```
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)

# Reference https://github.com/mimoralea/gym-walk
```

Select cell Ctrl+Shift+S
Copy link to cell

Delete cell Ctrl+M D
Copy cell
Cut cell Ctrl+X

Run the focused cell Ctrl+Enter
Copy to scratch cell

Add a comment Ctrl+Alt+M
Add a form

```
pip install git+https://github.com/mimoralea/gym-w
```

```
Collecting gym-walk
  Cloning https://github.com/mimoralea/gym-walk to /tmp/pi                                 255d2024529bf05d307beafb076
  Running command git clone --filter=blob:none --quiet htt                                 k /tmp/pip-install-quos7o46/gym-walk_268
  Resolved https://github.com/mimoralea/gym-walk to commit 5999016267d6de2f5a63307fb00dfd63de319ac1
  Preparing metadata (setup.py) ... done
Requirement already satisfied: gym in /usr/local/lib/python3.10/dist-packages (from gym-walk) (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym->gym-walk) (1.23.5)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym->gym-walk) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym->gym-walk) (0.0.8)
Building wheels for collected packages: gym-walk
  Building wheel for gym-walk (setup.py) ... done
  Created wheel for gym-walk: filename=gym_walk-0.0.2-py3-none-any.whl size=4054 sha256=00c3fda173ab3f8b47d7326f276b35f07c72a380ed4
  Stored in directory: /tmp/pip-ephem-wheel-cache-xs0ezovr/wheels/24/fe/c4/0cbc7511d29265bad7e28a09311db3f87f0cafba74af54d530
Successfully built gym-walk
Installing collected packages: gym-walk
Successfully installed gym-walk-0.0.2
```

```
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)
```

Slippery Walk Five MDP

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

```
P
```

```
{0: {0: [(0.5000000000000001, 0, 0.0, True),
   (0.3333333333333333, 0, 0.0, True),
   (0.1666666666666666, 0, 0.0, True)],
  1: [(0.5000000000000001, 0, 0.0, True),
   (0.3333333333333333, 0, 0.0, True),
   (0.1666666666666666, 0, 0.0, True)]},
 1: {0: [(0.5000000000000001, 0, 0.0, True),
   (0.3333333333333333, 1, 0.0, False),
   (0.1666666666666666, 2, 0.0, False)],
  1: [(0.5000000000000001, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.1666666666666666, 0, 0.0, True)]},
 2: {0: [(0.5000000000000001, 1, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.1666666666666666, 3, 0.0, False)],
  1: [(0.5000000000000001, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.1666666666666666, 1, 0.0, False)]},
 3: {0: [(0.5000000000000001, 2, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.1666666666666666, 4, 0.0, False)],
  1: [(0.5000000000000001, 4, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.1666666666666666, 2, 0.0, False)]},
 4: {0: [(0.5000000000000001, 3, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.1666666666666666, 5, 0.0, False)],
  1: [(0.5000000000000001, 5, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.1666666666666666, 3, 0.0, False)]},
 5: {0: [(0.5000000000000001, 4, 0.0, False),
   (0.3333333333333333, 5, 0.0, False),
   (0.1666666666666666, 6, 1.0, True)],
  1: [(0.5000000000000001, 6, 1.0, True),
   (0.3333333333333333, 5, 0.0, False),
   (0.1666666666666666, 4, 0.0, False)]},
 6: {0: [(0.5000000000000001, 6, 0.0, True),
   (0.3333333333333333, 6, 0.0, True),
   (0.1666666666666666, 6, 0.0, True)],
  1: [(0.5000000000000001, 6, 0.0, True),
   (0.3333333333333333, 6, 0.0, True),
   (0.1666666666666666, 6, 0.0, True)]}}
```

```
init_state
```

```
3
```

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

```
state:4 - reward:0.0 - done:False - info:{'prob': 0.5000000000000001}
```

```
# 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)
```

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

```
# Find the probability of success and the mean return of the first policy
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)))
```

```
Reaches goal 3.00%. Obtains an average undiscounted return of 0.0300.
```

```
# Create your own policy
pi_2 = lambda s: {
    0:LEFT, 1:RIGHT, 2:LEFT, 3:RIGHT, 4:LEFT, 5:RIGHT, 6:LEFT
}[s]
# Write your code here

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

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

```
# Find the probability of success and the mean return of you your policy
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
    probability_success(env, pi_2, goal_state=goal_state)*100,
    mean_return(env, pi_2)))
#write your code here
```

```
Reaches goal 52.00%. Obtains an average undiscounted return of 0.5200.
```

```
The probability of reaching the goal has increased. While the previous policy yielded only a 3% success the second policy got 52% chances
The average undiscounted return is also increased.
```

```
  File "<ipython-input-20-0df77931204f>", line 1
    The probability of reaching the goal has increased. While the previous policy yielded only a 3% success the second policy got 52
        ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

Policy Evaluation

```
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 the first policy
V1 = policy_evaluation(pi_1, P)
print_state_value_function(V1, P, n_cols=7, prec=5)
```

```
State-value function:
|           | 01 0.00275 | 02 0.01099 | 03 0.03571 | 04 0.10989 | 05 0.33242 |           |
```

```
# Code to evaluate the second policy
V2 = policy_evaluation(pi_2, P)
print_state_value_function(V2, P, n_cols=7, prec=5)
# Write your code here
```

```
State-value function:
|           | 01 0.00676 | 02 0.02703 | 03 0.08784 | 04 0.27027 | 05 0.81757 |           |
```

$$\pi \geq \pi' \text{ if and only if } v_{\pi}(s) \geq v_{\pi'}(s)$$

```
# Comparing the two policies

# Compare the two policies based on the value function using the above equation and find the best policy
```

```
V1
```

```
array([0.        , 0.00274725, 0.01098901, 0.03571429, 0.10989011,
       0.33241758, 0.        ])
```

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

```
State-value function:
|           | 01 0.00275 | 02 0.01099 | 03 0.03571 | 04 0.10989 | 05 0.33242 |           |
```

```
V2
```

```
array([0.        , 0.00675676, 0.02702703, 0.08783784, 0.27027027,
       0.81756757, 0.        ])
```

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

```
State-value function:
|           | 01 0.00676 | 02 0.02703 | 03 0.08784 | 04 0.27027 | 05 0.81757 |           |
```

```
V1>=V2
```

```
array([ True, False, False, False, False, False,  True])
```

```
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.")
```

```
The second policy is the better policy
```

```
def policy_improvement(V, P, gamma=1.0):
    Q = np.zeros((len(P), len(P[0])), dtype=np.float64)
    for s in range (len(P)):
        for a in range(len(P[s])):
          for prob,next_state,reward, done in P[s][a]:
            Q[s][a] += prob * (reward + gamma * V[next_state] * (not done))
    new_pi = lambda s: {s:a for s, a in enumerate(np.argmax(Q,axis=1))}[s]

    return new_pi
```

```
pi_2 = policy_improvement(V1, P)
print_policy(pi_2, P, action_symbols=('<', '>'), n_cols=7)
```

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

```
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
    probability_success(env, pi_2, goal_state=goal_state)*100,
    mean_return(env, pi_2)))
```

```
Reaches goal 97.00%. Obtains an average undiscounted return of 0.9700.
```

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

```
State-value function:
|           | 01 0.66758 | 02 0.89011 | 03 0.96429 | 04 0.98901 | 05 0.99725 |           |
```

```
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.")
```

```
    The second policy is the better policy
```

```
def policy_iteration(P, gamma=1.0, theta=1e-10):
    random_actions = np.random.choice(tuple(P[0].keys()), len(P))
    pi= lambda s:{s:a for s,a in enumerate(random_actions)}[s]
    while True:
      old_pi = {s:pi(s) for s in range (len(P))}
      V = policy_evaluation(pi,P,gamma,theta)
      pi = policy_improvement(V,P,gamma)
      if(old_pi=={s:pi(s) for s in range (len(P))}):
        break
    return V, pi
```

```
optimal_V, optimal_pi = policy_iteration(P)
```

+ Code   + Text

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

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

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

```
    Reaches goal 97.00%. Obtains an average undiscounted return of 0.9700.
```

```
print_state_value_function(optimal_V, P, n_cols=7, prec=5)
```

```
    State-value function:
    |           | 01 0.66758 | 02 0.89011 | 03 0.96429 | 04 0.98901 | 05 0.99725 |           |
```

✓ 0s   completed at 2:31 PM