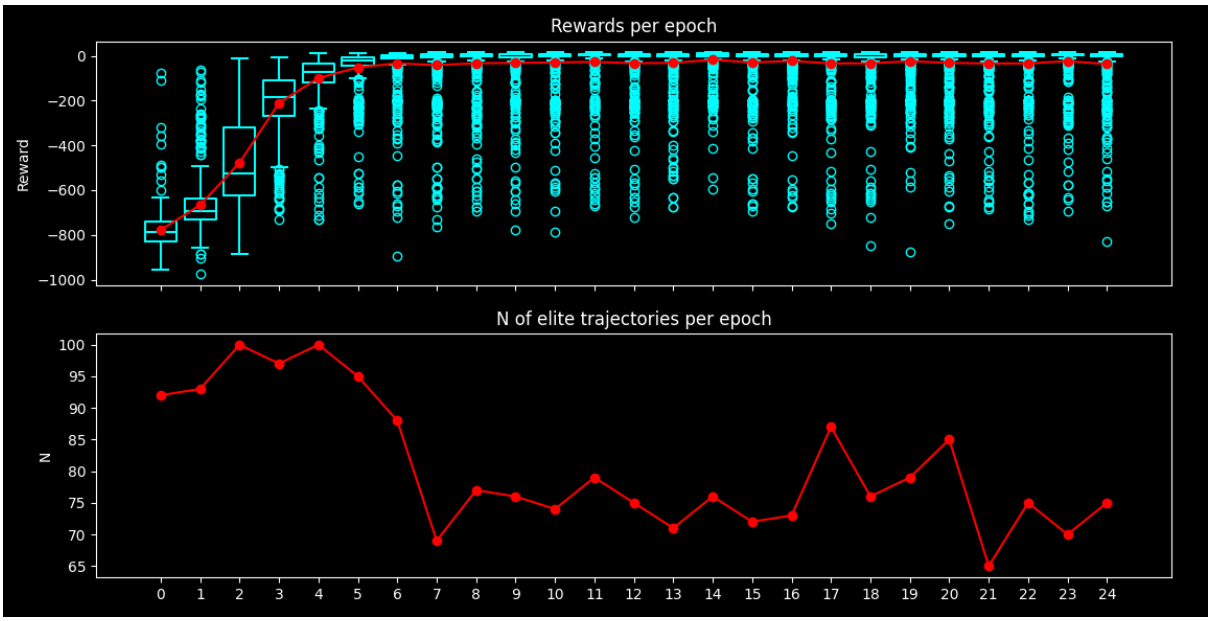
Задание 1



Код реализации алгоритма кросс-энтропии для Taxi-v3 идентичен показанному на практическом занятии. Гиперпараметры, с которыми пришлось экспериментировать - q_param и traj_per_epoch. Так как среда сложнее лабиринта, агенту нужно больше траекторий для того чтобы обучиться. Поэтому я постепенно увеличивал количество траекторий за эпоху и менял значение q. Графики выше - значение средней награды за эпоху (красная линия на графике сверху), распределение нарград на каждой эпохе (боксплоты на графике сверху) и количество элитных траекторий на каждой эпохе (график снизу).
График получился при следующих гиперпараметрах:

```
def main():
    STATE_N = 500
    ACTION_N = 6
    env = gym.make("Taxi-v3")
    agent = crossEntropyAgent(STATE_N, ACTION_N)
    trajecotry_len = 500
    q_param = 0.8
    traj_per_epoch = 500
    n_epochs = 25

    history = train(env, agent, n_epochs,
                    trajecotry_len, traj_per_epoch, q_param)
    last_trajecotry = get_trajectory(env, agent,
                                     trajecotry_len, viz=True
    print(last_trajecotry.total_reward)
    env.close()
    history.show()
    history.print()
```

Награда на последней траектории = 8.

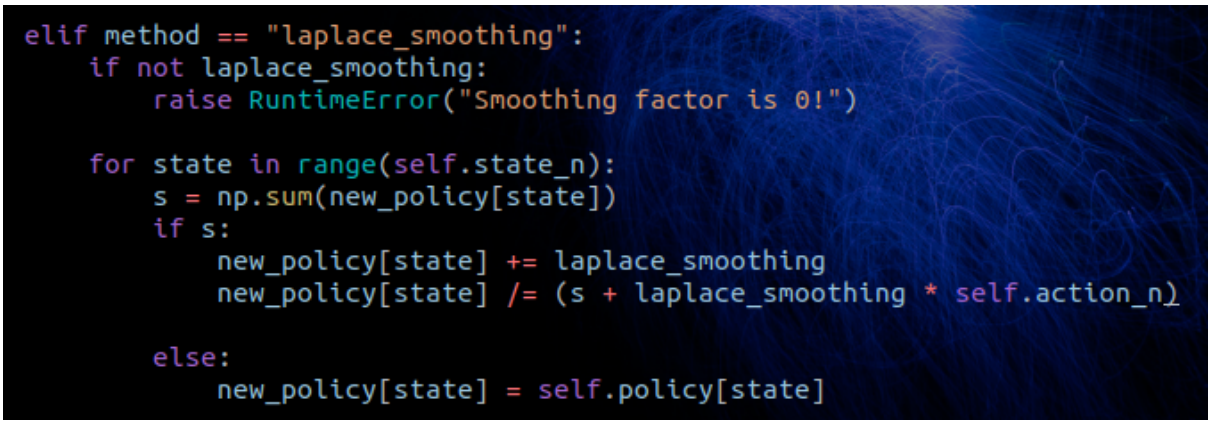
Как видно на боксплотах, большая часть траекторий (примерно > 70% траекторий на эпохе) получали награду с нижним пределом около -40. Однако, так же как видно на боксплотах, на каждой эпохе было много негативных аутлаеров, что лишний раз показывает, что с помощью алгоритма сложно получить политику, на основе которой будут получаться оптимальные траектории в стохастической среде.

Задание 2

Код для Лапласовского сглаживания и Полиси сглаживания я имплементировал как параметры в методе update_policy.

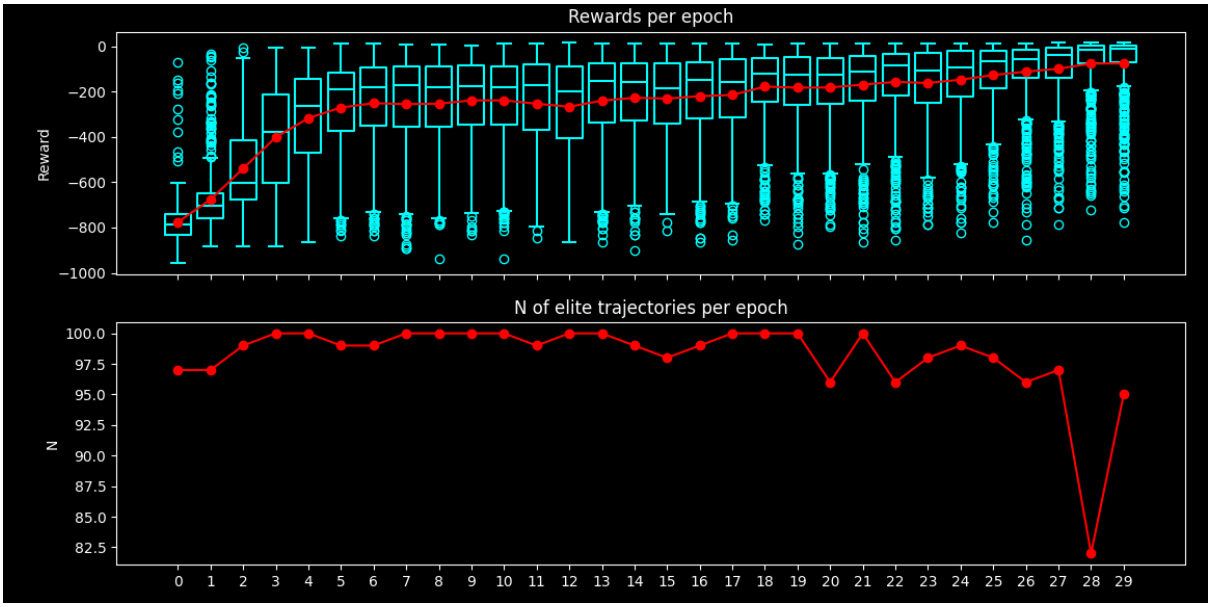
Laplace smoothing

Код для лапласовского сглаживания



```
elif method == "laplace_smoothing":
    if not laplace_smoothing:
        raise RuntimeError("Smoothing factor is 0!")

    for state in range(self.state_n):
        s = np.sum(new_policy[state])
        if s:
            new_policy[state] += laplace_smoothing
            new_policy[state] /= (s + laplace_smoothing * self.action_n)

        else:
            new_policy[state] = self.policy[state]
```

Лапласовское сглаживание: награды и количество элитных траекторий



Награда на последней траектории = -8.

Гиперпараметры идентичны первому заданию:

```
trajecotry_len = 500
q_param = 0.8
traj_per_epoch = 500
n_epochs = 30

method = "laplace_smoothing"
laplace_smoothing = 1
```

smoothing_factor был выбран равным единице. Алгоритм с Лапласовским сглаживанием сходится дольше, чем без него, что в какой то степени логично, так как вероятность неверных действий никогда не будет равна 0, и наш метод имплементации с numpy.random.choice вместо np.argmax при выборе действия подразумевает, что всегда будет вероятность выбора действия с малым шансом выбора.

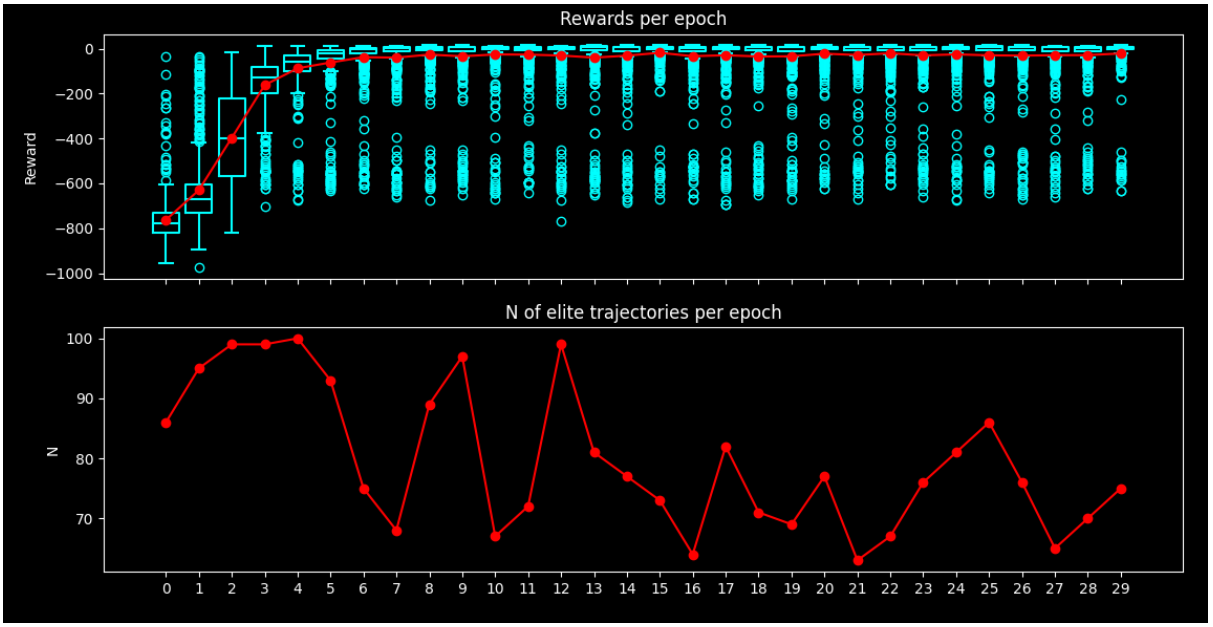
Как видно на графике, распределение наград на каждой эпохе имеет большие значение квантилей, чем без Лапласовского сглаживания. Возможно это будет сведено к минимуму после достаточного числа эпох обучения. По крайней мере, для меня этот график подтверждает, что я имплементировал лапласовское сглаживание корректно.

Policy smoothing

Код для Политического сглаживания

```
elif method == "policy_smoothing":
    if not policy_lambda:
        raise RuntimeError("Policy lambda is 0.0!")
    new_policy *= policy_lambda
    self.policy *= (1 - policy_lambda)
    new_policy += self.policy
    s = np.sum(new_policy, axis=1).reshape(-1, 1)
    new_policy /= s
```

Мне почему-то кажется, что этот метод сглаживания наиболее универсален.
График наград и элитных траекторий:



Гиперпараметры:

```
trajecotry_len = 500
q_param = 0.8
traj_per_epoch = 500
n_epochs = 30

method = "policy_smoothing"
policy_lambda = 0.85
```

Распределение наград на каждой эпохе показывает, что с данной средой Политическое сглаживание работает лучше Лапласовского. График наград почти идентичен наградам без каких либо алгоритмов сглаживания.

Задание 3

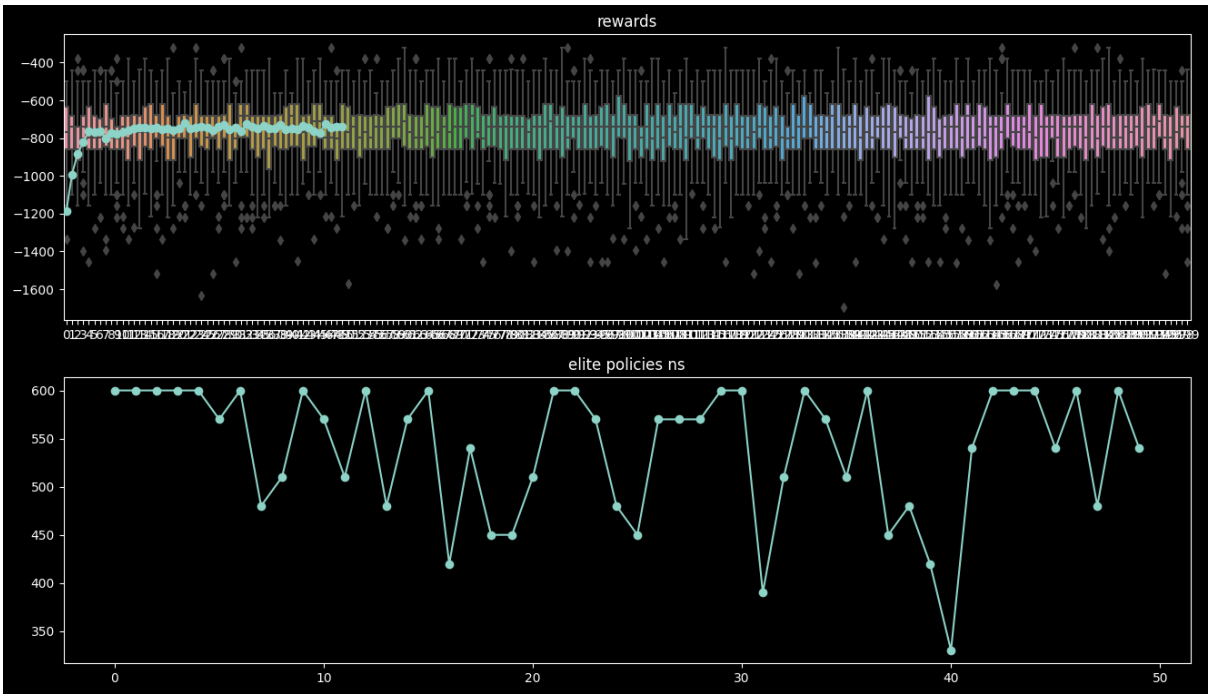
Модификация алгоритма для стохастических сред была имплементирована несколькими дополнительными функциями.
Сэмплирование детерминированных политик (сохранял просто как лист с действиями, индекс значения - соответствующее состояние)

```
def sample_deterministic_policies(agent: crossEntropyAgent,
                                  num_samples: int):
    deterministic_policies = []
    for _ in range(num_samples):
        deterministic_policy = []
        for state in range(agent.state_n):
            action = agent.get_action(state)
            deterministic_policy.append(action)

        deterministic_policies.append(deterministic_policy)

    return deterministic_policies
```

Также я реализовал два метода апдейта политики агента: на одном, как указано в задании, просто итерировался по наборам траекторий из тех, полученных на основе детерминированных политик, и другой - отбирать элитные политики целиком.

Ни один из реализованных мной вариантов не дал желаемого результата. Апдейт политики агента на основе элитных траекторий показал себя хуже, апдейт на основе политик целиком.

График для апдейта



Как вижно по графику наград, обучение агента стагнировало на награде равной примерно -600.
Гиперпараметры:

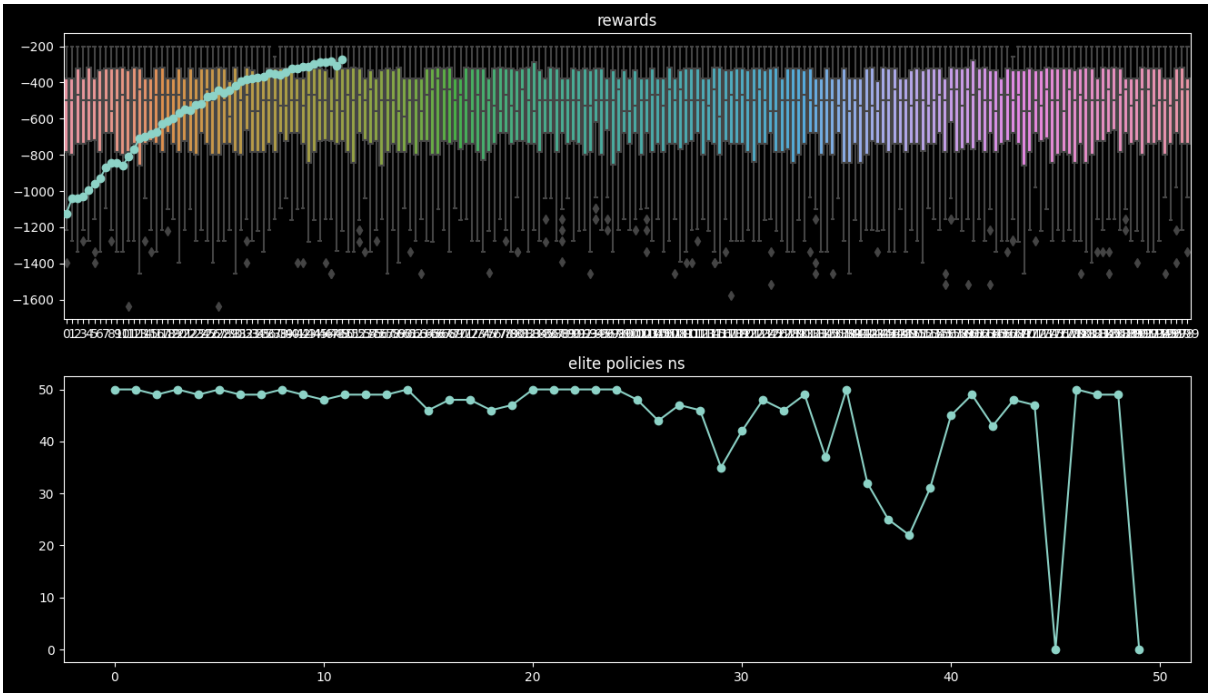
```
n_epochs = 50
n_policies = 200
traj_per_policy = 30
traj_len = 200
q_param = 0.9
```

(при меньшем значении q результат хуже, при большем количестве политик на эпоху сильного улучшения не видно, а процесс обучения занимает дольше)
Есть предположение, что данная модификация хуже показывает себя с данной средой так как среда недостаточно стохастическая. Обучение занимает дольше времени, в сравнении со всеми предыдущими модификациями алгоритма. Также есть предположение, что с данной модификацией алгоритм рано или поздно сойдется, но сэмплировать думаю придется достаточно много (долго).

Метод с апдейтом политики на основе элитных политик:

```
def update_policy_by_policies(self, elite_policies: list[list[int]]) -> None:
    new_policy = np.zeros((self.state_n, self.action_n))
    for p in elite_policies:
        for state, action in enumerate(p):
            new_policy[state][action] += 1
    for state in range(self.state_n):
        s = np.sum(new_policy[state])
        if s:
            new_policy[state] /= s
        else:
            new_policy[state] = self.policy[state]
    self.policy = new_policy
```

Эта имплементация показала себя лучше, чем метод апдейта просто по траекториям. Полагаю, что это связано с тем, что в данном варианте модификации не идет упор на какое то конкретное выбранное действие.
Например, при апдейте по траекториям, у нас есть 30 траекторий на каждую политику (гиперпараметр, значение выбрано мной). В каждой из траекторий есть какие то неоптимальные действия. Из-за того, что при апдейте одно и то же действие мы учитываем больше одного раза, вероятность неоптимального действия в политике становится выше. Это недостаток. Его можно попробовать обойти сглаживанием разобранным выше. Но я почему то решил вот апдейтить политику на основе политик целиком, так как тут хотя бы каждое действие будет учитываться один раз. Результат был немного лучше:



Тут распределение наград от примерно -300 до примерно - 800, что лучше, чем при апдейте по траекториям (там было от -700 до -900). Среднее значение награды под конец подобралось к -200. Так как среднее значение награды не вышло на плато, предполагаю, что при достаточно долгом обучении оно выйдет на оптимальную политику.

Вывод

В целом, в данной среде алгоритм кросс-энтропии без модификаций показал себя чуть ли не лучшим образом (учитывая время, которое тратится на обучение агента). Лапласовское сглаживание ухудшило процесс обучения алгоритма (оптимальная траектория достигается за большее количество итераций), Политическое сглаживание с данной средой работает так же хорошо как и кросс-энтропия без модификаций. Модификация для стохастических сред требует гораздо больше времени на обучение агента, что я рассматриваю как недостаток. Стоит учитывать, что такси – довольно простая среда, и данная модификация может показывать себя лучше на средах более сложных.