CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Markov decision process and numerical methods

AUTHOR: Qian Chen

SUPERVISOR: Minyi Huang

DATE: January 11, 2021

Contents

1 MDP with finite states and finite actions **4**

1.1 Introduction 4

1.2 MDP with finite states and finite actions 5

1.3 Policy Optimization by Linear Programming 7

2 Continuous MDP with different cases **13**

2.1 Solving MDP with discrete time, continuous state and action spaces 13

2.1.1 Introduction 13

2.1.2 Discretization 14

2.1.3 Example 15

2.2 Solving MDP with continuous time, discrete state and action spaces 20

2.2.1 Example 22

2.3 Solving MDP with continuous time, state and action spaces . . . 24

3 Long run average reward MDP **26**

3.1 Linear programming approach 26

3.2 Relative value iteration 28

3.3 Example of comparison between linear programming and relative value iteration 28

4 Applications of MDP **33**

4.1 Optimal stopping 33

4.1.1 Finite horizon 33

4.2 Examples 35

4.2.1 Secretary Problem 35

4.2.2 Other Example 37

4.3 Infinite horizon 41

5 Conclusion **42**

6 Appendix **43**

6.1 Code for matlab 43

6.1.1 Code for discretization example (2.1.2) 43

6.1.2 Code for example of comparison among linear programming vs oolicy iteration vs value iteration (2.2.1) 48

6.1.3 Code for example of comparison between linear programming and relative value iteration (3.3) 57

6.1.4 Code for Secretary problem 66

Abstract

In this project, I will focus on solving problems using the model of Markov decision process with different approaches. First, I will introduce the basic concept of Markov decision process and mainly focus on solving MDP by linear programming. Then, I will discuss more complex models of MDP and some comparison among several algorithms for different objectives. Finally, I will provide some practical applications of MDP.

1 MDP with finite states and finite actions

1.1 Introduction

A Markov decision process is a discrete time stochastic process. It can be seen as an extension of Markov chains. The difference is the existence of actions and rewards in Markov decision process. In certain situation, a Markov decision process can be reduced to a Markov chain. The process provides a stochastic model for decision making. The results of a Markov decision process are partially random and partially influenced by the decision maker. Markov decision process is useful for solving optimization problems by dynamic programming or linear programming.

Consider a process on the discrete time interval $T = \{0, 1, ..., N\}$. There is a state space $\boldsymbol{S}= \{S_n, n = 0, 1, 2, ..., N\}$. At each time step, the process is in some state i, and any action a that is available in state i. After an action is chosen, the process will move into a new state j at the next time step, and the decision maker received a reward $R(j, a)$.

The transition probability that the process moves into a new state j is influenced by the chosen action a. Once action a is chosen at the state i, the process will be in the new state j which is determined by the conditional probability $P\{S_{n+1} = j|S_0, a_0, S_1, a_1, ..., S_n = i, a_n = a\} = P_{ij}(a)$. Thus, the transition probability of next next state depends on the current state i and the chosen action a. If state i and action a are given, the transition probability depends only upon the given state and action, and it is independent of all previous states and actions. Hence, the state transitions of Markov decision process satisfy the Markov property [1].

1.2 MDP with finite states and finite actions

Generally, a finite state and action MDP can be modeled by $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R})$, where
$\boldsymbol{S}$=$\{S_n, n = 0, 1, 2, ..., N\}$ is a finite state space;
$\boldsymbol{A}$=$\{a_n, n = 0, 1, 2, ..., N\}$ is a finite set of actions allowed in each state;
$\boldsymbol{P}$=$P\{S_{n+1} = j|S_0, a_0, S_1, a_1, ..., S_n = i, a_n = a\} = P_{ij}(a)$ is the probility that given action a in state i at time n will transit to state j at time n+1;
$\boldsymbol{R}$=$R(S_i, a_i)$ is a immediate gain received after transiting from state i to state j by choosing an action a.

The goal of Markov decision process is to find a policy $\boldsymbol{\beta}$ for decision maker to choose action a when the process in state i, where
a policy $\boldsymbol{\beta} = \{\beta_i(a), a \in A, i = 0, 1, 2, ..., N\}$ is a finite set of numbers. In fact, a policy is a probability $\beta_i(a)$ to choose action a when the process in state i [1].

Define $h_t = (S_0, a_0, S_1, a_1...S_{t-1}, a_{t-1}, S_t)$, $t \geq 0$
as an accumulated history of the MDP, $h_0 = (S_0)$ when $t = 0$.

A randomized policy $\boldsymbol{\beta} = \{\beta_i(a), a \in A, i = 0, 1, 2, ..., N\}$ is a conditional probability distribution

$$\beta_i(\boldsymbol{A}|S_i),\ i = 0, 1, 2, ..., N$$

on the actions space $\boldsymbol{A}$. This probability distribution assigns a probability for taking each action in certain state given h_t. Also, $\sum_{i=0} \beta_i(\boldsymbol{A}|S_i) = 1$. In other words, a randomized policy is a mapping of state space $\boldsymbol{S}$ to probability distributions $\beta_i(\boldsymbol{A}|S_i)$ over action space $\boldsymbol{A}$, where the action may depend on h_t.

When there is a policy that determines actions for each state, it is a deterministic policy. For each t, $\beta_i(\boldsymbol{A}|S_i)$ assigns an action $a(h_t)$, which implies any action may depend on the previous accumulated history. A

deterministic policy always prescribes the same actions in each state, so there is no uncertainty of MDP with a deterministic policy.

Let $G_t = \sum_{k=0}^{\infty} \alpha^k R(S_{t+k+1}, a_{t+k+1})$ be the total return of a certain process t. α is a discount factor that discount rewards with rate α^n at time n.

For a given state i and policy β , the value of state i
$V_\beta(i) = E_\beta[G_t|S_t = i]$ is the state-value function of policy β.
Dynamic program equation of V is

$$\begin{aligned}
V_\beta(i) &= E_\beta[G_t|S_t = i] \\
&= E_\beta[\sum_{k=0}^{\infty} \alpha^k R(S_{t+k+1}, a_{t+k+1})|S_t = i] \\
&= E_\beta[R(S_{t+1}, a_{t+1}) + \alpha \sum_{i=0}^{\infty} \alpha^k R(S_{(t+1)+k+1}, a_{(t+1)+k+1})|S_t = i] \\
&= E_\beta[R(S_{t+1}, a_{t+1}) + \alpha G_{t+1}|S_t = i] \\
&= \sum_a \beta_i(a) \sum_j P_{ij}(a)(R(i,a) + \alpha E_\beta[G_{t+1}|S_{t+1} = j]) \\
&= \sum_a \beta_i(a) \sum_j P_{ij}(a)(R(i,a) + \alpha V_\beta(j))
\end{aligned}$$

After taking an action a, value of state S
$Q_\beta(i,a) = E_\beta[G_t|S_t = i, a_t = a]$ is the action-value function of policy β.

Dynamic program equation of Q is [2]

$$\begin{aligned}
Q_\beta(i,a) &= E_\beta[G_t|S_t=i,a_t=a] \\
&= E_\beta[\sum_{k=0}^{\infty}\alpha^k R(S_{t+k+1},a_{t+k+1})|S_t=i,a_t=a] \\
&= E_\beta[R(S_{t+1},a_{t+1})+\alpha G_{t+1}|S_t=i,a_t=a] \\
&= \sum_j P_{ij}(a)(R(i,a)+\alpha E_\beta[G_{t+1}|S_{t+1}=j]) \\
&= \sum_j P_{ij}(a)(R(i,a)+\alpha\sum_{a_j}\beta_j(a_j)E_\beta[G_{t+1}|S_{t+1}=j,a_{t+1}=a_j]) \\
&= \sum_j P_{ij}(a)(R(i,a)+\alpha\sum_{a_j}\beta_j(a_j)Q_\beta(j,a_j))
\end{aligned}$$

1.3 Policy Optimization by Linear Programming

In a Markov decision process problem, we try to choose a policy to maximize $E_\beta[\sum_{i=0}^{\infty}\alpha^i R(S_i,a_i)]$. First, we choose a initial state with probability $P\{S_0=i\}=b_i, i=1,...,n, \sum_{i=1}^{n} b_i=1$.
Under a policy $\boldsymbol{\beta}$, denote y_{ja} as the expected discounted time that choose action a when the process is in state j.

Define the indicator variable

$$I_A=\begin{cases}1, & \text{if } A \text{ occurs}\\ 0, & \text{otherwise}\end{cases}$$

for any event A. And so $y_{ja}=E_\beta[\sum_{n=0}^{\infty}\alpha^n I_{\{S_n=j,a_n=a\}}]$.
i.e. y_{ja} is a certain frequency of state and action pair that is expected in the long run MDP under policy β.

We will show that $\sum_a y_{ja} = E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j\}}]$, where $\sum_a y_{ja}$ is the expected discounted time in state j under policy β.

$$\begin{aligned}
\sum_a y_{ja} &= \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j, a_n=a\}}] \\
&= E_\beta[\sum_a \sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j, a_n=a\}}] \\
&= E_\beta[\sum_{n=0}^{\infty} \alpha^n \sum_a I_{\{S_n=j, a_n=a\}}] \\
&= E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j\}}]
\end{aligned}$$

Since

$$\sum_a I_{\{S_n=j, a_n=a\}} = \begin{cases} 1, & if\ S_n = j\ \ \forall a \\ 0, & otherwise \end{cases}$$

$$\Rightarrow \textstyle\sum_a I_{\{S_n=j, a_n=a\}} = I_{\{S_n=j\}}$$

Next, we will prove two equations.

$$\sum_j \sum_a y_{ja} = \frac{1}{1-\alpha},$$

$$\sum_a y_{ja} = b_j + \alpha \sum_i \sum_a y_{ia} P_{ij}(a)$$

For the first equation,

$$\begin{aligned}
\sum_j \sum_a y_{ja} &= \sum_j \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j\}}] \\
&= \sum_a E_\beta[\sum_j \sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j\}}] \\
&= \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^n \sum_j I_{\{S_n=j\}}] \\
&= \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^n] \ \ (Since \sum_j I_{\{S_n=j\}} = 1 \ when \ S_n = j) \\
&= E_\beta[\sum_{n=0}^{\infty} \alpha^n] = E_\beta[\frac{1}{1-\alpha}] \\
&= \frac{1}{1-\alpha}
\end{aligned}$$

For the second equation,

$I_{\{S_{n+1}=j\}}$, then $\sum_i \sum_a I_{\{S_{n+1}=i,a_n=a\}} = 1$

$$\begin{aligned} I_{\{S_{n+1}=j\}} &= I_{\{S_{n+1}=j\}} \sum_i \sum_a I_{\{S_n=i,a_n=a\}} \\ &= \sum_i \sum_a I_{\{S_n=i,a_n=a\}} I_{\{S_{n+1}=j\}} \\ &= \sum_i \sum_a I_{\{S_n=i,a_n=a,S_{n+1}=j\}} \end{aligned}$$

$$\begin{aligned} E[I_A] &= (1)(P(A)) + (0)(1-P(A)) = P(A) \\ \Rightarrow E[I_{\{S_{n+1}=j\}}] &= \sum_i \sum_a E[I_{\{S_n=i,a_n=a,S_{n+1}=j\}}] \\ \Rightarrow P(S_{n+1}=j) &= \sum_i \sum_a P(S_n=i,a_n=a,S_{n+1}=j) \\ &= \sum_i \sum_a P(S_{n+1}=j,S_n=i,a_n=a) \\ &= \sum_i \sum_a P(S_{n+1}=j|S_n=i,a_n=a)P(S_n=i,a_n=a) \\ &= \sum_i \sum_a P(S_n=i,a_n=a)P_{ij}(a) \\ \Rightarrow E[I_{\{S_{n+1}=j\}}] &= \sum_i \sum_a E[I_{\{S_n=i,a_n=a\}}]P_{ij}(a) \end{aligned}$$

$$\sum_a y_{ja} = E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_n=j\}}] = \sum_{n=0}^{\infty} \alpha^n E_\beta[I_{\{S_n=j\}}]$$

$$= E_\beta[I_{\{S_0=j\}}] + E_\beta[\sum_{n=1}^{\infty} \alpha^n I_{\{S_n=j\}}]$$

$$E_\beta[I_{\{S_0=j\}}] = P(S_0 = j) = b_j$$

$$\sum_{n=0}^{\infty} \alpha^n E_\beta[I_{\{S_n=j\}}] = \sum_{n=0}^{\infty} \alpha^n \sum_i \sum_a E_\beta[I_{\{S_{n-1}=i, a_{n-1}=a\}}] P_{ij}(a)$$

$$= \sum_i \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^n I_{\{S_{n-1}=i, a_{n-1}=a\}}] P_{ij}(a)$$

$$= \alpha \sum_i \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^{n-1} I_{\{S_{n-1}=i, a_{n-1}=a\}}] P_{ij}(a)$$

$$Let\ \ m = n-1\ \Rightarrow\ = \alpha \sum_i \sum_a E_\beta[\sum_{n=0}^{\infty} \alpha^m I_{\{S_m=i, a_m=a\}}] P_{ij}(a)$$

$$= \alpha \sum_i \sum_a y_{ia} P_{ij}(a)$$

$$\sum_a y_{ja} = b_j + \alpha \sum_i \sum_a y_{ia} P_{ij}(a)$$

Finally, we will argue that $E_\beta[\sum_{i=0}^{\infty} \alpha^i R(S_i, a_i)]$ is equivalent to $\sum_j \sum_a y_{ja} R(j, a)$.

$$E_\beta[\sum_{i=0}^{\infty} \alpha^i R(S_i, a_i)] = E_\beta[\sum_{i=0}^{\infty} \alpha^i (\sum_j I_{\{S_n=j\}} R(j, a))]$$

$$= E_\beta[\sum_j \sum_{i=0}^{\infty} \alpha^i I_{\{S_n=j\}} R(j, a)]$$

$$= \sum_j E_\beta[\sum_{i=0}^{\infty} \alpha^i I_{\{S_n=j\}}] R(j, a)$$

$$= \sum_j \sum_a y_{ja} R(j, a)$$

Thus, we can obtain the optimal policy with respect to the expected discountd return by solving the linear program with

Objective function: $\max \sum_j \sum_a y_{ja} R(j,a)$,
Constraints:

$$\sum_j \sum_a y_{ja} = \frac{1}{1-\alpha},$$
$$\sum_a y_{ja} = b_j + \alpha \sum_i \sum_a y_{ia} P_{ij}(a),$$
$$y_{ja} \geq 0, \ all \ j, a;$$

and then defining the policy β^* by $\beta_i^*(a) = \frac{y_{ia}^*}{\sum_a y_{ia}^*}$, where the solutions of the linear program is y_{ja}^* [1].

2 Continuous MDP with different cases

2.1 Solving MDP with discrete time, continuous state and action spaces

2.1.1 Introduction

For MDP with discrete time, continuous state and continuous (discrete) action space. We consider the model (***S,A,Q,R***), where

S is a general state space, which is a subset of multi-demensional space, i.e. $\boldsymbol{S} \in \mathbb{R}^n$, n is the dimension of the state space and $n \in \mathbb{N}$;

A is a general continuous (discrete) action space and $\boldsymbol{A} \in \mathbb{R}^n$, $n \in \mathbb{N}$;

$\boldsymbol{Q} = Q(y|x,a)$ is the transition function when state x process to state y given action a. Since the state space is continuous, the transition function can be stated as a probability density function,

$$Q(y|x,a) = P(S_{t+1} = y|S_t = x, a_t = a)$$

since the process time is discrete;

$\boldsymbol{R} = r(S_t, a_t)$ is the reward function.

When only the state space is continuous, the policy β of choosing an action a has the same form as discrete time, discrete state and action space, so that

$$\beta_x(a) = P(a_t = a|S_t = x),\ \textstyle\sum_a \beta_x(a) = 1,\ 0 \leq \beta_x(a) \leq 1, \text{ for all } x,a$$

The dynamic programming equation of discrete time, continuous state and discrete action space MDP with a policy is

$$V_\beta(x) = \textstyle\sum_a \beta_x(a) \int_y Q(y|x,a)(R(x,y,a) + \alpha V_\beta(y))dy,$$

where α is the discount factor. And thus the optimal policy dynamic programming equation is

$$V^*(x) = \max_{a \in \boldsymbol{A}} \{\textstyle\int_y Q(y|x,a)(R(x,y,a) + \alpha V^*(y))dy\}.$$

When both state space and action space are continuous. $\beta_x(a)$ is a probability density function on action space. In this case,

$$\int_a \beta_x(a)da = 1,\ 0 \leq \beta_x(a) \leq 1, \text{ for all } x,a$$

The dynamic programming equation of discrete time, continuous state and action space MDP with a policy is

$$V_\beta(x) = \int_a \beta_x(a) \int_y Q(y|x,a)(R(x,y,a) + \alpha V_\beta(y))dyda \text{ [3]},$$

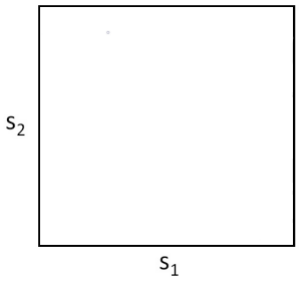
the dynamic programming equation with optimal policy is

$$V^*(x) = \max_{a \in \boldsymbol{A}} \{\int_y Q(y|x,a)(R(x,y,a) + \alpha V^*(y))dy\}.$$

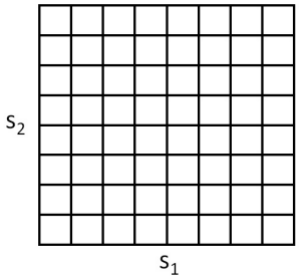
2.1.2 Discretization

We will solve a discrete time, continuous state and continuous action space MDP by discretizing the state and action space, and then apply linear programming approach. The original model is $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{Q}, \boldsymbol{R})$. After discretizing the continuous state space by splitting the time interval into N pieces, we will approximate the original model by the discreted model $(\bar{\boldsymbol{S}}, \bar{\boldsymbol{A}}, \bar{\boldsymbol{P}}, \bar{\boldsymbol{R}})$ by using linear programming.

Another way to achieve discretization is consider an $\mathbb{R}^2$ state space (i.e a 2-dimensional states $\boldsymbol{S} = (S_1, S_2)$)



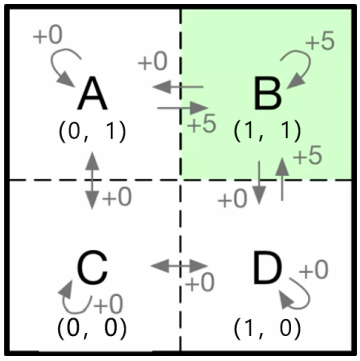
In this case, every pixel point in the square can be a state and the action can be any direction.



We can grid the state space as a discretization. In the picture above, each grid cell is a single discrete state [4]. Thus the state space become $\bar{\boldsymbol{S}} = (\bar{S}_1, \bar{S}_2)$. Then we can approximate the original MDP by solving the discrete MDP.

Since the states and actions in the original model is continuous, there can be infinte actions from states to states so that the transition probability will be very small in each transition. After discretization, we may have only a few states and actions to consider. That means we narrow the scope of the original space of action and state. For the transition probability, it will be determined by the number of different next states resulting by each action-state pair. If there are very many resulting next state, the scale of each transition probability will be very small and vice versa. When an action-state pair only results one next state, the transition probability will be 1. That is how discretization works for continuous MDPs.

2.1.3 Example



Suppose the state space is finite, choose the state space $\boldsymbol{S} = (S_1, S_2) = \{(x, y)|x \in (0, 1),\ y \in (0, 1),\ x,\ y \in \mathbb{R}\}$. After discretization, we have $\bar{\boldsymbol{S}} = (\bar{S_1}, \bar{S_2}) = \{(x, y)|x \in \{0, 1\},\ y \in \{0, 1\},\ x,\ y \in \mathbb{N}\}$. There are total four states in the grid state space.

Denote *A, B, C, D* as the four states (0,1), (1,1), (0,0), (1,0) on each grid cell respectively. The action space consists of moving up, down, left and right. That is, $\bar{\boldsymbol{A}}$= $\{a_u, a_d, a_l, a_r\}$. Actions which will move off the grid keep staying the same place. For example, if the process in state A and take an action up or left, the process will remian in state A. In this grid, the reward is zero everywhere except the reward 5 when the process is in state B, which includes the case the process starts in state B. So our goal is to choose the optimal policy to make the process stay in state B (1,1) as much time as possible.

Linear programming approach :

Use linear programming to optimize the policy
maximize$\sum_j \sum_a y_{ja} R(j, a)$
such that $\sum_j \sum_a y_{ja} = \frac{1}{1-\alpha}$
$\sum_a y_{ja} = b_j + \alpha \sum_i \sum_a y_{ia} P_{ij}(a)$
$y_{ia} \geq 0$, all j,a;

Four states A, B, C, D and denote a_u, a_d, a_l, a_r as action up, down, left, right respectively. Set the discount factor $\alpha = 0.7$, and the probability enter the initial state b_j be equal (i.e. $b_A = b_B = b_C = b_D = 0.25$). Since each chosen action of each state only go to one next state, all the transition probability $P_{ij}(a)$ will be 1. (So here, $\sum_a y_{ja} = b_j + \alpha \sum_i \sum_a y_{ia}$).

max $y_{Aa_r}(-5) + y_{Ba_u}(-5) + y_{Ba_r}(-5) + y_{Da_u}(-5)$

$$\begin{cases} y_{Aa_r} + y_{Ba_u} + y_{Ba_r} + y_{Da_u} = \dfrac{1}{1-0.7} \\ y_{Aa_u} + y_{Aa_d} + y_{Aa_l} + y_{Aa_r} = 0.25 + 0.7(y_{Aa_u} + y_{Aa_l} + y_{Ba_l} + y_{Ca_u}) \\ y_{Ba_u} + y_{Ba_d} + y_{Ba_l} + y_{Ba_r} = 0.25 + 0.7(y_{Aa_r} + y_{Ba_u} + y_{Ba_r} + y_{Da_u}) \\ y_{Ca_u} + y_{Ca_d} + y_{Ca_l} + y_{Ca_r} = 0.25 + 0.7(y_{Aa_d} + y_{Ca_d} + y_{Ca_l} + y_{Da_l}) \\ y_{Da_u} + y_{Da_d} + y_{Da_l} + y_{Da_r} = 0.25 + 0.7(y_{Ba_d} + y_{Ca_r} + y_{Da_d} + y_{Da_r}) \end{cases}$$

$$\begin{cases} y_{Aa_r} + y_{Ba_u} + y_{Ba_r} + y_{Da_u} = \frac{10}{3} \\ 0.3y_{Aa_u} + y_{Aa_d} + 0.3y_{Aa_l} + y_{Aa_r} - 0.7y_{Ba_l} - 0.7y_{Ca_u} = 0.25 \\ -0.7y_{Aa_r} + 0.3y_{Ba_u} + y_{Ba_d} + y_{Ba_l} + 0.3y_{Ba_r} - 0.7y_{Da_u} = 0.25 \\ -0.7y_{Aa_d} + y_{Ca_u} + 0.3y_{Ca_d} + 0.3y_{Ca_l} + y_{Ca_r} - 0.7y_{Da_l} = 0.25 \\ -0.7y_{Ba_d} - 0.7y_{Ca_r} + y_{Da_u} + 0.3y_{Da_d} + y_{Da_l} + 0.3y_{Da_r} = 0.25 \end{cases}$$

by MATLAB, we get the solutions

$y^*_{Aa_r} = 0.3375,\ y^*_{Ba_r} = 2.4083,\ y^*_{Ca_u} = 0.1250,\ y^*_{Ca_r} = 0.1250,\ y^*_{Da_u} = 0.3375$

and all other y^*_{ja} are 0.

Hence, the optimal policies for this problem are

$\beta^*_A(a_r) = \frac{0.3375}{0+0+0+0.3375} = 1$

$\beta^*_B(a_r) = \frac{2.4083}{0+0+0+2.4083} = 1$

$\beta^*_C(a_u) = \frac{0.1250}{0.1250+0+0+0.1250} = 0.5$

$\beta^*_C(a_r) = \frac{0.1250}{0.1250+0+0+0.1250} = 0.5$

$\beta^*_D(a_u) = \frac{0.3375}{0.3375+0+0+0} = 1$

and all other $\beta_i(a)$ are 0.

Under the optimal policy, the maximum expected reward is 15.4167.

But if we set $b_B = 1$ and other $b_j = 0$. The linear program become

$$\max\ y_{Aa_r}(-5) + y_{Ba_u}(-5) + y_{Ba_r}(-5) + y_{Da_u}(-5)$$

$$\begin{cases} y_{Aa_r} + y_{Ba_u} + y_{Ba_r} + y_{Da_u} = \frac{10}{3} \\ 0.3y_{Aa_u} + y_{Aa_d} + 0.3y_{Aa_l} + y_{Aa_r} - 0.7y_{Ba_l} - 0.7y_{Ca_u} = 0 \\ -0.7y_{Aa_r} + 0.3y_{Ba_u} + y_{Ba_d} + y_{Ba_l} + 0.3y_{Ba_r} - 0.7y_{Da_u} = 1 \\ -0.7y_{Aa_d} + y_{Ca_u} + 0.3y_{Ca_d} + 0.3y_{Ca_l} + y_{Ca_r} - 0.7y_{Da_l} = 0 \\ -0.7y_{Ba_d} - 0.7y_{Ca_r} + y_{Da_u} + 0.3y_{Da_d} + y_{Da_l} + 0.3y_{Da_r} = 0 \end{cases}$$

by MATLAB, we get the solutions $y^*_{Ba_r} = 3.3333$, and all other y^*_{ja} are 0.

The optimal policies are $\beta^*_B(a_r) = \frac{3.3333}{0+0+0+3.3333} = 1$ and all other $\beta_i(a)$ are 0. Under the optimal policy, the maximum expected reward is 16.6667.

For this example, we can see the probabilities that enter the initial states will affect the optimal policies and consequencely vary the expected reward.
The above linear program consider the probability of entering the initial state, if we use the linear programming function in the MDP toolbox for MATLAB, we get the optimtal value for the four states,

$V(A) = 13.1667$, the optimal policy for state A is $\beta^*_A(a_d)$,
$V(B) = 11.6667$, the optimal policy for state B is $\beta^*_B(a_d)$,
$V(C) = 11.6667$, the optimal policy for state C is $\beta^*_C(a_r)$,
$V(D) = 16.6667$, the optimal policy for state D is $\beta^*_D(a_d)$.

Policy iteration:

Policy iteration is an algorithm to determine the optimal policy for given states and actions.

We evaluate a given randomized policy by computing the value function

$$V_\beta(x) = R(x, y, a) + \alpha V_\beta(y)$$

Next, find a better action for given state x by improving the policy

$$\beta(x) = \arg\max_{a \in \boldsymbol{A}} \{R(x, y, a) + \alpha V_\beta(y)\}$$

Repeat the previous step until the value funtion $V_\beta(x)$ converges to the optimal value function [2]

$$V^*(x) = R(x, y, a) + \alpha V^*(y)$$

By MATLAB for this example, we get
$V(A) = 13.1667$, the optimal policy for state A is $\beta^*_A(a_d)$,
$V(B) = 11.6667$, the optimal policy for state B is $\beta^*_B(a_d)$,
$V(C) = 11.6667$, the optimal policy for state C is $\beta^*_C(a_r)$,
$V(D) = 16.6667$, the optimal policy for state D is $\beta^*_D(a_d)$.
The result of using policy is almost the same as using linear progamming.

Value iteration:

The idea of value iteration is given a value function with the estimation of k step, find the $k+1$ step value function.

$$V_{k+1}(x) = \max_{a \in \boldsymbol{A}} \{R(x, y, a) + \alpha V_k(y)\}$$

First, initialize $V(y)$ to 0, when the value function does not converged, compute $V(y)$ for each state.

Then, we need to extract the optimal policy [2]

$$\beta^*(x) = \arg\max_{a \in \boldsymbol{A}} \{R(x, y, a) + \alpha V^*(y)\}$$

By MATLAB with using value iteration, we get
$V(A) = 7.4500$, the optimal policy for state A is $\beta^*_A(a_d)$,

$V(B) = 5.9500$, the optimal policy for state B is $\beta_B^*(a_d)$,
$V(C) = 5.9500$, the optimal policy for state C is $\beta_C^*(a_r)$,
$V(D) = 10.9500$, the optimal policy for state D is $\beta_D^*(a_d)$.

The results reveal that value iteration converge much sooner than policy iteration, but the optimal policy for each state is the same as policy iteration and linear programming.

2.2 Solving MDP with continuous time, discrete state and action spaces

Now we consider a MDP with continuous time, state and action spaces still remain discrete. We take a model (***S,A,Q,R***), where
S is the discrete state space;
A is the discrete action space;
Q is defined by a probability distribution $Q_{ij}(\cdot|a)$, represent the transition kernal when the time transition from i to j under action a;
$\boldsymbol{R} = r(S_i, a_i)$ is the reward rate function.

In continuous time and infinite horizon case, we denote the value function as

$$V(i) = E[\int_0^\infty e^{-\alpha t} R(S_t, a_t) dt | S_t = i],\ t \geq 0 \text{ [5]}$$

where α is the discount rate, $0 \leq \alpha \leq 1$. Similar to the dynamic programming equation with discrete time and discrete state and action space of all possible policies,

$$V_\beta(i) = \sum_a \beta_i(a) \sum_j P_{ij}(a)(R(i,a) + \alpha V_\beta(j))$$

We try to derive the dynamic programming equation with the optimal policy.

By dynamic programming, for any $\epsilon > 0$

$$\begin{aligned}
V(i) &= E_\beta[\int_0^\epsilon e^{-\alpha s} R_S dS + \int_\epsilon^\infty e^{-\alpha s} R_S dS | S_0 = i] \\
&= E_\beta[\int_0^\epsilon e^{-\alpha s} R_S dS + e^{-\alpha\epsilon} \int_\epsilon^\infty e^{-\alpha(s-\epsilon)} R_S dS] \\
&\approx \max_{a \in \boldsymbol{A}} [R(i,a)\epsilon + e^{-\alpha\epsilon} EV(S_\epsilon)] \\
&\approx \max_{a \in \boldsymbol{A}} \{R(i,a)\epsilon + (1-\alpha\epsilon)[(1 - \sum_{j \neq i} Q_{ij}(a)\epsilon)V(i) + \sum_{j \neq i, j=1}^{n} Q_{ij}(a)\epsilon V(j)]\} \\
&\approx \max_{a \in \boldsymbol{A}} \{R(i,a)\epsilon + (1-\alpha\epsilon)(V(i) - \sum_{j \neq i} Q_{ij}(a)\epsilon V(i)) + \sum_{j \neq i, j=1}^{n} Q_{ij}(a)\epsilon V(j)\} \\
0 &\approx \max_{a \in \boldsymbol{A}} \{R(i,a)\epsilon + V(i) - \alpha\epsilon V(i) - (1-\alpha\epsilon)\sum_{j \neq i} Q_{ij}(a)\epsilon V(i) + (1-\alpha\epsilon)\sum_{j \neq i, j=1}^{n} Q_{ij}(a)\epsilon V(j) - V(i)\} \\
0 &\approx \max_{a \in \boldsymbol{A}} \{R(i,a)\epsilon - \alpha\epsilon V(i) - (1-\alpha\epsilon)\sum_{j \neq i} Q_{ij}(a)\epsilon V(i) + (1-\alpha\epsilon)\sum_{j \neq i, j=1}^{n} Q_{ij}(a)\epsilon V(j)\}
\end{aligned}$$

By dividing ϵ of both sides, we get

$$0 \approx \max_{a \in \boldsymbol{A}} \{R(i,a) - \alpha V(i) + (1-\alpha\epsilon)[\sum_{j \neq i, j=1}^{n} Q_{ij}(a)V(j) - \sum_{j \neq i} Q_{ij}(a)V(i)]\}$$

When $\epsilon \to 0$, the equation become

$$0 = \max_{a \in \boldsymbol{A}} \{R(i,a) - \alpha V(i) + [\sum_{j \neq i, j=1}^{n} Q_{ij}(a)V(j) - \sum_{j \neq i} Q_{ij}(a)V(i)]\}$$

And thus we get the optimal dynamic programming equation.

$$\begin{aligned}
0 &= \max_{a \in \boldsymbol{A}} \{R(i,a) - \alpha V^*(i) + [\sum_{j \neq i, j=1}^{n} Q_{ij}(a)V^*(j) - \sum_{j \neq i} Q_{ij}(a)V^*(i)]\} \\
\Rightarrow \alpha V^*(i) &= \max_{a \in \boldsymbol{A}} \{R(i,a) + [\sum_{j \neq i, j=1}^{n} Q_{ij}(a)V^*(j) - \sum_{j \neq i} Q_{ij}(a)V^*(i)]\}
\end{aligned}$$

In order to solve the dynamic programing equation, we apply an linear programming approach.

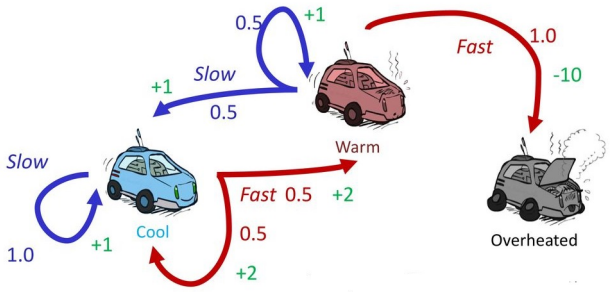
$$\max \sum_i w_i V(i)$$
$$\text{subject to } R(i,a) - \alpha V(i) + [\sum_{j\neq i, j=1}^{n} Q_{ij}(a)V(j) - \sum_{j\neq i} Q_{ij}(a)V(i)] \geq 0$$

where w_i is the probability that the process will first enter the state i. $V^*(i)$ is the solution of the system.

2.2.1 Example

Consider a car wants to travel far qucikly. There are three states, cool, warm and overheated (Denote these states as state C, W, O respectively). And two actions, slow and fast (Denote them as a_s and a_f). Going faster reveives double reward but if too fast the car will burn out.

The states, actions, transition probabilities and rewards are illrstrated below. We take a discount factor 0.9.



Linear programming approach :

By MATLAB, we get the optimal value function for each state.
$V(C) = 15.5000$, the optimal policy is $\beta_C^*(a_f)$,
$V(W) = 14.5000$, the optimal policy is $\beta_W^*(a_s)$,
$V(O) = 0$, the optimal policy is $\beta_O^*(a_s)$.

Policy iteration :

By MATLAB, we get the optimal value function for each state.
$V(C) = 15.5000$, the optimal policy is $\beta_C^*(a_f)$,
$V(W) = 14.5000$, the optimal policy is $\beta_W^*(a_s)$,
$V(O) = 0$, the optimal policy is $\beta_O^*(a_s)$.

There are 3 iterations. The result is the same as linear programming.

Value iteration :

By MATLAB, we get the optimal value function for each state.
$V(C) = 15.4909$, the optimal policy is $\beta_C^*(a_f)$,
$V(W) = 14.4909$, the optimal policy is $\beta_W^*(a_s)$,
$V(O) = 0$, the optimal policy is $\beta_O^*(a_s)$.

There are 70 iterations. The result is close to other two methods above.

The default ϵ is 0.01. If we use smaller ϵ, 0.0001, we get $V(C) = 15.4999$, the optimal policy is $\beta_C^*(a_f)$,
$V(W) = 14.4999$, the optimal policy is $\beta_W^*(a_s)$,
$V(O) = 0$, the optimal policy is $\beta_O^*(a_s)$.

There are 113 iterations. Here the result is much closer to the other two methods. If we use ϵ smaller than this, the results will definitely converge to the results of policy iteration and linear programming.

2.3 Solving MDP with continuous time, state and action spaces

We already seen MDP with continuous time, discrete state, discrete action spaces and MDP with discrete time, continuous state and action space. Now we consider the case that time, state and action spaces all are continuous. We use the model (***S,A,P,Q,R***).

S is the general state space, $\boldsymbol{S} \in \mathbb{R}^n$,$n \in \mathbb{N}$.

A is the general action space, $\boldsymbol{A} \in \mathbb{R}^n$,$n \in \mathbb{N}$.

$\boldsymbol{P} = P(j|i,a)$ is the transition probability function which replace the $\boldsymbol{Q} = Q(y|x,a)$ but work the same way as ***Q*** in section 2.1, thus

$$P(j|i,a) = P(S_{t+\Delta t} = j|S_t = i, a_t = a)$$

$\boldsymbol{Q} = Q_{ij}(\cdot|a)$ in this section, we use the same ***Q*** in section 2.2,
i.e. the transition kernal when the time transition from state i to state j under action a.

$\boldsymbol{R} = r(S_i, a_i)$ is the reward function.

Both section 2.2 and 2.3 are continuous time and infinite horizon, so initially the value function is identical to section 2.2,

$$V(i) = E[\textstyle\int_0^\infty e^{-\alpha t} R(S_t, a_t)dt|S_t = i],\ t \geq 0$$

where α is the same discout rate symbol , $0 \leq \alpha \leq 1$.

In continuous time, state and action spaces, the dynamic programming equation with a policy β become

$$V_\beta(i) = \int_a \beta_i(a) \int_j P(j|i,a) \int_0^\infty \int_0^t e^{-\alpha s} r(i,a) ds dQ_{ij}(t|a) dj da + \int_a \beta_i(a) \int_j P(j|i,a) \int_0^\infty e^{-\alpha t} dQ_{ij}(t|a) V_\beta(j) dj da$$

Alternatively, we can also set action to be discrete, then the equation will be

$$V_\beta(i) = \sum_a \beta_i(a) \int_j P(j|i,a) \int_0^\infty \int_0^t e^{-\alpha s} r(i,a) ds dQ_{ij}(t|a) dj + \sum_a \beta_i(a) \int_j P(j|i,a) \int_0^\infty e^{-\alpha t} dQ_{ij}(t|a) V_\beta(j) dj,$$

but in this section, we only explore case of continuous time, continuous state and continuous action spaces.

We define

$$R(i,j,a) = \int_0^\infty \int_0^t e^{-\alpha s} r(i,a) ds dQ_{ij}(t|a)$$

as the reward received from the transition of state i to state j by action a.

Then, define [5]

$$\gamma = \int_0^\infty e^{-\alpha t} dQ_{ij}(t|a)$$

as the discount factor function for the value of reward received from state i to state j by action a. Consequencely, the dynamic programming equation under policy β is

$$V_\beta(i) = \int_a \beta_i(a) \int_j P(j|i,a)(R(i,j,a) + \gamma V_\beta(j)) dj da$$

which is almost identical to the dynamic programming equation for discrete time, continuous state and action spaces MDP.

The optimal policy dynamic programming equation is

$$V^*(i) = \max_{a \in \boldsymbol{A}} \{ \int_j P(j|i,a)(R(i,j,a) + \gamma V^*(j)) dj \}$$

To expand the optimal dynamic programming equation, we get

$$V^*(i) = \max_{a \in \boldsymbol{A}} \{ \int_j P(j|i,a) \int_0^\infty \int_0^t e^{-\alpha s} r(i,a) ds dQ_{ij}(t|a) dj + \int_j P(j|i,a) \int_0^\infty e^{-\alpha t} dQ_{ij}(t|a) V^*(j) dj \}$$

3 Long run average reward MDP

3.1 Linear programming approach

So far we have seen the case of MDP with finite states and finite actions. When there is a discount factor, the reward will converge to 0 eventually after process is at time n, where $n \to \infty$. Hence, the accumlated reward of a certain starting state is finite. In this case, we are interested in maximizing the accumlated reward by finding an opitmal policy.

However, when there is no discount factor in a finite states and actions MDP, there is no depreciation of reward at any state and any time. As a consequence, the accumlated reward will be infinite. Therefore, the object will be the maximization of the expected average reward per unit time in long run this time.

Apart from the criterion that expected discounted time used in the accumlated reward maximization problem, we consider the probability that the process will be in state i with a chosen action a in long run. That is, the limiting probability of each state and acion pair. Let π_{ia} denote the limimting probability under policy $\boldsymbol{\beta}$,

i.e. $\pi_{ia} = \lim_{n\to\infty} P_\beta\{S_n = i, a_n = a\}$

$\pi = (\pi_{ia})$ must satisfy
(i)$\pi_{ia} \geq 0$ for all i, a,
(ii)$\sum_i \sum_a \pi_{ia} = 1$,
(iii)$\sum_a \pi_{ja} = \sum_i \sum_a \pi_{ia} P_{ij}(a)$,for all j

and thus $\boldsymbol{\beta}$ is defined by
$\beta_i(a) = P\{\boldsymbol{\beta} \text{ chooses } a | \text{state is } i\} = \dfrac{P\{\boldsymbol{\beta} \text{ chooses } a \cap \text{state is } i\}}{P\{\text{state is } i\}} = \dfrac{\pi_{ia}}{\sum_a \pi_{ia}}$

We can see (i), (ii) and (iii) is similar to the constraint of the linear program before. We need to set up the objective function when the vector π is the unknown.

We known that $R(S_i, a_i)$ represent the reward received at time i, $i = 1, 2, ...$
Expected average reward under $\boldsymbol{\beta}$ per unit time can be fomulated as

$$\text{expected average reward under } \boldsymbol{\beta} = \lim_{n\to\infty} E_{\boldsymbol{\beta}}\left[\frac{\sum_{i=1}^{n} R(S_i, a_i)}{n}\right]$$

Now, since $E[R(S_n, a_n)] = \sum_i \sum_a R(i, a)P\{S_n = i, a_n = a\}$
$\lim_{n\to\infty} E[R(S_n, a_n)] = \lim_{n\to\infty} \sum_i \sum_a R(i, a)P\{S_n = i, a_n = a\}$
$= \sum_i \sum_a \pi_{ia} R(i, a)$
which is the limiting expected reward at time n.

On the other hand, the expected value of $R(S_n, a_n)$ also means the long run limiting average of the values $R(S_n, a_n)$, which implies that
expected average reward under $\boldsymbol{\beta} = \sum_i \sum_a \pi_{ia} R(i, a)$

Hence, the policy that can maximize the expected average reward can be obtained by soving the linear program with
Objective function: max $\sum_i \sum_a \pi_{ia} R(i, a)$,
Constraints:

$$\sum_i \sum_a \pi_{ia} = 1$$

$$\sum_a \pi_{ja} = \sum_i \sum_a \pi_{ia} P_{ij}(a), \text{for all } j$$

$$\pi_{ia} \geq 0, \text{all } i, a;$$

If $\boldsymbol{\beta}^* = (\pi^*_{ia})$ maximizes the linear program, the optimal policy will be given by $\boldsymbol{\beta}^*$ where $\beta^*_i(a) = \frac{\pi^*_{ia}}{\sum_a \pi^*_{ia}}$ [1]

3.2 Relative value iteration

Let $R_n(i)$ be the average expected reward when the process starting at state i and go through n time points. Choose $R_0 \in \hat{R}$, where $\hat{R}$ is the space of bounded real valued functions on state space $\boldsymbol{S}$. Then select a ramdom state $k \in \boldsymbol{S}$, given $\epsilon > 0$, set $n = 0$ initially.

For any $i \in \boldsymbol{S}$, the function is

$$R_{n+1}(i) = \min_{a \in \boldsymbol{A}} \{R(i,a) - R_n(k) + \textstyle\sum_{j \in \boldsymbol{S}} P_{ij}(a) R_n(j)\}$$

If $span(R_{n+1} - R_n) < \epsilon$, we choose action $a_\epsilon(i)$, where

$$a_\epsilon(i) = \arg\min_{a \in \boldsymbol{A}} \{R(i,a) + \textstyle\sum_{j \in \boldsymbol{S}} P_{ij}(a) R_n(j)\},$$

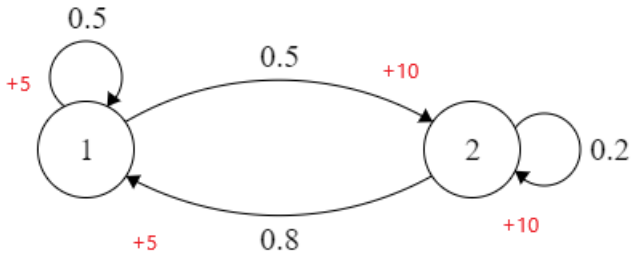
and stop the algorithm.

Otherwise, increase n by 1 and recompute $R_{n+1}(i)$ for a new iteration and so go through the following steps [6].

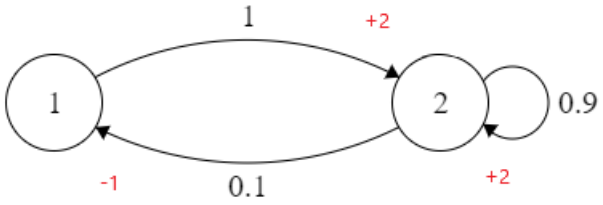
3.3 Example of comparison between linear programming and relative value iteration

We take an exmaple with two states and two actions. Red words are rewards that the process enter a state when an action is chosen.

action 1:



action 2:



Linear Programming:

max $\sum_i \sum_a \pi_{ia} R(i,a)$, such that
$\sum_i \sum_a \pi_{ia} = 1$
$\sum_a \pi_{ja} = \sum_i \sum_a \pi_{ia} P_{ij}(a)$, for all j
$\pi_{ia} \geq 0$, all i,a;

max $5\pi_{1a_1} + 10\pi_{1a_2} - \pi_{2a_1} + 2\pi_{2a_2}$
$\pi_{1a_1} + \pi_{1a_2} + \pi_{2a_1} + \pi_{2a_2} = 1$

$$\begin{cases} \pi_{1a_1} + \pi_{1a_2} = 0.5\pi_{1a_1} + 0.8\pi_{2a_1} + 0.1\pi_{2a_2} \\ \pi_{2a_1} + \pi_{2a_2} = 0.5\pi_{1a_1} + 0.2\pi_{2a_1} + \pi_{1a_2} + 0.9\pi_{2a_2} \end{cases}$$
$$\Rightarrow 0.5\pi_{1a_1} + \pi_{1a_2} - 0.8\pi_{2a_1} - 0.1\pi_{2a_2} = 0$$

by MATLAB, we get the solutions $\pi_{1a_2} = 0.4444$, $\pi_{2a_1} = 0.5556$ and all other π_{ia} are 0.

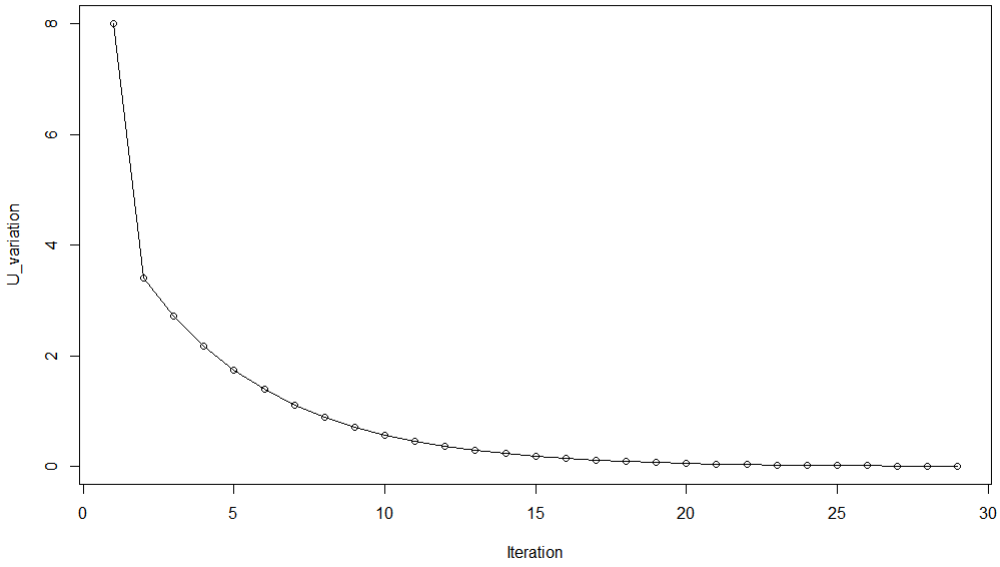
Hence, the optimal policies for this problem are
$\beta_1(a_2) = \frac{0.4444}{0+0.4444} = 1$
$\beta_2(a_1) = \frac{0.5555}{0.5555+0} = 1$
and all other $\beta_i(a)$ are 0. Under the optimal policy, the maximum expected average reward is 3.8889.

Relative Value Iteration:

We run the relative value iteration numerical algorithms by MATLAB, first we take the ϵ as 0.01.

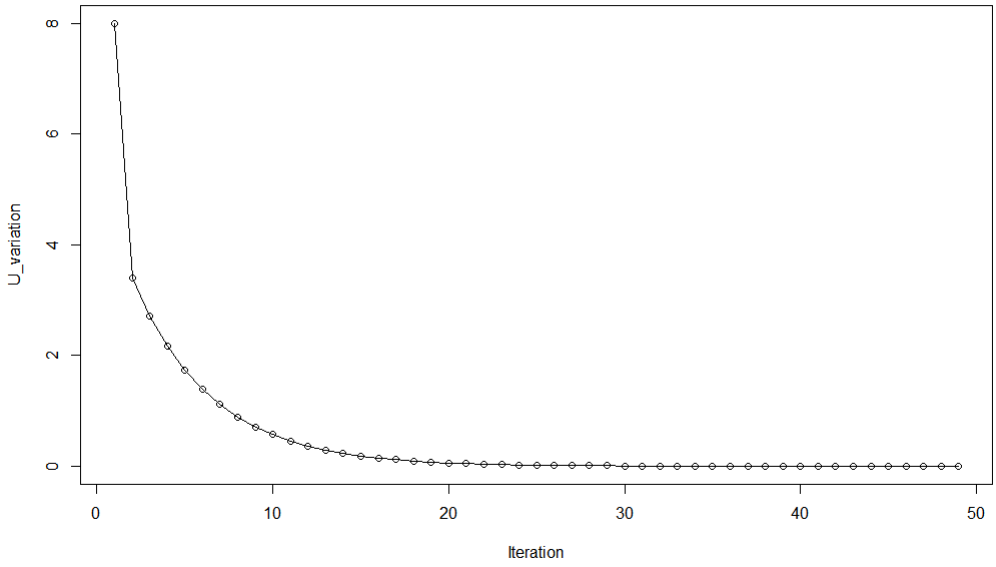
We get the average reward is 3.8852.

There are 29 iteration.



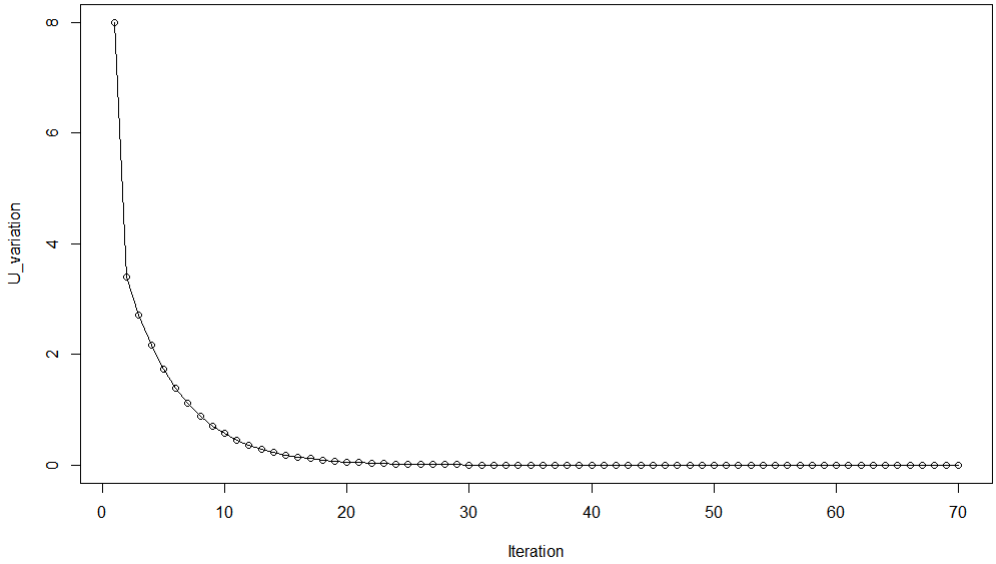
When we take $\epsilon = 0.0001$, the average reward is 3.8888.

There are 49 iteration.



When we take $\epsilon = 0.000001$, the average reward is 3.8889.

There are 70 iteration.



In conclusion, we can get the accurate answer by using linear programming approach. Since relative value iteration is a numerical algorithm, the error will be smaller when taking smaller ϵ. That is, the smaller ϵ we use, the closer result we obtain from relative value iteration algorithm in comparison to linear programming method.

4 Applications of MDP

4.1 Optimal stopping

4.1.1 Finite horizon

In mathematics, optimal stopping is concerned with the problem of taking a certain action at a given time in order to minimize an expected cost or maximize an expected reward. Optimal stopping problems can be applied in areas of statistics, economics and mathematical finance. In this section, we will focus on the reward maximization.

Optimal stopping problems can be solved by first formulating the dynamic programming equation. In discrete time, discrete state and action spaces problem over the finite horizon $t \in \{0, 1, ..., T\}$, stopping rule is associated with two actions *(STOP, CONTINUE)*. At each time step the chosen action is either to stop or to continue. Therefore the action space $\boldsymbol{A} = \{a_s, a_c\}$, where a_s is the stop action and a_c is the continue action respectively. If choose to stop at time t in state $S_t \in \boldsymbol{S}$, the reward with a discount factor α at time t is $\alpha^t R(S_t, a_c)$. We need a policy β as a function of state S_t and action a to specify the stopping decision at each time t. Because sometimes the process is required the stopping take place at some time t. Our goal is to acquire a optimal policy which maximizes the expected discounted reward. Here, we use the model ($\boldsymbol{S,A,P,R}$) in section 1.2, $\boldsymbol{P} = P_{ij}(a)$ is the transition probability from state i to state j under action a.

We define $G_t = \sum_{t=0}^{T} R(S_t, a_t)$ as the accumulative reward function. The optimal value function will be [7]

$$V_t^*(i) = \begin{cases} \max\{G_t, R(i,a) + \alpha \sum_j P_{ij}(a) V_{t+1}^*(j)\}, & \text{if } t < T, \\ G_t, & \text{if } t = T, \end{cases}$$

Then, the optimal stopping policy β^* with respect to the optimal value function V^* is

$$\beta^*_{t,i}(a) = \begin{cases} a_c \quad if\ t < T \text{ and } G_t < R(i,a) + \alpha \sum_j P_{ij}(a) V^*_{t+1}(j), \\ a_s, \qquad \text{otherwise,} \end{cases}$$

for all state $i \in S$ and $t \in T$.

In continuous time, discrete state and action spaces case. The model we use will change to (***S,A,P,Q,R***) in section 2.1, $\boldsymbol{Q} = Q_{ij}(\cdot|a)$ is the transition kernal when the time $t \in [0,T]$ transition from state i to j by action a.

The optimal value function will be

$$V^*(i) = \begin{cases} \max\{G_t, \frac{1}{\alpha}(R(i,a) + [\sum_{j\neq i, j=1}^{n} Q_{ij}(a)V^*(j) - \sum_{j\neq i} Q_{ij}(a)V^*(i)])\}, & \text{if } \ t < T, \\ G_t, & \text{if } t = T, \end{cases}$$

So here the optimal stopping policy β^* for continuous time relate to the optimal value function V^* is

$$\beta^*_i(a) = \begin{cases} a_c \quad \text{if } t < T \text{ and } G_t < \frac{1}{\alpha}(R(i,a) + [\sum_{j\neq i, j=1}^{n} Q_{ij}(a)V^*(j) - \sum_{j\neq i} Q_{ij}(a)V^*(i)]), \\ a_s, \qquad \text{otherwise,} \end{cases}$$

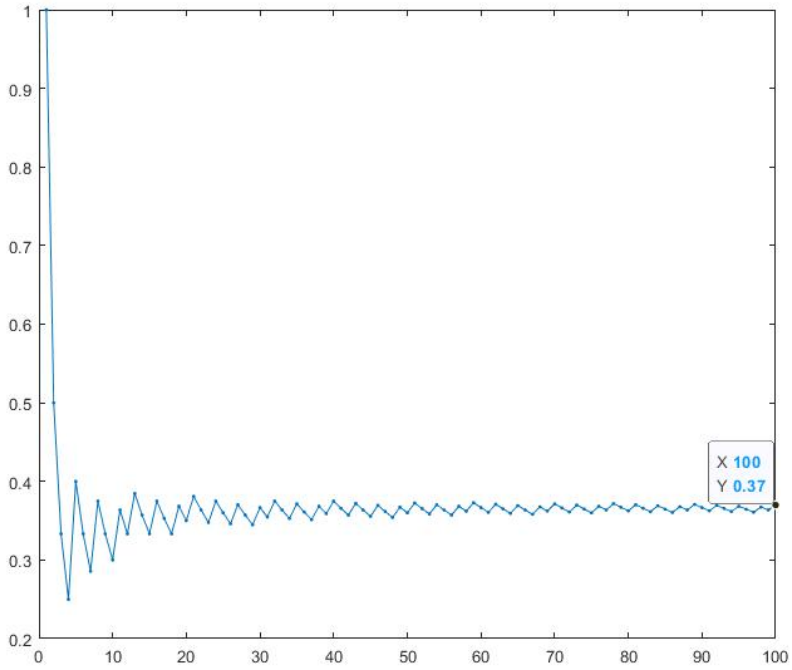
4.2 Examples

4.2.1 Secretary Problem

The secretary problem is a famous problem that uses optimal stopping theory. In this problem, only one position is to be fill. There are n applicants apply for the position, value n is known. The rank of each applicants is in random order, the decision maker will not be able to know the coming applicant is better or worse. After the interview, the decision maker need to accept or reject the current applicant immediately, and the decision is irreversible. The decision maker can only refer to the relative ranks of the applicants that interviewed so far. The object of this problem is to find the optimal probability of choosing the best applicant of the overall group of applicants.

It is proved that the optimal stopping rules prescribes always rejecting the first $\frac{n}{e}$ applicants, where e is the Euler's number. And the optimal probability to select the best applicant successfully is $\frac{1}{e}$ [8].

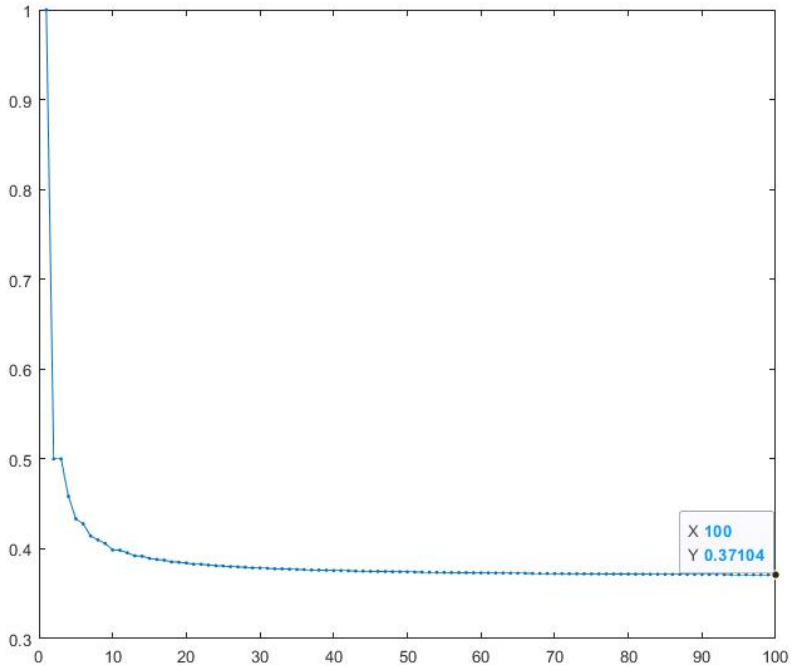
We simulate the problem by first without the stopping rule.



This figure illrustrates the probability of choosing best applicants against number of total applicants.

We can see the range of the fluctuation of the probability gets small as number of total applicants increase. Also, the probability is converge to $\frac{1}{e}$ when n increase.

Next, we simulate with the optimal stopping rule.



This figure illrustrates the probability of choosing best applicants with stopping rule against number of total applicants.

Obviously, with the optimal stopping rule, the probability of choosing the best applicants converges to $\frac{1}{e}$ smoothly.

4.2.2 Other Example

Consider three cards with different numbers, 1, 2, 3. These cards are randomly shuffled and present a sequence to player one by one.

The goal for the player is to decide "stop" when the card with the smallest number is presented. Once the player decide "stop", the game is over. The player will not lose or gain anything if decide "stop" when the card with the smallest number is presented. If not, (i.e. The player decide "stop" at the

wrong time) the player need to pay 10 dollars.

For this stopping problem, we have a MDP model with $T = \{1, 2, 3\}$, $\boldsymbol{S} = \{0, 1, 2\}$. Here state 0 represents the current card is not the best decision so far; state 1 represents the current card is the best choice; when the process is in state 2, means the game is over (i.e. state 2 is an absorbing state). $\boldsymbol{A} = \{a_s, a_c\}$, a_s is stop, a_c is continue.

Since the results of this problem is either "stop at the correct time" or "stop at the incorrect time". With a given policy β, denote the probability of stopping at the correct time by P_β, then the expected cost is $10(1 - P_\beta)$. The original problem is to minimize the expected cost (in order words, it is maximize the probability P_β). We can alternatively switch the problem to be a maxization problem by maximizing $-10(1 - P_\beta)$.

Then, we denote $\boldsymbol{R} = R_t(S, a)$ to be the expect reward rate. Here, $\boldsymbol{R}$ is another form of P_β and it is dependent on time T. For the 6 permutations (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1), we assume they are presented equally likely. Obviously, $R_t(0, a) = 0$ for any time t and any action a. $R_t(1, a_c) = 0$ for any time t. Next,

$$R_1(1, a_s) = \frac{1}{3},\ R_2(1, a_s) = \frac{2}{3},\ R_3(1, a_s) = 1$$

Here, $R_1(1, a_s) = \dfrac{1}{3}$ is obvious since there are only 2 best choice cases of all 6 permutations, the probability is $\dfrac{2}{6} = \dfrac{1}{3}$.

For $R_2(1, a_s)$, as we seem the it as a probability, for simplicity, we express $P[R_t(S, a)] = R_t(S, a)$.

$$P[R_2(1, a_s)|R_1(0, a_c)] = \frac{P[R_2(1, a_s)] \cap P[R_1(0, a_s)]}{P[R_1(0, a_s)]} = \frac{P[R_2(1, a_s)]}{P[R_1(0, a_s)]},$$

$$\Rightarrow P[R_2(1, a_s)] = P[R_1(0, a_s)] \cdot P[R_2(1, a_s)|R_1(0, a_c)] = 1 \cdot \frac{4}{6} = \frac{2}{3}$$

Thus, $R_2(1, a_s) = \frac{2}{3}$.
Similarly, we get $R_3(1, a_s) = 1$.
As a result, we can summarize the reward rate $R_t(1, a_s) = \frac{t}{3}$.

Since in stopping problem, the transition probabilities between state 0 and state 1 is independent to the action a. For $t = 1, 2$,

$$P_{t,00}(a) = \frac{t}{t+1},\ P_{t,01}(a) = \frac{1}{t+1},\ P_{t,10}(a) = \frac{t}{t+1},\ P_{t,11}(a) = \frac{1}{t+1}.$$

$P_{t,S2}(a_s) = 1,\ P_{t,S2}(a_c) = 0.$

At this stage, we can apply the value function $V_t(i)$,

$$V_t^*(1) = max\{\frac{t}{3}, \frac{1}{t+1}V_t^*(1) + \frac{t}{t+1}V_{t+1}^*(0)\},$$
$$V_t^*(0) = max\{0, \frac{1}{t+1}V_t^*(1) + \frac{t}{t+1}V_{t+1}^*(0)\}$$

Since the term $\frac{1}{t+1}V_t^*(1) + \frac{t}{t+1}V_{t+1}^*(0)$ is greater than 0, the second equation can be simplify to

$$V_t^*(0) = \frac{1}{t+1}V_t^*(1) + \frac{t}{t+1}V_{t+1}^*(0).$$

Then, $V_t^*(1) = max\{\frac{t}{3}, V_t^*(0)\}$.

Now we can solve the remains by using backward induction. Start from $t = 2$,

$$V_2^*(0) = \frac{1}{3}V_3^*(1) + \frac{2}{3}V_3^*(0) = \frac{1}{3}\cdot 1 + \frac{2}{3}\cdot 0 = \frac{1}{3},$$
$$V_2^*(1) = max\{-\frac{10}{3}, V_2^*(0)\} = max\{\frac{2}{3}, -\frac{1}{3}\} = \frac{2}{3}.$$

Next, $t = 1$,

$$V_1^*(0) = \frac{1}{2}V_2^*(1) + \frac{1}{2}V_2^*(0) = \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{2},$$
$$V_1^*(1) = max\{\frac{1}{3}, V_1^*(0)\} = max\{\frac{1}{3}, \frac{1}{2}\} = \frac{1}{2}.$$

In order to find the optimal policy β, we go back to the original optimal value function, start from $t = 1$,

$$V_1^*(1) = max\{\frac{1}{3}, \frac{1}{2}V_2^*(1) + \frac{1}{2}V_2^*(0)\} = max\{\frac{1}{3}, \frac{1}{2}\},$$
$$V_1^*(0) = max\{0, \frac{1}{2}V_2^*(1) + \frac{1}{2}V_2^*(0)\} = max\{0, \frac{1}{2}\}.$$

Hence, the action chosen by optimal policy for state 1 at $t = 1$ is a_c, i.e. the optimal policy is $\beta_{1,1}(a_c)$, the action chosen by optimal policy for state 0 at $t = 1$ is a_c, i.e. the optimal policy is $\beta_{1,0}(a_c)$.

For $t = 2$,

$$V_2^*(1) = max\{\frac{2}{3}, \frac{1}{3}V_3^*(1) + \frac{2}{3}V_3^*(0)\} = max\{\frac{2}{3}, \frac{1}{3}\},$$
$$V_2^*(0) = max\{0, \frac{1}{3}V_3^*(1) + \frac{2}{3}V_3^*(0)\} = max\{0, \frac{1}{3}\}.$$

Hence, the action chosen by optimal policy for state 1 at $t = 2$ is a_s, i.e. the optimal policy is $\beta_{2,1}(a_c)$, the action chosen by optimal policy for state 0 at $t = 2$ is a_c, i.e. the optimal policy is $\beta_{2,0}(a_c)$.
So the optimal expected reward for $t = 1$ is $-10(1 - \frac{1}{2}) = -5$, the optimal expected reward for $t = 2$ is $-10(1 - \frac{2}{3}) = -\frac{10}{3}$. In other words, the minimum expected cost at $t = 1$ and $t = 2$ is 5 and $\frac{10}{3}$ respectively.

Overall, the optimal policy is to continue on first card, if the number of the second card is greater than the number of first card then stop, if not, continue to the third card.

4.3 Infinite horizon

In infinite horizon case, we usually consider the expected cost instead of expected reward. We have
$\boldsymbol{C}= C(S_t, a_t)$ as the cost function, which depends on state $S_t = i$, action $a_t = a$ as well as the time t, $t \geq 0$.

As usual, we have a set of policies $\boldsymbol{\beta} = \{\beta_{t,i}(a), t = 0, 1, 2, ..., i = 0, 1, 2, ...\}$ which also depends on the time t.

Then we have the cost value function

$$J_0(i, \beta) = E_\beta[\sum_{t=0}^{T} C(S_t, a_t)|S_0 = i]$$

where T is the first when the stop action a_s is taken by policy β.

Since the horizon is infinite, the cost value $J_0(i)$ can be ∞ if the decision maker never stop the process. Nevertheless, it is no need for a discount factor for infinite horizon problem as there exist policies to limit the expected cost. And therefore we have a well defined value funtion

$$V(i) = \inf_{\beta \in \boldsymbol{\beta}} J_0(i, \beta)$$

5 Conclusion

In conclusion of the whole project, I introduce Markov decision process and its dynamic programming equation as well as linear programming formulation with discount factor. In continuous Markov decision process, I discretize the state space and then use different algorithm to compare the results, the linear program can always get the accurate result but the solution is harder to get. The two numerical methods policy iteration and value iteration can have different results of optimal value, but both them work on choosing the optimal policy. For the comparison between linear program and relative value iteration in long run average reward, still it is not simple to get the solution by linear programming, and there is no funciton for MDP without discount factor in the MDP toolbox for linear programming, so we can only simplify it by hand first, then use the linear programing function to solve it. If there is an acceptable error, relative value iteration may preferable.

6 Appendix

6.1 Code for matlab

6.1.1 Code for discretization example (2.1.2)

Linear Programming by using linear program function directly:

```
c=[0 0 0 -5 -5 0 0 -5 0 0 0 0 -5 0 0 0];
Aeq=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;0.3 1 0.3 1 0 0 -0.7 0 -0.7 0 0 0 0 0 0 0;0 0
0 -0.7 0.3 1 1 0.3 0 0 0 0 -0.7 0 0 0;0 -0.7 0 0 0 0 0 0 1 0.3 0.3 1 0 0 -0.7 0;0 0 0
0 0 -0.7 0 0 0 0 0 -0.7 1 0.3 1 0.3];
beq=[10/3;0.25;0.25;0.25;0.25];lb=zeros(16,1);ub=inf(16,1);
options = optimoptions('linprog','Algorithm','interior-point','Display','iter');
[x,fval]=linprog(c,[],[],Aeq,beq,lb,ub,options);
disp(x);
disp(fval);
```

LP preprocessing removed 0 inequalities, 0 equalities, 4 variables, and 8 non-zero elements.

Iter	Fval	Primal Infeas	Dual Infeas	Complementarity
0	-5.726427e+01	2.239928e+01	8.961477e+00	4.480739e+00
1	-5.171322e+01	1.306368e+01	4.480739e-03	2.067652e+00
2	-1.430764e+01	6.531838e-03	2.240369e-06	1.477663e+00
3	-1.538528e+01	3.265919e-06	1.720017e-09	3.785044e-03
4	-1.541667e+01	2.789102e-12	8.593224e-13	7.823417e-07

Minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in feasible directions, to within the selected value of the function tolerance, and constraints are satisfied to within the selected value of the constraint tolerance.

0

0.0000

0.0000

0.3375

0

0.0000

0.0000

2.4083

0.1250

0

0.0000

0.1250

0.3375

0

0.0000

0.0000

-15.4167

```
c=[0 0 0 -5 -5 0 0 -5 0 0 0 0 -5 0 0 0];
Aeq=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;0.3 1 0.3 1 0 0 -0.7 0 -0.7 0 0 0 0 0 0 0;0 0
0 -0.7 0.3 1 1 0.3 0 0 0 0 -0.7 0 0 0;0 -0.7 0 0 0 0 0 0 1 0.3 0.3 1 0 0 -0.7 0;0 0 0
0 0 -0.7 0 0 0 0 0 -0.7 1 0.3 1 0.3];
beq=[10/3; 0; 1; 0; 0];lb=zeros(16,1);ub=inf(16,1);
options = optimoptions('linprog','Algorithm','interior-point','Display','iter');
[x,fval]=linprog(c,[],[],Aeq,beq,lb,ub,options);
disp(x);
disp(fval);
```

LP preprocessing removed 0 inequalities, 0 equalities, 4 variables, and 8 non-zero elements.

Iter	Fval	Primal Infeas	Dual Infeas	Complementarity
0	-5.618940e+01	2.223641e+01	9.181280e+00	4.590640e+00
1	-5.130406e+01	1.311282e+01	4.590640e-03	1.789226e+00
2	-1.670356e+01	3.277902e-01	2.295320e-06	6.773290e-01
3	-1.669701e+01	9.836169e-03	1.804049e-09	1.787494e-03
4	-1.666668e+01	4.918085e-06	9.019618e-13	9.051931e-07

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the selected value of the function tolerance, and constraints are satisfied to within the selected value of the constraint tolerance.

0

0.0000

0.0000

0.0000

0

0.0000

0.0000

3.3333

0.0000

0

0.0000

0.0000

0.0000

0

0.0000

0.0000

-16.6667

Linear Programming by using MALTAB MDP Toolbox:

P(:,:,1) = [1 0 0 0; 0 1 0 0; 1 0 0 0; 0 1 0 0];

P(:,:,2) = [0 0 1 0; 0 0 0 1; 0 0 1 0; 0 0 0 1];

P(:,:,3) = [1 0 0 0; 1 0 0 0; 0 0 1 0; 0 0 1 0];

P(:,:,4) = [0 1 0 0; 0 1 0 0; 0 0 0 1; 0 0 0 1];

R = [0 5 0 5; 0 0 0 0; 0 0 0 0; 5 5 0 0];

[V, policy] = mdp_LP(P, R, 0.7)

Optimal solution found.

V =

13.1667

11.6667

11.6667

16.6667

policy =

2

2

4

2

Policy Iteration by using MALTAB MDP Toolbox:

[V, policy] = mdp_policy_iteration(P, R, 0.7)

Iteration	Number_of_different_actions
1	2
2	2
3	0

V =

13.1667

11.6667

11.6667

16.6667

policy =

2

2

4

2

Linear Programming by using MALTAB MDP Toolbox:

[V, policy] = mdp_value_iteration(P, R, 0.7)

Iteration	V_variation
1	5
2	3.5
3	0

MDP Toolbox: iterations stopped, epsilon-optimal policy found

V =

7.4500

5.9500

5.9500

10.9500

policy =

2

2

4

2

6.1.2 Code for example of comparison among linear programming vs oolicy iteration vs value iteration (2.2.1)

Linear Programming by using MALTAB MDP Toolbox:

P(:,:,1) = [1 0 0; 0.5 0.5 0; 0 0 0];

P(:,:,2) = [0.5 0.5 0; 0 0 1; 0 0 0];

R(:,1) = [1 1 0];

R(:,2) = [2 2 -10];

[V, policy,cpu_time] = mdp_LP(P, R, 0.9)

Optimal solution found.

V =

15.5000

14.5000

0

policy =

2

1

1

cpu_time =

0.0625

Policy Iteration by using MALTAB MDP Toolbox: [V, policy] = mdp_policy_iteration(P, R, 0.9)

Iteration	Number_of_different_actions
1	2
2	1
3	0

V =

15.5000

14.5000

0

policy =

2

1

1

Value Iteration by using MALTAB MDP Toolbox:

[V, policy] = mdp_value_iteration(P, R, 0.9)

Iteration	V_variation
1	2
2	1.8
3	1.17
4	1.053
5	0.9477
6	0.85293
7	0.76764
8	0.69087
9	0.62179
10	0.55961
11	0.50365
12	0.45328
13	0.40795
14	0.36716
15	0.33044
16	0.2974
17	0.26766
18	0.24089
19	0.2168
20	0.19512
21	0.17561
22	0.15805
23	0.14224
24	0.12802
25	0.11522
26	0.1037
27	0.093327
28	0.083994
29	0.075595
30	0.068035
31	0.061232
32	0.055109
33	0.049598
34	0.044638
35	0.040174
36	0.036157

46 0.012607

47 0.011346

48 0.010212

49 0.0091905

50 0.0082715

51 0.0074443

52 0.0066999

53 0.0060299

54 0.0054269

55 0.0048842

56 0.0043958

57 0.0039562

58 0.0035606

59 0.0032045

60 0.0028841

61 0.0025957

62 0.0023361

63 0.0021025

64 0.0018923

65 0.001703

66 0.0015327

67 0.0013795

68 0.0012415

69 0.0011174

70 0.0010056

MDP Toolbox: iterations stopped, epsilon-optimal policy found

V =

15.4909

14.4909

0

policy =

2

1

1

Value Iteration with a smaller ϵ by using MALTAB MDP Toolbox:

[V, policy,iter] = mdp_value_iteration(P, R, 0.9, 0.0001)

Iteration	V_variation
1	2
2	1.8
3	1.17
4	1.053
5	0.9477
6	0.85293
7	0.76764
8	0.69087
9	0.62179
10	0.55961
11	0.50365
12	0.45328
13	0.40795
14	0.36716
15	0.33044
16	0.2974
17	0.26766
18	0.24089
19	0.2168
20	0.19512
21	0.17561
22	0.15805
23	0.14224
24	0.12802

25	0.11522
26	0.1037
27	0.093327
28	0.083994
29	0.075595
30	0.068035
31	0.061232
32	0.055109
33	0.049598
34	0.044638
35	0.040174
36	0.036157
37	0.032541
38	0.029287
39	0.026358
40	0.023722
41	0.02135
42	0.019215
43	0.017294
44	0.015564
45	0.014008
46	0.012607
47	0.011346
48	0.010212
49	0.0091905
50	0.0082715
51	0.0074443
52	0.0066999
53	0.0060299
54	0.0054269
55	0.0048842

56	0.0043958
57	0.0039562
58	0.0035606
59	0.0032045
60	0.0028841
61	0.0025957
62	0.0023361
63	0.0021025
64	0.0018923
65	0.001703
66	0.0015327
67	0.0013795
68	0.0012415
69	0.0011174
70	0.0010056
71	0.00090506
72	0.00081455
73	0.0007331
74	0.00065979
75	0.00059381
76	0.00053443
77	0.00048099
78	0.00043289
79	0.0003896
80	0.00035064
81	0.00031557
82	0.00028402
83	0.00025562
84	0.00023005
85	0.00020705
86	0.00018634

87 0.00016771

88 0.00015094

89 0.00013584

90 0.00012226

91 0.00011003

92 9.9031e-05

93 8.9127e-05

94 8.0215e-05

95 7.2193e-05

96 6.4974e-05

97 5.8477e-05

98 5.2629e-05

99 4.7366e-05

100 4.2629e-05

101 3.8366e-05

102 3.453e-05

103 3.1077e-05

104 2.7969e-05

105 2.5172e-05

106 2.2655e-05

107 2.039e-05

108 1.8351e-05

109 1.6516e-05

110 1.4864e-05

111 1.3378e-05

112 1.204e-05

113 1.0836e-05

MDP Toolbox: iterations stopped, epsilon-optimal policy found

V =

15.4999

14.4999

0

policy =

2

1

1

iter =

113

6.1.3 Code for example of comparison between linear programming and relative value iteration (3.3)

Linear Programming:

```
c=[-5 -10 1 -2];
Aeq=[1 1 1 1;0.5 1 -0.8 -0.1];
beq=[1;0];lb=zeros(4,1);ub=inf(4,1);
options = optimoptions('linprog','Algorithm','interior-point','Display','iter');
[x,fval]=linprog(c,[],[],Aeq,beq,lb,ub,options);
disp(x);
disp(fval);
```

LP preprocessing removed 0 inequalities, 0 equalities, 0 variables, and added 0 non-zero elements.

Iter	Fval	Primal Infeas	Dual Infeas	Complementarity
0	-1.988120e+01	3.916396e+00	1.666890e+00	1.000000e+00
1	-6.030974e+00	4.248488e-01	8.334450e-04	5.191198e-01
2	-3.874512e+00	2.124244e-04	4.167225e-07	1.148812e-01
3	-3.888724e+00	1.062122e-07	4.823766e-10	6.213340e-05
4	-3.888889e+00	5.381322e-16	2.411504e-13	2.844818e-08

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the selected value of the function tolerance, and constraints are satisfied to within the selected value of the constraint tolerance.

```
0.0000
0.4444
0.5556
0.0000

-3.8889
```

Relative Value Iteration:

mdp_verbose

P(:,:,1) = [0.5 0.5; 0.8 0.2];

P(:,:,2) = [0 1; 0.1 0.9];

R = [5 10; -1 2];

[policy, average_reward, cpu_time] = mdp_relative_value_iteration(P, R)

Iteration	U_variation
1	8
2	3.4
3	2.72
4	2.176
5	1.7408
6	1.3926
7	1.1141
8	0.89129
9	0.71303
10	0.57043
11	0.45634
12	0.36507
13	0.29206
14	0.23365
15	0.18692
16	0.14953
17	0.11963
18	0.095701
19	0.076561
20	0.061249
21	0.048999
22	0.039199
23	0.031359
24	0.025088
25	0.02007
26	0.016056
27	0.012845
28	0.010276
29	0.0082207

MDP Toolbox : iterations stopped, epsilon-optimal policy found

policy =

6.1065

0

average_reward =

2

1

cpu_time =

3.8852

epsilon=0.0001

epsilon =

1.0000e-04

[policy, average_reward, cpu_time] = mdp_relative_value_iteration (P, R, epsilon)

Iteration	U_variation
1	8
2	3.4
3	2.72
4	2.176
5	1.7408
6	1.3926
7	1.1141
8	0.89129
9	0.71303
10	0.57043
11	0.45634
12	0.36507
13	0.29206
14	0.23365
15	0.18692
16	0.14953
17	0.11963
18	0.095701
19	0.076561
20	0.061249
21	0.048999
22	0.039199
23	0.031359
24	0.025088
25	0.02007
26	0.016056
27	0.012845
28	0.010276
29	0.0082207
30	0.0065766

31 0.0052612
32 0.004209
33 0.0033672
34 0.0026938
35 0.002155
36 0.001724
37 0.0013792
38 0.0011034
39 0.00088269
40 0.00070615
41 0.00056492
42 0.00045194
43 0.00036155
44 0.00028924
45 0.00023139
46 0.00018511
47 0.00014809
48 0.00011847
49 9.4778e-05
MDP Toolbox : iterations stopped, epsilon-optimal policy found
policy =
6.1111
0
average_reward =
2
1
cpu_time =
3.8888

epsilon=0.000001
epsilon =

1.0000e-06 [policy, average_reward, cpu_time] = mdp_relative_value_iteration (P, R, epsilon)

Iteration	U_variation
1	8
2	3.4
3	2.72
4	2.176
5	1.7408
6	1.3926
7	1.1141
8	0.89129
9	0.71303
10	0.57043
11	0.45634
12	0.36507
13	0.29206
14	0.23365
15	0.18692
16	0.14953
17	0.11963
18	0.095701
19	0.076561
20	0.061249
21	0.048999
22	0.039199
23	0.031359
24	0.025088
25	0.02007
26	0.016056
27	0.012845
28	0.010276

29	0.0082207
30	0.0065766
31	0.0052612
32	0.004209
33	0.0033672
34	0.0026938
35	0.002155
36	0.001724
37	0.0013792
38	0.0011034
39	0.00088269
40	0.00070615
41	0.00056492
42	0.00045194
43	0.00036155
44	0.00028924
45	0.00023139
46	0.00018511
47	0.00014809
48	0.00011847
49	9.4778e-05
50	7.5823e-05
51	6.0658e-05
52	4.8526e-05
53	3.8821e-05
54	3.1057e-05
55	2.4846e-05
56	1.9876e-05
57	1.5901e-05
58	1.2721e-05
59	1.0177e-05
60	8.1414e-06

61 6.5131e-06

62 5.2105e-06

63 4.1684e-06

64 3.3347e-06

65 2.6678e-06

66 2.1342e-06

67 1.7074e-06

68 1.3659e-06

69 1.0927e-06

70 8.7417e-07

MDP Toolbox : iterations stopped, epsilon-optimal policy found

policy =

6.1111 0

average_reward =

2 1

cpu_time =

3.8889

6.1.4 Code for Secretary problem

CalculateTotal.m

```
function total = CalculateTotal( N )
if N <= 2
total = 1;
else
for t = 1:(N - 1)
sum = 0;
for x = t:(N - 1)
sum = sum + 1/x;
end
if (sum < 1)
break
end
end
total = t - 1;
end
```

CalculateProb.m

```
function prob = CalculateProb( N )
if (N == 1)
prob = 1;
else
sum = 0;
total = CalculateTotal(N);
for x = total:(N - 1)
sum = sum + (1 / x);
end
prob = sum * total / N;
end
```

```
end
```

Graph.m

```
MaxN = 100;
props = zeros(1, MaxN);
probs = zeros(1, MaxN);
N = 1:MaxN;
for x = 1:MaxN
props(x) = CalculateTotal(x)/x;
probs(x) = CalculateProb(x);
end
figure();
plot(N, props,'- .');
figure();
```

References

[1] Sheldon M. (2014) Introduction to Probability Models, Eleventh Edition, 251-254, 274-275

[2] Richard S. Sutton ,Andrew G. Barto (2018) Reinforcement Learning An Introduction, second edition, 47-84

[3] Marco Wiering, Martijn van Otterlo (Eds.) (2012) Reinforcement Learning State-of-the-Art, 207-216

[4] Andrew Ng (2019) CS229 Lecture notes Part XIII Reinforcement Learning and Control. Retrieved from http://cs229.stanford.edu/notes-spring2019/cs229-notes12.pdf

[5] Steven J. Bradtke , Michael O. Duff (1994) Reinforcement Learning Methods for Continuous-Time Markov Decision Problems

[6] Abhijit Gosavi (2002) Reinforcement Learning for Long-Run Average Cost

[7] Frank L. Lewis, Derong Liu (2012) REINFORCEMENT LEARNING AND APPROXIMATE DYNAMIC PROGRAMMING FOR FEEDBACK CONTROL, 452-464

[8] P. R. Freeman (1983) The Secretary Problem and Its Extensions A Review, 189-206