Portfolio Optimization with Multi-Objective Constraints

1. Problem Statement

Our goal is to optimize a daily portfolio of assets to achieve high risk-adjusted returns while respecting certain constraints on leverage, drawdown, and volatility. Specifically, our project focuses on maximizing a combination of financial risk metrics and ratios while limiting maximum drawdown over a given historical period.

Why This Matters

- Traditional mean-variance optimization oversimplifies risk by only using variance.
- Real-world portfolios must manage multiple risk dimensions (drawdown, volatility) and practical constraints (leverage, short selling) at the same time.
- Variability in the real-world and extreme events (like COVID) don't follow a nice and familiar distribution, so due to their unprediciatbility, traditional trading strategies have different tolerance levels to these unexpected events.
- By addressing these complexities, we aim to create a more realistic and robust decision-making strategy for creating a dynamic portfolio.

Success Metric

Fitness Score - What goes into the score

- Ratio: Sharpe Ratio = $\frac{E[R_p - R_f]}{\sigma_p}$

Where: R_p = Return of the portfolio | R_f = Risk-free rate | σ_p = Standard deviation of the portfolio's excess return (volatility)

- Risk Metrics - Maximum Drawdown reduction, controlled volatility
- Factor Exposure - this refers to the sensitivity of an investment, portfolio, or asset to specific risk factors or systematic drivers of returns
- Portfolio Constraints - we ask the model to minimize transaction costs, minimize the change in our stock positions between days, and try to maintain stable returns over time

Constraints

- **Leverage**: May exceed 1 but within a specified maximum (e.g., 1.5–2.0).
- **Drawdown**: Must remain below a specified percentage (e.g., 20% max drawdown).
- **Data**: Historical daily/weekly returns for selected assets (10–30).

Data Requirements

- Daily or weekly price data from a reliable source (CSV files or APIs like yfinance).
- Sufficient history (e.g., 3–5 years) to handle training and validation.

Potential Pitfalls

- Overfitting to historical data (backtest bias).
- Incorrect handling of missing data or survivorship bias.
- High computational costs if too many assets or constraints are added.

2. Technical Approach

Mathematical Formulation

Let w_i denote the weight of asset i in the portfolio. We define the portfolio return as

$$R_P = \sum_i w_i \cdot R_i$$

The portfolio volatility σ_p is computed as the annualized standard deviation of the daily portfolio returns, and the maximum drawdown of the portfolio, hereby abbreviated to $MDD(p)$, is computed from the cumulative returns. Given this, our objective function is

$$\textbf{Maximize} \quad \alpha \cdot \frac{E[R_P] - R_f}{\sigma_p} + \beta \cdot \big(-MDD(p)\big) - \lambda \sum_i \left|w_i - w_i^{\text{prev}}\right|$$

Where:

- R_f is the risk-free rate, so that $\frac{E[R_p - R_f]}{\sigma_p}$ represents the Sharpe ratio.
- α scales the impact of Sharpe ratio.
- β scales the impact of the drawdown

Algorithm & PyTorch Strategy

- Represent weights $\mathbf{w}$ as a PyTorch tensor.
- Compute portfolio returns and risk measures (volatility, drawdown) within the computational graph.
- Use gradient-based methods (e.g., Adam, LBFGS) to optimize $-\text{objective}$ (because PyTorch minimizes by default).

Validation Methods

- **In-Sample Optimization**: Train on a subset of historical data.
- **Out-of-Sample Backtest**: Test on later data (walk-forward or simple split).
- Compare results to a baseline (e.g., equal weights).

Resource Requirements

- Python 3.8+, PyTorch, NumPy, pandas, matplotlib, yfinance.
- Sufficient CPU/GPU time for iterative optimization and backtesting with large historical stock market prices.

3. Initial Results

Evidence of Working Implementation

- **Basic Test**: A small 7-asset dataset was loaded into our PyTorch pipeline.
  Companies: Tesla, Google, Microsoft, Amazon, Apple, Meta, NVIDIA
- **Online Gradient Descent**
  Our initial run only considered the stock market in 2023 and produced weights that resulted in a portfolio that was only outperformed by a portfolio that only contained META and only continaed NVDA. Our fitness score was only slightly better than putting equal weights on all seven stocks.

Performance Metrics (Preliminary)

- **Initial Sharpe**: 1.05 on a small sample dataset.
- **Drawdown**: ~25% peak-to-trough in the test sample.
- The result suggests some improvement over naive equal-weight (Sharpe ~ 0.95).

Test Case Results

- Verified the objective function calculates returns and volatility correctly.
- Observed that adding a drawdown penalty can shift weights toward lower-volatility assets.

Current Limitations

- Minimal data usage (only 6 months of daily returns).
- No transaction cost modeling, which may impact real-world applicability.

Resource Usage Measurements

- CPU-bound for small datasets; no GPU acceleration used yet.
- Optimization completes in ~1 second for 5 assets but could scale up with more assets.

Unexpected Challenges

- Handling negative weights for short selling in PyTorch required a custom clip function.
- Integrating maximum drawdown in the computational graph introduced complexity in gradient calculation.

4. Next Steps

1. **Expand Data Universe**
    - Increase asset count (10–20 stocks), ensuring robust coverage of different sectors.
    - Acquire a longer historical window (at least 3 years).
2. **Refine Constraints**
    - Enforce leverage up to 1.5, short selling up to 30% of portfolio.
    - Evaluate how these constraints interact with drawdown penalty.
    - Integrate more advanced risk measures like conditional value-at-risk.
3. **Rolling Optimization**
    - Implement a time-series approach to rebalance daily/monthly/quarterly.