CONTINUE, QUIT, RESTART PROBABILITY MODEL AND RELATED PROBLEMS

by

Constantine Steinberg

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Applied Mathematics in
Mathematics and Statistics

Charlotte, North Carolina

2011

Approved by:

____

Dr. Isaac Sonin

____

Dr. Stanislav Molchanov

____

Dr. Joseph Quinn

____

Dr. Yu Wang

ABSTRACT

CONSTANTINE STEINBERG. Continue, Quit, Restart Probability Model and Related Problems. (Under the direction of DR. ISAAC SONIN)

We discuss a new class of applied probability models. There is a system whose evolution is described by a Markov chain with known transition matrix in a discrete state space. At each moment of a discrete time a decision maker can apply one of three possible actions: to continue, to quit, or to restart Markov chain to the "restarting point." Where restarting point is a fixed state of the Markov chain. The decision maker is earning a reward (fee), which is the function of the state and chosen action. The goal for the decision maker is to maximize expected total discounted reward on an infinite time horizon.

Such model is a generalization of a model of Katehakis and Veinott, Katehakis and Veinott [1987], where restart to a unique point is allowed without any fee, and quit action is absent. Both models are related to Gittins index and another index defined in a Whittle family of stopping retirement problems. We propose a transparent recursive finite algorithm to find an optimal strategy in $O(n^3)$ operations.

Contents

CHAPTER 1: INTRODUCTION

Our main goal is to study the new applied probability model and develop a recursive algorithm for its solution. This model is a special case of a general Markov Decision Process (MDP) model, while it is essentially more general than the Optimal Stopping (OS) model. The main definition from MDP are given in Chapter 1.1, for further details we direct the reader to, e.g., Feinberg and Shwartz [2002] or Puterman [2005].

The general finite MDP model is defined by a tuple $M = (X, A(x), P, r(x|a), \beta)$, where X is a finite state space, $A(x)$ is a finite set of actions available at point $x \in X$, $P = \{p(x, y|a)\}$ is a stochastic matrix, describing transitions of a system if an action $a \in A(x)$ is selected at state x, and $r(x|a)$ is a reward obtained at x if a is applied.

The goal of a decision maker (DM) is to maximize the total expected discounted reward on an infinite time horizon, or to average an expected reward or some other criterium. In the OS model, the set $A(x)$ consists of two actions: continue and stop (quit).

In our model a decision maker (DM) can apply one of three possible actions—*continue*, when system continues its evolution as Markov chain (MC); *quit* when dynamics is stopped forever and a terminal reward is obtained; and *restart*, when a system continues its dynamics from one of finite number of fixed "restarting" states. If there are $m > 1$ restarting states, then the last restart action consists in fact from m distinct actions. Each action is accompanied by a corresponding fee (reward), which can be positive or negative and depends on the state of a system where this action was taken. We consider the case when the goal of DM is to maximize the total expected reward on an infinite time horizon. For sake of simplicity we call this model Continue-Quit-Restart (CQR) model. CQR model is also a generalization of a model in Sonin [2008], which in turn is a natural generalization of a model of Katehakis and Veinott [1987], where a DM has two options to continue, or to restart to a unique point with zero fee for a restart. Our model is also related to such important notion as Gittins index and its generalizations. We will elaborate on this later.

Formally, a general CQR model is specified by a tuple

$$M = (X, B, P, A(x), c(x), q(x), r_i(x), i = 1, 2, ..., m, \beta),$$

where X is a finite (countable) state space, $B = \{s_1, ..., s_m\}$ is a fixed subset of X, and $P = \{p(x, y)\}$ is a stochastic matrix. At each state x a set of available actions $A(x) = \{c, q, r_j, j = 1, .., m\}$ is given. A reward function $r(x|a)$, with $x \in X, a \in A(x)$, is specified by particular functions $c(x)$, $q(x)$, and $r_i(x)$, $i = 1, 2, ..., m$. If an action c, *continue*, is selected, then $r(x|c) = c(x)$, and transition to a new state occurs according to transition probabilities $p(x, y)$. If an action q, *quit*, is selected, then $r(x|q) = q(x)$, and transition to an absorbing state e occurs with probability one. If an action r_i, *restart to state* s_i, is selected, then $r(x|r_i) = r_i(x)$, and transition to a state s_i occurs with probability one. Coefficient β is a *discount factor,* $\beta \leq 1$. Later we consider a more general case of a variable discount $\beta(x)$.

As in Katehakis and Veinott model, it is convenient to assume that after restart a new "cycle" starts *instantly* at the moment of restart. So, at the moment of restart to s_i from some point x, an action is also chosen at s_i, a transition according to this action occurs, and a corresponding extra reward $c(s_i)$, $q(s_i)$, or $r_j(s_i)$ is obtained. Here we consider the the case when $m = 1$, i.e., there is only one restart point.

We denote by $v(x)$ the value function in this model, i.e., sup of the total expected discounted reward on an infinite time horizon with an initial point x over all possible strategies. We assume that the value functions $v(x) < \infty$ for all $x \in X$. In this case, a general theory of MDP models implies that it is sufficient to consider only the nonrandomized stationary strategies. Such strategies can be defined by a partition of a state space into three disjoint sets, 3-partitions, where each of these sets specifies a particular action which is applied when MC hits this set. An optimal partition exists and is *uniformly optimal*, i.e., optimal for all initial points.

Our main goal is to construct an algorithm to find an optimal strategy (partition) and the value function for the CQR model.

We will extensively use the results and methods for a particular case of CQR and MDP models, a well-known *Optimal Stopping (OS) model.* In this model, an action set at each point consists of only two actions: $A(x) = \{c, q\}$, namely, continue and quit (usually called stop). We also have a *one step reward (cost)* function $r(x|c) = c(x)$, and a *terminal reward* function $r(x|q) = q(x)$; both functions are defined on X. The *value function* $v(x)$ for an OS model is defined as $v(x) = \sup_{\tau \geq 0} E_x[\sum_{i=0}^{\tau-1} \beta^i c(Z_i) + \beta^\tau q(Z_\tau)]$, where the sup is taken over all stopping times τ, $\tau \leq \infty$, and β is a discount factor, $\beta \leq 1$. If $\tau = \infty$ with positive probability, we assume that $q(Z_\infty) = 0$. It is well known that function v is a minimal solution of the corresponding Bellman equation, which has a form $v = \max(q, c+\beta P v)$, where $Pf(x) = \sum p(x, y) f(y)$. Denote by S the set $S = \{x : v(x) = q(x)\}$. If the state space X is finite, then the random time $\tau_0 = \min\{n \geq 0 : Z_n \in S\}$ is an optimal stopping time. The set S is called *the optimal stopping set.* We are going to extensively use the so-called State Elimination (SE) algorithm to solve OS problems, this algorithm was developed by one of the authors, see Sonin [1999a,b, 2006].

For remainder of the work we employ the notation of a *Reward Model* to describe the stopping model without termination reward, i.e. with $q(x) = -\infty$ for all x.

Under the assumption that there is only one point of restart, $m = 1$, we distinguish three situations, each of them is a special case of the next one

- CQR model with no quit action, free restart $q = -\infty$, $r = 0$,
- CQR model with no quit action allowed, $q = -\infty$, $r < \infty$,
- CQR model, $q < \infty$ and $r < \infty$.

The first case with no quit and free restart (it coincides with Katehakis-Veinott model, which is defined later), is a direct generalization of a classical Gittins index, and is described in Sonin [2008]. The algorithm solving CQR model, can also solve other cases, but it is substantially more complicated than its version to solve CQR model with not quit. Therefore, to make our ideas clear, we prefer to present the algorithm in two steps: solution for case with no quit action, and, separately, solution for case with quit action.

1.1 Markov Decision Processes

The goal of this section is to provide main definitions and facts from general theory of the Markov Decision Processes (MDP), used in this text.

MDP is defined through the following objects

- a state space X;
- an action space A;
- sets $A(x)$ of available actions at states $x \in A$;
- transition probabilities, denoted by $p(Y|x, a)$;
- reward functions $r(x, a)$ denoting the one step reward using action a in state x.

The meaning of these objects as follows. The *state space* defines possible states of underlying stochastic system. Given state $x \in X$, the decision maker (DM) can select an action from the set of available actions $A(x)$. After an action a is selected, the system moves to the next state according to the probability distribution $p(\cdot|x, a)$ and the decision maker collects one step reward $r(x, a)$.

An MDP is called finite if the state and the action sets are finite. An MDP is called discrete if both state and action sets are finite or countable. From now on we only consider finite and discrete MDPs.

For a discrete state space X we use letters x, y and also i, j, k to denote states. Transition probabilities are denoted as $p(x, y)$, p_{ij}, $p(x, y|a)$, or $p(y|x, a)$. Unless mentioned otherwise, we always assume $p(X|x, a) = 1$.

The time parameter is usually denoted by n, t, or $s \in \mathbb{N}$. The trajectory is a sequence $x_0 a_0 x_1 a_1 ....$ The set of all trajectories is $H_\infty = (X \times A)^\infty$. A trajectory of length n is called a history, and denoted by $h_n = x_0 a_0 ... x_{n-1} a_{n-1} x_n$. Let $H_n = X \times (A \times X)^n$ be the space of histories up to epoch $n \in \mathbb{N}$.

A non-randomized policy π is a sequence of mappings π_n, $n \in N$, from H_n to A such that $\pi_n(H_n) \in A(x_n)$. If for each n this mapping depends only on x_n, then the policy π is called Markov. A Markov policy is called stationary if π_n do not depend on n. A stationary policy is therefore defined by a single mapping $\pi : X \to A$.

The evolution rule for the stochastic process with policy π is as follows. If at time n the process is in state x, having followed the history h_n, then the action is chosen (perhaps randomly) according to the policy π. If action a ensued, then at time $n+1$ the process will be in the state y with probability $p(y|x, a)$.

We denote by Π, Π^M, and Π^S the sets of all non-randomized, Markov, and stationary policies.

A randomized policy π is a sequence of transition probabilities $\pi_n(a_n|h_n)$ from H_n to A, $n \in \mathbb{N}$, such that $\pi_n(A(x_n)|h_n) = 1$. We denote by Π^R, Π^{RM}, and Π^{RS} the sets of all randomized, randomized Markov, and randomized stationary policies respectively.

Given an initial state x and policy π, the evolution rule described above defines all finite-dimensional distributions $x_0, a_0, \ldots, x_n$, $n \in \mathbb{N}$. Kolmogorov's extension theorem guarantees that any initial state x and any policy π define a stochastic sequence $x_0 a_0 x_1 a_1 \ldots$. We denote by $\mathbb{P}_x^\pi$ and $\mathbb{E}_x^\pi$ respectively the probabilities and expectations related to this stochastic sequence; $\mathbb{P}_x^\pi \{x_0 = x\} = 1$.

Let f be the terminal reward function and β be the discount factor. We denote by $v_N(x, \pi, \beta, f)$ the expected total reward over the first N steps, $N \in \mathbb{N}$:

$$v_N(x, \pi, \beta, f) = E_x^\pi \left[\sum_{n=0}^{N-1} \beta^n r(x_n, a_n) + \beta^N \right],$$

whenever this expectation is well-defined.

The expected total reward over an infinite horizon is

$$v\,(x,\pi) = v\,(x,\pi,\beta) = v_\infty\,(x,\pi,\beta,0)\,.$$

If the reward function r is bounded either from above or from below, the expected total rewards over the infinite horizon are well-defined when $\beta \in [0,1)$.

If a performance measure $g\,(x,\pi)$ is defined for all policies , we denote

$$G\,(x) = \sup_{\pi\in\Pi^R} g\,(x,\pi)\,.$$

In terms of the performance measures defined above, this yields the values

$$V_N\,(x,\beta,f) \triangleq \sup_{\pi\in\Pi^R} v_N\,(x,\pi,\beta,f)\,,$$

$$V\,(x) = V\,(x,\beta) \triangleq \sup_{\pi\in\Pi^R} v\,(x,\pi,\beta)\,.$$

The main result we use from general theory of MDP is given in many textbooks, for example, in Corollary 2.3 of Feinberg and Shwartz [2002]: if for value function, defined as expected total reward over infinite horizon, there exists nonrandomized stationary optimal policy.

Therefore, from this point, we consider only nonrandomized stationary policies.

CHAPTER 2: OPTIMAL STOPPING OF MCS AND STATE REDUCTION ALGORITHM

Optimal stopping model lacks the restart action as an MDP model, however, as it will be shown later, it is an essential tool for finding the value function in an MDP model.

2.1 Classical and Generalized Gittins, Kathehakis-Veinott, and w Indices

In this section we discuss the relationship of CQR model, and its versions with no quit, or free restart, to the classical problems and indices. This material is a brief, revised text from Sonin [2008].

Traditionally, the most well-known and the most studied is the model related to the classical Gittins index, $\gamma(x)$. This index plays an important role in the theory of Multi-armed bandit problems with *independent* arms. It also naturally appears in many other problems of stochastic optimization.

Let us recall some useful facts related to Gittins index. Given a reward model $M = (X, P, c(x), \beta)$, $\beta = const$, $0 < \beta < 1$, and point $s \in X$, $\gamma(x)$, is defined as the maximum of the expected discounted total reward on the interval $[0, \tau)$ per unit of expected discounted time for the Markov chain starting from x, i.e.,

$$\gamma (x) = \sup_{\tau>0} \frac{E_x \sum_{n=0}^{\tau-1} \beta^n c\,(Z_n)}{E_x \sum_{n=0}^{\tau-1} \beta^n} = (1-\beta) \sup_{\tau>0} \frac{E_x \sum_{n=0}^{\tau-1} \beta^n c\,(Z_n)}{1 - E_x \beta^\tau}, \tag{2.1}$$

where $0 < \beta < 1$, τ is a stopping time, $\tau > 0$. Here we used trivial equality $(1-\beta) \sum_{n=0}^{k-1} \beta^n = 1-\beta^k$. Without loss of generality we consider only stopping times—the moments of a first visit to a certain set $G \subset X, x \notin G$.

An interesting interpretation of the Gittins index, the so-called *Restart in State* interpretation, was given in Katehakis and Veinott [1987]. Given a reward model $M = (X, P, c(x), \beta)$, let us consider a *family* of Markov Decision models indexed by a *fixed* initial point $s \in X$, where a set of actions $A(x)$ has two actions—either to continue, or to restart to s and continue from there. In other words, MC (Z_n) starting from a point s can be restarted after a positive stopping time $\tau > 0$, and returned to the same point s, and so on.

Let $h(x|s)$ denote the supremum over all strategies of the expected total reward on the infinite time interval in this model with an initial point x, and restart point s. Using the standard results of Markov Decision Processes theory, Katehakis and Veinott proved that function $h(x|s)$ satisfies the equality

$$h(x|s) = \sup_{\tau>0} E_x[\sum_{n=0}^{\tau-1} \beta^n c(Z_n) + \beta^\tau h(s)], \tag{2.2}$$

and that $\gamma(s) = (1-\beta)h(s)$, where $h(s) = h(s|s)$ by definition. We refer to this model as Katehakis and Veinott model and to an index $h(s)$ as Katehakis and Veinott index.

Another important interpretation of the Gittins index, the so-called *Retirement Process* formulation, was provided in Whittle [1980]. Given a reward model $M = (X, P, c(x), \beta)$, $0 < \beta < 1$, he introduced the parametric family of OS models $M(k) = (X, P, c(x), k, \beta)$, where parameter k is a real number, and the terminal reward function $q(x) = k$ for all $x \in X$. Denote by $v(x, k)$ the value function for such model, i.e., $v(x,k) = \sup_{\tau\geq 0} E_x, \left[\sum_{n=0}^{\tau-1} \beta^n c(Z_n) + \beta^\tau k\right]$; denote $w(x) = \inf\{k : v(x,k) = k\}$. Since $\beta < 1$, it is optimal to stop immediately for sufficiently large k and $v(x,k) = k$. Thus $w(x) < \infty$. The results of Whittle imply that $v(x,k) = k$ for $k \geq w(x)$, $v(x,k) > k$ for $k < w(x)$, and $w(x) = h(x)$. Since Whittle index is a term used in literature for the other index we will use the term index $w(x)$. For sake of brevity, instead of a parametric family of OS models we shall say just Whittle model $M(k)$.

Combined with the results of Katehakis and Veinott, the last equality implies that $\gamma(x) = (1-\beta)$ $h(x) = (1-\beta)w(x)$. In Theorem 3 we will prove the equality $h(x) = w(x) = \alpha(x)$, where $\alpha(x)$ is an index generalizing $\gamma(x)$ in a more general setting.

To describe this more general setting, let us make the following almost trivial, but important remark. As usual in MDP theory, the optimizations problems with an explicit discount factor β, such as described above for CQR or OS models, are equivalent to problems where a state space is complemented by an absorbing point e, and new transition probabilities are defined as follows: for any state $y \neq e$ the probability of entering an absorbing point e in one step (probability of termination) is equal to $1-\beta$, and all other initial transition probabilities are multiplied by β. In other words, β is the probability of "survival". To implement our algorithm it is convenient to consider more general case with the variable discount factor $\beta(x)$, $x \in X$. Such assumption is quite natural in many problems, e.g., in replacement models, where states represent the possible condition of a machine. But the main reason lies in the fact that to apply the SE algorithm we need possibly variable discount factor. Therefore, from now on, for every model we assume that the state space X contains an absorbing point e, with $p(e,e) = 1$. Function $\beta(x)$ is the probability of "survival" at point x, so $1-\beta(x) = p(x,e)$. Strictly speaking, function $\beta(x)$ is completely specified by a new transition matrix P. However, to stress the presence of e and $\beta(x)$, we sometimes include $\beta(x)$ in the tuple M. Correspondingly, the notation E_x, P_x, and (Z_n) refers to such model, and now survival probabilities $\beta(\cdot)$ are automatically included under the signs P_x and E_x. The Bellman equation now has a form $v = \max(q, c + Pv)$. We also assume that $c(e) = 0$, and, without loss of generality, that $r(s) = 0$ in CQR model. We remind that restart action is in fact a pair of actions: restart to s, and make one more step at s.

CQR model with no quit and free restart is nothing else than three models described above where constant discount factor β is replaced by a variable survival probability (discount) $\beta(x)$. In this case, models and results of Katehakis and Veinott and Whittle, almost do not need any adjustments. Given a reward model with termination $M = (X, P, c(x), \beta(x))$, we again consider a *family* of Markov Decision models indexed by a *fixed* initial point $x \in X$. We again define $h(x)$ as the value function in a restart in x problem with an initial point x, i.e., $h(x) = h(x|x)$.

Similarly, we define index $w(x) = \inf\{k : v(x,k) = k\}$, where $v(x,k)$ is a value function in the (generalized) Whittle model $M(k) = (X, P, c(x), \beta(x), k)$. In this model we assume that $g(x) = k$ for $x \neq e$; $c(e) = q(e) = 0$.

However, we can not replace β by $\beta(x)$ or by $\beta(Z_n)$ in the Gittins index in (2.1). As a result, the, classical Gittins index $\gamma(x)$ was replaced by a generalized Gittins Index in Sonin [2008] as follows.

In the presence of an absorbing state e and subset $G \subset X$, for $x \notin G$, the numerator in (2.1) equals to $E_x \sum_{n=0}^{\tau-1} c(Z_n)$, where $\tau = \min(n : Z_n \in G \cup e)$. Such equality holds independently of whether $\beta(x)$ is a constant or variable. Let us denote this numerator by $R^\tau(x)$. In the presence of an absorbing state e, and when $\beta = const$, the denominator in the last expression in (2.1) equals to $P_x(Z_\tau = e)$. In the general case, when $\beta(x)$ can be variable, we denote $P_x(Z_\tau = e)$ by $Q^\tau(x)$, which is the probability of termination on $[0, \tau)$. We *define* the *generalized Gittins index*, $\alpha(x)$, for the model with termination as

$$\alpha(x) = \sup_{\tau > 0} \frac{R^\tau(x)}{Q^\tau(x)}, \tag{2.3}$$

i.e., $\alpha(x)$ is the *maximum discounted total reward per chance of termination.* In fact, similar form of an index was used in Tsitsiklis [1994], and earlier by Denardo et al. [2004], and by Mitten [1960]. Note that if $\beta(x)$ is a constant β, then the denominator in the second equality in (2.1) coincides with $Q^\tau(x)$, and, therefore, in this case $\gamma(x) = (1 - \beta)\alpha(x)$.

Theorem 2.1 (Sonin [2008])**.** *The three indices defined for a reward model with termination $M = (X, P, c(x), \beta(x))$ coincide, i.e., $\alpha(x) = h(x) = w(x)$.*

This theorem was proved using the specifics of these three models. Later, Sonin [2011] proved this theorem as a special case of a general equality, presented in Section 4. As a result of Theorem 1, any of three problems can be used as a basis to calculate $\alpha(x)$. Because the problem of calculation

$v(x, k)$ for a particular k can be reduced to solving stopping problems using the State Elimination algorithm, we find the Whittle family of OS models $M(k)$ the most convenient. The corresponding algorithm, described in Sonin [2008], sequentially calculates the index $\alpha(x)$ for all points $x \in X$ in an order that is not known in advance. If, for a finite set X, the goal is to find $\alpha(s)$ for a particular s, then we know only that $\alpha(s)$ will be obtained at some stage. We also can apply this algorithm to some cases of countable X.

In our subsequent presentation the starting point is Katehakis and Veinott model, whereas Whittle OS family is the main tool for its solution.

2.2 The State Reduction (SR) Approach

The State Reduction (SR) Approach is a relatively new method to recursively calculate many important characteristics of MCs.

Let us assume that a *Markov model* $M_1 = (X_1, P_1)$ is given, let $D \subset X_1$, $S = X_1 \setminus D$. Then the matrix $P_1 = \{p_1(x, y)\}$ can be decomposed as follows

$$P_1 = \begin{bmatrix} Q & T \\ R & P_{10} \end{bmatrix}, \tag{2.4}$$

where the substochastic matrix Q describes the transitions inside of D, P_{10} describes the transitions inside of S, and so on. Let us introduce the sequence of Markov times $\tau_0, \tau_1, ..., \tau_n, ...$, the moments of zero, first, and so on, return of (Z_n) to the set S. I.e., $\tau_0 = 0$, $\tau_{n+1} = \min\{k > \tau_n, Z_k \in S\}$. Let us consider the random sequence $Y_n = Z_{\tau_n}$, $n = 0, 1, 2, ...$ The strong Markov property and standard probabilistic reasoning imply the following basic lemma of the SR approach, which probably should be credited to Kolmogorov and Doeblin.

Lemma 2.2. *(a) The random sequence (Y_n) is a Markov chain in model $M_2 = (X_2, P_2)$, where $X_2 = X_1 \setminus D$, and*

(b) the transition matrix $P_2 = \{p_2(x, y), x, y \in S\}$ is given by the formula

$$P_2 = P_{10} + RU = P_{10} + RNT. \tag{2.5}$$

In this formula U is a matrix of the distribution of the MC at the moment of first return to S, and N is the *fundamental matrix* for the substochastic matrix Q, i.e., $N = \sum_{n=0}^{\infty} Q^n = (I - Q)^{-1}$, where I is the $|D| \times |D|$ identity matrix. This representation is given, for example, in the classical text of Kemeny et al. [1976]. We call models M_1 and M_2 *adjacent*. An important case is when the set D consists of one nonabsorbing point z. In this case formula (2.5) takes the form

$$p_2(x,\cdot) = p_1(x,\cdot) + p_1(x,z)\, n_1(z)\, p_1(z,\cdot), \tag{2.6}$$

where $n_1(z) = 1/(1 - p_1(z,z))$. According to this formula, each row-vector of the new stochastic matrix P_2 is a linear combination of two rows of P_1 (with the column z deleted). Formally, this transformation corresponds to one step of the Gaussian elimination method.

Described above matrix $N = \{n(x,y), x, y \in D\}$, a *fundamental matrix* for the transient MC with transition matrix Q, has the following well-known probabilistic interpretation: $n(x,y) = E_x \sum_{n=0}^{\tau_S} I_y(Z_n)$. Here τ_S is the moment of the first visit to S, $\tau_S = \min(n > 0 : Z_n \in S)$ (moment of first exit from D), i.e., the expected number of visits to y starting from x till τ_S. The matrix N also satisfies the equality

$$N = I + QN = I + NQ. \tag{2.7}$$

If an initial Markov model $M_1 = (X_1, P_1)$ is finite, $|X_1| = k$, and only one point is eliminated at each time, then a sequence of stochastic matrices $(P_n), n = 2, ..., k$ can be calculated recursively on the basis of formula (2.6). Generally, a set of points D can be eliminated using formula (2.5). In both cases such sequence of stochastic matrices provides an opportunity to recursively calculate many characteristics of the initial Markov model M_1 starting from some reduced model $M_s, 1 < s \le k$. This approach was initiated by papers Grassmann et al. [1985] and Sheskin [1985], where the so-called GTH/S algorithm to calculate the invariant distribution for an ergodic Markov chain was obtained. The recursive calculation of the second fundamental matrix for the ergodic MC was described in Sonin and Thornton [2001].

2.2.1 Transformation of the cost function

Let us also introduce a *transformation of the cost function* $c_1(x)$ (or any function $f(x)$) defined on X_1 into the cost function $c_2(x)$ defined on $X_2 = S$, under the transition from model M_1 to model M_2.

Given the set $D, D \subset X_1$, let τ be the moment of the first *return* to X_2, i.e., $\tau = \min(n \ge 1, Z_n \in X_2)$. Then, given the function $c_1(x)$ defined for $x \in X_1$, let us define function $c_2(x)$ on $x \in X_2$ as

$$c_2(x) = E_x \sum_{n=0}^{\tau-1} c_1(Z_n) = c_1(x) + \sum_{z \in D} p_1(x,z) \sum_{w \in D} n(z,w) c_1(w). \tag{2.8}$$

In other words, the new function $c_2(x)$ represents the expected cost (reward) gained by a MC starting from point $x \in X_2$ up to the moment of first return to X_2. For a function $f(x)$ defined on a set X_1 and a set $B \subset X_1$ denote by f_B the column-vector function reduced to a set B. Then formula (2.8) can be written in a matrix form as

$$c_2 = c_{1,X_2} + RNc_{1,D}. \tag{2.9}$$

If the set $D = \{z\}$, then the function $c_1(x)$ is transformed as follows

$$c_2(x) = c_1(x) + p_1(x,z)n_1(z)c_1(z), \quad x \in X_2. \tag{2.10}$$

Remark 2.3. This formula was used first in Sheskin [1999] in the context of MDP.

2.2.2 Relation between G_1 and G_2

Now we present some useful formulas explaining how operators P_1 and P_2, and related operators act on functions in two adjacent models. We denote $F_i f(\cdot) = c_i + P_i f(\cdot)$, and $G_i f(\cdot) = f(\cdot) - (c_i + P_i f(\cdot))$. This lemma was not described in the original version of SE algorithm, and was proved in Sonin [2006].

Lemma 2.4. *Let M_1 and M_2 be two adjacent models with state spaces X_1 and $X_2 = X_1 \setminus D$, where $D \subseteq X_1$, P_i, and $F_i, i = 1,2$ be the corresponding averaging and reward operators, where functions c_1 and c_2 are related by (2.9), matrices R, T are as in (2.4) and matrix N is a fundamental matrix for Q. Let f be the function defined on X_1. Then*

$$f_{X_2} - P_2 f_{X_2} = (f - P_1 f)_{X_2} + RN(f - P_1 f)_D, \tag{2.11}$$

$$f_D = N[Tf_{X_2} + (f - P_1 f)_D]. \tag{2.12}$$

The formula similar to (2.11) holds if operators P_i are replaced by operators F_i and G_i, i.e.

$$G_2 f_{X_2} = (G_1 f)_{X_2} + RN(G_1 f)_D. \tag{2.13}$$

If set $D = \{z\}$, these formulas take the form ($x \in X_2$)

$$f(x) - P_2 f(x) = f(x) - P_1 f(x) + p_1(x,z)n_1(z)(f(z) - P_1 f(z)), \tag{2.14}$$

$$f(z) = n_1(z)(\sum_{y \in X_2} p_1(z,y)f(y) + f(z) - P_1 f(z)), \tag{2.15}$$

and

$$G_2 f(x) = G_1 f(x) + p_1(x,z) n_1(z) G_1 f(z). \tag{2.16}$$

2.2.3 The State Elimination Algorithm for optimal stopping of Markov chain

We consider here only the finite state space, though the method with some modifications can also be used in a countable state space. The State Elimination (SE) algorithm for the optimal stopping problem of an MC is based on three following facts.

Fact 2.5. *Though in the optimal stopping problem it may be difficult to find the states where it is optimal to stop, it is easy to find a state (states) where it is optimal not to stop. In reality, it is optimal to stop at z if $q(z) \geq c(z) + Pv(z) \equiv Fv(z)$, but v is unknown until the problem is solved. On the other side, it is optimal not to stop at z if $q(z) < Fq(z)$, i.e., the expected reward of doing one more step, then stopping, is larger than the reward from stopping. (Generally, it is optimal not to stop at any state where the expected reward of doing some, perhaps random number of steps, is larger than the reward from stopping).*

Fact 2.6. *After we have found states (state) that are not in the optimal stopping set, we can eliminate them and recalculate the transition matrix using (2.6), if one state is eliminated, or (2.5), if a larger subset of the state space is eliminated. Such transformation will keep the distributions at the moments of visits to any subset of remaining states the same, and the excluded states do not matter since it is not optimal to stop there. After that, in the reduced model we can repeat the first step and so on.*

Fact 2.7. *Finally, though if $q(z) \geq Fq(z)$ at a particular state z, we cannot make a conclusion about whether this state belongs to the stopping set or not, but if this inequality is true for all states in the state space, then we have the following simple statement*

Proposition 2.8. *Let $M = (X, P, q)$ be an optimal stopping problem, and $q(x) \geq Fq(x)$ for all $x \in X$. Then X is the optimal stopping set in the problem M, and $v(x) = q(x)$ for all $x \in X$.*

The formal justification of the transition from the initial model M_1 to the reduced model M_2 is given by Theorem 2.9 below. This theorem was formulated in Sonin [1995] and its proof was given in Sonin [1999a] for the case when $c(x) = 0$ for all x.

Algorithm 2.1 State Elimination (SE) Algorithm

Input: optimal stopping model $M(X, P, c, q)$
Output: optimal stopping set S^*, value function $v(x)$
Assumption: optimal stopping set $S^* = \{x : v(x) = q(x)\}$ does exists
$k \leftarrow 1$
$(X_k, P_k, c_k, q_k) \leftarrow M(X, P, c, q)$
while $\exists x : q(x) - P_k q(x) < 0$ **do**
  $k \leftarrow k + 1$
  $D_k \leftarrow \{x : q(x) < P_{k-1} q(x)\}$
  $X_k \leftarrow X_{k-1} \diagdown D_k$
  $(P_k, c_k) \leftarrow$ apply formulas (2.5), (2.9) to (P_{k-1}, c_{k-1}) by eliminating D_k
  $q_k \leftarrow$ remove states D_k from q_{k-1}
end while
$S^* \leftarrow X_k$
$v(x) \leftarrow q(x)$ for $x \in S^*$
$v(x) \leftarrow$ apply formula (2.17) for $x \in X \diagdown S^*$

Theorem 2.9 (Elimination theorem). *Let $M_1 = (X_1, P_1, c_1, q)$ be an OS model, $D \subseteq C_1 = \{z \in X_1 : q(z) < F_1 q(z)\}$. Consider an OS model $M_2 = (X_2, P_2, c_2, q)$ with $X_2 = X_1 \setminus D$, $p_2(x, y)$ defined by (2.5), and c_2 is defined by (2.9). Let S be the optimal stopping set in M_2. Then*

1. *S is also the optimal stopping set in M_1, and*

2. *$v_1(x) = v_2(x) \equiv v(x)$ for all $x \in X_2$, and for all $z \in D$*

$$v_1(z) = E_{1,z}[\sum_{n=0}^{\tau-1} c_1(Z_n) + v(Z_\tau)] = \sum_{w \in D} n_1(z, w) c_1(w) + \sum_{y \in X_2} u_1(z, y) v(y), \tag{2.17}$$

where $u_1(z, \cdot)$ is the distribution of an MC at the moment τ of first visit to X_2, and $N_1 = \{n_1(z, w), z, w \in D\}$ is the fundamental matrix for the substochastic matrix Q_1.

The state elimination algorithm is given in Algorithm 2.1. It takes OS model $M = (X, P, c, q)$ as input, and assumes that optimal stopping set $S^* = \{x : v(x) = q(x)\}$ does exists. For the finite space X this algorithm solves the OS problem in no more than $|X|$ steps, it also allows us to find the distribution of the MC at the moment of stopping in an optimal stopping set S^*. A similar idea was applied for a particular OS problem (the Secretary Problem with random number of objects) in Sonin and Presman [1972], and was proposed for the OS of general stochastic processes in Irle [1980] without the specification to MC situation.

2.3 State Elimination Algorithm with Full Size Matrices

It could be more convenient for the implementation of state elimination algorithm to have all stochastic matrices of equal *full size*. Denote deleted set as D; $X \setminus D = S$. Introduce two diagonal characteristic matrices I_D and I_S, e.g., I_D is a diagonal matrix with $d_i = 1$ if $i \in D$ and 0 otherwise. We remind that multiplication on diagonal matrix on the right is equivalent to multiplication of columns, and multiplication on the left is equivalent to multiplication of rows. Therefore, the formulas in the previous sections can be rewritten as follows.

Now we can skip index 1 for the initial model, and skip index 2 in a new model M_2, i.e. $P_1 = P$, $P_2 = P_2(D) = P(D)$. Note that P, $P_2(D)$, $N(D) = N_D$, I_D, I_S are full size $|X| \times |X|$ square matrices.

Lemma 1 remains true, but now we assume that (Y_n) is an MC with the *same* state space X, i.e., we allow the initial points x be in D as well as in $S = X \setminus D$, though after the first step MC is always in S. Then, additionally to (2.5) for $x \in S, y \in S$, we have term $T + QNT = (I + QN)T = NT$ for $x \in D, y \in S$. The last equality is true by (2.7). Thus, instead of (2.5) we have the following full size stochastic matrix for an MC (Y_n)

$$P_2(D) = PI_S + PI_D N_D PI_S = (I + PI_D N_D)PI_S = N_D PI_S = \begin{bmatrix} 0 & NT \\ 0 & P_{10} + RNT \end{bmatrix}, \tag{2.18}$$

where P_{10} in formula (2.5) is replaced by PI_S, R is replaced by PI_D, T is replaced by PI_S, and $N = (I - Q)^{-1}$ is replaced by N_D. Here $N_D = (I - PI_D)^{-1} = I + PI_D N_D$,

$$N_D = \begin{bmatrix} N & 0 \\ RN & I \end{bmatrix}. \tag{2.19}$$

Also note that for $x \in D$ the rows of matrix $P_2(D)$ (namely, submatrix NT) give the distribution of MC (Y_n) at the moment of first *visit* to set S: $P_{2,x}(Y_1 = y), x \in D, y \in S$. And this moment coincides with the moment of first *return* to set S. For the points from set S we are interested in the moment of a first return, corresponding distribution is given by submatrix $P_{10} + RNT$.

The *full matrix* analog of (2.9) will be

$$\mathbf{c}_2 = \mathbf{c}_2(D) = \mathbf{c} + PI_D N_D \mathbf{c} = (I + PI_D N_D)\mathbf{c} = N_D \mathbf{c} = \begin{bmatrix} Nc_{1,D} \\ RNc_{1,D} + c_{1,S} \end{bmatrix}, \tag{2.20}$$

where $c_{1,D}$ and $c_{1,S}$ are the parts of vector $\mathbf{c} = \mathbf{c}_1$ with coordinates in D and S respectively. Now $\mathbf{c}$ and $\mathbf{c}_2$ are both full vectors defined on the whole $X = X_1$. As in formula (2.8), function $\mathbf{c}_2$ can be also described as

$$c_2(x) = E_{1x} \sum_{n=0}^{\tau-1} c_1(Z_n), x \in X, \tag{2.21}$$

where $E_{1x} = E_x$ is an initial expectation, $\tau = \tau_S$ is the moment of first *return* to $S = X \setminus D$ if $x \in S$.

The analog of Lemma 2, i.e., analogs of formulas (2.11)–(2.13) in *full matrix* form are**:** **(**$P = P_1, \mathbf{c} = \mathbf{c}_1$)

$$P_2(D) = P + PI_D N_D(P - I) = P + PI_D N_D(P - I), \tag{2.22}$$

$$F_2(D)f = Ff + PI_D N_D(F - I)f, \tag{2.23}$$

$$F_2(D)f - f = (F - I)f + PI_D N_D(F - I)f = (I + PI_D N_D)(F - I)f = N_D(F - I)f, \tag{2.24}$$

where $Ff = \mathbf{c} + Pf$, $F_2 f = \mathbf{c}_2 + P_2(D)f$. Later in the text the most important role will play the formula applied to the case $f = g$, where g is the terminal reward function. In this case we use the shorthand notation $G_i(\cdot) = G_i g(\cdot)$. This *main formula* (compare with (2.16)) for the case $D = \{z\}$ is

$$G_2(z) = n_1(z)G_1(z), G_2(x) = G_1(x) + p_1(x,z)n_1(z)G_1(z) = G_1(x) + p_1(x,z)G_2(z). \tag{2.25}$$

Note that set D in Lemma 2 is not necessarily a subset of $C_1 = \{z \in X_1 : G_1 f(z)\}$, but if it is, then formula (2.25) immediately implies

Corollary 2.10. *If the elimination set* $D \subset \{C_1 = \{z \in X_1 : G_1 q(z) < 0\}$ *then* $G_2 q_{X_2} < G_1 q_{X_2}$. *This also means that if some points were eliminated at some stage, then they are eliminated forever.*

Remark 2.11. Formula (2.25) also helps to organize the recursive steps of the EA in a more efficient way. If a set D is eliminated and new model M_D is obtained, then the new transition probabilities p_D have the following property

$$p_D(x,z) = 0, \text{ if } x \in S = X \diagdown D, z \in D; \; p_D(z,u) = 0, \text{ if } z, u \in D. \tag{2.26}$$

We say that an OS model $M = (X, P, c(x), g(x), \beta(x))$ has an *escaping* set D if transition matrix P has the same structure as in the formula above. In other words, MC can be in a set D only at the initial moment. Later we will use the following simple proposition.

Proposition 2.12. *If OS model M has an escaping set D and $q(x) \geq c(x) + Pq(x)$ for all $x \in S = X \diagdown D$. Then $v(x) = q(x)$ if $x \in S$.*

The proof of Proposition 2.12 is similar to the proof of Proposition 2.8.

Remark 2.13. The usage of the full-size matrices P_i also allows to obtain the value function at the end of elimination stage. Let D_i be a set eliminated on a i-th step, $D_i = \{x : q(x) - (c_i + P_i q(x) < 0\}$, $i = 1, 2, ..., k$, $S_i = X \diagdown D_i$. Denote the value function on a i-th step by $v_i = q_{D_i}$: $v_i(x) = q(x)$ if $x \in S_i$, and $v_i(x) = c_i(x) + P_{i+1}q$, if $x \in D_i$. We always have $D_i \subset D_{i+1}$ and $g \leq ... \leq v_i \leq v_{i+1} \leq ... \leq v$. Therefore, if, for some k, we have $D_{k+1} = D_k$, it means that calculation is done, and it also happened that we have obtained the optimal stopping set $S = S_k$ and value function $v(x) = v_k(x)$ for all $x \in X$.

Using Corollary 1 and formulas for the elimination steps it is easy to show the important feature of SEA, namely, that the elimination of sets D_1 and D_2 in two steps is equivalent to elimination of a set $D_1 \cup D_2$ in one step. This feature implies also that we can eliminate only one point at a time. Therefore, the implementation of SE algorithm can be pretty straightforward, it only needs the formulas for one step of elimination. It starts with $D = \emptyset$ and it can recursively eliminate states one by one until $D_n = D_{n+1}$.

2.4 State Elimination and Insertion

The equations from the previous sections are useful when there is no need to insert states back into the model. The full-size matrices complexity to eliminate single state using full-size matrices is $O\left(|X|\,|X \backslash D|\right)$ because columns, corresponding to states $x \in D$ contain zeroes. In this section we develop an algorithm, allowing insertion of $x \in D$ back to the model, the complexity of elimination or insertion is $O\left(|X|^2\right)$.

Denote by W_D the matrix, obtained after elimination of set D. Set $W_\emptyset = P$. Apply elimination of single state z exactly as before

$$w_{D\cup z}\left(\cdot, y\right) = w_D\left(\cdot, y\right) + w_D\left(\cdot, z\right) \frac{1}{1 - w_D\left(z, z\right)} w_D\left(z, y\right), \tag{2.27}$$

with one important difference: we apply this equation to all states $x \in X$, even to the states from $x \in D$.

Remark 2.14. Given D and $x, y \in X \backslash D$, the equation 2.27 is using only elements inside of $X \backslash D$, therefore $p_D\left(x, y\right) = w_D\left(x, y\right)$. Therefore, the portion of W_D, corresponding to the $X \backslash D$ is exactly equal to the values in P_D. In general, W_D contains non-zero elements at columns, corresponding to $x \in D$, whereas P_D has zeroes at these columns.

Remark 2.15. The equation (2.10) to eliminate state z for cost function c uses only elements P_D from $X \backslash D$, therefore we can use W_D in this equation, namely

$$c_{D \cup z}(y) = c_D(y) + w_D(y, z) \frac{1}{1 - w_D(z, z)} c_D(z), \tag{2.28}$$

Corollary 2.16. *Elimination can be performed on a matrix W_D, transition matrix P_D is obtained from W_D by setting columns, corresponding to set D to zero.*

Transition matrix P_D can be treated as matrix with $|X| - |D|$ columns and $|X|$ rows, matrix W_D has size $|X|$ rows and columns, regardless of the set D. As a result, elimination on W_D is computationally more expensive than on P_D. The main advantage of elimination on W_D is the ability to perform inverse operation to elimination, i.e. insertion of any state $j \in D$. Indeed, by applying simple algebra to equation 2.27 we have

$$w_D(\cdot, y) = w_{D \cup j}(\cdot, y) - w_{D \cup j}(\cdot, z) \frac{1}{1 + w_{D \cup j}(j, j)} w_{D \cup j}(j, y), \tag{2.29}$$

and

$$c_D(y) = c_{D \cup j}(y) - w_{D \cup j}(y, z) \frac{1}{1 + w_{D \cup j}(j, j)} c_{D \cup j}(z). \tag{2.30}$$

Matrix form for W_D is

$$W_D = \begin{bmatrix} QN & NT \\ RN & P_0 + RNT \end{bmatrix},$$

where new elements QN and RN have the following meaning

- for $x \in S$, $y \in D$, the element $(RN)_{xy}$ is expected number of times state y is visited while chain stays in D given that chain enters to D through x,
- for $x \in D$, $y \in D$, the element $(QN)_{xy} = (N - I)_{xy}$ is expected number of times state y is visited while chain stays in D given that chain enters to D through x, in other words even though chain starts in D, the first state is not counted.

Since P_D is exactly the same as in previous section, all statements, derived for P_D, in particular relations between F_1 and F_2, and relations between G_1 and G_2 are still true.

Remark 2.17. Elimination with W_D instead of P_D provides tradeoff between slightly increased complexity and new functionality, i.e., ability to insert.

Remark 2.18. Elimination of z and insertion of j has simpler form

$$w_{D\bigcup z}(\cdot, z) = \frac{w_D(\cdot, z)}{1 - w_D(z, z)}, \tag{2.31}$$

$$c_{D\bigcup z}(z) = \frac{c_D(z)}{1 - w_D(z, z)}, \tag{2.32}$$

$$w_D(\cdot, j) = \frac{w_{D\bigcup j}(\cdot, j)}{1 + w_{D\bigcup j}(j, j)}, \tag{2.33}$$

$$c_D(j) = \frac{c_{D\bigcup j}(j)}{1 + w_{D\bigcup j}(j, j)}. \tag{2.34}$$

2.5 Three Abstract Optimization Problems

The common part of all three problems described above in Section 2 is a maximization over the set of all positive stopping times τ, or, equivalently, over all partitions of the state set X into two sets, continuation and stopping (restart) regions. This is a special case of a very general situation.

Let us consider the following three abstract optimization problems 1, 2, and 3. Suppose there is an abstract index set U, let $A = \{a_u\}$ and $B = \{b_u\}$ be two sets of real numbers indexed by the elements of U. Suppose that the following assumption holds

$$-\infty < a_u \leq a < \infty, \quad 0 < b \leq b_u \leq 1.$$

We assume, that DM knows sets U, A, and B in all three problems.

Problem 2.19 (Restart Problem)**.** Find solution(s) of the equation

$$h = \sup_{u \in U}[a_u + (1 - b_u)h] \equiv H(h). \tag{2.35}$$

It is easy to see that equation (2.35) is a Bellman (optimality) equation for the "value of the game," i.e., the supremum over all possible strategies in the optimization problem with two equivalent interpretations. In both cases set U represents a set of available actions, which we call "buttons." A DM can select one of them and push (test). She obtains a *reward* a_u, and, according to the first interpretation, with *probability* b_u, the game is *terminated*, and, with complimentary probability

$1 - b_u$, she is again in an initial situation, i.e., she can select any button and push. Her goal is to maximize the total (undiscounted) reward.

According to the second interpretation, the game is continued sequentially without possibility of random termination, but the value $1 - b_u$ is now not a probability, but a discount factor applied to the future rewards after a button u was used at the first step.

Our second optimization problem is

Problem 2.20 (Ratio (cycle) Problem)**.** Find

$$\alpha = \sup_{u \in U} \frac{a_u}{b_u}. \tag{2.36}$$

The interpretation of this problem is straightforward: a DM can push some button u only once and her goal is to maximize the ratio in (2.36), the one step reward per "chance of termination." Since the game is terminated after the first push anyway, $1/b_u$ has an interpretation of a "multiplicator" applied to a "direct" reward a_u.

In the sequel we shall use shorthand notation $a \vee b$ for $\max(a, b)$. Let $H(k)$ be a function defined in the right side of (2.35).

Problem 2.21 (A Parametric Family of Retirement Problems)**.** Find w defined as follows: given parameter $k, -\infty < k < \infty$, let

$$v(k) = k \vee H(k), \quad w = \inf\{k : v(k) = k\}. \tag{2.37}$$

In this problem, given number k, a DM has the following one step choice: to obtain k immediately, or to push some button u once, then obtain a reward a_u, after that, additionally with probability $1 - b_u$, to obtain k, and, with complimentary probability, to obtain zero.

Using the fact that functions $H(k)$ and $v(k), -\infty < k < \infty$, are nondecreasing, continuous, and convex (concave up), the following theorem was proved in Sonin [2011].

Theorem 2.22 (Abstract Optimization Equality)**.** *a) Solution h of equation (2.35) is finite and unique;*

b) $h = \alpha = w$, and

c) the optimal index, or an optimizing sequence for any of the three problems is the optimal index (an optimizing sequence) for the other two problems.

See the brief discussion of one more problem initially analyzed in one page seminal paper Mitten [1960], and its relation to the classical- and generalized Gittins index in Sonin [2008].

Theorem 2.22 shows the equivalence of three abstract problems, but leaves an open question: which one of them should be solved. Probably, there is no general answer to this question. It is possible that in some situations Problem 1 will be the easiest, and in some other—Problem 2. At the same time Problem 3 provides the most general approach, since its solution breaks up into two stages: a solution for a particular k, and finding w. This exact situation occurs in Markov reward model and three related indices. Let us formally show how the three problems, described in sections 1 and 2 can be presented as abstract problems.

Given a reward model with termination $M = (X, P, c(x), \beta(x))$ and an initial point x, let us define the set $U = \{u\} = \{$ set of all Markov moments $\tau > 0\}$, $\tau = \tau_G = \min(n : Z_n \in G \cup e), G \subset X, x \notin G$. We define rewards a_u as $a_u = R^\tau(x) = E_x \sum_{n=0}^{\tau-1} c(Z_n)$, the total expected reward till moment τ; the probabilities b_u are defined as $Q^\tau(x) = P_x(Z_\tau = e)$, the probability of termination on $[0, \tau)$. These are quantities participating in (2.3). Then the function $H(k)$ coincides with $\sup_{\tau>0} E_x q(Z_\tau)$, where $g(x) = k$. Respectively, $v(x|k) = k \vee H(k) = \sup_{\tau \geq 0} E_x q(Z_\tau)$, i.e., $v(x|k)$ is the value function in an OS for MC in model $M(k)$.

Also note that the equivalence of the three problems does not lend itself to the solution of these problems. The set of all partitions of X, which gives the size of the set U, grows exponentially with $|X| = n$; but the algorithm in Sonin [2008] to calculate generalized Gittins index is polynomial with complexity of order $O\left(n^3\right)$. A similar algorithm to calculate the classical Gittins index was obtained in Niño-Mora [2007].

CHAPTER 3: ALGORITHM FOR CQR MODEL

Consider the CQR model $(X, P, A, c(x), q(x), r(x))$ with single restart point $s \in X$, as defined in Introduction. We follow previous assumption that the discount factor $\beta(x)$ is already factored in into the transition probabilities by using transition to the terminal state e. Our final goal is find optimal strategy π, maximizing the value function $h(x)$.

The algorithm is based on solving an equivalent problem. The equivalence of problems is established using three abstract optimization problems, i.e. theorem 2.22.

The derivation consists of several steps. First step is to write value function for CQR problem in form of abstract optimization problem. Second step is to define indices α, w, and h in modified form. Modified indices are defined for every state, however, the theorem 2.22 is applicable to the restart state s only. The last step is define a family of models, corresponding to the modified index $w(s)$ and to develop algorithm to find this index.

3.1 Value function for CQR model

An action set $A(x)$ at each point x consists of three actions $\{c, q, r\}$: continue, quit, and restart (to a fixed point s). The exception is a restart point s, where action set consists only of two actions, $\{c, q\}$. In addition, the absorbing state, e, has only continue action, $A(e) = \{c\}$. In order to simplify all equations, we consider, that state s still has restart fee, $r(s) := 0$. Also, the absorbing state has all fees equal to 0.

Respectively a stationary strategy π is defined as a 3-partition of $X = C \cup S_q \cup S_r$, where C is a continuation region, S_q is a quit region, S_r is a restart region. Denote the value function in this model as

$$h(x) = \sup_{\pi} h^{\pi}(x).$$

Since three possible actions are available at each state x, the value function $h(x)$ satisfies optimality equation

$$h(x) = q(x) \vee (r(x) + h(s)) \vee (c(x) + Ph(x)), \tag{3.1}$$

where $v(e) = 0$, and, for any function g, defined on states, $Pg(x) = \sum_y p(x, y)g(y)$. Notice that $h(s)$ has simpler form, $v(s) = q(s) \vee (c(s) + Pv(s))$.

Define a "stopping set" as set of point outside of continue action, $S = X\backslash C = S_q \cup S_r \cup \{e\}$. Given a strategy $\pi = \{C, S_q, S_r\}$, let the stopping time$\tau = \tau(S) = \tau(\pi)$ be a moment of a first visit to S. The moment τ is a moment when a cycle ends, i.e., when a DM stops (quits or restarts).

The following expected rewards and probabilities help with rewriting value function in terms of moment τ. Define the *probability of the termination* (of a cycle) on $[0,\tau]$ as probability of choosing quit action or reaching absorbing state at the moment τ

$$Q^{\pi}(x) = \mathbb{P}_x\left[Z_\tau \in S_q\right] + \mathbb{P}_x\left[Z_\tau = e\right].$$

Define $R^{\pi}(x)$, the total *expected reward obtained during one cycle as* sum of rewards for continue action, obtained before moment τ, plus reward at moment τ, which can be either reward for quit action or reward for restart to s action

$$R^{\pi}(x) = \mathbb{E}_x\left[\sum_{n=0}^{\tau-1} c\left(Z_n\right) + I\left(Z_\tau \in S_q\right) q\left(Z_\tau\right) + I\left(Z_\tau \in S_r\right) r\left(Z_\tau\right)\right],$$

do not forget that all rewards at absorbing state are zero.

Then, using the standard results from the theory of MDP we have value function for strategy π as

$$h^{\pi}(x) = R^{\pi}(x) + \left(1 - Q^{\pi}(x)\right) h^{\pi}(s). \tag{3.2}$$

This equation means that, starting from state x we obtain expected reward during one cycle, $R^{\pi}(x)$, then, with complimentary probability to the termination probability $Q^{\pi}(x)$, we obtain $h^{\pi}(s)$.

Taking supremum over all possible strategies in (3.2), using $x = s$ and assumption that $r(s) = 0$, we obtain that the optimality equation (3.1) can be written as

$$h(s) = \sup_{\pi}[R^{\pi}(s) + (1 - Q^{\pi}(s))h(s)] = q(s) \vee \sup_{\pi: A(s)=c}[R^{\pi}(s) + (1 - Q^{\pi}(s))h(s)]. \tag{3.3}$$

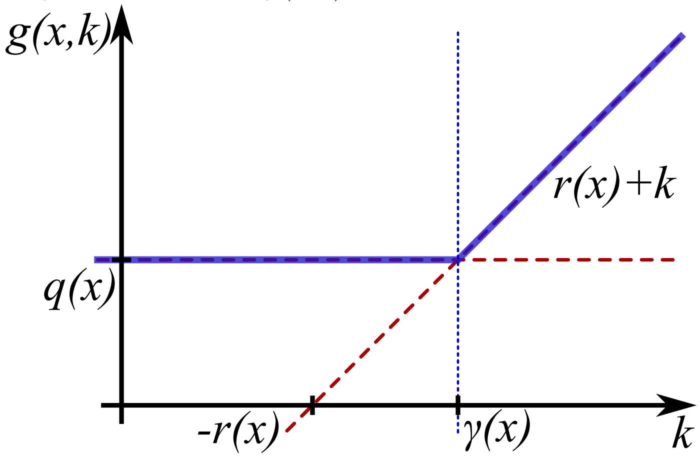
Therefore we represented value function for a restart point, $h(s)$, as one of the three abstract problems, namely, restart problem, as in equation 2.35.

3.2 Definition of modified indices

In order to find optimal strategy, we need to move away from the value function in CQR model and define modified indices.

The idea is to introduce indices $\alpha(x)$, $\tilde{h}(x)$, and $w(x)$ for *all initial states* x in such a way, that,

Figure 3.1: Graph of $g(x,k)$ for fixed state x, red dashed lines correspond to $q(x)$ and $r(x)+k$.



on one hand, the theorem 2.1 is preserved for all x, and, on the other hand, the value $\tilde{h}(x)$ for $x = s$ will coincide with value function $h(s)$ as defined in (3.3). Then, we can reduce problem of finding strategy, maximizing value function $h(s)$ to finding optimal strategy, maximizing modified index $\tilde{h}(x)$.

The modified indices $w(x)$ and $t(x)$ require introduction of a Whittle family of models $M(k)$ with the same state space X, transition probability P, action set consisting only from two actions, $A(x) = \{continue, stop\}$, with the same as CQR model $c(x)$ and terminal reward function defined as

$$g(x,k) = q(x) \vee (r(x)+k),\ x \neq e, \tag{3.4}$$

the absorbing state has the same properties, i.e. it has only one action, continue; in order to simplify notation, we set $g(e,k) = 0$. In short, $M(k) = (X, P, A = \{continue, stop\}, c(x), g(x,k), k \in \mathbb{R})$. The graph of $g(x,k)$ is given in Figure 3.1.

Problem 3.1 (Modified Restart index $\tilde{h}(x)$)**.** We define an index $\tilde{h}(x)$ for all $x \in X$ as

$$\begin{aligned} \tilde{h}(x) &= \sup_{\pi}[R^{\pi}(x) - r(x) + (1 - Q^{\pi}(x)\tilde{h}(x)] \\ &= (q(x) - r(x)) \vee \sup_{\pi:A(x)=c}[R^{\pi}(x) - r(x) + (1 - Q^{\pi}(x)\tilde{h}(x)], \end{aligned} \tag{3.5}$$

where strategy π is a partition (C, S_q, S_r) of X, $\tau = \tau_S$, $S = S_q \cup S_r \cup \{e\}$; notation $\pi : A(x) = c$ means that $x \in C$, so moment$\tau > 0$.

In other words, we define $\tilde{h}(x)$ as a value function for CQR problem with an initial point x not s, where, additionally, we subtract extra "initiation" fee $r(x)$ from expected reward during one cycle.

Condition $r(s) = 0$ implies that $h(s)$ defined by (3.3) coincides with $\tilde{h}\,(\mathrm{s})$. In general, $\tilde{h}(x) \neq h(x)$ for $x \neq s$ even if $r(x) = 0$ because the return points are different, x for the index $\tilde{h}\,(x)$, and s for $h\,(x)$.

The reason for subtracting $r(x)$ will become clear when a modified index $w(x)$ is introduced.

Problem 3.2 (Modified Gittins index $\alpha(x)$)**.** We define index $\alpha(x)$ as

$$\alpha(x) = \sup_{\pi} \frac{R^{\pi}(x) - r(x)}{Q^{\pi}\,(x)} = (q(x) - r(x)) \vee \sup_{\pi : A(x) = c} \frac{R^{\pi}(x) - r(x)}{Q^{\pi}\,(x)}, \tag{3.6}$$

where π, $R^{\pi}(x)$, and $Q^{\pi}(x)$ are defined as before.

We can use any index to find value of all others, for CQR find value of all indices through generalized index $w(x)$.

Problem 3.3 (Modified index $w(x)$ and index $t(x)$)**.** These indices are defined on Whittle family of models, $M\,(k)$, defined above. The strategy for this model is defined by stopping set $S \subset X$, where $S = \{x : A\,(x) = \mathit{stop}\}$. Let stopping time τ for $M\,(k)$ be the moments of a first visit to sets $S \subset X$. Then, $v^{\tau}(x, k)$ the value of a strategy τ at point x, is

$$v^{\tau}(x, k) = E_x[\sum_{n=0}^{\tau-1} c(Z_n) + g(Z_{\tau}, k). \tag{3.7}$$

Let

$$v(x, k) = \sup_{\tau \geq 0} v^{\tau}(x, k)$$

be the value function for model $M(k)$. The optimality equation has a standard form:

$$v(x, k) = g(x, k) \vee (c(x) + Pv(x, k)) = g(x, k) \vee \sup_{\tau > 0} \mathbb{E}_x g(Z_{\tau}). \tag{3.8}$$

Now we can define *modified indices* $w(x)$ and $t(x)$:

$$\begin{aligned} w(x) &= \inf_{k} \{k : v(x, k) = r(x) + k\}, \\ t(x) &= \sup_{k \leq w(x)} \{k : v(x, k) = q(x)\}. \end{aligned} \tag{3.9}$$

Even though, the indices $w\,(x)$ and $t\,(x)$ are defined for family of Whittle models $M\,(k)$, we can show that $w\,(s) = \tilde{h}\,(s) = h\,(s)$.

Similarly to the statement in Sonin [2008] we have the following proposition.

Figure 3.2: Graph of $v(x,k)$ for fixed state x, red dashed lines correspond to $q(x)$ and $r(x)+k$. Case when optimal strategy, as function of k, consists of quit, continue, and restart.

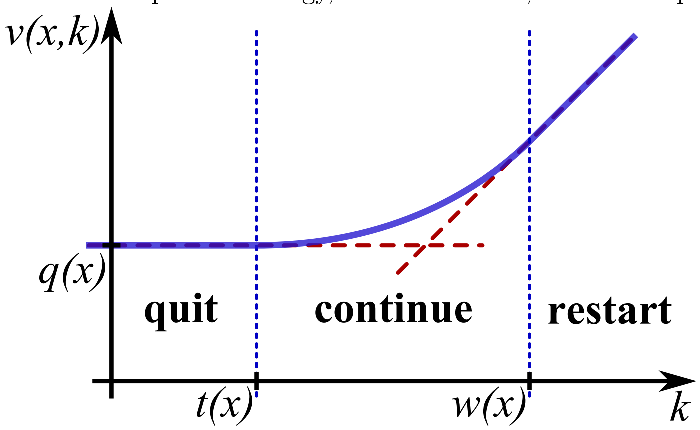
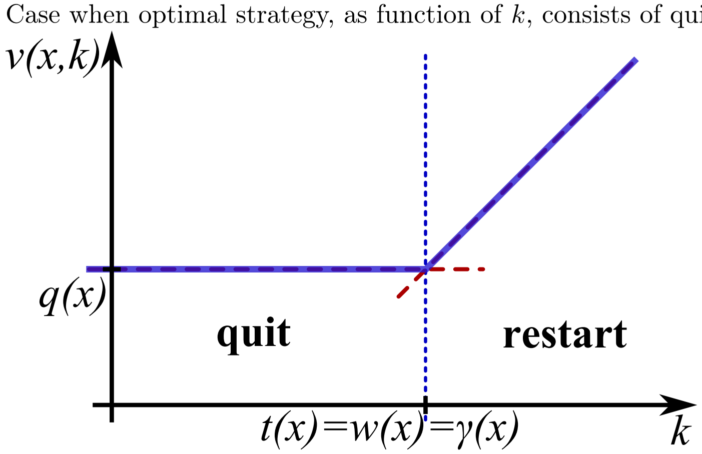


Figure 3.3: Graph of $v(x,k)$ for fixed state x, red dashed lines correspond to $q(x)$ and $r(x)+k$. Case when optimal strategy, as function of k, consists of quit and restart.



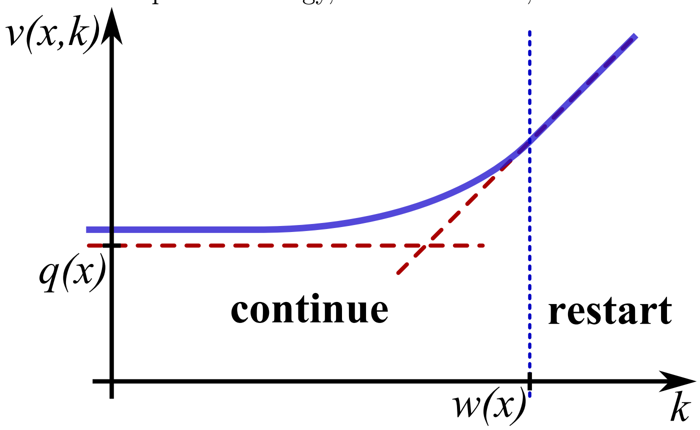
Proposition 3.4. *The indices $t(x)$ and $w(x)$ satisfy $t(x) \le w(x) < \infty$. Value function $v(x,k)$ is concave upward and*

$$\begin{aligned} v(x,k) &= r(x)+k, \quad k \ge w(x) \\ v(x,k) &> g(x,k), \quad k \in (t(x), w(x)), \\ v(x,k) &= q(x), \quad k \le t(x). \end{aligned}$$

It follows from the proposition, that $x \in S(k)$ if $k \le t(x)$ or $k \ge w(x)$. The function $v(x,k)$ for given state x is shown at Figures 3.2-3.4. As the result of this chapter, we prove that function $v(x,k)$ is continuous, concave upward and can have three shapes. The graphs are given here in advance in order to help with understanding of the rest of the section.

The set of all strategies on $M(k)$ consist of partition of state space X into two sets, S and $X \backslash S$.

Figure 3.4: Graph of $v\,(x,k)$ for fixed state x, red dashed lines correspond to $q\,(x)$ and $r\,(x)+k$. Case when optimal strategy, as function of k, consists of continue and restart.



However, the set of all strategies for CQR problem is richer, consists of partition of X into three sets: S_q, S_r, and $X\setminus(S_q\cup S_r)$. In order to apply Theorem 2.22, the abstract optimization equality, we need to transform equation (3.8) to an equation with supremum over all 3-partitions.

Lemma 3.5. *Value function $v(x,k)$ satisfies an equation*

$$v(x,k) = q(x) \vee (r(x)+k) \vee \sup_{\pi:A(x)=c} [R^{\pi}(x) + (1-Q^{\pi}(x))k]. \tag{3.10}$$

Proof. Given stopping set S and value k, use the reward function $g(x,k)$ to partition set S into $S_q(k)$ and $S_r(k)$. Set $S_q\,(k)$ is subset of S where $g\,(x,k)=q\,(x)$, $S_q(k)=\{x\in S: q(x)\geq r(x)+k\}$, set $S_r\,(k)$ is subset of S where $g\,(x,k)=r\,(x)+k$, $S_r(k)=S\diagdown S_q(k)=\{x\in S: q(x)<r(x)+k\}$. Denote a strategy, resulting from this partition by π_0.

For any set $S\subset X$ by definition of $\tau=\tau_S$ and definition of function $g(x,k)$ we have

$$\mathbb{E}_x^{\pi_0} g(Z_\tau) = \mathbb{E}_x^{\pi_0}\left[\sum_{n=0}^{\tau-1} c(Z_n) + I\left(Z_\tau\in S_q\right)q(Z_\tau) + I\left(Z_\tau \in S_r\right)\left(r(Z_\tau)+k\right)\right].$$

Since $\sum_{n=0}^{\tau-1} c(Z_n)$ does not depend on how stopping set S is partitioned into S_q and S_r, then, for all partitions π with the same set S, the expectations of the first sum are equal, $\mathbb{E}_x^{\pi}\left[\sum_{n=0}^{\tau-1} c(Z_n)\right] = \mathbb{E}_x^{\pi_0}\left[\sum_{n=0}^{\tau-1} c(Z_n)\right]$.

Also, partition π_0 of the set S differs from partition π by the fact, that π_0 uses the maximal reward, $q\,(x)$ or $r\,(x)+k$, therefore

$$\mathbb{E}_x^{\pi}\left[I\left(Z_\tau\in S_q^{\pi}\right)q(Z_\tau) + I\left(Z_\tau\in S_r^{\pi}\right)\left(r(Z_\tau)+k\right)\right] \leq$$

$$\leq \mathbb{E}_x^{\pi_0}\left[I\left(Z_\tau \in S_q^{\pi_0}\right)q(Z_\tau) + I\left(Z_\tau \in S_r^{\pi_0}\right)(r(Z_\tau)+k)\right].$$

As a result $R^\pi(x) + (1-Q^\pi(x))k \leq R^{\pi_0}(x) + (1-Q^{\pi_0}(x))k$. Finally,

$$\sup_{\pi:A(x)=c}[R^\pi(x) + (1-Q^\pi(x))k] = \sup_{\tau>0}\left[R^{\pi_0(\tau)}(x) + (1-Q^{\pi_0(\tau)}(x))k\right].$$

□

Note that equation (3.10) for the value function can be rewritten as

$$v(x,k) - r(x) = (q(x)-r(x)) \vee k \vee \sup_{\pi:A(x)=c}[R^\pi(x) - r(x) + (1-Q^\pi(x))k]. \tag{3.11}$$

Theorem 3.6. *The three modified indices defined for a general CQR model coincide, i.e.* $\alpha(x) = \tilde{h}(x) = w(x)$. *If* $\pi = (C, S_q, S_r)$ *is an optimal strategy in Problem 3.1, then it is also an optimal strategy in Problem 3.2 and set* $S = S_q \cup S_r$ *is an optimal stopping set* $S(k)$ *in OS Problem* $M(k)$ *for* $k = w(x)$.

Proof. A similar theorem was proved in Sonin [2008] for the case when $q(x) = -\infty$ and $r(x) = 0$ for all $x \in X$. Here we prove this theorem differently by using theorem 2.22. Given $x \in X$ let us introduce the set of indices $U = \{\pi : A(x) = c\} \cup \{\pi : A(x) \neq c\}$, where $\pi : A(x) \neq c$ is an index such that $a_0 = q(x) - r(x)$, $b_0 = 1$. Then, according to formulas (3.5), (3.6), and (3.11), three Problems 3.1-3.3 are represented as three abstract problems, therefore we can apply theorem 2.22 (Abstract Optimization Equality). □

3.3 Main Theorem for the Whittle Family of Optimal Stopping Models

In a previous section we defined Whittle family of models $M(k) = (X, P, c(x), g(x|k))$ and indices $w(x)$, $t(x)$ for this model. We established, that, according to the theorem 3.6, the value function at state s for CQR model, $h(s)$, is equal to index $w(s)$. Moreover, the optimal stopping set for model $M(k)$ when $k = w(s)$ is also an optimal strategy for CQR model. Therefore, instead of finding optimal strategy for CQR directly, we can find index $w(s)$ and optimal stopping set for model $M(w(s))$.

For a given value of k, the model $M(k)$ become a standard optimal stopping problem, therefore it is straightforward to obtain value functions $v(\cdot, k)$ by applying State Elimination (SE) algorithm.

Define function $G(x,k)$ as

$$G(x,k) = g(x,k) - [c(x) + Pg(x,k)].$$

For a fixed k, $G(x,k)$ has the same meaning, as $G(x)$ of SE algorithm. Negative value of $G(x,k)$ means that stopping at state x produces lower expected reward compared to making one step and stopping after that. Also, the equation (2.25) is valid for $G(x,k)$.

3.3.1 Whittle family with no quit action

First study the case when quit action is not allowed. The goal of this section is to develop intuition and main facts, applicable to more general case.

The removal of quit action can be achieved by setting $q = -\infty$. Then the terminal reward function becomes $g(x,k) = r(x) + k$, function $Pg(x,k)$ can be written in simplified form, $Pg(x,k) = \sum_{y \neq e} p(x,y)(r(y)+k)$. The equation for $G(x,k)$ becomes

$$G(x,k) = (1-\beta(x))\,k + r(x) - c(x) - Pr(x), \quad -\infty < k < \infty,$$

where $\beta(x)$ has the meaning of discount factor and defined as $\beta(x) = \sum_{y \neq e} p(x,y)$.

For each state x denote by $d(x)$ the value of k which makes $G(x,k) = 0$. The value of $d(x)$ can be found as:

$$d(x) = \frac{c(x) - r(x) + Pr(x)}{1 - \beta(x)}. \tag{3.12}$$

Since for any state x there is positive probability to go to an absorbing state e, the slope $\beta(x)$ satisfies inequality $0 < (1-\beta(x)) \leq 1$. Therefore, function $G(x,k)$ is a linear function with strictly positive slope.

Remark 3.7. The case when all restart rewards are zero, we obtain the problem, which was studied in Sonin [2008]: when $r(x) := 0$ for all $x \in X$, the function $G(x,k) = (1-\beta(x))\,k - c(x)$ and $d(x) = c(x) / (1-\beta(x))$.

Let $\pi^*(x,k)$ be the optimal strategy for the model $M(k)$ and $S(k)$ be the corresponding optimal stopping set. In other words, $S(k)$ is a mapping from $\mathbb{R}$ to set of all subsets of state space X. If state x belongs to the stopping set $S(k)$, the value function $v(x,k) = r(x) + k$. If state x belongs to $X \backslash S(k)$, then $v(x,k) \geq r(x) + k$, moreover, this inequality is strict unless it is indifferent for the value function whether to stop at this state or continue.

Define k_s and k_c as values of k, such that for all $x \in X$, $G(x,k_s) > 0$ and $G(x,k_c) < 0$. By proposition 2.8 positivity of all $G(x, k_s)$ means that the optimal stopping set coincides with state space, $S(k_s) = X$. From the other hand, all $G(x, k_c) < 0$, which means that it is optimal not to stop at all states and optimal stopping set is empty, $S(k_c) = \emptyset$.

Proposition 3.8. *For given $x \in X$, the optimal strategy $\pi^*(x, k)$ is to continue on $[-\infty, w(x)]$ and to stop on $[w(x), \infty]$. The index $w(x)$ is the only value when $G(x, k) = 0$.*

Proof. The proof is done through direct calculation of optimal strategy $\pi^*(x, k)$. Let us start from $k = k_s$. The optimal strategy for any $k > k_s$ is to stop at every state $S(k) = X$.

Functions $G(x, k)$ are linear with positive slope, therefore, if we continually decrease k, we will reach first $G(y, k) = 0$ for some $y \in X$. Denote this value of k by d^+. The optimal stopping set for $k < d^+$ does not contain state y because $G(y, k) < 0$ for any $k < d^+$. Therefore, $v(y, k) = r(x) + k$ for $k > d^+$ and $v(y, k) > r(x) + k$ for $k < d^+$. In other words we found $w(y)$ and have proven the proposition for state y.

The rest of state space $X \setminus \{y\}$ can be dealt with recursively. Construct a new model, $M_{\{y\}}(k)$ by eliminating state y from the state space and by applying state elimination equations (2.6), (2.10) to transition matrix P and reward function c. Since $G(y, d^+) = 0$, therefore, by the equation (2.16), the sign of functions $G(x, d^+)$ does not change. As a result, by elimination theorem 2.9, the optimal strategy for reduced model $M_{\{y\}}$ coincides with optimal strategy for the original model for $k < d^+$. □

The proof gives an algorithm to compute $w(x)$. Notice, that we do not know the order in which $w(x)$ are calculated, but once we have $w(s)$, we also obtain the solution for CQR problem: $h(s) = w(s)$ and $\pi^*(w(s))$ is the optimal strategy.

The algorithm is given in algorithm 3.1. The algorithm uses set notation because it is possible to have several states with $G(x, k) = 0$ for the same value of k. The elimination can be done one-by-one or by applying matrix equations. The complexity of single iteration of the algorithm is $O\left(n^2\right)$, the number of iterations, in general case, is $O(n)$, therefore the total complexity of the algorithm is $O\left(n^3\right)$.

3.3.2 Whittle model

Let us go back to the general case. Our goal is the same, we need to find value of index $w(s)$. The stop reward for the state x is piecewise linear function $g(x, k) = q(x) \vee (r(x) + k)$. The value $q(x) - r(x)$ is a threshold where $g(x, k)$ changes its value from $q(x)$ to $r(x) + k$. It is convenient

Algorithm 3.1 Finding $h(s)$ and optimal strategy for CQR problem with no quit

Input: CQR model with no quit $M(X,P,c,r)$, return state $s \in X$
Output: optimal strategy π for the model, value function $h(s)$ for state s
$C \leftarrow \emptyset$, $S_r \leftarrow X$ $\{C$ is a continue set, S_r is a restart to s set$\}$
while $w(s)$ not found **do**
  $d^+ \leftarrow \max(k : G(x,k)=0, x \in S_r)$ {use 3.12 to solve $G(x,k)=0$}
  $D \leftarrow \{x : G(x,d^+) \leq 0, x \in S_r\}$
  $w(x) = d^+$, for $x \in D$
  $S_r \leftarrow S_r \diagdown D$, $C \leftarrow C \cup D$
  $(P,c) \leftarrow$ update model: use algorithm 2.1 to eliminate D from (X,P,c)
  if $w(s)$ is found **then**
    set optimal strategy π as partition into continue set C ans restart to s set S_r
    $h(s) \leftarrow w(s)$
    return π and $h(s)$
  end if
end while

to define threshold value by $\gamma(x) = q(x) - r(x)$. Function $G(x,k)$ is linear function of $g(x,k)$, therefore it should be piecewise linear function too.

Define partial discount factor as a function of k as

$$\beta(x,k) = \sum_{\gamma(y)\leq k} p(x,y),$$

it plays important role in function $G(x,k)$. In particular:

Lemma 3.9. *For any fixed $x \in X$ function $G(x,k)$, as a function of k, is continuous, piecewise linear function. The slope of $G(x,k)$ is changing when $k = \gamma(y)$, moreover, the slope is*

- *negative, $-\beta(x,k)$, which is decreasing in k, if $k < \gamma(x)$, and*
- *positive, $1-\beta(x,i)$, which is increasing in k, if $k \geq \gamma(x)$.*

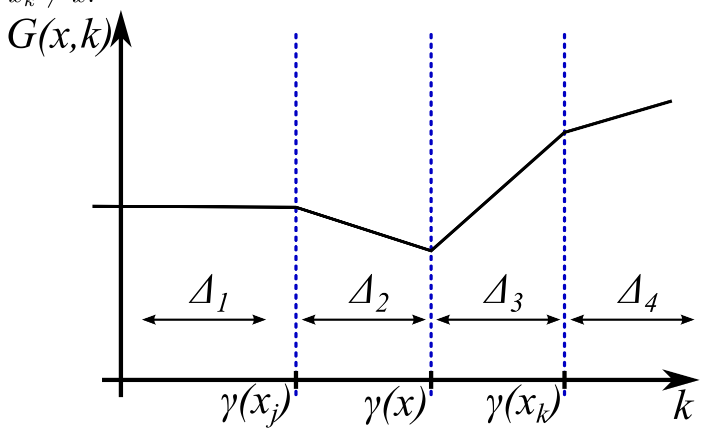
Proof. First, let us write $Pg(x,k)$,

$$\begin{aligned} Pg(x,k) &= \sum_{\gamma(y)>k} p(x,y)\,q(y) + \sum_{\gamma(y)\leq k} p(x,y)\,r(y) + k\sum_{\gamma(y)\leq k} p(x,y) \\ &= \sum_{\gamma(y)>k} p(x,y)\,q(y) + \sum_{\gamma(y)\leq k} p(x,y)\,r(y) + k\beta(x,k). \end{aligned}$$

Since $G(x,k) = g(x,k) - [c(x) + Pg(x,k)]$, the slope $G(x,k)$ is $-\beta(x,k)$ for $k < \gamma(x)$ and $1-\beta(x,i)$ for $k \geq \gamma(x)$. Lastly, since $\beta(x,k)$ is partial sum, conditional on k, it is only decreasing when k is decreasing. □

Corollary 3.10. *Function $G(x,k)$ is minimal when $k = \gamma(x)$.*

Figure 3.5: Graph of $G(x,k)$ for fixed state x, states x_j and x_k are some states from X, $x_j \neq x$, $x_k \neq x$.



Corollary 3.11. *For all $x \in X$, function $G(x,k)$ changes slope at points $\gamma(y)$, $y \in X$. In other words, for all states x, functions $G(x,k)$ change slope at the same set of values of k.*

It is convenient to introduce intervals Δ_i, $i = 1 \ldots M$ as set of ordered intervals, where all functions are linear. The first interval has form $\Delta_1 = (-\infty, \min_x \gamma(x)]$, last interval has form $\Delta_M = [\max_x \gamma(x), \infty)$, all other intervals contain pairs of values $\gamma(x)$. In general, we can have at most $|X| + 1$ intervals, and less intervals, if not all $\gamma(x)$ are different.

The graph of $G(x,k)$ as a function of k is given in Figure 3.5. The graphs shows intervals Δ_i, values of $\gamma(y)$ and illustrates main properties of $G(x,k)$.

It is useful to write down full equations for $G(x,k)$, these equations are used in the algorithm later, if $k < \gamma(x)$:

$$G(x,k) = -\beta(x,k)\, k + q(x) - c(x) - \sum_{\gamma(y)>k} p(x,y)\, q(y) - \sum_{\gamma(y)\leq k} p(x,y)\, r(y), \tag{3.13}$$

and, if $k > \gamma(x)$

$$G(x,k) = (1 - \beta(x,k))\, k + r(x) - c(x) - \sum_{\gamma(y)>k} p(x,y)\, q(y) - \sum_{\gamma(y)\leq k} p(x,y)\, r(y). \tag{3.14}$$

The proof for the main theorem needs introduction of several more definitions. Let k_0 be some fixed value of k. The model $M(k_0)$ is the standard optimal stopping problem, therefore it has optimal stopping set $S(k_0) = S_q(k_0) \cup S_r(k_0)$. Since we are only considering model at point k_0, let us just write S, S_q, and S_r. Let the set C be the complement of S, $C = X \backslash S$.

If $x \in S$, then

- $x \in S_q$, if $v(x, k_0) = q(x) \Longleftrightarrow \gamma(x) \leq k_0$, for $k = \gamma(x)$ we choose S_q.
- $x \in S_r$ if $v(x, k_0) = r(x) + k_0 \Longleftrightarrow \gamma(x) > k_0$.

By $M_S(k)$ define result of SE algorithm, performed on Whittle model $M(k)$. In other words, for the model $M_S(k)$, the transition matrix P_S and continue reward c_S are the result of application of elimination equations (2.18) and (2.20) to P and c with S being set to eliminate.

Define by $G_S(x, k)$ function G for model $M_S(k)$.

Remark 3.12. The function $G_S(x, k) \geq 0$ for $x \in S$.

Remark 3.13. It follows from the Theorem 2.9,that the set S is no longer an optimal stopping set for the model $M(k)$, $k < k_0$, when at least one of $G_S(x, k)$ changes sign.

We are interested in the largest value $k < k_0$ such that the function $G_S(x, k)$ changes its sign for some x. The algorithm behavior depends on the way, the sign is changed, in order to accommodate this difference, we can define values d^+ and d^- as

$$d^+ = \inf\{k < k_0 : x \in S_r, G_S(x, k) > 0\},$$

and

$$d^- = \sup\{k < k_0 : x \in C, G_S(x, k) < 0\},$$

in cases when d^+ (d^-) does not exist is convenient to assign value of $-\infty$ to d^+ (d^-).

Theorem 3.14 (General step of iteration)**.** *Suppose that in a Whittle model $M_S(k)$ at least one of the following inequalities is true:$d^+ > -\infty$, $d^- > -\infty$. Then*

1. *If $d^+ > d^-$, then $w(x) = d^+$ for all x such that $G_S(x, d^+) = 0$.*
2. *If $d^- > d^+$, then $t(x) = d^-$ for all for x such that $G_S(x, d^-) = 0$*
3. *If $d^- = d^+$, then $w(x) = d^+$ for all $x \in S_r$ such that $G_S(x, d^+) = 0$, and $t(x) = d^-$ for all for $x \in C$ such that $G_S(x, d^-) = 0$.*

Proof. First, define the set D as the set where we found first change of sign for $G_S(x, k)$, $D = \{x : G_S(x, \max(d^+, d^-)) = 0\}$. Now let us prove each statement one by one.

Case $d^+ > d^-$. Definition of d^+, inequality $d^+ > d^-$ and Proposition 2.12 imply that $S = S_q \cup S_r$ is an optimal stopping set for model $M(k)$ for all $k \in [d^+, k_0)$, and for any small $\varepsilon > 0$

$$G_S(x, d^+) = 0,\ x \in D,\ G_S(x, d^+ - \varepsilon) < 0, x \in C \cup D,\ G_S(x, d^+ - \varepsilon) > 0,\ x \in S_r \diagdown D.$$

These inequalities, definitions of S_q and S_r and Proposition 2.12 immediately imply that $w(x) = d^+$ for all $x \in D$ and there is no other values of $w(x)$ or $t(x)$ on the interval $[d^+, k_0)$.

In addition, since $G_S(x, d^+ - \varepsilon) < 0$ for $x \in C \cup D$, the optimal stopping set $S(d^+ - \varepsilon) = S(k_0) \backslash D$. Since it is irrelevant, whether to stop or continue for $x \in D$ at $k = d^+$, we can set $S(d^+) = S(d^+ - \varepsilon)$.

Create a new model $M_{S(d^+)}$ by using applying elimination procedure to M_S with eliminated set D. Because $G_S(x, d^+) = 0$ for all $x \in D$, then by equation (2.25) the sign of all $G_{S(d^+)}(x, d^+)$ coincides with sign of $G_S(x, d^+)$. By Proposition 2.8 this means that $S(d^+)$ is an optimal stopping set for model $M(k)$ at $k = d^+ - \varepsilon$.

Case $d^- < d^+$. This part is very similar to the previous one, the difference is that set D will be added to the stopping set S, and for this case we obtain $t(x)$. Definition of d^+, inequality $d^+ > d^-$ and Proposition 2.12 imply that $S = S_q \cup S_r$ is an optimal stopping set for model $M(k)$ for all $k \in [d^-, k_0)$, and for any small $\varepsilon > 0$

$$G_S(x, d^-) = 0,\ x \in D,\ G_S(x, d^- - \varepsilon) < 0, x \in C \backslash D,\ G_S(x, d^- - \varepsilon) > 0,\ x \in S_q \cup D.$$

These inequalities, definitions of S_q and S_r and Proposition 2.12 immediately imply that $t(x) = d^-$ for all $x \in D$ and there is no other values of $w(x)$ or $t(x)$ on the interval $[d^-, k_0)$.

The rest is analogous to the previous case, using the same derivation, the set $S(d^-) = S(k_0) \cup D$ is the optimal stopping set for for model $M(k)$ at $k = d^+ - \varepsilon$.

Case $d^- = d^+$. This case is a combination of previous two cases, the proof is exactly the same: define D^+ and D^- as sets, which lead to d^+ and d^-, then set $w(x) = d^+$ for $x \in D^+$, set $t(x) = d^-$ for $x \in D^-$, then define $S(d^+) = (S(k_0) \backslash D^+) \cup D^-$. Using the same derivation, $S(d^+)$ is the optimal stopping set for for model $M(k)$ at $k = d^+ - \varepsilon$. □

Corollary 3.15. *Similarly to CQR model with no quit, for large enough k_0, the optimal strategy for general case is to stop at all $x \in X$ and obtain reward $g(x, k_0) = r(x) + k$, in other words, $S(k_0) = X$, moreover $S_r = X$. Using this large k_0 as initial point, we can use proof of theorem as a basis of algorithm to find $w(s)$ together with optimal strategy $\pi^*(w(s))$.*

Remark 3.16. It could happen that for some x the $G_S(x,k)$ is never zero. Then, $x \in S(k)$ for all values of k, in particular, for $k = \gamma(x)$, the value function is $v(x, \gamma(x)) = r(x) + \gamma(x)$, and $v(x, \gamma(x) - \varepsilon) = q(x)$. Therefore, for such states x, $w(x) = \gamma(x)$.

Corollary 3.17. *From previous remark and Lemma 3.9 and the fact that elimination does not change sign of $G_S(x,k)$, if $w(x)$ is not found when $k = \gamma(x)$, we immediately have that $w(x) = t(x) = \gamma(x)$. The meaning of this special case is that optimal strategy for this state, as a function of k is always to stop—and obtain $q(x)$ or $r(x) + k$. The graph of value function $v(x,k)$ for this case is given in Figure 3.3.*

Remark 3.18. Previous corollary means that $w(x)$ exists for each state x. It might happen that $t(x)$ does not exists, it means that for this state x, $v(x,k) > q(x)$ for all k. The graph of value function $v(x,k)$ for this case is given in Figure 3.4.

3.4 Algorithm

Theorem 3.14 and Corollary 3.17 serve as foundation of the algorithm. Indeed, for large enough k, all $G_S(x,k)$ are positive, therefore, function $G_S(x,k)$ should either be equal to zero for some k or stay positive. Therefore, Theorem 3.14 and Corollary 3.17 cover all possible cases and allow us to compute $w(x)$ for all x.

Corollary 3.15 providing starting value k_0 and optimal strategy for this k_0, i.e., $S_r = S(k_0) = X$. The results of previous section allow calculation algorithm to directly track sets $S_r(k)$, $S_q(k)$, and C in the following way

- we begin with $S_r = X$,
- elimination can only happen to $x \in S_r$, it means that x moves from set S_r to set C,
- insertion can only happen if $x \in C$, it means that x moves from set C to set S_q,
- application of Corollary 3.17 to x also means that x moves from set S_r to S_q.

Remark 3.19. By the Abstract Optimization Theorem, Theorem 2.22, the sets S_r, S_q, and C at $k = w(s)$ provide optimal strategy for CQR problem. Namely, the optimal strategy is to continue if $x \in C$, restart to s if $x \in S_r$, and to quit if $x \in S_q$.

Remark 3.20. The values of $\gamma(x)$ do not change with elimination or insertion, as a result, the intervals Δ_i stay the same with elimination or insertion.

Theorem 3.14 provides recursive way to compute $w(x)$ and $t(x)$, moreover, after $w(x)$ or $t(x)$ is found, we need to recompute all $G_S(x,k)$. Therefore it is more convenient to consider k in intervals Δ_i one by one, starting from Δ_M. Each $G_S(x,k)$ is linear on intervals Δ_i, which simplifies solution of equation $G_S(x,k) = 0$.

The algorithm works in two directions. From the one hand it tracks change in the function $G_S(x,k)$ caused by moving k from one interval Δ_i to another, from the other hand, evolves model $M_S(k)$, which changes when optimal stopping set is being changed.

The algorithm is given in Algorithm 3.2. The complexity of the algorithm is $O\left(n^3\right)$, where $n = |X|$.

Algorithm 3.2 Finding $h(s)$, $t(s)$ in CQR problem

Input: CQR model $M(X, P, c, q, r)$, return state $s \in X$
Output: optimal strategy π for the model, value function $h(s)$ for state s

```
S_q ← ∅, C ← ∅, S_r ← X
k_0 ← ∞
γ(x) ← q(x) − r(x), x ∈ X
Δ_i ← intervals based on ordered set of −∞, {γ(x)}, ∞
M ← |Δ|
for i = M to 1 do
  repeat
    d⁺ ← max(k : G_S(x,k) = 0, x ∈ S_r, k < k_0, k ∈ Δ_i; −∞) {set d⁺ to −∞ if it does not exists}
    d⁻ ← max(k : G_S(x,k) = 0, x ∈ C, k < k_0, k ∈ Δ_i; −∞) {set d⁻ to −∞ if it does not exists}
    if d⁺ > −∞ or d⁻ > −∞ then
      D⁺ ← ∅
      D⁻ ← ∅
      if d⁺ > d⁻ or d⁺ = d⁻ then
        D⁺ ← {x : G_S(x, d⁺) = 0, x ∈ S_r}
        w(x) ← d⁺, x ∈ D⁺
      end if
      if d⁻ > d⁺ or d⁺ = d⁻ then
        D⁻ ← {x : G_S(x, d⁻) = 0, x ∈ C}
        t(x) ← d⁻, x ∈ D⁻
      end if
      (P, c) ← use equations (2.27)-(2.30) to eliminate set D⁺ and insert set D⁻
      S_r ← S_r\D⁺, S_q ← S_q ∪ D⁻, C ← (C ∪ D⁺)\D⁻
      k_0 ← max(d⁺, d⁻)
      if w(s) is found then
        set optimal strategy π based on partition into sets S_q, C, and S_r
        h(s) ← w(s)
        return π and h(s)
      end if
    end if
  until there was elimination or insertion on interval Δ_i
  k_0 ← min(Δ_i) {k_0 is equal to the leftmost point of interval Δ_i}
  {check condition of Corollary 3.17: find non-eliminated states for which G_S(x,k) is minimal}
  if γ(x) = k_0 for some x ∈ S_r then
    D ← {x : γ(x) = k_0, x ∈ S_r} {apply Corollary 3.17}
    w(x) ← k_0, t(x) ← k_0, x ∈ D
    S_r ← S_r\D, S_q ← S_q ∪ D
    if w(s) is found then
      set optimal strategy π based on partition into sets S_q, C, and S_r
      h(s) ← w(s)
      return π and h(s)
    end if
  end if
end for
```

CHAPTER 4: ALGORITHM ANALYSIS

4.1 Complexity

This section is devoted to estimation Let us find the The algorithm consists of two main parts: find d^{+}, d^{-} and elimination/insertion.

The elimination and insertion, done by equations (2.27)-(2.30). The elimination/insertion on W_D have exactly the same complexity. Let us compute complexity to eliminate single state z, it involves

- one addition and two multiplications to compute new value of $w(x, y)$, except for $w(x, z)$,
- one multiplication to compute new value for column $w(\cdot, z)$,
- one addition and one multiplication to compute new value of $c(y)$, except for $c(z)$,
- one multiplication to compute new value of $c(z)$.

In total, elimination/insertion step requires $n(n-1)+n-1 = n^2-1$ additions, let us round number of additions to n^2, and $2n(n-1)+n+n = 2n^2$ multiplications.

Corollary 4.1. *Complexity to eliminate or insert one state is $2n^2$ multiplications and n^2 additions.*

Another time consuming step of the algorithm is to find solution for linear equation $G_S(x, k) = 0$ for the given interval Δ_i. The equations for this step are equations (3.13)-(3.14). For given x, only one of these equations is used, the complexity for these equations is the same

- finding slope of $G_S(x, k)$ requires partial summation of $p(x, y)$, consider the worst case, then we have n additions for this step,
- finding intercept is done by summation of $p(x, y) f(y)$, where is $f(y)$ can be $q(y)$ or $r(y)$, depending on the test $\gamma(y) > k$, therefore, intercept requires n multiplications and n additions.

Therefore, finding d^{+} or d^{-} for each state x, requires $2n$ multiplications and n additions. If $x \in S_q$, then it is impossible for $G_S(x, k)$ to become zero again, therefore we do not have to solve for $G_S(x, k) = 0$. However, we do not know in advance how big S_q will be, and, it is possible for S_q to be empty at the end of algorithm, therefore, assume worst case scenario, and consider the complexity of this step to be $2n^2$ multiplications and n^2 additions. It also involves n divisions, which we ignore.

Corollary 4.2. *Complexity to find d^+ and d^- is $2n^2$ multiplications and n^2 additions.*

Remark 4.3. It is possible to avoid recomputing equations (3.13)-(3.14) completely, when current interval changes from Δ_i to Δ_{i-1}: the change of current interval changes result of only one comparison $\gamma(y) > k$, therefore, we can only apply this change to recompute slope and intercept of $G_S(x, k)$.

In general, we do not know when algorithm finds $w(s)$, in addition, it is hard to assume what would be the distribution of optimal strategies for given problem. Therefore, it makes sense to consider worst case scenario, when we have to perform the most amount of work possible.

Worst case scenario means that $w(s)$ is found at the last interval after eliminating and inserting all other states. Therefore, we have $n-1$ insertions and $n-1$ eliminations, which amounts to $4n^3$multiplications and $2n^3$ additions. Moreover, for each elimination or insertion, we have to recompute d^+ and d^-, this gives another $4n^3$multiplications and $2n^3$ additions.

Recomputation of $d^+$and d^- when current interval changes from Δ_i to Δ_{i-1} has complexity of $2n(n+1)$ additions and $n(n+1)$ divisions, which we ignore.

Remark 4.4. The constant before n^2 term for number of multiplications and additions is less than 4, it is implementation dependent and can be ignored.

Corollary 4.5. *The worst case complexity for the CQR algorithm is $8n^3$ multiplications and $4n^3$ additions.*

4.2 Linear Programming Formulation

Let us develop some alternate way to calculate optimal strategy for CQR model. The generic, yet, still efficient alternate way is to find optimal strategy for models, maximizing total discounted expected reward for infinite time horizon, is to use linear programming approach, for more details see Puterman [2005].

Consider model with multiple restart points,

$$M = (X, B, P, A(x), c(x), q(x), r_j(x), j = 1, 2, ..., m, \beta(x), |X| = n).$$

We assume that this problem is well-defined and has finite solution. For example, the problem in this formulation could have infinite solution in case when restart loops with positive reward exist.

Since we can easily obtain the value function given strategy π, the simplest possible approach could be the total enumeration of all strategies. Unfortunately, the complexity of this approach is exponential. At each state the number of possible actions is $m+1$ for states to which restart is possible and $m+2$ for states, not used as restart destinations. Therefore, the number of possible strategies is $(m+2)^{n-m}(m+1)^m$. The complexity to compute value function is $O\left(n^3\right)$, which might be possible to decrease to $O\left(n^2\right)$ if we consider only changes in strategies, for estimation of the complexity, we can safely assume that it is $O\left(n^2\right)$. Since this approach is inefficient for number of states greater than 20, we need to look into more efficient methods.

The solution of CQR problem is the minimal solution of corresponding Bellman equation, therefore it can be written as

$$
\begin{aligned}
& \min \sum v_i, \\
& \text{subject to} \\
& \text{(continue)} \quad (I-P)\,v \quad \geq c, \\
& \text{(quit)} \quad Iv \quad \geq q, \\
& \text{(restart)} \quad v_i - v_j \quad \geq r_j\,(i)\,,\ i \neq j,
\end{aligned}
\tag{4.1}
$$

where $i = 1..n$, $j = 1..m$. Each constraint corresponds to the appropriate action

- continue constraint means that value v_i should be greater or equal than continue reward plus value, obtained from $(Pv)_i$, i.e. $v_i \geq c_i + (Pv)_i$, which can be written in a matrix form as $(I-P)\,v \geq c$,
- quit constraint means that each value v_i should be greater or equal than the quit reward c_i,
- restart constraint means for every restart state j, $j = 1...m$, and every state i, $i \neq j$, v_i is greater than restart reward and value, obtained at restart point, i.e. $v_i \geq r_j\,(i) + v_j$.

There are n continue constraints, n quit constraints. Each restart constraint does not include restart point which gives total of $m\,(n-1)$ restart constraints. Total number of constraints is equal to $2n + (n-1)\,m$, for single restart point, the number of constraints is $3n-1$.

Change the inequality sign, and, in order to simplify further notation, let us write the linear programming problem in the matrix form, which also serves as definition of vector b and matrix A

$$
\begin{aligned}
& \min \sum v_i, \\
& \text{subject to}
\end{aligned}
$$

$$-Av \qquad \leq -b, \tag{4.2}$$

here

- b_k corresponds for continue, quit, and restart costs,
- $A = (a_{kl})$ is a matrix corresponding to left side of constraints,
- $l = 1...n$, $k = 1..2n + (n-1)m$.

The dual problem can be written as

$$\max \sum y_k b_k,$$

subject to

$$\begin{aligned} \textstyle\sum_k y_k a_{kl} \qquad &= 1, \\ y_k \qquad &\geq 0. \end{aligned} \tag{4.3}$$

The dual problem is written in the standard form and can be used as an input to the simplex algorithm without any further modifications.

The quit constraint in primal problem is $Iv \geq q$. The columns in dual problem, corresponding to the quit constraint are very convenient choice for initial basis for the simplex algorithm.

Since the dual problem has n constraints, its optimal solution y^* contains exactly n non-zero values. By the theorem on complementary slackness, knowing, which optimal values are non-zero, or form the basis of the simplex algorithm, gives us the optimal strategy for the primal problem.

The complexity to solve the problem in linear programming formulation is not less, than our algorithm, simply because solution of linear programming problem, at least implicitly, involves matrix inversion of the size $n \times n$, which is already $O\left(n^3\right)$. Moreover, our algorithm can be started at arbitrary value of k_0 and has transparent probabilistic meaning.

BIBLIOGRAPHY

Eric V. Denardo, Uriel G. Rothblum, and Ludo Van der Heyden. Index policies for stochastic search in a forest with an application to r&d project management. *Math. Oper. Res.*, 29:162–181, February 2004. ISSN 0364-765X. doi: http://dx.doi.org/10.1287/moor.1030.0072.

E. A. Feinberg and A. Shwartz. *Handbook of Markov decision processes*. International Series in Operations Research & Management Science, 40. Kluwer Academic Publishers, Boston, MA, 2002. ISBN 0-7923-7459-2. Methods and applications.

W. K. Grassmann, M. I. Taksar, and D. P. Heyman. Regenerative analysis and steady state distributions for markov chains. *Operations Research*, 33:1107–1116, 1985. doi: 10.1287/opre.33.5.1107.

A. Irle. On the best choice problem with random population size. *Mathematical Methods of Operations Research*, 24:177–190, 1980. doi: 10.1007/BF01919245.

M. N. Katehakis and A. F. Veinott. The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12:262–268, 1987. doi: 10.1287/moor.12.2.262.

J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Finite Markov Chains*. Springer, Princeton, NJ, 1976.

L. G. Mitten. An analytic solution to the least cost testing sequence problem. *J. Ind. Eng.*, 11:17, 1960.

José Niño-Mora. A (2/3)n3 Fast-Pivoting Algorithm for the Gittins Index and Optimal Stopping of a Markov Chain. *Informs Journal on Computing*, 19:596–606, 2007. doi: 10.1287/ijoc.1060.0206.

M. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., New York, 2005.

T. J. Sheskin. Technical note–a markov chain partitioning algorithm for computing steady state probabilities. *Operations Research*, 33:228–235, 1985. doi: 10.1287/opre.33.1.228.

T. J. Sheskin. State reduction in a markov decision process. *Internat. J. Math. Ed. Sci. Tech.*, 30 (2):167–185, 1999.

I. Sonin. Two simple theorems in the problems of optimal stopping. In *Proc. INFORMS Appl. Prob. Conf.*, Atlanta, Georgia, 1995.

I. Sonin. The elimination algorithm for the problem of optimal stopping. *Mathematical methods of operations research*, 49(1):111–123, 1999a. doi: 10.1007/s001860050016.

I. Sonin. The state reduction and related algorithms and their applications to the study of markov chains, graph theory, and the optimal stopping problem. *Advances in Mathematics*, 145:159–188, 1999b. doi: 10.1006/aima.1998.1813.

I. Sonin. The optimal stopping of a markov chain and recursive solution of poisson and bellman equations. In Yuri Kabanov, Robert Liptser, and Jordan Stoyanov, editors, *From Stochastic Calculus to Mathematical Finance. The Shiryaev Festschrift*, pages 609–621. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-30788-4.

I. Sonin. A generalized gittins index for a markov chain and its recursive calculation. *Statistics & Probability Letters*, 78(12):1526 – 1533, 2008. ISSN 0167-7152. doi: DOI: 10.1016/j.spl.2008.01.049.

I. Sonin. Optimal stopping of markov chain and three abstract optimization problems. to appear in Stochastics, 2011.

I. Sonin and E. Presman. The problem of best choice in the case of a random number of objects. *Theory Probab. Appl.*, 17:695–706, 1972.

I. Sonin and J. Thornton. Recursive algorithm for the fundamental/group inverse matrix of a markov chain from an explicit formula. *Siam Journal on Matrix Analysis and Applications*, 23(1):209–224, 2001. doi: 10.1137/S0895479899351234.

John N. Tsitsiklis. A short proof of the gittins index theorem. *Annals of Applied Probability*, 4: 194–199, 1994. doi: 10.1214/aoap/1177005207.

P. Whittle. Multi-armed bandits and the gittins index. *J. Roy. Statist. Soc. Ser. B*, 42(2):143–149, 1980.

APPENDIX A: SAMPLE CALCULATION

The goal of this appendix is to show how algorithm works for both cases, considered in the text, case with no quit action, and general case. Both sample calculations are performed on a sample chain with 5 states and variable discount factor. The transition matrix for both cases does not correspond to the full graph, this done intentionally in order to show how elimination and insertion change transition matrix. Transitions are shown on Figure

All probabilities in transition matrix, rewards, and calculated values are shown rounded to two digits after decimal point.

A.1 Sample calculation for case with no quit action

The transition matrix and costs are given in Table A.1. First we need to introduce absorbing state e and apply discount factor to obtain transition matrix, required for the algorithm. The graph of $G(x,k)$ for step 1 is shown in Figure A.2.

The calculation in total takes 3 steps, the elimination step is performed twice. The optimal strategy for CQR problem is determined by the optimal strategy for Whittle family $M(k)$ at point $w(s)$, which is to continue at states $\{b,d,s\}$, restart at states $\{a,c\}$. Note again, that algorithm works on a family of OS problems $M(k)$, however it finds solution for CQR problem. The correctness of this solution was verified by running simplex method for the dual formulation of corresponding linear programming problem.

A.2 Sample calculation for general case

Let us add quit action to the problem. The $w(s)$ is found on step 6 of the algorithm. Note, that optimal strategy for s is continue, if we were to find $t(s) = w(s)$, then the optimal strategy would

Figure A.1: Graph for the transition matrix in CQR sample calculation

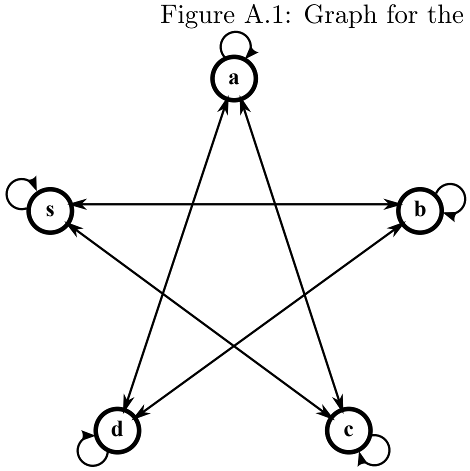


Table A.1: Sample data for CQR problem with no quit action

Original transition matrix							Rewards	
	a	b	c	d	s	$\beta(x)$	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$
a	0.3	0	0.2	0.5	0	0.7	1	0
b	0	0.3	0	0.4	0.3	0.7	6	1
c	0.2	0	0.2		0.6	0.7	1	4
d	0.7	0.1	0	0.2	0	0.5	1	6
s	0	0.1	0.8	0	0.1	0.3	1	0

Figure A.2: CQR problem with no quit action. Graph of $G(x,k)$ for step 1. The green thick line on the right in the middle correspond to the maximal value of $d^+(x)$.

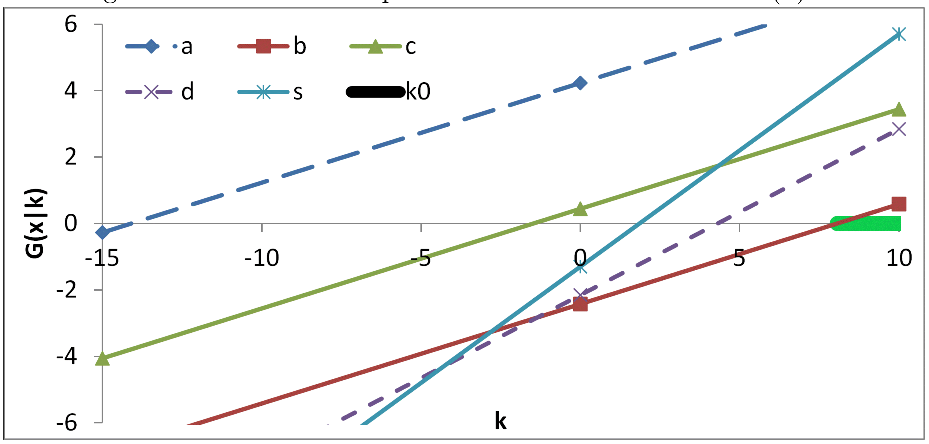


Table A.2: CQR problem with no quit action, step 1. This is the first step of calculation, k_0 is set to ∞, $S_r = X$, no states are eliminated. The value $d^+(x)$ is maximal for the state b, which means that we need to eliminate state b for the next step, also we found $w(b) = 8.07$.

Transition matrix P, state e not shown						Rewards		Calculated values		
	a	b	c	d	s	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$d^+(x)$	$w(x)$	$\pi(\max d^+(x))$
a	0.21	0	0.14	0.35	0	1	0	-14.10		restart
b	0	0.21	0	0.28	0.21	6	1	8.07	8.07	continue
c	0.14	0	0.14	0	0.42	1	4	-1.47		restart
d	0.49	0.07	0	0.14	0	1	6	4.30		restart
s	0	0.03	0.24	0	0.21	0	0	1.86		restart

Table A.3: CQR problem with no quit action, step 2. Set k_0 to the last found $w(x)$, $k_0 = 8.07$. State b is eliminated and is no longer in consideration. The matrix shown in the output is no longer transition matrix, transition matrix can be obtained by setting values in row b to zero. The value $d^+(x)$ is maximal for the state d, which means that we need to eliminate state d for the next step, also we found $w(d) = 4.68$.

Matrix W, eliminated states: b						Rewards		Calculated values		
	a	b	c	d	s	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$d^+(x)$	$w(x)$	$\pi(\max d^+(x))$
a	0.21	0.00	0.14	0.35	0.00	1.00	0	-14.10		restart
b	0.00	0.27	0.00	0.35	0.27	7.59	1		8.07	continue
c	0.14	0.00	0.14	0.00	0.42	1.00	4	-1.47		restart
d	0.35	0.06	0.00	0.12	0.01	1.38	6	4.68	4.68	continue
s	0.00	0.04	0.24	0.01	0.04	1.23	0	2.06		restart

Table A.4: CQR problem with no quit action, step 3. Set k_0 to the last found $w(x)$, $k_0 = 4.68$. Found $w(s)$ and optimal strategy: continue at states $\{b, d, s\}$, restart at states $\{a, c\}$

Matrix W, eliminated states: b, d						Rewards		Calculated values		
	a	b	c	d	s	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$d^+(x)$	$w(x)$	$\pi(\max d^+(x))$
a	0.35	0.03	0.14	0.40	0.01	1.55	0	-6.46		restart
b	0.14	0.29	0.00	0.40	0.27	8.15	1		8.07	continue
c	0.14	0.00	0.14	0.00	0.42	1.00	4	-1.47		restart
d	0.40	0.07	0.00	0.13	0.02	1.56	6		4.68	continue
s	0.00	0.04	0.24	0.01	0.04	1.24	0	2.09	2.09	continue

Figure A.3: CQR problem with no quit action. Graph of $G(x, k)$ for step 3. The black line labeled k0 correspond to the values k, already covered by the algorithm. The green thick line on the right in the middle correspond to the maximal value of $d^+(x)$.

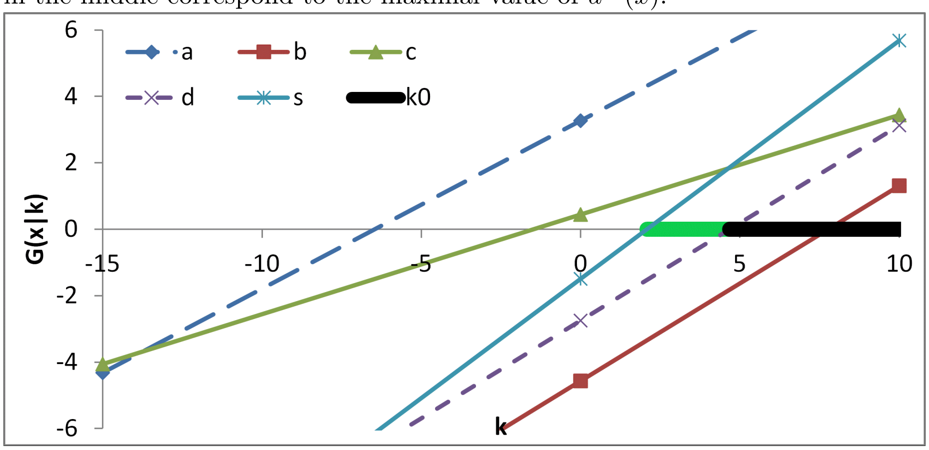
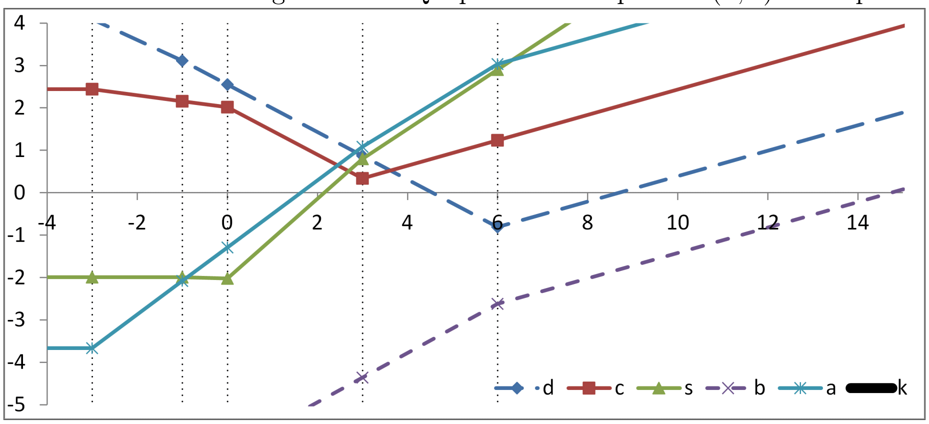


Table A.5: Sample data for CQR problem

Original transition matrix							Rewards		
	a	b	c	d	s	$\beta(x)$	$\mathbf{c(x)}$	$\mathbf{q(x)}$	$\mathbf{r(x)}$
a	0.3	0	0.2	0.5	0	0.7	1	0	3
b	0	0.3	0	0.4	0.3	0.7	6	1	2
c	0.2	0	0.2	0	0.6	0.7	1	4	1
d	0.7	0.1	0	0.2	0	0.7	1	6	0
s	0	0.1	0.8	0	0.1	0.3	1	0	0

Figure A.4: CQR problem. Graph of $G(x,k)$ for step 1.



be to quit at state s.

Optimal strategy for the CQR problem: continue at states $\{b, s\}$, quit at states $\{c, d\}$, restart at states $\{a\}$. The correctness of this solution was verified by running simplex method for the dual formulation of corresponding linear programming problem.

Table A.6: CQR problem, step 1. Note, that states are reordered in order of decreasing $\gamma(x)$. This is the first step of calculation, k_0 is set to ∞, working interval $\Delta_i = [6, \infty)$, $S_r = X$, no states are eliminated. Found $w(b) = 14.73$, state b is eliminated for step 2.

Matrix W						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c(x)}$	$\mathbf{q(x)}$	$\mathbf{r(x)}$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.21	0	0.14	0.35	0	1	0	3	8.70			restart
c	0	0.21	0	0.28	0.21	6	1	2				restart
s	0.14	0	0.14	0	0.42	1	4	1				restart
b	0.49	0.07	0	0.14	0	1	6	0	14.73		14.73	continue
a	0	0.03	0.24	0	0.21	0	0	0				restart

Table A.7: CQR problem, step 2. Value k_0 is set to $k_0 = 14.73$, working interval $\Delta_i = [6, \infty)$. Found $w(d) = 9.19$. State d is eliminated for step 3. Note, that these 2 steps are performed exactly in the same way, as in case with no quit.

Matrix W, eliminated states: $\{b\}$						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$\mathbf{r}(\mathbf{x})$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.16	0.00	0.02	0.09	0.49	1.53	0	3	9.19		9.19	continue
c	0.00	0.14	0.42	0.00	0.14	1.00	1	2	1.87			restart
s	0.01	0.24	0.04	0.04	0.00	1.23	4	1	2.06			restart
b	0.35	0.00	0.27	0.27	0.00	7.59	6	0	14.73		14.73	continue
a	0.35	0.14	0.00	0.00	0.21	1.00	0	0	-4.10			restart

Table A.8: CQR problem, step 3. Value k_0 is set to $k_0 = 9.19$, working interval $\Delta_i = [6, \infty)$. Not found any $d^{+/-}$ on working interval. Proceeding to the next interval $\Delta_i = [3, 6]$.

Matrix W, eliminated states: $\{b, d\}$						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$\mathbf{r}(\mathbf{x})$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.20	0.00	0.02	0.11	0.59	1.83	0	3			9.19	continue
c	0.00	0.14	0.42	0.00	0.14	1.00	1	2				restart
s	0.01	0.24	0.04	0.04	0.01	1.25	4	1				restart
b	0.42	0.00	0.27	0.30	0.21	8.24	6	0			14.73	continue
a	0.42	0.14	0.01	0.04	0.42	1.64	0	0				restart

Table A.9: CQR problem, step 4. Value k_0 is set to $k_0 = 6$, working interval $\Delta_i = [3, 6]$. For the state $G(d, k) = 0$ for $k = 3.95$, since state d is eliminated, it mean that we found $d^- = 3.95$; $t(d) = 3.95$ and state d should be inserted back.

Matrix W, eliminated states: $\{b, d\}$						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c}(\mathbf{x})$	$\mathbf{q}(\mathbf{x})$	$\mathbf{r}(\mathbf{x})$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.20	0.00	0.02	0.11	0.59	1.83	0	3	3.95	3.95	9.19	quit
c	0.00	0.14	0.42	0.00	0.14	1.00	1	2				restart
s	0.01	0.24	0.04	0.04	0.01	1.25	4	1				restart
b	0.42	0.00	0.27	0.30	0.21	8.24	6	0	13.25		14.73	continue
a	0.42	0.14	0.01	0.04	0.42	1.64	0	0				restart

Figure A.5: CQR problem. Graph of $G(x, k)$ for step 4. Think black line starts at value $k_0 = 6$. Eliminated states are b and d.

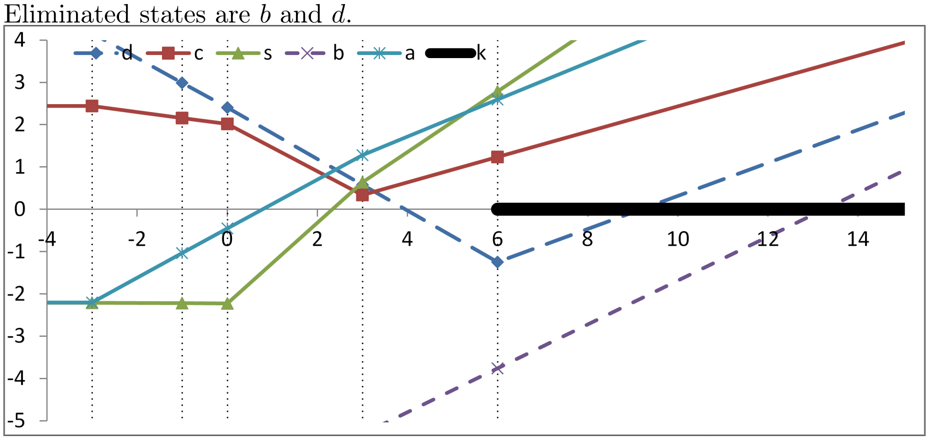


Table A.10: CQR problem, step 5. Value k_0 is set to $k_0 = 3.95$, working interval $\Delta_i = [3, 6]$. Not found any $d^{+/-}$ on working interval. Since $\gamma(c) = 3$ and we reached $k_0 = 0$, we found $t(c) = w(c) = 3$, change optimal strategy for c to quit. Proceed to the next interval $\Delta_i = [0, 3]$.

Matrix W, eliminated states: $\{b\}$						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c(x)}$	$\mathbf{q(x)}$	$\mathbf{r(x)}$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.16	0.00	0.02	0.09	0.49	1.53	0	3		3.95	9.19	quit
c	0.00	0.14	0.42	0.00	0.14	1.00	1	2		3	3	quit
s	0.01	0.24	0.04	0.04	0.00	1.23	4	1				restart
b	0.35	0.00	0.27	0.27	0.00	7.59	6	0			14.73	continue
a	0.35	0.14	0.00	0.00	0.21	1.00	0	0				restart

Figure A.6: CQR problem. Graph of $G(x, k)$ for step 5. Think black line starts at value $k_0 = 3.95$. Note, that $G(x, k)$ for state c is minimal at $k = 3$.

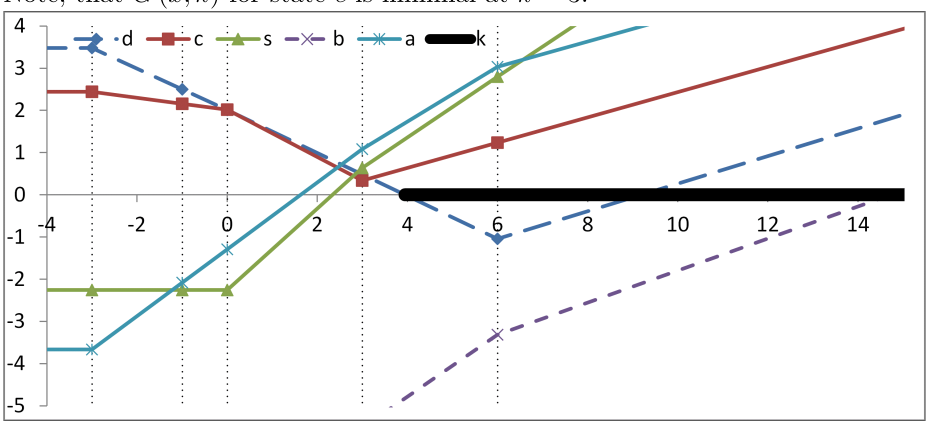
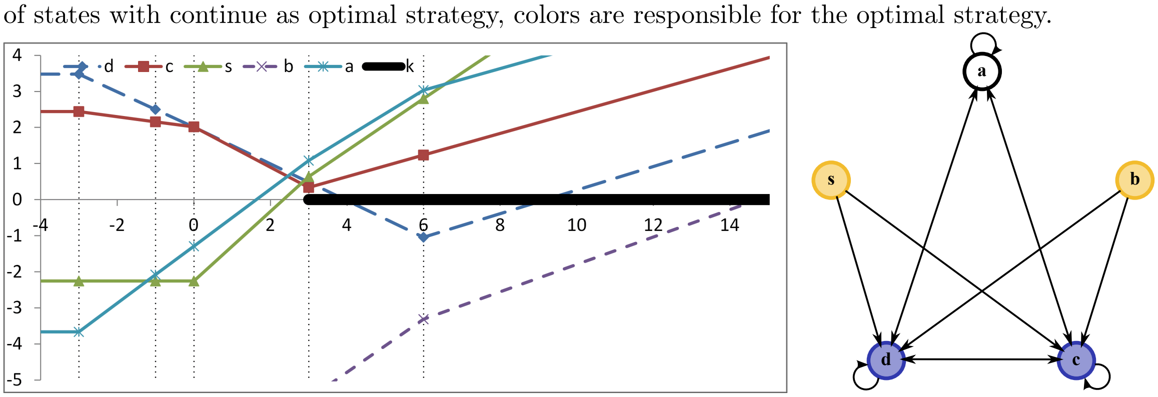


Table A.11: CQR problem, step 6. Value k_0 is set to $k_0 = 3$, working interval $\Delta_i = [0, 3]$. Found $w(s)$. Set optimal strategy for s to be continue. Found optimal strategy for the CQR problem: continue at states $\{b, s\}$, quit at states $\{c, d\}$, restart at states $\{a\}$.

Matrix W, eliminated states: $\{b\}$						Rewards			Calculated values			
	d	c	s	b	a	$\mathbf{c(x)}$	$\mathbf{q(x)}$	$\mathbf{r(x)}$	$d^+(x)$ or $d^-(x)$	$t(x)$	$w(x)$	π
d	0.16	0.00	0.02	0.09	0.49	1.53	0	3		3.95	9.19	quit
c	0.00	0.14	0.42	0.00	0.14	1.00	1	2		3	3	quit
s	0.01	0.24	0.04	0.04	0.00	1.23	4	1	2.34		2.34	continue
b	0.35	0.00	0.27	0.27	0.00	7.59	6	0			14.73	continue
a	0.35	0.14	0.00	0.00	0.21	1.00	0	0	1.63			restart

Figure A.7: CQR problem. Graph of $G(x,k)$ for step 6. Think black line starts at value $k_0 = 3$. Found $w(s)$. The figure on the right shows the final state of the transition matrix after elimination of states with continue as optimal strategy, colors are responsible for the optimal strategy.



APPENDIX B: PROGRAM LISTING

We used Visual Basic for Applications in Excel 2007 as language of choice. The reasons to choose this language were: ease of input test data, including pretty large matrices, ability to plot results. Since algorithm is fairly quick, we did not need any high-performance language.

B.1 StateEliminationInsertion.bas

This module is responsible for performing elimination and insertion step.

```
Option Explicit
Option Private Module
' Performs state elimination and insertion procedures on the transition
' matrix and cost function of Markov Decision Process
'
' Note, that size of transition matrix P is not changing with elimination;
' also, eliminated state still has non-zero values in P.
' These values should be ignored while calculating most of value functionals,
' they are needed to perform insertion of the state back to the MDP.
'
' Requirements:
' Expects substochastic matrix as input.
' The eliminated state z should have p(z,z)<1
'
' Usage:
' Call appropriate function, provide transition matrix, and,
' possibly cost function
'
' Limitations:
'
' define elimination enum
Public Enum stateElimStatus
    sesInclude          ' initial state, the state is included in the calculation
    sesEliminate        ' state is eliminated
    sesFinalInclude     ' final include, after state was eliminated and put back
```

```
End Enum
' Checks dimensions of the input arrays. The dimensions should match.
'
' Variables:
' [P]          in - transition matrix
' [cntCost]    in - continue cost vector
'
' Return:
'    True if dimensions are OK
'    False if dimensions do not match
'
Private Function checkDimensions(P() As Double, cntCost() As Double, _
    idxState As Long) As Boolean
    ' lower bound should be 1
    If LBound(P) <> 1 Or LBound(P, 2) <> 1 Or LBound(cntCost) <> 1 Then
        checkDimensions = False
        Exit Function
    End If
    ' upper bound should match
    Dim size As Long
    size = UBound(cntCost)
    If UBound(P) <> size Or UBound(P, 2) <> size Then
        checkDimensions = False
        Exit Function
    End If
End Function
' Performs elimination of single state
'
' Variables:
' [P]          in/out - transition matrix
' [cntCost]    in/out - continue cost vector
' [idxState]   in     - index of state to eliminate
```

```
'
' Remarks:
'
Public Sub eliminateState(P() As Double, cntCost() As Double, idxState As Long)
    Dim size As Long
    size = UBound(cntCost)

    ' need to compute continue cost first, then change the probability matrix
    Dim cz As Double ' c(z)
    Dim nz As Double ' 1/(1-p(z,z)

    cz = cntCost(idxState)
    nz = 1# / (1# - P(idxState, idxState))

    Dim idxRow As Long, idxCol As Long, idx As Long

    ' compute new continue cost
    For idxRow = 1 To size
        If idxRow = idxState Then
            ' transformation is the same, but has simpler form if x=z
            cntCost(idxRow) = nz * cz
        Else
            cntCost(idxRow) = cntCost(idxRow) + P(idxRow, idxState) * nz * cz
        End If
    Next idxRow

    ' carefully compute new transition matrix
    ' 1. Go over all rows and columns in all states, except eliminated state
    For idxRow = 1 To size
        For idxCol = 1 To size
            If idxCol <> idxState And idxRow <> idxState Then
                P(idxRow, idxCol) = P(idxRow, idxCol) + _
```

```
                                    P(idxRow, idxState) * nz * P(idxState, idxCol)
            End If
        Next idxCol
    Next idxRow

    ' 2. Compute the values for eliminated state. Again,
    '    the formula is the same as in (1.), but it has simpler form
    '
    For idx = 1 To size
        ' need to have this comparison in order to avoid divinding P(z, z) twice
        If idx <> idxState Then
            P(idxState, idx) = P(idxState, idx) * nz
            P(idx, idxState) = P(idx, idxState) * nz
        Else
            ' idx is equal to idxState
            P(idxState, idxState) = P(idxState, idxState) * nz
        End If
    Next idx
End Sub
' Performs insertion of single state
'
' Variables:
' [P]          in/out - transition matrix
' [cntCost]    in/out - continue cost vector
' [idxState]   in     - index of state to eliminate
'
' Remarks:
'
Public Sub insertState(P() As Double, cntCost() As Double, idxState As Long)
    Dim size As Long
    size = UBound(cntCost)
```

```
' need to compute continue cost first, then change the probability matrix
Dim cz As Double ' c(z)
Dim nz As Double ' 1/(1+p(z,z)

cz = cntCost(idxState)
nz = 1# / (1# + P(idxState, idxState))

Dim idxRow As Long, idxCol As Long, idx As Long

' compute new continue cost
For idxRow = 1 To size
    If idxRow = idxState Then
        ' transformation is the same, but has simpler form if x=z
        cntCost(idxRow) = nz * cz
    Else
        cntCost(idxRow) = cntCost(idxRow) - P(idxRow, idxState) * nz * cz
    End If
Next idxRow

' carefully compute new transition matrix
' 1. Go over all rows and columns in all states, except eliminated state
For idxRow = 1 To size
    For idxCol = 1 To size
        If idxCol <> idxState And idxRow <> idxState Then
            P(idxRow, idxCol) = P(idxRow, idxCol) - _
                                P(idxRow, idxState) * nz * P(idxState, idxCol)
        End If
    Next idxCol
Next idxRow

' 2. Compute the values for eliminated state.
' Again, the formula is the same as in (1.), but it has simpler form
```

```
    '
    For idx = 1 To size
        ' need to have this comparison in order to avoid divinding P(z, z) twice
        If idx <> idxState Then
            P(idxState, idx) = P(idxState, idx) * nz
            P(idx, idxState) = P(idx, idxState) * nz
        Else
            ' idx is equal to idxState
            P(idxState, idxState) = P(idxState, idxState) * nz
        End If
    Next idx
End Sub
```

B.2 ModelCqr.cls

This module is responsible for storing definition of CQR model.

```
Option Explicit
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Reward Model with continue, quit, and restart
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' All arrays start with index 1
Private m_name As String                    ' name
Private m_stateNames() As String            ' state names
Private m_transitionMatrix() As Double      ' original transition matrix
Private m_restartCost() As Double           ' original restart cost,
                                            ' has the same size as transition matrix
Private m_restartAllowed() As Boolean       ' vector of restart flags, has true,
                                            ' if restart to this state is allowed
Private m_contCost() As Double              ' cost function for continue
Private m_quitCost() As Double              ' cost function for quit
Private m_terminationProb() As Double       ' probability of termination
Private m_size As Long                      ' size of original model,
                                            ' not including terminal state
```

```
''''''''''''''''''''''''''''''''''''''''''''''
' Expose elements to user
''''''''''''''''''''''''''''''''''''''''''''''
' name
Public Property Get name() As String
    name = m_name
End Property
Public Property Let name(modelName As String)
    m_name = modelName
End Property
' Transition Matrix
Public Property Get transitionMatrix() As Double()
    transitionMatrix = m_transitionMatrix
End Property
Public Property Let transitionMatrix(matrix() As Double)
    m_transitionMatrix = matrix
End Property
' State Names
Public Property Get stateNames() As String()
    stateNames = m_stateNames
End Property
Public Property Let stateNames(names() As String)
    m_stateNames = names
End Property
' Cost
Public Property Get contCost() As Double()
    contCost = m_contCost
End Property
Public Property Let contCost(costVector() As Double)
    m_contCost = costVector
End Property
' Cost
```

```
Public Property Get quitCost() As Double()
    quitCost = m_quitCost
End Property
Public Property Let quitCost(costVector() As Double)
    m_quitCost = costVector
End Property
' Probability of termination
Public Property Get terminationProb() As Double()
    terminationProb = m_terminationProb
End Property
Public Property Let terminationProb(terminationProbability() As Double)
    m_terminationProb = terminationProbability
End Property
' Restart cost
Public Property Get restartCost() As Double()
    restartCost = m_restartCost
End Property
Public Property Let restartCost(restartCostMatrix() As Double)
    m_restartCost = restartCostMatrix
End Property
' Restart allowed
Public Property Get restartAllowed() As Boolean()
    restartAllowed = m_restartAllowed
End Property
Public Property Let restartAllowed(restartAllowedVector() As Boolean)
    m_restartAllowed = restartAllowedVector
End Property
' size
Public Property Get size() As Long
    size = m_size
End Property
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' Error check
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function VerifyInput() As Boolean
    Dim size As Long
    On Error GoTo errHandler

    If LBound(m_transitionMatrix) <> 1 Or LBound(m_transitionMatrix, 2) <> 1 Or _
        LBound(m_restartCost) <> 1 Or LBound(m_restartCost, 2) <> 1 Then
        VerifyInput = False
        Exit Function
    End If

    If LBound(m_stateNames) <> 1 Or LBound(m_contCost) <> 1 Or _
        LBound(m_quitCost) <> 1 Or LBound(m_terminationProb) <> 1 Or _
        LBound(m_restartAllowed) <> 1 Then
        VerifyInput = False
        Exit Function
    End If

    size = UBound(m_terminationProb)
    m_size = size

    If UBound(m_transitionMatrix) <> size Or _
        UBound(m_transitionMatrix, 2) <> size Or UBound(m_restartCost) <> size Or _
        UBound(m_restartCost, 2) <> size Then
        VerifyInput = False
        Exit Function
    End If

    If UBound(m_stateNames) <> size Or UBound(m_contCost) <> size Or _
        UBound(m_quitCost) <> size Or UBound(m_terminationProb) <> size Or _
        UBound(m_restartAllowed) <> size Then
```

```
        VerifyInput = False
        Exit Function
    End If



    VerifyInput = True
    Exit Function
errHandler:
    VerifyInput = False
End Function
```

B.3 SimplexMethodDirect.cls

This module solves linear programming problem using simplex method, assuming that initial feasible basis is given as input.

```
Option Explicit
' simplex method for problems in the form
'
'    max sum c_j*x_j
' subject to
'    sum a_ij*x_j = b_i
'
' the input also contains the list of basis variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Initial variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private m_objective() As Double     ' objective function [vector]
Private m_rhs() As Double           ' rhs of constraints [vector],
                                    ' will be made positive
Private m_lhs() As Double           ' lhs of constraints [matrix]
Private m_basis() As Long           ' current basis
Private m_numVar As Long            ' number of variables
Private m_numCon As Long            ' number of constraints
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' these variables are created during solution
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private m_M() As Double            ' simplex table
' Enumeration for single iteration of simplex method
Private Enum SimplexStatus
    ssOK
    ssUnbounded
    ssFoundSolution
End Enum
' Prints current state of the solver to worksheet
'
' Variables:
' [sheet] in - sheet to use as output
' [row]   in - sheet row where output should be started
' [col] in - sheet column where output should be started
' [step] in - step #
' [message] in - custom message
'
Public Function StatePrint(sheet As Worksheet, ByVal row As Long, _
        ByVal col As Long, step As Long, message As String) As Long
    StatePrint = row
    Exit Function
    ' print current step and name of the model

    sheet.Cells(row, col).value = "Algorithm: simplex method"
    row = row + 1

    sheet.Cells(row, col).value = "Status: " & message
    row = row + 1
```

```
    sheet.Cells(row, col).value = "Iteration: " & (step)
    row = row + 1

    OutputMatrixArbitrary sheet, row, col, "Simplex Matrix", m_M
    StatePrint = row + m_numCon + 1 + 2
End Function
' Purpose:
'   Initialize solver
'
' Variables:
'   objective [in] -- objective function coefficients, c_j
'   lhs       [in] -- matrix of left hand side coefficients, a_ij
'   rhs       [in] -- vector of right hand side coefficients, b_i
'
' Side effects:
'   Changes values of internal class variables
'
Public Sub setup(objective() As Double, lhs() As Double, rhs() As Double, _
                 basis() As Long)
    m_objective = objective
    m_rhs = rhs
    m_lhs = lhs
    m_basis = basis

    ' check dimensions
    m_numVar = UBound(m_objective)
    m_numCon = UBound(m_rhs)

    ' left hand side should have dimension m_numCon, m_numVars
    If UBound(m_lhs) <> m_numCon Or UBound(m_lhs, 2) <> m_numVar Then
        Err.Raise 513, "Simplex Method", "Dimensions do not match"
    End If
```

```
    ReDim m_varSlack(1 To m_numCon) As Long
    ReDim m_varExcess(1 To m_numCon) As Long
    ReDim m_varArt(1 To m_numCon) As Long
    'ReDim m_basis(1 To m_numCon) As Long

    ' setup is done
End Sub
Public Function solve(sheet As Worksheet, ByRef row As Long, ByRef col As Long, _
    basis() As Long) As Boolean
    Dim idxRow As Long
    Dim idxCol As Long

    ReDim variableValues(1 To m_numVar) As Double

    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' PHASE 1
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    ' construct simplex table
    ReDim m_M(0 To m_numCon, 1 To m_numVar + 1) As Double

    ' 1. Cost (objective) function
    For idxCol = 1 To m_numVar
        m_M(0, idxCol) = -m_objective(idxCol)
    Next idxCol
    m_M(0, m_numVar + 1) = 0#

    ' 2. LHS
    Dim multiplier As Double
    For idxRow = 1 To m_numCon
        For idxCol = 1 To m_numVar
```

```
            ' copy initial LHS
            m_M(idxRow, idxCol) = m_lhs(idxRow, idxCol)
        Next idxCol
        m_M(idxRow, m_numVar + 1) = m_rhs(idxRow)
Next idxRow

' 3. Subtract current basis from objective function
Dim basisColumn As Long
For idxRow = 1 To m_numCon
        basisColumn = m_basis(idxRow)
        For idxCol = 1 To m_numVar + 1
            ' subtract
            m_M(0, idxCol) = m_M(0, idxCol) + m_M(idxRow, idxCol) * _
                             m_objective(basisColumn)

            ' round to 0
            If Abs(m_M(0, idxCol)) < 0.00000000001 Then
                m_M(0, idxCol) = 0
            End If
        Next idxCol
Next idxRow




Dim sStatus As SimplexStatus

Dim stepNumber As Long
stepNumber = 1

' print initial setup
row = StatePrint(sheet, row, col, stepNumber, "Phase 1")
```

```
    ' loop until solution is found (or until error)
    Do
        Debug.Print "Phase 1 step # " & stepNumber
        stepNumber = stepNumber + 1
        sStatus = simplexStep
        ' print current step
        row = StatePrint(sheet, row, col, stepNumber, "Phase 1")
    Loop While sStatus = ssOK

    If sStatus <> ssFoundSolution Then
        solve = False
        Exit Function
    End If

    ' return basis
    basis = m_basis
    solve = True

End Function
Private Sub pivotStep(ByVal newBasisCol As Long, ByVal oldBasisRow As Long)
    ' Update basis matrix
    m_basis(oldBasisRow) = newBasisCol



    ' pivoting step
    Dim ratio As Double
    Dim idxRow As Long, idxCol As Long

    ratio = 1# / m_M(oldBasisRow, newBasisCol)

    ' 1. Make incoming m_M 1.0
```

```
        For idxCol = 1 To m_numVar + 1
            m_M(oldBasisRow, idxCol) = m_M(oldBasisRow, idxCol) * ratio
        Next idxCol
        ' put 1.0 at new basis
        m_M(oldBasisRow, newBasisCol) = 1#

        ' 2. All values in pivoting column should be 0.0, except for the basis
        For idxRow = 0 To m_numCon
            If idxRow <> oldBasisRow Then
                ratio = -m_M(idxRow, newBasisCol)
                For idxCol = 1 To m_numVar + 1
                    m_M(idxRow, idxCol) = m_M(idxRow, idxCol) + ratio * _
                                          m_M(oldBasisRow, idxCol)

                    ' round to 0
                    If Abs(m_M(idxRow, idxCol)) < 0.00000000001 Then
                        m_M(idxRow, idxCol) = 0
                    End If
                Next idxCol
                ' make values exactly 0.0
                m_M(idxRow, newBasisCol) = 0#
            End If
        Next idxRow


    End Sub
    Private Function simplexStep() As SimplexStatus
        ' find 1st negative
        Dim idxRow As Long
        Dim idxCol As Long

        Dim newBasisCol As Long   ' this variable will be introduced to
```

```
                        ' the basis, this index goes from 1 to m_numVar
Dim oldBasisRow As Long ' this variable goes out of the basis, _
                        ' this index goes from 1 to m_numCon

newBasisCol = -1#
oldBasisRow = -1#

For idxCol = 1 To m_numVar
    If m_M(0, idxCol) < 0 Then
        newBasisCol = idxCol
        Exit For
    End If
Next idxCol

' no negative coefficients ==> found optimal solution
If newBasisCol = -1 Then
    simplexStep = ssFoundSolution
    Exit Function
End If

Dim pivotFound As Boolean   ' indicates that at least one
Dim minRatio As Double
Dim currRatio As Double



pivotFound = False

' find which variable to take out
For idxRow = 1 To m_numCon
    If m_M(idxRow, newBasisCol) > 0# And m_M(idxRow, m_numVar + 1) > 0# Then
        currRatio = m_M(idxRow, m_numVar + 1) / m_M(idxRow, newBasisCol)
```

```
            ' differentiate 1st pivot vs all other
            If pivotFound Then
                If currRatio < minRatio Then
                    minRatio = currRatio
                    oldBasisRow = idxRow
                End If
            Else
                minRatio = currRatio
                oldBasisRow = idxRow
            End If
            pivotFound = True
        End If
    Next idxRow

    ' if pivot is not found ==> unbounded solution
    If pivotFound = False Then
        simplexStep = ssUnbounded
        Exit Function
    End If

    pivotStep newBasisCol, oldBasisRow

    simplexStep = ssOK

End Function
```

B.4 SolverCqrLPDual.cls

Solver for the CQR model. Uses Linear Programming dual formulation. The solution is obtained by calling simplex method.

```
Option Explicit
'
' Solver for the CQR model. Uses Linear Programming dual formulation.
```

```
' The solution is performed using simplex method.
'
' Requirements:
' CQR model should have substochastic matrix.
'
' Usage:
' Call setModel to pass CQR model
' Call solve after that
'
' Limitations:
' Currently only 1 restart point is supported. It is possible
' to extend this solver to support arbitrary restart points.
'
Private m_cqrModel As ModelCQR    ' input CQR model
Private m_size As Long            ' size of the model
Private m_P() As Double           ' transition matrix
Private m_Cc() As Double          ' continue cost function
Private m_Cq() As Double          ' quit cost function
Private m_Cr() As Double          ' restart to the single point cost function
Private m_stateName() As String   ' state names
Private m_restartIdx As Long      ' index for the state with restart
' Set model to the solver
'
' Variables:
' [cqrModel] In - input CQR model
'
Public Sub setModel(cqrModel As ModelCQR)
    Set m_cqrModel = cqrModel
    PrepareForCalc

End Sub
' Prepare for calculation
```

```
' Resizes all arrays, and initializes all calc variables
'
' Side effects:
'   All member variables are reset
'
Public Sub PrepareForCalc()
    m_cqrModel.VerifyInput
    m_size = m_cqrModel.size

    ReDim m_stateName(1 To m_size + 1) As String
    ReDim m_P(1 To m_size + 1, 1 To m_size + 1) As Double
    ReDim m_Cc(1 To m_size + 1) As Double
    ReDim m_Cq(1 To m_size + 1) As Double
    ReDim m_Cr(1 To m_size + 1) As Double

    Dim idxState As Long

    For idxState = 1 To m_size
        If m_cqrModel.restartAllowed()(idxState) Then
            m_restartIdx = idxState
        End If
        m_stateName(idxState) = m_cqrModel.stateNames()(idxState)
    Next idxState

    m_stateName(m_size + 1) = "*"



    ResetCalcVariables
End Sub
' Initializes all calc variables using CQR model as input
'
' Side effects:
```

```
'    All member variables are reset
'
Private Sub ResetCalcVariables()
    Dim idxRow As Long
    Dim idxCol As Long

    For idxRow = 1 To m_size
        For idxCol = 1 To m_size
            m_P(idxRow, idxCol) = _
                m_cqrModel.transitionMatrix()(idxRow, idxCol) * _
                (1# - m_cqrModel.terminationProb()(idxRow))
            m_P(m_size + 1, idxCol) = 0#
        Next idxCol
        m_P(idxRow, m_size + 1) = m_cqrModel.terminationProb()(idxRow)
        m_Cc(idxRow) = m_cqrModel.contCost()(idxRow)
        m_Cq(idxRow) = m_cqrModel.quitCost()(idxRow)
        m_Cr(idxRow) = m_cqrModel.restartCost()(idxRow, m_restartIdx)
    Next idxRow

    m_P(m_size + 1, m_size + 1) = 1#
    m_Cc(m_size + 1) = 0#
    m_Cq(m_size + 1) = 0#
    m_Cr(m_size + 1) = 0#

End Sub
' Solves CQR model using linear programming formulation
'
' Variables:
' [sheet] In - worksheet to output results
' [sheetRow] In/out - worksheet row where results should go
'
' Side effects:
```

```
'    variable sheetRow is updated to point to the 1st empty line in worksheet
'
Public Sub solve(sheet As Worksheet, ByRef sheetRow As Long)

    PrepareForCalc

    Dim objectiveFunction() As Double
    Dim actionName() As String
    Dim actionState() As Long
    Dim lhs() As Double
    Dim rhs() As Double

    Dim numVar As Long, numCon As Long
    Dim idxVar As Long, idxCon As Long

    Dim sheetCol As Long
    sheetCol = 1

    numVar = 3 * m_size - 1
    numCon = m_size

    ReDim objectiveFunction(1 To numVar) As Double
    ReDim actionName(1 To numVar) As String
    ReDim actionState(1 To numVar) As Long
    ReDim rhs(1 To numCon) As Double
    ReDim lhs(1 To numCon, 1 To numVar) As Double

    ' value of underlying variables
    Dim variableValues() As Double

    ' define objective function
    Dim counter As Long
```

```
counter = 1
For idxVar = 1 To m_size
    ' logic for continue
    objectiveFunction(idxVar) = m_Cc(idxVar)
    actionName(idxVar) = "Continue"
    actionState(idxVar) = idxVar
    ' logic for quit
    objectiveFunction(idxVar + m_size) = m_Cq(idxVar)
    actionName(idxVar + m_size) = "Quit"
    actionState(idxVar + m_size) = idxVar
    ' logic for restart
    If idxVar <> m_restartIdx Then
        objectiveFunction(counter + 2 * m_size) = m_Cr(idxVar)
        actionName(counter + 2 * m_size) = "Restart"
        actionState(counter + 2 * m_size) = idxVar
        counter = counter + 1
    End If
Next idxVar

' LHS, transition matrix P is transposed
For idxCon = 1 To m_size
    counter = 1
    For idxVar = 1 To m_size
        If idxCon = idxVar Then
            lhs(idxCon, idxVar) = 1# - m_P(idxVar, idxCon)
            lhs(idxCon, idxVar + m_size) = 1#
        Else
            lhs(idxCon, idxVar) = -m_P(idxVar, idxCon)
            lhs(idxCon, idxVar + m_size) = 0#
        End If
        ' take care of restart
        If idxVar <> m_restartIdx Then
```

```
            If idxCon <> m_restartIdx Then
                If idxVar = idxCon Then
                    lhs(idxCon, counter + 2 * m_size) = 1
                End If
            Else
                lhs(idxCon, counter + 2 * m_size) = -1
            End If
            counter = counter + 1
        End If

    Next idxVar
Next idxCon

' RHS is equal to 1.0
For idxCon = 1 To m_size
    rhs(idxCon) = 1#
Next idxCon

' create basis, it points to 'quit' dual variables
Dim basis() As Long
ReDim basis(1 To numCon) As Long
For idxCon = 1 To m_size
    basis(idxCon) = m_size + idxCon
Next idxCon

' call Simplex Method
Dim simplexMethod As New SimplexMethodDirect
simplexMethod.setup objectiveFunction, lhs, rhs, basis
Dim success As Boolean
success = simplexMethod.solve(sheet, sheetRow, sheetCol, basis)
' output result in case of success
If success Then
```

```
        Dim action() As String

        ReDim action(1 To m_size + 1) As String
        ' assign action based on optimal basis
        For idxCon = 1 To m_size
            action(actionState(basis(idxCon))) = actionName(basis(idxCon))
        Next idxCon

        sheet.Cells(sheetRow, sheetCol) = "Solution using Linear " & _
            " Programming for dual formulation"
        sheetRow = sheetRow + 1

        OutputVector sheet, sheetRow, sheetCol, "State", m_stateName, m_size + 1

        OutputVector sheet, sheetRow, sheetCol, "Action", action, m_size + 1

        sheetRow = sheetRow + m_size + 1 + 1 + 1
    End If

End Sub
```

B.5 SolverCqrSEA.cls

State elimination algorithm.

```
Option Explicit
'
' Solver for the CQR model, handles 2 cases
'   * no restart points
'   * single restart point with known value function
'
' Requirements:
' CQR model should have substochastic matrix.
'
```

```
' Usage:
' Call setModel to pass CQR model
' Call solve after that
'
' Limitations:
' Currently only 1 restart point is supported. It is possible
' to extend this solver to support arbitrary restart points.
'
Private m_cqrModel As ModelCQR       ' input CQR model
Private m_size As Long               ' size of the model
Private m_P() As Double              ' transition matrix after elimination
Private m_Cc() As Double             ' continue cost function after elimination
Private m_Cq() As Double             ' quit cost function
Private m_Cr() As Double             ' restart to the single point cost function
Private m_stateStatus() As stateElimStatus  ' current elimination status
Private m_stateName() As String      ' state names
Private m_restartIdx  As Long        ' index for the state with restart
' Set model to the solver
'
' Variables:
' [cqrModel] In - input CQR model
'
Public Sub setModel(cqrModel As ModelCQR)
    Set m_cqrModel = cqrModel
    PrepareForCalc

End Sub
' Prepare for calculation
' Resizes all arrays, and initializes all calc variables
'
' Side effects:
'    All member variables are reset
```

```
'
Public Sub PrepareForCalc()
    m_cqrModel.VerifyInput

    m_size = m_cqrModel.size

    ReDim m_stateName(1 To m_size + 1) As String
    ReDim m_stateStatus(1 To m_size + 1) As stateElimStatus
    ReDim m_P(1 To m_size + 1, 1 To m_size + 1) As Double
    ReDim m_Cc(1 To m_size + 1) As Double
    ReDim m_Cq(1 To m_size + 1) As Double
    ReDim m_Cr(1 To m_size + 1) As Double

    Dim idxState As Long
    m_restartIdx = -1

    For idxState = 1 To m_size
        If m_cqrModel.restartAllowed()(idxState) Then
            m_restartIdx = idxState
        End If
        m_stateName(idxState) = m_cqrModel.stateNames()(idxState)
    Next idxState

    m_stateName(m_size + 1) = "*"



    ResetCalcVariables
End Sub
' Initializes all calc variables using CQR model as input
'
' Side effects:
'   All member variables are reset
```

```
'
Private Sub ResetCalcVariables()
    Dim idxRow As Long
    Dim idxCol As Long

    For idxRow = 1 To m_size
        For idxCol = 1 To m_size
            m_P(idxRow, idxCol) = _
                m_cqrModel.transitionMatrix()(idxRow, idxCol) * _
                (1# - m_cqrModel.terminationProb()(idxRow))
            m_P(m_size + 1, idxCol) = 0#
        Next idxCol
        m_P(idxRow, m_size + 1) = m_cqrModel.terminationProb()(idxRow)
        m_Cc(idxRow) = m_cqrModel.contCost()(idxRow)
        m_Cq(idxRow) = m_cqrModel.quitCost()(idxRow)
        m_Cr(idxRow) = m_cqrModel.restartCost()(idxRow, m_restartIdx)
    Next idxRow

    m_P(m_size + 1, m_size + 1) = 1#
    m_Cc(m_size + 1) = 0#
    m_Cq(m_size + 1) = 0#
    m_Cr(m_size + 1) = 0#

End Sub
' Prints current state of the model
' Return: number of rows used
Public Function StatePrintSEA(sheet As Worksheet, _
    ByVal row As Long, ByVal col As Long, step As Long) As Long
    Exit Function
    Dim status() As String
    Dim w As Variant
    Dim t As Variant
```

```
Dim c() As Double
Dim beta() As Double
Dim dpm() As Double
Dim sign() As String
Dim slope() As Double

ReDim status(1 To m_size + 1) As String
ReDim w(1 To m_size + 1) As Variant
ReDim t(1 To m_size + 1) As Variant
ReDim c(1 To m_size + 1) As Double
ReDim beta(1 To m_size + 1) As Double
ReDim dpm(1 To m_size + 1) As Double
ReDim sign(1 To m_size + 1) As String
ReDim slope(1 To m_size + 1) As Double

Dim idxRow As Long
Dim isnegative As Boolean

For idxRow = 1 To m_size + 1
    status(idxRow) = "Include"
    If m_stateStatus(idxRow) = sesEliminate Then
        status(idxRow) = "Eliminate"
    End If
    If m_stateStatus(idxRow) = sesFinalInclude Then
        status(idxRow) = "FinalInclude"
    End If


Next idxRow

' print current step and name of the model
sheet.Cells(row, col).value = m_cqrModel.name
```

```
    row = row + 1

    sheet.Cells(row, col).value = "Iteration: " & (step)
    row = row + 1

    OutputVector sheet, row, col, "State", m_stateName, m_size + 1

    OutputVector sheet, row, col, "Status", status, m_size + 1

    OutputVector sheet, row, col, "Continue", m_Cc, m_size + 1

    OutputVector sheet, row, col, "Quit", m_Cq, m_size + 1

    OutputMatrixColorCols sheet, row, col, "Transition Matrix", m_P, _
        m_stateStatus, m_size + 1

    StatePrintSEA = row + m_size + 1 + 2
End Function
Public Function solve(restartStateValue As Double, _
    useRestartValue As Boolean, sheet As Worksheet, _
    ByRef sheetRow As Long) As Boolean

    ' prepare for calculation
    PrepareForCalc

    Dim idxRow As Long, idxCol As Long
    Dim elementStatus() As String ' action, continue, quit, or restart
    Dim value() As Double ' value function

    ReDim elementStatus(1 To m_size + 1) As String
    ReDim value(1 To m_size + 1) As Double
```

```
' if restart point shouldn't be considered set restart index to
'  nonexisting element
If useRestartValue = False Then
    m_restartIdx = -1
End If

If useRestartValue Then
    ' if restart point value is known, use it
    ' apply restartStateValue=h(s|s) to all states except s
    For idxRow = 1 To m_size
        If idxRow <> m_restartIdx Then
            elementStatus(idxRow) = "Quit"
            If m_Cq(idxRow) < m_Cr(idxRow) + restartStateValue Then
                m_Cq(idxRow) = m_Cr(idxRow) + restartStateValue
                elementStatus(idxRow) = "Restart"
            End If
        End If
    Next idxRow
    ' make restart state 's' to be the absorbing state
    ' with q=h(s|s), c=-inf
    elementStatus(m_restartIdx) = "Quit"
    If m_Cq(m_restartIdx) < restartStateValue Then
        elementStatus(m_restartIdx) = "Continue"
    End If
    m_Cq(m_restartIdx) = restartStateValue
    m_Cc(m_restartIdx) = -1E+300
    For idxCol = 1 To m_size
        m_P(m_restartIdx, idxCol) = 0#
    Next idxCol
    m_P(m_restartIdx, m_size + 1) = 1#
    value(m_restartIdx) = restartStateValue
```

```
Else
    ' restart point value is not used
    ' we're solving plain CQ (continue and quit) problem
    For idxRow = 1 To m_size
        elementStatus(idxRow) = "Quit"
    Next idxRow
End If

' success of the current step
Dim elimination As Boolean
Dim gValue As Double
Dim rowToEliminate As Long

sheetRow = StatePrintSEA(sheet, sheetRow, 1, 0)

Do
    elimination = False
    rowToEliminate = -1
    For idxRow = 1 To m_size
        ' process only included states
        If m_stateStatus(idxRow) = sesInclude And _
           idxRow <> m_restartIdx Then
            gValue = 0#
            gValue = m_Cq(idxRow) - m_Cc(idxRow)
            For idxCol = 1 To m_size
                If m_stateStatus(idxCol) <> sesEliminate Then
                    gValue = gValue - m_P(idxRow, idxCol) * m_Cq(idxCol)
                End If
            Next idxCol

            If gValue < 0 Then
                ' it is optimal to continue here
```

```
                    elimination = True
                    rowToEliminate = idxRow
                    m_stateStatus(idxRow) = sesEliminate
                    eliminateState m_P, m_Cc, idxRow
                    elementStatus(idxRow) = "Continue"
                    Exit For
                End If
            End If
        Next idxRow
        sheetRow = StatePrintSEA(sheet, sheetRow, 1, 0)

    Loop While elimination = True

    ' loop over states and calculate value function
    ' all the states, which are not eliminated are 'Quit'
    For idxRow = 1 To m_size
        If m_stateStatus(idxRow) = sesInclude And idxRow <> m_restartIdx Then
            value(idxRow) = m_Cq(idxRow)
        End If
        If m_stateStatus(idxRow) = sesEliminate And idxRow <> m_restartIdx Then
            value(idxRow) = m_Cc(idxRow)
            For idxCol = 1 To m_size
                If m_stateStatus(idxCol) <> sesEliminate Then
                    value(idxRow) = value(idxRow) + _
                                    m_P(idxRow, idxCol) * m_Cc(idxCol)
                End If
            Next idxCol
        End If
    Next idxRow

    ' continue states need precomputed values for 'Quit/Restart' states
    For idxRow = 1 To m_size
```

```
        If m_stateStatus(idxRow) = sesEliminate And idxRow <> m_restartIdx Then
            value(idxRow) = m_Cc(idxRow)
            For idxCol = 1 To m_size
                If m_stateStatus(idxCol) <> sesEliminate Then
                    value(idxRow) = value(idxRow) + _
                                    m_P(idxRow, idxCol) * value(idxCol)
                End If
            Next idxCol
        End If
    Next idxRow

    ' print result
    sheetRow = sheetRow + 1
    sheet.Cells(sheetRow, 1).value = "Results of SEA algorithm"
    sheetRow = sheetRow + 1

    Dim col As Long
    col = 1

    OutputVector sheet, sheetRow, col, "State", m_stateName, m_size + 1

    OutputVector sheet, sheetRow, col, "Value", value, m_size + 1

    OutputVector sheet, sheetRow, col, "Action", elementStatus, m_size + 1


End Function
```

B.6 SolverCqrNoQuit.cls

Solves CQR problem for the case when there is no quit action.

```
Option Explicit
'
```

```
' Solver for the CQR model with single restart point and no quit action.
' The main algorithm of this solver finds h(s),
' the value at the restart state.
' Then it calls SE algorithm to verify optimality
' of strategy found at point h(s)
'
' Requirements:
' CQR model should have substochastic matrix.
'
' Usage:
' Call setModel to pass CQR model
' Call solve after that
'
' Limitations:
' Currently only 1 restart point is supported. It is possible
' to extend this solver to support arbitrary restart points.
'
Private m_cqrModel As ModelCQR              ' input CQR model
Private m_size As Long                      ' size of the model
Private m_stateName() As String      ' state names, sorted against gamma
Private m_restartIdx  As Long        ' index for the state with restart
Private m_stateStatus() As stateElimStatus  ' current elimination status
Private m_P() As Double              ' transition matrix after elimination
Private m_Cc() As Double             ' continue cost function after elimination
Private m_Cr() As Double             ' restart to the single point cost function
Private m_wIndex() As Double
Private m_wIndexKnown() As Boolean
Private m_h_s_s As Double            ' index h(s)
Private m_hsFound As Boolean         ' flag, indicating if h(s) is found
' Set model to the solver
'
' Variables:
```

```
' [cqrModel] In - input CQR model
'
Public Sub setModel(cqrModel As ModelCQR)
    Set m_cqrModel = cqrModel
    ' set -inf to the quit action
    Dim q() As Double
    q = m_cqrModel.quitCost

    Dim idxRow As Long
    For idxRow = 1 To UBound(q)
        q(idxRow) = 0#
    Next idxRow
    m_cqrModel.quitCost = q
    PrepareForCalc

End Sub
' Prepare for calculation
' Resizes all arrays, and initializes all calc variables
'
' Side effects:
'   All member variables are reset
'
Public Sub PrepareForCalc(Optional ByVal reorderGamma As Boolean = True)
    m_size = m_cqrModel.size
    ReDim m_stateName(1 To m_size + 1) As String
    ReDim m_stateStatus(1 To m_size + 1) As stateElimStatus
    ReDim m_P(1 To m_size + 1, 1 To m_size + 1) As Double
    ReDim m_Cc(1 To m_size + 1) As Double
    ReDim m_Cq(1 To m_size + 1) As Double
    ReDim m_Cr(1 To m_size + 1) As Double
    ReDim m_gamma(1 To m_size + 1) As Double
    ReDim m_gammaIdx(1 To m_size + 1) As Long
```

```
    ReDim m_gammaInverseIdx(1 To m_size + 1) As Long
    ReDim m_tIndex(1 To m_size + 1) As Double
    ReDim m_wIndex(1 To m_size + 1) As Double
    ReDim m_tIndexKnown(1 To m_size + 1) As Boolean
    ReDim m_wIndexKnown(1 To m_size + 1) As Boolean
    Dim idxState As Long
    For idxState = 1 To m_size
        m_stateStatus(idxState) = sesInclude
        If m_cqrModel.restartAllowed()(idxState) Then
            m_restartIdx = idxState
        End If
        m_stateName(idxState) = m_cqrModel.stateNames()(idxState)
    Next idxState
    m_stateName(m_size + 1) = "*"
    ResetCalcVariables
End Sub
' Initializes all calc variables using CQR model as input
'
' Side effects:
'   All member variables are reset
'
Private Sub ResetCalcVariables()
    Dim idxRow As Long
    Dim idxCol As Long

    For idxRow = 1 To m_size
        For idxCol = 1 To m_size
            m_P(idxRow, idxCol) = _
                m_cqrModel.transitionMatrix()(idxRow, idxCol) * _
                    (1# - m_cqrModel.terminationProb()(idxRow))
            m_P(m_size + 1, idxCol) = 0#
        Next idxCol
```

```
        m_P(idxRow, m_size + 1) = m_cqrModel.terminationProb()(idxRow)
        m_Cc(idxRow) = m_cqrModel.contCost()(idxRow)
        m_Cr(idxRow) = m_cqrModel.restartCost()(idxRow, m_restartIdx)
    Next idxRow

    m_P(m_size + 1, m_size + 1) = 1#
    m_Cc(m_size + 1) = 0#
    m_Cr(m_size + 1) = 0#

End Sub
' Prints current state of the solver to worksheet
'
' Variables:
' [sheet] in - sheet to use as output
' [row]   in - sheet row where output should be started
' [col] in - sheet column where output should be started
' [step] in - step #
' [message] in - custom message
'
Public Function StatePrint(sheet As Worksheet, ByVal row As Long, _
    ByVal col As Long, step As Long, message As String) As Long

    Dim status() As String
    Dim w As Variant
    Dim t As Variant
    Dim c() As Double
    Dim beta() As Double
    Dim dpm() As Double
    Dim slope() As Double

    ReDim status(1 To m_size + 1) As String
    ReDim w(1 To m_size + 1) As Variant
```

```
ReDim c(1 To m_size + 1) As Double
ReDim beta(1 To m_size + 1) As Double
ReDim dpm(1 To m_size + 1) As Double
ReDim slope(1 To m_size + 1) As Double

Dim idxRow As Long
Dim isnegative As Boolean

For idxRow = 1 To m_size + 1
    status(idxRow) = "Include"
    If m_stateStatus(idxRow) = sesEliminate Then
        status(idxRow) = "Eliminate"
    End If
    If m_stateStatus(idxRow) = sesFinalInclude Then
        status(idxRow) = "FinalInclude"
    End If

    If m_wIndexKnown(idxRow) Then
        w(idxRow) = m_wIndex(idxRow)
    End If



    c(idxRow) = calculateC(idxRow)
    beta(idxRow) = calculateBeta(idxRow)



    If idxRow <= m_size Then
        calcGIntersection idxRow, dpm(idxRow)
        slope(idxRow) = 1# - beta(idxRow)
    End If

Next idxRow
```

```
' print current step and name of the model
sheet.Cells(row, col).value = "Model name: " & m_cqrModel.name
row = row + 1

sheet.Cells(row, col).value = "Algorithm: finding h(s) " & _
    "with no quit action allowed"
row = row + 1

sheet.Cells(row, col).value = "Status: " & message
row = row + 1


sheet.Cells(row, col).value = "Iteration: " & (step)
row = row + 1


OutputVector sheet, row, col, "State", m_stateName, m_size + 1

OutputVector sheet, row, col, "Status", status, m_size + 1

OutputVector sheet, row, col, "Continue", m_Cc, m_size + 1


OutputVector sheet, row, col, "Restart", m_Cr, m_size + 1


OutputVector sheet, row, col, "w", w, m_size + 1


OutputVector sheet, row, col, "C(x)", c, m_size + 1
```

```
    OutputVector sheet, row, col, "beta(x)", beta, m_size + 1

    OutputVector sheet, row, col, "Slope", slope, m_size + 1

    OutputVector sheet, row, col, "d+", dpm, m_size + 1

    OutputMatrixColorCols sheet, row, col, "Transition Matrix (full)", _
        m_P, m_stateStatus, m_size + 1


    StatePrint = row + m_size + 1 + 2
End Function
' calculate function C(x)
'
' Variables:
' [rowX] In - index of the state x
'
' Return value:
' value of C(x) for the case of no quit
'
Private Function calculateC(ByVal rowX As Long)
    Dim Result As Double
    Dim idxCol As Long


    Result = m_Cc(rowX) - m_Cr(rowX)

    ' don't add eliminated columns
    For idxCol = 1 To m_size
        If m_stateStatus(idxCol) <> sesEliminate Then
            Result = Result + m_P(rowX, idxCol) * m_Cr(idxCol)
        End If
```

```
    Next idxCol

    Result = Result - m_Cr(rowX)
    calculateC = Result
End Function
' calculate function beta(x)
'
' Variables:
' [rowX] In - index of the state x
'
' Return value:
' value of beta(x) for the case of no quit
'
Private Function calculateBeta(rowX As Long)
    Dim Result As Double
    Dim idxCol As Long

    Result = 0
    For idxCol = 1 To m_size
        If m_stateStatus(idxCol) <> sesEliminate Then
            Result = Result + m_P(rowX, idxCol)
        End If
    Next idxCol

    calculateBeta = Result
End Function
Private Sub calcGIntersection( _
    rowX As Long, _
    ByRef intersection As Double)

    Dim c As Double
    Dim beta As Double
```

```
    Dim G As Double

    c = calculateC(rowX)
    beta = calculateBeta(rowX)

    intersection = -1E+300

    ' beta shouldn't be 1.0, check for it
    If beta < 1# - 0.000000000000001 Then
        intersection = c / (1 - beta)
    End If

End Sub
Public Function solve(sheet As Worksheet, ByRef sheetRow As Long) As Boolean

    solve = True
    PrepareForCalc

    Dim idxRow As Long, idxCol As Long

    m_hsFound = False


    ' go over each interval Delta_i
    Dim idxInterval As Long       ' current interval for onsideration
    Dim foundSolution As Boolean  ' 'true' if found solution on current interval
    Dim k0 As Double              ' the smallest value k0 used so far,
                                  ' initially it is +INF


    ' intermediate calculation results
    Dim isnegative As Boolean
```

```
Dim k As Double

' message to be printed
Dim statusMessage As String

Dim iteration As Long
iteration = 1

' prepare everything for calculation (reset model)
ResetCalcVariables

statusMessage = "Initial model"
sheetRow = StatePrint(sheet, sheetRow, 1, iteration, statusMessage)

k0 = 1E+300

Dim maxK As Double
Dim maxKIndex As Long

' loop until all elements are eliminated
Do

    foundSolution = False

    ' reset max value/index
    maxK = -1E+300
    maxKIndex = -1

    ' loop through all states
    For idxRow = 1 To m_size
        ' need to process only included states
        If m_stateStatus(idxRow) <> sesEliminate Then
```

```
                foundSolution = True
                ' find intersection
                calcGIntersection idxRow, k

                ' check if it is intersection with maximal k
                If maxKIndex < 0 Or k > maxK Then
                    maxKIndex = idxRow
                    maxK = k
                End If
            End If
        Next idxRow

        ' analyze if solution is found
        If foundSolution Then
            ' found w index, eliminate state
            m_stateStatus(maxKIndex) = sesEliminate
            m_wIndex(maxKIndex) = maxK
            m_wIndexKnown(maxKIndex) = True
            k0 = maxK
            ' check if we found h(s)
            If m_restartIdx = maxKIndex Then
                m_h_s_s = maxK
                m_hsFound = True
            End If
            ' eliminate state and print matrix
            statusMessage = "Eliminated state: " & m_stateName(maxKIndex)
            eliminateState m_P, m_Cc, maxKIndex
            sheetRow = StatePrint(sheet, sheetRow, 1, iteration, statusMessage)
        End If
        iteration = iteration + 1
    Loop While foundSolution = True 'And m_hsFound = False
```

```
    If m_hsFound Then
        statusMessage = "Found h(s)=" & m_h_s_s & _
            ". End of algorithm, continue to SE algorithm."
        sheetRow = StatePrint(sheet, sheetRow, 1, idxInterval, statusMessage)

        Dim solverSea As New SolverCqrSEA
        solverSea.setModel m_cqrModel
        solverSea.solve m_h_s_s, True, sheet, sheetRow
    End If

End Function
```

B.7 SolverCqrSingleR.cls

Main solver for CQR problem.

```
Option Explicit
'
' Solver for the CQR model with single restart point.
' The main algorithm of this solver finds h(s),
' the value at the restart state. The strategy at this point
' is the optimal strategy for CQR problem. Then it calls SE
' algorithm to verify optimality of found strategy.
'
' Requirements:
' CQR model should have substochastic matrix.
'
' Usage:
' Call setModel to pass CQR model
' Call solve after that
'
' Limitations:
```

```
'
Private m_cqrModel As ModelCQR              ' input CQR model
Private m_size As Long                      ' size of the model
Private m_stateNameSorted() As String ' state names, sorted against gamma
Private m_restartIdx  As Long        ' index for the state with restart
Private m_stateStatus() As stateElimStatus  ' current elimination status
Private m_P() As Double              ' transition matrix after elimination
Private m_Cc() As Double             ' continue cost function after elimination
Private m_Cq() As Double             ' quit cost function
Private m_Cr() As Double             ' restart to the single point cost function
Private m_gamma() As Double      ' value of the index gamma_i
Private m_gammaIdx() As Long     ' remapping of x_i in such way,
                                 ' that gamma_1>gamma_2>...
Private m_gammaInverseIdx() As Long ' to map from ordered space back
                                 ' to original one
Private m_tIndex() As Double
Private m_wIndex() As Double
Private m_tIndexKnown() As Boolean
Private m_wIndexKnown() As Boolean
Private m_h_s_s As Double        ' index h(s)
Private m_hsFound As Boolean     ' flag, indicating if h(s) is found
' Set model to the solver
'
' Variables:
' [cqrModel] In - input CQR model
'
Public Sub setModel(cqrModel As ModelCQR)
    Set m_cqrModel = cqrModel
    PrepareForCalc

End Sub
' Prepare for calculation
```

```
' Resizes all arrays, and initializes all calc variables
'
' Side effects:
'   All member variables are reset
'
Public Sub PrepareForCalc(Optional ByVal reorderGamma As Boolean = True)
    m_size = m_cqrModel.size
    ReDim m_stateNameSorted(1 To m_size + 1) As String
    ReDim m_stateStatus(1 To m_size + 1) As stateElimStatus
    ReDim m_P(1 To m_size + 1, 1 To m_size + 1) As Double
    ReDim m_Cc(1 To m_size + 1) As Double
    ReDim m_Cq(1 To m_size + 1) As Double
    ReDim m_Cr(1 To m_size + 1) As Double
    ReDim m_gamma(1 To m_size + 1) As Double
    ReDim m_gammaIdx(1 To m_size + 1) As Long
    ReDim m_gammaInverseIdx(1 To m_size + 1) As Long
    ReDim m_tIndex(1 To m_size + 1) As Double
    ReDim m_wIndex(1 To m_size + 1) As Double
    ReDim m_tIndexKnown(1 To m_size + 1) As Boolean
    ReDim m_wIndexKnown(1 To m_size + 1) As Boolean
    Dim idxState As Long
    For idxState = 1 To m_size
        m_stateStatus(idxState) = sesInclude
        If m_cqrModel.restartAllowed()(idxState) Then
            m_restartIdx = idxState
        End If
    Next idxState
    m_stateNameSorted(m_size + 1) = "*"
    calcGamma (reorderGamma)
    ResetCalcVariables
End Sub
Private Sub calcGamma(Optional reorderGamma As Boolean = True)
```

```
Dim idxRow As Long

For idxRow = 1 To m_size
    m_gamma(idxRow) = m_cqrModel.quitCost()(idxRow) - _
        m_cqrModel.restartCost()(idxRow, m_restartIdx)
    m_gammaIdx(idxRow) = idxRow
Next idxRow
m_gammaIdx(m_size + 1) = m_size + 1

' sort gammas
If reorderGamma Then
    Dim i As Long, j As Long
    Dim tmpDbl As Double, tmpLong As Long
    For i = 1 To m_size
        For j = 1 To m_size - 1
            If m_gamma(j + 1) > m_gamma(j) Then
                ' swap both gamma and gammaIdx
                tmpDbl = m_gamma(j + 1)
                m_gamma(j + 1) = m_gamma(j)
                m_gamma(j) = tmpDbl

                tmpLong = m_gammaIdx(j + 1)
                m_gammaIdx(j + 1) = m_gammaIdx(j)
                m_gammaIdx(j) = tmpLong
            End If
        Next j
    Next i
End If
' create backward map
For idxRow = 1 To m_size + 1
    m_gammaInverseIdx(m_gammaIdx(idxRow)) = idxRow
    If idxRow <= m_size Then
```

```
            m_stateNameSorted(idxRow) = _
                m_cqrModel.stateNames()(m_gammaIdx(idxRow))
        End If
    Next idxRow
End Sub
' Initializes all calc variables using CQR model as input
'
' Side effects:
'   All member variables are reset
'
Private Sub ResetCalcVariables()
    Dim idxRow As Long
    Dim idxCol As Long

    For idxRow = 1 To m_size
        For idxCol = 1 To m_size
            m_P(idxRow, idxCol) = _
                m_cqrModel.transitionMatrix()(m_gammaIdx(idxRow), _
                    m_gammaIdx(idxCol)) * _
                (1# - m_cqrModel.terminationProb()(m_gammaIdx(idxRow)))
            m_P(m_size + 1, idxCol) = 0#
        Next idxCol
        m_P(idxRow, m_size + 1) = _
            m_cqrModel.terminationProb()(m_gammaIdx(idxRow))
        m_Cc(idxRow) = m_cqrModel.contCost()(m_gammaIdx(idxRow))
        m_Cq(idxRow) = m_cqrModel.quitCost()(m_gammaIdx(idxRow))
        m_Cr(idxRow) = _
            m_cqrModel.restartCost()(m_gammaIdx(idxRow), m_restartIdx)
    Next idxRow

    m_P(m_size + 1, m_size + 1) = 1#
    m_Cc(m_size + 1) = 0#
```

```
    m_Cq(m_size + 1) = 0#
    m_Cr(m_size + 1) = 0#

End Sub
' Calculate value of g(x|k)=max(q(x), r(x)+k)
Private Function calcGreward(state As Long, k As Double)
    If m_Cq(state) > m_Cr(state) + k Then
        calcGreward = m_Cq(state)
    Else
        calcGreward = m_Cr(state) + k
    End If
End Function
' Calculate value of G-function for given state and value of parameter k
Private Function calcGValue(state As Long, k As Double) As Double
    Dim Result As Double
    Dim idx As Long

    ' initial value
    Result = calcGreward(state, k) - m_Cc(state)

    For idx = 1 To m_size
        ' transition probability to eliminated state is 0
        If m_stateStatus(idx) <> sesEliminate Then
            Result = Result - m_P(state, idx) * calcGreward(idx, k)
        End If
    Next idx
    calcGValue = Result
End Function
' Calculate value of G-function at all points of Gamma(i) and 2 more on sides
' The first row of the gTable array contains g(x|k) arguments
' Resizes two input arrays
'
```

```
Private Sub CalcGTable(args() As Double, gTable() As Double)
    ' resize args to have n+2 values
    ReDim args(1 To m_size + 2) As Double
    ' resize G function values table
    ReDim gTable(1 To m_size + 1, 1 To m_size + 2) As Double

    Dim idxRow As Long
    Dim idxCol As Long

    ' initialize arguments
    For idxCol = 2 To m_size + 1
        args(idxCol) = m_gamma(m_size + 2 - idxCol)
    Next idxCol

    Dim argSpan As Double ' difference between smallest gamma and largest gamma
    argSpan = Abs(args(m_size + 1) - args(1))
    If argSpan < 1 Then argSpan = 1
    args(m_size + 2) = args(m_size + 1) + argSpan * 2
    args(1) = args(2) - argSpan * 2



    ' copy arguments to the first row of gTable
    For idxCol = 1 To m_size + 2
        gTable(1, idxCol) = args(idxCol)
    Next idxCol

    ' calculate value of function G for each state
    For idxRow = 1 To m_size
        For idxCol = 1 To m_size + 2
            gTable(idxRow + 1, idxCol) = calcGValue(idxRow, args(idxCol))
        Next idxCol
    Next idxRow
```

```
End Sub
' Return: number of rows used
Public Function StatePrint(sheet As Worksheet, ByVal row As Long, _
    ByVal col As Long, step As Long, gammaMax As Double, _
    gammaMin As Double, message As String) As Long
    Exit Function
    Dim status() As String
    Dim chartTitle() As String
    Dim w As Variant
    Dim t As Variant
    Dim c() As Double
    Dim beta() As Double
    Dim dpm() As Double
    Dim sign() As String
    Dim slope() As Double

    ReDim status(1 To m_size + 1) As String
    ReDim chartTitle(1 To m_size + 1) As String
    ReDim w(1 To m_size + 1) As Variant
    ReDim t(1 To m_size + 1) As Variant
    ReDim c(1 To m_size + 1) As Double
    ReDim beta(1 To m_size + 1) As Double
    ReDim dpm(1 To m_size + 1) As Double
    ReDim sign(1 To m_size + 1) As String
    ReDim slope(1 To m_size + 1) As Double

    Dim gTableArgs() As Double       ' G Table
    Dim gTableVals() As Double       ' G table arguments



    Dim idxRow As Long
```

```
Dim isnegative As Boolean

CalcGTable gTableArgs, gTableVals

' create chart title
chartTitle(1) = "k"
For idxRow = 1 To m_size
    chartTitle(idxRow + 1) = m_stateNameSorted(idxRow)
Next idxRow

For idxRow = 1 To m_size + 1
    status(idxRow) = "Include"
    If m_stateStatus(idxRow) = sesEliminate Then
        status(idxRow) = "Eliminate"
    End If
    If m_stateStatus(idxRow) = sesFinalInclude Then
        status(idxRow) = "FinalInclude"
    End If

    If m_wIndexKnown(idxRow) Then
        w(idxRow) = m_wIndex(idxRow)
    End If

    If m_tIndexKnown(idxRow) Then
        t(idxRow) = m_tIndex(idxRow)
    End If

    c(idxRow) = calculateC(idxRow, step)
    beta(idxRow) = calculateBeta(idxRow, step)


    If idxRow <= m_size Then
```

```
            calcGSignAndIntersection idxRow, step, gammaMax, _
                gammaMin, isnegative, dpm(idxRow)
            sign(idxRow) = "+"
            If isnegative Then
                sign(idxRow) = "-"
            End If
            If idxRow < step Then
                slope(idxRow) = -beta(idxRow)
            Else
                slope(idxRow) = 1# - beta(idxRow)
            End If
        End If

    Next idxRow

    ' print current step and name of the model
    sheet.Cells(row, col).value = "Model name: " & m_cqrModel.name
    row = row + 1

    sheet.Cells(row, col).value = "Algorithm: finding h(s)"
    row = row + 1

    sheet.Cells(row, col).value = "Status: " & message
    row = row + 1



    sheet.Cells(row, col).value = "Interval delta index: " & (step)
    row = row + 1

    sheet.Cells(row, col).value = _
        "Interval delta values = [ " & gammaMin & ", " & gammaMax & " ]"
    row = row + 1
```

```
    OutputVector sheet, row, col, "State", m_stateNameSorted, m_size + 1
    OutputVector sheet, row, col, "Status", status, m_size + 1
    OutputVector sheet, row, col, "Continue", m_Cc, m_size + 1
    OutputVector sheet, row, col, "Quit", m_Cq, m_size + 1
    OutputVector sheet, row, col, "Restart", m_Cr, m_size + 1
    OutputVector sheet, row, col, "gamma", m_gamma, m_size + 1
    OutputVector sheet, row, col, "w", w, m_size + 1
    OutputVector sheet, row, col, "t", t, m_size + 1
    OutputVector sheet, row, col, "C(x|i)", c, m_size + 1
    OutputVector sheet, row, col, "beta(x|i)", beta, m_size + 1
    OutputVector sheet, row, col, "Sign at the beginning of the interval", sign, m_s
    OutputVector sheet, row, col, "Slope", slope, m_size + 1
    OutputVector sheet, row, col, "d+-", dpm, m_size + 1
    OutputMatrixColorCols sheet, row, col, "Transition Matrix (full)", m_P, m_stateS
    OutputVector sheet, row, col, "Chart Title", chartTitle, m_size + 1
    OutputMatrix sheet, row, col, "Value of G(x|k)", gTableVals, m_size + 1, m_size
    StatePrint = row + m_size + 1 + 2
End Function
' calculate function C(x, i) in the terms of rearranged
' variables (all calc variables are rearranged)
Private Function calculateC(ByVal rowX As Long, ByVal rowI As Long)
    Dim Result As Double
    Dim idxCol As Long

    If rowI = 0 Then
        rowI = 1
    End If

    Result = m_Cc(rowX)

    ' A(x|i)
```

```
    For idxCol = 1 To rowI - 1
        If m_stateStatus(idxCol) <> sesEliminate Then
            Result = Result + m_P(rowX, idxCol) * m_Cq(idxCol)
        End If
    Next idxCol

    ' B(x|i)
    For idxCol = rowI To m_size
        If m_stateStatus(idxCol) <> sesEliminate Then
            Result = Result + m_P(rowX, idxCol) * m_Cr(idxCol)
        End If
    Next idxCol

    Result = Result - m_Cr(rowX)
    calculateC = Result
End Function
' calculate function beta(x|i)
Private Function calculateBeta(rowX As Long, rowI As Long)
    Dim Result As Double
    Dim idxCol As Long

    If rowI = 0 Then
        rowI = 1
    End If

    Result = 0
    For idxCol = rowI To m_size
        If m_stateStatus(idxCol) <> sesEliminate Then
            Result = Result + m_P(rowX, idxCol)
        End If
    Next idxCol
```

```
    calculateBeta = Result
End Function
Private Sub calcGSignAndIntersection( _
    rowX As Long, _
    rowI As Long, _
    gammaMax As Double, _
    gammaMin As Double, _
    ByRef isnegative As Boolean, _
    ByRef intersection As Double)

    Dim c As Double
    Dim beta As Double
    Dim G As Double

    c = calculateC(rowX, rowI)
    beta = calculateBeta(rowX, rowI)

    intersection = -1E+300

    If rowX < rowI Then
        ' use formula 50 if gamma(x)>gamma_i, or, alternatively rowX<rowI
        ' 1. Get G at point gammaMin
        G = m_gamma(rowX) - c - beta * gammaMin
        If beta > 0 Then
            intersection = (m_gamma(rowX) - c) / beta
        Else
            intersection = -1E+300
        End If
    Else
        ' use formula 51
        ' 1. Get G at point gammaMin
        G = -c + gammaMin * (1# - beta)
```

```
        intersection = c / (1# - beta)
    End If
    If G < 0# Then
        isnegative = True
    Else
        isnegative = False
    End If
End Sub
Public Function solve(sheet As Worksheet, ByRef sheetRow As Long) As Boolean

    solve = True
    PrepareForCalc
    Dim gammaMin() As Double
    Dim gammaMax() As Double

    ReDim gammaMin(1 To m_size + 1) As Double
    ReDim gammaMax(1 To m_size + 1) As Double

    Dim idxRow As Long, idxCol As Long

    m_hsFound = False

    For idxRow = 1 To m_size
        ' don't forget, that m_gamma is decreasing when index is increasing
        gammaMax(idxRow + 1) = m_gamma(idxRow)
        gammaMin(idxRow) = m_gamma(idxRow)
    Next idxRow
    gammaMax(1) = 1E+300
    gammaMin(m_size + 1) = -1E+300

    ' go over each interval Delta_i
    Dim idxInterval As Long     ' current interval for onsideration
```

```
Dim foundSolution As Boolean ' 'true' if found solution on
                             ' current interval
Dim k0 As Double             ' the smallest value k0
                             ' used so far, initially it is +INF

' d+ and d-, i.e. intersection when going
' from + to - and when going from - to +
Dim maxDPlus As Double
Dim maxDMinus As Double
Dim maxDPlusIndex As Long
Dim maxDMinusIndex As Long

' intermediate calculation results
Dim isnegative As Boolean
Dim k As Double

' message to be printed
Dim statusMessage As String

' prepare everything for calculation (reset model)
ResetCalcVariables

Dim iteration As Long
iteration = 1

For idxInterval = 1 To m_size + 1
    statusMessage = "Entering inverval: " & idxInterval
    sheetRow = StatePrint(sheet, sheetRow, 1, idxInterval, _
        gammaMax(idxInterval), gammaMin(idxInterval), statusMessage)

    k0 = gammaMax(idxInterval)
    Do
```

```
foundSolution = False
maxDPlusIndex = -1
maxDMinusIndex = -1

For idxRow = 1 To m_size
    If m_stateStatus(idxRow) <> sesFinalInclude Then
        calcGSignAndIntersection idxRow, idxInterval, _
            gammaMax(idxInterval), gammaMin(idxInterval), _
            isnegative, k
    End If
    ' handle included states
    If m_stateStatus(idxRow) = sesInclude Then
        ' there is an intersection on this interval
        If isnegative Then
            If k >= gammaMin(idxInterval) And _
               k <= gammaMax(idxInterval) Then
                If maxDPlusIndex = -1 Or k > maxDPlus Then
                    maxDPlus = k
                    maxDPlusIndex = idxRow
                    foundSolution = True
                End If
            Else
                Debug.Print "Something is wrong!"
            End If
        End If
    End If
    ' handle eliminated states
    If m_stateStatus(idxRow) = sesEliminate Then
        ' there is an intersection on this interval
        If isnegative = False Then
            If k >= gammaMin(idxInterval) And _
```

```
                k <= gammaMax(idxInterval) Then
                    If maxDMinusIndex = -1 Or k > maxDMinus Then
                        maxDMinus = k
                        maxDMinusIndex = idxRow
                        foundSolution = True
                    End If
                Else
                    Debug.Print "Something is wrong!"
                End If
            End If
        End If
    Next idxRow
    ' analyze if solution is found
    If foundSolution Then
        ' check which one is bigger
        If maxDPlusIndex > 0 And _
            (maxDPlus >= maxDMinus Or maxDMinusIndex <= 0) Then
            ' found w index, eliminate state
            m_stateStatus(maxDPlusIndex) = sesEliminate
            m_wIndex(maxDPlusIndex) = maxDPlus
            m_wIndexKnown(maxDPlusIndex) = True
            k0 = maxDPlus
            ' check if we found h(s)
            If m_restartIdx = m_gammaIdx(maxDPlusIndex) Then
                m_h_s_s = maxDPlus
                m_hsFound = True
            End If
            ' eliminate state and print matrix
            statusMessage = "Eliminated state: " & _
                m_stateNameSorted(maxDPlusIndex)
            eliminateState m_P, m_Cc, maxDPlusIndex
            sheetRow = StatePrint(sheet, sheetRow, 1, _
```

```
                    idxInterval, gammaMax(idxInterval), _
                    gammaMin(idxInterval), statusMessage)
            End If
            If maxDMinusIndex > 0 And _
                (maxDMinus > maxDPlus Or maxDPlusIndex <= 0) Then
                ' found t index
                m_stateStatus(maxDMinusIndex) = sesFinalInclude
                m_tIndex(maxDMinusIndex) = maxDMinus
                m_tIndexKnown(maxDMinusIndex) = True
                k0 = maxDPlus
                ' insert state and print matrix
                statusMessage = "Inserted back state: " & _
                    m_stateNameSorted(maxDMinusIndex)
                insertState m_P, m_Cc, maxDMinusIndex
                sheetRow = StatePrint(sheet, sheetRow, 1, _
                    idxInterval, gammaMax(idxInterval), _
                    gammaMin(idxInterval), statusMessage)
            End If
        End If
        iteration = iteration + 1
    Loop While foundSolution = True 'And m_hsFound = False

    ' special handling for case when idxInterval is equal to the state
    ' and state G-function is positive
    If idxInterval <= m_size Then 'And m_hsFound = False Then
        If m_stateStatus(idxInterval) = sesInclude Then
            m_stateStatus(idxInterval) = sesFinalInclude
            m_tIndex(idxInterval) = m_gamma(idxInterval)
            m_wIndex(idxInterval) = m_gamma(idxInterval)
            m_tIndexKnown(idxInterval) = True
            m_wIndexKnown(idxInterval) = True
            If m_restartIdx = m_gammaIdx(idxInterval) Then
```

```
                m_h_s_s = m_gamma(idxInterval)
                m_hsFound = True
            End If
            statusMessage = "Found h(x) and t(x) for state " & _
                "with always positive G(x|k,i). State: " & _
                m_stateNameSorted(idxInterval)
            sheetRow = StatePrint(sheet, sheetRow, 1, _
                idxInterval, gammaMax(idxInterval), _
                gammaMin(idxInterval), statusMessage)

        End If
    End If
    If m_hsFound Then
        'Exit For
    End If
Next idxInterval

If m_hsFound Then
    statusMessage = "Found h(s)=" & m_h_s_s & _
        ". End of algorithm, continue to SE algorithm."
    sheetRow = StatePrint(sheet, sheetRow, 1, idxInterval, _
        gammaMax(m_size + 1), gammaMin(m_size + 1), statusMessage)

    ' call state elimination to verify that it
    ' finds the same optimal strategy
    Dim solverSea As New SolverCqrSEA
    solverSea.setModel m_cqrModel
    solverSea.solve m_h_s_s, True, sheet, sheetRow
End If

End Function
```