E1 solution

reinforcement learning study

2021-02-04

차 례

21p

```
# 21p

R=np.hstack((np.repeat(-1.5,4),-0.5,np.repeat(-1.5,2),0)).reshape(-1,1)
states=np.arange(0,70+10,step=10)

P=np.matrix([[.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]])

P=pd.DataFrame(P,columns=states)

R
```

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

```
P
```

```
##     0    10   20   30   40   50   60   70
## 0  0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 1  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 2  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 3  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 4  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 5  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 6  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

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



v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R+np.dot(gamma*P,v_old)

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

print(v_new.T)
```

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

Rewritten with intermediate saving

```
R=np.hstack((np.repeat(-1.5,4),-0.5,np.repeat(-1.5,2),0)).reshape(-1,1)
states=np.arange(0,70+10,step=10)

P=np.matrix([[.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]])

P=pd.DataFrame(P,columns=states)

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

v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R+np.dot(gamma*P,v_old)

results=v_old.T
results=np.vstack((results,v_new.T))

while np.max(np.abs(v_new-v_old)) > epsilon:
    v_old=v_new
    v_new=R+np.dot(gamma*P, v_old)
    results=np.vstack((results,v_new.T))

results=pd.DataFrame(results, columns=states)

results.head()
```

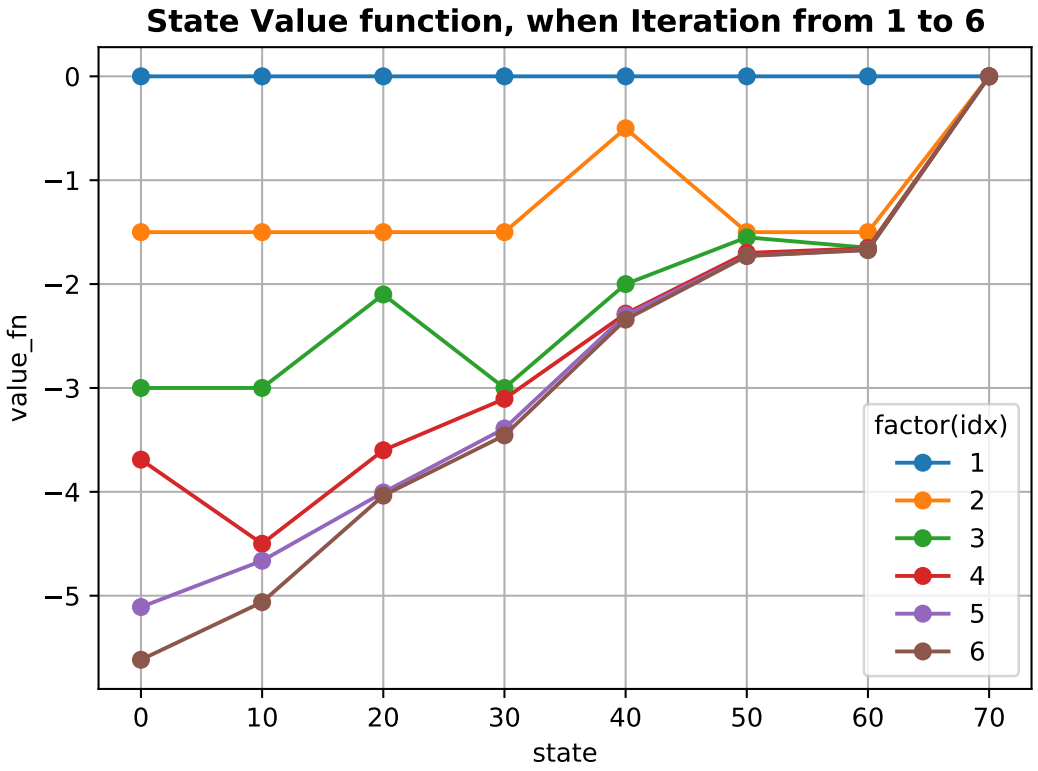
```
##       0       10      20     30     40      50     60   70
## 0  0.000  0.0000  0.0000  0.000  0.000  0.0000  0.000  0.0
## 1 -1.500 -1.5000 -1.5000 -1.500 -0.500 -1.5000 -1.500  0.0
## 2 -3.000 -3.0000 -2.1000 -3.000 -2.000 -1.5500 -1.650  0.0
## 3 -3.690 -4.5000 -3.6000 -3.105 -2.285 -1.7000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

```
results.tail()
```

```
##           0         10        20        30       40        50        60   70
## 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
## 22 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
```

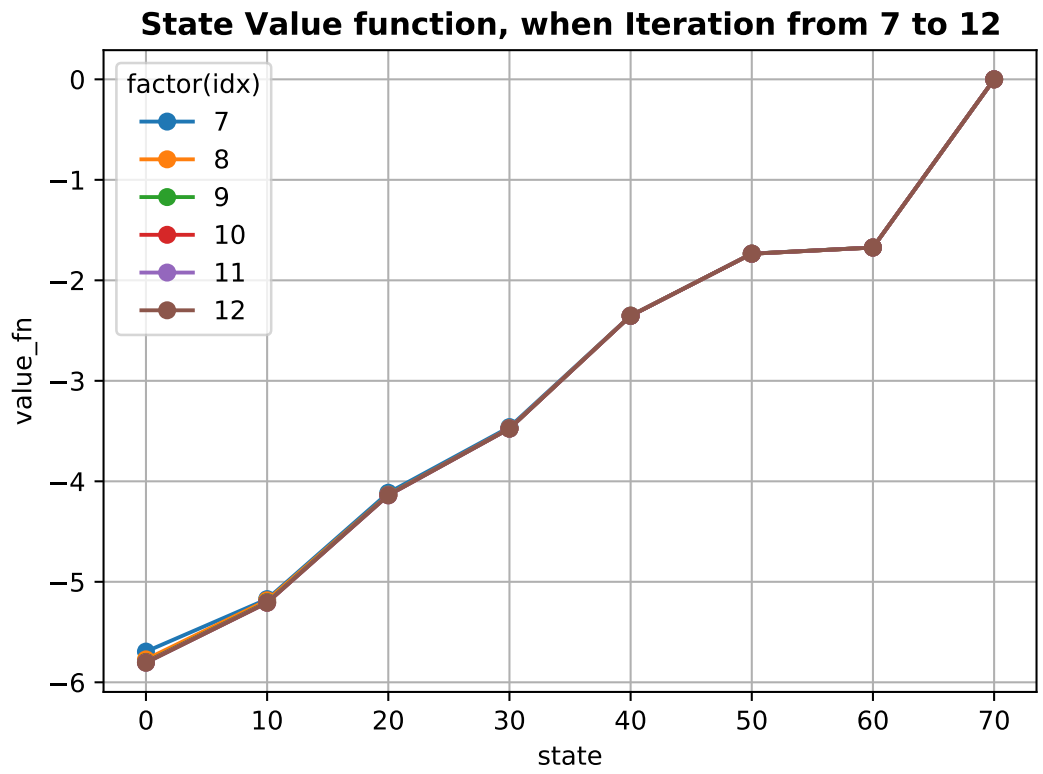
Plots iteration 1 to 6

```
plt.plot(states,results.iloc[0],marker='o',label='1')
plt.plot(states,results.iloc[1],marker='o',label='2')
plt.plot(states,results.iloc[2],marker='o',label='3')
plt.plot(states,results.iloc[3],marker='o',label='4')
plt.plot(states,results.iloc[4],marker='o',label='5')
plt.plot(states,results.iloc[5],marker='o',label='6')
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()
```



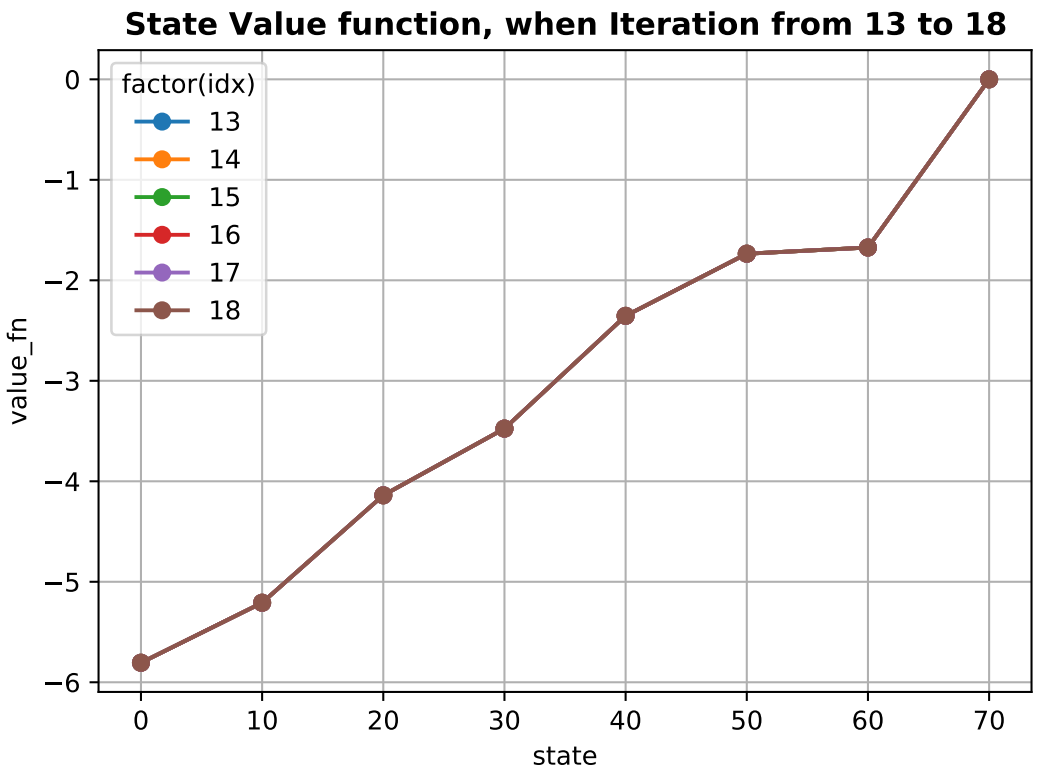
Plots iteration 7 to 12

```
plt.plot(states,results.iloc[6],marker='o',label='7')
plt.plot(states,results.iloc[7],marker='o',label='8')
plt.plot(states,results.iloc[8],marker='o',label='9')
plt.plot(states,results.iloc[9],marker='o',label='10')
plt.plot(states,results.iloc[10],marker='o',label='11')
plt.plot(states,results.iloc[11],marker='o',label='12')
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()
```



Plots iteration 13 to 18

```
plt.plot(states,results.iloc[12],marker='o',label='13')
plt.plot(states,results.iloc[13],marker='o',label='14')
plt.plot(states,results.iloc[14],marker='o',label='15')
plt.plot(states,results.iloc[15],marker='o',label='16')
plt.plot(states,results.iloc[16],marker='o',label='17')
plt.plot(states,results.iloc[17],marker='o',label='18')
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()
```



Preparation for 1-3

1) $\pi : S \rightarrow A$

```
states=np.arange(0,70+10,10).astype('str')
pi_speed=np.c_[np.repeat(0,len(states)),np.repeat(1,len(states))]

pi_speed=pd.DataFrame(data=pi_speed, index=states, columns=['normal','speed'])
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
```

2) $R^{\pi} : S \rightarrow \mathbb{R}$

```
R_s_a=pd.DataFrame(np.matrix([-1,-1,-1,-1,0.0,-1,-1,0,-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]).reshape(len(stat
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
```

```
def reward_fn(given_pi):
    R_s_a=pd.DataFrame(np.matrix([-1,-1,-1,-1,0.0,-1,-1,0,-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]).reshape(len(

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

    return R_pi

reward_fn(pi_speed)
```

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

3) $P^{\pi} : S \times A \to S$

```
P_normal=pd.DataFrame(np.matrix([[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]]), index=states,columns=states)

P_normal
```

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

```
P_speed=pd.DataFrame(np.matrix([[.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]]), index=states, columns=states)

P_speed
```

```
##       0   10   20   30   40   50   60   70
## 0   0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

```
def transition(given_pi, states, P_normal, P_speed):
    P_out=pd.DataFrame(np.zeros((len(states),len(states))),index=states, columns=states)

    for s in states:
        action_dist=given_pi.loc[s]
        P=action_dist['normal']*P_normal+action_dist['speed']*P_speed
        P_out.loc[s]=P.loc[s]

    return P_out
```

Test

1) Test-1

```
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
```

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

```
##       0   10   20   30   40   50   60   70
## 0   0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

2) 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)
```

```
##        0    10    20    30    40    50    60    70
## 0   0.05  0.50  0.45  0.00  0.00  0.00  0.00  0.00
## 10  0.05  0.00  0.50  0.45  0.00  0.00  0.00  0.00
## 20  0.00  0.05  0.00  0.50  0.45  0.00  0.00  0.00
## 30  0.00  0.00  0.05  0.00  0.50  0.45  0.00  0.00
## 40  0.00  0.00  0.00  0.05  0.00  0.50  0.45  0.00
## 50  0.00  0.00  0.00  0.00  0.05  0.00  0.50  0.45
## 60  0.00  0.00  0.00  0.00  0.00  0.05  0.00  0.95
## 70  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00
```

Summary

1) $\pi : S \to A$

```
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
```

```
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
```

2) $R^{\pi} : S \to \mathbb{R}$

```
reward_fn(pi_speed)
```

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

```
reward_fn(pi_50)
```

```
## array([[-1.25],
##        [-1.25],
##        [-1.25],
##        [-1.25],
##        [-0.25],
##        [-1.25],
##        [-1.25],
##        [ 0.  ]])
```

3) $P^{\pi} : S \times A \to S$

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

```
##       0   10   20   30   40   50   60   70
## 0   0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

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

```
##        0    10    20    30    40    50    60    70
## 0   0.05  0.50  0.45  0.00  0.00  0.00  0.00  0.00
## 10  0.05  0.00  0.50  0.45  0.00  0.00  0.00  0.00
## 20  0.00  0.05  0.00  0.50  0.45  0.00  0.00  0.00
## 30  0.00  0.00  0.05  0.00  0.50  0.45  0.00  0.00
## 40  0.00  0.00  0.00  0.05  0.00  0.50  0.45  0.00
## 50  0.00  0.00  0.00  0.00  0.05  0.00  0.50  0.45
## 60  0.00  0.00  0.00  0.00  0.00  0.05  0.00  0.95
## 70  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00
```

Final Implementation

```
def policy_eval(given_pi):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_normal=P_normal, P_speed=P_speed)

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

    v_old=np.repeat(0,8).reshape(8,1)
    v_new=R+np.dot(gamma*P, v_old)

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

    return v_new.T

policy_eval(pi_speed)
```

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

```
policy_eval(pi_50)
```

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