Robust Portfolio Allocation under Expected, CVaR, and DRO Objectives (Sequence B)

Santiago de Jesus Villalobos-Gonzalez

June 27, 2025

Abstract

This paper studies the theoretical and numerical properties of robust portfolio allocation strategies evaluated under three regimes: Expected Utility maximization, Conditional Value-at-Risk (CVaR) minimization, and Distributionally Robust Optimization (DRO) formulations. We demonstrate how CVaR regularization introduces tail-risk control and illustrate the convergence between CVaR and DRO solutions. The numerical results show strong consistency with the theoretical risk metrics.

1 Introduction

In modern risk-sensitive portfolio theory, moving beyond expected value towards coherent risk measures such as CVaR allows for greater protection against downside risk. Furthermore, DRO frameworks aim to hedge against distributional ambiguity by optimizing worst-case tail losses within a Wasserstein ball or similar ambiguity sets.

2 Mathematical Formulation

Let $x \in \mathbb{R}^n$ denote portfolio weights, L the loss vector scenarios.

Expected Value Optimization

$$\min_{x} \quad \mathbb{E}[L^\top x] \quad \text{s.t.} \quad x \geq 0,\ \mathbf{1}^\top x = 1.$$

CVaR Minimization

$$\min_{x,\eta,\xi} \quad \eta + \frac{1}{(1-\alpha)N} \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \quad \xi_i \geq L_i^\top x - \eta,\ \xi_i \geq 0,\ x \geq 0,\ \mathbf{1}^\top x = 1.$$

DRO Formulation

$$\min_{x,\eta,\xi} \quad \eta + \frac{1}{(1-\alpha)N} \sum_{i=1}^{N} \xi_i + \epsilon \|x\|_2.$$

3 Numerical Results

- Expected x: [1.0, 0.0]
- CVaR x: [0.5562, 0.4438]
- DRO x: [0.5562, 0.4438]

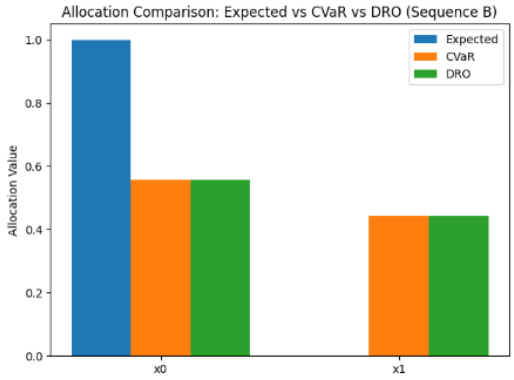

Figure 1: Allocation Comparison: Expected vs CVaR vs DRO (Sequence B)

4 Discussion

We observe a transition from full concentration in expected value to more diversified solutions under CVaR and DRO, reflecting aversion to extreme tail events.

5 Implementation Snippet

```
import cvxpy as cp
import numpy as np

x = cp.Variable(n)
eta = cp.Variable()
xi = cp.Variable(N)
```

```
objective = cp.Minimize(eta + (1 / ((1 - alpha) * N)) *
cp.sum(xi) + epsilon * cp.norm(x, 2))
constraints = [xi >= losses @ x - eta, xi >= 0, x >= 0, cp.sum(x) == 1]
problem = cp.Problem(objective, constraints)
problem.solve(solver=cp.GUROBI)
```