vi_and_pi

August 11, 2019

1 FrozenLake-v0

The agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water. Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.

SFFF	(S: starting point, safe)
FHFH	(F: frozen surface, safe)
FFFH	(H: hole, fall to your doom)
HFFG	(G: goal, where the frisbee is located)

The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise.

[1]:
```
### MDP Value Iteration and Policy Iteration

import numpy as np
import gym
import time
from lake_envs import *

np.set_printoptions(precision=3)

"""
For policy_evaluation, policy_improvement, policy_iteration and value_iteration,
the parameters P, nS, nA, gamma are defined as follows:

 P: nested dictionary
  From gym.core.Environment
  For each pair of states in [1, nS] and actions in [1, nA], P[state][action] is a
  tuple of the form (probability, nextstate, reward, terminal) where
   - probability: float
    the probability of transitioning from "state" to "nextstate" with "action"
```

```
    - nextstate: int
     denotes the state we transition to (in range [0, nS - 1])
    - reward: int
     either 0 or 1, the reward for transitioning from "state" to
     "nextstate" with "action"
    - terminal: bool
      True when "nextstate" is a terminal state (hole or goal), False otherwise
 nS: int
  number of states in the environment
 nA: int
  number of actions in the environment
 gamma: float
  Discount fhhhactor. Number in range [0, 1)
"""
```

[1]:

```
'\nFor policy_evaluation, policy_improvement, policy_iteration and
value_iteration,\nthe parameters P, nS, nA, gamma are defined as follows:\n\n P:
nested dictionary\n  From gym.core.Environment\n  For each pair of states in [1,
nS] and actions in [1, nA], P[state][action] is a\n  tuple of the form
(probability, nextstate, reward, terminal) where\n   - probability: float\n
the probability of transitioning from "state" to "nextstate" with "action"\n   -
nextstate: int\n    denotes the state we transition to (in range [0, nS - 1])\n
- reward: int\n    either 0 or 1, the reward for transitioning from "state" to\n
"nextstate" with "action"\n   - terminal: bool\n     True when "nextstate" is a
terminal state (hole or goal), False otherwise\n nS: int\n  number of states in
the environment\n nA: int\n  number of actions in the environment\n gamma:
float\n  Discount fhhhactor. Number in range [0, 1)\n'
```

[2]:

```
def policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):

    """Evaluate the value function from a given policy.
    Parameters
    ----------
    P, nS, nA, gamma:
    defined at beginning of file
    policy: np.array[nS]
    The policy to evaluate. Maps states to actions.
    tol: float
    Terminate policy evaluation when
    max |value_function(s) - prev_value_function(s)| < tol
    Returns
    -------
    value_function: np.ndarray[nS]
    The value function of the given policy, where value_function[s] is
    the value of state s
    """

    V_prime, V = np.zeros(nS), np.zeros(nS)
```

```
############################
# YOUR IMPLEMENTATION HERE #
    while True:
        V = V_prime
        V_prime = np.zeros(nS)
        for s in range(nS):
            for pr, new_s, reward,_ in P[s][ policy[s] ]:
                V_prime[s] += pr*reward + gamma*pr*V[new_s]
        if np.max(V_prime-V) < tol:
            break
    value_function =  V_prime
############################
    return value_function
```

[3]:
```
def policy_improvement(P, nS, nA, value_from_policy, policy, gamma=0.9):

    """Given the value function from policy improve the policy.

    Parameters
    ----------
    P, nS, nA, gamma:
        defined at beginning of file
    value_from_policy: np.ndarray
        The value calculated from the policy
    policy: np.array
        The previous policy.

    Returns
    -------
    new_policy: np.ndarray[nS]
        An array of integers. Each integer is the optimal action to take
    in that state according to the environment dynamics and the
        given value function.
    """

    new_policy = np.zeros(nS, dtype='int')
    value = np.zeros(nA)

############################
    # YOUR IMPLEMENTATION HERE

    for s in range(nS):
        for a in range(nA):
            for pr, new_s, reward,_ in P[s][a]:
                value[a] += pr*reward + gamma*pr*value_from_policy[new_s]
        new_policy[s] = np.argmax(value)
```

```
        value = np.zeros(nA)

############################
    return new_policy


def policy_iteration(P, nS, nA, gamma=0.9, tol=10e-3):
    """Runs policy iteration.

    You should call the policy_evaluation() and policy_improvement() methods to
    implement this method.

    Parameters
    ----------
    P, nS, nA, gamma:
        defined at beginning of file
    tol: float
        tol parameter used in policy_evaluation()
    Returns:
    ----------
    value_function: np.ndarray[nS]
    policy: np.ndarray[nS]
    """


    policy = np.random.randint(nA, size=nS, dtype = int)

############################
# YOUR IMPLEMENTATION HERE #

    while True:
        value = policy_evaluation(P, nS, nA, policy, gamma, tol)
        policy_improved = policy_improvement(P, nS, nA, value, policy, gamma)
        if  all(policy_improved == policy):
            break
        policy = policy_improved
############################
    policy = policy_improved
    value_function = policy_evaluation(P, nS, nA, policy_improved, gamma, tol)
    return value_function, policy
```

[4]:
```
def value_iteration(P, nS, nA, gamma=0.9, tol=1e-3):
    """
    Learn value function and policy by using value iteration method for a given
    gamma and environment.

    Parameters:
```

```
    ----------
    P, nS, nA, gamma:
        defined at beginning of file
    tol: float
        Terminate value iteration when
            max |value_function(s) - prev_value_function(s)| < tol
    Returns:
    ----------
    value_function: np.ndarray[nS]
    policy: np.ndarray[nS]
    """

    policy = np.random.randint(nA, size=nS)
    V_prime, V = np.zeros(nS), np.zeros(nS)
    value = np.zeros(nA)
############################
# YOUR IMPLEMENTATION HERE #

    while True:
        V  = V_prime
        for s in range(nS):
            for a in range(nA):
                for pr, new_s, reward,_ in P[s][a]:
                    value[a] += pr*reward + gamma*pr*V_prime[new_s]
            policy[s] = np.argmax(value)
            value = np.zeros(nA)
            V_prime = policy_evaluation(P, nS, nA, policy, gamma, tol)
        if np.max(V_prime-V) < tol:
            break
    value_function = V_prime

############################
    return value_function, policy
```

[9]:
```
def render_single(env, policy, max_steps=100):
    """
    This function does not need to be modified
    Renders policy once on environment. Watch your agent play!

    Parameters
    ----------
    env: gym.core.Environment
      Environment to play on. Must have nS, nA, and P as
      attributes.
    Policy: np.array of shape [env.nS]
      The action to take at a given state
    """
```

```
    episode_reward = 0
    ob = env.reset()
    for t in range(max_steps):
        env.render()
        time.sleep(0.25)
        a = policy[ob]
        ob, rew, done, _ = env.step(a)
        episode_reward += rew
        if done:
            break
        env.render();
    env.render();
    if not done:
        print("The agent didn't reach a terminal state in {} steps.".format(max_steps))
    else:
        print("Episode reward: %f" % episode_reward)
```

[10]:

```
# Edit below to run policy and value iteration on different environments and
# visualize the resulting policies in action!
# You may change the parameters in the functions below
if __name__ == "__main__":

    # comment/uncomment these lines to switch between deterministic/stochastic environments
#     env = gym.make("Deterministic-4x4-FrozenLake-v0")
    env = gym.make("Deterministic-8x8-FrozenLake-v0")
#     env = gym.make("Stochastic-4x4-FrozenLake-v0")

    print("\n" + "-"*25 + "\nBeginning Policy Iteration\n" + "-"*25)

    V_pi, p_pi = policy_iteration(env.P, env.nS, env.nA, gamma=0.9, tol=1e-3)
    render_single(env, p_pi, 50)

    print("\n" + "-"*25 + "\nBeginning Value Iteration\n" + "-"*25)

    V_vi, p_vi = value_iteration(env.P, env.nS, env.nA, gamma=0.9, tol=1e-3)
    render_single(env, p_vi, 50)
```

```
-------------------------
Beginning Policy Iteration
-------------------------

SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
```

```
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
```

```
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
```

```
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
```

```
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
```

```
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
```

```
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
Episode reward: 1.000000

-------------------------
Beginning Value Iteration
-------------------------

SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
```

```
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
```

```
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
```

```
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
```

```
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
```

```
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
  (Right)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
Episode reward: 1.000000
```

[ ]: