assignment4

February 20, 2021

1 NOTE: Please see PDF file in same folder.

[1]:
```
import sys
sys.path.append("../rl/")
```

1.1 Problem 1

1.1.1 Part (A) : Manually Calculate

$$V_0 = \begin{bmatrix} 10.0 \\ 1.0 \\ 0.0 \end{bmatrix}$$

$$q_1(s_1, a_1) = 10.6 q_1(s_1, a_2) = 11.2 \pi_1(s1) = a_2 v_1(s1) = 11.2$$

$$q_1(s_2, a_1) = 4.3 q_1(s_2, a_2) = 4.3 \pi_1(s2) = a_1 \text{ or } a_2 v_1(s2) = 4.3$$

$$V_1 = \begin{bmatrix} 11.2 \\ 4.3 \\ 0.0 \end{bmatrix} \pi_1 = \begin{bmatrix} a_2 \\ a_1 \text{ or } a_2 \end{bmatrix}$$

$$q_2(s_1, a_1) = 12.82 q_2(s_1, a_2) = 11.98 \pi_2(s1) = a_1 v_2(s1) = 12.82$$

$$q_2(s_2, a_1) = 5.65 q_2(s_2, a_2) = 5.89 \pi_2(s2) = a_2 v_2(s2) = 5.89$$

$$V_2 = \begin{bmatrix} 12.82 \\ 5.89 \\ 0.0 \end{bmatrix} \pi_2 = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

1.1.2 Part (B) : Policy remains the same

NOTE: The value function for s_3 will remain 0.0.

For State s_1: The action a_1 puts more weight on s_1 and s_2 than action a_2. Since the value function is non-decreasing, the optimal action for s_1 is a_1.

For State s_2: Similar to the reasoning above ^, action a_2 for state s_2 assigns equal or more weight to states s_1 and s_2. So we expect a_2 to be the optimal action for state s_2.

[ ]:

```

```

1.2 Problem 2

[6]:

```
import sys
sys.path.append("../rl/")

from markov_decision_process import FiniteMarkovDecisionProcess, FinitePolicy
from distribution import Categorical, Choose, Constant
from collections import defaultdict
import numpy as np
import seaborn as sns
from rl.dynamic_programming import (policy_iteration, value_iteration)

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[5]:

```
def get_frog_MDP(N=10):
    action_mapping = {}

    action_mapping[0], action_mapping[N] = None, None

    reward_func = lambda s: 1.0*(s==N) #- 10.0*(s==0)

    for i in range(1, N, 1):
        action_mapping[i] = {
            '0': Categorical({ (i-1, reward_func(i-1)):(i/N), (i+1,␣
↪reward_func(i+1)):(1-(i/N)) }),
            '1': Categorical({ (j, reward_func(j)): (1/N) for j in range(0,␣
↪N+1, 1) if j != i })
        }

    return FiniteMarkovDecisionProcess(action_mapping)
```

[40]:

```
# N=100
```

[43]:
```python
# Value Iteration
def VI_iter(N):
    v_vi_stable = None
    for (i, v_vi) in enumerate(value_iteration(mdp=get_frog_MDP(N=N), gamma=0.
 ↪99)):
        if v_vi_stable is not None:
            if v_vi_stable == v_vi:
#                   print(f"finished Value Iteration in iteration {i}")
#                   break
                return i
        v_vi_stable = v_vi
```

[44]:
```python
# Policy Iteration
def PI_iter(N):
    v_pl_stable = None
    for (i, v_pl) in enumerate(policy_iteration(mdp=get_frog_MDP(N=N), gamma=0.
 ↪99)):
        if v_pl_stable is not None:
            if v_pl_stable[0] == v_pl[0]:
#                   print(f"finished Policy Iteration in iteration {i}")
#                   break
                return i
        v_pl_stable = v_pl
```

[46]:
```python
Ns
```

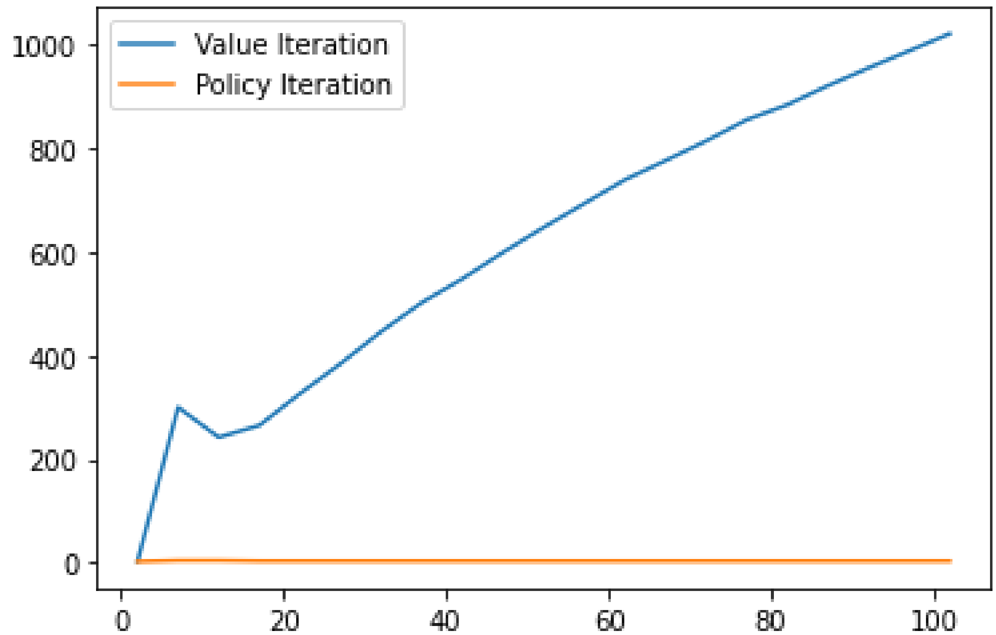
[46]:
```
array([1])
```

[51]:
```python
Ns = np.arange(2, 103, 5)

VI_num_iters = [VI_iter(n) for n in Ns]
PI_num_iters = [PI_iter(n) for n in Ns]
```

[54]:
```python
sns.lineplot(x=Ns, y=VI_num_iters, label="Value Iteration")
sns.lineplot(x=Ns, y=PI_num_iters, label="Policy Iteration")
```

[54]:
```
<AxesSubplot:>
```




```
[ ]:
```

1.3 Problem 3

State The state is the tuple of (current/offered wage, employment status) = $(w, \{E, U\})$ for $w \in \{w_1, .., w_n\}$, and E stands for Employed, U unemployed.

Action For currently employed people, the only action is Accept (A). For unemployed people, the action space is Accept (A) or Reject (R) job offer.

Transition Probability:

$$P((w_i, E), A, (w_i, E) = (1 - \alpha)$$

$$P((w_i, E), A, (w_j, U) = \alpha * p_j$$

$$P((w_i, U), A, (w_i, E) = (1 - \alpha)$$

$$P((w_i, U), A, (w_j, U) = \alpha * p_j$$

$$P((w_i, U), R, (w_j, U) = p_j$$

Reward Function: $R((w_i, \_), A) = log(w_i)$
$R((w_i, \_), R) = 0.0$

Bellman Optimal Eq $V((w, U)) = max_{a \in \{A,R\}} \Bigg[R((w, U), a) + \gamma * [(1 - \alpha) * V((w, E)) + \sum_i P((w, U), a, (w_i, U) * V((w, U))] \Bigg]$

```
[ ]:
```

```
[ ]:
```

1.4 Problem 4

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```