Average cost MDPs

Imen Jendoubi ; ID : 260904662

1) Introduction :

In the case of infinite horizon MDPs, there are several optimality criteria. The most common criteria are the total expected discounted cost and the average expected cost. We have already studied the case of discounted cost. However, in some cases when the control actions are taken very frequently the discount factor is very close to 1. In such cases, it would be preferred to consider the average expected cost rather than the total discounted cost. There are also various cases in which the discounting is unsuitable and the objective is unbounded due to the absence of a zero-cost absorbing terminal state. In such cases, it would be of interest to minimize the average cost per-step. Consequently, the average cost MDPs have wide applications in queuing control theory [1][2].

The average cost MDP model which was first introduced by Howard (1960) is presented. The average cost problem can be connected with the discounted cost problem through a series expansion. In finite state and action spaces, it can be shown that a stationary optimal solution always exists. This is not the case when either the state space or control space is infinite [3]. In this work, the focus will be on the finite state and action spaces. Under this assumption, the existence of a stationary optimal solution can be proved through the consideration of a special type of policy: Blackwell optimal policy (BOP) [4][5]. This particular policy is guaranteed to exist, and is optimal simultaneously for the average cost and discounted cost problems for discount factors close to 1. The series expansion and BOPs are the basis for proving that an optimal solution for the average cost problem verifies a set of optimality equations. If a policy verifies these set of equations then it is optimal.

However, unlike the discounted cost MDP problem, solving these optimality equations is relatively difficult as we need to solve a two-stage problem of optimality equations. This fact has motivated the search for assumptions under which these equations can be simplified. It has been shown that if a condition known as the weak accessibility (WA) is verified in an MDP, the optimality equations are reduced to a single optimality equation similar to the Bellman equation in the discounted cost problems. The WA condition was first introduced by Platzman in [6]. As the average cost criterion depends on the limiting behavior of the MDP, it would be of interest to distinguish different classes of MDPs, and check whether particular classes verify the WA condition.

This work will focus on the computational methods that aim to find the optimal policy for the average cost problems when the WA condition holds. Three main methods are developed based on the concepts of linear programming, value iteration and policy iteration.

The average cost problem can be formulated as a linear program which was first formulated in [7] and [8]. By solving the primal and dual linear program, the optimal average cost and the optimal policy can be found. As the WA results in an equation similar to the Bellman equation, a relative value iteration algorithm can be developed which was first introduced in [9], with error bounds previously introduced in [10]. A transformation must be applied to the transition matrix in case of aperiodic Markov chains to guarantee the convergence of the algorithm. Furthermore, a policy iteration algorithm can be developed to solve the average cost problem. The algorithm was first introduced in [11], and consists of a policy evaluation step followed by a policy improvement step.

This work is structured as follows: First the average cost MDP is presented. Then, the connection between the discounted cost and average cost problems is clarified. The particular BOP is defined and the optimality equations are derived. Then, the WA condition is presented. Finally, computational methods are developed in order to solve the average cost problem under the WA condition, namely the linear programming, value iteration and policy iteration algorithms.

It is noted that due to space limit, only the basic theorems and propositions related to average cost MDPs are presented and the proofs are presented in the appendix.

2) Average cost MDP model

a) Model description and assumptions:

As we are dealing with an infinite horizon problem, we assume that the cost and transition probabilities are time-homogenous. The focus will be on finite state and action spaces.

$X = \{1, \dots, n\}$ is the set of states *;* $P_{ij}(u)$ is the transition probability of going from state *i* to state *j* under control action u; $c(i,u)$ is the cost incurred when the system is in state *i* and control *u* is applied. The set $U(i) = \{1, \dots, m\}$ is the set of control actions available at state $i \in X$. $\pi = (g_0, g_1, \dots)$ denotes the set of all possible policies, with $g_k(i) \in U(i)$. It is noted that using the matrix notation, g_k is an $(n, 1)$ vector.

The objective is to find an optimal policy with respect to the following average cost optimality criterion:

$$J_\pi(x_0) = \limsup_{N \to \infty} \frac{1}{N} E\left\{\sum_{k=0}^{N-1} c(x_k, g_k(x_k))\right\}$$

where $J_\pi(x_0)$ is the average cost per step of starting from an initial state x_0. In the case of infinite horizon problem, we focus only on stationary policies. We will use the following matrix notation:

$$c_g = \begin{bmatrix} c(1, g(1)) \\ \vdots \\ c(n, g(n)) \end{bmatrix} \quad P_g = \begin{bmatrix} p_{11}(g(1)) & \cdots & p_{11}(g(1)) \\ \vdots & & \vdots \\ p_{n1}(g(n)) & \cdots & p_{nn}(g(n)) \end{bmatrix} \quad J_g = \begin{bmatrix} J_g(1) \\ \vdots \\ J_g(n) \end{bmatrix}$$

Where c_g, P_g and J_g are the cost, transition matrix and average cost corresponding to g. We recall that the β-discounted cost is defined as:

$$J_{\beta,g} = \sum_{k=0}^{\infty} \beta^k P_g^k c_g = \left(I - \beta P_g\right)^{-1} c_g$$

b) Connections with the discounted cost problem

A relation between a β-discounted cost problem and the average cost problem can be established by inspection through manipulating the limit as follows:

$$J_g(i) = \lim_{\beta \to 1} (1 - \beta)\, J_{\beta,g}(i) \quad \forall\, i$$

A more formal relation between J_g and $J_{\beta,g}$ can be obtained by using the Laurent series expansion. Therefore, we will use present first a fundamental theorem that represents the basis of the Laurent series expansion.

Theorem 1:

For any transition matrix P, and scalar $\alpha \in (0, 1)$:

$$(I - \alpha P)^{-1} = (1 - \alpha)^{-1} P^* + H + O(|1 - \alpha|)$$

where $O(|1 - \alpha|)$: is an α-dependent matrix such that its limit is 0 when α is close to 1.
The matrices P^* and H are defined by:

$$P^* = \lim_{\alpha \to 1} (1 - \alpha)(I - \alpha P)^{-1} = \lim_{N \to \infty} \frac{1}{N} \sum_{k=0}^{N-1} P^k$$

$$H = \lim_{\alpha \to 1} (I - \alpha P)^{-1} - (1 - \alpha)^{-1} P^* = (I - P + P^*)^{-1} - P^*$$

Proposition1

The following properties hold for the Laurent series expansion:

$$P^* = PP^* = P^*P = P^*P^*$$
$$P^*H = HP^* = 0$$
$$P^* + H = I + PH$$

The proof is provided in Appendix A.

In the following, we apply theorem 1 to find the relation between the discounted and average costs:

Proposition 2:

For any scalar $\beta \in (0,1)$ and stationary policy g:

$$J_{\beta,g} = (1-\beta)^{-1}J_g + h_g + O(|1-\beta|)$$

where
$J_g = P_g^* c_g$ denotes the gain of g.
$h_g = H_g c_g$ denotes the bias of g.
With :
$H_g = \left(I - P_g + P_g^*\right)^{-1} - P_g^*$ denotes the fundamental matrix
$P_g^* = \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} P_g^k$

Proof:
It follows immediately from the application of theorem 1. First we add the subscript g to the matrices P and H, and we multiply (at the right) the main equation of theorem 1 by c_g. Then, the result of proposition 2 follows immediately from defining $J_g = P_g^* c_g$ and $h_g = H_g c_g$.

Implications:

1) From proposition 2, we conclude that:

$$J_g = (1-\beta)\, J_{\beta,g} - (1-\beta)h_g + O(|1-\alpha|^2)$$

2) Using proposition 1, we conclude that:

$$P_g^* h_g = 0 \quad \text{(because } h_g = H_g c_g \text{ and } P_g^* H_g = 0\text{)}$$

3) In case $P_g^* = \lim_{N\to\infty} P_g^N$ (true for aperiodic Markov chains):

$$h_g = \lim_{N\to\infty} \sum_{k=0}^{N} P_g^k\, (c_g - J_g)$$

From this result, we can say that h_g represents a relative cost that is the difference between the total cost incurred by using policy g and the total cost incurred in case the average cost was used as a per-step cost. The proof of the expression above is as follows:
We have from proposition 1:

$$P_g^* + H_g = I + P_g H_g$$

We multiply (at right) by c_g to obtain: $J_g + h_g = c_g + P_g h_g$

which is equivalent to: $c_g - J_g = h_g - P_g h_g$

Therefore: $\sum_{k=0}^{N} P_g^k \left(c_g - J_g\right) = \sum_{k=0}^{N} P_g^k \left(h_g - P_g h_g\right) = h_g - P_g^{N+1} h_g$

By taking the limit $N \to \infty$: $h_g - P_g^* h_g = \lim_{N \to \infty} \sum_{k=0}^{N} P_g^k \left(c_g - J_g\right)$

Or, $P_g^* h_g = 0$ (implication 2) : Therefore: $h_g = \lim_{N \to \infty} \sum_{k=0}^{N} P_g^k \left(c_g - J_g\right)$.

Example 2.1:

We consider a particular policy that results in a Markov chain with a transition matrix and a cost defined as follows: $P = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$; $c = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

The aim is to find the gain and bias based on proposition 2. Therefore, we search first for P^*.

In the general case, it can be shown by induction that if a matrix has the following form: $P = \begin{bmatrix} 1-a & a \\ b & 1-b \end{bmatrix}$ with $a, b \in (0,1); 0 < a + b < 2$; Then: for any integer n:

$$P^n = \frac{1}{a+b} \begin{bmatrix} b & a \\ b & a \end{bmatrix} + \frac{(1-a-b)^n}{a+b} \begin{bmatrix} a & -a \\ -b & b \end{bmatrix}$$

Therefore: $\lim_{n \to \infty} P^n = \frac{1}{a+b} \begin{bmatrix} b & a \\ b & a \end{bmatrix}$. Therefore: $P^* = \lim_{n \to \infty} P^n = \begin{bmatrix} \frac{2}{5} & \frac{3}{5} \\ \frac{2}{5} & \frac{3}{5} \end{bmatrix}$

Consequently: $H = (I - P + P^*)^{-1} - P^* = \begin{bmatrix} 0.48 & -0.48 \\ -0.32 & 0.32 \end{bmatrix}$;

$J = P^* c = \begin{bmatrix} 1.6 \\ 1.6 \end{bmatrix}$; $h = Hc = \begin{bmatrix} -0.48 \\ 0.32 \end{bmatrix}$

Proposition 3:

For any stationary policy g, the gain and bias verify:

$$(S1) \begin{cases} J_g = P_g J_g \\ J_g + h_g = c_g + P_g h_g \end{cases}$$

Proof:

By definition: $J_g = P_g^* c_g$

We multiply by P_g from both sides: $P_g J_g = P_g P_g^* c_g = P_g^* c_g = J_g$

From proposition 1: $P_g^* + H_g = I + P_g H_g$

We multiply the equation above by c_g to obtain: $J_g + h_g = c_g + P_g h_g$.

In the case of discounted cost MDPs, we had to solve the bellman equations. In fact, we had to find the unique fixed point of the bellman operator and use it in policy iteration algorithm. However, the system of equations (S1), cannot be used to find the gain and bias of a policy g as it admits an infinite number of solutions. In fact, if (J_g, h_g) is a solution, then $(J_g, h_g + \alpha)$ is also a solution.

From proposition 2, we can say that if g is an optimal policy that minimizes $J_{\beta,g}$ for β close to 1, then it is an optimal policy for the average cost problem. Based on this remark, we can define a special type of policies to make the link between the discounted cost and average cost problems.

3) Blackwell optimality and optimality equations

a) Blackwell optimality

Definition:

A stationary policy is said to be Blackwell optimal if it is optimal for all the β discounted problems with β being in $[\bar{\beta}, 1]$ and $\bar{\beta} \in (0, 1)$.

Proposition 4:

There exists a Blackwell optimal policy (BOP)

We consider the following proposition that represents the basis for demonstrating that a BOP is optimal for the average cot problem.

Proposition 5:

1) All BOPs have the same gain J^* and bias h^*
2) Let (J^*, h^*) be the gain and bias of a BOP. We have:

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\, J^*(j) \qquad \forall\, i = 1, \dots, n$$

If $\bar{U}(i)$ is the set of controls that reaches the minimum in the equation above, then:

$$J^*(i) + h^*(i) = \min_{u \in \bar{U}(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h^*(j) \right\} \qquad \forall\, i = 1, \dots, n$$

If g^* is BOP, it reaches the minimum of the RHS of these two equations.

Proof: The proof is provided in Appendix A.

b) Optimality equations

We have already established that a BOP is optimal for the average cost problem by proposition 5. Now, we want to find sufficient conditions for the optimality. In fact, we define the optimality equations as:

$$(OEs) \begin{cases} J_g(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\, J_g^*(j) \qquad \forall\, i = 1, \dots, n \\ J_g^*(i) + h_g^*(i) = \min_{u \in \bar{U}(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h^*(j) \right\} \end{cases}$$

where $\bar{U}(i)$ is the set of controls that reaches the minimum of the first equation.

From proposition 5, we conclude that a BOP satisfies the (OEs). Or, from proposition 4, we know that there exists a BOP. Therefore, the (OEs) have at least one solution. In the following, we show that the (OEs) are sufficient conditions for optimality. First, we recall (Bellman operator):

$$B_g V = c_g + \beta P_g V; \qquad (BV)(i) = \min_u \left\{ c(i,u) + \textstyle\sum_{j=1}^{n} p_{ij}(u)\, V(j) \right\} \quad \forall i$$

Proposition 6:

Let g be an admissible policy (stationary or not). We consider two vectors J and h that satisfy the two inequalities:

$$J \le P_g J \quad (*)$$
$$J + h \le B_g h \quad (**)$$

Then: $J \le J_g$. If these inequalities are verified with equalities, then: $J = J_g$

Proof:
The proof is given in appendix A.

Proposition 7:

We suppose that the (OEs) are satisfied by a pair (J^*, h^*). Then: J^* is the optimal average cost vector. Moreover, if a stationary policy g^* satisfies the RHS of (OEs), then g^* is an optimal policy.

Proof: The proof is provided in appendix A

Example 3.1:

To capture how we can find a solution to the optimality equations, we consider the MDP below:

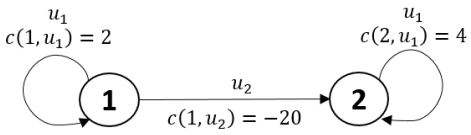


Figure 1: Symbolic representation of example 3.1

Thus, $U(1)=\{u_1,u_2\}$ and $U(2)=\{u_1\}$. The aim is to find a policy g that satisfies the optimality equations. By inspection, the optimal solution g that minimizes the long-run average cost is to apply action 1 at state 1 and action 1 at state 2.

First, we write the optimality equation, and then check if the policy g satisfies these optimality equations. If g satisfies the (OEs) then it is optimal. We consider the 1st set of equations in order to find the set $\bar{U}(i)$:

$$(1)\begin{cases} J^*(1)=\min\{J^*(1),J^*(2)\} \\ J^*(2)=J^*(2) \end{cases}$$

because at state 1, there are two possible actions u_1 or u_2, while only one action is possible at state 2.

Therefore: $\bar{U}(1)=\{u_1\}$ and $\bar{U}(2)=\{u_1\}$. Consequently, the second set of equations is:

$$(2)\begin{cases} J^*(1)+h^*(1)=c(1,u_1)+h^*(1)=2+h^*(1) \\ J^*(2)+h^*(2)=4+h^*(2) \end{cases}$$

Now, we consider the previously described particular case: $g=\begin{bmatrix}u_1\\u_1\end{bmatrix}$. The transition matrix corresponding to this policy is the (2, 2) identity matrix i.e. $P_g=I_{(2,2)}$. Based on proposition 2, $P_g^*=\lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N-1}P_g^k=I_{(2,2)}$; $J^*(1)=c_g(1)=1$; $J^*(2)=c_g(2)=2$; $h^*(1)=h^*(2)=0$. Obviously, the particular policy g satisfies the optimality equations (1) and (2). Thus, it is optimal.

To conclude, the (OEs) for the average cost problem are similar to the Bellman equation. However, in the case of (OEs), we have a two-stage minimization. In fact, we need to find first the constraint set $\bar{U}(i)$ from the first equation, and afterwards solve the 2nd equation. This makes the (OEs) more difficult to solve. To this aim, we search for conditions under which the (OEs) can be simplified further.

c) Case of equal average cost for all initial states:

We consider the special case in which the average cost is the same for any initial state i, i.e.

$$J^*(i)=\lambda \qquad \forall i$$

Consequently, the 1st equation of (OEs) is always satisfied under this assumption, and the sets $\overline{U}(i)$ and $U(i)$ are the same. Therefore, the optimality equations are reduced to:

$$\lambda + h(i) = \min_{u \in U(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h(j) \right\} \quad \forall i = 1, \dots, n$$

The equation is known as Bellman's equation for the average cost problem. We resume these results in the following proposition:

Proposition 8:

If a scalar λ and h a vector that satisfy:

$$\lambda + h(i) = \min_{u \in U(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h(j) \right\} \quad \forall i = 1, \dots, n$$

Then, the optimal cost is λ ($\lambda = \min_{g} J_g(i) = J^*(i)\ \forall i$). Moreover, if g^* attains the minimum in the first expression, $\lambda = J_{g^*}(i) \quad \forall i$=1, …, n.

The first equation can be written in matrix format as follows: $\lambda e + h = Bh$ where e is the unit vector.

Under the condition of equal average costs starting from any initial condition, we can evaluate the cost of any policy based on the following result:

Proposition 9:

We consider a stationary policy g. If a scalar λ_g and a vector h_g satisfy:

$$\lambda_g + h(i) = c\big(i, g(i)\big) + \sum_{j=1}^{n} p_{ij}\big(g(i)\big)\, h(j) \quad \forall i = 1, \dots, n$$

Then: $\lambda_g = J_g(i) \quad \forall i = 1, \dots, n$
The 1st equation is equivalent to : $\lambda_g e + h = B_g h$

In the following, we represent a condition under which the (OEs) can be simplified based on the equal average cost for all initial states assumption: The weak accessibility (WA) condition.

4) WA condition and MDPs classification

a) WA condition

As the optimality equations are simplified in the case of equal average cost for all initial states, it would be of interest to find conditions under which the hypothesis of equal average costs for all initial states is verified. To this aim, we define the weak accessibility (WA) condition, and whenever this condition is verified we are in the case of equal average cost for all initial states. Before introducing the concept of WA, we define some concepts related to the states of a Markov chain.

First, we define the concept of accessibility as follows. A state i is accessible from state j if:

$$\exists\ stationary\ policy\ g\ and\ k\ integer\ such\ that\colon\ P(x_k = j | x_0 = i,\ g) > 0$$

We say that two states communicate if i is accessible from j, and j is accessible from i. The states that communicate with each other form a communicating class.

A state is recurrent if whenever the Markov chain visits it once at some time, it will remain in this state forever. A state is said to be transient if it is not recurrent, that is, at some time the Markov chain will leave that state, and will never return to it [12]. Figure 2 represents a Markov chain of some policy comprising 4 states. It can be seen that S1 and S2 communicate with each other, and therefore form one communicating class which is transient. The communicating class formed by S3, S4 and S5 is recurrent.

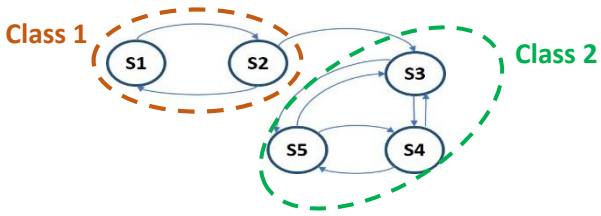


Figure 2: Example of a Markov chain

It is noted that the class formed by S1 and S2 is periodic with period 2. In fact, if we start from S1, we return to S1 at times steps, 4, 6, 8,… . The same is true for state S2. However, class 2 is aperiodic because if we start at S4 we can return to it in 3 steps if we follow the path S4-S5-S3-S4, or in 2 steps if we follow the path S4-S3-S4.

We say that the WA condition holds if we can partition the state space into two subsets S1 and S2 such that:

*) The states in the subset S1 are all transient under any stationary policy

*) For any states i and j in S2, j is accessible from i

It can be shown as in [2] that when the WA condition holds, the optimal average cost is the same for all initial states.

b) Classification of MDPs

In this part, we investigate if some special structures of MDPs verify the WA condition. In general, we distinguish between two main classes of MDPs [1]: The multichain and the weakly communicating classes.

An MDP is multichain if for every policy, the state space can be partitioned into two or more recurrent classes and a transient class.

An MDP is weakly communicating if we can find a set of states in which each state is accessible from the other states within the same set under some policy, plus some other transient states under every policy. The weakly communicating class can be divided into other subclasses. We will focus on a particular subclass that is the class of unichain MDPs. In fact, an MDP is said to be unichain if for every policy we can partition the state space into one recurrent class and some transient states.

From the definition of a unichain MDP, we can see that the WA condition holds. Therefore, the optimal average cost is the same for all initial states. However, the WA condition is more general than the unichain condition, and is usually easier to check. In fact, to verify if the unichain condition holds, one should enumerate all possible stationary policies and check if the Markov chain corresponding to each policy is unchain or not. It has been shown in [13] that this verification process in an NP-complete problem. In contrast, the WA condition is easier to check by using graph connectivity algorithms. If the WA condition holds, then it is not necessary that all the stationary policies are unichain. However, we can always construct a stationary policy that is unichain [2].

It is noted that if the MDP is multichain then the equal average cost hypothesis does not hold.

To conclude, the WA condition that implies the condition of equal average cost for all initial states results in more simplified optimality equations. In particular, we distinguish the class of unichain MDP which verifies the WA condition. The simplified optimality equation makes easier the development of computational methods for solving average cost MDPs.

5) Linear programming

a) Preliminaries

This part aims to solve the average cost MDP problem using linear programming. We assume that the WA condition holds.

We recall that, in case the WA condition is verified, the optimality equations are:

$$\lambda e + h = Bh = \min_g B_g h$$

Therefore: $\lambda e + h \leq B_g h$ for any g. Consequently, we obtain by proposition 6: $\lambda e \leq \lambda_g e$ which is equivalent to $\lambda \leq \lambda_g$ for any policy g. Therefore, λ is the largest scalar that verifies $\lambda e + h \leq Bh$.

Based on this result, we can write the linear program when the WA condition holds as follows:

$$\max \boldsymbol{\lambda}$$

$$s.t. \quad \boldsymbol{\lambda + h(x) - \sum_{j=1}^{n} p_{xj}(u) h(j) \leq c(x,u) \;\; \forall\, x = 1, \dots, n \; and \; \forall\, u \in U(i)}$$

By solving this problem, we obtain the optimal average cost λ^* (and bias). However, we cannot determine the optimal policy using this formulation. To clarify this point, we consider two examples.

Example 5.1:

We consider the following MDP:

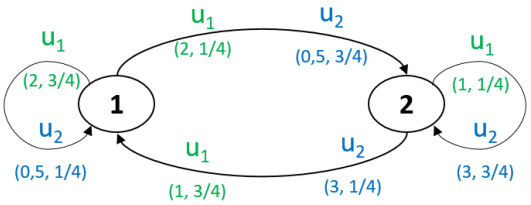


Figure 2: Symbolic representation of example 5.1.

Consider $u_1 = 0;\ u_2 = 1$.

$U(1) = \{u_1, u_2\}\ and\ U(2) = \{u_1, u_2\}$ are the sets of control actions available at state 1 and state 2, respectively. At each state (1 or 2), we can choose either action $u_1 = 0$ or $u_2 = 1$.

$P(u_1) = \begin{bmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix}$; $P(u_2) = \begin{bmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix}$; We have: $C = \begin{bmatrix} 2 & 0.5 \\ 1 & 3 \end{bmatrix}$ with the rows denoting the state, and the columns denoting the action (e.g. $C(1,2) = c(1,u_2)$). It can be verified that the MDP is unichain because we can have 4 possible stationary policies. The Markov chain corresponding to each of these policies contains only one recurrent class.

We solve this average cost MDP problem by implementing the primal linear program. The matrix formulation of the problem and the Matlab code are illustrated in Appendix B. We find an optimal average cost equal to $\lambda = 0.75$. The bias vector associated with the optimal average cost is $h = \begin{bmatrix} 0 \\ -\frac{1}{3} \end{bmatrix}$.

Example 5.2:

In this example, we consider the infinite horizon version of the service rate control in queuing systems problem (assignment 3, problem1). We solve it using the average cost as an optimality criterion. We try to find a primal linear program based solution for this average cost MDP problem. The matrix formulation of the problem is detailed in appendix B. The code is implemented with Matlab (appendix B). We find that the optimal average cost is -5.8841, and the bias is represented in figure 3.

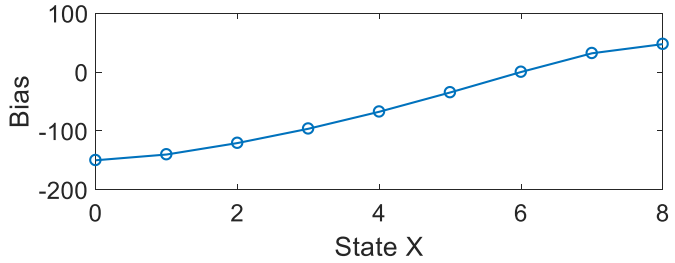


Figure 3: Bias corresponding to the primal linear problem

Therefore, by the primal problem we can only determine λ and h using the primal linear program.

To circumvent this problem, we consider the dual linear program which is defined by:

$$\min \sum_{x=1}^{n} \sum_{u \in U(i)} c(i,u)\, z(x,u)$$

$$s.t. \sum_{u \in U(i)} z(x,u) - \sum_{y=1}^{n} \sum_{u \in U(i)} p_{xy}(u)\, z(y,u) = 0 \qquad \forall\, x = 1, \dots, n$$

$$\sum_{x=1}^{n} \sum_{u \in U(i)} z(x,u) = 1$$

$$z(x,u) \geq 0 \qquad \forall\, u \in U(i), \forall\, x = 1, \dots, n$$

From the primal-dual theory, we have that the minimum value of the cost in dual problem has a minimum equal to λ^*. The primal problems allows to know only the optimal λ^*. In contrast, considering the dual program allows to get the optimal policy. Let $z^*(x,u)$ be the optimal solution of the dual problem. We define the sets:

$$C^* = \left\{ x \mid \sum_{u \in U(i)} z^*(x,u) > 0 \right\}$$

$$U^*(x) = \{u \in U(x) \mid z^*(x,u) > 0\} \qquad i \in C^*$$

By the second equality of the dual program, C^* is non-empty and $U^*(x)$ is also non-empty for any x. Then, the optimal policy is defined as follows:

$$g^*(x) = \begin{cases} any\ u \in U^*(x) & if\ i \in I^* \\ any\ u \in U^*(x) & if\ i \notin I^* \end{cases}$$

If λ^*, h^* are the optimal solution of the primal problem, and z^* is the solution of the dual problem, from the primal-dual theory we have:

$$\forall\, u \in U(i), \forall\, i = 1, \dots, n$$

$$z(i,u) > 0\ \textit{implies} \quad \lambda^* + h^*(i) = c(i,u) + \sum_{j=1}^{n} p_{ij}(u) h^*(j)$$

This is equivalent to:

$$\lambda^* + h^*(i) = c(x, g^*(i)) + \sum_{j \in I^*} p_{ij}(u) h^*(j) \quad \forall\, i \in I^*$$

Example 5.3:

We consider the dual problem of example 5.1. The matrix formulation of the problem and the corresponding Matlab code are in appendix C. We find the optimal solution:

$$Z^* = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix}$$

where the rows denote the state, and the columns denote the action. Therefore:

$$C^* = \left\{ x \mid \sum_{u \in U(i)} z^*(x,u) > 0 \right\} = \{1,2\}$$

$U^*(1) = \{1\}$; $U^*(2) = \{0\}$; We conclude that: $g^* = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Example 5.4:

We consider the dual problem of example 5.2. Details about the matrix formulation and the implementations of the problem are in appendix C. The code is implemented with Matlab in appendix C. Based on the provided solution (Recall that $X = \{0, \dots, 8\}$ and $U = \{0,1,2,3\}$)

$C^* = \{x \mid \sum_{u \in U(i)} z^*(x,u) > 0\} = \{0,1,2,\dots,8\} = X$ (i.e. in this particular case $C^* = X$)

$U^*(x) = \{u \in U(x) \mid z^*(x,u) > 0\}$

$U^*(0) = \{0\}$; $U^*(1) = \{3\}$; $U^*(2) = \{4\}$; $U^*(3) = \{4\}$; $U^*(4) = \{4\}$; $U^*(5) = \{4\}$; $U^*(6) = \{4\}$; $U^*(7) = \{4\}$; $U^*(8) = \{3\}$.

---

6) Value iteration

Similar to the case of discounted cost problems, we develop computational methods to solve the optimality equations using the value iteration algorithm.

a) Preliminaries:

As $T^k h$ represents the undiscounted k-stage cost, we expect the optimal average cost J^* to be the limit of $\frac{1}{k} T^k h$. We will show that this result is true regardless of the structure of the chain (unichain, multichain, etc). The idea of the proof is to find an upper and lower bounds to $T^k h$ for any h.

Proposition 10:

If J^* is the optimal average cost, and $\hat{h}$ a vector satisfying $J^* + \hat{h} = B\hat{h}$. Then for any other vector h:

$$\underline{\alpha} + kJ^*(x) + \hat{h}(x) \leq \left(B^k h\right)(x) \leq \bar{\alpha} + kJ^*(x) + \hat{h}(x)$$

With $\bar{\alpha} = \max_x \{h(x) - \bar{h}(x)\}$ and $\underline{\alpha} = \min_x \{h(x) - \bar{h}(x)\}$

Proof: The proof is provided in appendix A.

Theorem 2:

$$J^* = \lim_{k \to \infty} \frac{1}{k} B^k h$$

Proof:
The theorem is an immediate corollary of the previous proposition by dividing by k and take the limit to ∞.

Trying to find J^* using the limit of $\frac{1}{k} B^k h$,has two problems. The first problems is related to the convergence because it is not guaranteed to have $B^k h$ converging. The second problem is that the use of such method does not allow us to know the differential cost vector.

We focus on unichain models. An interesting result in this case is that the width of the function $T^k h(x)$ does not increase with k in the case of unichain models. In other words the maximum and minimum of $T^k h(x)$ is independent of k. From this observation comes the idea of relative value iteration which consists in subtracting a constant term from $T^k h(x)$.

Proposition 11:
We consider a unichain MDP. Let J^* be the optimal average cost, and $\hat{h}$ a vector satisfying $J^* + \hat{h} = B\hat{h}$. Then for any other vector h:

$$\max_x (B^k h)(x) - \min_x \left(B^k h\right)(x) \leq \max_x \hat{h}(x) - \min_x \hat{h}(x)$$

Proof: The proof is provided in appendix A.

In the case of average cost MDPs, the iterates were defined as: $v_k = B^k v$. By analogy, we define the iterates of a unichain MDP in average cost problems as:

$$h_{k+1} = Bh_k - Bh_k(t)e$$

Where t is an arbitrary state, and e is the unit vector. Furthermore, the stopping criteria (by analogy with the discounted cost) is for some $\epsilon > 0$:

$$sp(h_{k+1} - h_k) < \epsilon$$

Where sp denotes the span semi-norm of a vector: $p(h) = \max_x\{h(x)\} - \min_x\{h(x)\}$. This algorithm is known as the relative value iteration.

It is noted that by subtracting a constant term from h_k at each iteration, we guarantee that the iterates remain bounded, and the upper and lower bounds are independent of k. However, this result is still insufficient to guarantee the convergence of the value iteration. To clarify this point, we consider the following example:

Example 6.1:

We consider: $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$; $c = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$; $h0 = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$; and t =1.

In this particular case where $c = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, the iterates are:

$$h_k = Bh_k - Bh_k(t)e = B^k h0 - B^k h_0(t)e = P^k h0 - P^k h_0(t)e$$

It can be shown by induction that: $P^k h_0 = \begin{cases} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} & if\ k\ is\ even \\ \begin{bmatrix} \beta \\ \alpha \end{bmatrix} & if\ k\ is\ odd \end{cases}$

Therefore: $h_k = \begin{cases} \begin{bmatrix} 0 \\ \beta - \alpha \end{bmatrix} & if\ k\ is\ even \\ \begin{bmatrix} 0 \\ \alpha - \beta \end{bmatrix} & if\ k\ is\ odd \end{cases}$.

Note: if k is even, k+1 is odd and inversely. We obtain: $sp(h_{k+1} - h_k) = 2\,|\alpha - \beta|$
Therefore, the algorithm will not converge unless $\alpha = \beta$.

A stronger hypothesis is therefore needed to ensure convergence in the case of unichain models. This hypothesis is the aperiodicity of the transition matrix. In fact, in the previous example, the Markov chain is periodic with period 2 which is the reason for non-convergence.

To conclude, the relative value iteration convergence in the case of unichain models with aperiodic transition matrix. The algorithm of the relative value iteration is as follows

Algorithm: Relative value iteration

1) **Start with a guess h_0, and pick a random state t. Define $\epsilon \approx 0$ $(\epsilon > 0)$**
2) **Iteratively compute $h_k = Bh_{k-1} - Bh_{k-1}(t)e$**
3) **If $sp(h_k - h_{k-1}) = \max_x\{h_k(x) - h_{k-1}(x)\} - \min_x\{h_k(x) - h_{k-1}(x)\} < \epsilon$.**
   Stop
   Otherwise, repeat steps 2) and 3)

It is noted that when the algorithm converges, the last iterate h^* verifies:

$h^* = Bh^* - Bh^*(t)e$ Which is equivalent to $h^* + Bh^*(t)e = Bh^*$. By the optimality equations derived in the case of equal average cost (applies to unichain MDPs): h^* is the relative cost vector (bias) and $Bh^*(t)$ is the optimal average cost per step.

Example 6.2:

We consider the MDP of example 5.1, and we aim to solve it using the relative value iteration algorithm. The relative value iteration algorithm is implemented with Matlab (Appendix D). We choose state 1 as a reference state (i.e. t=1). The number of iterations needed for convergence is 17. The optimal control action is:

$$u^* = \begin{bmatrix} u_2 \\ u_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The optimal average cost is $J^* = Bh^*(t) = 0.75$, and its associated bias is: $h^* = \begin{bmatrix} 0 \\ \frac{1}{3} \end{bmatrix}$.

Example 6.3:

In this example, we consider the infinite horizon version of the rate control problem. We solve the average cost as an optimality criterion. The relative value iteration algorithm is implemented with Matlab (Appendix D). The number of iterations needed for convergence is 268, and the optimal average cost is -5.8841. The optimal control and the bias associated with the average cost are shown in Figure 3 and Figure 4, respectively.

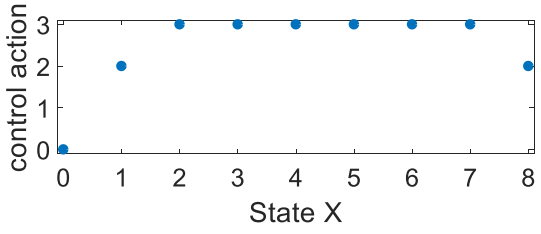


Figure 3: Optimal control action using relative value iteration

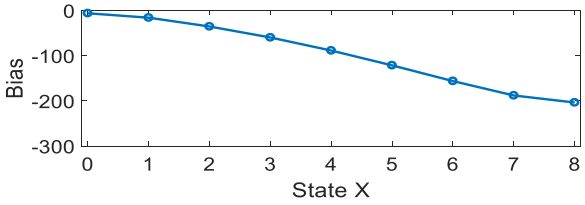


Figure 4: Bias obtained using relative value iteration

Note: If we consider the case of discounted cost problems with setting $\beta = 0.999$, we notice that it takes 482 iterations for the algorithm to converge. The optimal action obtained with the case of discounted cost problem with β close to 1 is the same for states from 0 to 7. The difference is in the control action of state 8.

To conclude, as discussed previously, the unichain assumption is not sufficient to guarantee the convergence, and it requires the aperiodicity of the chain.
We can circumvent this problem by modifying the transition matrix in a way that eliminates any periodicity. The previously described relative value iteration algorithm is then applied using the modified matrix to obtain the results. Such transformation guarantees the convergence of the algorithm even when using periodic transition matrices. The transformation consists in considering some scalar $\lambda \in (0,1)$, and the modified transition matrix

$$\tilde{P} = (1 - \lambda)I + \lambda P_{\mu}$$

7) Policy iteration:

The policy iteration can be used for both unichain and multichain MDPs. The focus will be on MDPs that verify the WA condition such as unichain MDPs. The policy iteration algorithm is very similar to the case of discounted cost MDPs. In fact, it comprises mainly a policy evaluation step and a policy improvement step. At each iteration step k, we have a unichain stationary policy g_k.The policy evaluation step consists in finding the optimal average cost and its associated bias by solving the optimality equations (in the case of equal average cost):

$$\lambda_k e + h_k = B_{g_k} h_k$$

However, this system of equations has n equations and n+1 unknowns. Therefore, we choose an arbitrary state t, and we set $h_k(t) = 0$. In this way, we obtain the following new system of equations that admits a unique solution:

$$\lambda_k e + h_k = B_{g_k} h_k$$
$$h_k(t) = 0$$

In this case, λ_k represents the optimal average cost of g_k.

The policy improvement step consists in finding a new policy g_{k+1} such that:

$$B_{g_{k+1}} h_k = B h_k$$

The algorithm stops when $g_{k+1} = g_k$.

To conclude, the steps of the policy iteration algorithm are as follows:

Policy iteration algorithm:

1) **Start with an initial policy g_0 and a random state t**
2) **At each iteration k, find λ_k and h_k by solving:**
$$\boldsymbol{\lambda_k e + h_k = B_{g_k} h_k}$$
$$\boldsymbol{h_k(t) = 0}$$
3) **Find a new policy g_{k+1} satisfying:**
$$\boldsymbol{B_{g_{k+1}} h_k = B h_k}$$
4) **If $g_{k+1} = g_k$ Stop.**
Otherwise, repeat steps 2) and 3)

Example 7.1:

We consider the average cost MDP problem of example 5.1, and we solve it using the policy iteration algorithm. The code is implemented with Matlab in appendix E. We find exactly the

same optimal policy and optimal average cost as in the case where the value iteration algorithm was used. The difference is in the number of iterations that is equal to two in the case of policy iteration algorithm.

Example 7.2:

We consider the average cost MDP problem of example 5.2, and we solve it using the policy iteration algorithm. The code is implemented with Matlab in appendix E. The number of iterations needed for convergence is 5 (compared with 268 iterations when the relative value iteration is used), and the optimal average cost is -5.8841, which is the same as in the case of value iteration. The optimal control and the bias associated with the average cost are the same as those in Figure 3 and Figure 4, respectively.

8) Discussion and conclusion

In this work, the average cost MDP problem is presented under the assumption of finite state and control spaces. First, the relation between the optimal average cost and discounted cost is introduced. It has been shown that the average cost can be obtained by a series expansion of the discounted cost. Furthermore, it has been shown that the optimal solution of the average cost problem satisfies a set of optimality equations which are sufficient conditions for optimality. That is, if a particular policy satisfies these equations then it is optimal. It has been found that these equations have at least one solution through considering a particular type of policies that is the BOP.

However, solving these equations is relatively difficult compared with the discounted cost problems as it consists of a pair of coupled optimality equations. To this aim, we consider the WA condition, and if an MDP verifies this condition the optimality equations are reduced to a single optimality equation. The WA condition is equivalent to the equal average cost for all initial states hypothesis. Then, we checked if some particular classes of MDPs always verify the WA condition.

Finally, we focus on the implementation of computational methods to solve the average cost MDP problem under the WA condition. First, we start by searching for a linear-program based solution. The primal linear program allows to find only the optimal average cost and bias. The implementation of the dual of the problem is needed to find the optimal policy. Second, a value iteration algorithm that consists in generating iterates successively using the Bellman operator can be developed. As we notice that the iterates remain bounded with the upper and lower bounds being independent of the current iteration, a relative value iteration algorithm can be developed. In case the MDP induces aperiodic Markov chains, the value iteration can be used directly. However, in case the MDP induces periodic Markov chains, a transformation must be applied to the transition matrix to guarantee the convergence of the algorithm. When we compare the relative value iteration algorithm of the average cost problem with the value iteration of the discounted cost for a discount factor extremely close to 1, we found that the average cost problem takes less iteration to converge. Finally, the policy iteration algorithm is developed. The algorithm consists of a policy evaluation step followed by a policy improvement step. It was found that it takes less iterations to converge compared with the relative value iteration algorithm. These computational methods can always be improved to yield better results in terms of convergence. In fact, as an example, there has been the Gauss-Seidel implementation of the value iteration which improved the algorithm performance. Therefore, more robust versions of these computational methods can be developed in order to improve the results.

References

[1] PUTERMAN, M.L. 2014. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons. DOI: 10.1002/9780470316887.

[2] BERTSEKAS, D.P. 2011. *Dynamic programming and optimal control Volume II*. Athena Scientific. Available at: http://www.athenasc.com/dpbook.html.

[3] Feinberg, E.A., P.O. Kasyanov, N.V. Zadioanchuk. 2012. Average cost Markov decision processes with weakly continuous transition probabilities. Math. Oper. Res. 37(4) 591–607.

[4] C. Derman, "On sequential decisions and Markov chains", Manage. Sci., vol. 9, pp. 16-24, 1962.

[5] D. Blackwell, "Discrete dynamic programming", Ann. Math. Statist., vol. 33, pp. 719–726, 1962.

[6] Platzman, L., 1977. Finite Memory Estimation and Control of Finite Probabilistic systems, Ph.D. Thesis, Dept. of EECS, MIT, Cambridge, MA.

[7] De Ghellinck, G. T. "Les problemes de Decisions Sequentielles," Cah. Centre d'Etudes Rec. Oper., Vol 2, pp.161-179, 1960.

[8] Manne A., 1960. "Linear Programming and Sequential Decisions, " Man. Science, Vol. 6, pp.256-267.

[9] White, D. J., 1963. "Dynamic programming, Markov Chains, and the method of sucessive approximations, " J. Math. Anal and Appl., Vol. 6, pp. 373-376.

[10] Odoni, A. R., 1969. "On finding the Maximal Gain for Markov Decision Processes," Operations Research, Vol. 17, pp. 857-860.

[11] Howard, R., 1960. Dynamic Programming and Markov Processes, MIT Press, Cambridge, MA.

[12] "Classification of States", *Probabilitycourse.com*, 2020. [Online]. Available: https://www.probabilitycourse.com/chapter11/11_2_4_classification_of_states.php. [Accessed: 30-Apr- 2020].

[13] J. N. Tsitsiklis, "NP-Hardness of Checking the Unichain Condition in Average Cost MDPs," Oper. Res. Lett., vol. 35, no. 3, pp. 319–323, May 2007.

Appendix A

1) Proof of proposition 1:

Proposition 1:

The following properties holds in the case of the Laurent series expansion:

$$P^* = PP^* = P^*P = P^*P^*$$
$$P^*H = HP^* = 0$$
$$P^* + H = I + PH$$

Proof:

In general:

$$(I - \alpha P)\,(I - \alpha P)^{-1} = I$$

By development and multiplication by $(1 - \alpha)$, we obtain:

$$(1-\alpha)(I-\alpha P)^{-1} - \alpha(1-\alpha)P(I-\alpha P)^{-1} = (1-\alpha)I$$

We take limit $\alpha \to 1$: $P^* - PP^* = 0$
Therefore: $P^* = PP^*$ (a)

Similarly, we develop $(I-\alpha P)^{-1}(I-\alpha P) = I$ and we multiply it by $(1-\alpha)$ to obtain:

$$(1-\alpha)(I-\alpha P)^{-1} - \alpha(1-\alpha)(I-\alpha P)^{-1}P = 0$$

We take limit $\alpha \to 1$: $P^* - P^*P = 0$
Therefore: $P^* = P^*P$ (b)

$(I-\alpha P)P^* = P^* - \alpha PP^* =^{(a)} (1-\alpha)P^*$

We multiply the expression above by $(I-\alpha P)^{-1}$ to obtain: $P^* = (1-\alpha)(I-\alpha P)^{-1}P^*$
We take limit $\alpha \to 1$: $P^* = P^*P^*$ (c)

We multiply H by $(I - P + P^*)$ to obtain:

$$(I - P + P^*)H = I - P^*$$

We further develop the expression above to obtain:

$$(H - PH + P^*H) = I - P^* \qquad (*)$$

We multiply the equation above by P^* to obtain: $(P^*H - P^*PH + P^*P^*H) = P^* - P^*P^*$
Using (b) and (c), the expression above is equivalent to: $(P^*H - P^*H + P^*H) = P^* - P^* = 0$
Therefore: $P^*H = 0$
Following the same procedure, it can be shown that $HP^* = 0$.

From (*), we have : $I + PH = H + P^*H + P^*$
Or we have already shown that $P^*H = 0$, therefore: $I + PH = H + P^*$

---

2) Proof of proposition 5:

Proposition 5:

All BOPs have the same gain J^* and bias h^*

Let (J^*, h^*) be the gain and bias of a BOP. We have:

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\, J^*(j) \quad \forall\, i = 1, \dots, n$$

If $\bar{U}(i)$ is the set of controls that reaches the minimum in the equation above, then:

$$J^*(i) + h^*(i) = \min_{u \in \bar{U}(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h^*(j) \right\} \quad \forall\, i = 1, \dots, n$$

If g^* is BOP, it reaches the minimum of the RHS of these two equations.

Proof:

Let (J_1^*, h_1^*) and (J_2^*, h_2^*) be the gain-bias of g_1 and g_2, respectively, with g_1 and g_2 two BOPs. We need to show that $J_1^* = J_2^*$ and $h_1^* = h_2^*$. In fact, from the definition of a BOP, $J_{\beta,1} = J_{\beta,2}$ for β close to 1. Or:

$J_{\beta,g_1} = (1-\beta)^{-1} J_1^* + h_1^* + O(|1-\beta|)$ and $J_{\beta,g_2} = (1-\beta)^{-1} J_2^* + h_2^* + O(|1-\beta|)$
Then, if we take the limit β to 1 we obtain: $J_1^* = J_2^*$.

Consequently, $h_1^* = h_2^* + O(|1-\beta|)$. Therefore, as β goes to 1, we obtain $h_1^* = h_2^*$

2) Let g^* be a BOP, with (J^*, h^*) its gain-bias pair. Let g be any other stationary policy. Let β be in $[\bar{\beta}, 1]$ with $\bar{\beta} \in (0,1)$. From the definition of a BOP (and properties of Bellman operator), we have that:

Therefore: $BJ_{\beta,g^*} \leq B_g J_{\beta,g^*}$ and consequently $c_{g^*} + (\beta P_{g^*}) J_{\beta,g^*} \leq c_g + (\beta P_g) J_{\beta,g^*}$ (1)

Or, from the Laurent series expansion:

$J_{\beta,g^*} = (1-\beta)^{-1} J^* + h^* + O(|1-\beta|)$ (2)

Then, from (1) and (2) , we conclude that:

$$c_{g^*} + (\beta P_{g^*})\left((1-\beta)^{-1} J^* + h^* + O(|1-\beta|) \leq c_g + (\beta P_g)\right)\left((1-\beta)^{-1} J^* + h^* + O(|1-\beta|)\right)$$

Which is equivalent to:

$$0 \leq c_g - c_{g^*} + (\beta)(P_g - P_{g^*})\left((1-\beta)^{-1} J^* + h^* + O(|1-\beta|)\right) \quad (3)$$

We multiply the equation above by $(1-\beta)$ to obtain:

$$0 \leq (1-\beta)(c_g - c_{g^*}) + (\beta)(P_g - P_{g^*})\left(J^* + (1-\beta)h^* + O(|1-\beta|^2)\right)$$

We take the limit when β goes to 1, to obtain: $(P_g - P_{g^*}) J^* \geq 0$.

Therefore: $P_g J^* \geq P_{g^*} J^*$

Or: $P_{g^*} J^* = J^*$. We conclude that: $P_g J^* \geq J^*$

To resume, we have shown that: for any policy g: $J^* \leq P_g J^*$. Therefore:

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) J^*(j) \quad \forall\, i = 1, \dots, n$$

Now, we prove the second part 2) :

Let g be a policy such that: $P_g J^* = P_{g^*} J^*$. Then $P_g J^* = J^*$.

Therefore, from (3) we obtain:

$$0 \leq c_g - c_{g^*} + (\beta)(P_g - P_{g^*})\,(h^* + O(|1-\beta|)$$

We take the limit when β goes to 1, to obtain:

$$0 \leq c_g - c_{g^*} + (P_g - P_{g^*})h^*$$

Which is equivalent to:

$$c_g + \beta P_g h^* \geq c_{g^*} + \beta P_{g^*} h^*$$

Therefore: g^* minimizes $c_g + \beta P_g h^*$ over all policies satisfying $P_g J^* = P_{g^*} J^*$.

Or from proposition 3: $J^* + h^* = c_{g^*} + P_{g^*} h^*$

Therefore: $J^* + h^* \leq c_g + P_g h^*$ for any policy g satisfying $P_g J^* = P_{g^*} J^*$.

Which is equivalent to:

$$J^*(i) + h^*(i) = \min_{u \in \bar{U}(i)} \left\{ c(i,u) + \sum_{j=1}^{n} p_{ij}(u)\, h^*(j) \right\} \quad \forall\, i = 1, \dots, n$$

where : $\bar{U}(i)$ is the set of controls that reaches the minimum of the first equation.

---

3) Proof of proposition 6:

Proposition 6:
Let g be an admissible policy (stationary or not). We consider two vectors J and h that satisfy the two inequalities:

$$J \leq P_g J \quad (*)$$
$$J + h \leq B_g h \quad (**)$$

Then: $J \leq J_g$. If these inequalities are verified with equalities, then: $J = J_g$

Proof:
For any policy g: $B_g(J+h) = c_g + P_g(J+h) = P_g J + B_g h$

Therefore: This is true in particular for g_k: $B_{g_k}(J+h) = P_{g_k} J + B_{g_k} h \overset{(*)}{\geq} J + B_{g_k} h$ (from (*))
Let N be an integer:
From (**), we have as assumption that: $J + h \leq B_{g_{N-1}} h$
We apply the operator $B_{g_{N-1}}$ on both sides of the inequality above, and by monotonicity of the Bellman operator, we obtain: : $B_{g_{N-2}}(J+h) \leq B_{g_{N-2}} B_{g_{N-1}} h$ (***)
Or $B_{g_{N-2}}(J+h) \geq J + B_{g_{N-2}} h \geq 2J + h$ (****)
By combining (***) and (****), we obtain: $B_{g_{N-2}} B_{g_{N-1}} h \geq 2J + h$
By iterating the process, we obtain:

$$B_0 \dots B_{g_{N-3}} B_{g_{N-2}} B_{g_{N-1}} h \geq NJ + h \quad (eq1)$$

Or $(B_0 \dots B_{g_{N-3}} B_{g_{N-2}} B_{g_{N-1}} h)(i)$ represents the N-stage cost corresponding to state i, which is equal to:

$$\left(B_0 \dots B_{g_{N-3}} B_{g_{N-2}} B_{g_{N-1}} h\right)(i) = E\left\{\sum_{k=0}^{N-1} g(x_k, g_k(x_k)) \mid x_0 = i,\ g\right\} \quad (eq2)$$

By using $(eq1)$ and $(eq2)$:

$$E\left\{h(x_N) + \sum_{k=0}^{N-1} g(x_k, g_k(x_k)) \mid x_0 = i, g\right\} \geq NJ(i) + h(i) \quad for\ each\ i$$

We divide the inequality above by N:

$$\frac{E\{h(x_N)\}}{N} + \frac{E\left\{\sum_{k=0}^{N-1} g\left(x_k, g_k(x_k)\right) \mid x_0 = i, g\right\}}{N} \geq J(i) + \frac{h(i)}{N}$$

Finally, we take the limit to infinity to obtain: $J \leq J_g$.

It is noted that if (*) and (**) hold with equality, then (eq1) holds with equality. Therefore, we find finally that $J = J_g$.

---

4) **Proof of proposition 7:**

Proposition 7:

We suppose that the (OEs) are satisfied by a pair (J^*, h^*). Then: J^* is the optimal average cost vector.
Moreover, if a stationary policy g^* satisfies the RHS of (OEs), then g^* is an optimal policy.

Proof:

Proposition 6 is the basis for demonstrating proposition 7. In fact, we consider any J_g, and we show that $J^* \leq J_g$.
In fact, as J^* verifies the two equations of (OEs): $J^* \leq P_g J^*$ and $J^* + h^* \leq B_g h^*$. Therefore, by proposition 6: $J^* \leq J_g$.
Therefore, J^* is the optimal average cost vector.
If g^* satisfies the RHS of (OEs): $J^* = P_{g^*} J^*$ and $J^* + h^* = B_{g^*} h^*$.
If we apply again the proposition 6 : : $J^* = J_{g^*}$. Therefore, g^* is an optimal policy.

---

5) **Proof of proposition 10:**

Proposition 10:

If J^* is the optimal average cost, and $\hat{h}$ a vector satisfying $J^* + \hat{h} = B\hat{h}$. Then for any other vector h:

$$\underline{\alpha} + kJ^*(x) + \hat{h}(x) \leq \left(B^k h\right)(x) \leq \bar{\alpha} + kJ^*(x) + \hat{h}(x)$$

With $\bar{\alpha} = \max_x \{h(x) - \bar{h}(x)\}$ and $\underline{\alpha} = \min_x \{h(x) - \bar{h}(x)\}$

Proof:

By definition: $B_g h = c_g + P_g h = c_g + P_g h + P_g \hat{h} - P_g \hat{h} = c_g + P_g\left(h - \hat{h}\right) + P_g \hat{h} = B_g \hat{h} + P_g\left(h - \hat{h}\right)$

Similar to the results obtained with the Bellman operator in the case of discounted cost, we iterate the process k times to obtain: $B_g^k h = B_g^k \hat{h} + P_g^k (h - \hat{h})$. Therefore:

$$B_g^k h - B_g^k \hat{h} = P_g^k\left(h - \hat{h}\right) \overset{(a)}{\leq} \max_x \{h(x) - \bar{h}(x)\} e \quad (e\ is\ unit\ vector) \quad (*)$$

Where (a) follows from the stochasticity of the matrix P_g and the definition of the maximum. As $J^* + \hat{h} = B\hat{h} \leq B_g\hat{h}$, we apply the operator B_g on both sides to obtain by monotonicity of B_g:

$$\begin{aligned} B_g^2\hat{h} &\geq B_g(J^* + \hat{h}) = c_g + P_g(J^* + \hat{h}) = B_g\hat{h} + P_g J^* \\ &\geq B_g\hat{h} + J^* \\ &\geq 2J^* + \hat{h} \end{aligned}$$

By iterating the process k times, we find that $B_g^k\hat{h} \geq kJ^* + \hat{h}$. Therefore, $-B_g^k\hat{h} \leq -kJ^* - \hat{h}$ $(**)$ By combining $(*)$ and $(**)$, we obtain the RHS of the proposition inequality. The LHS of the inequality can be obtained following the same procedure.

---

6) **Proof of proposition 11:**

Proposition 11:
We consider a unichain MDP. Let J^* be the optimal average cost, and $\hat{h}$ a vector satisfying $J^* + \hat{h} = B\hat{h}$. Then for any other vector h:

$$\max_x (B^k h)(x) - \min_x (B^k h)(x) \leq \max_x \hat{h}(x) - \min_x \hat{h}(x)$$

Proof:

From proposition 1,

$$\begin{aligned} (B^k h)(i) \leq \max_i (B^k h)(i) &\overset{(a)}{\leq} \max_x \left\{ \max_y \{h(y) - \hat{h}(y)\} + kJ^*(x) + \hat{h}(x) \right\} \\ &\overset{(b)}{\leq} \max_y \{h(y) - \hat{h}(y)\} + kJ^*(x) + \max_x (\hat{h}(x)) \quad (*) \end{aligned}$$

Where (a) and (b) following from proposition 1 and the equal cost assumption, respectively. Similarly:

$$-(B^k h)(i) \leq -\min_x (B^k h)(x) \leq -\max_y \{h(y) - \hat{h}(y)\} - kJ^*(x) - \min_x \hat{h}(x) \quad (**)$$

By adding inequalities (*) and (**), we finally obtain:

$$0 \leq \max_x (B^k h)(x) - \min_x (B^k h)(x) \leq \max_x \hat{h}(x) - \min_x \hat{h}(x)$$

This implies that the width of $B^k h$ does not depend on k.

Appendix B
(Primal Linear Program)

Example 5.1:

In this part, the primal linear program is written in a matrix format that makes the problem easier to solve. Let n be the number of states. We consider the case where only two actions are available; u_1 and u_1 as in example 7.1. We recall that, λ_k is a scalar, h_k is an (n,1) vector;
$P(u1)$ is the (n,n) transition matrix corresponding to u_1. Similarly, $P(u2)$ is the (n,n) transition matrix corresponding to u_2.
$C(u_1)$ is an (n,1) matrix corresponding to action u_1 and $C(u_2)$ is an (n,1) matrix corresponding to action u_2.
The primal linear program for a unichain MDP is as follows:

$$\max \lambda$$

$$s.t. \quad \lambda + h(x) - \sum_{j=1}^{n} p_{xj}(u)h(j) \leq c(x,u) \qquad \forall\, x = 1, \dots, n \; and \; \forall\, u \in U(i)$$

We need to find the (n+1, 1) vector X defined by:

$$X = \begin{bmatrix} \lambda_k \\ h_k \end{bmatrix}$$

We can write the system of equations $(*)$ as:

$$\min f^T X$$

$$s.t. AX - BX \leq C$$

The matrices A, B, C and f are defined by:

$$A = \left[\begin{array}{c|c} \begin{matrix} 1 \\ \vdots \\ 1 \end{matrix} & I_{(n,n)} \\ \hline \begin{matrix} 1 \\ \vdots \\ 1 \end{matrix} & I_{(n,n)} \end{array}\right] \qquad B = \left[\begin{array}{c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & P(u_1) \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & P(u_2) \end{array}\right] \qquad C = \left[\begin{array}{c} C(u_1) \\ \hline C(u_2) \end{array}\right]$$

$$f = \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Code of example 5.1:

```
clear all
close all
clc
%  The solution is the X vector : X=[lambda; h(1); h(2)]
```

```
%define the parameters of the problem
n=2; % number of states
m=2; % number of control actions
% Transition matrix
P1=[3/4 1/4;3/4 1/4];
P2=[1/4 3/4;1/4 3/4];
P=[P1;P2];
Paug=[zeros(n*m,1),P]
% cost
c=[2 0.5;1 3];
C=reshape(c,[4,1]);

M=[ones(n,1),eye(n,n)]
Maug=[M;M]
B=Maug-Paug

f=[-1 0 0];
% Recall: The solution is the x vector : X=[lambda; h(1),
h(2)]
fprintf('The optimal solution is')
x = linprog(f,B,C) % solve the problem

fprintf('Therefore:  ')
fprintf('  lambda=')
x(1)
fprintf('h(1)=')
x(2)
fprintf('h(2)=')
x(3)
```

Example 5.2:

In this part, the primal linear program is written in a matrix format that makes the problem easier to solve. Let n and m be the number of states and the number of control actions, respectively. We recall that, λ_k is a scalar, h_k is an (n,1) vector;
In general, $P(u_k)$ is the (n,n) transition matrix corresponding to u_k. $C(u_k)$ is an (n,1) matrix corresponding to action u_k.

The primal linear program for a unichain MDP is as follows:

$$\max \lambda$$

$$s.t. \quad \lambda + h(x) - \sum_{j=1}^{n} p_{xj}(u)h(j) \leq c(x,u) \qquad \forall\, x = 1, \dots, n \; and \; \forall\, u \in U(i)$$

We need to find the (n+1, 1) vector X defined by:

$$X = \begin{bmatrix} \lambda_k \\ h_k \end{bmatrix}$$

We can write the system of equations for the primal linear program:

$$\min f^T X$$

$$s.t. AX - BX \leq C$$

The matrices A, B, C and f are defined by:

$$A=_{(n\times m,\,n+1)}\begin{bmatrix} 1 & & \\ \vdots & I_{(n,n)} & \\ 1 & & \end{bmatrix}\begin{matrix}1^{st}\ block\\ \\ \end{matrix}\quad\begin{matrix} \\ 2^{nd}\ block\\ \\ \end{matrix}\quad\begin{matrix} \\ \\ m^{th}\ block\end{matrix}$$

$$B=_{(n\times m,\,n+1)}\begin{bmatrix} 0 & & \\ \vdots & P(u_1) & \\ 0 & & \\ 0 & & \\ \vdots & P(u_2) & \\ 0 & & \\ & \vdots & \\ 0 & & \\ \vdots & P(u_m) & \\ 0 & & \end{bmatrix}\begin{matrix}1^{st}\ block\\ \\ \end{matrix}\quad\begin{matrix} \\ 2^{nd}\ block\\ \\ \end{matrix}\quad\begin{matrix} \\ \\ m^{th}\ block\end{matrix}$$

$$C=_{(n\times m,\,1)}\begin{bmatrix} c(u_1) \\ c(u_2) \\ \vdots \\ c(u_m) \end{bmatrix}\begin{matrix}1^{st}\ block\\ 2^{nd}\ block\\ \\ m^{th}\ block\end{matrix}$$

$$X=_{(n+1,\,1)}\begin{bmatrix} \lambda \\ h(1) \\ \vdots \\ h(n) \end{bmatrix}\qquad f=_{(n+1,\,1)}\begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Code of example 5.2:

```
clear all
close all
clc
%define parameters of the problem
n=8;
m=3;
q=0.6;
R=6;
h=1;
p=[0 0.25 0.5 0.8];
c=[0 1 4 12];
%define reward matrix
Rew=zeros(n+1,m+1);
Rew(1,:)=-c;
X=0:n;
```

```
for x=2:n+1
    for u=1:m+1
        Rew(x,u)=R*p(u)-h*X(x)-c(u);
    end
end

% Here we enter the n+1, n+1 matrix P(u) for U=u
for u=1:m+1
    P1=zeros(n+1,n+1);
    for x=1:n+1
        if x==1
            for y=1:n+1
                if X(y)==0 P1(x,y)=1-q;
                elseif X(y)==1 P1(x,y)=q;
                else P1(x,y)=0;
                end
            end
        elseif x==n+1
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=1-(1-q)*p(u);
                else P1(x,y)=0;
                end
            end
        else
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=(1-q)*(1-
p(u))+q*p(u);
                elseif X(y)==X(x)+1 P1(x,y)=q*(1-p(u));
                else P1(x,y)=0;
                end
            end
        end

    end
    Pbar(:,:,u)=P1;
end

P=[];
for i=1:m+1
  P=[P;Pbar(:,:,i)] ;
end
Paug=[zeros((n+1)*(m+1),1),P];

C=reshape(-Rew,[(n+1)*(m+1),1]);
```

```
M=[ones(n+1,1),eye(n+1,n+1)];
Maug=[];
for i=1:m+1
  Maug=[Maug;M] ;
end

B=Maug-Paug;

f=[-1 zeros(1,n+1)];
x = linprog(f,B,C) % solve the problem

fprintf('The optimal average cost is')
x(1)
% extract the bias
h=x(2:end)
%plot the  bias
figure
plot([0: n], h)
xlabel('State X')
ylabel('Bias')
```

Appendix C
(Dual Linear Program)

Example 5.1:

The dual problem is defined as follows:

$$\min \sum_{x=1}^{n} \sum_{u \in U(i)} c(i,u)\, z(x,u)$$

$$s.t. \sum_{u \in U(i)} z(x,u) - \sum_{y=1}^{n} \sum_{u \in U(i)} p_{xy}(u)\, z(y,u) = 0 \qquad \forall\, x = 1, \dots, n$$

$$\sum_{x=1}^{n} \sum_{u \in U(i)} z(x,u) = 1$$

$$z(i,u) \geq 0 \qquad \forall\, u \in U(i), \forall\, i = 1, \dots, n$$

The problem can be written in a matrix format as follows:

$$\min f^T X$$

$$s.t. \quad \begin{cases} A_{eq}.X = b_{eq} \\ lb \leq X \end{cases}$$

As previously mentioned, we consider the case of 2 states and 2 control actions u_1 and u_2. The matrices A_{eq}, b_{eq}, and lb are defined as follows:

$$A_{eq} = \left[\begin{array}{cc|cc} 1 - P_{11}(u_1) & 1 - P_{11}(u_2) & -P_{21}(u_1) & -P_{21}(u_2) \\ -P_{12}(u_1) & -P_{12}(u_2) & 1 - P_{22}(u_1) & 1 - P_{22}(u_2) \\ 1 & 1 & 1 & 1 \end{array}\right] \qquad b_{eq} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$lb = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad X = \begin{bmatrix} z(1,u_1) \\ z(1,u_2) \\ z(2,u_1) \\ z(2,u_2) \end{bmatrix} \qquad f = \begin{bmatrix} c(1,u_1) \\ c(1,u_2) \\ c(2,u_1) \\ c(2,u_2) \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Code of example 5.1:

```
clear all
close all
clc

% Implementation of Example 7.2
```

```
% Note : Here we do not set h(1)=0 , the X vector is
[lambda; h(1), h(2)]

%define parameters of the problem
n=2;
m=2;
% Transition matrix
P1=[3/4 1/4;3/4 1/4];
P2=[1/4 3/4;1/4 3/4];
Pbar(:,:,1)=P1;
Pbar(:,:,2)=P2;

% cost
c=[2 0.5;1 3];

C=[c(1,1);c(1,2);c(2,1);c(2,2)]';

A=ones(n+1,n*m);
A(1,1)=1-P1(1,1);
A(1,2)=1-P2(1,1);
A(1,3)=-P1(2,1);
A(1,4)=-P2(2,1);

A(2,1)=-P1(1,2);
A(2,2)=-P2(1,2);
A(2,3)=1-P1(2,2);
A(2,4)=1-P2(2,2);

A(3,1)=1;
A(3,2)=1;
A(3,3)=1;

B=[0;0;1];
f=C;
lb=zeros(4,1);
ub=[];
[x,fval,exitflag,output,lambda] =
linprog(f,[],[],A,B,lb,ub);
x
```

Example 5.2:

In this part, the primal linear program is written in a matrix format that makes the problem easier to solve. Let n and m be the number of states and the number of control actions, respectively. We recall that, λ_k is a scalar, h_k is an (n,1) vector;
In general, $P(u_k)$ is the (n,n) transition matrix corresponding to u_k. $C(u_k)$ is an (n,1) matrix corresponding to action u_k.

The dual problem is defined as follows:

$$\min \sum_{x=1}^{n} \sum_{u \in U(i)} c(i,u)\, z(x,u)$$

$$s.t. \sum_{u \in U(i)} z(x,u) - \sum_{y=1}^{n} \sum_{u \in U(i)} p_{xy}(u)\, z(y,u) = 0 \qquad \forall\, x = 1, \dots, n$$

$$\sum_{x=1}^{n} \sum_{u \in U(i)} z(x,u) = 1$$

$$z(i,u) \geq 0 \qquad \forall\, u \in U(i), \forall\, i = 1, \dots, n$$

The problem can be written in a matrix format as follows:

$$\min f^T X$$

$$s.t. \quad \begin{cases} A_{eq}.X = b_{eq} \\ lb \leq X \end{cases}$$

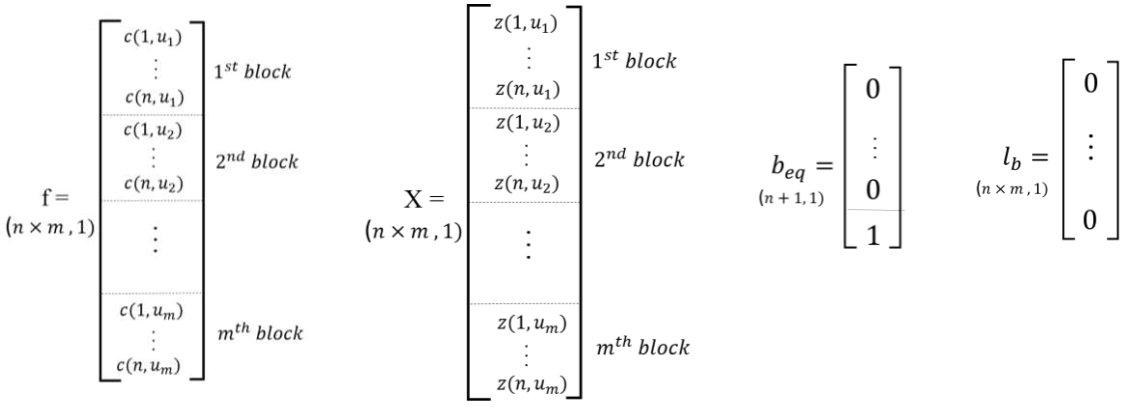
We consider the general case of n states and m control actions.
The matrices A, b, A_{eq}, b_{eq}, and lb are defined as follows:

$$A_{eq} = A - B$$

$$A_{(n+1,\ n \times m)} = \begin{bmatrix} \overbrace{I_{(n,n)}}^{1^{st}\ block} & \overbrace{I_{(n,n)}}^{2^{nd}\ block} & \cdots & \overbrace{I_{(n,n)}}^{m^{th}\ block} \\ 1 \cdots 1 & 1 \cdots 1 & \cdots & 1 \cdots 1 \end{bmatrix}$$

$$B_{(n+1,\ n \times m)} = \begin{bmatrix} \overbrace{P(u_1)(:,1)^T}^{1^{st}\ block} & \overbrace{P(u_2)(:,1)^T}^{2^{nd}\ block} & \cdots & \overbrace{P(u_k)(:,1)^T}^{k^{th}\ block} & \cdots & \overbrace{P(u_m)(:,1)^T}^{m^{th}\ block} \\ P(u_1)(:,2)^T & P(u_2)(:,2)^T & & P(u_k)(:,2)^T & \cdots & P(u_m)(:,2)^T \\ \vdots & \vdots & & \vdots & & \vdots \\ P(u_1)(:,j)^T & P(u_2)(:,j)^T & & P(u_k)(:,j)^T & \cdots & P(u_m)(:,j)^T \\ \vdots & \vdots & & \vdots & & \vdots \\ P(u_1)(:,n)^T & P(u_2)(:,n)^T & & P(u_k)(:,n)^T & \cdots & P(u_m)(:,n)^T \\ 1 \cdots 1 & 1 \cdots 1 & \cdots & 1 \cdots 1 & \cdots & 1 \cdots 1 \end{bmatrix}$$

Transpose of the j^{th} column of $P(u_k)$



Code of example 5.2:

```
clear all
close all
clc
%define parameters of the problem
n=8;
m=3;
q=0.6;
R=6;
h=1;
p=[0 0.25 0.5 0.8];
c=[0 1 4 12];
%define reward matrix
Rew=zeros(n+1,m+1);
Rew(1,:)=-c;
X=0:n;

for x=2:n+1
    for u=1:m+1
        Rew(x,u)=R*p(u)-h*X(x)-c(u);
    end
end

% Here we enter the n+1, n+1 matrix P(u) for U=u
for u=1:m+1
    P1=zeros(n+1,n+1);
    for x=1:n+1
        if x==1
            for y=1:n+1
                if X(y)==0 P1(x,y)=1-q;
                elseif X(y)==1 P1(x,y)=q;
                else P1(x,y)=0;
                end
            end
        elseif x==n+1
```

```
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=1-(1-q)*p(u);
                else P1(x,y)=0;
                end
            end
        else
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=(1-q)*(1-
p(u))+q*p(u);
                elseif X(y)==X(x)+1 P1(x,y)=q*(1-p(u));
                else P1(x,y)=0;
                end
            end
        end

    end
    Pbar(:,:,u)=P1;
end


C=reshape(-Rew,[(n+1)*(m+1),1]);

f=C;
M=ones(1,(n+1)*(m+1));

A=zeros(n+1,(n+1)*(m+1));
    A=[];
  for j=1:m+1
      A=[A,eye(n+1,n+1)]
  end
% **********

for j=1:m+1
   for i=1:n+1
    B(i,(j-1)*(n+1)+1:j*(n+1))= Pbar(:,i,j)'
   end
end

D=zeros(n+1,1);


f=C;
lb=zeros((n+1)*(m+1),1);
ub=[];
E=ones(1,(n+1)*(m+1));
Aeq=[A-B;E];
```

```
beq=[D;1];
[x,fval,exitflag,output,lambda] =
linprog(f,[],[],Aeq,beq,lb,ub);
x
```

Appendix D
(Value iteration)

Code of example 6.2:

```
clear all
close all
clc

% Implementation of Example 5.2

%Define the parameters of the problem
n=2; % number of states
% Transition matrix
P1=[3/4 1/4;3/4 1/4]; % P(u1)
P2=[1/4 3/4;1/4 3/4]; % P(u2)
Pbar(:,:,1)=P1;
Pbar(:,:,2)=P2;

% Cost
c=[2 0.5;1 3];

t=1; % Choose a random state

k=0; % Initialize iterations
h=zeros(n,1); % initilize h as a zero vector
eps=10000; % start with an arbitrary large epsilon just to
enter the ''while loop''
while eps>1e-5
for u=1:2
        PV(:,u)=Pbar(:,:,u)*h;
    end
    B=c+PV;% apply the operator B to h

    for x=1:n
        %For each row of B we search the minimum and its
index
        [M,I]=min(B(x,:));
        hnew(x,1)=M;
        uopt(x,1)=I-1;
    end
    % Find the upper and lower bounds
    lw=min(hnew-h);
    up=max(hnew-h);
    eps=up-lw; % calculate the span semi-norm sp(h_(k+1)-
h_k )
    % update the error as the difference betwen upper and
lower bounds
    h=hnew-hnew(t).*ones(n,1);
    k=k+1; % update the number of iterations
end
```

```
% Relative Value iteration results
fprintf('The number of iterations of the relative value
iteration is :')
k
fprintf('The optimal average cost is:')
hnew(t)
fprintf('The associated bias is:')
h
fprintf('The optimal control action is:')
uopt
```

Code of example 6.3:

```
clear all
close all
clc
tic
%define parameters of the problem
n=8;
m=3;
q=0.6;
R=6;
h=1;
p=[0 0.25 0.5 0.8];
c=[0 1 4 12];
%define reward matrix
Rew=zeros(n+1,m+1);
Rew(1,:)=-c;
X=0:n;

for x=2:n+1
    for u=1:m+1
        Rew(x,u)=R*p(u)-h*X(x)-c(u);
    end
end

% Here we enter the n+1, n+1 matrix P(u) for U=u
for u=1:m+1
    P1=zeros(n+1,n+1);
    for x=1:n+1
        if x==1
            for y=1:n+1
                if X(y)==0 P1(x,y)=1-q;
                elseif X(y)==1 P1(x,y)=q;
                else P1(x,y)=0;
                end
```

```
            end
        elseif x==n+1
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=1-(1-q)*p(u);
                else P1(x,y)=0;
                end
            end
        else
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=(1-q)*(1-
p(u))+q*p(u);
                elseif X(y)==X(x)+1 P1(x,y)=q*(1-p(u));
                else P1(x,y)=0;
                end
            end
        end

    end
    Pbar(:,:,u)=P1;
end

% Value iteration algorithm
k=0;
V=zeros(n+1,1); % Value function is initilized as a zero
vector
eps=10000; % start with an arbitrary large epsilon just to
enter the ''while loop''
t=1;

while eps>1e-5
    for u=1:m+1
        PV(:,u)=Pbar(:,:,u)*V;
    end
    B=Rew+PV;% apply the bellman operator to V

    for x=1:n+1
        %For each row of B we search the maximum and its
index
        [M,I]=max(B(x,:));
        hnew(x,1)=M;
        uopt(x,1)=I-1;
    end

    % Find the upper and lower bounds
    lw=min(hnew-V);
```

```
    up=max(hnew-V);
    eps=up-lw; % update the error as the difference betwen
upper and lower bounds
    V=hnew-hnew(t).*ones(n+1,1);
    k=k+1; % update the number of iterations
end
toc
fprintf('nbre of iterations of the relative value
iteration is :')
k
fprintf('The optimal average cost is :')
hnew(t)

%plot the optimal policy
figure
plot([0: n], uopt,'-o')
xlabel('X')
ylabel('optimal control action')

%plot the bias
figure
plot([0: n], hnew)
xlabel('X')
ylabel('bias')
title('value iteration algorithm: bias')
```

Appendix E
(Policy iteration)

Code of example 7.1:

```
clear all
close all
clc

% Implementation of Example 6.1
%define parameters of the problem
n=2;
m=2;
% Transition matrix
P1=[3/4 1/4;3/4 1/4];
P2=[1/4 3/4;1/4 3/4];
Pbar(:,:,1)=P1;
Pbar(:,:,2)=P2;

% cost
c=[2 0.5;1 3];

t=1; % pick a random state
k=0; % Initialize iteration
h=zeros(n,1); % Value function is initilized as a zero
vector
uopt=[0;0]; % start with an arbitrary policy of zeros
control actions
uopt_old=[1;1];
h=zeros(n,1);
hnext=ones(n,1);
k=0;
v=sum(uopt~=uopt_old);
while v
   % We construct the A, B and C matrices
    % Construct the A matrix
A =zeros(n+1,n+1);
A(1:end-1,1)=ones(n,1);
A(end,2)=1;
A(1:end-1,2:end)=eye(n,n);
    % Construct the C matrix
        % First we construct the Cg column vector
           Cg=zeros(n,1);
            for i=1:n
            Cg(i,1)=c(i,uopt(i)+1);
            end

        C=zeros(n+1,1);
        C(1:end-1)=Cg;
    % Construct the B matrix
        % First we construct the Pg matrix
```

```
            Pg=zeros(n,n);
             for i=1:n
             for j=1:n
             Pg(i,j)= Pbar(i,j,uopt(i)+1);
             end
             end
     B=zeros(n+1,n+1);
     B(1:end-1,2:end)=Pg;

sol=inv(A-B)*C; % The sol vector is [lambda_k;h_k]

h=sol(2:end,1); % extract the h_k vector from sol
 for u=1:m
        PV(:,u)=Pbar(:,:,u)*h;
    end
    Be=c+PV;% apply the bellman operator to V
    uopt_old=uopt; % store the old uopt
    for x=1:n
        %For each row of B we search the maximum and its
index
        [M,I]=min(Be(x,:));
        hnext(x,1)=M;
        uopt(x,1)=I-1;
    end

 k=k+1  ;  %update the number of iterations
v=sum(uopt~=uopt_old); % If v=0 then uopt~=uopt_old and
the algorithm stops, otherwise the algorithm performs an
additional iteration
end

fprintf('The number of iterations is:')
k
fprintf('The optimal policy is ')
uopt
fprintf('The optimal average cost is ')
sol(1)
```

Code of example 7.2:

```
clear all
close all
clc
%define parameters of the problem
n=8;
m=3;
q=0.6;
R=6;
h=1;
p=[0 0.25 0.5 0.8];
c=[0 1 4 12];
beta=0.9;
%define reward matrix
Rew=zeros(n+1,m+1);
Rew(1,:)=-c;
X=0:n;

for x=2:n+1
    for u=1:m+1
        Rew(x,u)=R*p(u)-h*X(x)-c(u);
    end
end

% Here we enter the n+1, n+1 matrix P(u) for U=u
for u=1:m+1
    P1=zeros(n+1,n+1);
    for x=1:n+1
        if x==1
            for y=1:n+1
                if X(y)==0 P1(x,y)=1-q;
                elseif X(y)==1 P1(x,y)=q;
                else P1(x,y)=0;
                end
            end
        elseif x==n+1
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=1-(1-q)*p(u);
                else P1(x,y)=0;
                end
            end
        else
            for y=1:n+1
                if X(y)==X(x)-1 P1(x,y)=(1-q)*p(u);
                elseif X(y)==X(x) P1(x,y)=(1-q)*(1-
p(u))+q*p(u);
```

```
                elseif X(y)==X(x)+1 P1(x,y)=q*(1-p(u));
                else P1(x,y)=0;
                end
            end
        end

    end
    Pbar(:,:,u)=P1;
end

%Policy iteration algorithm

t=1; % pick a random state
k=0; % Initialize iteration
h=zeros(n,1); % Value function is initilized as a zero
vector
uopt=zeros(n+1,1); % start with an arbitrary policy of
zeros control actions
uopt_old=ones(n+1,1);
h=zeros(n+1,1);
hnext=ones(n+1,1);
k=0;
v=sum(uopt~=uopt_old);
reward=zeros(n+1);

while v
    % We construct the A, B and C matrices
    % Construct the A matrix
    A =zeros(n+2,n+2);
    A(1:end-1,1)=ones(n+1,1);
    A(end,2)=1;
    A(1:end-1,2:end)=eye(n+1,n+1);

    % Construct the C matrix
        % First we construct the Cg column vector
         Cg=zeros(n+1,1);
         for i=1:n+1
         Cg(i,1)=Rew(i,uopt(i)+1);
         end
         C=zeros(n+2,1);
         C(1:end-1)=Cg;
     % Construct the B matrix
         % First we construct the Pg matrix
            Pg=zeros(n+1,n+1);
              for i=1:n+1
                for j=1:n+1
               Pg(i,j)= Pbar(i,j,uopt(i)+1);
```

```
                end
            end

        B=zeros(n+2,n+2);
        B(1:end-1,2:end)=Pg;

sol=inv(A-B)*C; % The sol vector is [lambda_k;h_k]
h=sol(2:end,1); % extract the h_k vector from sol

        % Now we calculate Vnext=BV and its optimal policy
        for u=1:m+1
        PV(:,u)=Pbar(:,:,u)*h;
        end
    Be=Rew+PV; % apply Bellman operator to V
    uopt_old=uopt; % store the old uopt
    for x=1:n+1
        [M,I]=max(Be(x,:));
        hnext(x,1)=M;
        uopt(x,1)=I-1;
    end
k=k+1 ; %update the number of iterations
v=sum(uopt~=uopt_old); % If v=0 then uopt~=uopt_old and
the algorithm stops, otherwise the algorithm performs an
additional iteration
end

fprintf('nbre of iterations of the policy iteration is :')
k
fprintf('the optimal average cost is')
sol(1)
%plot the optimal control policy
figure
plot([0: n], uopt,'*')
xlabel('X')
ylabel('optimal control action')

%plot the  bias
figure
plot([0: n], hnext)
xlabel('X')
ylabel('Bias')
```