# F4 python

Lee Sung Ho

2021 2 23

## Setting

```
import numpy as np
import pandas as pd

# model

states=np.arange(0,70+10,10).astype('str')

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_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)

R_s_a=pd.DataFrame(np.c_[[-1,-1,-1,-1,0,-1,-1,0],[-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]], index=states,

q_s_a_init=pd.DataFrame(np.c_[np.repeat(0.0,len(states)), np.repeat(0.0,len(states))], index=states, co


# Policy
pi_speed=pd.DataFrame(np.c_[np.repeat(0,len(states)), np.repeat(1, len(states))], index=states, columns=

pi_50=pd.DataFrame(np.c_[np.repeat(0.5, len(states)), np.repeat(0.5, len(states))], index=states, colum


# simul_step()
def simul_step(pi, s_now, P_normal, P_speed, R_s_a):
    if np.random.uniform(0,1) < pi.loc[s_now]['n']:
        a_now='n'
        P=P_normal
    else:
        a_now='s'
        P=P_speed

    r_now=R_s_a.loc[s_now][a_now]
    s_next=states[np.argmin(P.loc[s_now].cumsum() < np.random.uniform(0,1))].item()

    if np.random.uniform(0,1) < pi.loc[s_next]['n']:
        a_next='n'
    else:
```

```
        a_next='s'

    sarsa=[s_now, a_now, r_now, s_next, a_next]

    return sarsa

# pol_eval_TD
def pol_eval_TD(sample_step, q_s_a, alpha):
    s=sample_step[0]
    a=sample_step[1]
    r=sample_step[2]
    s_next=sample_step[3]
    a_next=sample_step[4]

    q_s_a.loc[s][a]=q_s_a.loc[s][a]+alpha*(r+q_s_a.loc[s_next][a_next]-q_s_a.loc[s][a])

    return q_s_a


# pol_imp()
def pol_imp(pi, q_s_a, epsilon):

  for i in range(len(pi)):
    if (np.random.uniform(1) > epsilon):
        pi.iloc[i] = 0
        pi.iloc[i][np.argmax(q_s_a.iloc[i])]=1


    else:
      pi.iloc[i] = 1/q_s_a.shape[1]

    return pi
```

## Page 6 Write pol-eval_Q

```
def pol_eval_TD(sample_step, q_s_a, alpha):

    s=sample_step[0]
    a=sample_step[1]
    r=sample_step[2]
    s_next=sample_step[3]
    a_next=sample_step[4]

    q_s_a.loc[s][a]=q_s_a.loc[s][a]+alpha*(r+q_s_a.loc[s_next][a_next]-q_s_a.loc[s][a])

    return q_s_a

def pol_eval_Q(sample_step, q_s_a, alpha):

    s=sample_step[0]
    a=sample_step[1]
    r=sample_step[2]
    s_next=sample_step[3]
    a_next=sample_step[4]
    q_s_a.loc[s][a]=q_s_a.loc[s][a]+alpha*(r+max(q_s_a.loc[s_next])-q_s_a.loc[s][a])

    return q_s_a
```

## Page 7 Q-learning

```
import time

num_ep = 10**5
beg_time = time.time()
q_s_a = q_s_a_init
pi = pi_50
exploration_rate = 1

for epi_i in range(1,num_ep+1):
  s_now = "0"

  while (s_now != "70"):
    sample_step = simul_step(pi, s_now, P_normal, P_speed, R_s_a)
    q_s_a = pol_eval_Q(sample_step, q_s_a, alpha=max(1/epi_i, 0.05))

    if(epi_i % 100 == 0):
      pi = pol_imp(pi, q_s_a, epsilon=exploration_rate)

    s_now = sample_step[3]
    exploration_rate = max(exploration_rate * 0.9995 , 0.001)

end_time = time.time()

print(q_s_a.T)
```

```
##           0        10        20        30        40       50        60   70
## n -5.532634 -4.528917 -3.642879 -2.655441 -1.728528 -2.00000 -1.000000  0.0
## s -4.912921 -4.355652 -3.382265 -3.421287 -1.692573 -1.64744 -1.518792  0.0
```

```
print(end_time - beg_time, "sec")
```

```
## 628.5503752231598 sec
```

```
print(pi.T)
```

```
##      0   10   20   30   40   50   60   70
## n  0.0  0.5  0.5  0.5  0.5  0.5  0.5  0.5
## s  1.0  0.5  0.5  0.5  0.5  0.5  0.5  0.5
```

## Page 10 pol_eval_dbl_Q()

```
def pol_eval_Q(sample_step, q_s_a, alpha):

    s=sample_step[0]
    a=sample_step[1]
    r=sample_step[2]
    s_next=sample_step[3]
    a_next=sample_step[4]
    q_s_a.loc[s][a]=q_s_a.loc[s][a]+alpha*(r+max(q_s_a.loc[s_next])-q_s_a.loc[s][a])

    return q_s_a

def pol_eval_dbl_Q(sample_step, q_s_a_1, q_s_a_2, alpha):

    s=sample_step[0]
    a=sample_step[1]
    r=sample_step[2]
    s_next=sample_step[3]

    if np.random.uniform(0,1) < 0.5 :
        q_s_a_1.loc[s][a]=q_s_a_2.loc[s][a]+alpha*(r+q_s_a_2.loc[s_next][q_s_a_1.loc[s_next].idxmax()]-
    else:
        q_s_a_2.loc[s][a]=q_s_a_2.loc[s][a]+alpha*(r+q_s_a_1.loc[s_next][q_s_a_2.loc[s_next].idxmax()]-

    return [q_s_a_1, q_s_a_2]
```

## Page 11 Double_Q_learning

```
num_ep = 10**5
beg_time = time.time()
q_s_a_1 = q_s_a_init
q_s_a_2 = q_s_a_init
pi = pi_50
exploration_rate = 1

for epi_i in range(1,num_ep+1):
  s_now = "0"

  while (s_now != "70"):
    sample_step = simul_step(pi, s_now, P_normal, P_speed, R_s_a)
    q_s_a = pol_eval_dbl_Q(sample_step, q_s_a_1, q_s_a_2, alpha=max(1/epi_i, 0.05))
    q_s_a_1 = q_s_a[0]
    q_s_a_2 = q_s_a[1]

    if(epi_i % 100 == 0):
      pi = pol_imp(pi, q_s_a_1 + q_s_a_2, epsilon=exploration_rate)

    s_now = sample_step[3]
    exploration_rate = max(exploration_rate * 0.9995 , 0.001)

end_time = time.time()

print(0.5*(q_s_a_1+q_s_a_2).T)
```

```
##           0        10        20        30        40       50        60   70
## n -5.532634 -4.565241 -3.661394 -2.666247 -1.671340 -2.00000 -1.000000  0.0
## s -5.162810 -4.703701 -3.585579 -3.437270 -1.888204 -1.68176 -1.648661  0.0
```

```
print(end_time - beg_time, "sec")
```

```
## 771.320234298706 sec
```

```
print(pi.T)
```

```
##      0   10   20   30   40   50   60   70
## n  0.0  0.5  0.5  0.5  0.5  0.5  0.5  0.5
## s  1.0  0.5  0.5  0.5  0.5  0.5  0.5  0.5
```