D2 Aggreagated) Markov Reward Process 2

Winter RL Study Members

2021-02-01

Contents

Method 3- Analytic Solution (p.17)

Setting Environment

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter # for setting y-axis decimal points
```

- Option 1. Using matrix ( )

```
P = np.matrix([[0.7,0.3],[0.5,0.5]])
R = np.matrix([[1.5,1]]).reshape(2,1)
gamma = 0.9
v = np.linalg.inv(np.identity(2) - gamma * P) * R
print(v[0])
```

```
## [[13.35365854]]
```

- Option 2. Using array ( )

```
P=np.array([0.7,0.5,0.3,0.5]).reshape(2,2,order='F')
R=np.array([1.5,1.0]).reshape(2,1)

gamma=.9

V=np.dot(np.linalg.inv(np.eye(2)-gamma*P),R)
V
```

```
## array([[13.35365854],
##        [12.74390244]])
```

Method 4- Iterative Solution - by fixed point theorem (p.21)

- Option 1 : Use np.linalg.norm(v_new-v_old) & matrix (   )

```
P = np.matrix([[0.7,0.3],[0.5,0.5]])
R = np.matrix([[1.5,1]]).reshape(2,1)
gamma =0.9
epsilion = 10**(-8)
v_old = np.zeros(2).reshape(2,1)

#do-while
while True:
    v_new =R+gamma*np.dot(P,v_old)
    if np.linalg.norm(v_new-v_old)> epsilion:
        v_old = v_new
        continue
    break
print(v_new)
```

```
## [[13.35365847]
##  [12.74390238]]
```

- Option 2: Use max(np.abs(v_new-v_old) & array (   )

```
import numpy as np
import pandas as pd

R = np.array([1.5,1])[:, None] # return Column vector in 1D
P = np.array([[0.7,0.3],[0.5,0.5]])
gamma =0.9
epsilion = 10**(-8)
v_old = np.repeat(0,2)[:,None]

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

print(v_new)
```

```
## [[13.35365845]
##  [12.74390235]]
```

Option 3: Use amax(v_new - v_old) & array (   )

```
P = np.array([0.7,0.5,0.3,0.5]).reshape(2,2,order = 'F')
R = np.array([1.5,1]).reshape(2,1)
gamma = 0.9
epsilon = 10**-8
v_old = np.zeros(2).reshape(2,1)
v_new = v_new = R + np.dot(gamma*P,v_old)

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

```
print(v_new)
```

```
## [[13.35365845]
##  [12.74390235]]
```

Method 4- Iterative Solution - full iteration process (p.24)

- Option 1: Use append() (   )

```
import numpy as np
import pandas as pd

R = np.array([1.5,1])[:, None] # return Column vector in 1D
P = np.array([[0.7,0.3],[0.5,0.5]])
gamma =0.9
epsilion = 10**(-8)
v_old = np.repeat(0,2)[:,None]

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))> epsilion:
        v_old = v_new
        continue
    break

result=pd.DataFrame(np.array(result).reshape(len(result),2), columns = ['coke', 'pepsi'])

print(result)
```

```
##           coke      pepsi
## 0     0.000000   0.000000
## 1     1.500000   1.000000
## 2     2.715000   2.125000
## 3     3.784200   3.178000
## 4     4.742106   4.132990
## ..         ...        ...
## 174  13.353658  12.743902
## 175  13.353658  12.743902
## 176  13.353658  12.743902
## 177  13.353658  12.743902
## 178  13.353658  12.743902
##
## [179 rows x 2 columns]
```

- Option 2: Use vstack() (   )

```
R=np.array([1.5,1.0]).reshape(2,1)
P=np.array([0.7,0.5,0.3,0.5]).reshape(2,2,order='F')
gamma=0.9
epsilon=10**(-8)

v_old=np.array(np.zeros(2,)).reshape(2,1)
v_new=R+np.dot(gamma*P,v_old)

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

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

results=pd.DataFrame(results, columns=['coke','pepsi'])

print(results.head())
```

```
##         coke    pepsi
## 0  0.000000  0.00000
## 1  1.500000  1.00000
## 2  2.715000  2.12500
## 3  3.784200  3.17800
## 4  4.742106  4.13299
```

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

```
##            coke      pepsi
## 175  13.353658  12.743902
## 176  13.353658  12.743902
## 177  13.353658  12.743902
## 178  13.353658  12.743902
## 179  13.353658  12.743902
```