Assignment 2 by Haobin Tang

1.Write the Bellman equation for MRP Value Function and code to calculate MRP Value Function (based on Matrix inversion method you learnt in this lecture)

$$V_t(s) = E[G_t|s_t = s]$$
$$G_t = \sum_{i=t}^{H-1} \gamma^{i-t} r_i, \forall 0 \leq t \leq H-1$$

Using this, the fact that the horizon is inifnite, and using the stationary Markov property we have for any state $s \in S$:

$$V(s) = V_0(s) = E[G_0|s_0 = s] = E[\sum_{i=0} \gamma^i r_i|s_0 = s] = E[r0|s0 = s] + \sum_{i=0} \gamma^i E[r_i|s_0 = s]$$

There is a nice interpretation of the final result of the above equation, namely that the rssult term $R(s)$ is the immediate reward while the second term $\gamma \sum_{s' \in S} P(s'|s)V(s')$is the discounted sum of future rewards. The value function $V(s)$ is the sum of these two quantities. As $|S| < \infty$, it is possible to write this equation in matrix form as:

$$V = R + \gamma PV$$

We can solve by Matrix inversion:

$$V = (I - \gamma P)^{-1}R$$

Code:

I will use the data in Assignment 1.

In [2]:

```
import numpy as np
```

In [1]:

```
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}
```

In [27]:

```
def Solve_V(MP_transistion_Data,Reward,gama):
    n=len(MP_transistion_Data)
    V_matrix=np.zeros((n,n))
    R_vector=np.zeros(n)
    all_one=np.ones((n,n))
    for i in range(n):
        for j in range(n):
            V_matrix[i][j]=MP_transistion_Data[i+1][j+1]*gama
    for i in range(n):
        R_vector[i]=Reward[i+1]

    Result=np.dot(R_vector,np.linalg.inv(all_one-np.transpose(V_matrix)))

    return Result
```

In [30]:

```
print("the result of value function is " )
print(Solve_V(MP_transistion_Data,Reward,0.9))
```

```
the result of value function is
[ -1.30222105   1.29340737  12.7688172  -12.2311828 ]
```

2.Write out the MDP definition, Policy definition and MDP Value Function definition (in LaTeX) in your own style/notation (so you really internalize these concepts)

We are now in a position to de ne a Markov decision process (MDP). A MDP inherits the basic struc- ture of a Markov reward process with some important key di erences, together with the speci cation of a set of actions that an agent can take from each state. It is formally represented using the tuple (S, A, P, R, γ) which are listed below:

•S :A finite state space.

•A :A finite set of actions which are available from each state s.

•P :A transition probability model that specifies$P(s'|s, a)$.

•R :A reward function that maps a state-action pair to rewards (real numbers), $i.e. R : S \times A \to R$.

•γ :Discount factor $\gamma \in [0, 1]$.

It is important to note that the policy may be varying with time, which is especially true in the case of finite horizon MDPs. We will denote a generic policy by the boldface symbol π, dfined as the infinite dimensional tuple $\pi = (\pi 0, \pi 1, \ldots)$, where π_t refers to the policy at time t.

The state value function $V_t^{\pi(s)}$ for a state $s \in S$ is defined as the expected return starting from the state $s_t = s$ at time t and following policy π, and is given by the expression $V_t^{\pi(s)} = E^{\pi}[G_t|s_t = s]$, where E_π denotes that the expectation is taken with respect to the policy π. Frequently we will drop the subscript π in the expectation to simplify notation going forward. Thus E will mean expectation with respect to the policy unless specified otherwise, and so we can write

$$V_t^{\pi(s)} = E[G_t|s_t = s]$$

.

3.Think about the data structure/class design (in Python 3) to represent MDP, Policy, Value Function, and implement them with clear type definitions

Policy is a dictionary, which corresponds to each states a distribution of the action of that states, which should be float. The type of value function should be float.

4. Write code to convert/cast the r(s,s',a) definition of MDP to the R(s,a) definition of MDP

First to explain the data structure: For each pair of states in [1, nS] and actions in [1, nA], P[state][action] is a tuple of the form (probability, nextstate, reward, terminal) where

```
        - probability: float
            the probability of transitioning from "state" to "nextstate" w
ith "action"
        - nextstate: int
            denotes the state we transition to (in range [0, nS - 1])
        - reward: int
            either 0 or 1, the reward for transitioning from "state" to
            "nextstate" with "action"
        - terminal: bool
          True when "nextstate" is a terminal state, False otherwise
```

In [60]:

```
def r_to_R_2(r):
    dic1={}
    for i in r:
        dic1[i]={}
        for j in r[i]:
            dic1[i][j]=0
            for a,b,c in r[i][j]:
                dic1[i][j] +=a*c
    return dic1
```

In [62]:

```
data={
    1:{1:[(0.2,1,5),(0.4,2,1),(0.3,3,-1),(0.1,4,0)],
       2:[(0.3,1,3),(0.3,2,2),(0.4,3,-1),(0,4,0)],
       3:[(0.4,1,4),(0.2,2,3),(0.3,3,-1),(0.1,4,0)]},
    2:{1:[(0.2,1,5),(0.4,2,4),(0.3,3,-1),(0.1,4,0)],
       2:[(0.3,1,6),(0.3,2,5),(0.4,3,-1),(0,4,0)],
       3:[(0.4,1,-1),(0.2,2,4),(0.3,3,-1),(0.1,4,0)]},
    3:{1:[(0.2,1,-5),(0.4,2,3),(0.3,3,-1),(0.1,4,0)],
       2:[(0.3,1,3),(0.3,2,2),(0.4,3,-1),(0,4,0)],
       3:[(0.4,1,2),(0.2,2,1),(0.3,3,-1),(0.1,4,0)]},
    4:{1:[(0.2,1,8),(0.4,2,0),(0.3,3,-1),(0.1,4,0)],
       2:[(0.3,1,-6),(0.3,2,5),(0.4,3,-1),(0,4,0)],
       3:[(0.4,1,-2),(0.2,2,8),(0.3,3,-1),(0.1,4,0)]}
}

r_to_R_2(data)
```

Out[62]:

```
{1: {1: 1.0999999999999999, 2: 1.1, 3: 1.9000000000000001},
 2: {1: 2.3000000000000003, 2: 2.9, 3: 0.10000000000000003},
 3: {1: -0.09999999999999981, 2: 1.1, 3: 0.7},
 4: {1: 1.3, 2: -0.6999999999999998, 3: 0.5}}
```

5.Write code to create a MRP given a MDP and a Policy

In [75]:

```
def MDR_to_MRP(r, policy):
    dic1={}
    for i in r:
        dic1[i]=[]
        reward=0
        pro=0
        for j in r:
            for action in r[i]:
                reward+=r[i][action][j-1][2]*policy[i][action]
                pro+=r[i][action][j-1][0]*policy[i][action]

            dic1[i].append((pro,j,reward))

    return dic1
```

In [76]:

```
policy={
    1:{1:0,2:1,3:0},
    2:{1:0.2,2:0.6,3:0.2},
    3:{1:0.5,2:0.3,3:0.2},
    4:{1:0,2:1,3:0}
}
MDR_to_MRP(data, policy)
```

Out[76]:

```
{1: [(0.3, 1, 3), (0.6, 2, 5), (1.0, 3, 4), (1.0, 4, 4)],
 2: [(0.30000000000000004, 1, 4.3999999999999995),
  (0.6000000000000001, 2, 9.0),
  (0.9600000000000002, 3, 8.000000000000002),
  (1.0000000000000002, 4, 8.000000000000002)],
 3: [(0.27, 1, -1.2000000000000002),
  (0.6000000000000001, 2, 1.0999999999999999),
  (0.9300000000000002, 3, 0.09999999999999987),
  (1.0000000000000002, 4, 0.09999999999999987)],
 4: [(0.3, 1, -6), (0.6, 2, -1), (1.0, 3, -2), (1.0, 4, -2)]}
```

6. Write out all 8 MDP Bellman Equations and also the transformation from Optimal Action-Value function to Optimal Policy

$$v_\pi(s) = \sum_{a\in A} \pi(a|s) q_\pi(s,a)$$
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_\pi(s')$$
$$v_\pi(s) = \sum_{a\in A} \pi(a|s)(R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_\pi(s'))$$
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a (\sum_{a\in A} \pi(a|s') q_\pi(s',a'))$$

$$v_*(s) = max_{a\in A} q_*(s,a)$$
$$q_*(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_*(s')$$
$$v_*(s) = max_{a\in A} R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_*(s')$$
$$q_*(s,a) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a max_{a\in A} q_*(s',a')$$

In [ ]: