Lecture E1. MDP with Model 1

Baek, Jong min

2021-01-31

차 례

Preparation(page 7)

```
gamma = 1
states = np.arange(0,80,10)
p_normal = pd.DataFrame(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
]).reshape(8,8),index=states, columns=states)
p_speed = pd.DataFrame(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,
]).reshape(8,8),index=states, columns=states)
R_s_a = np.array([[-1,-1,-1,-1,0.0,-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])
```

Implementation(page 8)

```
# 1.Initialize V
V_old = pd.DataFrame(np.zeros(shape=(len(states),1)),index=[states])
print(V_old.T)
# 2.Evaluate the Q-function
```

```
##     0    10   20   30   40   50   60   70
## 0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
q_s_a = R_s_a + np.c_[np.dot(gamma*p_normal,V_old),np.dot(gamma*p_normal,V_old)]
print(q_s_a)
# 3.Find the best action for each state
```

```
##     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
```

```
V_new=np.array([q_s_a.apply(max,axis=1)]).reshape(len(states),1)
print(V_new.T)
```

```
## [[-1. -1. -1. -1.  0. -1. -1.  0.]]
```

Implementation(page 11)

```
cnt = 0
epsilon= 10**(-8)
V_old = pd.DataFrame(np.zeros(shape=(len(states),1)),index=[states])
results = V_old.T

while True :
  q_s_a = R_s_a + np.c_[np.dot(gamma*p_normal,V_old),np.dot(gamma*p_speed,V_old)]
  V_new=np.array([q_s_a.apply(max,axis=1)]).reshape(len(states),1)
  if np.max(np.abs(V_new-V_old)).item() < epsilon :
    break
  results = np.vstack([results,V_new.T])
  V_old = V_new
  cnt = cnt+1
```

```
results = pd.DataFrame(results,columns=states)
value_iter_process = results
print(results.head())
```

```
##     0    10   20   30   40    50   60   70
## 0  0.0  0.0  0.0  0.0  0.0  0.00  0.0  0.0
## 1 -1.0 -1.0 -1.0 -1.0  0.0 -1.00 -1.0  0.0
## 2 -2.0 -2.0 -1.6 -1.0 -1.0 -1.50 -1.0  0.0
## 3 -3.0 -2.6 -2.0 -2.0 -1.5 -1.60 -1.0  0.0
## 4 -3.6 -3.0 -3.0 -2.5 -1.6 -1.65 -1.0  0.0
```

```
print(results.tail())
```

```
##           0         10        20        30        40        50   60   70
## 17 -5.107743 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 18 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 19 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 20 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
```

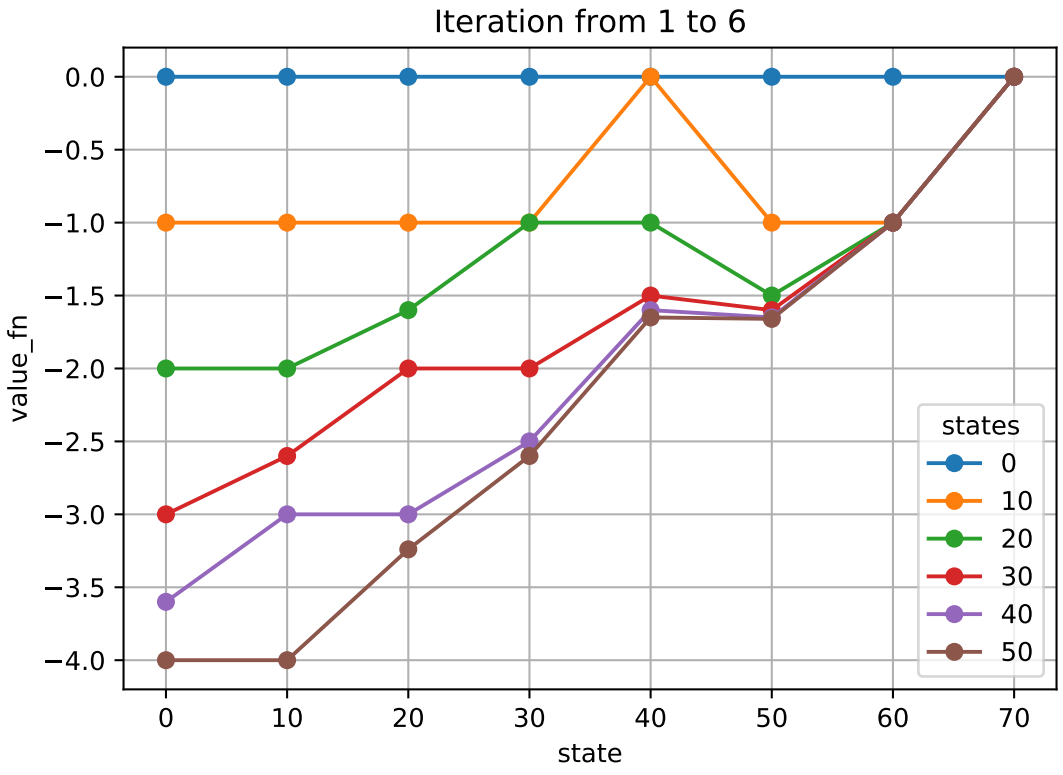
Visualization

```
fig1=results[results.index < 6]
fig2=results[(results.index >= 7)&(results.index < 12)]
fig3=results[(results.index >= 13)&(results.index < 18)]
```

```
plt.plot(fig1.T,marker='o')
```

```
## [<matplotlib.lines.Line2D object at 0x000000002CB459E8>, <matplotlib.lines.Line2D object at 0x000000002CB45A9
```

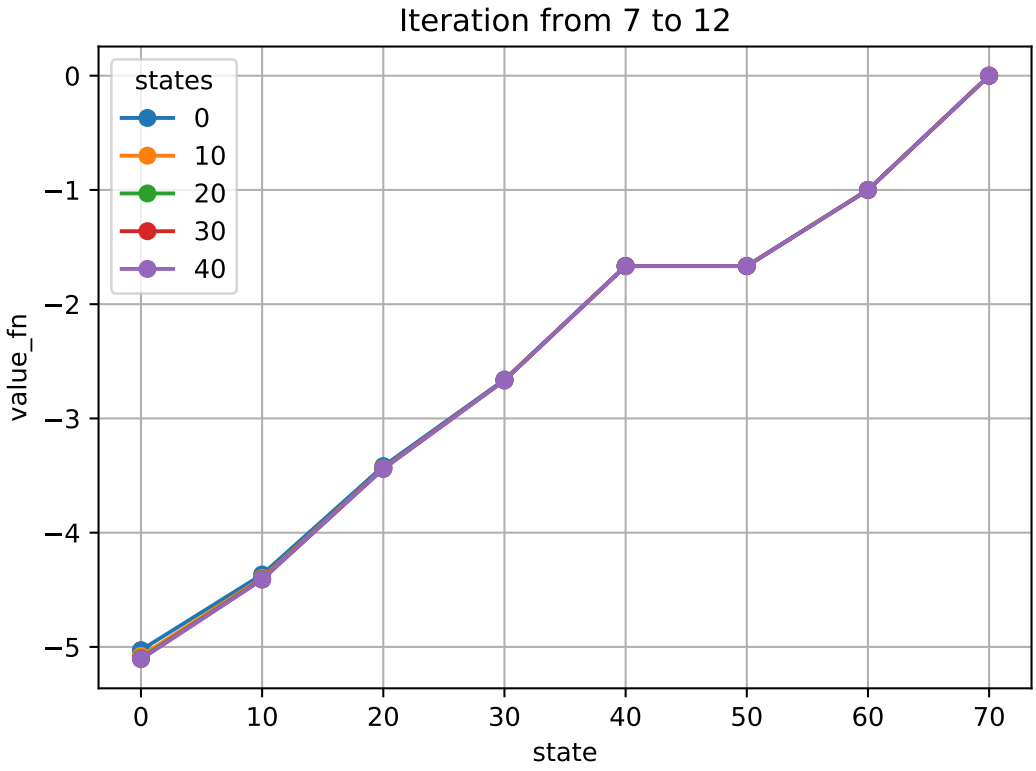
```
plt.legend(fig1.columns,title='states')
plt.grid(True)
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 1 to 6')
plt.show()
```



```
plt.plot(fig2.T,marker='o')
```

```
## [<matplotlib.lines.Line2D object at 0x000000002DC4ABE0>, <matplotlib.lines.Line2D object at 0x000000002DC4AB
```

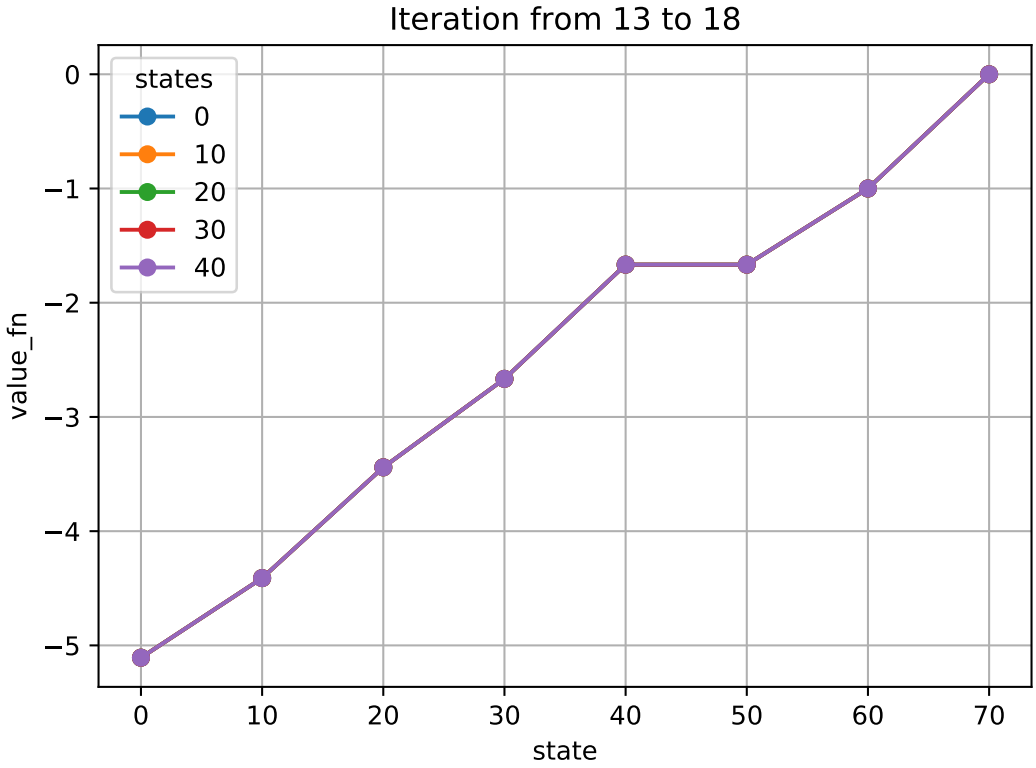
```
plt.legend(fig1.columns,title='states')
plt.grid(True)
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 7 to 12')
plt.show()
```




```
plt.plot(fig3.T,marker='o')
```

```
## [<matplotlib.lines.Line2D object at 0x000000002CB16668>, <matplotlib.lines.Line2D object at 0x000000002CB169
```

```
plt.legend(fig1.columns,title='states')
plt.grid(True)
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 13 to 18')
plt.show()
```


Iteration from 13 to 18
states
0
10
20
30
40
value_fn
0
−1
−2
−3
−4
−5
0
10
20
30
40
50
60
70
state

Optimal policy

```
V_opt = value_iter_process.tail(1).T
print(V_opt.T)
```

```
##           0         10        20        30        40        50   60   70
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
```

```
# It is corresponding optimal policy?
q_s_a = R_s_a + np.c_[np.dot(gamma*p_normal,V_old),np.dot(gamma*p_speed,V_old)]
print(q_s_a)
```

```
##       normal     speed
## 0  -5.410774 -5.107744
## 10 -4.441077 -4.410774
## 20 -3.666667 -3.441077
## 30 -2.666667 -3.344108
## 40 -1.666667 -1.666667
## 50 -2.000000 -1.666667
## 60 -1.000000 -1.666667
## 70  0.000000  0.000000
```

```
pi_opt_vec=pd.DataFrame(q_s_a.apply(np.argmax,axis=1))
print(pi_opt_vec.T)
```

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

```
# You may stop here, or transform pi_opt_vec into a matrix form as below
pi_opt = pd.DataFrame(np.zeros(shape=(len(states),2)),columns=['normal','speed'],index=states)
for i in range(len(pi_opt_vec)):
  pi_opt.iloc[i,pi_opt_vec.iloc[i]] = 1
print(pi_opt.T)
```

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

E3.Rmd

```
"Hello"
```

```
## [1] "Hello"
```