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

#import gym
import gymnasium as gym
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);

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

Collecting gym-walk
  Cloning https://github.com/mimoralea/gym-walk to /tmp/pip-install-
glm8yil4/gym-walk_1c24d091af4349849602eb919a95eca8
  Running command git clone --filter=blob:none --quiet
https://github.com/mimoralea/gym-walk /tmp/pip-install-glm8yil4/gym-
walk_1c24d091af4349849602eb919a95eca8
  Resolved https://github.com/mimoralea/gym-walk to commit
b915b94cf2ad16f8833a1ad92ea94e88159279f5
  Preparing metadata (setup.py) ... ent already satisfied: gym in
/usr/local/lib/python3.12/dist-packages (from gym-walk) (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in
/usr/local/lib/python3.12/dist-packages (from gym->gym-walk) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from gym->gym-walk) (3.1.1)
Requirement already satisfied: gym-notices>=0.0.4 in
/usr/local/lib/python3.12/dist-packages (from gym->gym-walk) (0.1.0)
Building wheels for collected packages: gym-walk
  Building wheel for gym-walk (setup.py) ... -walk: filename=gym_walk-
0.0.2-py3-none-any.whl size=5377
sha256=514d27712c72300aed41d7a31705a85803be858e2ebbda5b07b908ab3764fdd
d
  Stored in directory:
/tmp/pip-ephem-wheel-cache-3hufv6az/wheels/bf/23/e5/a94be4a90dd18f7ce9
58c21f192276cb01ef0daaf2bc66583b
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="")
```

```python
        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, h = env.reset(seed=123)
        done, steps = 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, h = env.reset(seed=123)
        done, steps = 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)
```

Frozen Lake MDP

```
env = gym.make('FrozenLake-v1')
P = env.unwrapped.P
#init_state, _ = env.reset()
goal_state = 15
LEFT, DOWN, RIGHT, UP = range(4)

# Create your own policy

pi_2 = lambda s: {
    0: DOWN,  # Move Down from state 0
    1: RIGHT, # Move Right from state 1
    2: DOWN,  # Move Down from state 2
    3: LEFT,  # Move Left from state 3
    4: DOWN,  # Move Down from state 4
    5: LEFT,  # Stay at the hole
    6: DOWN,  # Move Down from state 6
    7: LEFT,  # Stay at the hole
    8: RIGHT, # Move Right from state 8
    9: RIGHT, # Move Right from state 9
    10: DOWN, # Move Down from state 10
    11: LEFT, # Stay at the hole
    12: RIGHT, # Stay at the hole
    13: DOWN, # Move Down from state 13
    14: RIGHT, # Move Right from state 14
    15: LEFT  # Stay at the goal
}[s]

print("Name:JEEVANESH S")
print("Register Number: 212222243002")
print_policy(pi_2, P, action_symbols=('<', 'v', '>', '^'), n_cols=4)
```

```
Name:JEEVANESH S
Register Number: 212222243002
Policy:
| 00      v | 01      > | 02      v | 03      < |
| 04      v |           | 06      v |           |
| 08      > | 09      > | 10      v |           |
|           | 13      v | 14      > |           |
```

```
# Find the probability of success and the mean return of you your
policy

print('Policy 2:')
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)))
```

```
Policy 2:
Reaches goal 0.00%. Obtains an average undiscounted return of 0.0000.
```

```python
def policy_evaluation(pi, P, gamma=1.0, theta=1e-10):
    prev_v = np.zeros(len(P))
    while True:
        v = np.zeros(len(P))
        for s in range(len(P)):
            a = pi(s)
            for prob, next_s, reward, done in P[s][a]:
                v[s] += prob * (reward + gamma * prev_v[next_s] * (not
done))
        if np.max(np.abs(prev_v - v)) < theta:
            return v

        prev_v = v.copy()
    return v

pi_frozenlake = lambda s: {
    0: RIGHT,
    1: DOWN,
    2: RIGHT,
    3: LEFT,
    4: DOWN,
    5: LEFT,
    6: RIGHT,
    7:LEFT,
    8: UP,
    9: DOWN,
    10:LEFT,
    11:DOWN,
    12:RIGHT,
    13:RIGHT,
    14:DOWN,
    15:LEFT #Stop
}[s]

# Code to evaluate the first policy
V1 = policy_evaluation(pi_frozenlake, P,gamma=0.99)
print_state_value_function(V1, P, n_cols=4, prec=5, title='State-value
function (Policy 1):')

# Code to evaluate the second policy
V2 = policy_evaluation(pi_2, P, gamma=0.99)
print_state_value_function(V2, P, n_cols=4, prec=5, title='State-value
function (Policy 2):')
```

```
State-value function (Policy 1):
| 00 0.11448 | 01 0.08191 | 02 0.13372 | 03 0.06586 |
| 04 0.15053 |            | 06 0.20562 |            |
```

```
| 08 0.30562 | 09 0.46997 | 10 0.48938 |            |
|            | 13 0.62915 | 14 0.80739 |            |
State-value function (Policy 2):
| 00 0.02903 | 01 0.0216 | 02 0.04386 | 03 0.0216 |
| 04 0.03733 |            | 06 0.0897 |            |
| 08 0.07579 | 09 0.19233 | 10 0.27183 |            |
|            | 13 0.31099 | 14 0.6314 |            |
```

```python
# Comparing the two policies

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

if np.sum(V1 >= V2) > np.sum(V2 >= V1):
    print("The first policy is the better policy based on state
values.")
elif np.sum(V2 >= V1) > np.sum(V1 >= V2):
    print("The second policy is the better policy based on state
values.")
else:
    print("Both policies have similar state values.")
```

```
The first policy is the better policy based on state values.
```

```python
P
```

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

```
 2: {0: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 6, 0.0, False)],
  1: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 3, 0.0, False)],
  2: [(0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False)],
  3: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False)]},
 3: {0: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 7, 0.0, True)],
  1: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 3, 0.0, False)],
  2: [(0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 3, 0.0, False)],
  3: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False)]},
 4: {0: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False)],
  1: [(0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)],
  2: [(0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 0, 0.0, False)],
  3: [(0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False)]},
 5: {0: [(1.0, 5, 0, True)],
  1: [(1.0, 5, 0, True)],
  2: [(1.0, 5, 0, True)],
  3: [(1.0, 5, 0, True)]},
 6: {0: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 10, 0.0, False)],
  1: [(0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 7, 0.0, True)],
  2: [(0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 2, 0.0, False)],
```

```
  3: [(0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)]},
 7: {0: [(1.0, 7, 0, True)],
  1: [(1.0, 7, 0, True)],
  2: [(1.0, 7, 0, True)],
  3: [(1.0, 7, 0, True)]},
 8: {0: [(0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 12, 0.0, True)],
  1: [(0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 12, 0.0, True),
   (0.3333333333333333, 9, 0.0, False)],
  2: [(0.3333333333333333, 12, 0.0, True),
   (0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 4, 0.0, False)],
  3: [(0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False)]},
 9: {0: [(0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 13, 0.0, False)],
  1: [(0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 10, 0.0, False)],
  2: [(0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)],
  3: [(0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 8, 0.0, False)]},
 10: {0: [(0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 14, 0.0, False)],
  1: [(0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 11, 0.0, True)],
  2: [(0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 11, 0.0, True),
   (0.3333333333333333, 6, 0.0, False)],
  3: [(0.3333333333333333, 11, 0.0, True),
   (0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 9, 0.0, False)]},
 11: {0: [(1.0, 11, 0, True)],
  1: [(1.0, 11, 0, True)],
  2: [(1.0, 11, 0, True)],
  3: [(1.0, 11, 0, True)]},
 12: {0: [(1.0, 12, 0, True)],
  1: [(1.0, 12, 0, True)],
```

```
  2: [(1.0, 12, 0, True)],
  3: [(1.0, 12, 0, True)]},
 13: {0: [(0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 12, 0.0, True),
   (0.3333333333333333, 13, 0.0, False)],
  1: [(0.3333333333333333, 12, 0.0, True),
   (0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 14, 0.0, False)],
  2: [(0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 9, 0.0, False)],
  3: [(0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 9, 0.0, False),
   (0.3333333333333333, 12, 0.0, True)]},
 14: {0: [(0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 14, 0.0, False)],
  1: [(0.3333333333333333, 13, 0.0, False),
   (0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 15, 1.0, True)],
  2: [(0.3333333333333333, 14, 0.0, False),
   (0.3333333333333333, 15, 1.0, True),
   (0.3333333333333333, 10, 0.0, False)],
  3: [(0.3333333333333333, 15, 1.0, True),
   (0.3333333333333333, 10, 0.0, False),
   (0.3333333333333333, 13, 0.0, False)]},
 15: {0: [(1.0, 15, 0, True)],
  1: [(1.0, 15, 0, True)],
  2: [(1.0, 15, 0, True)],
  3: [(1.0, 15, 0, True)]}}
```

```
init_state
```

```
0
```

```
state, h = env.reset()
state, reward, terminated, truncated, info = env.step(RIGHT)
print("state:{0} - reward:{1} - terminated:{2} - truncated:{3} - info:
{4}".format(state, reward, terminated, truncated, info))
```

```
state:4 - reward:0.0 - terminated:False - truncated:False - info:
{'prob': 0.3333333333333333}
```

```
pi_frozenlake = lambda s: {
    0: RIGHT,
    1: DOWN,
    2: RIGHT,
    3: LEFT,
    4: DOWN,
    5: LEFT,
```

```
    6: RIGHT,
    7:LEFT,
    8: UP,
    9: DOWN,
    10:LEFT,
    11:DOWN,
    12:RIGHT,
    13:RIGHT,
    14:DOWN,
    15:LEFT #Stop
}[s]
print_policy(pi_frozenlake, P, action_symbols=('<', 'v', '>', '^'),
n_cols=4)
```

```
Policy:
| 00      > | 01      v | 02      > | 03      < |
| 04      v |           | 06      > |           |
| 08      ^ | 09      v | 10      < |           |
|           | 13      > | 14      v |           |
```

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

```
Reaches goal 0.00%. Obtains an average undiscounted return of 0.0000.
```

```
# Create your own policy

pi_2 = lambda s: {
    0: DOWN,  # Move Down from state 0
    1: RIGHT, # Move Right from state 1
    2: DOWN,  # Move Down from state 2
    3: LEFT,  # Move Left from state 3
    4: DOWN,  # Move Down from state 4
    5: LEFT,  # Stay at the hole
    6: DOWN,  # Move Down from state 6
    7: LEFT,  # Stay at the hole
    8: RIGHT, # Move Right from state 8
    9: RIGHT, # Move Right from state 9
    10: DOWN, # Move Down from state 10
    11: LEFT, # Stay at the hole
    12: RIGHT, # Stay at the hole
    13: DOWN, # Move Down from state 13
    14: RIGHT, # Move Right from state 14
    15: LEFT  # Stay at the goal
}[s]

print("Name:JEEVANESH S")
```

```
print("Register Number: 212222243002")
print_policy(pi_2, P, action_symbols=('<', 'v', '>', '^'), n_cols=4)
```

```
Name:JEEVANESH S
Register Number: 212222243002
Policy:
| 00      v | 01      > | 02      v | 03      < |
| 04      v |           | 06      v |           |
| 08      > | 09      > | 10      v |           |
|           | 13      v | 14      > |           |
```

```
# Compare your policy with the first policy
```

Policy Evaluation

```
def policy_evaluation(pi, P, gamma=1.0, theta=1e-10):
    prev_v = np.zeros(len(P))
    while True:
        v = np.zeros(len(P))
        for s in range(len(P)):
            a = pi(s)
            for prob, next_s, reward, done in P[s][a]:
                v[s] += prob * (reward + gamma * prev_v[next_s] * (not
done))
        if np.max(np.abs(prev_v - v)) < theta:
            return v

        prev_v = v.copy()
    return v

# Code to evaluate the first policy
V1 = policy_evaluation(pi_frozenlake, P,gamma=0.99)
print_state_value_function(V1, P, n_cols=4, prec=5)
```

```
State-value function:
| 00 0.11448 | 01 0.08191 | 02 0.13372 | 03 0.06586 |
| 04 0.15053 |            | 06 0.20562 |            |
| 08 0.30562 | 09 0.46997 | 10 0.48938 |            |
|            | 13 0.62915 | 14 0.80739 |            |
```

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

```
V1>=V2
```

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

```python
if(np.sum(V1>=V2)==11):
  print("The first policy is the better policy")
elif(np.sum(V2>=V1)==11):
  print("The second policy is the better policy")
else:
  print("Both policies have their merits.")
```

```
Both policies have their merits.
```