Machine Learning Approximation of Risk-Averse Policies in Markov Decision Processes with Average-Value-at-Risk

by

Yerlan Sardarbekov

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Master of Science in Applied Mathematics

at the

NAZARBAYEV UNIVERSITY

April 2025

Author .....
Department of Mathematics

Certified by .....
Kerem Uğurlu
Assistant Professor
Thesis Supervisor

Accepted by .....
Gonzalo Hap Hortelano
Dean, School of Sciences and Humanities

Abstract

This thesis investigates the integration of Average-Value-at-Risk (AVaR) into Approximate Dynamic Programming (ADP) frameworks for solving Markov Decision Processes (MDPs) under uncertainty. We develop analytical and numerical methods to derive risk-averse policies for a quadratic cost minimization problem with stochastic dynamics. The problem is formulated as a nested optimization where the inner problem determines optimal actions for given states and risk thresholds, while the outer problem identifies optimal thresholds. We compare analytical solutions with Monte Carlo simulations across various risk aversion levels. To address computational challenges in real-time applications, we develop polynomial regression models that accurately approximate the optimal policies. The resulting models preserve critical structural features of the original solutions while enabling efficient implementation. Our findings contribute to the growing field of risk-aware decision-making by providing both theoretical insights and practical methods for implementing AVaR-based optimization in complex stochastic environments. The proposed machine learning approach offers a computationally efficient alternative to traditional numerical methods without sacrificing accuracy, making risk-sensitive optimization more accessible for real-world applications in finance, robotics, healthcare, and energy management.

Contents

Chapter 1

Introduction

1.1 Motivation and Background

Markov decision processes have been widely studied and applied in various industries like finance, robotics, control systems, and healthcare. Approximate Dynamic Programming (ADP) is an effective method for solving discrete-time Markov decision processes (MDP) where state and action spaces can be infinite. ADP computes the approximated optimal value using various strategies to solve the problem of dimensionality and computational difficulties. Although ADP has been successfully applied to various MDP problems, its integration into real-world scenarios remains challenging due to the inherent stochasticity present in practical applications. This complexity necessitates the development of robust and coherent risk measures, making this area a critical focus of ongoing research.

Average-Value-at-Risk (AVaR) is one such measure that tackles the issues related to randomness and unexpected outcomes by focusing not just on the likelihood of bad scenarios, but also on how severe they might be. Unlike traditional risk measures like Value-at-Risk, AVaR takes into account the full range of potential losses beyond the threshold, making it a more reliable tool for managing risk. The application of AVaR into ADP represents an enhancing approach to decision-making processes under uncertainty. This is valuable in real-world problems like financial portfolio management, energy distribution, and supply chain optimization - where maximizing

long-term rewards while mitigating potential losses is critical.

Machine learning (ML) techniques have emerged as powerful tools in addressing the computational challenges of ADP, particularly when incorporating risk measures such as AVaR. These techniques offer significant advantages in handling the curse of dimensionality inherent in complex MDPs. By using neural networks and other approximation architectures, machine learning approaches can efficiently represent high-dimensional value functions and policies while capturing the severe risk considerations that AVaR provides. The integration of machine learning and AVaR is particularly promising, as it combines the computational efficiency and scalability of ML methods with the sophisticated risk assessment capabilities of AVaR, creating a framework that is both mathematically rigorous and practical in real-world decision systems.

Over the past 15 years, research has explored the application of machine learning methods to optimal control problems with AVaR criteria, demonstrating promising results in managing risk and uncertainty. While traditional analytical methods provide theoretical guarantees, ML-based approaches offer greater scalability and adaptability in complex environments. The integration of machine learning with ADP and AVaR risk measures presents an opportunity for developing more sophisticated, risk-aware decision-making frameworks that can operate effectively in uncertain environments.

This thesis aims to explore and evaluate machine learning techniques for obtaining optimal policies in MDP problems incorporating AVaR measures. By comparing these techniques with traditional numerical methods, the study will assess their effectiveness in balancing risk and reward under uncertainty. The goal is to determine whether machine learning can offer more scalable and adaptable solutions for risk-aware decision-making, ultimately contributing to the broader application of ADP in real-world scenarios.

1.2 Literature Review

With the increasing interest in risk-sensitive stochastic optimal control problems, the number of papers addressing the integration of machine learning techniques into MDPs with risk measures is growing steadily. Various studies have presented different techniques and algorithms that merge ADP with risk measures such as AVaR, though significant gaps remain in this evolving field.

Bäuerle and Glauner (2021) discuss the integration of recursive risk measures into Markov decision processes, highlighting the importance of considering risk in sequential decision-making. Their work lays the groundwork for understanding how risk measures can be systematically applied within MDPs, a core element of ADP. Moreover, Bäuerle and Ott (2011), and Bäuerle and Rieder (2013) explored AVaR criteria in MDPs, offering early insights into the potential of AVaR as a more informative risk measure for decision-making under uncertainty.

Extending the application of risk measures, Carpin et al. (2016) delve into the use of total cost criteria and AVaR in finite MDPs, offering a practical perspective on risk aversion in robotic automation scenarios. This application to robotics automation provides a tangible example of ADP's versatility when augmented with sophisticated risk measures.

Liu et al. (2021) tackle the challenge of unbounded costs in MDPs using an AVaR criterion, addressing a significant obstacle in risk-sensitive decision-making models. Their methodology for managing unbounded costs is particularly relevant for financial applications where extreme losses can have substantial impacts.

The educational aspect of ADP is covered by Mes and Rivera (2017), who provide practical examples of ADP in action. Their work, while not directly focused on AVaR, enriches the understanding of ADP's applications and potential for adaptation to include advanced risk measures.

Lastly, Uğurlu (2017, 2018) and Yoshida and Kumamoto (2019) contribute to the body of knowledge by offering models and frameworks that incorporate AVaR criteria in controlled MDPs and asset management, respectively. These studies underscore the

versatility and critical importance of AVaR in enhancing decision-making frameworks across various domains, from operational control to financial portfolio management.

The collective insights from these works illustrate the evolving landscape of risk-sensitive decision-making in operations research. By applying AVaR within the ADP framework, this research contributes to the field by demonstrating the practical integration of established risk measures into dynamic decision models, thereby supporting more informed and robust decision-making under uncertainty.

Chapter 2

Problem Statement

In this master's thesis, we delve into the optimization of controlled Markov Decision Processes (MDPs) under risk criteria, specifically employing Average-Value-at-Risk (AVaR) to address the challenges presented by uncertain environments. The objective is to develop both analytical and numerical methods for deriving risk-averse policies that are computationally tractable and applicable in real-world scenarios.

Components of the MDP:

We consider a discrete-time MDP characterized by the following components:

- State space S: The set of all possible states S_t of the system.
- Action space A: The set of all possible actions or decisions a_t available at each state.
- Cost function $C(S_t, a_t)$: The immediate cost associated with taking action a_t in state S_t.
- Transition probability $\mathbb{P}(S_{t+1}|S_t, a_t)$: The probability of transitioning to state S_{t+1} from state S_t under action a_t.
- Policy π: A decision function that maps states to actions.
- Time $t \in \{0, 1, 2, \ldots, T\}$: The discrete time steps of decision-making, where T can approach infinity in an infinite horizon problem.

The system starts in a particular S_t from which we make a decision a_t which will bring the actor to the next state S_{t+1} with a certain probability $\mathbb{P}(S_{t+1}, |S_t, a_t)$. The goal is to find a policy $\pi \in \Pi$ that can be seen as a decision function $a_t = \pi(S_t)$ where Π is a set of potential decision policies.

The core of our study is the Bellman equation for an infinite horizon problem, which traditionally seeks to find the policy that minimizes the total expected cost. The Bellman equation is represented as:

$$V^{\pi}(S_t) = \min_{a_t \in A} \left(C(S_t, a_t) + \gamma \sum_{S_{t+1} \in S} \mathbb{P}(S_{t+1}|S_t, a_t) V^{\pi}(S_{t+1}) \right) \tag{2.1}$$

where $V^{\pi}(S_t)$ is the value function under policy π for state S_t, and γ is the discount factor that balances immediate versus future costs.

2.1 Definition of Risk Measures

Standard approaches to MDPs focus on minimizing the expected total cost. However, such formulations may not adequately capture the risk of rare but severe losses. These extreme outcomes, often referred to as tail risk, lie in the lower-probability regions of the cost distribution but can have large impacts. To overcome this limitation, we incorporate the AVaR measure into our objective function, allowing for explicit consideration of such high-impact, low-probability events.

Given a real-valued random variable X representing total costs and a risk level $\tau \in (0, 1)$, we define (Uğurlu, 2017):

1. $VaR_{\tau}(X) = \inf\{x \in \mathbb{R} : \mathbb{P}(X \leq x) \geq \tau\}$. This represents the minimum threshold that the cost will not exceed with probability τ.

2. Average-Value-at-Risk: $AVaR_{\tau}(X) = \frac{1}{1-\tau} \int_{\tau}^{1} VaR_t(X)dt$, which simplifies to $E[X|X \geq VaR_{\tau}(X)]$ for continuous distributions. This measures the expected cost in the worst $(1 - \tau)$ fraction of scenarios.

For an infinite horizon problem, our goal is to minimize the AVaR-adjusted expected total cost:

$$\min_{\pi \in \Pi} AVaR_{\alpha}^{\pi} \left[\sum_{n=0}^{\infty} \gamma^n C(S_n, a_n) \right] \tag{2.2}$$

This formulation emphasizes not only the expected cost but also the tail risk associated with the cost distribution. Unlike the standard Bellman equation, which only considers the expected value, this approach accounts for the severity of worst-case scenarios, making it particularly valuable for risk-sensitive applications.

To effectively solve the risk-averse optimization problem, we need a computationally tractable approach. The direct computation of AVaR in a dynamic programming context presents significant challenges due to the non-linear and non-separable nature of the AVaR operator. We address this by employing a nested optimization approach based on a key insight from Uğurlu (2017) and equation 2.2 can be reformulated as:

$$\inf_{\pi \in \Pi} AVaR_{\alpha}^{\pi} \left[\sum_{n=0}^{\infty} \gamma^n c(S_n, a_n) \right] = \inf_{s \in \mathbb{R}} \left\{ s + \frac{1}{1-\alpha} \inf_{\pi \in \Pi} \mathbb{E}^{\pi} \left[(\sum_{n=0}^{\infty} \gamma^n c(S_n, a_n) - s)^+ \right] \right\}$$

In this expression the parameter s acts as a threshold that segregates the acceptable cost from the tail risks. The notation $(x)^+$ denotes the positive part of x, i.e., $\max(x, 0)$. The inner optimization problem focuses on finding the optimal policy given a threshold. The outer optimization problem determines the optimal threshold s.

This reformulation offers several advantages. It transforms the complex AVaR minimization into a more manageable two-stage optimization problem. The inner optimization remains within the realm of expected value minimization, allowing us to use modified versions of standard dynamic programming techniques. Meanwhile, the outer optimization is reduced to a single-variable problem that can be efficiently solved using standard numerical methods.

2.2 Research Objectives

The primary objectives of this thesis are as follows:

1. To derive analytical insights into the structure of the risk-averse optimization problem, identifying properties that can inform algorithm design even when closed-form solutions are not available.

2. To develop and implement numerical algorithms to solve the nested optimization problem, ensuring that both the inner (policy optimization) and outer (threshold determination) problems are effectively addressed.

3. To construct and validate machine learning models that approximate the optimal risk-averse policies, potentially offering computational advantages for real-time applications.

4. To compare the performance and computational efficiency of the numerical and machine learning approaches, analyzing their strengths and limitations for real-time decision-making under uncertainty.

Chapter 3

Methodology

This chapter describes the simulation and optimization framework used in this study and its explicit solution. To make the discussion concrete, we assume a simplified cost function and state dynamics. Specifically, we consider:

- Terminal time: $T = 1$.
- Cost function: $C(x_t, a_t) = x_t^2 + a_t^2$, where x is the current state and a is action.
- State dynamics: The next state is given by: $x_{t+1} = x_t + a_t + \xi_t$.

where ξ is a random noise term modeled as a standard normal variable, i.e., $\xi \sim \mathcal{N}(0, 1)$, this noise captures the inherent uncertainty in the system dynamics.

3.1 Analytic Solution

In this section, we derive an analytical solution for the risk-averse optimization problem formulated as a nested minimization. We start with the general AVaR minimization problem (equation 2.2) and since $T = 1$, the infinite sum simplifies to just two terms:

$$\sum_{n=0}^{\infty} c(S_n, a_n) = c(S_0, a_0) + c(S_1, a_1) = (x_0)^2 + (a_0)^2 + (x_1)^2 + (a_1)^2$$

However, we only have one transition, a_1 does not exist, we substitute it by 0, and now everything becomes:

$$\sum_{t=0}^{T=1} c(S_n, a_n) = (x_0)^2 + (a_0)^2 + (x_0 + a_0 + \xi)^2$$

Now, the AVaR problem becomes:

$$\inf_{s\in\mathbb{R}} \left\{ s + \frac{1}{1-\alpha} \inf_{a} \mathbb{E}\left[\left((x_0)^2 + (a_0)^2 + (x_0 + a_0 + \xi)^2 - s \right)^+ \right] \right\}$$

For simplicity, we drop the subscript and denote x_0 as x and a_0 as a, giving us our final problem formulation:

$$\min_{s\geq 0} \left\{ s + \min_{a\in\mathbb{R}} \left\{ x^2 + a^2 + \frac{1}{1-\alpha} \mathbb{E}\left([(x + a + \xi)^2 - s]_+ \right) \right\} \right\} \tag{3.1}$$

We solve this nested optimization problem by working from the inner minimization to the outer one.

$$\min_{a\in\mathbb{R}} \left\{ x^2 + a^2 + \frac{1}{1-\alpha} \mathbb{E}\left([(x + a + \xi)^2 - s]_+ \right) \right\}$$

The most challenging part is computing the expectation term. To facilitate this calculation, we decompose the domain where $(x + a + \xi)^2 - s > 0$, which occurs when $x + a + \xi > \sqrt{s}$ or $x + a + \xi < -\sqrt{s}$.

Calculating the expectation

$$\begin{aligned}
\mathbb{E}\left\{ [(x + a + \xi)^2 - s]_+ \right\} &= \mathbb{E}\left\{ [(x + a + \xi)^2 - s] \cdot \mathbb{I}_{\{(x+a+\xi)^2 - s \geq 0\}} \right\} \\
&= \mathbb{E}\left\{ [(x + a + \xi)^2 - s] \cdot \mathbb{I}_{\left\{(x+a+\xi) \geq \sqrt{s}\right\}} \right\} \\
&+ \mathbb{E}\left\{ [(x + a + \xi)^2 - s] \cdot \mathbb{I}_{\left\{(x+a+\xi) \leq -\sqrt{s}\right\}} \right\}
\end{aligned}$$

Expressing these expectations as integrals where $\phi(\xi)$ represents the standard normal

density function ($\phi(\xi) = \frac{1}{\sqrt{2\pi}} e^{-\xi^2/2}$):

$$\mathbb{E}\left\{[(x+a+\xi)^2 - s]_+\right\} = \int_{\sqrt{s}-x-a}^{\infty} \left((x+a+\xi)^2 - s\right) \phi(\xi) d\xi + \int_{-\infty}^{-\sqrt{s}-x-a} \left((x+a+\xi)^2 - s\right) \phi(\xi) d\xi$$

Expanding the quadratic term:

$$\begin{aligned} &= \int_{\sqrt{s}-x-a}^{\infty} \left(x^2 + a^2 + 2ax - s + \xi^2 + 2x\xi + 2a\xi\right) \phi(\xi) d\xi \\ &+ \int_{-\infty}^{-\sqrt{s}-x-a} \left(x^2 + a^2 + 2ax - s + \xi^2 + 2x\xi + 2a\xi\right) \phi(\xi) d\xi \end{aligned}$$

By applying properties of the normal distribution, integrating term by term, and after simplification, we obtain:

$$\begin{aligned} &= (x^2 + a^2 + 2xa - s)[1 - \Phi(\sqrt{s} - x - a)] + 2(x+a)\phi(\sqrt{s} - x - a) \\ &+ (\sqrt{s} - x - a)\phi(\sqrt{s} - x - a) + 1 - \Phi(\sqrt{s} - x - a) \\ &+ (x^2 + a^2 + 2xa - s)\Phi(-\sqrt{s} - x - a) + (-2(x+a)\phi(-\sqrt{s} - x - a)) \\ &+ (\sqrt{s} + x + a)\phi(-\sqrt{s} - x - a) + \Phi(-\sqrt{s} - x - a) \end{aligned}$$

Putting it all together we get:

$$\begin{aligned} &\mathbb{E}\left\{[(x+a+\xi)^2 - s]_+\right\} = (x^2 + a^2 + 2ax - s + 1)[1 - \Phi(\sqrt{s} - x - a) + \Phi(-\sqrt{s} - x - a)] \\ &+ (\sqrt{s} + x + a)\phi(\sqrt{s} - x - a) - (-\sqrt{s} + x + a)\phi(-\sqrt{s} - x - a) \end{aligned}$$

where $\Phi(z) = \int_{-\infty}^{z} \phi(t) dt$ is a cumulative distribution function of the standard normal distribution.

Reformulating the objective function

Substituting this result into our inner minimization problem:

$$\min_a f(x,s) = \min_a \Big\{ x^2 + a^2 + \frac{1}{1-\alpha}\left[(x^2+a^2+2ax-s+1)[1-\Phi(\sqrt{s}-x-a)+\Phi(-\sqrt{s}-x-a)]\right.$$
$$\left.+(\sqrt{s}+x+a)\phi(\sqrt{s}-x-a)-(-\sqrt{s}+x+a)\phi(-\sqrt{s}-x-a)\right]\Big\}$$

To determine the optimal value of a, we differentiate $f(x,s)$ with respect to a and set the result to zero:

$$\frac{d}{da}f(x,s) = 2a + \frac{1}{1-\alpha}\left\{(2a+2x)[1-\Phi(u)-\Phi(v)]+(x^2+a^2+2ax-s+1)[\phi(u)-\phi(v)]\right.$$
$$\left.+\phi(u)+(\sqrt{s}+x+a)(u\cdot\phi(u))-(\phi(v)+(-\sqrt{s}+x+a)(v\cdot\phi(v))\right\}=0$$

where we've substituted $u = \sqrt{s} - x - a$ and $v = -\sqrt{s} - x - a$ for brevity.

This complex expression does not yield a simple closed-form solution. Therefore, we resort to numerical methods to find the optimal value of a for given parameters.

3.2 Numerical Solution

Due to the complexity of the analytical expressions, two computational methods were implemented to solve this optimization problem: a Monte Carlo approach and an analytical method.

3.2.1 Direct Numerical Optimization

To solve the nested optimization problem, we implement a direct numerical approach using optimization algorithms from scientific computing libraries (SciPy). Our implementation consists of two key components: an inner optimization that finds the optimal decision variable a^* for a given state variable x and threshold s, and an outer optimization that determines the optimal threshold s^* to minimize the overall objective function.

The inner optimization problem is defined by minimizing the function $f(x, s)$, which represents the expected cost function given threshold s and state x. Mathematically, this is expressed as:

$$f(x,s) = \arg\min_{a\in\mathbb{R}} \left\{ x^2 + a^2 + \frac{1}{1-\alpha}\mathbb{E}\left([(x+a+\xi)^2 - s]_+\right) \right\}$$

The expectation term involves both the cumulative distribution function (CDF) and probability density function (PDF) of the standard normal distribution, making the problem nonlinear and analytically challenging. We use the "Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Box constraints" (L-BFGS-B) optimization algorithm to numerically determine a^*. The algorithm iteratively updates a to minimize the cost function, starting from an initial guess $a_0 = 0$.

Once we can compute a^* for any given x and s, the outer optimization seeks to determine the optimal threshold s^* that minimizes the total objective function:

$$V(x) = \min_{s\geq 0} \{s + f(a^*, s, x)\}$$

This function captures both the direct cost of s and the expected cost from the inner optimization. The optimization is subject to the constraint $s \geq 0$, ensuring feasibility. We employ the "L-BFGS-B' algorithm, which is well-suited for constrained optimization problems. The algorithm begins with an initial guess $s = 0$ and iteratively adjusts s, using gradient-based updates, to achieve the minimal value of the outer objective function.

Solution Procedure

The complete solution is obtained through the following sequential process:

1. Inner Optimization: For a given state x and a candidate threshold s, solve for a^* using $L-BFGS-B$.

2. Outer Optimization: For a given state x, minimize the outer objective function over s using $L-BFGS-B$, subject to $s \geq 0$. This step involves multiple evaluations

Algorithm 1 Nested Optimization for Finding a^* and s^*

function INNER FUNCTION(a, s, x, α)
    $u \leftarrow \sqrt{s} - x - a$
    $v \leftarrow -\sqrt{s} - x - a$
    Compute terms using $\Phi(u)$, $\Phi(v)$, and $\phi(u)$, $\phi(v)$
    return $x^2 + a^2 + \frac{1}{1-\alpha}$(terms)
end function
function FIND OPTIMAL A(s, x, α)
    return $\arg\min_a$ INNER FUNCTION(a, s, x, α) using (L-BFGS-B)
end function
function OUTER OBJECTIVE(s, x, α)
    $a^* \leftarrow$ FIND OPTIMAL A(s, x, α)
    return $s+$ INNER FUNCTION(a^*, s, x, α)
end function
Define grid: $x \in [0, 20]$ (step 0.25), $s \in [0, 50]$ (step 0.25)
for all x in x-values **do**
    for all s in s-values **do**
        $a^* \leftarrow$ FIND OPTIMAL A(s, x, α)
        Solve $s^* \leftarrow \arg\min_s$ OUTER OBJECTIVE(s, x, α) using (L-BFGS-B)
        Store a^*, s^*, and objective value
    end for
end for

of the inner optimization.

3. Final Computation: Using the optimal threshold s^* and a^* for (x, s), compute the the final objective value $V(x)$.

It is important to note that for each state x, a complete nested optimization must be performed. During the outer optimization process, many different candidate values of s are evaluated, with each evaluation requiring a full solution of the inner optimization problem to find the corresponding a^*. The optimization algorithms efficiently navigate this nested structure to converge to the optimal solution.

To analyze how the optimal solution varies with the state variable x, we perform a parameter sweep across a range of x values from 0 to 20.0 and s values from 0 to 50 with a step size of 0.25. For each value of (x, s), we compute the optimal action a^*, and then using the results we compute the optimal threshold s^* and objective function value $V(x)$. This comprehensive analysis provides insights into the solutions' sensitivity to different state values and reveals important patterns in the optimal

policy.

3.2.2 Monte Carlo Simulation Approach

In addition to the analytical method described above, we implemented a Monte Carlo simulation approach to validate our results and provide an alternative solution method. The Monte Carlo method approximates the expectation term in the objective function through random sampling from the distribution of ξ. Instead of using the analytical expressions involving the normal CDF and PDF, this approach directly evaluates the expected value by averaging over a large number of random samples:

Algorithm 2 Monte Carlo Approximation for Inner Function

function INNER FUNCTION MONTE CARLO($a, s, x, \alpha, N_{\text{samples}}$)
    Generate N_{samples} random samples $\xi_i \sim \mathcal{N}(0, 1)$
    Compute $\text{term}_i = \max((x + a + \xi_i)^2 - s, 0)$ for each sample
    expectation $\leftarrow \frac{1}{N_{\text{samples}}} \sum_{i=1}^{N_{\text{samples}}} \text{term}_i$
    return $x^2 + a^2 + \frac{1}{1-\alpha} \cdot \text{expectation}$
end function

For our implementation, we used $N_{\text{samples}} = 10^5$ to ensure an accurate estimation of the expectation. To maintain result consistency across multiple runs, we fixed the random seed (seed = 46). The Monte Carlo implementation follows the same nested optimization structure as the analytical approach:

$$f(x, s) = \arg\min_{a \in \mathbb{R}} \left\{ x^2 + a^2 + \frac{1}{1-\alpha} \frac{1}{N_{\text{samples}}} \sum_{i=1}^{N_{\text{samples}}} \max((x + a + \xi_i)^2 - s, 0) \right\}$$

$$V(x) = \min_{s \geq 0} \{ s + f_{\text{MC}}(a^*, s, x) \}$$

where f_{MC} represents the inner objective function evaluated using the Monte Carlo method.

This method can be used as a benchmark, by comparing results from both meth-

ods, we can validate the correctness of our analytical implementation. Moreover, it can be readily adapted to handle arbitrary distributions for ξ, not just the standard normal distribution.

However, the Monte Carlo approach also has limitations, such as the accuracy of the method depends on the number of samples used, which introduces statistical error into the optimization. In addition, for high-precision results, a large number of samples is required, which can be computationally intensive.

To address these concerns, we implemented both methods and conducted a comparative analysis. The analytical method was used as the primary approach for generating our results, while the Monte Carlo method served as a verification tool and provided insights into the robustness of our solutions. For the parameter exploration, we used the same grid structure for both methods, allowing for direct comparison of results. This dual-method approach strengthens the confidence in our numerical findings and provides a more comprehensive understanding of the optimization problem's characteristics.

3.3 Machine Learning Approximation

While the numerical optimization approach provides accurate solutions, it is computationally expensive for real-time decision-making where repeated optimizations are required. Solving the nested optimization problem for a single-state x requires iterative methods, making it impractical for large-scale applications.

To address this limitation, we develop a machine learning model that approximates the mapping from x, s to optimal decisions a^* and from x to s^* and objective values $V(x)$. By training these models on precomputed solutions from numerical optimization, we can achieve near-instantaneous inference while preserving accuracy.

3.3.1 Model Selection Process

To efficiently approximate the optimal solutions without repeatedly solving the computationally intensive nested optimization problem, we developed polynomial regres-

sion models. These models learn the relationships between the x, s and optimal action a^*, and between state variable x and the optimal threshold s^*, objective values $V(x)$. The trained models can then be used to predict optimal solutions for any value of x, s within the studied range with minimal computational cost.

We evaluated several machine learning algorithms to approximate our optimization functions, including neural networks, random forests, gradient boosting and polynomial regression. After comprehensive benchmark testing across these methods, polynomial regression was selected for several reasons:

Interpretability: The resulting polynomial models offer closed-form analytical expressions that are interpretable and can provide insights into the underlying mathematical structure of the optimal policies.

Computational Efficiency: At inference time, polynomial models require minimal computational resources compared to deep learning approaches, making them suitable for embedded systems and real-time applications.

Performance: For our specific problem structure, polynomial regression achieved comparable accuracy to more complex models when appropriate regularization was applied.

Analytical Properties: The smooth, continuous nature of polynomial approximations preserves important mathematical properties of the original optimization problem, facilitating further theoretical analysis.

3.3.2 Polynomial Regression Models

To efficiently approximate the optimal solutions without repeatedly solving the computationally intensive nested optimization problem, we developed polynomial regression models. These models learn the relationships between the state variable x, threshold parameter s, and the optimal action a^*, as well as between x and the optimal threshold s^* and objective values $V(x)$.

Model Formulation and Training

We employed polynomial regression with different degrees and feature combinations based on the hierarchical structure of the optimization problem:

Model for predicting a^*: Recognizing the dependency of a^* on both x and s, we constructed a multivariate polynomial regression model that takes both variables as input features:

$$a^*(x, s) \approx \gamma_0 + \gamma_1 x + \gamma_2 s + \gamma_3 x^2 + \gamma_4 xs + \gamma_5 s^2 + \cdots = \sum_{i,j} \gamma_{ij} x^i \cdot s^j$$

This approach respects the hierarchical structure of the optimization problem, where a^* is determined by solving the inner optimization problem with both x and s as parameters.

Model for predicting s^*: We used a polynomial regression model with x as the sole input feature:

$$s^*(x) \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots = \sum_{i=0}^{n} \beta_i \cdot x^i$$

This model effectively captures the nonlinear relationship between the state variable x and the optimal threshold s^*.

Model for predicting objective values: For the objective function values, we also employed a polynomial regression model with x as the input:

$$V(x) \approx \delta_0 + \delta_1 x + \delta_2 x^2 + \delta_3 x^3 + \delta_4 x^4 + \cdots = \sum_{i=0}^{n} \delta_i \cdot x^i$$

This model accommodates the complex relationship between x and the objective function value.

Feature Engineering and Model Complexity

The selection of appropriate polynomial degrees for each model involved a careful balance between model complexity and generalization capability:

Cross-interaction terms: For the $a^*(x, s)$ model, we included cross-terms (e.g., xs, x^2s, $xs^2, ...$) to capture the interdependencies between state and threshold variables. These interaction terms proved crucial for accurately modeling the decision boundaries in the optimization space.

Degree selection: We performed systematic degree selection using cross-validation techniques to determine the optimal polynomial degree for each model:

- For $a^*(x, s)$, a 6th-degree polynomial provided the best balance of accuracy and generalization.
- For $s^*(x)$ and $V(x)$, 10th-degree polynomials were required to capture the more complex nonlinear relationships.

The models were trained using data from a parameter sweep across x values ranging from 0 to 20 and s values from 0 to 50, with each data point representing the solution to the complete nested optimization problem at that specific parameter combination.

Comparison with Alternative Approaches

While polynomial regression proved most effective for our specific problem, we also evaluated several alternative machine learning approaches:

Neural Networks: We implemented fully connected neural networks with varying architectures and activation functions. While these models are capable of approximating highly nonlinear relationships, they require substantially more training time compared to polynomial regression. This was due to the need for iterative optimization over multiple epochs and careful tuning of hyperparameters. Moreover, achieving comparable accuracy necessitated larger training datasets.

Decision Tree Ensembles: We also evaluated ensemble methods such as Random Forests and Gradient Boosted Trees. These models showed good predictive performance and robustness to overfitting. However, like neural networks, they incurred higher training times and required more memory compared to polynomial models. Additionally, their piecewise-constant prediction structure introduced small

discontinuities in the output, which can be problematic in optimization contexts that benefit from smooth functional approximations.

The polynomial regression approach ultimately provided the best balance of accuracy, interpretability, and computational efficiency for approximating our optimization functions.

Chapter 4

Numerical Experiments and Results

This chapter presents the experimental study of a risk-sensitive optimization problem, combining dynamic programming and machine learning techniques. The problem is first solved analytically, yielding expressions for the optimal threshold s^*, value function $V(x)$, and optimal action a^*. These analytical results are then validated through Monte Carlo simulations to assess their robustness and accuracy. To further enhance computational efficiency, polynomial regression models are trained on the computed data to approximate the optimal policies. The performance of these models is evaluated by comparing their predictions against the original solutions, both visually and quantitatively. This integrated approach provides insight into how risk sensitivity shapes decision-making and demonstrates the effectiveness of using machine learning models to approximate complex optimization behavior with high precision and low computational cost.

4.1 Analysis of Numerical Results

In this section, we present and analyze the numerical results of the risk-sensitive optimization problem, solved using both analytical methods and Monte Carlo simulations. The analysis focuses on the behavior of the optimal value function $V(x)$, optimal threshold s^*, and optimal action a^*, across a range of state values x, threshold values s, and different levels of risk aversion parameter α.

Optimal Action Surface a^* vs x vs s

Figures in 4-1 show a 3D visualization of the optimal action a^*, revealing its structure across the (x, s) space. For a fixed threshold s, we observe that the optimal action $a^*(x, s)$ becomes more negative as x increases. This behavior strategically counterbalances the x component in the quadratic cost term $(x + a + \xi)^2$, effectively minimizing expected losses.

Additionally, the graphs demonstrate that as the threshold s increases, the optimal action gradually approaches zero. This pattern indicates that when a sufficient threshold is available, the need for aggressive corrective actions diminishes. The surface displays a distinctive sharp transition zone where the action changes rapidly in response to small variations in state and threshold. This characteristic reflects the non-linearity introduced by the positive-part operator and risk-sensitive expectation in the optimization problem.

As α increases from 0.4 to 0.99, the overall structure of the optimal solution remains consistent, though subtle changes in the transition zone can be observed. This consistency across different risk aversion levels suggests a robust solution structure to the underlying optimization problem.

The color gradient from yellow (higher values) to purple (lower values) illustrates the range of optimal actions, with values spanning approximately from 0 down to -10. This visualization effectively captures how the optimal decision variable 'a' should be selected given different combinations of the state variable x and threshold parameter s under varying levels of risk aversion α.

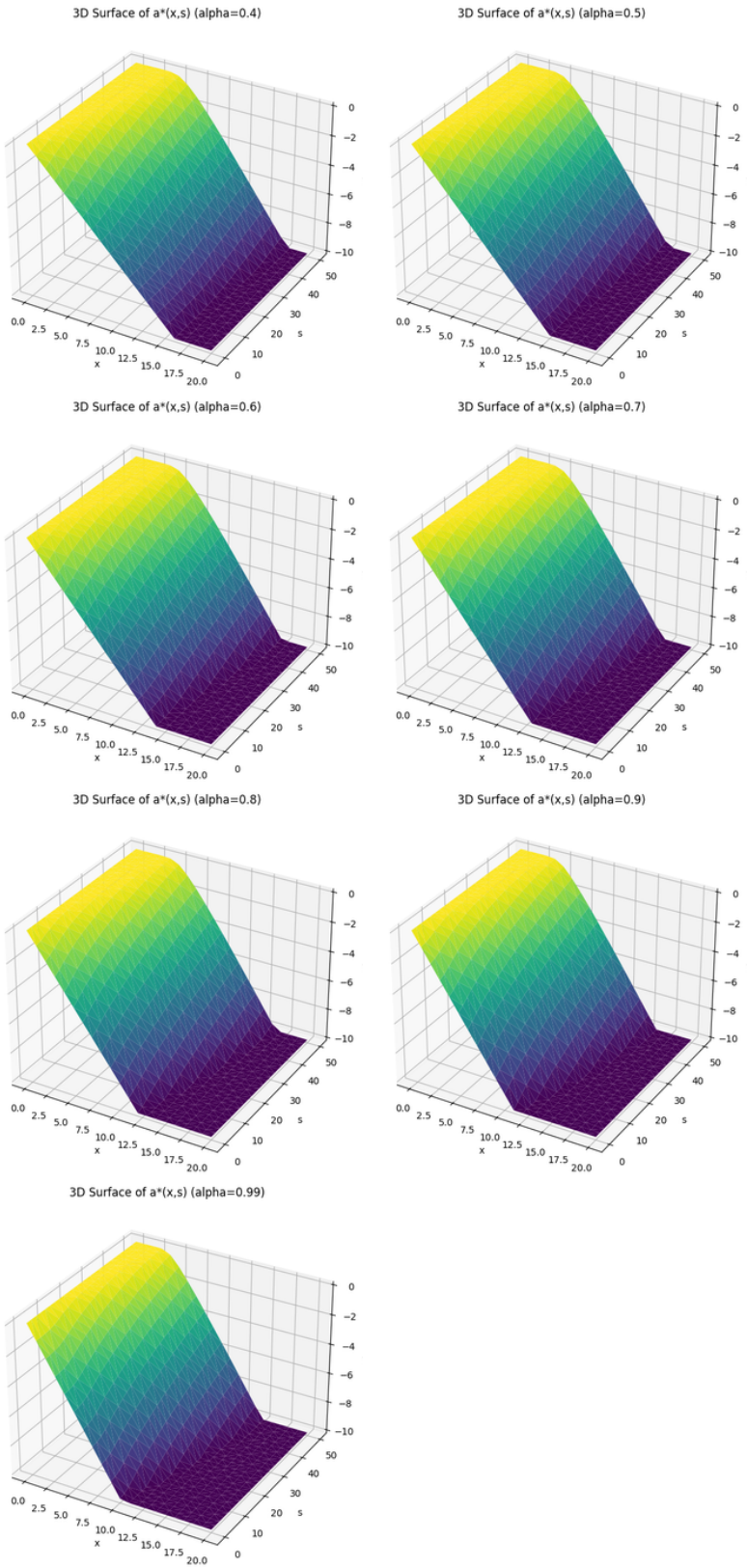

Figure 4-1: Results of a^* for different alphas

Optimal Threshold s^* and Value Function $V(x)$ Across Risk Parameters

Figures in 4-2 present a comparative overview of s^* (left column) and $V(x)$ (right column) for values of the risk-aversion parameter $\alpha \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$. Several important trends emerge:

Monotonicity and Saturation: For all values of α, the optimal threshold s^* increases monotonically with x, before saturating near the imposed upper bound of $s^* = 50$. This reflects the growing need for a higher risk threshold as the state x increases.

Sensitivity to Risk Aversion: As α increases, the transition in s^* becomes noticeably steeper. The threshold grows faster in the intermediate range of $x \in [7, 15]$, indicating heightened sensitivity to risk in this region.

Anomalous Behavior at High α: For $\alpha = 0.99$, an irregular spike in the s^* curve is observed near $x = 2$, signifying extreme sensitivity to tail events under near-AVaR conditions. This instability reflects the dominance of rare, high-impact outcomes.

Value Function Growth: Across all α, the value function $V(x)$ increases rapidly with x, with the rate of growth accelerating as $\alpha \to 1$. The function transitions from a moderately convex shape to near-exponential behavior.

Impact of Risk Aversion on Cost: At $x = 20$, the value of $V(x)$ increases dramatically with risk aversion: from roughly 600 at $\alpha = 0.4$ to over 5000 at $\alpha = 0.99$. This illustrates how even modest increases in α can drastically amplify the perceived cost due to rare but severe losses.

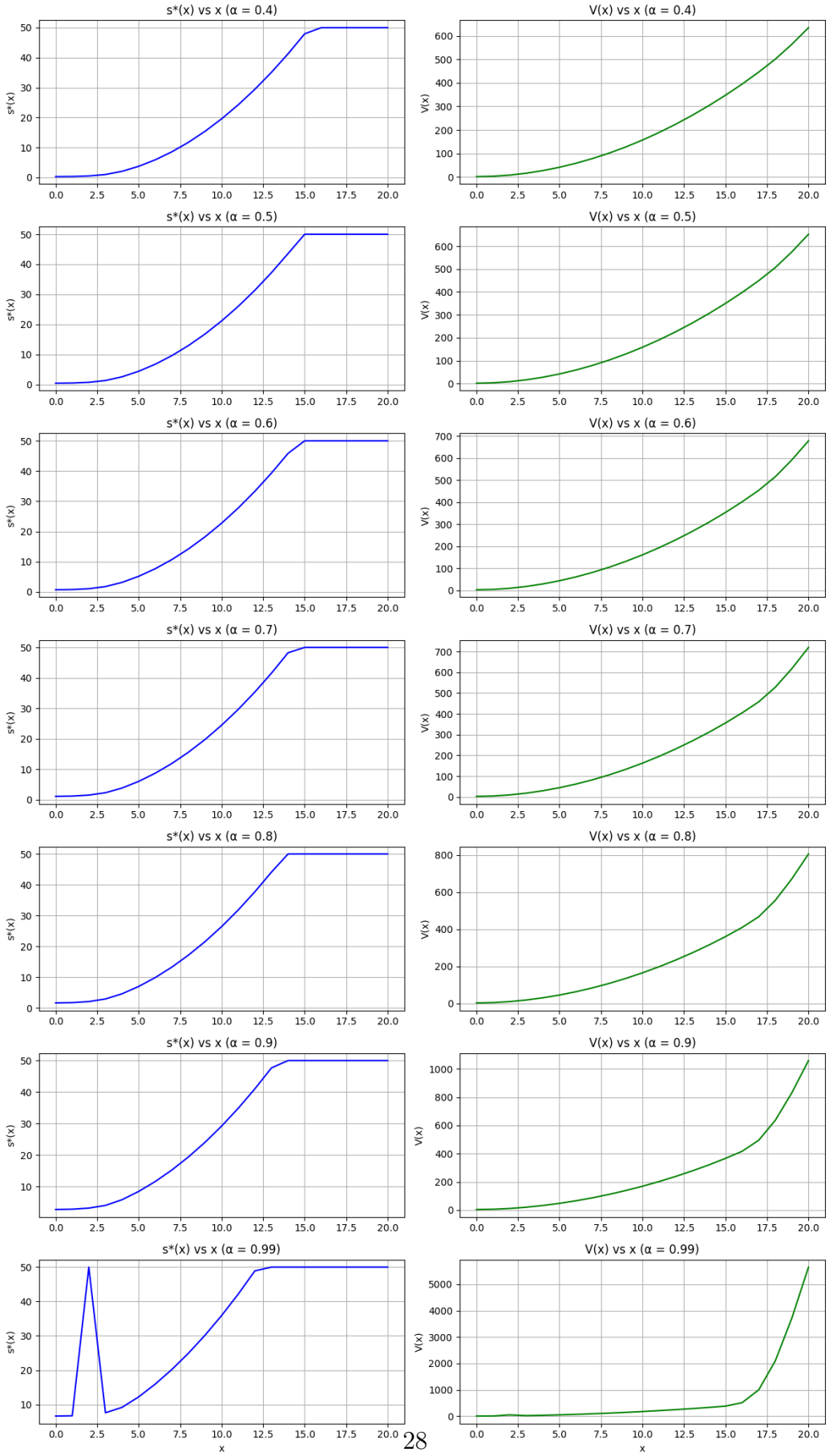

Figure 4-2: Results of $s^*, V(x)$ for different alphas

Analytical vs. Monte Carlo Simulation Comparison

Figures 4-3, 4-4 and 4-5 provide a detailed comparison between the analytical solution and the Monte Carlo-based approximation for $\alpha = 0.95$. The Monte Carlo estimation of s^* (Figure 4-4, left) aligns closely with the analytical result (Figure 4-3, left), with a mean absolute error of only 0.0802. Both follow the characteristic sigmoid-like shape and saturate at the maximum allowed threshold. The value function $V(x)$ obtained from the Monte Carlo simulation (Figure 4-4, right) mirrors the shape and scale of the analytical result (Figure 4-3, right), demonstrating a consistent exponential growth pattern while maintaining a mean absolute error of 0.3135. Moreover, to assess the agreement between the analytical and the Monte Carlo methods, we conducted a point-wise comparison across the (x, s) domain(Figure 4-5 and Figure 4-6). The mean absolute error between the Monte Carlo simulation and the analytical method was ≈ 0.003, indicating no statistically significant difference between the two methods, confirming the robustness of our analytical approach. The tight correspondence between the two methods validates the correctness of the analytical approach and confirms the reliability of the simulation, even for high α values where rare events dominate the cost function.

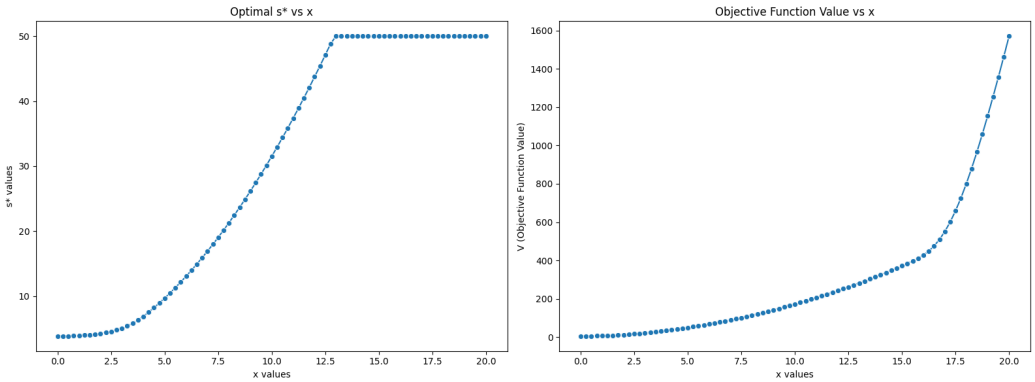

Figure 4-3: Results of s^* and $V(x)$ using analytical method

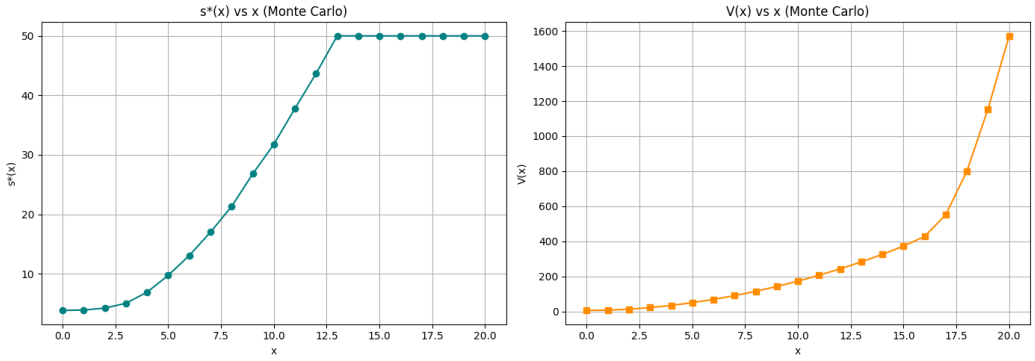

Figure 4-4: Results of Monte Carlo simulation for s^* and $V(x)$

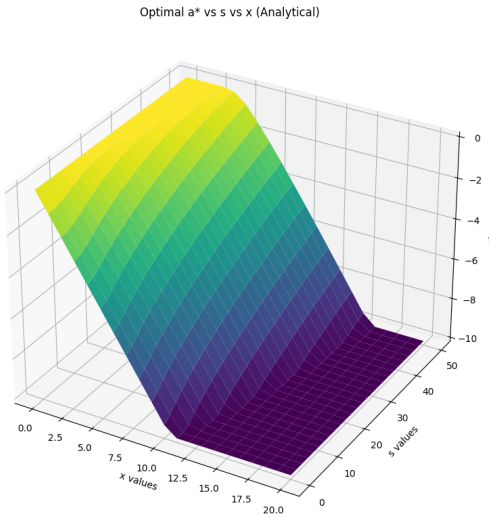

Figure 4-5: Results of a^* using analytical method

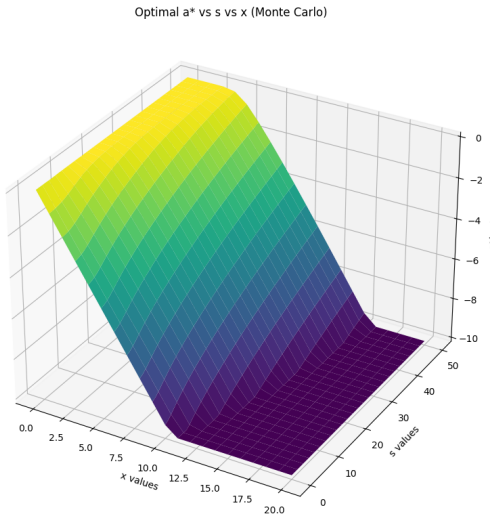

Figure 4-6: Results of Monte Carlo simulation for a^*

These results confirm that the analytical and Monte Carlo approaches produce equivalent results within acceptable margins of error, validating both methodologies even for high α values where rare events dominate the cost function. These results demonstrate that the risk-sensitive optimization framework effectively models the increasing conservatism required under higher risk aversion. As α approaches 1, the system becomes increasingly sensitive to tail events, leading to:

- Larger optimal thresholds s^*,
- More conservative (negative) actions a^*,

- Rapidly increasing value functions $V(x)$, and
- Greater non-linearity and volatility in decision surfaces.

Together, these insights confirm the theoretical expectations of risk-sensitive decision-making and highlight the trade-offs between risk aversion and cost.

4.2 Polynomial Regression Results

The polynomial regression models demonstrate outstanding performance in approximating the core functions of the risk-sensitive optimization problem. Despite the inherent complexity and nonlinearity of the original formulations, the models achieve high predictive accuracy, enabling efficient and practical approximations of optimal decision rules. As shown in Table 4.1, all models exhibit near-perfect fit, with R^2 values near 1.

Table 4.1: Performance Metrics of Final Models

Model	**MSE**	$\boldsymbol{R^2}$	**Degree**
$a^*(x, s)$	0.0198	0.9987	6
$s^*(x)$	0.1647	0.9995	10
$V(x)$	12.8737	0.9999	10

Optimal Action Function $a^*(x, s)$

The polynomial regression for $a^*(x, s)$ achieves an accurate fit, with an R^2 value of 0.9987 and a mean squared error (MSE) of 0.0198. As illustrated in Figure 4-7, the predicted surface (right panel) closely mirrors the computed action values (left panel), effectively capturing the nonlinear interactions between state x, threshold s, and optimal action a^*. Crucially, the model preserves key behavioral characteristics:

- A consistent decrease in a^* with increasing x, reflecting the compensatory strategy inherent in the cost minimization.
- A flattening of the surface as s increases, indicating reduced need for aggressive action when larger buffers are in place.

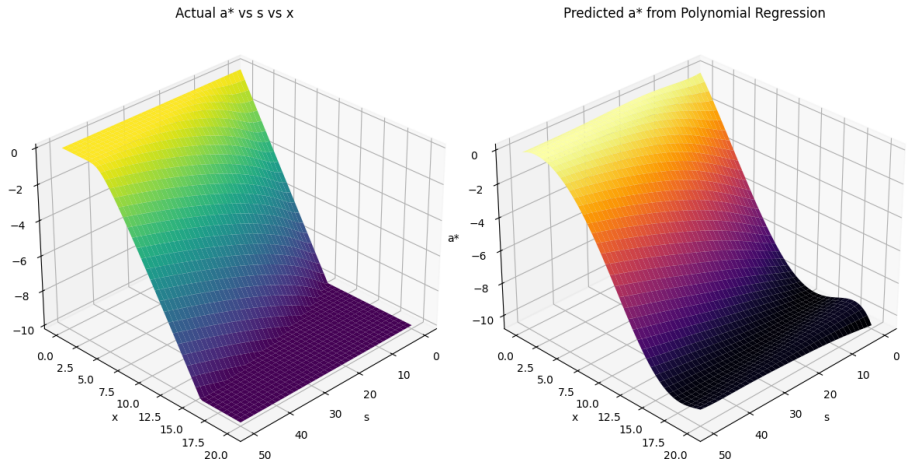

Figure 4-7: Actual results vs Polynomial Regression Model's results

Optimal Threshold Function $s^*(x)$

The 10th-degree polynomial regression approximates the optimal threshold function with high accuracy, yielding an R^2 of 0.9995 and an MSE of 0.1647. In Figure 4-8 (left panel), the predicted curve (black) aligns closely with the computed data points (green), effectively replicating the sigmoid-like structure of the original function. The model captures:

- The rapid transition region between $x = 7$ and $x = 15$, where the optimal threshold rises sharply.
- The plateau effect near $s^* = 50$, which corresponds to the imposed maximum threshold.

Value Function $V(x)$

The polynomial model for $V(x)$ achieves the highest fit among all three functions, with an R^2 of 0.9999 and MSE of 12.8737. Despite the relatively larger error magnitude, this is expected due to the exponential scale of $V(x)$ at higher x values. As shown in Figure 4-8 (right panel), the predicted curve closely follows the computed data across the full domain, faithfully capturing the rapid growth in cost under increasing state values and risk sensitivity.

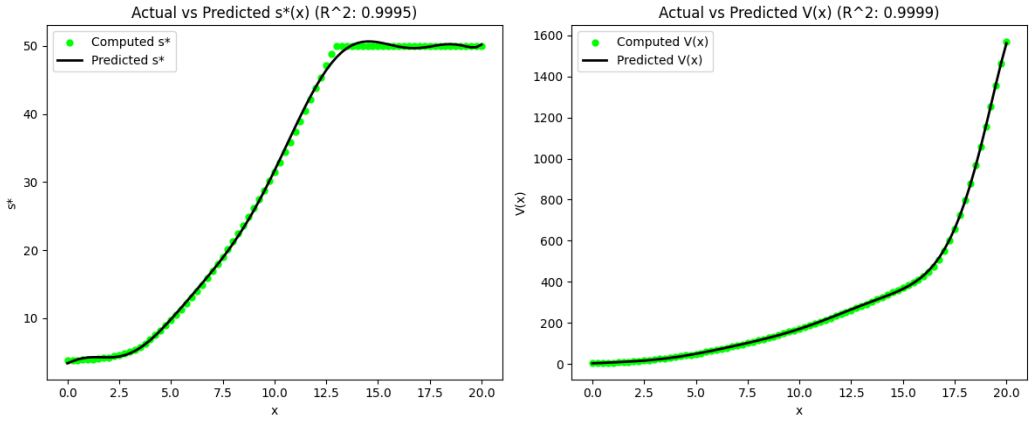

Figure 4-8: Results of Polynomial Regression Model

While the resulting polynomial expressions are relatively high in degree and contain numerous terms, they offer closed-form approximations that eliminate the need for computationally intensive numerical optimization during runtime. This provides a significant computational advantage, especially in real-time or resource-constrained environments.

Overall, these results confirm that polynomial regression is a viable and highly accurate modeling approach for risk-sensitive optimization problems. The models retain essential structural features of the optimal policies while enabling fast, interpretable, and efficient implementation of complex risk-aware decision strategies. The comprehensive statistical validation across multiple metrics strengthens confidence in the machine learning models' reliability for practical applications.

Chapter 5

Conclusion

This thesis has explored the integration of AVaR into ADP frameworks for solving MDPs under uncertainty. Our analytical methodology successfully captured the complex relationship between state variables, risk thresholds, and optimal decisions. The comparison between analytical solutions and Monte Carlo simulations confirmed the robustness of our approach, with negligible errors between the two methods. The experimental results reveal several important insights into risk-sensitive decision-making. Furthermore, polynomial regression models developed in this study provide an efficient alternative to direct numerical optimization, achieving exceptional accuracy. These polynomial regression models preserve the essential structural features of the optimal policies while enabling fast implementation, making them valuable for real-time applications in resource-constrained environments.

Future research directions could include extending the current framework to handle more complex state dynamics, exploring alternative risk measures, and developing adaptive algorithms that can adjust risk sensitivity based on environmental feedback.

In conclusion, this work bridges the stochastic optimal control problem with computational methods to create practical solutions for decision-making problems while using machine learning methods to predict the values of optimal action, optimal threshold and objective values.

Chapter 6

Appendix

```
# Set seed and generate samples once
np.random.seed(46)

xi_samples = 10**5
xi = np.random.normal(0, 1, xi_samples)

alpha = 0.95
x_val = 20

# Monte Carlo inner expectation
def inner_func_Monte(a, s, x, alpha=alpha):
    term = np.maximum((x + a + xi)**2 - s, 0)
    expectation = np.mean(term)
    return x**2 + a**2 + (1/(1 - alpha)) * expectation

# Minimize over a for fixed s
def optimal_a(s, x=x_val, alpha=alpha):
    res = minimize(inner_func_Monte, x0=0, args=(s, x, alpha),
                   method='L-BFGS-B', bounds=[(-10,10)])
    return res.x[0]

```

```
# Outer function: s + minimized inner value
def outer_func(s, x=x_val, alpha=alpha):
    a_star = optimal_a(s, x, alpha)
    return s + inner_func_Monte(a_star, s, x, alpha)

# Solve for s*
bounds_s = [(0, 50)]  # s >= 0 and s <= 50

a_optimal_Monte = []
s_optimal_Monte = []
objective_values_Monte = []

x_values = np.arange(0.0, 20.25, 0.25)
s_values = np.arange(0.0, 50.25, 0.25)

# For all values of x and s find a*, s* and the optimal value
for x in x_values:
    for s in s_values:
        res_s = minimize(outer_func, x0=0.1, bounds=bounds_s,
           args=(x, 0.95), method='L-BFGS-B')
        a_opt = optimal_a(s, x, 0.95)
            a_optimal_Monte.append(a_opt)
        s_optimal_Monte.append(res_s.x[0])
        objective_values_Monte.append(res_s.fun)
```

Listing 6.1: Monte Carlo Simuation

```
# Define the inner function
def inner_func(a, s, x=1, alpha=0.95):
    u = np.sqrt(s) - x - a
    v = -np.sqrt(s) - x - a

    term1 = (x**2 + a**2 + 2*x*a - s + 1) * (1 - stats.norm.cdf
```

```
        (u) + stats.norm.cdf(v))
    term2 = (np.sqrt(s) + x + a) * stats.norm.pdf(u)
    term3 = (np.sqrt(s) - x - a) * stats.norm.pdf(v)

    result = x**2 + a**2 + (1 / (1 - alpha)) * (term1 + term2 +
        term3)
    return result

# Define function to find optimal a* for given s
def optimal_a(s, x=1, alpha=0.95):
    res = minimize(inner_func, x0=0, args=(s, x, alpha), method
        ='L-BFGS-B', bounds=[(-10, 10)])
    return res.x[0]

# Define the outer function
def outer_func(s, x=1, alpha=0.95):
    a_star = optimal_a(s, x, alpha)
    return s + inner_func(a_star, s, x, alpha)

# Solve for s*
bounds_s = [(0, 50)]  # s >= 0 and s <= 50

a_optimal = []
s_optimal = []
objective_values = []

x_values = np.arange(0.0, 20.25, 0.25)
s_values = np.arange(0.0, 50.25, 0.25)

# For all values of x and s find a*, s* and the optimal value
for x in x_values:
    for s in s_values:
```

```
        res_s = minimize(outer_func, x0=0.1, bounds=bounds_s,
            args=(x, 0.95), method='L-BFGS-B')
        a_optimal.append(optimal_a(s, x, 0.95))
        s_optimal.append(res_s.x[0])
        objective_values.append(res_s.fun)
```

Listing 6.2: Using Analytical Solution

```
alpha_values = [0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.99]
results = {}

# Loop over alpha values
for alpha in alpha_values:
    a_optimal = []
    s_optimal = []
    objective_values = []

    print(f"Processing alpha = {alpha}...")

    for x in x_values:
        for s in s_values:
            res_s = minimize(outer_func, x0=0.1, bounds=
                bounds_s, args=(x, alpha), method='L-BFGS-B')
            a_optimal.append(optimal_a(s, x, alpha))
            s_optimal.append(res_s.x[0])
            objective_values.append(res_s.fun)

    # Save df1
    df1 = pd.DataFrame({
        'x': np.repeat(x_values, len(s_values)),
        's': np.tile(s_values, len(x_values)),
        'a*(x,s)': a_optimal
    })
```

```
25
26      # Save df2
27      df2_data = []
28      num_s = len(s_values)
29
30      for i, x in enumerate(x_values):
31          s_slice = s_optimal[i*num_s:(i+1)*num_s]
32          obj_slice = objective_values[i*num_s:(i+1)*num_s]
33          min_idx = np.argmin(obj_slice)
34
35          df2_data.append({
36              'x': x,
37              's*(x)': s_slice[min_idx],
38              'V(x)': obj_slice[min_idx]
39          })
40
41      df2 = pd.DataFrame(df2_data)
42
43      # Store results by alpha
44      results[alpha] = {
45          'df1': df1,
46          'df2': df2
47      }
```

Listing 6.3: Solving for different alphas

```
1  # Function to fit polynomial regression and make predictions
2  def poly_reg(X, y, degree):
3      polynomial = PolynomialFeatures(degree)
4      X_poly = polynomial.fit_transform(X)
5
6      model = LinearRegression()
7      model.fit(X_poly, y)
```

```
    y_pred = model.predict(X_poly)

    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    return model, poly, y_pred, mse, r2

# Prepare feature matrices
X_a = df1[['s', 'x']].values  # Features for predicting a*
y_a = df1['a*(x,s)'].values  # Target a*

X_s = df2[['x']].values  # Feature for predicting s*
y_s = df2['s*(x)'].values  # Target s*

X_V = df2[['x']].values  # Feature for predicting V
y_V = df2['V(x)'].values  # Target V

# Train polynomial regression models
model_a, poly_a, y_a_pred, mse_a, r2_a = poly_reg(X_a, y_a,
   degree = 6)
model_s, poly_s, y_s_pred, mse_s, r2_s = poly_reg(X_s, y_s,
   degree = 10)
model_V, poly_V, y_V_pred, mse_V, r2_V = poly_reg(X_V, y_V,
   degree = 10)

# Print performance metrics
print(f"a* Prediction: MSE = {mse_a:.4f}, R   = {r2_a:.4f}")
print(f"s* Prediction: MSE = {mse_s:.4f}, R   = {r2_s:.4f}")
print(f"V Prediction: MSE = {mse_V:.4f}, R   = {r2_V:.4f}")
```

Listing 6.4: Polynomial Regression Model

```
# Generate polynomial formula strings
```

```
def get_polynomial_formula(model, poly, feature_names):
    """Generate a string representation of the polynomial
       formula"""
    coefficients = model.coef_
    intercept = model.intercept_
    powers = poly.powers_

    formula = f"{intercept:.6f}"

    for i, coef in enumerate(coefficients[1:]):  # Skip the
       intercept term
        power = powers[i+1]  # Skip the first row which is for
           the intercept
        term = f"{coef:.6f}"

        for j, p in enumerate(power):
            if p > 0:
                term += f" * {feature_names[j]}"
                if p > 1:
                    term += f"^{p}"

        formula += f" + ({term})"

    return formula

# Generate formulas
a_formula = get_polynomial_formula(model_a, poly_a, ['s', 'x'])
s_formula = get_polynomial_formula(model_s, poly_s, ['x'])
V_formula = get_polynomial_formula(model_V, poly_V, ['x'])

print("\nPolynomial Formulas:")
print(f"\na*(x,s) = {a_formula}")
```

```
31 print(f"\ns*(x) = {s_formula}")
32 print(f"\nV(x) = {V_formula}")
```

Listing 6.5: Polynomial Functions

Bibliography

[1] Bäuerle, N., and Glauner, A. (2021). Markov decision processes with recursive risk measures. European Journal of Operational Research. 296(3), 953-966. https://doi.org/10.1016/j.ejor.2021.04.030

[2] Bäuerle, N., and Ott, J. (2011). Markov decision processes with average-value-at-risk criteria. Mathematical Methods of Operations Research, 74(3), 361–379. https://doi.org/10.1007/s00186-011-0367-0

[3] Bäuerle, N., and Rieder, U. (2013). More risk-sensitive Markov decision processes. Mathematics of Operation Research, 39(1), 105-120. https://doi.org/10.1287/moor.2013.0601

[4] Carpin, S., Chow, Y., and Pavone, M. (2016). Risk aversion in finite Markov decision processes using total cost criteria and average value at risk. In 2016 IEEE International Conference on Robotics and Automation (ICRA). 335-342. https://doi.org/10.1109/icra.2016.7487152

[5] Liu, Q., Ching, W., Zhang, J., and Wang, H. (2021). An average-value-at-risk criterion for Markov decision processes with unbounded costs. Frontiers of Mathematics in China, 17(4), 673–687. https://doi.org/10.1007/s11464-021-0944-3

[6] Mes, M., and Rivera, A. E. P. (2017). Approximate dynamic programming by practical examples. In R. J. Boucherie and N. M. van Dijk (Eds.), Markov Decision Processes in practice (pp. 63–101). Springer. https://doi.org/10.1007/978-3-319-47766-4_3

[7] Uğurlu, K. (2017). Controlled Markov decision processes with AVaR criteria for unbounded costs. Journal of Computational and Applied Mathematics, 319, 24–37. https://doi.org/10.1016/j.cam.2016.11.052

[8] Uğurlu, K. (2018). Robust optimal control using conditional risk mappings in infinite horizon. Journal of Computational and Applied Mathematics, 344, 275–287. https://doi.org/10.1016/j.cam.2018.05.030

[9] Yoshida, Y., and Kumamoto, S. (2019). Dynamic average value-at-risk allocation on worst scenarios in asset management. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 11436 LNCS, pp. 674–683). Springer Verlag. https://doi.org/10.1007/978-3-030-14812-6_42