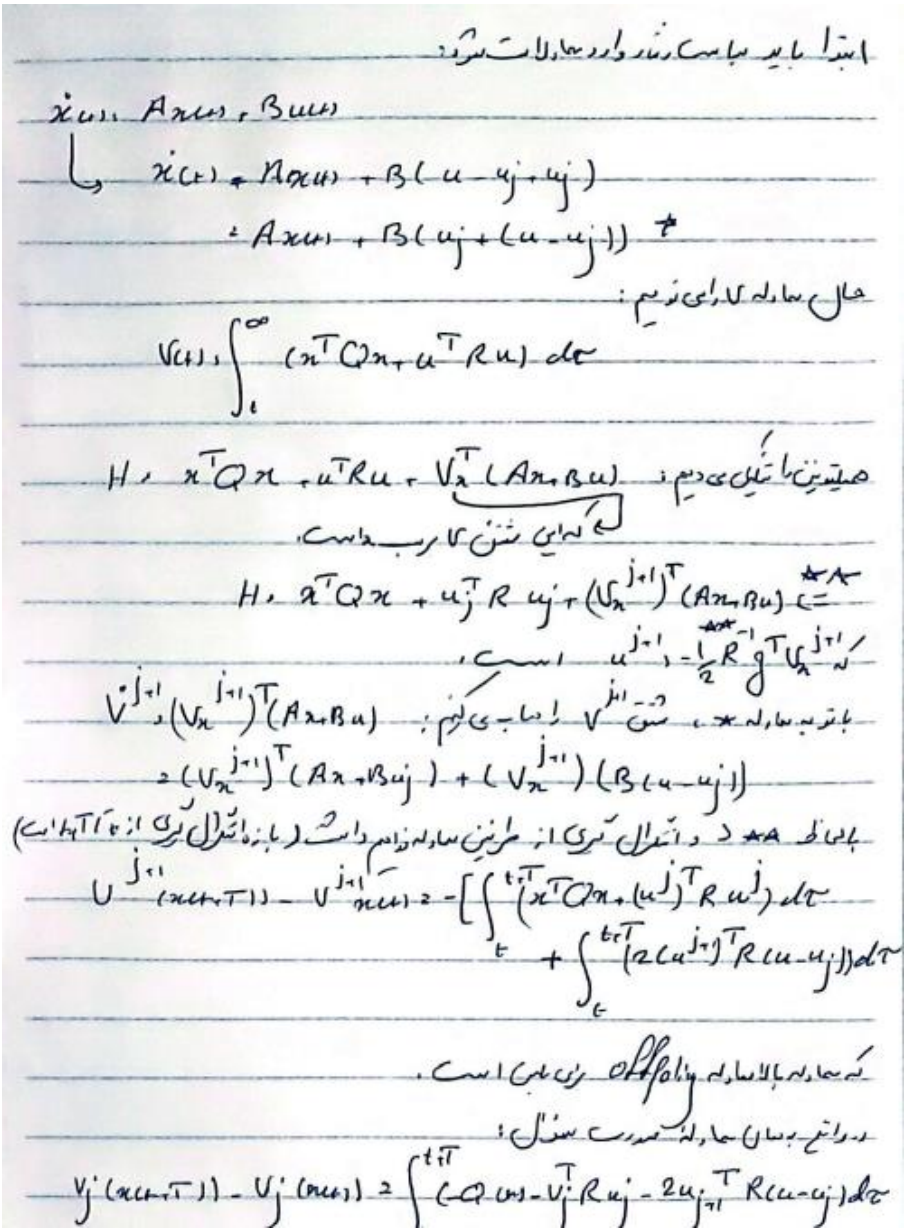
Algorithm 7 Off-Policy IRL Algorithm to Find the Solution of HJB

1: **procedure**
2: Given admissible policy u_0
3: for $j = 0, 1, \ldots$ given u_j, solve for the value V_j and u_{j+1} using off-policy Bellman equation

$$V_j(x(t+T)) - V_j(x(t)) = \int_t^{t+T} \left(-Q(x) - u_j^T R u_j - 2 u_{j+1}{}^T R (u - u_j)\right) d\tau.$$

on convergence, set $V_{j+1} = V_j$.
4: Go to 3.
5: **end procedure**

پاسخ بخش ب:


ابتدا باید بیاییم رفتار دینامیکی سیستم را بررسی کنیم:
$\dot{x}(t) = Ax(t) + Bu(t)$
$\dot{x}(t) = Ax(t) + B(u - u_j + u_j)$
$= Ax(t) + B(u_j + (u - u_j))$ *
حال برای معادله HJB داریم:
$V(t) = \int_t^\infty (x^T Q x + u^T R u)\, d\tau$
حال برای معادله HJB داریم:
$H = x^T Q x + u^T R u + V_x^T (Ax + Bu)$
که برای شرط بهینگی باید
$H = x^T Q x + u_j^T R u_j + (V_x^{j+1})^T (Ax + Bu)$ **
$u^{j+1} = -\frac{1}{2} R^{-1} g^T V_x^{j+1}$ *** است،
با توجه به معادله ** حال V^{j+1} را بدست می‌آوریم:
$V^{j+1} = (V_x^{j+1})^T (Ax + Bu)$
$= (V_x^{j+1})^T (Ax + Bu_j) + (V_x^{j+1})(B(u - u_j))$
با توجه به معادله ** و انتگرال گیری از طرفین معادله و با بازه انتگرال گیری از t تا $t+T$ است)
$V^{j+1}(x(t+T)) - V^{j+1}(x(t)) = -\left[\int_t^{t+T} (x^T Q x + (u_j)^T R u_j)\, d\tau + \int_t^{t+T} (2(u^{j+1})^T R(u - u_j))\, d\tau\right]$
که معادله بالا معادله Off-policy زیر می‌باشد.
و برای بدست آوردن معادله مورد سئوال:
$V_j(x(t+T)) - V_j(x(t)) = \int_t^{t+T} (-Q u_j - V_j^T R u_j - 2u_{j+1}^T R(u - u_j))\, d\tau$

صورت سوال بخش پ:

پ) با استفاده از الگوریتم off-policy زیر، شبیه‌سازی قسمت الف را تکرار کنید. پارامترهای مورد نیاز را مشابه بخش الف در نظر گرفته و یک سیاست پایدارساز به‌عنوان سیاست رفتار به‌دست آورید. نویز اکتشاف را پس از همگرایی حذف کنید. همگرایی و پایداری الگوریتم زیر را با الگوریتم اول مقایسه نمایید و نمودارهای پاسخ زمانی حالت‌ها را رسم کنید.

پاسخ بخش پ:

در این بخش نیز، کد مشابه قبل است و صرفا محاسبه پاداش به صورت زیر تغییر یافته است:

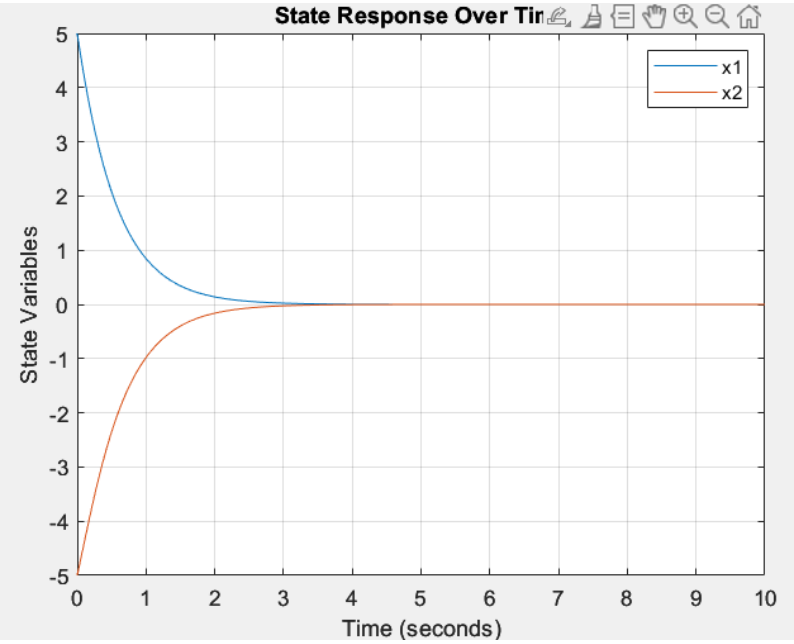
```
r(k) = (x(:, k)'*Q*x(:, k)) + u(k)'*R*u(k) + 2*u(k)'*R*(u(k) + K(j , :)*x(:, k));
```

K بهینه به صورت زیر بدست می‌آید:

```
1.6187       0.9103
```

که بسیار نزدیک به بهره بدست آمده با دستور LQR است.

حالات سیستم نیز به صورت زیر بدست می‌آیند.



شکل ۱۰: نمودار حالات سیستم

همگرایی با الگوریتم Off Policy دیر تر از On Policy رخ می‌دهد، اما پاسخ Off Policy با حالت بهینه منطبق‌تر است، هرچند که پاسخ حالت On Policy نیز تا حد خوبی تطابق دارد.