Risk-sensitive LQR problems with exponential noise

Olzhas Shortanbaiuly

A thesis submitted in partial fulfillment of the requirements

for the degree of

Master of Science in Applied Mathematics

NAZARBAYEV UNIVERSITY

Principal Supervisor:

Prof. Kerem Ugurlu (Nazarbayev University)

Second reader:

Prof. Manat Mustafa (Nazarbayev University)

Spring 2024

Abstract

This thesis is about optimal control of Markov Decision Processes and solving risk-sensitive cost minimization and reward maximization problems, specifically, the Linear Quadratic Regulator (LQR) problem with Average-Value-at-Risk criteria. The problem is solved for different risk levels, different random noises (theoretical and sampled), and using different methods: analytical and approximate dynamic programming. The obtained results were analyzed and discussed for the presence of certain patterns and trends. The results show that approximate dynamic programming is a very accurate method for solving risk-sensitive LQR problems with exponential noise.

Keywords: LQR problem, Markov Decision Process, Average-Value-at-Risk, Approximate Dynamic Programming, Exponential Distribution

Acknowledgments

I want to thank Dr.Ugurlu for helping and supporting me throughout my academic journey. Additional gratitude is towards the professors of Department of Mathematics of Nazarbayev University for all the knowledge and guidance through the way.

My eternal gratitude to my parents, who gave their all for me and made me the best person I am. I would also like to thank my dearest friends, who shared with me all my highest and lowest moments with support. Thank you all!

Table of Contents

Chapter 1

Introduction

Optimal control problems involve using optimal control theory to solve the optimization problem over the controls. Different kinds of optimal control problems exist in different fields including science and engineering.

In this thesis, discrete control is considered under a discrete-time setting. The solved problem is about minimizing the cost through performing the optimization over the controls [1]. It can be solved through dynamic programming when the optimal control problem is formulated as a dynamic programming problem [2].

There is an existing risk in a real-world environment, such as a financial market. It is important to solve optimal control problems under such an environment, subject to a certain risk. This thesis aims at solving a certain optimal control problem under two environments: riskless and risk-sensitive with existing risk.

Neural networks and dynamic programming are two approaches that are commonly used in such problems. Due to the curse of dimensionality introduced by neural networks, the research focused more on dynamic programming methods, specifically, approximate dynamic programming.

1.1 Motivation

This problem needs to be solved due to the many applications involved. For example, we may aim to optimize the total cost of making a flight between two cities having no direct route. The states are cities or places that we are located at, the next state is the place we landed after making some flight, the controls are chosen flight routes and costs are the ticket costs. We are only given the initial and terminal states indicating the two sides of the flight.

Other than that, there is a main motivation to study this topic - the problem of minimizing the expected cost for trading blocks of stocks over a fixed time horizon [6]. Given a_t the number of shares and p_t prices, we optimize

$$\min_{\{a_t\}_{t=0}^{T-1}} \mathbb{E} \sum_{t=0}^{T-1} p_t^T a_t$$

Solving the minimization problem for the Average-Value-at-Risk, risk measure used in this thesis, is not an easy task considering the properties this indicator holds. In comparison with other expected performance criteria as an expected value in probability, linearity property does not necessarily hold:

$$AVaR_\alpha(X+Y) \neq AVaR_\alpha(X) + AVaR_\alpha(Y)$$

while

$$\mathbb{E}(X+Y) = \mathbb{E}(X) + \mathbb{E}(Y)$$

for any random variables X, Y.

1.2 Preliminaries

1.2.1 Markov Decision Process

The following definition gives the key concept of a Markov decision process [9].

Definition 1.1. A Markov decision process is a 4-tuple (X, A, P_a, R_a) such that:

- State space X - set of all possible states
- Action space A - set of all possible actions
- $P_a(x_1, x_2) = P(x_{t+1} = x_2 | x_t = x_1, a_t = a)$ is a probability that using action a, we move from state x_t at time t to the state x_{t+1} at time $t+1$.

- $R_a(x_t, x_{t+1})$ is an immediately received reward by moving from state x_t to x_{t+1} by performing an action a

Lemma 1.1. A decision process (X, A, P_a, R_a) is Markovian if and only if

$$P_a(x_{t+1}|x_t) = P_a(x_{t+1}|x_t, x_{t-1}, x_{t-2}, ..., x_0)$$

This is an important property allowing to disregard all states except the previous one, simplifying the computations for this thesis.

1.2.2 Dynamic programming

We are given:

- set of times $t = 0, 1, 2, \dots T$
- the states of the dynamic program $x_t \in X$ where X is the set of states and x_0 is the initial state
- policy $\pi(s, x) = a_s$ for actions $a_s \in A$ and $s = t, ..., T$

  $\pi = (\pi_t : t = 0, 1, 2, \dots T)$ with controls(actions) π_t at each time $t = 0, 1, 2, \dots T$
- costs for taking action a_t at state x_t given by $c_t(a_t, x_t)$
- the sequence of states defined as: $x_{t+1} = f(x_t, \pi_t)$ and transition given by Markov Decision Process [3]

With this given information, we solve the following optimization problem through iterating backward in time:

$$\begin{aligned} \underset{\pi}{\text{minimize}} \quad & C(x_0, \pi) := \sum_{t=0}^{T-1} c(x_t, \pi_t) + c_T(x_T) \\ \text{subject to} \quad & x_{t+1} = f(x_t, \pi_t), \; \pi_t \in A \qquad t = 0, 1, 2, \dots, T. \end{aligned}$$

This approach is called **dynamic programming**.

1.2.3 Approximate dynamic programming

There are certain issues with applying backward dynamic programming.

- **the state space** X for the problem may be too large (difficult to evaluate the value function $V_t\ (x_t)$ for all states within a reasonable time);
- **the decision space** A may be too large (difficult to find the optimal decision for all states within reasonable time);
- computing the expectation of 'future' costs may be intractable when **the outcome space** (set of all possible states in time period $t+1$, given the state and decision in time period t) is large;

This motivates us to introduce alternative approach. For this, approximate dynamic programming is introduced.

For each time step t, **states** $x_t \in X$, **decisions** $a_t \in A$ bringing to a new state x_{t+1} with probability $\mathbb{P}(x_{t+1}|x_t, a_t)$, **discount factor** γ, **deterministic/direct cost** $c_t(x_t, a_t)$, **planning horizon** T, **policy** $\pi \in \Pi$ with **decision function** $X^\pi(x_t)$ returning a decision $a_t \in A$ for all states $x_t \in X$, we solved

$$\min_{\pi\in\Pi} \mathbb{E}^\pi \left\{\sum_{t=0}^{T} \gamma c_t(x_t, X_t^\pi(x_t))\right\}$$

Reformulate it as a combination of smaller subproblems, giving us the Bellman equation (for making a Markov Decision Process model)

$$V_t(S_t) = \min_{a_t\in A} \left(c_t(x_t, a_t) + \gamma \sum_{x'\in X} \mathbb{P}(x_{t+1} = x'|x_t, a_t)V_{t+1}(x')\right)$$

For outcome space - set of possible states in period $t+1$ given the state and decision in period t, its size is driven by the random information W_{t+1} (independent of all prior information) that arrives between t and $t+1$. This new information is integrated in a transition function $x_{t+1} = X^M(x_t, a_t, W_{t+1})$.

Let $\mathbb{P}(W_{t+1} = \omega)$ denote the probability of outcome $\omega \in \Omega_{t+1}$. Then rewrite the previous equation with

$$V_t(S_t) = \min_{a_t\in A} \Big(c_t(x_t, a_t) + \gamma \sum_{\omega\in\Omega_{t+1}} \mathbb{P}(W_{t+1} = \omega)V_{t+1}(x_{t+1}|x_t, a_t, \omega)\Big))$$

Combine optimization with simulation (sampling from Ω_{t+1}), use approximations of the optimal values of Bellman's equations, and use approximate policies (further integrating post-decision states)

$$V_t(S_t) = \min_{a_t \in A} \left(c_t(x_t, a_t) + \gamma \mathbb{E}^{\omega} \left[V_{t+1}(x_{t+1} | x_t, a_t, \omega) \right] \right)$$

The post-decision state x_t^a - the state immediately after action a_t, but before the arrival of new information W_{t+1} allowing to estimate the downstream costs. Assigning the expected downstream costs $\mathbb{E}^{\omega} \left[V_{t+1}(x_{t+1} | x_t^a, \omega) \right]$ to every post-decision state x_t^a eliminates the need to evaluate all possible outcomes ω for every action.

$$V_{t-1}^a(x_{t-1}^a) = \mathbb{E}^{\omega} \left[V_t(x_t | x_{t-1}^a, \omega) \right]$$
$$V_t(x_t) = \min_{a_t \in A} (c_t(x_t, a_t) + \gamma V_t^a(x_t^a))$$
$$V_t^a(x_t^a) = \mathbb{E}^{\omega} \left[V_{t+1}(x_{t+1} | x_t^a, \omega) \right]$$

Using these equations, the optimization problem changes to

$$V_{t-1}^a(x_{t-1}^a) = \mathbb{E}^{\omega} \left[\min_{a_t \in A} (c_t(x_t, a_t) + \gamma V_t^a(x_{t-1}^a, \omega)) \right]$$

Solve the Bellman's equations only for one state at each stage, using estimates of the downstream values, and performing iterations n to learn these downstream values. We introduce the construct of **approximated next-stage costs** (estimated downstream values) $\overline{V}_t^n(x_t^{a,n})$, replacing the standard expectation in Bellman's equations, see (14), with an approximation

$$\overline{V}_t^a(x_t^{x,n}) = \mathbb{E}^{\omega} \left\{ V_{t+1}(x_{t+1}^n | x_t^{a,n}, \omega) \right\}$$

We obtain the ADP forward optimality equations using the post-decision state and the approximated next-stage cost.

$$\hat{v}_t^n = \min_{a_t \in A} \left(c(x_t^n, a_t^n) + \gamma V_t^a \left(x^{M,x}(x_t^n, a_t^n) \right) \right)$$

$$\hat{a}_t^n = \arg\min_{a_t^n \in A} \left(c(x_t^n, a_t^n) + \gamma V_t^a \left(x^{M,x}(x_t^n, a_t^n) \right) \right)$$

Then the approximate dynamic programming algorithm would have the following steps:

- For each feasible decision a_t^n, obtain an associated post-decision state $x_t^{a,n}$
- The ADP forward optimality equations are solved first at stage $t = 0$ for an initial state x_0, and then for subsequent stages and states until the end of the horizon
- In each iteration n, a sample path $\omega^n \in \Omega$ (set of all sample paths) is drawn
- To advance "forward" in time, from stage t to $t+1$, the sample $W_{t+1}(\omega^n)$ (sample realization at time t using the sample path ω^n in iteration n) is used
- Update $\overline{V}_{t-1}^a(x_{t-1}^{a,n})$ immediately after the forward optimality equations are solved
- At stage t, the information arrives and decision taken in the new state x_t^n that incurs a cost. The approximated next-stage cost that was calculated at the previous stage $t-1$, $\overline{V}_{t-1}^a(x_{t-1}^{a,n})$, has now been observed at stage t.
- The algorithm updates this approximated next-stage cost of the previous post-decision state $x_{t-1}^{a,n}$ using the old approximation, i.e., $\overline{V}_{t-1}^a(x_{t-1}^{a,n})$ and the new approximation, i.e., the value $\hat{v}_t^n$ given by (18).
- U^V: the process "tuning" the approximating function:

$$\overline{V}_{t-1}^a(x_{t-1}^{a,n}) \leftarrow U^V\left(\overline{V}_{t-1}^a(x_{t-1}^{a,n}), x_{t-1}^{a,n}, \hat{v}_t^n\right)$$

1.2.4 Risk measures

Definition 1.2. For real-valued random variable $X \in L^1(\Omega, \mathcal{A}, \mathbb{P})$ defined on measurable space $(\Omega, \mathcal{A}, \mathbb{P})$, a **coherent risk measure** is defined to be a mapping $\rho : L^1 \to \mathbb{R}$ such that the following axioms hold [5]:

- Convexity: $\rho(\gamma X + (1-\gamma)Y) \leq \gamma\rho(X) + (1-\gamma)\rho(Y) \quad \forall \gamma \in (0,1), X, Y \in L^1$
- Monotonicity: if $X \leq Y$ $\mathbb{P}$-a.s. then $\rho(X) \leq \rho(Y) \quad \forall X, Y \in L^1$
- Translational invariance: $\rho(c + X) = c + \rho(X) \quad \forall c \in \mathbb{R}, X \in L^1$

- Homogeneity: $\rho(\beta X) = \beta\rho(X) \quad \forall X \in L^1, \beta \geq 0$

In this thesis, the risk-averse operator selected as the risk measure is Average-Value-at-Risk or $AVaR_\alpha(X)$. This parameter can be defined in two ways. First, we give the definition more common in the literature related to risk management.

Definition 1.3. For real-valued random variable $X \in L^1(\Omega, \mathcal{A}, \mathbb{P})$ defined on measurable space $(\Omega, \mathcal{A}, \mathbb{P})$ with finite mean (stating integrability with $E|X| < \infty$), at given risk level $\alpha \in (0,1)$ the following can be defined [3]:

- Value-at-Risk at risk level α:

$$VaR_\alpha(X) = inf\{x \in \mathbb{R} : \mathbb{P}(X \leq x) \geq \alpha\}$$

- Average-Value-at-Risk at risk level α:

$$AVaR_\alpha(X) = \frac{1}{1-\alpha}\int_\alpha^1 VaR_t(X)dt$$

This definition explains the core meaning of the Average-Value-at-Risk. $AVaR_\alpha(X)$ can be interpreted as average of Value-at-Risk's which are larger than the Value-at-Risk at risk level α. $AVaR_\alpha(X)$ gives the value for the losses greater than the given $VaR_\alpha(X)$ level. Computation is done through averaging, by the use of an integral.

However, the following lemma is used as the definition of the Average-Value-at-Risk in this thesis. It is then used to formulate the finite horizon problem.

Lemma 1.2. For real-valued random variable $X \in L^1(\Omega, \mathcal{A}, \mathbb{P})$ defined on measurable space $(\Omega, \mathcal{A}, \mathbb{P})$ with finite mean (stating integrability with $E|X| < \infty$), at given risk level $\alpha \in (0,1)$, the Average-Value-at-Risk can be defined as [5]:

$$AVaR_\alpha(X) = \min_{s\in\mathbb{R}}\{s + \frac{1}{1-\alpha}E[(X-s)^+]\}$$

with the minimum point $s^* = VaR_\alpha(X)$.

$AVaR_\alpha(X)$ indeed satisfies all the axioms stated in Definition 1.1. Furthermore, it has an interesting end behavior to be considered for computational reasons.

Remark 1.1. For the Average-Value-at-Risk $AVaR_\alpha(X)$ defined on real-valued random variable $X \in L^1(\Omega, \mathcal{A}, \mathbb{P})$, the following end behavior holds [3]:

- $\lim_{\alpha \to 0} \text{AVaR}_\alpha(X) = \mathbb{E}[X]$
- $\lim_{\alpha \to 1} \text{AVaR}_\alpha(X) = \text{ess sup} X \leq \infty$

1.2.5 Probability distributions

In this thesis, Bernoulli and exponential distributions are introduced as a noise term in a transition function. Each of these distributions has certain basic properties that need to be considered.

Bernoulli distribution

Bernoulli distribution is a special case of Binomial distribution corresponding to a single trial. It can be thought of as a result of a "yes-no" experiment with two outcomes. In this thesis, two possible outcomes are taken to be 1 with probability p and -1 with probability $1-p$, meaning that for $X \sim Bernoulli(p)$, $P(X=1)=p$ and $P(X=-1)=q=1-p$.

The probability mass function would be the following:

$$f(n,p) = \begin{cases} p & \text{if } k=1, \\ q = 1-p & \text{if } k=-1 \end{cases}$$

From this, the first two moments of the distribution can be derived. For $X \sim Bernoulli(p)$,

$E[X] = 1 \cdot p + (-1) \cdot (1-p) = p - 1 + p = 2p - 1$

$E[X^2] = 1^2 \cdot p + (-1)^2 \cdot (1-p) = p + 1 - p = 1.$

Exponential distribution

The exponential distribution is a probability distribution corresponding to a Poisson point process and this process' event distance. The Poisson point process can be understood as a process with an average constant rate, where the events are independent and continuous. Distribution has a parameter λ, corresponding to the rate of events.

λ can only take positive values so $\lambda > 0$ and for $X \sim Exponential(\lambda)$, $X \geq 0$ respectively.

The probability mass function would be the following:

$$f(x,\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0, \\ 0 & \text{if } x < 0 \end{cases}$$

Then, we can derive the first two moments of the distribution. For $X \sim Exponential(\lambda)$,

$E[X] = \int_0^{+\infty} x \cdot f_X(x)dx$

$= \int_0^{+\infty} x \cdot \lambda e^{-\lambda x}dx = \lambda \int_0^{+\infty} x \cdot e^{-\lambda x}$

Knowing the anti-derivative

$\int x \cdot \lambda e^{-\lambda x}dx = (-\frac{1}{\lambda}x - \frac{1}{\lambda^2})e^{-\lambda x}$,

$E[X] = \frac{1}{\lambda^2}\ \lambda[(-\frac{1}{\lambda}x - \frac{1}{\lambda^2})e^{-\lambda x}]_0^{+\infty}$

$= \lambda[\lim_{x\to+\infty}(-\frac{1}{\lambda}x - \frac{1}{\lambda^2})e^{-\lambda x}]_0^{+\infty} - (-\frac{1}{\lambda}\cdot 0 - \frac{1}{\lambda^2})e^{-\lambda\cdot 0}$

$= \lambda[0 + \frac{1}{\lambda^2}] = \frac{1}{\lambda}$.

The second moment can be derived similarly, so that

$E[X^2] = \frac{1}{\lambda^2}$.

1.2.6 Quantile function

The computation of the quantile s^* is done using the *numpy.quantile* function. The linear interpolation method is used.

For finding the new virtual index $i + g$ of an element lying between i and $i + 1$, the following formula is used for quantile $0 \leq q \leq 1$ with a sorted list of n elements:

$$i + g = q \cdot (n - \alpha - \beta + 1) + \alpha$$

For the linear interpolation, $\alpha = 1$ and $\beta = 1$. So, the formula becomes

$$i + g = q \cdot (n - 1) + 1$$

For example, for a list $[1, 2, 3, 4]$ and $q = 0.25$, the virtual index of an element corresponding to 25% quantile is:

$i + g = 0.25(4 - 1) + 1 = 1.75$

The 1.75th element lies between 1st element of 1 and 2nd element of 2. Finding the value of the 1.75th element:

$1 + (2 - 1) * 0.75 = 1.75$.

So, 25% quantile for a list $[1, 2, 3, 4]$ equals to 1.75.

1.3 Problem statement

The problem is defined using the dynamic programming definition given in Section 1.2.2.

In this problem, the objective function is the total cost. Since the optimization problem involves using some expected performance criteria, there is a need to use some risk-averse operator. In this thesis, $AVaR_\alpha(X)$ is used to measure the reward or cost dependent on risk level α [4].

We are interested in solving the optimal control problem over the infinite time horizon, meaning having not limited terminal time T, which can be equal to ∞.

So, using the AVaR criteria, the optimization problem is reformulated as follows:

$$\underset{\pi \in A}{\text{minimize}}\ AVaR_\alpha^\pi(\sum_{t=0}^{\infty} c(x_t, a_t))$$

The optimization problem to solve:

$\min\limits_{\pi \in A} AVaR_\alpha^\pi(\sum_{t=0}^{\infty} c(x_t, a_t))$
where $c(x_t, a_t)$ is the cost of taking action a_t at state x_t.

From this, the finite horizon problem can be formulated.

Theorem 1.1. The finite horizon problem for real-valued random variable $X \in L^1(\Omega, \mathcal{A}, \mathbb{P})$ defined on measurable space $(\Omega, \mathcal{A}, \mathbb{P})$ and risk level $\alpha \in (0, 1)$ is defined as follows:

$$\inf_{\pi \in A} AVaR_\alpha^\pi(X|X_0 = x) = \inf_{s \in \mathbb{R}} \{s + \frac{1}{1-\alpha} \inf_{\pi \in A} E_x^\pi[(X - s)^+]\}$$

Proof. Using the **Lemma 1.2**,

$$\begin{aligned}\inf_{\pi \in A} AVaR_\alpha^\pi(X|X_0 = x) &= \inf_{\pi \in A} \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-\alpha} E_x^\pi[(X - s)^+]\} \\ &= \inf_{s \in \mathbb{R}} \inf_{\pi \in A} \{s + \tfrac{1}{1-\alpha} E_x^\pi[(X - s)^+]\} \\ &= \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-\alpha} \inf_{\pi \in A} E_x^\pi[(X - s)^+]\}.\end{aligned}$$

Therefore, the finite horizon problem is formulated.

Chapter 2

Literature review

There is an abundance of different papers about risk-averse optimization problems and solving optimal control problems and LQR problems.

The research heavily relies on the idea of optimal control and [1] fully explains the concepts behind optimal control problems and approaches to solving them. This paper is key to seeing different methods of solving the optimal control problems analytically, using different approaches, such as HJB equations, Lagrange multipliers, or gradient descent method.

However, there are two papers this thesis heavily relies on. First is Bauerle's work in [5] giving a new definition for the Average-Value-at-Risk, serving as a basis for the computations in this thesis. In [3], a paper written by my supervisor on this topic gives all the fundamental knowledge, theorems, and derivations needed to solve the optimal control problem analytically. Most of the theoretical background was taken from this work. On top of that, [2] gives insights into the idea of dynamic programming and explains the simplest problems in dynamic programming and [9] provides the definition and explanation for the Markov decision process concept.

The meaning of risk measures is defined, and the definition and explanation for risk measures on the example of the Entropic-Value-at-Risk risk measure is provided in [4]. Furthermore, it gives an example of solving optimization problems involving EVaR that can be redone with AVaR. Similarly, [10] solves the risk-sensitive optimization problem but with Conditional-Value-at-Risk, which can also be referred to as an example.

Most literature on optimal control problems mainly uses neural networks. [6] and [7] are sources giving fresh ideas on solving LQR problems using neural networks. The first paper gives an intuition of solving the optimal control problems using neural networks built on the relationship between actions and states and provides implementation examples on real data such as portfolio optimization. The other paper describes using Q-learning to solve optimal control problems with risk measures of CVaR similar to EVaR. The implementation provided in this paper can be reused for AVaR.

The majority of the literature does not focus on LQR problems specifically but elaborates on optimal control problems in general. Even in cases with mentioning the LQR problem, it is given in different formulation, mostly in matrix form as in [12], [13] and [14] opposed to [3] stating the formulation of the LQR problem similar to my thesis problem.

In terms of methodology, Out of the collected literature, [15] was the most relevant to the research, stating the approximate dynamic programming algorithm with an example of the nomadic trucker. The algorithm presented in this paper was reformulated and changed to fit the selected LQR problem.

[8] is not as relevant as other papers, but gives some new ideas about analytical solutions to optimal control problems considering the time, state and actions as some nonlinear dynamical system.

Chapter 3

Analytic solution

3.1 Hamiltonian-Jacobi-Bellman equations

To solve the optimal control problems, Bellman's optimality principle is used [8]. This principle states that there is no dependence of future states from past states resulting in the present state. From this, the computations are started at a final state and move backward performing calculations step-by-step. This idea is realized through the concept of "optimal value function", meaning the minimum optimal total cost for a path started at some state.

The optimal value function is given through the following HJB equation [5] for (t, x) where $0 \leq t \leq T$:

$$
\begin{aligned}
Q^{\pi}(t,x) &\triangleq c(x,a,t) + Q(t+1, x+a+\xi) \\
Q^{\pi}(T,x) &\triangleq g(x) \\
V(t,x) &= \inf_{\pi} Q^{\pi}(t,x)
\end{aligned}
$$

Through the iterations, each value function is obtained. Using the value function, the optimization problem will be as follows:

$$\min_{a} AVaR_{\alpha}(Q^{\pi}(t,x)|\mathcal{F}_T)$$

for some history or information given by σ-algebra $\mathcal{F}_T$ and terminal cost $g(t)$ at terminal time T.

3.2 LQR Problem

For the Linear Quadratic Regulator (LQR) Problem, the following conditions are given:

- Transition (linear) function: $x_{t+1} = x_t + a_t + \xi_t$ for the **random noise** ξ_t
- Running (quadratic) cost: $c_t(x, a, t) = x_t^2 + a_t^2$
- Total cost: $\sum_{s=t}^{s=T} c(x_s, a_s)$, where $\pi(s, x) = a_s$, for $s = t, ..., T$.

At terminal time $t = T$, the total cost would be $c_T(x_T, a_T)$.

Using the equation above, the total cost up to each rime from $t = T$ to $t = 0$ is computed. We are also given **history** $\mathcal{F}_T$.

The optimization problem is formulated as a minimization of a total cost:

$$\min_{a_t} AVaR_\alpha(Q^\pi(T, x)|\mathcal{F}_T)$$

which is equivalent to

$$\min_{a_T \in A} AVaR_\alpha(c_T(x_T, a_T))$$

Using the dynamic programming approach, computations are done from $t = T$ to $t = 0$.

$$\min_{a_t} AVaR_\alpha(Q^\pi(T, x)|\mathcal{F}_T) = \min_{a_t} c_t(x_t, a_t) + AVaR_\alpha(Q^\pi(t+1, x_{t+1})|\mathcal{F}_T)$$

The minimal AVaR value is obtained by iterating through the possible action values, the procedure is repeated until $t = 0$.

In this problem, the randomness of x_{t+1} caused by random noise ξ_t is a key factor. The effect of a distribution of a random noise has to be investigated thoroughly. The simplest case with a Bernoulli noise is considered first, then the problem with exponential noise is studied carefully afterwards.

3.3 LQR Problem with Bernoulli Noise

The thesis work takes the LQR problem with Bernoulli noise as a baseline setting, where

- $\xi_t \sim Bernoulli(p = 1/2)$
- $\alpha \in \{0, 0.25, 0.5, 0.75, 0.99\}$
- $a_t \in \{-1, -0.5, 0, 0.5, 1\}$
- $x_0 = 1, t_0 = 0$
- $T = 2.$

The problem now is about computing AVaR for given $\phi(a_t, \xi_t)$ and set of values. For this problem, the dynamic programming approach with backward computations is also applied. $\xi_t \sim Bernoulli(p = 1/2)$ means that $\xi_t = 1$ with probability $p = \frac{1}{2}$ and $\xi_t = -1$ with probability $p = \frac{1}{2}$.

We start the computations at time $t = 2$, then go backwards in time in 1 time unit steps.

Step 1. t = 2

$J(2, x_2) = \inf_{a_2} E[x_2^2 + a_2^2 | x_2, a_2] = x_2^2.$

Therefore, the minimizing action $a_2 = 0$.

Step 2. t = 1

$$\begin{aligned}
J(1, x_1) &= \inf_{a_1} E[x_1^2 + a_1^2 + J(2, x_2) | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + (x_1 + a_1 + \xi_1)^2 | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + x_1^2 + a_1^2 + \xi_1^2 + 2x_1a_1 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1] \\
&= \inf_{a_1} E[2x_1^2 + 2a_1^2 + \xi_1^2 + 2x_1a_1 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1] \\
&= 2x_1^2 + \inf_{a_1} \{2a_1^2 + 2x_1a_1 + E[\xi_1^2 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1]\} \\
&= 2x_1^2 + 2x_1E[\xi_1 | x_1, a_1] + E[\xi_1^2 | x_1, a_1] + \inf_{a_1} \{2a_1^2 + 2x_1a_1 + 2a_1E[\xi_1 | x_1, a_1]\}
\end{aligned}$$

Given that $\xi_1 \sim Bernoulli(p = 1/2)$, $E[\xi_1 | x_1, a_1]$ and $E[\xi_1^2 | x_1, a_1]$ can be computed.

$E[\xi_1|x_1, a_1] = 1(\frac{1}{2}) + (-1)(\frac{1}{2}) = 0$

$E[\xi_1^2|x_1, a_1] = 1^2(\frac{1}{2}) + (-1)^2(\frac{1}{2}) = \frac{1}{2} + \frac{1}{2} = 1.$

Then,

$J(1, x_1) = 2x_1^2 + 1 + 2\inf_{a_1}\{a_1^2 + x_1 a_1\}.$

Meaning that

$\phi(a_1) = a_1^2 + x_1 a_1$

$a_1 = -1 : \phi(-1) = 1 - x_1$

$a_1 = -0.5 : \phi(-1) = 0.25 - 0.5x_1$

$a_1 = 0 : \phi(0) = 0$

$a_1 = 0.5 : \phi(-1) = 0.25 + 0.5x_1$

$a_1 = 1 : \phi(1) = 1 + x_1$

The behavior of $\phi(a_1)$ was checked graphically. In the plot below we deduce the the piecewise function giving the minimum value at each interval.

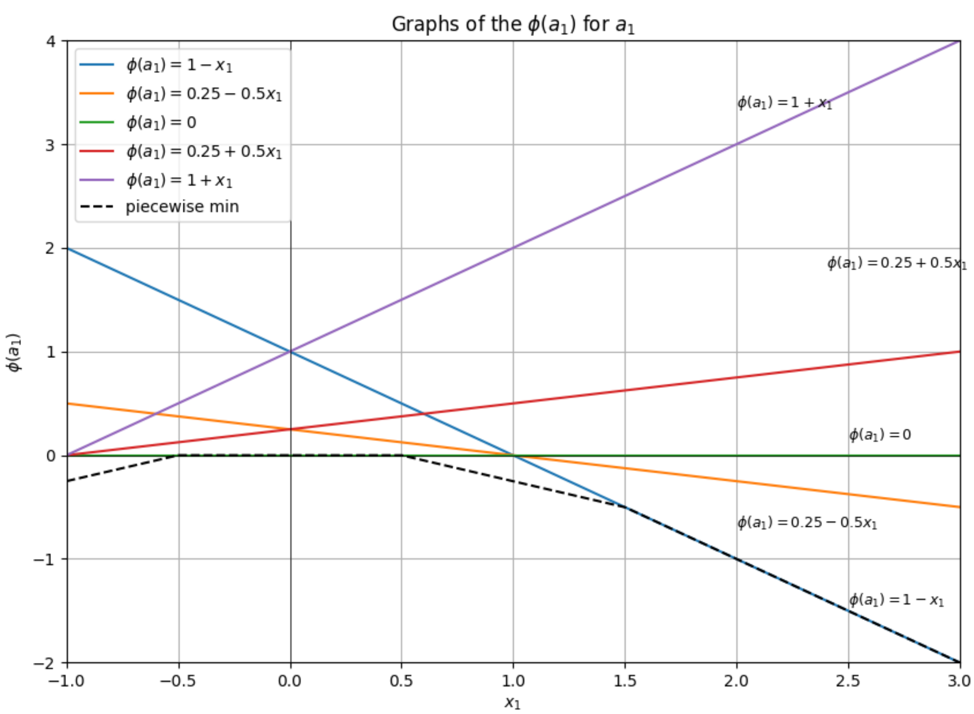

Figure 3.1. Obtaining the minimizing piecewise function

So, the piecewise function giving the minimum value at each interval is

$$\phi(a_1) = \begin{cases} 0.25 + 0.5x_1, & x_1 \in [-1, -0.5) \\ 0, & x_1 \in [-0.5, 0.5) \\ 0.25 - 0.5x_1, & x_1 \in [0.5, 1.5) \\ 1 - x_1 & x_1 \in [1.5, 3] \end{cases}$$

So, each interval will be considered as a separate case.

Step 3. $\mathbf{t = 0}$

We obtain the equations for $J(1, x_1)$ for some given x_1 in certain interval.

Case 1. $\mathbf{x_1 \in [-1, -0.5)}$.

In this case, the optimizing action $a_1 = 0.5$.

So,

$$J(1, x_1) = 2x_1^2 + 1 + 2(0.25 + 0.5x_1).$$
$$= 2x_1^2 + x_1 + 1.5.$$

Case 2. $\mathbf{x_1 \in [-0.5, 0.5)}$.

In this case, the optimizing action $a_1 = 0$.

So,

$$J(1, x_1) = 2x_1^2 + 1 + 2 \times 0.$$
$$= 2x_1^2 + 1.$$

Case 3. $\mathbf{x_1 \in [0.5, 1.5)}$.

In this case, the optimizing action $a_1 = -0.5$.

So,

$$J(1, x_1) = 2x_1^2 + 1 + 2(0.25 - 0.5x_1).$$
$$= 2x_1^2 - x_1 + 1.5.$$

Case 4. $\mathbf{x_1 \in [1, 3]}$.

In this case, the optimizing action $a_1 = -1$.

So,

$$J(1, x_1) = 2x_1^2 + 1 + 2(1 - x_1).$$
$$= 2x_1^2 - 2x_1 + 3.$$

Therefore,

$$J(1,x_1)=\begin{cases}2x_1^2+x_1+1.5, & x_1\in[-1,-0.5)\\ 2x_1^2+1, & x_1\in[-0.5,0.5)\\ 2x_1^2-x_1+1.5, & x_1\in[0.5,1.5)\\ 2x_1^2-2x_1+3 & x_1\in[1.5,3]\end{cases}$$

For further computations for $J(0,x_0)$, it has to be noted that each noise was sampled randomly, so we know that each of the noises has equal probabilities.

Knowing that $x_1 = x_0 + a_0 + \xi_0$ and $x_0 = 1$, $a_0 \in \{-1,-0.5,0,0.5,1\}$, $\xi_t \in \{1,-1\}$, different cases should be considered. Each a_0 case will be considered separately given that $x_0 = 1$ is fixed.

Case a. $\mathbf{a_0 = -1}$

$$\begin{aligned}J(0,x_0,a_0=-1) &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_1)|x_0=1,a_0=-1]\\ &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_0+a_0+\xi_0)|x_0=1,a_0=-1]\\ &= x_0^2+a_0^2+(1/2)(J(1,1-1+1)+J(1,1-1-1)\\ &= x_0^2+a_0^2+(1/2)(J(1,1)+J(1,-1))\\ &= 1^2+(-1)^2+(1/2)(2.5+2.5)\\ &= 4.5\end{aligned}$$

Case b. $\mathbf{a_0 = -0.5}$

$$\begin{aligned}J(0,x_0,a_0=0) &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_1)|x_0=1,a_0=-0.5]\\ &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_0+a_0+\xi_0)|x_0=1,a_0=-0.5]\\ &= x_0^2+a_0^2+(1/2)(J(1,1-0.5+1)+J(1,1-0.5-1))\\ &= x_0^2+a_0^2+(1/2)(J(1,1.5)+J(1,-0.5))\\ &= 1^2+(-0.5)^2+(1/2)(4.5+1.5)\\ &= 4.25\end{aligned}$$

Case c. $\mathbf{a_0 = 0}$

$$\begin{aligned}J(0,x_0,a_0=0) &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_1)|x_0=1,a_0=0]\\ &= \inf_{a_0} E[x_0^2+a_0^2+J(1,x_0+a_0+\xi_0)|x_0=1,a_0=0]\\ &= x_0^2+a_0^2+(1/2)(J(1,1+0+1)+J(1,1+0-1))\\ &= x_0^2+a_0^2+(1/2)(J(1,2)+J(1,0))\end{aligned}$$

$$= 1^2 + 0^2 + (1/2)(7 + 1)$$

$$= 5$$

Case d. $\mathbf{a_0 = 0.5}$

$$J(0, x_0, a_0 = 0) = \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0 = 1, a_0 = 0.5]$$

$$= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_0 + a_0 + \xi_0)|x_0 = 1, a_0 = 0.5]$$

$$= x_0^2 + a_0^2 + (1/2)(J(1, 1 + 0.5 + 1) + J(1, 1 + 0.5 - 1))$$

$$= x_0^2 + a_0^2 + (1/2)(J(1, 2.5) + J(1, 0.5))$$

$$= 1^2 + 0.5^2 + (1/2)(10.5 + 1.5)$$

$$= 7.25$$

Case e. $\mathbf{a_0 = 1}$

$$J(0, x_0, a_0 = 0) = \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0 = 1, a_0 = 1]$$

$$= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_0 + a_0 + \xi_0)|x_0 = 1, a_0 = 1]$$

$$= x_0^2 + a_0^2 + (1/2)(J(1, 1 + 1 + 1) + J(1, 1 + 1 - 1))$$

$$= x_0^2 + a_0^2 + (1/2)(J(1, 3) + J(1, 2))$$

$$= 1^2 + 1^2 + (1/2)(15 + 7)$$

$$= 13$$

After considering all these cases for a_0 values, we deduce the final $J(0, x_0)$.

$$J(0, x_0) = \inf_{a_0 \in \{-1, -0.5, 0, 0.5, 1\}} J(0, x_0)$$

$$= \inf_{a_0 \in \{-1, 0, 1\}} \{J(0, x_0, a_0 = -1), J(0, x_0, a_0 = -0.5), J(0, x_0, a_0 = 0),$$

$$J(0, x_0, a_0 = 0.5), J(0, x_0, a_0 = 1)\}$$

$$= \inf\{4.5, 4.25, 5, 7.25, 13\}$$

$$= 4.25$$

So the final answer for $\alpha = 0$ is $\mathbf{J(0, x_0) = 4.25}$.

3.4 LQR Problem with Exponential Noise

3.4.1 LQR Problem with Theoretical Exponential Noise at risk level $\alpha = 0$

Having made observations from the case with Bernoulli noise, the problem can be solved for the noise coming from Exponential distribution. The problem setting remains similar except for the noise term and the set of actions.

The noise $\xi_t \sim Exponential(\lambda)$, for $\lambda > 0$ is used. Calculations are made for $\lambda \in \{0.5, 1, 1.5, 2\}$. Computation with $\lambda = 1$ is taken as a baseline. The action set is changed to be $a_t \in \{-1, 0, 1\}$ due to the dimensionality problem, to decrease the number of possible state x_t values for $t \geq 1$. Terminal time is taken to be $T = 2$.

Step 1. t = 2

$J(2, x_2) = \inf_{a_2} E[x_2^2 + a_2^2 | x_2, a_2] = x_2^2.$

Therefore, the minimizing action $a_2 = 0$.

Step 2. t = 1

$$
\begin{aligned}
J(1, x_1) &= \inf_{a_1} E[x_1^2 + a_1^2 + J(2, x_2) | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + (x_1 + a_1 + \xi_1)^2 | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + x_1^2 + a_1^2 + \xi_1^2 + 2x_1 a_1 + 2x_1 \xi_1 + 2a_1 \xi_1 | x_1, a_1] \\
&= \inf_{a_1} E[2x_1^2 + 2a_1^2 + \xi_1^2 + 2x_1 a_1 + 2x_1 \xi_1 + 2a_1 \xi_1 | x_1, a_1] \\
&= 2x_1^2 + \inf_{a_1} \{2a_1^2 + 2x_1 a_1 + E[\xi_1^2 + 2x_1 \xi_1 + 2a_1 \xi_1 | x_1, a_1]\} \\
&= 2x_1^2 + 2x_1 E[\xi_1 | x_1, a_1] + E[\xi_1^2 | x_1, a_1] + \inf_{a_1} \{2a_1^2 + 2x_1 a_1 + 2a_1 E[\xi_1 | x_1, a_1]\}
\end{aligned}
$$

Given that $\xi_1 \sim Exponential(\lambda)$, $E[\xi_1 | x_1, a_1]$ and $E[\xi_1^2 | x_1, a_1]$ can be computed.

$E[\xi_1 | x_1, a_1] = \frac{1}{\lambda}$

$E[\xi_1^2 | x_1, a_1] = Var[\xi_1 | x_1, a_1] + E^2[\xi_1 | x_1, a_1] = \frac{1}{\lambda^2} + \frac{1}{\lambda^2} = \frac{2}{\lambda^2}.$

Then,

$J(1, x_1) = 2x_1^2 + \frac{2}{\lambda} x_1 + \frac{2}{\lambda^2} + 2 \inf_{a_1} \{a_1^2 + (x_1 + \frac{1}{\lambda}) a_1\}.$

Meaning that

$\phi(a_1) = a_1^2 + (x_1 + \frac{1}{\lambda}) a_1$

$a_1 = -1 : \phi(-1) = 1 - x_1 - \frac{1}{\lambda}$

$a_1 = 0 : \phi(0) = 0$

$a_1 = 1 : \phi(1) = 1 + x_1 + \frac{1}{\lambda}$

The behavior of $\phi(a_1)$ was checked for several λ values graphically. Different from Bernoulli noise, only two cases were identified for further computations. Here we use the fact that $E[\xi_1|x_1, a_1] = \frac{1}{\lambda}$

Step 3. t = 0

Case 1. $\lambda < \mathbf{1}$ **OR**

$\lambda \geq \mathbf{1}$ and $\mathbf{x} \geq \mathbf{1} - \mathbf{E}[\xi_{\mathbf{1}}|\mathbf{x_1}, \mathbf{a_1}]$.

In this case, the optimizing action $\mathbf{a_1} = -\mathbf{1}$.

So,

$$J(1, x_1) = 2x_1^2 + \frac{2}{\lambda}x_1 + \frac{2}{\lambda^2} + 2(1 - x_1 - \frac{1}{\lambda}).$$

$$= 2x_1^2 + 2(\frac{1}{\lambda} - 1)x_1 + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2}).$$

$$J(0, x_0) = \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0, a_0]$$

$$= \inf_{a_0} E[x_0^2 + a_0^2 + 2(x_0 + a_0 + \xi_0)^2 + 2(\frac{1}{\lambda} - 1)(x_0 + a_0 + \xi_0) + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2})|x_0, a_0]$$

Knowing that $x_0 = 1$,

$$J(0, x_0) = \inf_{a_0} E[1 + a_0^2 + 2(1 + a_0 + \xi_0)^2 + 2(\frac{1}{\lambda} - 1)(1 + a_0 + \xi_0) + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2})|a_0]$$

$$= \inf_{a_0} E[1 + a_0^2 + 2 + 2a_0^2 + 2\xi_0^2 + 4a_0 + 4\xi_0 + 4a_0\xi_0 + 2(\frac{1}{\lambda} - 1) + 2(\frac{1}{\lambda} - 1)a_0$$

$$+2(\frac{1}{\lambda} - 1)\xi_0 + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2})|a_0]$$

$$= 3 + 2(\frac{1}{\lambda} - 1) + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2}) + 2(\frac{1}{\lambda} + 1)E[\xi_0|a_0] + 2E[\xi_0^2|a_0] + \inf_{a_0}\{3a_0^2$$

$$+2(\frac{1}{\lambda} + 1)a_0 + 4a_0 E[\xi_0|a_0]\}$$

We know that $E[\xi_0|x_0, a_0] = \frac{1}{\lambda}$ and $E[\xi_0^2|x_0, a_0] = \frac{2}{\lambda^2}$.

$$J(0, x_0) = 3 + 2(\frac{1}{\lambda} - 1) + 2(1 - \frac{1}{\lambda} + \frac{1}{\lambda^2}) + 2(\frac{1}{\lambda} + 1)(\frac{1}{\lambda}) + \frac{4}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(\frac{3}{\lambda} + 1)a_0\}$$

$$= 3 + \frac{2}{\lambda} - 2 + 2 - \frac{2}{\lambda} + \frac{2}{\lambda^2} + \frac{2}{\lambda^2} + \frac{2}{\lambda} + \frac{4}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(\frac{3}{\lambda} + 1)a_0\}$$

$$= 3 + \frac{2}{\lambda} + \frac{8}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(\frac{3}{\lambda} + 1)a_0\}$$

Meaning that

$$\phi(a_0) = 3a_0^2 + 2(\frac{3}{\lambda} + 1)a_0$$

$a_1 = -1 : \phi(-1) = 3 - 2(\frac{3}{\lambda} + 1) = 1 - \frac{6}{\lambda}$

$a_1 = 0 : \phi(0) = 0$

$a_1 = 1 : \phi(1) = 3 + 2(\frac{3}{\lambda} + 1) = 4 + \frac{6}{\lambda}$

Knowing that $\lambda > 0$, the minimizing action is $\mathbf{a_0 = -1}$.

With $\mathbf{J(0, x_0)} = 3 + \frac{2}{\lambda} + \frac{8}{\lambda^2} + 1 - \frac{6}{\lambda} = \mathbf{4 - \frac{4}{\lambda} + \frac{8}{\lambda^2}}$.

Case 2. $\boldsymbol{\lambda \geq 1}$ **and** $\mathbf{x \in [0, 1 - E[\xi_1 | x_1, a_1])}$

In this case, the optimizing action $\mathbf{a_1 = 0}$.

$$J(1, x_1) = 2x_1^2 + \tfrac{2}{\lambda}x_1 + \tfrac{2}{\lambda^2} + 2 \times 0$$
$$= 2x_1^2 + \tfrac{2}{\lambda}x_1 + \tfrac{2}{\lambda^2}$$
$$J(0, x_0) = \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1) | x_0, a_0]$$
$$= \inf_{a_0} E[x_0^2 + a_0^2 + 2(x_0 + a_0 + \xi_0)^2 + \tfrac{2}{\lambda}(x_0 + a_0 + \xi_0) + \tfrac{2}{\lambda^2} | x_0, a_0]$$

Knowing that $x_0 = 1$,

$$J(0, x_0) = \inf_{a_0} E[1 + a_0^2 + 2(1 + a_0 + \xi_0)^2 + \tfrac{2}{\lambda}(1 + a_0 + \xi_0) + \tfrac{2}{\lambda^2} | a_0]$$
$$= \inf_{a_0} E[1 + a_0^2 + 2 + 2a_0^2 + 2\xi_0^2 + 4a_0 + 4\xi_0 + 4a_0\xi_0 + \tfrac{2}{\lambda} + \tfrac{2}{\lambda}a_0 + \tfrac{2}{\lambda}\xi_0 + \tfrac{2}{\lambda^2} | a_0]$$
$$= 3 + \tfrac{2}{\lambda} + \tfrac{2}{\lambda^2} + 2(2 + \tfrac{1}{\lambda})E[\xi_0 | a_0] + 2E[\xi_1^2 | a_0] + \inf_{a_0}\{3a_0^2 + 2(2 + \tfrac{1}{\lambda})a_0$$
$$+ 4a_0 E[\xi_0 | a_0]\}$$

We know that $E[\xi_0 | x_0, a_0] = \frac{1}{\lambda}$ and $E[\xi_0^2 | x_0, a_0] = \frac{2}{\lambda^2}$.

$$J(0, x_0) = 3 + \tfrac{2}{\lambda} + \tfrac{2}{\lambda^2} + 2(2 + \tfrac{1}{\lambda})(\tfrac{1}{\lambda}) + \tfrac{4}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(2 + \tfrac{3}{\lambda})a_0\}$$
$$= 3 + \tfrac{2}{\lambda} + \tfrac{2}{\lambda^2} + \tfrac{4}{\lambda} + \tfrac{2}{\lambda^2} + \tfrac{4}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(2 + \tfrac{3}{\lambda})a_0\}$$
$$= 3 + \tfrac{6}{\lambda} + \tfrac{8}{\lambda^2} + \inf_{a_0}\{3a_0^2 + 2(2 + \tfrac{3}{\lambda})a_0\}$$

Meaning that

$$\phi(a_0) = 3a_0^2 + 2(\tfrac{3}{\lambda} + 2)a_0$$

$a_1 = -1 : \phi(-1) = 3 - 2(\frac{3}{\lambda} + 2) = -1 - \frac{6}{\lambda}$

$a_1 = 0 : \phi(0) = 0$

$a_1 = 1 : \phi(1) = 3 + 2(\frac{3}{\lambda} + 2) = 7 + \frac{6}{\lambda}$

Knowing that $\lambda > 0$, the minimizing action is $\mathbf{a_0 = -1}$.

With $\mathbf{J(0, x_0)} = 3 + \frac{6}{\lambda} + \frac{8}{\lambda^2} - 1 - \frac{6}{\lambda} = \mathbf{2 + \frac{8}{\lambda^2}}$.

3.4.2 LQR Problem with Sampled Exponential Noise at risk level $\alpha = 0$

Using the theoretical exponential noise for practical or experimental settings is difficult to do realistically. By the Law of Large Numbers, for a sample generated from an experiment, we should maximize the sample size for the sample's distribution to converge to the original distribution. However, this would require taking large samples creating infinitely many possibilities for states. For example, having taken 3 samples for noise distribution and having 3 possible actions, at each time t, there would be 9^t possible states.

To address this concern, it was attempted to approximate the exponential distribution by taking 6 samples for noise term. That means, there would be 18 possible x_1 values. However, having 324 possible x_2 values has no effect since time $t = 2$ is terminal.

numpy.random.exponential was used to take 6 random samples from the exponential distribution with $\lambda = 1.0$ and some fixed random seed of 41 for the reproducibility of the experiment.

We are now solving the LQR problem for

- $\xi_t \in \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\}$
- $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$
- $a_t \in \{-1, 0, 1\}$
- $x_0 = 1, t_0 = 0$
- $T = 2$.

Step 1. t = 2

$J(2, x_2) = \inf_{a_2} E[x_2^2 + a_2^2 | x_2, a_2] = x_2^2.$

Therefore, the minimizing action $a_2 = 0$.

Step 2. t = 1

$$
\begin{aligned}
J(1, x_1) &= \inf_{a_1} E[x_1^2 + a_1^2 + J(2, x_2) | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + (x_1 + a_1 + \xi_1)^2 | x_1, a_1] \\
&= \inf_{a_1} E[x_1^2 + a_1^2 + x_1^2 + a_1^2 + \xi_1^2 + 2x_1a_1 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1] \\
&= \inf_{a_1} E[2x_1^2 + 2a_1^2 + \xi_1^2 + 2x_1a_1 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1] \\
&= 2x_1^2 + \inf_{a_1} \{2a_1^2 + 2x_1a_1 + E[\xi_1^2 + 2x_1\xi_1 + 2a_1\xi_1 | x_1, a_1]\}
\end{aligned}
$$

$$= 2x_1^2 + 2x_1 E[\xi_1|x_1, a_1] + E[\xi_1^2|x_1, a_1] + 2\inf_{a_1}\{a_1^2 + x_1 a_1 + a_1 E[\xi_1|x_1, a_1]\}$$

Now that we have the samples from an exponential distribution, the expected value is taken to be the sample mean and the sample mean squared replaces the second moment. So,

$E[\xi_1|x_1, a_1] = \frac{1}{6}(0.04 + 0.05 + 0.12 + 0.29 + 0.93 + 1.13) = 0.43$

$E[\xi_1^2|x_1, a_1] = \frac{1}{6}(0.04^2 + 0.05^2 + 0.12^2 + 0.29^2 + 0.93^2 + 1.13^2) = 0.37.$

Now we have

$J(1, x_1) = 2x_1^2 + 0.86x_1 + 0.37 + 2\inf_{a_1}\{a_1^2 + (x_1 + 0.43)a_1\}$

Meaning that

$\phi(a_1) = a_1^2 + (x_1 + 0.43)a_1$

$a_1 = -1 : \phi(-1) = 1 - x_1 - 0.43 = -x_1 + 0.57$

$a_1 = 0 : \phi(0) = 0$

$a_1 = 1 : \phi(1) = 1 + x_1 + 0.43 = x_1 + 1.43$

From the theoretical noise example, we know that there are two cases given that $\lambda = 1.0$. Note that $E[\xi_1|x_1, a_1] = 0.43$. This will be used to obtain the equations for $J(1, x_1)$ for some given x_1.

Case 1. $\mathbf{x_1 \geq 0.57}$.

In this case, the optimizing action $a_1 = -1$.

So,

$J(1, x_1) = 2x_1^2 + 0.86x_1 + 0.37 + 2(-x_1 + 0.57).$

$= 2x_1^2 - 1.14x_1 + 1.51.$

Case 2. $\mathbf{x_1 < 0.57}$.

In this case, the optimizing action $a_1 = 0$.

So,

$J(1, x_1) = 2x_1^2 + 0.86x_1 + 0.37 + 2 \times 0.$

$= 2x_1^2 + 0.86x_1 + 0.37.$

For further computations for $J(0, x_0)$, it has to be noted that each noise was sampled randomly, so we know that each of the noises has equal probabilities.

Knowing that $x_1 = x_0 + a_0 + \xi_0$ and $x_0 = 1$, $a_0 \in \{-1, 0, 1\}$, $\xi_t \in \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\}$, different cases should be considered. Each a_0 case will be considered separately given that $x_0 = 1$ is fixed.

Case a. $\mathbf{a_0 = -1}$

$$\begin{aligned}
J(0, x_0, a_0 = -1) &= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0 = 1, a_0 = -1] \\
&= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_0 + a_0 + \xi_0)|x_0 = 1, a_0 = -1] \\
&= x_0^2 + a_0^2 + (1/6)(J(1, 1 - 1 + 0.04) + J(1, 1 - 1 + 0.05) \\
&+ J(1, 1 - 1 + 0.12) + J(1, 1 - 1 + 0.29) + J(1, 1 - 1 + 0.93) \\
&+ J(1, 1 - 1 + 1.13) \\
&= x_0^2 + a_0^2 + (1/6)(J(1, 0.04) + J(1, 0.05) + J(1, 0.12) + J(1, 0.29) \\
&+ J(1, 0.93) + J(1, 1.13))] \\
&= 1^2 + (-1)^2 + (1/6)(0.41 + 0.42 + 0.50 + 0.79 + 2.18 + 2.78) \\
&= 3.18
\end{aligned}$$

Case b. $\mathbf{a_0 = 0}$

$$\begin{aligned}
J(0, x_0, a_0 = 0) &= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0 = 1, a_0 = 0] \\
&= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_0 + a_0 + \xi_0)|x_0 = 1, a_0 = 0] \\
&= x_0^2 + a_0^2 + (1/6)(J(1, 1.04) + J(1, 1.05) + J(1, 1.12) + J(1, 1.29) \\
&+ J(1, 1.93) + J(1, 2.13)) \\
&= 1^2 + 0^2 + (1/6)(2.49 + 2.52 + 2.74 + 3.37 + 6.76 + 8.16) \\
&= 5.34
\end{aligned}$$

Case c. $\mathbf{a_0 = 1}$

$$\begin{aligned}
J(0, x_0, a_0 = 1) &= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_1)|x_0 = 1, a_0 = 1] \\
&= \inf_{a_0} E[x_0^2 + a_0^2 + J(1, x_0 + a_0 + \xi_0)|x_0 = 1, a_0 = 1] \\
&= x_0^2 + a_0^2 + (1/6)(J(1, 2.04) + J(1, 2.05) + J(1, 2.12) + J(1, 2.29) \\
&+ J(1, 2.93) + J(1, 3.13))] \\
&= 1^2 + 1^2 + (1/6)(7.51 + 7.58 + 8.08 + 9.39 + 15.34 + 17.54) \\
&= 12.91
\end{aligned}$$

After considering all these cases for a_0 values, we deduce the final $J(0, x_0)$.

$$\begin{aligned}
J(0, x_0) &= \inf_{a_0 \in \{-1,0,1\}} J(0, x_0) \\
&= \inf_{a_0 \in \{-1,0,1\}} \{J(0, x_0, a_0 = -1), J(0, x_0, a_0 = 0), J(0, x_0, a_0 = 1)\}
\end{aligned}$$

$$= \inf\{3.18, 5.34, 12.91\}$$
$$= 3.18$$

So the final answer for $\alpha = 0$ is $J(0, x_0) = 3.18$.

3.4.3 LQR Problem with Sampled Exponential Noise at risk level $\alpha = 0.25$

From **Remark 1.1**, we know that with a risk level $\alpha = 0$, $AVaR_\alpha(X) = \mathbb{E}[X]$. The behavior of $AVaR_\alpha(X)$ for $\alpha \neq 0$ should be investigated next. For this, we take an example $\alpha = 0.25$.

For $\alpha = 0.25$, we are solving the similar LQR problem with

- $\xi_t \in \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\}$
- $a_t \in \{-1, 0, 1\}$
- $x_0 = 1, t_0 = 0$
- $T = 2$.

Step 1. t = 2

$J(2, x_2) = \inf_{a_2} AVaR_{0.25}[x_2^2 + a_2^2 | x_2, a_2] = \inf_{a_2} AVaR_{0.25}[x_2^2 + a_2^2] = x_2^2$.

Therefore, the minimizing action $a_2 = 0$.

Step 2. t = 1

$$J(1, x_1) = \inf_{a_1} AVaR_{0.25}[x_1^2 + a_1^2 + J(2, x_2) | x_1, a_1]$$
$$= \inf_{a_1} AVaR_{0.25}[x_1^2 + a_1^2 + (x_1 + a_1 + \xi_1)^2 | x_1, a_1]$$

Then, we use the **Theorem 1.1**.

$$J(1, x_1) = \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-0.25} \inf_{a_1} E[(x_1^2 + a_1^2 + (x_1 + a_1 + \xi_1)^2 - s)^+ | x_1, a_1]\}$$
$$= x_1^2 + a_1^2 + \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-0.25} \inf_{a_1} E[((x_1 + a_1 + \xi_1)^2 - s)^+ | x_1, a_1]\}$$
$$= x_1^2 + a_1^2 + s^* + \tfrac{1}{1-0.25} \inf_{a_1} E[((x_1 + a_1 + \xi_1)^2 - s^*)^+ | x_1, a_1]$$

We have to find the quantile s^*.

It's known that $s^* \triangleq VaR_\alpha((x_1 + a_1 + \xi_1)^2)) = inf\{x \in \mathbb{R} : \mathbb{P}((x_1 + a_1 + \xi_1)^2 \leq x) \geq \alpha\}$.

For this purpose, all possible x_1 values are computed to consider each case of possible x_1 and a_1 value pairs.

x_0	a_0	ξ_0	x_1
1	-1	0.04	0.04
1	-1	0.05	0.05
1	-1	0.12	0.12
1	-1	0.29	0.29
1	-1	0.93	0.93
1	-1	1.13	1.13
1	0	0.04	1.04
1	0	0.05	1.05
1	0	0.12	1.12
1	0	0.29	1.29
1	0	0.93	1.93
1	0	1.13	2.13
1	1	0.04	2.04
1	1	0.05	2.05
1	1	0.12	2.12
1	1	0.29	2.29
1	1	0.93	2.93
1	1	1.13	3.13

Table 3.1. x_1 values

For each case of x_1, each case of a_1 value is considered separately. The quantile computation is done through computing $(x_1+a_1+\xi_1)^2$ for $\xi_1 \in \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\}$ and getting the quantile using these 6 values by the help of *numpy.quantile* function as given in Section 1.2.4.

If we denote the set of all possible sampled noise values as $\Xi = \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\}$, then

$J(1, x_1) = x_1^2 + a_1^2 + s^* + \frac{4}{3} \sum_{\xi_1 \in \Xi} (\frac{1}{6})((x_1 + a_1 + \xi_1)^2 - s^*)^+)$

Since this process involves 54 computations, it was automized by using the Python code looping through all x_1, a_1 combinations, giving the s^* and $J(1, x_1, a_1)$ values. In the table, $\pi^*(1, x_1)$ stands for the optimal action for the given $t = 1$ and x_1 value.

x_1	a_1	s^*	$J(1, x_1, a_1)$	$J(1, x_1)$	$\pi^*(1, x_1)$
	-1	0.13	1.64		
0.04	0	0.01	0.55	0.55	0
	1	1.23	3.74		
	-1	0.13	1.64		
0.05	0	0.01	0.56	0.56	0
	1	1.25	3.77		
	-1	0.13	1.54		
0.12	0	0.04	0.66	0.66	0
	1	1.41	4.01		
	-1	0.18	1.42		
0.29	0	0.13	0.99	0.99	0
	1	1.84	4.68		
	-1	0.00	2.29		
0.93	0	1.00	3.26	2.29	-1
	1	3.99	8.24		
	-1	0.04	2.94		
1.13	0	1.43	4.31	2.94	-1
	1	4.83	9.68		
	-1	0.01	2.63		
1.04	0	1.23	3.82	2.63	-1
	1	4.44	9.01		

x_1	a_1	s^*	$J(1, x_1, a_1)$	$J(1, x_1)$	$\pi^*(1, x_1)$
1.05	-1	0.01	2.66	2.66	-1
	0	1.25	3.87		
	1	4.48	9.08		
1.12	-1	0.04	2.90	2.90	-1
	0	1.41	4.25		
	1	4.79	9.61		
1.29	-1	0.13	3.57	3.57	-1
	0	1.84	5.26		
	1	5.56	10.96		
1.93	-1	1.00	7.12	7.12	-1
	0	3.99	10.10		
	1	8.99	17.07		
2.13	-1	1.43	8.57	8.57	-1
	-0	4.83	11.94		
	1	10.22	19.31		
2.04	-1	1.23	7.90	7.90	-1
	-0	4.44	11.09		
	1	9.66	18.28		
2.05	-1	1.25	7.97	7.97	-1
	-0	4.48	11.18		
	1	9.72	18.40		
2.12	-1	1.41	8.49	8.49	-1
	-0	4.79	11.85		
	1	10.16	19.20		
2.29	-1	1.84	9.84	9.84	-1
	-0	5.56	13.54		
	1	11.27	21.23		

x_1	a_1	s^*	$J(1, x_1, a_1)$	$J(1, x_1)$	$\pi^*(1, x_1)$
2.93	-1	3.99	15.96	15.96	-1
	-0	8.99	20.93		
	1	15.98	29.90		
3.13	-1	4.83	18.20	18.20	-1
	-0	10.22	23.57		
	1	17.62	32.95		

Table 3.2. $J(1, x_1, a_1)$ values

Step 3. t = 0

Now, having obtained $J(1, x_1, a_1)$ values, we can move to computing $J(1, x_0, a_0)$ values. For this, a similar approach is used.

$J(0, x_0) = \inf_{a_0} AVaR_{0.25}[x_0^2 + a_0^2 + J(1, x_1)|x_0, a_0]$

Knowing that $x_0 = 1$,

$J(0, x_0) = \inf_{a_0} AVaR_{0.25}[1^2 + a_0^2 + J(1, x_1)|a_0]$

Using **Theorem 1.1**,

$$\begin{aligned} J(0, x_0) &= \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-0.25} \inf_{a_0} E[(1 + a_0^2 + J(1, x_1) - s)^+|a_0]\} \\ &= 1 + a_0^2 + \inf_{s \in \mathbb{R}} \{s + \tfrac{1}{1-0.25} \inf_{a_0} E[(J(1, x_1) - s)^+|a_0]\} \\ &= 1 + a_0^2 + s^* + \tfrac{1}{1-0.25} \inf_{a_0} E[(J(1, x_1) - s)^+|a_0] \\ &= 1 + a_0^2 + s^* + \tfrac{4}{3} \textstyle\sum_{\xi_1 \in \Xi} (\tfrac{1}{6})(J(1, x_1) - s^*)^+) \end{aligned}$$

As we know, $x_1 = x_0 + a_0 + \xi_0$ and $x_0 = 1$, $a_0 \in \{-1, 0, 1\}$, $\xi_t \in \{0.04, 0.05, 0.12, 0.29, 0.93, 1.13\} = \Xi$. So, each a_0 case will be considered separately given that $x_0 = 1$ is fixed.

Table 3.1 is used to obtain x_1 values resulted by certain a_0 value. We apply the **Table 3.2** to retrieve corresponding $J(1, x_1)$ values.

Case a. a = −1

$s^* = VaR_{0.25}(J(1, 1 - 1 + \xi_0)) = VaR_{0.25}(J(\xi_0)) = 0.59$

Therefore,

$$\begin{aligned} J(0, x_0, a_0 = -1) &= 1 + (-1)^2 + s^* + \tfrac{4}{3} \textstyle\sum_{\xi_1 \in \Xi} (\tfrac{1}{6})(J(1, x_1) - s^*)^+) \\ &= 1 + (-1)^2 + s^* + \tfrac{4}{3} \textstyle\sum_{\xi_1 \in \Xi} (\tfrac{1}{6})(J(1, 1 - 1 + \xi_1) - s^*)^+) \end{aligned}$$

$$= 1 + 1 + 0.59 + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1,\xi_1) - 0.59)^+)$$

$$= 2.92 + \tfrac{2}{9}[(J(1,0.04) - 0.59)^+ + (J(1,0.05) - 0.59)^+$$

$$+(J(1,0.12) - 0.59)^+ + (J(1,0.29) - 0.59)^+ + (J(1,0.93) - 0.59)^+$$

$$+(J(1,1.13) - 0.59)^+]$$

$$= 2.59 + \tfrac{2}{9}[0 + 0.03 + 0.07 + 0.4 + 1.7 + 2.35]$$

$$= 3.59$$

Case b. a = 0

$s^* = VaR_{0.25}(J(1, 1 + 0 + \xi_0)) = VaR_{0.25}(J(1 + \xi_0)) = 2.72$

Therefore,

$$J(0, x_0, a_0 = 0) = 1 + 0^2 + s^* + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, x_1) - s^*)^+)$$

$$= 1 + 0^2 + s^* + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, 1 + 0 + \xi_1) - s^*)^+)$$

$$= 1 + 2.72 + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, 1 + \xi_1) - 2.72)^+)$$

$$= 3.72 + \tfrac{2}{9}[(J(1,1.04) - 2.72)^+ + (J(1,1.05) - 2.72)^+$$

$$+(J(1,1.12) - 2.72)^+ + (J(1,1.29) - 2.72)^+ + (J(1,1.93) - 2.72)^+$$

$$+(J(1,2.13) - 2.72)^+]$$

$$= 3.72 + \tfrac{2}{9}[0 + 0 + 0.18 + 0.85 + 4.4 + 5.85]$$

$$= 6.23$$

Case c. a = 1

$s^* = VaR_{0.25}(J(1, 1 + 1 + \xi_0)) = VaR_{0.25}(J(2 + \xi_0)) = 8.1$

Therefore,

$$J(0, x_0, a_0 = 1) = 1 + 1^2 + s^* + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, x_1) - s^*)^+)$$

$$= 1 + 1^2 + s^* + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, 1 + 1 + \xi_1) - s^*)^+)$$

$$= 2 + 8.1 + \tfrac{4}{3}\textstyle\sum_{\xi_1 \in \Xi}(\tfrac{1}{6})(J(1, 2 + \xi_1) - 8.1)^+)$$

$$= 10.1 + \tfrac{2}{9}[(J(1,2.04) - 8.1)^+ + (J(1,2.05) - 8.1)^+$$

$$+(J(1,2.12) - 8.1)^+ + (J(1,2.29) - 8.1)^+ + (J(1,2.93) - 8.1)^+$$

$$+(J(1,3.13) - 8.1)^+]$$

$$= 10.1 + \tfrac{2}{9}[0 + 0 + 0.39 + 1.74 + 7.86 + 10.1]$$

$$= 14.56$$

After considering all these cases for a_0 values, we deduce the final $J(0, x_0)$.

$$
\begin{aligned}
J(0, x_0) &= \inf_{a_0 \in \{-1,0,1\}} J(0, x_0) \\
&= \inf_{a_0 \in \{-1,0,1\}} \{J(0, x_0, a_0 = -1), J(0, x_0, a_0 = 0), J(0, x_0, a_0 = 1)\} \\
&= \inf\{3.59, 6.23, 14.56\} \\
&= 3.59
\end{aligned}
$$

So the final answer for $\alpha = 0.25$ is $\mathbf{J(0, x_0) = 3.59}$.

The obtained results can be collected in the following table, where $\pi^*(0, x_0)$ stands for the optimal action for the given $t = 0$ and x_0 value.

x_0	a_0	s^*	$J(0, x_0, a_0)$	$J(0, x_0)$	$\pi^*(0, x_0)$
1	-1	0.59	3.59	3.59	-1
	0	2.72	6.23		
	1	8.10	14.56		

Table 3.3. $J(0, x_0, a_0)$ values

3.4.4 LQR Problem with Sampled Exponential Noise at risk levels $\alpha = 0.5$, $\alpha = 0.75$ and $\alpha = 0.99$

The calculation with nonzero risk α is done similarly to the $\alpha = 0.25$ case. The results of the work are collected in a table.

α	$J(0, x_0)$
0	3.18
0.25	3.59
0.5	4.37
0.75	5.61
0.99	6.81

Table 3.4. $J(0, x_0)$ values versus risk level α

From Table 3.4, it can be observed that a higher α value would result in a higher $J(0, x_0)$ value.

3.4.5 Plots for LQR Problem with Sampled Exponential Noise at risk level $\alpha = 0$

$V_{\min}(0,1)$ values were computed using code incorporating an approximate dynamic programming algorithm, mentioned in Section 4. To investigate the behavior of the $V_{\min}(0,1)$ depending on terminal time $t = T$, the plots of $V_{\min}(0,1)$ versus the number of iterations $n \in \{1, ..., N = 200\}$ and the terminal time $T \in \{2, 3, 4, 5\}$ were made. The plots show convergence to a true value after a certain number of iterations in the range $n \in \{1, ..., N = 200\}$.

The plots demonstrate that the number of iterations needed for $V_{\min}(0,1)$ to converge increases as the terminal time T increases, which is explained by the number of possible states increasing each time as in Section 3.4.2. As there are 18^t possible states at each time t, the problem becomes much more computationally complex as time increases.

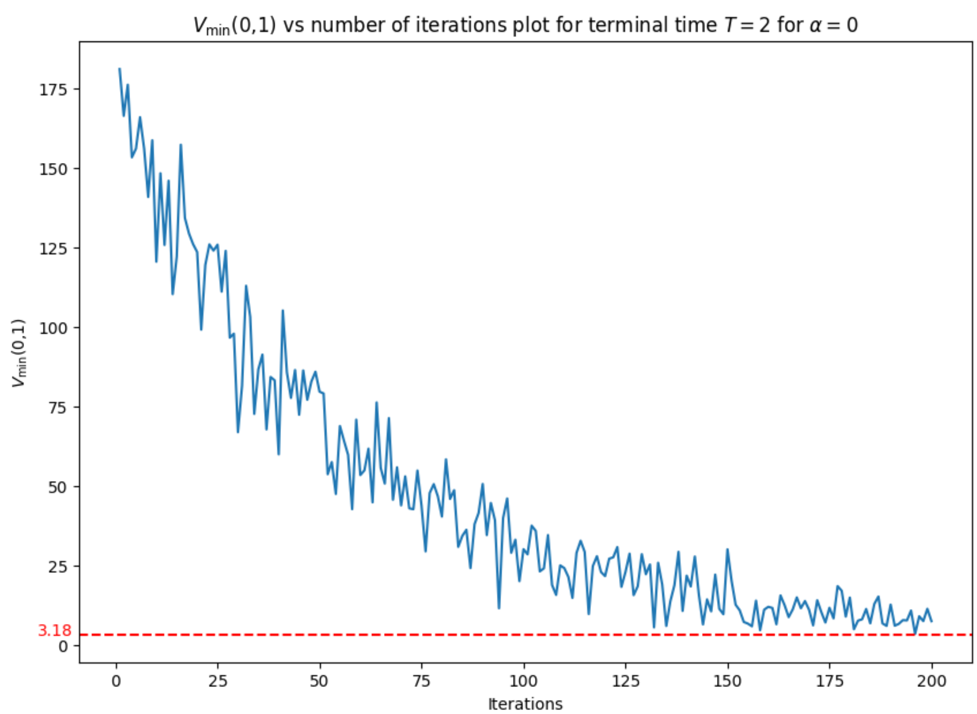

Figure 3.2. $V_{\min}$(0,1) vs number of iterations plot for terminal time $T = 2$ for $\alpha = 0$

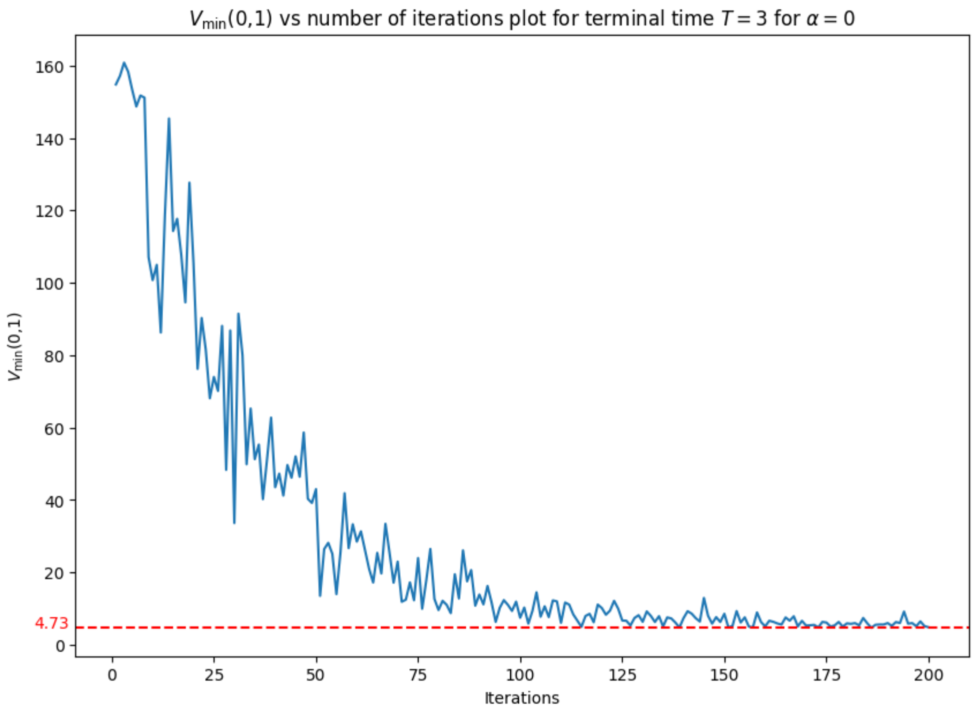

Figure 3.2. $V_{\min}$*(0,1) vs number of iterations plot for terminal time* $T = 3$ *for* $\alpha = 0$

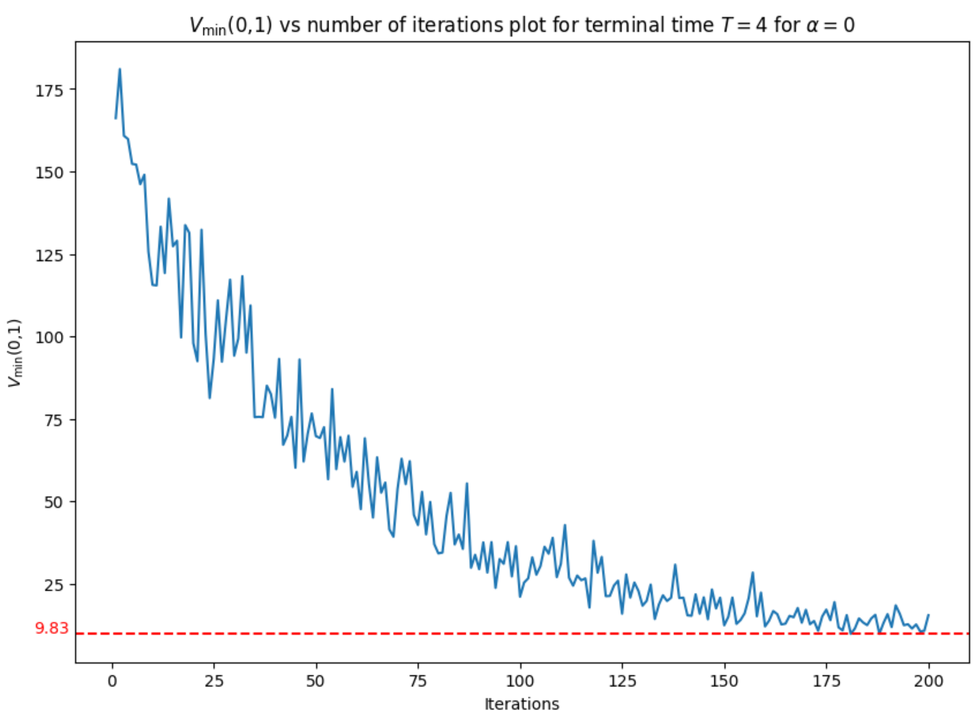

Figure 3.2. $V_{\min}$*(0,1) vs number of iterations plot for terminal time* $T = 4$ *for* $\alpha = 0$

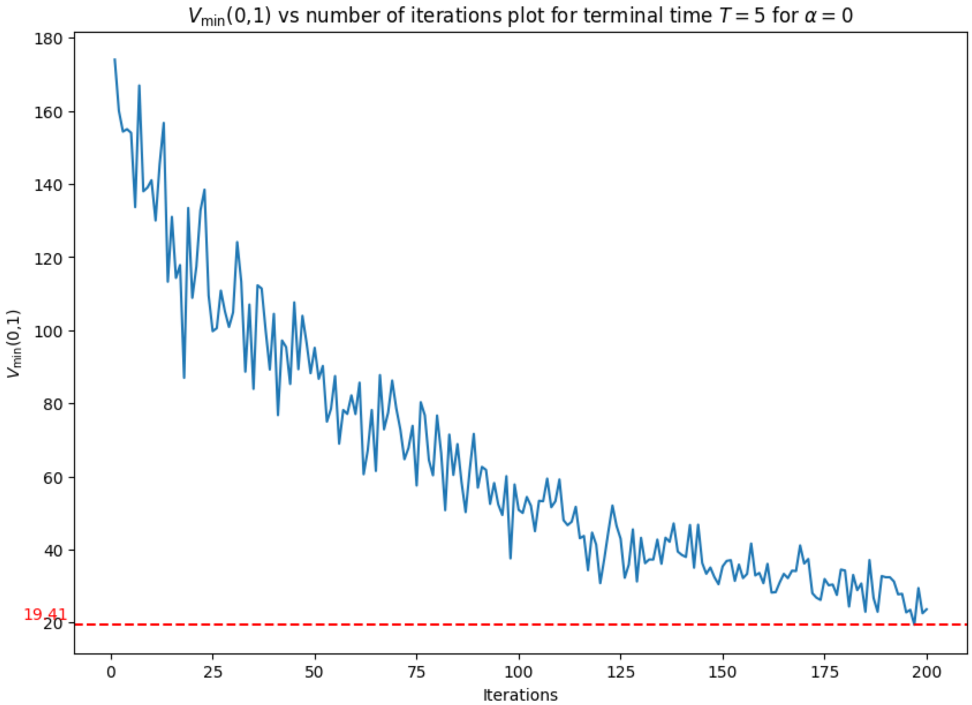

Figure 3.2. $V_{\min}$*(0,1) vs number of iterations plot for terminal time* $T = 5$ *for* $\alpha = 0$

Chapter 4

Approximate dynamic programming algorithm

4.1 The algorithm

The approximate dynamic programming approach described in [12] was simplified and modified for our problem. The computation consists of the following steps:

- Select and fix the number of iterations N.
- Set the iteration counter n = 1, set the initial parameters for state(x_0), initial time t_0, terminal time T.
- Set the action space A (so that $a_t \in A$) and take a random noise samples ξ_t (so that $\xi_t \sim SelectedDistribution$)
- Initialize an initial approximation $\overline{V}_t^0$, $\forall t \in \{1, ..., T\}$
- **Forward pass**: For each $t \in \{1, ..., T\}$ create a random path by randomly choosing (a_t, ξ_t).
- **Backward pass**: For each $t \in \{1, ..., T\}$ compute following using the the selected learning rate α and the decision $\hat{a}_t^n$ obtained from forward pass:

$$\hat{v}_t^n = c(x_t^n, \hat{a}_t^n) + \hat{v}_t^n, \quad \text{with } \hat{v}_{T+1}^n = 0$$

$$\overline{V}_{t-1}^n(x_{t-1}^{a,n}) = U^V\left(\overline{V}_{t-1}^{n-1}(x_{t-1}^{a,n}), x_{t-1}^{a,n}, \hat{v}_t^n\right) = (1-\alpha)\overline{V}_{t-1}^{n-1} + \alpha\hat{v}_t^n$$

- Increment n until the iteration number $n > N$.
- Return the value functions $\overline{V}_t^N(x_t^{a,n}) \quad \forall t \in \{1, ..., T\}$ and $x_t \in X$.

4.2 Implementation

The algorithm is implemented in Python 3, using the NumPy, random and Matplotlib libraries. The code is given in the Appendix.

4.3 Evaluation

To evaluate the performance of the code on the LQR problem of this thesis, the check was done with all the given risk levels $\alpha \in \{0, 0.25, 0.5, 0.75, 0.99\}$.

α	$J_{theor}(0, x_0)$	$J_{code}(0, x_0)$	error
0	3.18	3.18	0
0.25	3.59	3.59	0
0.5	4.37	4.37	0
0.75	5.61	5.61	0
0.99	6.81	6.81	0

Table 3.5. $J_{theor}(0, x_0)$ values comparison to $J_{code}(0, x_0)$ values versus risk level α

This, with the addition of the convergence of the plots to the true values obtained analytically, shows the perfect accuracy of this algorithm to the given LQR problem. However, further tests may be needed for the higher terminal time T values.

Chapter 5

Summary of main results

LQR Problem with Theoretical Exponential Noise at a risk level $\alpha = 0$ gave a pattern regarding the optimal policy. It considered two cases:

- **Case 1.** $\boldsymbol{\lambda < 1}$ **OR**

  $$\boldsymbol{\lambda \geq 1 \text{ and } \mathbf{x} \geq 1 - \mathbf{E}[\xi_1 | \mathbf{x_1}, \mathbf{a_1}]}.$$

- **Case 2.** $\boldsymbol{\lambda \geq 1 \text{ and } \mathbf{x} \in [0, 1 - \mathbf{E}[\xi_1 | \mathbf{x_1}, \mathbf{a_1}])}$

So, for Case 1, the optimal policy is $a_0 = -1, a_1 = 0, a_2 = 0$.

For Case 2, the optimal policy is $a_0 = -1, a_1 = -1, a_2 = 0$.

The LQR Problem with Sampled Exponential Noise at a risk level $\alpha = 0.25$ justifies this observation. With the $\lambda = 1$ and $E[\xi_1 | x_1, a_1] = 0.43$, we know that $1 - E[\xi_1 | x_1, a_1] = 1 - 0.43 = 0.57$. As seen in Table 3.2, for all $x_1 < 0.57$, the optimal action $a_1 = 0$, while the optimal action $a_1 = -1$ for $x_1 \geq 0.57$.

Both for theoretical and exponential noise, $a_0 = -1$ is satisfied for any case. Intuitively, $c_t(x_t, a_t) = x_t^2 + a_t^2$, so $a_0 = 0$ may be assumed to be a minimizing action. However, since $c_1(x_1, a_1) = x_1^2 + a_1^2$ and $x_1 = x_0 + a_0 + \xi_0$ for $x_0 = 1$ and $\xi_0 > 0$, x_1 is minimized by $a_0 = -1$ due to the transition function. So, compared with Bernoulli noise which can take a value of -1, in the case of the exponential noise, the action value has to be minimized even further as exponential noise punishes action a_t more, giving a higher weight to it. So, this point is verified both theoretically and in experiments.

The LQR Problem with Sampled Exponential Noise at a risk level $\alpha \neq 0$ also proves the **Lemma 1.1.**, where

$$\lim_{\alpha \to 1} AVaR_{\alpha}(X) = \text{ess sup} X \leq \infty$$

This is proved by the pattern that the value function is directly proportional to the risk level α. So, for higher risks, higher-value functions are expected as stated in the Lemma.

References

1. Todorov, E. Optimal control theory (2006). *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT Press: Cambridge, MA, USA, 269–298.

2. Walton, N. Stochastic control (2020)

3. Ugurlu, K. (2017). Controlled Markov Processes with AVaR Criteria for Unbounded Costs. *Journal of Computational and Applied Mathematics*, 319, 24-37.

4. Ahmadi-Javid, A. (2012). Entropic value-at-risk: A new coherent risk measure. *Journal of Optimization Theory and Applications*. 155 (3): 1105–1123.

5. Bauerle, N. Ott, J. (2011). Markov Decision Processes with Average-Value-at-Risk criteria. Mathematical Methods of Operations Research, 74, 361-379.

6. Han, J. Weinan E. (2016). Deep Learning Approximation for Stochastic Control Problems. ArXiv. https://doi.org/10.48550/arXiv.1611.07422

7. Yu, X. Shen S. (2023). Risk-Averse Reinforcement Learning via Dynamic Time-Consistent Risk Measures. ArXiv. https://doi.org/10.48550/arXiv.2301.05981

8. Crespo L. Sun J-Q. (2003). Stochastic optimal control via Bellman's principle. Automatica, 39, 2109-2114.

9. Wrobel, A. (1984). On Markovian Decision Models with a Finite Skeleton". *Mathematical methods of Operations Research*. 28: 17-27.

10. Benac, L. Godin, F. (2021). Risk averse non-stationary multi-armed bandits. ArXiv.

11. Mes, M. R. K., Perez Rivera, A. E. (2017). Approximate Dynamic Programming by Practical Examples. Markov Decision Processes in Practice (pp. 63-101). *International Series in Operations Research Management Science; No. 248).* Springer.

12. Nazemi, A., Talebi, F. Abdolbaghi Ataabadi A. (2022). Mean-AVaR in credibilistic portfolio management via an artificial neural network scheme. *Journal of Experimental Theoretical Artificial Intelligence.*

13. Ni, Y-H., Zhang, J-F., Li, X. (2016). Indefinite Mean-Field Stochastic Linear-Quadratic Optimal Control: From Finite Horizon to Infinite Horizon. *IEEE Transactions on Automatic Control*, 61(11), 3269-3284.

14. Hu, Y., Jin, H. Zhou, X. (2011). Time-Inconsistent Stochastic Linear-Quadratic Control. SIAM Journal on Control and Optimization, 50(3), 1548-1572.

Chapter 6

Appendix

Appendix A. Code for graphical analysis for formulating minimizing piecewise value function

```
# -*- coding: utf-8 -*-
"""appendix_a.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1gwUcGbxVand5G3g4kAcZU1OIeXpBsD3B
"""

import numpy as np
import matplotlib.pyplot as plt

# defining the range for x
x_vals = np.linspace(-1, 3, 1000)

# defining the functions for each interval
y1 = 1 - x_vals
y2 = 0.25 - 0.25*x_vals
y3 = 0*x_vals
y4 = 0.25 + 0.25*x_vals
```

```
y5 = 1 + x_vals

# defining the piecewise function giving the optimal value function
def piecewise(x):
  if x < -0.5:
    return 0.25 + 0.5*x
  elif -0.5 <= x and x < 0.5:
    return 0
  elif 0.5 <= x and x < 1.5:
    return 0.25 - 0.5*x
  else:
    return 1 - x

#vectorizing the piecewise function so that it can take numpy array as
#an input and computing the piecewise function values immediately

piecewise_vectorized = np.vectorize(piecewise)
y_piecewise = piecewise_vectorized(x_vals)

# making the plots

plt.figure(figsize = (10, 7))
plt.xlim(-1, 3)
plt.ylim(-2, 4)
plt.xlabel('$x_1$')
plt.ylabel('$\phi(a_{1})$')
plt.title('Graphs of the $\phi(a_{1})$ for $a_1$')

plt.plot(x_vals, y1, label = '$\phi(a_{1}) = 1 - x_1$')
plt.plot(x_vals, y2, label = '$\phi(a_{1}) = 0.25 - 0.5x_1$')
plt.plot(x_vals, y3, label = '$\phi(a_{1}) = 0$')
plt.plot(x_vals, y4, label = '$\phi(a_{1}) = 0.25 + 0.5x_1$')
plt.plot(x_vals, y5, label = '$\phi(a_{1}) = 1 + x_1$')
```

```python
plt.plot(x_vals, y_piecewise, label = 'piecewise min', color = 'k', linestyle = '--')

plt.text(2.5, -1.5, '$\phi(a_{1}) = 1 - x_1$', fontsize = 9, verticalalignment = 'bottom')
plt.text(2, -0.75, '$\phi(a_{1}) = 0.25 - 0.5x_1$', fontsize = 9, verticalalignment = 'bottom')
plt.text(2.5, 0.1, '$\phi(a_{1}) = 0$', fontsize = 9, verticalalignment = 'bottom')
plt.text(2.4, 1.75, '$\phi(a_{1}) = 0.25 + 0.5x_1$', fontsize = 9, verticalalignment = 'bottom')
plt.text(2, 3.3, '$\phi(a_{1}) = 1 + x_1$', fontsize = 9, verticalalignment = 'bottom')

plt.axvline(0, color='black', linewidth = 0.5)
plt.axhline(0, color='black', linewidth = 0.5)
plt.grid(True)
plt.legend()

plt.show()
```

Appendix B. Code for analytical computation of LQR problem at risk level $\alpha \neq 0$.

```python
# -*- coding: utf-8 -*-
"""appendix_b.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1DHP-n8NW4sovcnOQlamROpBDlIu_bL6R
"""

import numpy as np

alpha = 0.25 #setting risk level
```

```
action_arr = np.array([-1, 0, 1]) # initial set of actions

np.random.seed(41) # setting the seed for reproducibility
random_arr = np.random.exponential(scale = 1.0, size = 6) # sample 6 points from exponential distribution with lambda = 1
random_arr = np.array(sorted(np.round(random_arr, 2)))

x1_vals = [] # array to store x1 values

for a in action_arr:
  for w in random_arr:
    x1_vals.append(round(1 + a + w, 2)) # #iterating through all possible actions and noises

# computing q-quantile given x1, a and random noise values for 0 <= q < = 1
def quantile_finder(x1, a, random_arr, q):
  lst = []
  for ran in random_arr:
    lst.append((x1 + a + ran)**2)
  qt = np.quantile(lst, q) # Numpy function for quantile
  return qt

# solving for optimal j(1, x1) for each x1
def j1_finder (x1_vals, a_vals, random_arr):
  j_vals = [] # array to store j1 values
  for x1 in x1_vals:
    q_vals = [] # array to store qt values

    for a in action_arr:
      # get a quantile for each x1 and a pair
      qt = np.round(quantile_finder(x1, a, random_arr, q = alpha), 2)

```

```
      sum = 0
      # replacing the expected value by average sum through all random noises
      for ran in random_arr:
          sum += (1/6)*max((x1 + a + ran)**2 - qt, 0)

      # computing the value function
      q = np.round(x1**2 + a**2 + qt + sum*2, 2)
      print('x1: {}, a: {}, qt: {}, q: {}'.format(x1, a, qt, q))
      q_vals.append(q)

    # getting the minimum value J(1, x1) for each x1
    j1 = np.min(q_vals)
    j_vals.append(j1)

  return j_vals

# function call to get an array of j1 values for all possible x1 values
j1 = j1_finder(x1_vals, action_arr, random_arr)


def j0_finder (j1):
  j0_vals = [] # array to store j0 values
  a = -1
  for i in range(3):
    # compute quantile for each action
    qt = np.round(np.quantile(j1[6*i:6*i+6], q = alpha), 2)

    sum = 0
    # replacing the expected value by average sum through all random noises
    for j in range(6*i, 6*i + 6):
      sum += (1/6)*max(j1[j] - qt, 0)

```

```
    # computing the value function
    q = np.round(1 + a**2 + qt + sum*2, 2)
    print('a: {}, qt: {}, q: {}'.format(a, qt, q))

    # getting the J(0, x0, a0) for each a0
    j0_vals.append(q)
    a += 1

  # getting the minimum value J(0, x0) for each x0
  j0 = np.min(j0_vals)
  return j0

# function call to get an array of j0 values for all possible x0 values
j0 = j0_finder(j1)
```

Appendix C. Code for approximate dynamic programming algorithm and running simulations

```
# -*- coding: utf-8 -*-
"""appendix_c.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1aBO5QcPcOZdVMcIgXEW9MVaDVzRYjuUw
"""

import numpy as np
import time
import random
import matplotlib.pyplot as plt

# returns costs array for the given curent states and action
# real valued
def cost(state, action):
```

```
  cost = (state ** 2) + (action ** 2)
  cost = np.round(cost, 2)

  return cost

# real valued scalar
def random_next_element(state, action, ran):
  next_state = round(state + action + ran, 2)

  return next_state

def V_hash(V_table, time, state, value):
  key = (time, state)
  V_table[key] = value

  return V_table

def V_lookup(V_table, time, state):
  key = (time, state)
  value = V_table[key]

  return value

def init_cost(state, time):
  max_action = 2
  max_random = 2
  terminal = 5
  val = state**2 + max_action**2
  for i in range(0, terminal - time + 1):
    next_state = state + max_random + max_action
    tmp = next_state**2 + max_action**2
    val+= tmp

  return val

```

```
# dynamic avar calculation
# at each time t; calculates avar using the next time's avar value as the r.v.
def avar(V_table, alpha, state, s, action, time, terminal, ran_arr):
  # if terminal calculate the terminal value AND hash the value for that c(state,...)+ aggr - s for that (state, aggr)

  if time == terminal:
    avar_val = np.round(cost(state, action),2)

    return (avar_val, V_table)
  #otherwise calculate next stage and next aval avar
  else:
    ran_len = len(ran_arr)
    next_time = time + 1
    s_arr = np.array([])
    tmp = 0
    avar_val = 0
    for ran in ran_arr:
    #find the avar for the next space for the given fixed action
      next_state = random_next_element(state, action, ran)  # F(x_t, a_t, \xi_t) # real valued
      next_time = time + 1
      next_key = (next_time, next_state)
      val_2 = 0
      if next_key in V_table: # in V_table_val
        print('KEY {} found next_time: {}, next_state: {} with used action: {}'.format(next_key, next_time, next_state, action))
        val_2 = V_lookup(V_table, next_time, next_state) #V lookup val without act
        print('val_2 by looking up: ', val_2)
      else:
        print('KEY {} NOT found next_time: {}, next_state: {} with used action: {}'.format(next_key, next_time, next_state, action)
```

```
)
        val_2 = init_cost(state, time)
        print('init cost val2: ', val_2)
        #hash that void value to the corresponding key
        V_table = V_hash( V_table, next_time, next_state, val_2) # V hash val without act
      s_arr = np.append(s_arr, val_2)
    s_q = np.quantile(s_arr, alpha, interpolation='linear')  # real valued quantile for s_q
    s_q = np.round(s_q, 2)
    for elt in s_arr:
      avar_val += (1 / ran_len) * max(elt - s_q, 0) # expected value part
    avar_val = s_q + (1 / (1 - alpha)) * avar_val #the remaining arithmetic operations
    avar_val = cost(state, action) + avar_val # accumulating the cost
    avar_val = np.round(avar_val,2)

    print('time, state, action: ', time, state,  action)

    return (avar_val, V_table)

# dynamic avar calculation
# at each time t; calculates avar using the next time's avar value as the r.v.
def avar(V_table, alpha, state, s, action, time, terminal, ran_arr):
  # if terminal calculate the terminal value AND hash the value for that c(state,...)+ aggr - s for that (state, aggr)

  if time == terminal:
    avar_val = np.round(cost(state, action),2)

    return (avar_val, V_table)
```

```python
    #otherwise calculate next stage and next aval avar
    else:
        ran_len = len(ran_arr)
        next_time = time + 1
        s_arr = np.array([])
        tmp = 0
        avar_val = 0
        for ran in ran_arr:
        #find the avar for the next space for the given fixed action
            next_state = random_next_element(state, action, ran)  # F(x_t, a_t, \xi_t) # real valued
            next_time = time + 1
            next_key = (next_time, next_state)
            val_2 = 0
            if next_key in V_table: # in V_table_val
                print('KEY {} found next_time: {}, next_state: {} with used action: {}'.format(next_key, next_time, next_state, action))
                val_2 = V_lookup(V_table, next_time, next_state) #V lookup val without act
                print('val_2 by looking up: ', val_2)
            else:
                print('KEY {} NOT found next_time: {}, next_state: {} with used action: {}'.format(next_key, next_time, next_state, action))
                val_2 = init_cost(state, time)
                print('init cost val2: ', val_2)
                #hash that void value to the corresponding key
                V_table = V_hash( V_table, next_time, next_state, val_2) # V hash val without act
            s_arr = np.append(s_arr, val_2)
        s_q = np.quantile(s_arr, alpha, interpolation='linear')  # real valued quantile for s_q
        s_q = np.round(s_q, 2)
        for elt in s_arr:
            avar_val += (1 / ran_len) * max(elt - s_q, 0) # expected
```

```
value part
    avar_val = s_q + (1 / (1 - alpha)) * avar_val #the remaining arithmetic operations
    avar_val = cost(state, action) + avar_val # accumulating the cost
    avar_val = np.round(avar_val,2)

    print('time, state, action: ', time, state,  action)

    return (avar_val, V_table)

#running the simulations
def simulate(x_0, t_0, T, alpha, action_arr, random_arr, N, s, lr):

  print('s in simulate: ', s)

  V_table = Running_Bellman(alpha, T, action_arr, random_arr, N, s, x_0, lr)

  return  V_table


# Simulation with Bernoulli noise

x_0 = 1 # initial state
t_0 = 0 # initial time
T = 2 # terminal time
N = 200 # maximum number of iterations
alpha = 0.00 # risk level, can be 0, 0.25, 0.5, 0.75, 0.99

s_array = np.array([0]) # s to be minimized in AVaR formula
action_arr = np.array([-1, -0.5, 0, 0.5, 1]) # initial set of actions
random_arr = np.array([-1, 1]) # array of random Bernoulli noises

```

```
key = (t_0, x_0) # initial key
min_value = float('inf')
V_min = {}

for r_a in s_array:
  V_table = simulate(x_0, t_0, T, alpha, action_arr, random_arr, N, r_a, lr = 0.95)
  tmp = V_table[key]
  if tmp < min_value:
    #if the initial value is minimum then assign the value function as the minimum
    V_min = V_table
    min_value = tmp
    s_min = r_a

print('(V_min[(time,state, aggr)]: min_value)', V_min)
print('s_min: ', s_min)
print('V_min[({}, {})]: {}'.format(t_0, x_0, min_value))


# Simulation with Exponential noise

x_0 = 1 # initial state
t_0 = 0 # initial time
T = 2 # terminal time
N = 200 # maximum number of iterations
alpha = 0.00 # risk level, can be 0, 0.25, 0.5, 0.75, 0.99

s_array = np.array([0])
action_arr = np.array([-1, 0, 1]) # initial set of actions

np.random.seed(41) # setting the seed for reproducibility
random_arr = np.random.exponential(scale = 1.0, size = 6) # sample 6 points from exponential distribution with lambda = 1
random_arr = np.array(sorted(np.round(random_arr, 2)))
```

```python

key = (t_0, x_0) # initial key
min_value = float('inf')
V_min = {}

for r_a in s_array:
  V_table = simulate(x_0, t_0, T, alpha, action_arr, random_arr, N, r_a, lr = 0.99)
  tmp = V_table[key]
  if tmp < min_value:
    #if the initial value is minimum then assign the value function as the minimum
    V_min = V_table
    min_value = tmp
    s_min = r_a

print('(V_min[(time,state, aggr)]: min_value)', V_min)
print('s_min: ', s_min)
print('V_min[({}, {})]: {}'.format(t_0, x_0, min_value))
```

Appendix D. Code for $V_{\min}$(0,1) vs number of iterations plot for terminal time $T \in \{2, 3, 4, 5\}$

```python
# -*- coding: utf-8 -*-
"""appendix_d.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1r019nE00oxXE-Bk0zGZFEhtPBqCw1zFR
"""

def value_array_creater(terminal_time):
  min_vals = [] # array to store J(0, x0) value for each number of iterations n

```

```
for N in range(1, 201):
  x_0 = 1 # initial state
  t_0 = 0 # initial time
  T = terminal_time # terminal time, can be 2, 3, 4, 5
  alpha = 0.00 # risk level, can be 0, 0.25, 0.5, 0.75, 0.99

  s_array = np.array([0]) # s to be minimized in AVaR formula
  action_arr = np.array([-1, 0, 1]) # initial set of actions

  np.random.seed(41) # setting the seed for reproducibility
  random_arr = np.random.exponential(scale = 1.0, size = 6) # sample 6 points from exponential distribution with lambda = 1
  random_arr = np.array(sorted(np.round(random_arr, 2)))

  key = (t_0, x_0) # initial key
  min_value = float('inf')
  V_min = {}

  for r_a in s_array:
    V_table = simulate(x_0, t_0, T, alpha, action_arr, random_arr, N, r_a, lr = 0.99)
    tmp = V_table[key]

    if tmp < min_value:
      #if the initial value is minimum then assign the value function as the minimum
      V_min = V_table
      min_value = tmp
      s_min = r_a

  min_vals.append(min_value)
  print('(V_min[(time,state, aggr)]: min_value)', V_min)
  print('s_min: ', s_min)
  print('V_min[({}, {})]: {}'.format(t_0, x_0, min_value))

```

```
    return min_vals


# Obtaining J(0, x0) values for terminal time = 2
t = 2
min_vals2 = value_array_creater(terminal_time = t)

# Getting the minimum J(0, x0) value for terminal time = 2
np.min(min_vals2)

# Making a plot
n = np.arange(1, 201, 1) # number of iteration
plt.figure(figsize=(10,7))
plt.plot(n, min_vals2) # J(0, x0) value versus number of iteration plot
plt.title(r'$V_\min$(0,1) vs number of iterations plot for terminal time $T = {}$ for $\alpha = 0$'.format(t)) # plot title
plt.axhline(y=3.18, color='r', linestyle='--')  # Add horizontal line at y = 3.98 (the minimum J(0, x0) value for terminal time = 2)
plt.text(-10.3, 3.2, '3.18', color='red', ha='right')  # Annotate the y-axis at the level 3.98 (the minimum J(0, x0) value for terminal time = 2)
plt.xlabel('Iterations')
plt.ylabel('$V_\min$(0,1)')
plt.show()
```