POLICY EVALUATION

AIM

To develop a Python program to evaluate the given policy.

PROBLEM STATEMENT

The bandit slippery walk problem is a reinforcement learning problem in which an agent must learn to navigate a 7-state environment in order to reach a goal state. The environment is slippery, so the agent has a chance of moving in the opposite direction of the action it takes.

Developed by: Harshavardhan

Register no: 212222240114

Program

```
pip install git+https://github.com/mimoralea/gym-walk#egg=gym-walk
```

```
import warnings ; warnings.filterwarnings('ignore')

import gym, gym_walk
import numpy as np

import random
import warnings

warnings.filterwarnings('ignore', category=DeprecationWarning)
np.set_printoptions(suppress=True)
random.seed(123); np.random.seed(123)
```

```
def print_policy(pi, P, action_symbols=('<', 'v', '>', '^'), n_cols=4, title='Policy:'
    print(title)
    arrs = {k:v for k,v in enumerate(action_symbols)}
    for s in range(len(P)):
        a = pi(s)
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), arrs[a].rjust(6), end=" ")
        if (s + 1) % n_cols == 0: print("|")
```

```
def print_state_value_function(V, P, n_cols=4, prec=3, title='State-value function:'):
    print(title)
    for s in range(len(P)):
        v = V[s]
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), '{}'.format(np.round(v, prec)).rjust(6), end=" ")
        if (s + 1) % n_cols == 0: print("|")

def probability_success(env, pi, goal_state, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        while not done and steps < max_steps:
            state, _, done, h = env.step(pi(state))
            steps += 1
        results.append(state == goal_state)
    return np.sum(results)/len(results)

def mean_return(env, pi, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        results.append(0.0)
        while not done and steps < max_steps:
            state, reward, done, _ = env.step(pi(state))
            results[-1] += reward
            steps += 1
    return np.mean(results)

env = gym.make('SlipperyWalkFive-v0')
P = env.env.P
init_state = env.reset()
goal_state = 6
LEFT, RIGHT = range(2)
P
init_state

state, reward, done, info = env.step(RIGHT)
print("state:{0} - reward:{1} - done:{2} - info:{3}".format(state, reward, done, info)
```

First Policy

```
pi_1 = lambda s: {
    0:LEFT, 1:LEFT, 2:LEFT, 3:LEFT, 4:LEFT, 5:LEFT, 6:LEFT
}[s]
print_policy(pi_1, P, action_symbols=('<', '>'), n_cols=7)

print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format
    probability_success(env, pi_1, goal_state=goal_state)*100,
    mean_return(env, pi_1)))
```

Second Policy

```
pi_2= lambda s:{
    0:LEFT,1:RIGHT,2:LEFT,3:LEFT,4:RIGHT,5:RIGHT,6:LEFT
}[s]

print_policy(pi_2, P, action_symbols=('<', '>'), n_cols=7)
```

POLICY EVALUATION FUNCTION

```
def policy_evaluation(pi, P, gamma=1.0, theta=1e-10):
    prev_V = np.zeros(len(P), dtype=np.float64)
    while True:
      V=np.zeros(len(P),dtype=np.float64)
      for s in range(len(P)):
        for prob, next_state, reward, done in P[s][pi(s)]:
          V[s]+=prob*(reward+gamma*prev_V[next_state]*(not done))
      if np.max(np.abs(prev_V-V))<theta:
        break
      prev_V=V.copy()
    return V
```

Code to evaluate first policy

```
V1 = policy_evaluation(pi_1, P)
print(V1)
print_state_value_function(V1, P, n_cols=7, prec=5)
```

Code to evaluate second policy

```
V2 = policy_evaluation(pi_2, P)
print(V2)
print_state_value_function(V2, P, n_cols=7, prec=5)
```

Code to compare the two policies

```
print(V1>=V2)
if(np.sum(V1>=V2)==7):
  print("The first policy is the better policy")
elif(np.sum(V2>=V1)==7):
  print("The second policy is the better policy")
else:
  print("Both policies have their merits.")
```

OUTPUT:

Mention the first and second policies along with its state value function and compare them

Policy 1:

```
Policy:
|           | 01      < | 02      < | 03      < | 04      < | 05      < |           |
Reaches goal 3.00%. Obtains an average undiscounted return of 0.0300.
```

State value Function 1:

```
[0.         0.00274725 0.01098901 0.03571429 0.10989011 0.33241758
 0.        ]
State-value function:
|           | 01 0.00275 | 02 0.01099 | 03 0.03571 | 04 0.10989 | 05 0.33242 |           |
```

Policy 2:

```
Policy:
|           | 01      > | 02      < | 03      < | 04      > | 05      > |           |
```

State value Function 2:

```
[0.   0.15 0.2  0.35 0.8  0.95 0.  ]
State-value function:
|           | 01   0.15 | 02    0.2 | 03   0.35 | 04    0.8 | 05   0.95 |           |
```

Comparision:

```
[ True False False False False False  True]
The second policy is the better policy
```

RESULT:

Therefore a Python program has been developed to evaluate the two policies.