FrozenLake 4

Wczytaj też plik **plot_utils.py**.

```
import gym
import numpy as np
from plot_utils import plot_values

env = gym.make("FrozenLake-v0",is_slippery=False)
```

Objaśnienie algorytmu

Algorytm wygląda następująco:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\left[r + \gamma V(s')\right]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

W algorytmie można wyróżnić dwa główne bloki.

Blok 1

Algorytm wylicza **wartość zwrotów *V(s)*** dla wszystkich stanów ***s*** przy zadanej **polityce deterministycznej** oznaczonej ***pi*** (wyliczenie ***V*** odbywa się w **punkcie 2** algorytmu).

Blok 2

Algorytm znajduje **politykę deteministyczną *pi*** na podstawie wyliczonych wcześniej wartości ***V(s)***(**punkt 3** algorytmu). Warto zwrócić uwagę, że zarówno ***V(s)*** jak i polityka ***pi*** są początkowo dowolne (***punkt 1*** algorytmu).

To co ciekawe odbywa się w nastepującej **pętli**:

Algorytm wylicza ***V(s)*** dla zadanej **polityki deterministycznej *pi***, a następnie wyliczone wartości ***V(s)*** wykorzystuje **do modyfikacji polityki *pi***. Zmodyfikowana polityka służy do ponownego wyliczenia ***V(s)*** itd.

W efekcie takich wzajemnych modyfikacji ***V*** i ***pi*** znaleziona zostaje **polityka optymalna** (najlepsza - gwarantująca najlepsze zwroty) dla danego problemu.... ***MAGIA*** :)

Warto zwrócić uwagę na **punkt 3** w algorytmie:

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
      $old\text{-}action \leftarrow \pi(s)$
      $\boxed{\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]}$
      If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

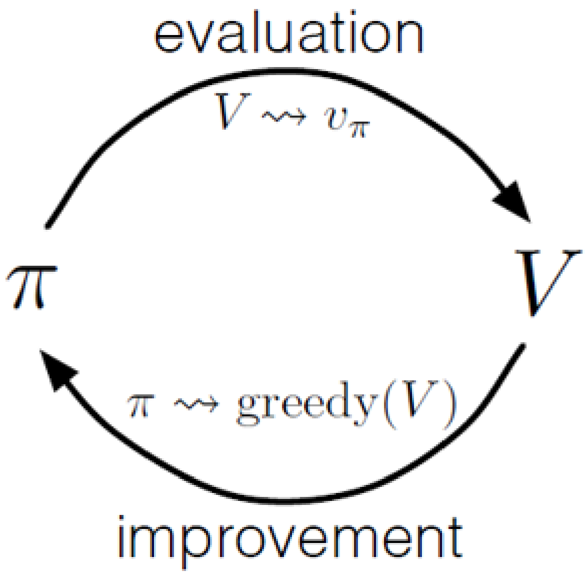
W formule w ramce użyta jest funkcja **argmax**. Jej definicja jest następująca:

$$\underset{x \in S \subseteq X}{\arg\max}\, f(x) := \{x \mid x \in S \wedge \forall y \in S : f(y) \leq f(x)\}.$$

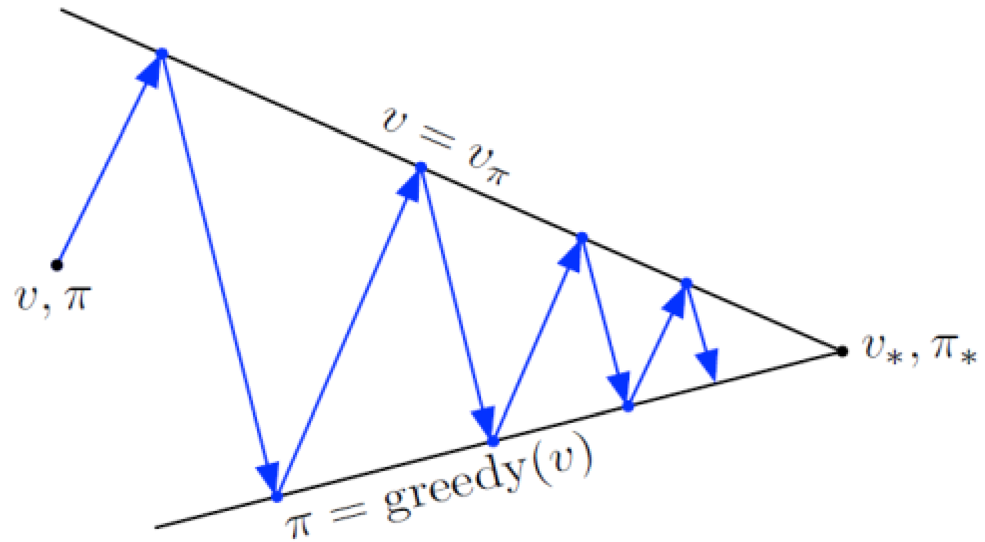
argmax f(x) zwraca wartość argumentu ***x*** dla którego funkcja ***f*** osiąga **wartość maksymalną**.

Linijka oznaczona powyżej **czerwoną** ramką oznacza, że **nowa polityka *pi*** przypisuje stanowi ***s*** akcję, która (dla **aktualnych wartości *V***) daje **największy zwrot**! Czyli stosowana jest tutaj **strategia zachłanna (greedy)**.

Petlę w algotymie możemy zobrazować następująco:



Wynikiem działania algorytmu jest **optymalna polityka** (deterministyczna) **pi*** i **V*** dla tej polityki.



Polityka deterministyczna

W implemntacji algorytmu będziemy stosowali **politykę deteministyczną**. Jest to polityka, która każdemu stanowi ***s*** przypisuje akcję ***a***, która ma być wykonana w tym stanie. Możemy ją zdefiniować następująco (***env.nS*** to liczba stanów w środowisku ***env***):

```
pi = np.random.randint(0, env.nA, size=env.nS)

print(pi)
```

```
[1 1 0 1 0 3 3 0 2 0 0 1 3 1 2 1]
```

Polityka ta dla każdego stanu **s** określa akcję.

Przykład: akcja wykonana w stanie 4 (stany numerowane są od 0 do 15):

```
pi[4]
```

```
0
```

Wyliczenie V dla zadanej polityki

Zajmijmy się **punktem 2** algorytmu czyli wyliczeniem ***V(s)*** dla zadanej **polityki deterministycznej *pi***. Problemu tego **dla polityki stochastycznej** dotyczyło **zadanie 3** z **RL_lab_4.pdf**.

Polecenie 1 (do uzupełnienia)

Uzupełnij poniższą definicję funkcji zwracającej ***V*** dla zadanej **polityki deterministycznej *policy***.

```
#def det_policy_evaluation(env, policy, gamma=0.9, theta=1e-8):
#def det_policy_evaluation(env, policy, gamma=0.99, theta=1e-8):
def det_policy_evaluation(env, policy, gamma=0.1, theta=1e-8):
    V = np.zeros(env.nS)

    while True:
        delta = .0
        for state in range(env.nS):
            Vs = 0
            for next_state in range(len(env.P[state][policy[state]])):

                prob, next_state, reward, done = env.P[state][policy[state]][next_state]
                Vs+= prob* (reward + gamma *V[next_state])
                #DO UZUPEŁNIENIA
```

```
            delta = max(delta, np.abs(V[state] - Vs))
            V[state] = Vs
        if delta < theta:
            break

    return V
```

UWAGA: warto zwrócić uwagę, gdzie w powyższym kodzie użyta jest polityka deterministyczna **policy**:

```
for next_state in range(len(env.P[state][policy[state]])):
    #deterministic policy

    prob, next_state, reward, done = env.P[state][policy[state]][next_state]

    #DO UZUPEŁNIENIA
```

Testujemy działanie funkcji dla **polityki deterministycznej *pi***:

```
pi = np.array([1,0,0,0,1,0,0,0,2,2,1,0,0,0,2,0])
V = det_policy_evaluation(env,pi)
print(V)
```

```
[1.e-05 1.e-06 1.e-07 1.e-08 1.e-04 0.e+00 0.e+00 0.e+00 1.e-03 1.e-02
 1.e-01 0.e+00 0.e+00 0.e+00 1.e+00 0.e+00]
```

Wartości *V(s)* można przedstawić graficznie korzystając z funkcji **plot_values()** z **biblioteki plot_utils**:

```
plot_values(V)
```

Znalezienie polityki dla zadanego V metodą zachłanną

Zajmijmy się teraz **punktem 3** algorytmu czyli znalezieniem **polityki deterministycznej *pi*** dla danego ***V***. Interesuje nas następujący fragment kodu:

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_a \boxed{\sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]}$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Warto zauważyć, że w powyższym kodzie nie tylko jest znajdowana nowa polityka, ale **także jest ona porównywana z dotychczasową**. I jeżeli obie są takie same, wówczas algorytm kończy działanie. W przeciwnym razie następuje powrót do **punktu 2**.

Polecenie 2 (do uzupełnienia)

Znalezie polityki dla danego ***V*** można "zamknąć" w funkcji **det_policy_iteration**, która jako argumenty otrzyma ***env***, ***V*** i wartość ***gamma***, a zwróci nam wyliczoną politykę. Warto zwrócić uwagę na to, że **formuła oznaczona powyżej czerwoną ramką** była wykorzystywana w **Zadaniu 1 z części 5**. Czyli możemy wykorzystać funkcję już zdefiniowaną (w **FrozeLake_3.ipynb**). Uzupełnij jej definicję poniżej:

```
#def Q_from_V(env, V, s, gamma=0.99):
#    Q = np.zeros(env.nA)
#
#        #DO UZUPEŁNIENIA
#
#    return Q
#
#
#
```

```
def Q_from_V(env, V, s, gamma=0.99):
    Q = np.zeros(env.nA)

    for action in range(env.nA):
      action_value = 0
      for i in range(len(env.P[s][action])):
        prob, next_state, r, _ = env.P[s][action][i]
        action_value += prob * (r + gamma * V[next_state])
     # Q.append(action_value)
      Q[action]= action_value
   # Q = np.argmax(np.asarray(action_values))




        #DO UZUPEŁNIENIA

    return Q
```

Czas na funkcję **det_policy_iteration**:

```
def det_policy_iteration(env, V, gamma=0.99):
    policy = np.zeros([env.nS])

    for state in range(env.nS):
     policy[state] = np.argmax(Q_from_V(env, V, state))
       #DO UZUPEŁNIENIA

    return policy
```

Przetestuj działanie funkcji **det_policy_iteration** dla poniższego ***V***:

```
V = np.array([0.1,0.5,0.8,0.2,0.2,0.,0.4,0.,0.3,0.5,0.8,0.,0.,0.1,1.,0.])
pi = det_policy_iteration(env,V)
```

```
print(pi)
```

```
[2. 2. 3. 0. 1. 0. 1. 0. 2. 2. 1. 0. 0. 2. 2. 0.]
```

Implementacja algorytmu

W implementowanym algorytmie po znalezieniu nowej polityki następuje **porównanie** jej ze starą polityką:

$$\text{If } \textit{old-action} \neq \pi(s), \text{ then } \textit{policy-stable} \leftarrow \textit{false}$$

Polityki są u nas zapisane **w tablicach**. Musimy mieć zatem **metodę porównującą dwie tablice** i zwracającą **True/False** jeżeli tablice polityk **są/nie są** identyczne.

Polecenie 3 (do uzupełnienia)

Uzupełnij poniższą funkcję pozwalającą porównywać dwie polityki (tablice) będące jej argumentami i zwracającą **True/False**:

```
def compPolicy(p1,p2):
    comp = p1==p2

    return comp.all()
    #DO UZUPEŁNIENIA
```

Przetestuj działanie funkcji dla dwóch przykładowych polityk:

```
pi_1 = np.array([3,2,0,3,0,3,3,0,1,3,3,0,0,3,0,0])
pi_2= np.array([3,2,0,3,0,3,3,0,1,3,3,0,0,3,0,0])
#pi_2 = np.array([3,2,0,3,0,0,1,0,1,3,3,0,0,3,0,0])

print(compPolicy(pi_2,pi_1))
```

```
True
```

Mamy już wszystkie elementy wymagane do implementacji całego **algorytmu**.

Polecenie 4 (do uzupełnienia)

Uzupełnij poniższą petlę tak, aby otrzymać **pełną implementację algorytmu**:

```
pi = np.array([1,0,0,0,1,0,0,0,2,2,1,0,0,0,2,0])
V = np.array([0.1,0.5,0.8,0.2,0.2,0.,0.4,0.,0.3,0.5,0.8,0.,0.,0.1,1.,0.])

while True:

        #DO UZUPEŁNIENIA
  n_V = det_policy_evaluation(env, pi)

  n_pi = det_policy_iteration(env, n_V)
  comp = compPolicy(pi, n_pi)

  if comp:

    break

  else:

    pi = n_pi
```

Po wykonaniu powyższego kodu w zmiennej ***pi*** będzie zapisana **optymalna polityka**. Wypiszmy ją:

```
print(pi)
```

```
[1. 2. 1. 0. 1. 0. 1. 0. 2. 1. 1. 0. 0. 2. 2. 0.]
```

Polecenie 5 (do uzupełnienia)

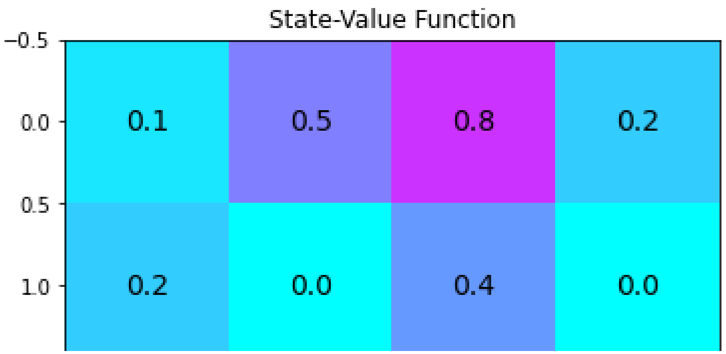
Sprawdź czy **znaleziona polityka** jest optymalna. Odpowiedź uzasadnij.

TUTAJ WPISZ ODPOWIEDŹ:

Tak została znaleziona polityka optymalna, gdyż oczekiwane wartości zwrotów V są

Zobaczmy jak wyglądają **wartości oczekiwane zwrotów V** dla znalezionej polityki:

```
plot_values(V)
```



Polecenie 6 (do uzupełnienia)

Sprawdź czy znaleziona powyżej **polityka optymalna** zmienia się jeżeli przyjmiemy dwie różne wartości parametru **gamma 0.1** i **0.99**.

TUTAJ WPISZ ODPOWIEDŹ:

Przy wartości gamma 0.99 polityka wynosi [1. 2. 1. 0. 1. 0. 1. 0. 2. 1. 1. 0. 0. 2. 2. 0.]

Przy wartości gamma 0.1 polityka wynosi

[1. 2. 1. 0. 1. 0. 1. 0. 2. 1. 1. 0. 0. 2. 2. 0.]

Polityka się nie zmieniła pomimo zmiany wartości gamma

0 s ukończono o 12:32