ROBUST PORTFOLIO OPTIMIZATION: SOLVER ALTERNATIVES FOR SHARPE RATIO MAXIMIZATION

OPENAI

Overview

This document outlines several alternatives and strategies for solving Sharpe ratio maximization in portfolio optimization when default solvers (like SCS) fail. It includes solver options, reformulations, and heuristic methods to ensure robustness in constrained optimization tasks.

I. Alternative Solvers in CVXPY

- **ECOS** Efficient conic solver, ideal for SOCP problems.
- **OSQP** Fast QP solver; suitable after reformulating Sharpe as a QP.
- **MOSEK** Commercial-grade solver for large-scale conic optimization.
- **GUROBI** High-performance QP solver; requires Sharpe reformulation.

II. Reformulating Sharpe Maximization

Several reformulations are useful when direct maximization of Sharpe fails:

(1) Maximize log Sharpe ratio:

$$\max_w \log(\mu^T w) - \frac{1}{2}\log(w^T \Sigma w)$$

(2) Fix risk, maximize return:

$$\max_w \mu^T w \quad \text{s.t.} \quad w^T \Sigma w \leq \sigma_{\max}^2$$

(3) Fix return, minimize risk:

$$\min_w w^T \Sigma w \quad \text{s.t.} \quad \mu^T w \geq R_{\min}$$

III. Heuristic and Metaheuristic Alternatives

These are effective for non-convex problems or when black-box optimization is needed:

- Genetic Algorithms (e.g., `DEAP`)
- Simulated Annealing / Particle Swarm Optimization
- Bayesian Optimization (e.g., `Optuna`, `Nevergrad`)

IV. Summary Table

Method	Type	Sharpe Support	Constraints	Robustness
ECOS	SOCP Solver	Yes	Yes	Good
OSQP	QP Solver	Indirect	Yes	Excellent
MOSEK	Conic Solver	Yes	Yes	Excellent
GUROBI	QP Solver	Indirect	Yes	Excellent
Fix Risk, Max Return	Reformulation	Indirect	Yes	Excellent
Fix Return, Min Risk	Reformulation	Indirect	Yes	Excellent
Genetic Algorithms	Heuristic	Direct	Yes	High
Bayesian Optimization	Heuristic	Direct	Yes	Medium

V. Python Code for Sharpe Maximization with Constraints

```
import numpy as np
import pandas as pd
import cvxpy as cp

def optimize_sharpe_with_constraints(
    returns: pd.DataFrame,
    drawdown: pd.Series,
    hit_rate: pd.Series,
    D_max: float,
    H_min: float,
    w_min: float = 0.0,
    w_max: float = 1.0
):
    N = returns.shape[1]
    mu = returns.mean().values
    Sigma = np.cov(returns.T)
    w = cp.Variable(N)
    port_return = mu @ w
    port_risk = cp.norm(cp.matmul(cp.sqrt(Sigma), w), 2)
    t = cp.Variable()
    constraints = [
        cp.sum(w) == 1,
        w >= w_min,
        w <= w_max,
        drawdown.values @ w <= D_max,
        hit_rate.values @ w >= H_min,
        port_return >= t * port_risk
    ]
    problem = cp.Problem(cp.Maximize(t), constraints)
    problem.solve(solver=cp.SCS)
    if problem.status not in ["optimal", "optimal_inaccurate"]:
        raise ValueError(f"Optimization failed: {problem.status}")
    return pd.Series(w.value, index=returns.columns, name="Weight")
```

VI. Example Usage

```
# Example usage
N = 10
T = 252
np.random.seed(0)
returns = pd.DataFrame(np.random.randn(T, N) * 0.01, columns=[f'Model_{i}' for i in range(N)])
drawdown = pd.Series(np.random.uniform(0.1, 0.4, N), index=returns.columns)
hit_rate = pd.Series(np.random.uniform(0.4, 0.7, N), index=returns.columns)
D_max = 0.25
H_min = 0.55

weights = optimize_sharpe_with_constraints(
    returns=returns,
    drawdown=drawdown,
    hit_rate=hit_rate,
    D_max=D_max,
    H_min=H_min
)
```

```
print(weights.round(4))
```