Lecture E1.MDP with Model1

Bong Seok Kim

2021-01-22

차 례

Iterative estimation of state-value function for given policy pi^speed

```
import numpy as np

R = np.hstack((np.repeat(-1.5, 4), -0.5, np.repeat(-1.5, 2), 0)).reshape(8, 1)

states = np.array(range(0, 80, 10))
P = np.array ([[.1, 0, .9, 0, 0, 0, 0, 0],
               [.1, 0, 0, .9, 0, 0, 0, 0],
               [0, .1, 0, 0, .9, 0, 0, 0],
               [0, 0, .1, 0, 0, .9, 0, 0],
               [0, 0, 0, .1, 0, 0, .9, 0],
               [0, 0, 0, 0, .1, 0, 0, .9],
               [0, 0, 0, 0, 0, .1, 0, .9],
               [0, 0, 0, 0, 0,  0, 0, 1]])
gamma = 1.0
epsilon = 10**(-8)
v_old = np.array(np.repeat(0, 8)).reshape(8,1)

while True:
    v_new = R+gamma*np.dot(P, v_old)
    if np.max(np.abs(v_new-v_old)) > epsilon:
        v_old = v_new
        continue
    break

print(v_new.T)
```

```
## [[-5.80592905 -5.2087811  -4.13926239 -3.47576467 -2.35376031 -1.73537603
##   -1.6735376   0.        ]]
```

Rewritten with intermediate saving

```
# Rewriiten with intermediate saving
import numpy as np
import pandas as pd

gamma = 1.0
epsilon = 10**(-8)

v_old = np.array(np.repeat(0, 8)).reshape(8,1)

result = [ ]
while True:
    result.append(v_old.T)
    v_new = R+gamma*np.dot(P, v_old)
    if np.max(np.abs(v_new-v_old)) > epsilon:
        v_old = v_new
        continue
    break

result = pd.DataFrame(np.array(result).reshape(len(result),8), columns=states)

result.head()
```

```
##        0      10      20     30     40      50     60   70
## 0  0.000  0.0000  0.0000  0.000  0.000  0.0000  0.000  0.0
## 1 -1.500 -1.5000 -1.5000 -1.500 -0.500 -1.5000 -1.500  0.0
## 2 -3.000 -3.0000 -2.1000 -3.000 -2.000 -1.5500 -1.650  0.0
## 3 -3.690 -4.5000 -3.6000 -3.105 -2.285 -1.7000 -1.655  0.0
## 4 -5.109 -4.6635 -4.0065 -3.390 -2.300 -1.7285 -1.670  0.0
```

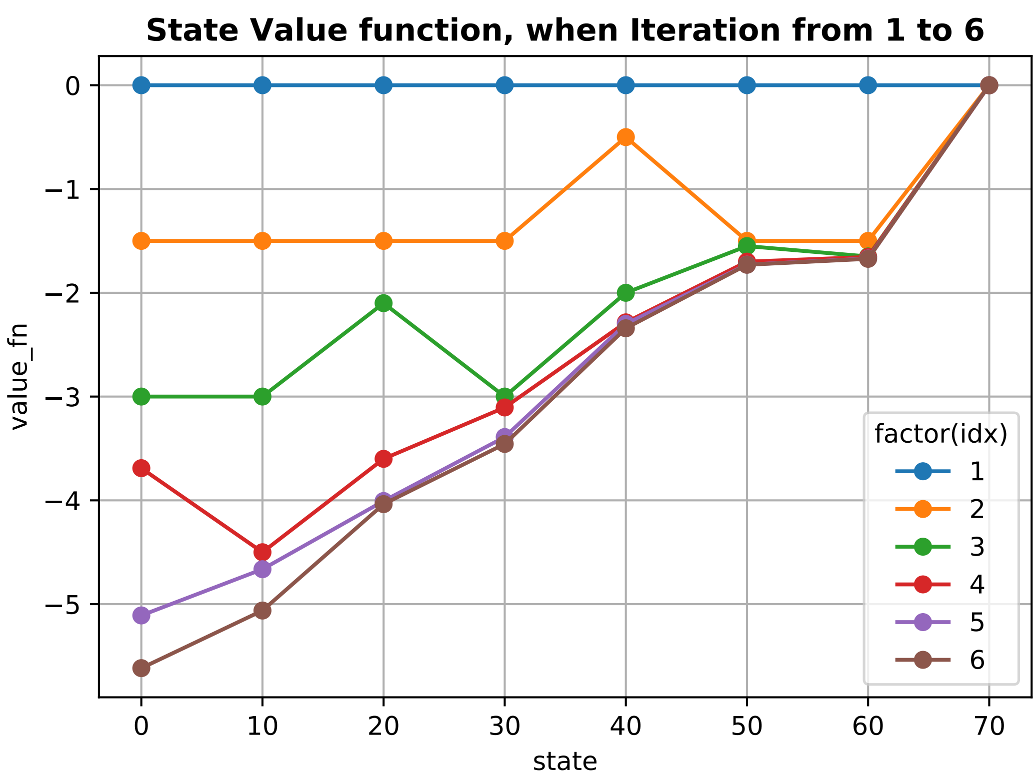
```
result.tail()
```

```
##            0         10         20         30        40         50         60   70
## 17 -5.805928 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 18 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 19 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 20 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 21 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
```

Plot

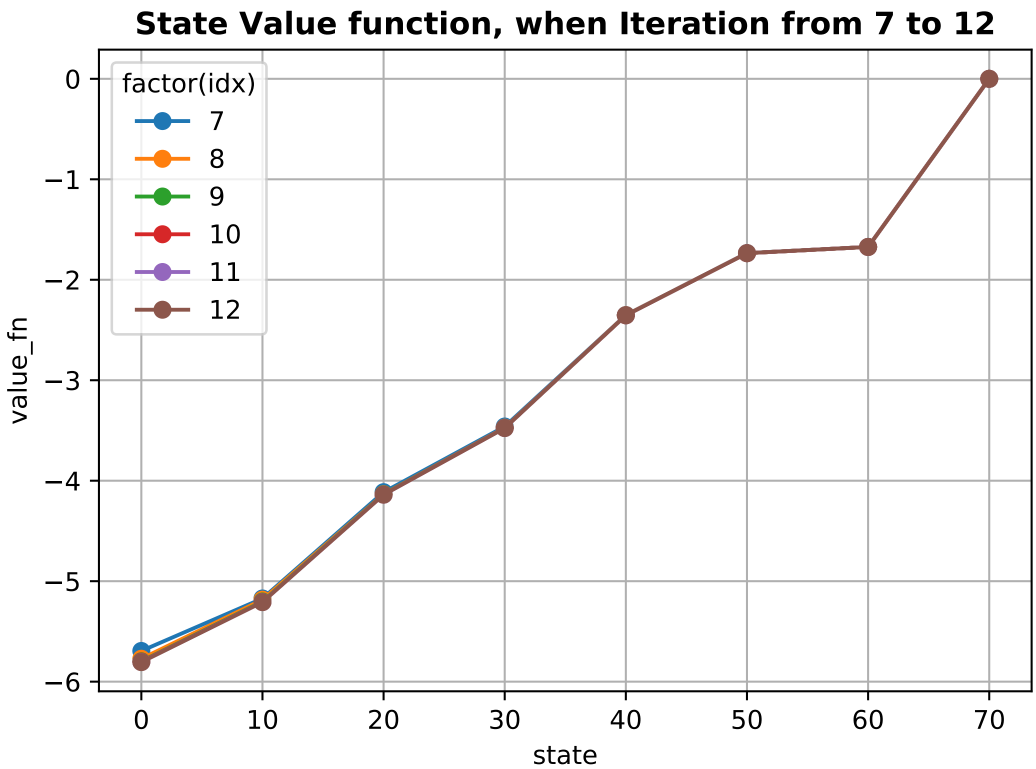
Iteration from 1 to 6

```
import matplotlib.pyplot as plt


for i in range(0,6):
    plt.plot(states, result.iloc[i], marker='o', label=str(i+1))
    plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 1 to 6',fontweight='bold')
plt.show()
```



Iteration from 7 to 12

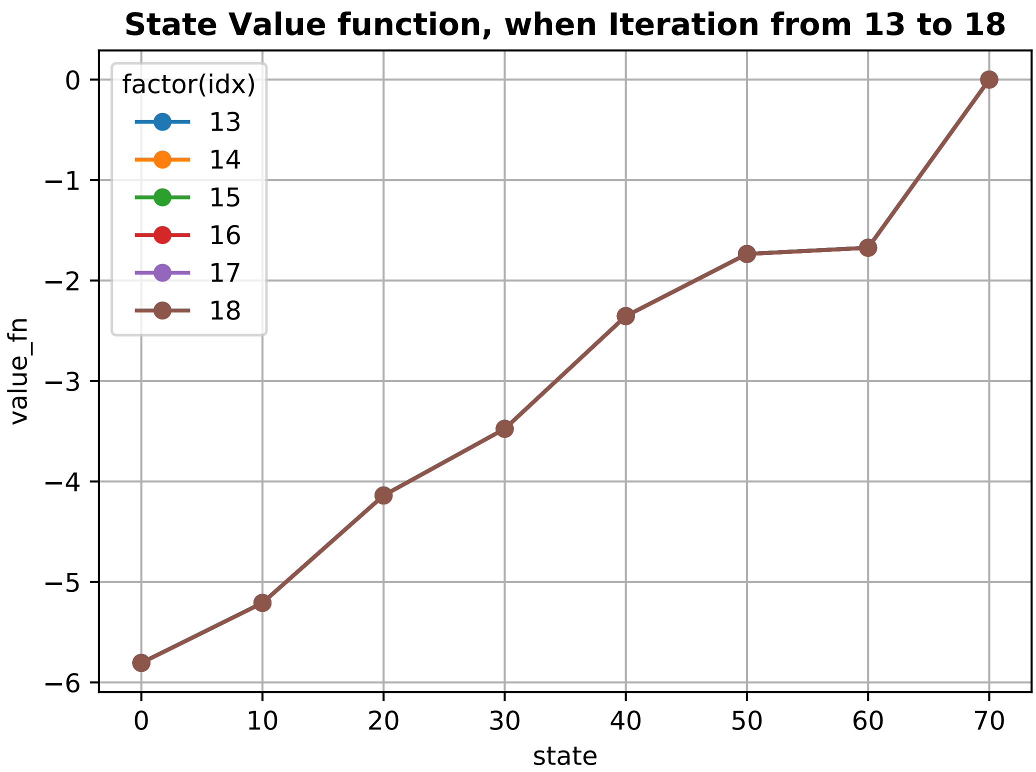
```
import matplotlib.pyplot as plt

for i in range(6,12):
    plt.plot(states, result.iloc[i], marker='o', label=str(i+1))
    plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 7 to 12',fontweight='bold')
plt.show()
```



Iteration from 13 to 18

```
import matplotlib.pyplot as plt

for i in range(12,18):
    plt.plot(states, result.iloc[i], marker='o', label=str(i+1))
    plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
```

```
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 13 to 18',fontweight='bold')
plt.show()
```



Policy evaluation 2

Pi : S —-> A

```
import numpy as np
import pandas as pd

states = np.array(range(0,80,10)).astype(str)
pi_speed=np.c_[np.repeat(0,len(states)),np.repeat(1,len(states))]
pi_speed=pd.DataFrame(pi_speed, columns=['normal','speed'],index=states)

pi_speed
```

```
##     normal  speed
## 0        0      1
## 10       0      1
## 20       0      1
## 30       0      1
## 40       0      1
## 50       0      1
## 60       0      1
## 70       0      1
```

R^pi = S —-> R

```
R_s_a=np.array([[-1, -1, -1, -1,0, -1, -1, 0],
          [-1.5, -1.5, -1.5,-1.5, -0.5, -1.5, -1.5, 0]]).T
R_s_a=pd.DataFrame(R_s_a,columns=['normal','speed'],index=states)

R_s_a
```

```
##     normal  speed
## 0     -1.0   -1.5
## 10    -1.0   -1.5
## 20    -1.0   -1.5
## 30    -1.0   -1.5
## 40     0.0   -0.5
## 50    -1.0   -1.5
## 60    -1.0   -1.5
## 70     0.0    0.0
```

```python
def reward_fn(given_pi):

    R_s_a=pd.DataFrame(
        np.array([[-1, -1, -1, -1,0, -1, -1, 0],
          [-1.5, -1.5, -1.5,-1.5, -0.5, -1.5, -1.5, 0]]).T,columns=['normal','speed'],index=states)

    R_pi=np.sum(R_s_a*given_pi,axis=1)

    return R_pi

reward_fn(pi_speed).values
```

```
## array([-1.5, -1.5, -1.5, -1.5, -0.5, -1.5, -1.5,  0. ])
```

P^pi : S X A ->S

```python
P_normal = np.array([
                   [0,1,0,0,0,0,0,0],
                   [0,0,1,0,0,0,0,0],
                   [0,0,0,1,0,0,0,0],
                   [0,0,0,0,1,0,0,0],
                   [0,0,0,0,0,1,0,0],
                   [0,0,0,0,0,0,1,0],
                   [0,0,0,0,0,0,0,1],
                   [0,0,0,0,0,0,0,1]])

P_speed = np.array([[.1,0,.9,0,0,0,0,0],
                   [.1,0,0,.9,0,0,0,0],
                   [0,.1,0,0,.9,0,0,0],
                   [0,0,.1,0,0,.9,0,0],
                   [0,0,0,.1,0,0,.9,0],
                   [0,0,0,0,.1,0,0,.9],
                   [0,0,0,0,0,.1,0,.9],
                   [0,0,0,0,0,0,0,1]])



def transition(given_pi,states,P_normal,P_speed):
    P_out = np.zeros(shape=(8,8))
```

```
    for i in range(len(states)):
        action_dist=given_pi.iloc[i,:]

        P = action_dist['normal']*P_normal + action_dist['speed']*P_speed

        P_out[i,]=P[i,]

    return P_out
```

test 1

```
transition(pi_speed, states=states, P_normal=P_normal, P_speed=P_speed)
```

```
## array([[0.1, 0. , 0.9, 0. , 0. , 0. , 0. , 0. ],
##        [0.1, 0. , 0. , 0.9, 0. , 0. , 0. , 0. ],
##        [0. , 0.1, 0. , 0. , 0.9, 0. , 0. , 0. ],
##        [0. , 0. , 0.1, 0. , 0. , 0.9, 0. , 0. ],
##        [0. , 0. , 0. , 0.1, 0. , 0. , 0.9, 0. ],
##        [0. , 0. , 0. , 0. , 0.1, 0. , 0. , 0.9],
##        [0. , 0. , 0. , 0. , 0. , 0.1, 0. , 0.9],
##        [0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. ]])
```

test 2

```
pi_50=pd.DataFrame(np.c_[np.repeat(0.5,len(states)),np.repeat(0.5,len(states))], index=states, columns=['norm

pi_50
```

```
##     normal  speed
## 0      0.5    0.5
## 10     0.5    0.5
## 20     0.5    0.5
## 30     0.5    0.5
## 40     0.5    0.5
## 50     0.5    0.5
## 60     0.5    0.5
## 70     0.5    0.5
```

```
transition(pi_50,states=states, P_normal=P_normal, P_speed=P_speed)
```

```
## array([[0.05, 0.5 , 0.45, 0.  , 0.  , 0.  , 0.  , 0.  ],
##        [0.05, 0.  , 0.5 , 0.45, 0.  , 0.  , 0.  , 0.  ],
##        [0.  , 0.05, 0.  , 0.5 , 0.45, 0.  , 0.  , 0.  ],
##        [0.  , 0.  , 0.05, 0.  , 0.5 , 0.45, 0.  , 0.  ],
##        [0.  , 0.  , 0.  , 0.05, 0.  , 0.5 , 0.45, 0.  ],
##        [0.  , 0.  , 0.  , 0.  , 0.05, 0.  , 0.5 , 0.45],
##        [0.  , 0.  , 0.  , 0.  , 0.  , 0.05, 0.  , 0.95],
##        [0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  ]])
```

Implementation, finally

```
def policy_eval(given_pi):
    R = reward_fn(given_pi).values.reshape(8,1)
    P = transition(given_pi,states, P_normal = P_normal, P_speed = P_speed)

    gamma = 1.0
    epsilon = 10**(-8)
    v_old = np.array(np.repeat(0, 8)).reshape(8,1)

    while True:
        v_new = R+gamma*np.dot(P, v_old)
        if np.max(np.abs(v_new-v_old)) > epsilon:
            v_old = v_new
            continue
        break

    return v_new
```

```
policy_eval(pi_speed).T
```

```
## array([[-5.80592905, -5.2087811 , -4.13926239, -3.47576467, -2.35376031,
##         -1.73537603, -1.6735376 ,  0.        ]])
```

```
policy_eval(pi_50).T
```

```
## array([[-5.96923786, -5.13359222, -4.11995525, -3.38922824, -2.04147003,
##         -2.02776769, -1.35138838,  0.        ]])
```

```
"Done, Lecture E1.MDP with Model1 "
```

```
## [1] "Done, Lecture E1.MDP with Model1 "
```