Задание 1.

Произведена изучение работы алгоритма для задачи Taxi-v3.
Из основных измений входных параметров задачи - появилось 500 возможных состояний, и 6 активных действий. Для возможности фиксирования и чистоты эксперементов был выбран и применен random_state = 1771.

Для базового случая я проверил несколько вариантов запуска алгоритма и его сходимость. Результаты эксперементов приведены в таблице ниже (Столбцы Mean и Max отображены по значениям последней эпохи в каждом случае.

Q_param	Trajectory_n	Itern_n	Max_length	Mean(total_rewards)	Max(total_rewards)
0.8	40	30	100	-124.575	14
0.6	100	30	100	-156.82	8
0.6	200	30	300	-105.175	15
0.6	300	30	150	-56.93	14
0.8	300	30	300	-76.29	15
0.6	400	40	500	-40.8	15
0.4	400	40	500	6.75	15

Вывод о проделанных эксперементах:
В отличие, от обычного лабиринта 5x5 у нас слишком мало траекторий заканчивается в в финишной точке. И для более успешной сходимости алгоритма, нам необоходимо выбирать q-значение намного меньше, чтобы в него попало как можно больше «лучших» траектиорий. Смотря на «лог» итерации при лучших параметрах можно сделать вывод, что такое колличество эпохо нам не нужно. Мы могли ограничиться и меньшим кол-вом эпох.

«Лог лучшего эксперимента»
‘ ‘ ‘ initial state (389, {'prob': 1})
iteration 0 mean total reward: -769.17
iteration 1 mean total reward: -717.1325
iteration 2 mean total reward: -653.345
iteration 3 mean total reward: -590.2275
iteration 4 mean total reward: -522.495
iteration 5 mean total reward: -446.0775
iteration 6 mean total reward: -369.97
iteration 7 mean total reward: -287.385
iteration 8 mean total reward: -208.71
iteration 9 mean total reward: -126.51
iteration 10 mean total reward: -84.8125
iteration 11 mean total reward: -56.4075
iteration 12 mean total reward: -32.435
iteration 13 mean total reward: -19.405
iteration 14 mean total reward: -9.31
iteration 15 mean total reward: -1.6425
iteration 16 mean total reward: 1.6825
iteration 17 mean total reward: 3.645
iteration 18 mean total reward: 5.0625
iteration 19 mean total reward: 5.7125
iteration 20 mean total reward: 6.285
iteration 21 mean total reward: 6.3
iteration 22 mean total reward: 6.56
iteration 23 mean total reward: 6.4875
iteration 24 mean total reward: 7.1425
iteration 25 mean total reward: 7.105
iteration 26 mean total reward: 6.5325
iteration 27 mean total reward: 6.9375

iteration 28 mean total reward: 6.76
…
iteration 38 mean total reward: 6.49
iteration 39 mean total reward: 6.75
Средняя награда на последней эпохе: 6.75
Максимальная награда на последней эпохе: 15

Process finished with exit code 0
’’’

Задание 2.1

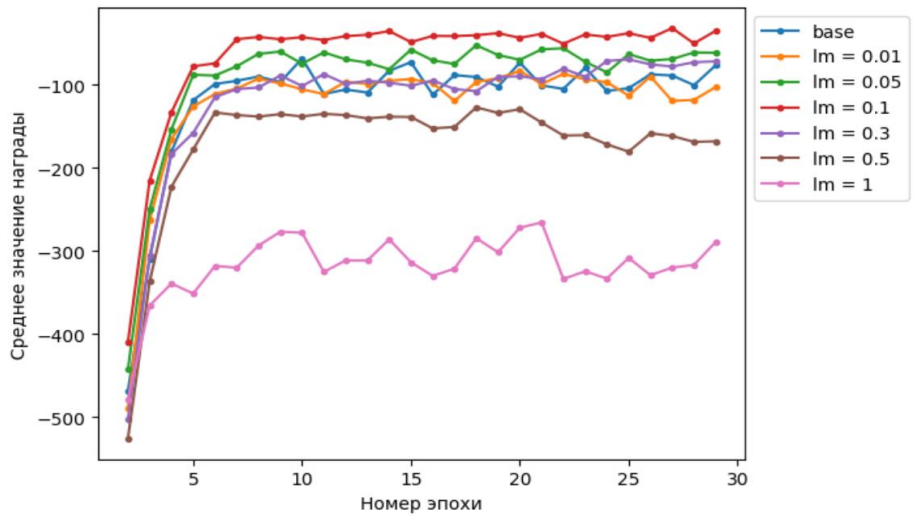
Основываясь на формуле Laplace smoothing

$$\pi_{n+1}(a|s) = \frac{|(a|s) \in \mathcal{T}_n| + \lambda}{|s \in \mathcal{T}_n| + \lambda|\mathcal{A}|}, \quad \lambda > 0$$

Я переписал модель вычисления вероятность для совершенния разных действий в разных состояних.
Я взял не самую лучшую траекторию из задания 1 и попробовал её улучшить. В таблице приведены результаты.

```
for state in range(self.state_n):
    if np.sum(np.sum(new_model[state])) > 0:
        new_model[state] = (new_model[state] + self.lm) / (np.sum(new_model[state]) + self.lm * action_n)
        new_model[state] /= np.sum(new_model[state])
    else:
        new_model[state] = self.model[state].copy()
```

Номер эксперимента	Q	Trajectory_n	Itern_n	Max_length	lm	Mean(total_rewards)	Max(total_rewards)
0й (база)	0.8	300	30	300	0	-76.29	15
1	0.8	300	30	300	0.05	-61.30	15
2	0.8	300	30	300	0.1	-34.785	15
3	0.8	300	30	300	0.3	-71.6225	15
4	0.8	300	30	300	0.5	-167.8575	14
5	0.8	300	30	300	1	-288.9575	7
6	0.8	300	30	300	0.01	-102.55	15



Можно сделать вывод, что нельзя в слепую ставить значени lm. Не все опыты закончились лучше, чем вообще без лямбды(См. Базовый график). Наиболее лучшие результаты были достингуты при значение lm = 0.1 и lm =0.05.

Задание 2.2

Policy smoothing

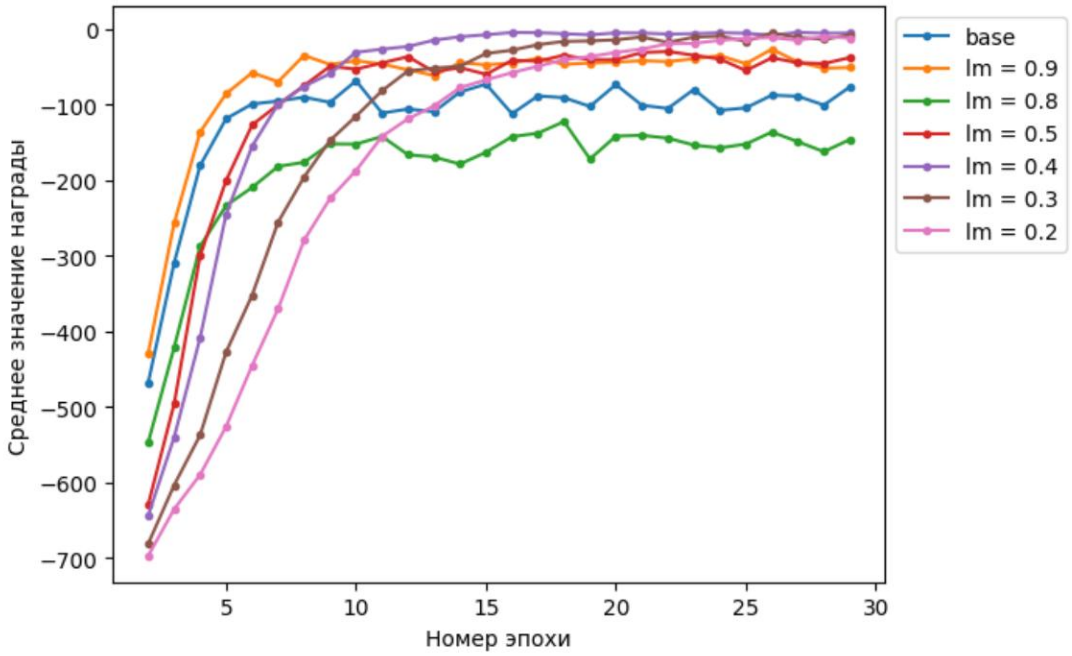
$$\pi_{n+1}(a|s) \leftarrow \lambda\pi_{n+1}(a|s) + (1-\lambda)\pi_n(a|s), \quad \lambda \in (0,1]$$

Необходимо применить второй тип сглаживания, основываясь на предыдущих значениях модели.

```
for state in range(self.state_n):
    if np.sum(np.sum(new_model[state])) > 0:
        new_step = new_model[state] / np.sum(new_model[state])
        old_step = self.model[state].copy()
        new_model[state] = new_step * self.lm + old_step * (1 - self.lm)
    else:
        new_model[state] = self.model[state].copy()
```

Я переписал способ обучения модели на стейджах, использовав старое состояние стейджа.
И провел эксперементы со значениями параметра self.lm

Номер эксперимента	Q	Trajectory_n	Itern_n	Max_length	lm	Mean(total_rewards)	Max(total_rewards)
0й (база)	0.8	300	30	300	0	-76.29	15
1	0.8	300	30	300	0.9	-50.58	15
2	0.8	300	30	300	0.8	-145	15
3	0.8	300	30	300	0.5	-37.46	15
4	0.8	300	30	300	0.4	-4.99	15
5	0.8	300	30	300	0.3	-7.52	15
6	0.8	300	30	300	0.2	-11.68	14



Данный способ регулязации алгоритма оказался гораздо лучше и удобнее. Да, при некоторых параметрах алгоритм начал сходиться дольше, но в результате среднее значение последних эпох намного выше базовой модели.

Задание 3

Я переписал идею обучение Стохастического Агента. Не уверен, что это то, что от меня требовалось, но сделал как посчитал нужным.
Агенту добавляется 3 параметра eps, k, max_eps.
Где - eps - значение коэффициента “thresholda-” для выбора схемы действия агентом на каждой шаге. При каждом обучение агента значение eps функции повышается до максимального значения max_eps. K - коэффициент повышение (шаг).

```
def get_action(self, state):
    if np.random.uniform(0, 1) > self.eps:
        action = np.random.choice(np.arange(self.action_n))

    else:
        action = np.argmax(self.model[state])

    return int(action)

def fit(self, elite_trajectories):
    new_model = np.zeros((self.state_n, self.action_n))
    for trajectory in elite_trajectories:
        for state, action in zip(trajectory['states'], trajectory['actions']):
            new_model[state][action] += 1

    for state in range(self.state_n):
        if np.sum(np.sum(new_model[state])) > 0:
            new_model[state] /= np.sum(new_model[state])
        else:
            new_model[state] = self.model[state].copy()

    self.model = new_model

    # обновляем значение eps
    self.eps = min(self.max_eps, self.eps * self.k)
    print('Новое значение eps', self.eps)
    return None
```

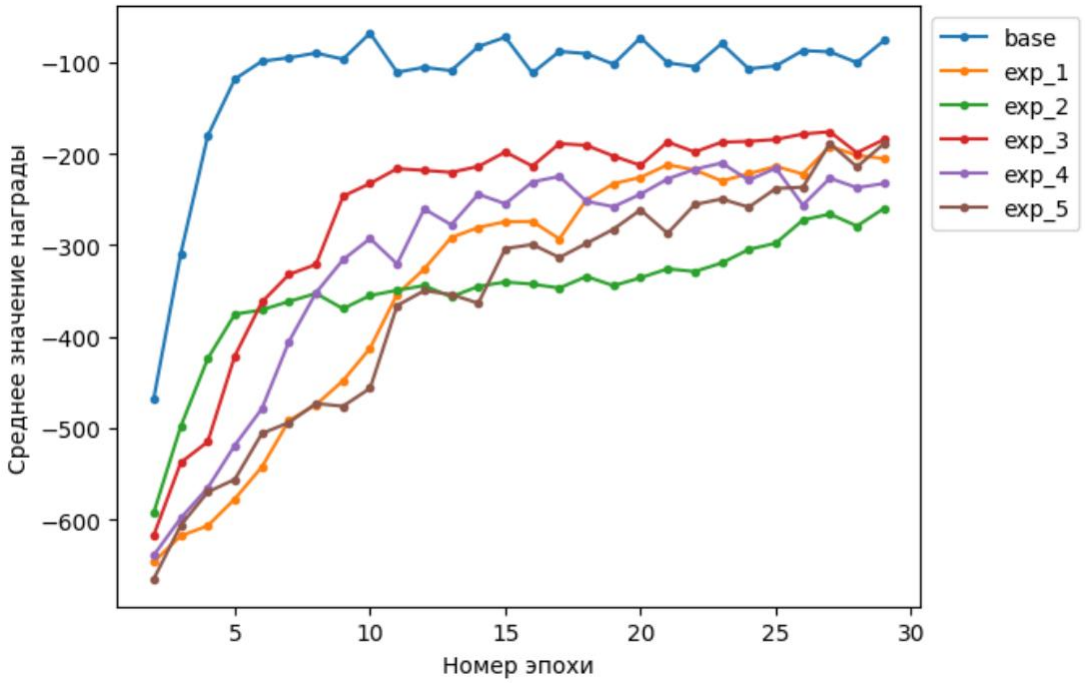
Результаты будем сравнивать с базовой траекторией и параметрами для эксперемента.
q_param=0.8,trajectory_n=300, iteration_n=30, max_length=300,

Номер эксперимента	eps	k	Max_eps	Mean(total_rewards)	Max(total_rewards)
0й (база)				-76.29	15
1	0.1	1.1	0.7	-146.6	9
2	0.2	1.1	0.9	-189.06	15
3	0.15	1.05	0.5	-220.4	12

Проведя буквально 3 эксперемента, я был недоволен результатом модели. Я решил переписать способ обновления eps

```
# обновляем значение eps
self.eps = min(self.max_eps, self.eps + self.k)
print('Новое значение eps', self.eps)
return None
```

Номер эксперимента	eps	k	Max_eps	Mean(total_rewards)	Max(total_rewards)
0й (база)				-76.29	15
1	0.1	0.1	0.7	-259.69	12
2	0.15	0.02	0.8	-205.2	12
3	0.15	0.04	0.7	-184.43	10
4	0.15	0.03	0.86	-232.55	12
5	0	0.1	0.6	-188.92	9



К сожалению, не один из моих эксперементов не смог превзойти базовую модель обучения. По графику мы видим, что модели не дошли еще до «плата обучения» и возможно с большим кол-вом эпох они смогли бы показать более лучшие результаты.

Общий вывод:

Рассмотрев 3 способа преобразования базовой модели обучения, могу сделать вывод что самый простой способ - это policy smoothing. В этом методе получилось наибольшее количество моделей со случайно взятыми параметрами более успешными, чем базовая модель обучения.