Final Report: Improved Portfolio Optimization using Linear Shrinkage Covariance Estimator and Hierachical Risk Parity Algorithm

Author: Zhuoyi(Carl) Cui
Date: December 11, 2025

1. Project Overview and Background(Video 0:00 - 1:13)

1. Overview

In a typical quantitative pipeline, **stock selection** and **portfolio construction** are separate steps. Here we assume that alpha models have already selected a fixed stock universe, e.g., **100 promising stocks**.
The remaining question is: **how do we allocate capital across these 100 names?**

We treat portfolio construction as a mapping

$$f : (\mu, \Sigma) \to w,$$

where

- $\mu \in \mathbb{R}^N$: estimated expected returns
- $\Sigma \in \mathbb{R}^{N \times N}$: estimated return covariance matrix
- $w \in \mathbb{R}^N$: portfolio weights that fully invest the capital

This project studies how different estimates of Σ and the allocation rule $f(\cdot)$ affect portfolio risk and performance.

2. Markowitz Minimum-Variance Portfolio（Implemented in code）

In the Markowitz framework, a common baseline is the **minimum-variance portfolio**:

$$\begin{aligned} \min_{w \in \mathbb{R}^N} \quad & w^\top \Sigma w \\ \text{s.t.} \quad & \mathbf{1}^\top w = 1, \\ & w \geq 0\,, \end{aligned}$$

with closed-form solution (ignoring $w \geq 0$):

$$w^\star = \frac{\Sigma^{-1}\mathbf{1}}{\mathbf{1}^\top \Sigma^{-1}\mathbf{1}}.$$

It ignores expected returns and focuses on **risk reduction**, exploiting correlations in Σ to construct the least volatile fully-invested portfolio.

Key drawback: the method requires Σ^{-1}.
When Σ is noisy or nearly singular (common when N is large relative to T), the inverse becomes numerically unstable, leading to extreme and unstable weights out of sample.

3. Sample Covariance Estimator（Implemented in code）

Given N assets and T return observations $X_t \in \mathbb{R}^N$, the **sample covariance** is

$$S = \frac{1}{T-1} \sum_{t=1}^{T} (X_t - \bar{X})(X_t - \bar{X})^\top,$$

where $\bar{X}$ is the sample mean.

Under i.i.d. Gaussian returns, S is the standard unbiased / MLE estimator of the true covariance. However, in high dimensions (N large, T limited) the matrix S often becomes **ill-conditioned** or nearly singular. As a result:

- S^{-1} is numerically unstable,
- Markowitz optimization becomes highly sensitive to estimation noise,
- portfolio weights tend to be extreme and unstable out of sample.

These issues motivate more robust covariance estimators and alternative portfolio construction methods, which we explore in the rest of the project.

4. Linear Shrinkage Covariance Estimator（Implemented in code）

One remedy for the instability of S is to **shrink** it toward a well-conditioned target, typically a scaled identity matrix.
In the linear shrinkage approach, we form

$$\Sigma_{\text{shrink}} = \alpha S + (1-\alpha)\, mI,$$

where $m = \frac{1}{N}\,\mathrm{tr}(S)$ is the average variance, I is the identity matrix, and $0 \le \alpha \le 1$ is a data-driven shrinkage intensity (estimated following Ledoit–Wolf style formulas).

Intuition:

- When T is small relative to N, S is noisy and nearly singular; the identity target mI is very stable but ignores cross-asset structure.
- The shrinkage estimator interpolates between them, preserving some correlation information while improving conditioning.
- In portfolio optimization, this typically yields **more stable weights** and **lower realized risk** than using the raw sample covariance.

5. Hierarchical Risk Parity (HRP, Implemented in code)

Even with a better covariance estimator, classic Markowitz still relies on matrix inversion and tends to produce highly concentrated, unstable portfolios.
Hierarchical Risk Parity (HRP) takes a different approach:

1. Build a **distance matrix** from correlations and perform **hierarchical clustering** (e.g., single linkage) to group assets into a tree.
2. Apply **quasi-diagonalization** to reorder the covariance matrix so that similar assets sit next to each other (block-like structure).
3. Use **recursive bisection** on this ordering to allocate capital top-down, splitting risk across clusters without ever inverting Σ.

Key advantages:

- Avoids direct covariance matrix inversion, so it is more robust when N is large and T is limited.
- Tends to produce **better diversified** and **more stable** allocations, especially in correlated, clustered universes (like equities by sector or industry).
- In our experiments, HRP often achieves **lower or comparable volatility** than minimum-variance Markowitz built on the same covariance input.

2. Program Workflow and Features(Video 1:13 - 1:37)

2.1 Project Flow

At a high level, the project takes a **fixed stock universe with returns**, applies different **covariance estimators** and **portfolio construction rules**, and evaluates their out-of-sample performance in a rolling backtest.

The core work flow is:

1. **Data generation / loading**
   - A `MarketData` object (configured by `DataConfig`) provides simulated asset returns.
   - In the simulated case, assets are assigned to clusters, a block-structured correlation matrix and true covariance Σ_{true} are built, and arithmetic returns with random positive drifts are drawn and converted into price paths.
   - The backtest then uses `MarketData.get_returns()` as the core input.
2. **Backtest**
   - A `BacktestConfig` controls the look-back window, rebalance frequency, and initial Net Asset Value.
   - The `Backtest` class wires together the core portfolio engine:
     - Instantiates a `portfolioConstructor` (in `portfolio.py`), where most of the **algorithmically intensive logic** lives (covariance estimation, Min-Var optimization, and full HRP pipeline).

   `.
3. **Rolling-window portfolio construction & returns**
   - `Backtest.calculateReturns(retData)` treats `retData` as an (N, T) return panel.
   - At each rebalance date, it:
     - Takes a look-back window to estimate risk (`pastRetData`),
     - Calls `portfolioConstructor.fit(...)` to get portfolio weights for each strategy,
     - Applies these weights to the next `rebalance_freq` periods (`futureRetData`) to obtain portfolio returns and the cumulative return over that holding period.
4. **Performance evaluation**

- `Backtest.calculateEvalMetrics()` maps `self.retSeries` into standard performance statistics:
  - annualized return,
  - annualized volatility,
  - Sharpe ratio,
  - maximum drawdown.

5. **Visualization**
   - `Backtest.plotReturnCurves()` (or `plotEquityCurves()`) forms cumulative products of $(1 + r_t)$ to build net asset value (NAV) curves for each strategy and plots them together.

Other Features

- **Multiple covariance estimators（Involve much literature review!）**
  The framework can switch between sample covariance and shrinkage covariance, so different risk estimators can be compared in an identical environment.

2.2 Major Features Involved(based on literature review)

1. **Hierarchical Risk Parity (HRP)**
   - Full HRP pipeline is implemented: distance matrix → hierarchical clustering → quasi-diagonalization → recursive bisection.
   - This allows a direct comparison between Markowitz-style minimum variance and HRP under the same simulated universe.
2. **Shrinkage covariance estimator**
   - Implements a Ledoit–Wolf–style linear shrinkage towards a scaled identity to stabilize Σ when N is large relative to T.
3. **Monte Carlo robustness analysis** (`MC.py`)
   - `monte_carlo_annualized_volatility(...)` repeatedly simulates markets, runs the backtest, and records annualized volatility per strategy.
   - It then plots histograms of these volatilities for each strategy, marking the mean of `('sample', 'minVar')` optimized portfolio as a vertical line

3. Implementation Details(Video 1:37 - 3:22)

This section gives more detail on how each main feature is implemented and where the corresponding code lives. The ordering mirrors the workflow above.

3.1 MVP

1. **Data generation / `MarketData` (`data.py`)**
   - Core interface: `MarketData(DataConfig)`. On initialization, it either generates simulated data or loads real data.
   - In the simulated case, `_generate_simulated_data()`:
     - Assigns each of the N assets to one of `n_clusters` (roughly balanced).
     - Builds a block correlation matrix with high within-cluster and lower between-cluster correlations, then sets the diagonal to 1.
     - Draws heterogeneous volatilities and forms the true covariance $\Sigma_{\text{true}} = D\,\text{corr}\,D$, where $D = \text{diag}(\text{vols})$.
     - Converts annualized return bounds into daily means, samples arithmetic returns from a multivariate normal with mean `mu` and covariance `Sigma_true`, and clips extreme negatives (e.g. below –99%).
     - Constructs price paths via cumulative products of $(1 + r_t)$ from an initial price S_0.
   - Public methods:
     - `get_returns()` $\rightarrow (T, N)$ DataFrame of returns
     - `get_prices()` $\rightarrow (T, N)$ DataFrame of prices
     - `get_true_covariance()` $\rightarrow$ underlying `Sigma_true`
2. **Covariance estimators** (`estimator.py`)
   - **Base class:** `covEstimator`
     - Implements generic helper methods:
       - `computeFrobDis(covMatrix, trueCovMatrix)` $\rightarrow$ Frobenius distance between estimated and true covariance, normalized by $1/N$.
       - `computeEigenStats(covMatrix)` $\rightarrow$ max, min, and standard deviation of eigenvalues using `np.linalg.eigvalsh`.
   - **Sample covariance:** `class sampleCovEstimator(covEstimator)`
     - `computeCovMatrix(X)`:
       - Casts `X` to a float NumPy array and treats it as a (N, T) matrix (rows = assets, columns = time).
       - Centers each row and computes

$$S = \frac{1}{T-1} X_{\text{centered}} X_{\text{centered}}^{\top}.$$

   - **Shrinkage covariance:** `class shrinkageCovEstimator(covEstimator)`
     - `computeCovMatrix(X)`:
       - Starts from the same sample covariance S.
       - Computes the target scalar $m = \frac{1}{N}\text{tr}(S)$ and identity I.

- Estimates the shrinkage intensities r_1^2, r_2^2 via Frobenius norms of deviations across time.
- Constructs

$$\hat{\Sigma} = \alpha S + (1 - \alpha) m I$$

where α is the estimated shrinkage weight derived from d^2, r_1^2, r_2^2.

- This improves conditioning and mitigates over-fitting in high dimensions.

3. **Portfolio construction** (`portfolio.py`)
   - **Global minimum-variance weights:** `calcMinVarWeights`
     - Symmetrizes covariance: `cov = 0.5 * (cov + cov.T)` .
     - Solves the closed-form solution of the fully invested min-variance portfolio:

$$w^\star = \frac{\Sigma^{-1}\mathbf{1}}{\mathbf{1}^\top \Sigma^{-1}\mathbf{1}}.$$

     - Implemented explicitly using NumPy linear algebra (instead of generic CVX optimization), which is efficient and numerically robust.
   - **HRP pipeline:** `calcHrpWeights`
     a. **Tree clustering:** `treeClustering(covMatrix)`
        - Converts covariance to correlation and then to a distance matrix $d_{ij} = \sqrt{0.5(1 - \rho_{ij})}$.
        - Either:
          - Uses SciPy: `ssd.squareform` → `sch.linkage(..., method="single")` , or
          - Uses custom `singleLinkage(distMatrix)` when `useCustomSLinkage=True` .

     b. **Quasi-diagonalization:** `quasi_diagonalization(link)`
        - Traverses the dendrogram to generate an ordering of asset indices such that **similar names are adjacent**.
        - This linear ordering is used to permute the covariance matrix into a nearly block-diagonal form.

     c. **Recursive bisection:** `recurBisection(covMatrix, sortedIndex)`
        - Reorders the covariance matrix according to `sortedIndex` .
        - Recursively splits the ordered list of assets into left/right clusters.
        - For each half, computes an inverse-variance portfolio and uses **cluster variances** to allocate risk between halves.
        - The final vector of weights is then permuted back to the original asset ordering.
   - **Custom single linkage:** `singleLinkage(distMatrix)`
     - Implements hierarchical clustering from scratch using:

        - A `UnionFind` structure to maintain merging clusters.
        - A priority queue (`heapq`) storing pairwise distances between active clusters.
    - At each step:
        - Pops the smallest distance that still connects two active clusters.
        - Merges them, records a row in the linkage matrix `(i, j, dist, cluster_size)` .
        - Updates distances from the new cluster to all others using single-linkage definition: $d(A \cup B, C) = \min(d(A, C), d(B, C))$.
    - Returns a SciPy-compatible linkage matrix that can be passed to the existing quasi-diagonalization routine.

4. **Backtest & metrics** (`backtest.py`)
    - **Rolling-window backtest:**
      `Backtest.calculateReturns(retData)` runs a rolling-window backtest on an (N, T) return panel.
      At each rebalance date, it:
        - uses a fixed **look-back window** to estimate risk and construct portfolio weights,
        - applies these weights over the next **holding period** (rebalance frequency),
        - records one portfolio return per strategy (estimator–optimizer pair + equal-weight).
    - **Performance metrics:**
      `Backtest.calculateEvalMetrics()` summarizes each strategy's periodic returns into:
        - **annualized return**
        - **annualized volatility**
        - **Sharpe ratio**
        - **maximum drawdown**

The results are stored in a `DataFrame` with one row per strategy.

3.2 Additional Features(Video 3:22 - end, this includes the UI)

3. **Monte Carlo simulation & histograms** (`MC.py`)
   We run a Monte Carlo experiment using `monte_carlo_annualized_volatility(n_assets, lookBackWindow, trials, ...)` , where the **data-generating structure** (clustered covariance, correlation levels, volatility and return ranges) is fixed, and only the **random seed** (and optionally N and the look-back window) is varied across trials.
   For each trial, we generate a new simulated market, run the full `Backtest` pipeline, and record the **annualized volatility** for every strategy (e.g. `('sample','minVar')` , `('shrinkage','hrp')` , equal-weight, etc.). After aggregating over many trials, we plot **histograms of annualized**

volatility for each strategy and draw a vertical reference line at the mean volatility of the `('sample','minVar')` portfolio.

These histograms can be read as the **distribution of risk** each method tends to produce when N and the effective sample size T are of similar magnitude. In our experiments, the distributions for **shrinkage-based** and **HRP-based** portfolios are clearly shifted left (lower average volatility) relative to the **sample covariance + minimum-variance** strategy, indicating that they achieve **more stable and lower risk** under high-dimensional conditions.