Assignment 1 by Haobin Tang

1.Write out the MP/MRP definitions and MRP Value Function definition

Markov process:

A Markov process is a stochastic process that satis es the Markov property. We also recall the notion of Markov property from the last lecture. Consider a stochastic process $(s0, s1, s2, \dots)$ evolving according to some transition dynamics. We say that the stochastic process has the Markov property if and only if $P(si|s0, \dots, si-1) = P(si|si-1), \forall i = 1, 2, \dots$,i.e. the transition probability of the next state conditioned on the history including the current state is equal to the transition probability of the next state conditioned only on the current state. In such a scenario, the current state is a su cient statistic of history of the stochastic process, and we say that the future is independent of the past given present.

MRP:

A Markov reward process is a Markov process, together with the speci cation of a reward function and a discount factor. It is formally represented using the tuple (S, P, R, γ) which are listed below:

• S : A nite state space.

• P : A transition probability model that speci es $P(s'|s)$.

• R : A reward function that maps states to rewards (real numbers), i.e $R : S \to R$.

• γ: Discount factor between 0 and 1.

MRP Value Function definition:

The state value function $V_t(s)$ for a Markov reward process and a state $s \in S$ is de ned as the expected return starting from state s at time t, and is given by the following expression:

$$V_t(s) = E[G_t|s_t = s]$$

For G_t, return of a Markov reward process is defined as the discounted sum of rewards starting at time t up to the horizon H, and is given by the following mathematical formula:

$$G_t = \sum_{i=t}^{H-1} \gamma^{i-t} r_i, \forall 0 \leq t \leq H-1$$

.

2.Think about the data structures/class design to represent MP/MRP and implement them with clear type declarations

The data structures should be: state (S): if it is discrete type=int; if it is continuous type=float

Action (A): if it is discrete type=int; if it is continuous type=float

Return (R): type=float

Probability (P): type=float

Discount factor (γ): type=float

3.Code

In [23]:

```
import numpy as np
import random
from typing import Mapping, Set, Tuple, Sequence, Any, Callable
```

1.Specifically the data structure/code design of MRP should be incremental (and not independent) to that of MP

In [1]:

```
def MP_to_MRP(MP_Data,Reward):
    dic={}
    for key in MP_Data:
        if Reward.get(key):
            dic[key]=(MP_Data[key],Reward[key])
        else:
            dic[key]=MP_Data[key]
    for key in Reward:
        if MP_Data.get(key):
            pass
        else:
            dic[key]=Reward[key]
    return dic
```

In [2]:

```
MP_transistion_Data={
        1: {1: 0.1, 2: 0.6, 3: 0.1, 4: 0.2},
        2: {1: 0.25, 2: 0.22, 3: 0.24, 4: 0.29},
        3: {1: 0.7, 2: 0.3, 3:0, 4:0},
        4: {1: 0.3, 2: 0.5, 3: 0.2, 4:0}
    }

Reward={1:1,2:1,3:1,4:-2}

MP_to_MRP(MP_transistion_Data,Reward)
```

Out[2]:

```
{1: ({1: 0.1, 2: 0.6, 3: 0.1, 4: 0.2}, 1),
 2: ({1: 0.25, 2: 0.22, 3: 0.24, 4: 0.29}, 1),
 3: ({1: 0.7, 2: 0.3, 3: 0, 4: 0}, 1),
 4: ({1: 0.3, 2: 0.5, 3: 0.2, 4: 0}, -2)}
```

1. implement the $r(s, s')$ and the $R(s) = \sum_{s'} p(s, s') * r(s, s')$ definitions of MRP

In [136]:

```
def r_to_R(r):
    dic1={}
    dic2={}
    for i in r:
        dic1[i]={}
        for j in r[i]:
            dic1[i][j]=r[i][j][0]
    for i in r:
        dic2[i]=0
        for j in r[j]:
            dic2[i]=dic2[i]+r[i][j][0]*r[i][j][1]
    return MP_to_MRP(dic1,dic2)
```

In [137]:

```
MP_transistion_Data={
        1: {1: (0.1,1), 2: (0.6,1), 3: (0.1,1), 4: (0.2,2)},
        2: {1: (0.25,1), 2: (0.22,1), 3: (0.24,0.5), 4: (0.29,0.5)},
        3: {1: (0.7,1), 2: (0.3,1), 3:(0,-1), 4:(0,-2)},
        4: {1: (0.3,1), 2: (0.5,1), 3: (0.2,0.5), 4:(0,0)}
    }
r_to_R(MP_transistion_Data)
```

Out[137]:

```
{1: ({1: 0.1, 2: 0.6, 3: 0.1, 4: 0.2}, 1.2),
 2: ({1: 0.25, 2: 0.22, 3: 0.24, 4: 0.29}, 0.735),
 3: ({1: 0.7, 2: 0.3, 3: 0, 4: 0}, 1.0),
 4: ({1: 0.3, 2: 0.5, 3: 0.2, 4: 0}, 0.9)}
```

3.Write code to generate the stationary distribution for an MP

In [155]:

```
def stationary_distribution(nS):
    dic={}
    for i in range(nS):
        dic[i]={j:random.random() for j in range(nS)}
    for i in range(nS):
        a=0
        for j in range(nS):
            a=a+dic[i][j]
        for j in range(nS):
            dic[i][j]=dic[i][j]/a
    return dic
P=stationary_distribution(2)
print(P)
```

```
{0: {0: 0.20367321054922707, 1: 0.796326789450773}, 1: {0: 0.4553113
3025903175, 1: 0.5446886697409682}}
```

In [ ]:

In [ ]:

```
```