Лабораторная работа №3.

Марковский процесс принятия решений (MDP)

1. Подготовительные действия

В работе будет использован внешний модель mdp.py, который следует разместить в папке проекта с данной лабораторной работой. Ссылка для скачивания файла: https://github.com/yandexdataschool/Practical_RL/blob/master/week02_value_based/mdp.py

```
import sys, os

transition_probs = {
    's0': {
        'a0': {'s0': 0.5, 's2': 0.5},
        'a1': {'s2': 1}
    },
    's1': {
        'a0': {'s0': 0.7, 's1': 0.1, 's2': 0.2},
        'a1': {'s1': 0.95, 's2': 0.05}
    },
    's2': {
        'a0': {'s0': 0.4, 's2': 0.6},
        'a1': {'s0': 0.3, 's1': 0.3, 's2': 0.4}
    }
}
rewards = {
    's1': {'a0': {'s0': +5}},
    's2': {'a1': {'s0': -1}}
}

from mdp import MDP
mdp = MDP(transition_probs, rewards, initial_state='s0')
```

Теперь можно использовать MDP аналогично Gym среде.

```
print('initial state =', mdp.reset())
next_state, reward, done, info = mdp.step('a1')
print('next_state = %s, reward = %s, done = %s' % (next_state, reward, done))
```

Вывод:

initial state = s0

next_state = s2, reward = 0.0, done = False

У модуля также есть другие методы, которые понадобятся для итераций по значению.

```
print("mdp.get_all_states =", mdp.get_all_states())
print("mdp.get_possible_actions('s1') = ", mdp.get_possible_actions('s1'))
print("mdp.get_next_states('s1', 'a0') = ", mdp.get_next_states('s1', 'a0'))
print("mdp.get_reward('s1', 'a0', 's0') = ", mdp.get_reward('s1', 'a0',
's0'))
print("mdp.get_transition_prob('s1', 'a0', 's0') = ",
mdp.get_transition_prob('s1', 'a0', 's0'))
```

Вывод:

mdp.get_all_states = ('s0', 's1', 's2')
mdp.get_possible_actions('s1') = ('a0', 'a1')
mdp.get_next_states('s1', 'a0') = {'s0': 0.7, 's1': 0.1, 's2': 0.2}
mdp.get_reward('s1', 'a0', 's0') = 5
mdp.get_transition_prob('s1', 'a0', 's0') = 0.7

Визуализация графа (не обязательно)

```
from mdp import has_graphviz
from IPython.display import display
print("Graphviz available:", has_graphviz)

if has_graphviz:
    from mdp import plot_graph, plot_graph_with_state_values,
plot_graph_optimal_strategy_and_state_values
    display(plot_graph(mdp))
```

2. Итерации по значениям

Итерации по значениям (Value Iterations или VI). Вот псевдокод для VI:

1. Initialize $V^{(0)}(s) = 0$, for all s
2. For $i = 0, 1, 2, \ldots$
3. $V_{(i+1)}(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')]$, for all s

Во-первых, давайте напишем функцию для вычисления функции значения состояния-действия Q^{π}, определенной следующим образом:

$$Q_i(s,a) = \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] \quad (1)$$

```
def get_action_value(mdp, state_values, state, action, gamma):
    """ Вычисляет Q(s,a) из формулы (1) описания лабораторной работы """

    <ВАШ КОД>

    return <ВАШ КОД>
```

```
import numpy as np
test_Vs = {s: i for i, s in enumerate(sorted(mdp.get_all_states()))}
assert np.isclose(get_action_value(mdp, test_Vs, 's2', 'a1', 0.9), 0.69)
assert np.isclose(get_action_value(mdp, test_Vs, 's1', 'a0', 0.9), 3.95)
```

Используя Q(s,a), теперь мы можем определить «следующую» V(s) для итерации значения.

$$V_{(i+1)}(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] = \max_a Q_i(s,a) \quad (2)$$

```
def get_new_state_value(mdp, state_values, state, gamma):
    """ Вычисление следующего V(s) по формуле (2). В процессе не меняйте
state_values. """
    if mdp.is_terminal(state):
        return 0

    < ВАШ КОД >

    return < ВАШ КОД >
```

Наконец, объединим все написанные функции в рабочий алгоритм итерации значений.

```
# параметры
gamma = 0.9             # дисконт MDP
```

```
num_iter = 100          # максимальное число итераций, за исключением
инициализации
# Останавливаем VI если новые значения ближе к старым менее чем:
min_difference = 0.001

# инициализация V(s)
state_values = {s: 0 for s in mdp.get_all_states()}

if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))

for i in range(num_iter):

    # Вычисление новых начений состояния, используя ранее написанные функции.
    # Они имеют формат словаря {state : float V_new(state)}
    new_state_values = <ВАШ КОД>

    assert isinstance(new_state_values, dict)

    # Вычисление отклонений
    diff = max(abs(new_state_values[s] - state_values[s])
               for s in mdp.get_all_states())
    print("iter %4i   |   diff: %6.5f   |   " % (i, diff), end="")
    print('   '.join("V(%s) = %.3f" % (s, v) for s, v in
state_values.items()))
    state_values = new_state_values

    if diff < min_difference:
        print("Terminated")
        break
```

```
if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))

print("Final state values:", state_values)

assert abs(state_values['s0'] - 3.781) < 0.01
assert abs(state_values['s1'] - 7.294) < 0.01
assert abs(state_values['s2'] - 4.202) < 0.01
```

Теперь давайте используем эти V∗(s) для поиска оптимальных действий в каждом состоянии.

$$\pi^*(s) = argmax_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] = argmax_a Q_i(s,a) \quad (3)$$

Единственное отличие от V(s) в том, что здесь мы берем не max, а argmax: найти такое действие с максимальным Q(s,a).

```
def get_optimal_action(mdp, state_values, state, gamma=0.9):
    """ Находит оптимальное действие используя формулу (3). """
    if mdp.is_terminal(state):
        return None

    <ВАШ КОД>

    return <ВАШ КОД>
```

```
assert get_optimal_action(mdp, state_values, 's0', gamma) == 'a1'
assert get_optimal_action(mdp, state_values, 's1', gamma) == 'a0'
assert get_optimal_action(mdp, state_values, 's2', gamma) == 'a1'
```

```
assert get_optimal_action(mdp, {'s0': -1e10, 's1': 0, 's2': -2e10}, 's0',
0.9) == 'a0', \
    "Убедитесь, что вы правильно обрабатываете отрицательные значения Q
произвольной величины."
assert get_optimal_action(mdp, {'s0': -2e10, 's1': 0, 's2': -1e10}, 's0',
0.9) == 'a1', \
    "Убедитесь, что вы правильно обрабатываете отрицательные значения Q
произвольной величины."
```

```
# Измерение среднего вознаграждения агента
s = mdp.reset()
rewards = []
for _ in range(10000):
    s, r, done, _ = mdp.step(get_optimal_action(mdp, state_values, s, gamma))
    rewards.append(r)

print("average reward: ", np.mean(rewards))

assert(0.40 < np.mean(rewards) < 0.55)
```

Вывод:

iter 0 | diff: 3.50000 | V(s0) = 0.000 V(s1) = 0.000 V(s2) = 0.000
iter 1 | diff: 0.64500 | V(s0) = 0.000 V(s1) = 3.500 V(s2) = 0.000
iter 2 | diff: 0.58050 | V(s0) = 0.000 V(s1) = 3.815 V(s2) = 0.645
…
iter 57 | diff: 0.00110 | V(s0) = 3.779 V(s1) = 7.292 V(s2) = 4.200
iter 58 | diff: 0.00099 | V(s0) = 3.780 V(s1) = 7.293 V(s2) = 4.201
Terminated
Final state values: {'s0': 3.7810348735476405, 's1': 7.294006423867229, 's2': 4.202140275227048}
average reward: 0.4553

3. Среда Frozen lake

```
#Frozen Lake Env
from mdp import FrozenLakeEnv
mdp = FrozenLakeEnv(slip_chance=0)

mdp.render()
```

Вывод:

*FFF
FHFH
FFFH
HFFG

```
def value_iteration(mdp, state_values=None, gamma=0.9, num_iter=1000,
min_difference=1e-5):
    """ выполняет шаги итерации значения num_iter, начиная с state_values. То
же, что и раньше, но в функции """
    state_values = state_values or {s: 0 for s in mdp.get_all_states()}
    for i in range(num_iter):

        # Вычислите новые значения состояния, используя функции, которые вы
определили выше. Это должен быть dict {state: new_V(state)}
        new_state_values = <ВАШ КОД>

        assert isinstance(new_state_values, dict)
```

```
        # Вычисление отклонений
        diff = max(abs(new_state_values[s] - state_values[s])
                   for s in mdp.get_all_states())

        print("iter %4i   |   diff: %6.5f   |   V(start): %.3f " %
              (i, diff, new_state_values[mdp._initial_state]))

        state_values = new_state_values
        if diff < min_difference:
            break

    return state_values
```

```
state_values = value_iteration(mdp)
```

```
s = mdp.reset()
mdp.render()
for t in range(100):
    a = get_optimal_action(mdp, state_values, s, gamma)
    print(a, end='\n\n')
    s, r, done, _ = mdp.step(a)
    mdp.render()
    if done:
        break
```

Визуализация. Обычно интересно посмотреть, что алгоритм на самом деле умеет. Для этого мы нанесем на график функции значений состояния и оптимальные действия на каждом шаге VI.

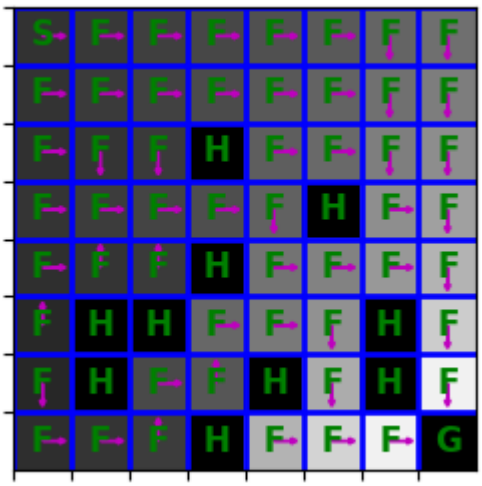
```
import matplotlib.pyplot as plt

def draw_policy(mdp, state_values):
    plt.figure(figsize=(3, 3))
    h, w = mdp.desc.shape
    states = sorted(mdp.get_all_states())
    V = np.array([state_values[s] for s in states])
    Pi = {s: get_optimal_action(mdp, state_values, s, gamma) for s in states}
    plt.imshow(V.reshape(w, h), cmap='gray', interpolation='none', clim=(0,
1))
    ax = plt.gca()
    ax.set_xticks(np.arange(h)-.5)
    ax.set_yticks(np.arange(w)-.5)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    Y, X = np.mgrid[0:4, 0:4]
    a2uv = {'left': (-1, 0), 'down': (0, -1), 'right': (1, 0), 'up': (0, 1)}
    for y in range(h):
        for x in range(w):
            plt.text(x, y, str(mdp.desc[y, x].item()),
                     color='g', size=12,  verticalalignment='center',
                     horizontalalignment='center', fontweight='bold')
            a = Pi[y, x]
            if a is None:
                continue
            u, v = a2uv[a]
            plt.arrow(x, y, u*.3, -v*.3, color='m',
                      head_width=0.1, head_length=0.1)
    plt.grid(color='b', lw=2, ls='-')
    plt.show()
```

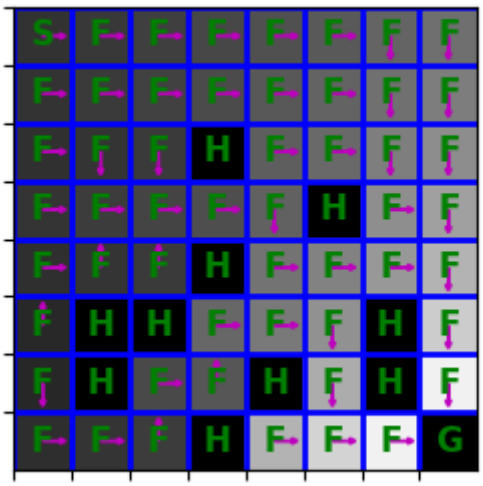
```
state_values = {s: 0 for s in mdp.get_all_states()}

for i in range(10):
    print("after iteration %i" % i)
    state_values = value_iteration(mdp, state_values, num_iter=1)
    draw_policy(mdp, state_values)
# please ignore iter 0 at each step
```

```
from IPython.display import clear_output
from time import sleep
mdp = FrozenLakeEnv(map_name='8x8', slip_chance=0.1)
state_values = {s: 0 for s in mdp.get_all_states()}

for i in range(30):
    clear_output(True)
    print("after iteration %i" % i)
    state_values = value_iteration(mdp, state_values, num_iter=1)
    draw_policy(mdp, state_values)
    sleep(0.5)
# please ignore iter 0 at each step
```

Вывод:

after iteration 0
iter 0 | diff: 1.00000 | V(start): 0.000
after iteration 1
iter 0 | diff: 0.90000 | V(start): 0.000
after iteration 2
iter 0 | diff: 0.81000 | V(start): 0.000
…
after iteration 28
iter 0 | diff: 0.00000 | V(start): 0.198
after iteration 29
iter 0 | diff: 0.00000 | V(start): 0.198





Массовые испытания.

```
mdp = FrozenLakeEnv(slip_chance=0)
state_values = value_iteration(mdp)
```

```
total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))

print("average reward: ", np.mean(total_rewards))
assert(1.0 <= np.mean(total_rewards) <= 1.0)
print("Well done!")
```

```
# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.1)
state_values = value_iteration(mdp)

total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))

print("average reward: ", np.mean(total_rewards))
assert(0.8 <= np.mean(total_rewards) <= 0.95)
print("Well done!")
```

```
# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.25)
state_values = value_iteration(mdp)

total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))

print("average reward: ", np.mean(total_rewards))
assert(0.6 <= np.mean(total_rewards) <= 0.7)
print("Well done!")
```

```
# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.2, map_name='8x8')
state_values = value_iteration(mdp)

total_rewards = []
```

```
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))

print("average reward: ", np.mean(total_rewards))
assert(0.6 <= np.mean(total_rewards) <= 0.8)
print("Well done!")
```

Вывод (пример по одному из тестов):

```
iter   0   |   diff: 0.75000   |   V(start): 0.000
iter   1   |   diff: 0.50625   |   V(start): 0.000
…
iter  19   |   diff: 0.00003   |   V(start): 0.325
iter  20   |   diff: 0.00002   |   V(start): 0.325
iter  21   |   diff: 0.00001   |   V(start): 0.325
average reward:  0.631
Well done!
```

Задания к лабораторной работе

1. Написать свой код согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>

```
<ВАШ КОД: >
```

2. Запустить полученную программу, получить ожидаемые выходные данные.

Требование к отчету

1. В качестве отчета принимается Python файл с кодом.
2. Код должен быть в достаточной мере прокомментирован.
3. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip