CROSS-ASSET HIERARCHICAL RISK PARITY

MOISHE KESELMAN, SAPTARSHI KUMAR, SOUMIL BANERJEE, AND THOMAS HORSTKAMP

Introduction

One of the most central problems in finance is portfolio optmization, where investors aim to allocate capital among different assets to maximize returns and control for risk. Traditional methods like Markowitz's minimum-variance optimization (MV) and inverse-variance portfolio (IVP) try to achieve this goal but are often too simplistic. For example, MV is a susceptible process that can lead to unstable and concentrated portfolio weights, in part because MV requires inverting a covariance matrix. Such inversion does not work well when when there is too much or too little asset return data, or when there are too many or too few assets considered.

An alternative portfolio construction method introduced by López de Prado in 2016 [4], is known as Hierarchical Risk Parity (HRP). HRP addresses the limitations of traditional quadratic optimization techniques, such as MV, by using hierarchical clustering to group assets based on their correlations. Such clustering is natural and reflects the underlying asset relationships. Once it achieves a natural asset hierarchy, HRP allocates capital across the hierarchical structure and by using a simple risk parity condition. Applying HRP can lead to stable and diversified portfolios and superior out-of-sample risk-adjusted performance when compared to traditional optimization techniques.

In this paper, we extend the work of [4] by comparing HRP with three traditional portfolio optimization techniques: MV, Inverse-Variance Portfolio, and uniform weighting. We compare these strategies on historical returns from several asset classes: USD to foreign currency exchange rates, US equities (the component stocks of the S&P 500), commodities, and US index funds. We obtained US equity, commodity, and US index fund data from Yahoo Finance, using a Python API [1]. For foreign exchange rates, we obtained data from the Federal Reserve Economic Data (FRED) database, using their Python API [11]. We also examine the effects of using Exponentially Weighted Moving Average (EWMA) returns alongside HRP, which allows us to assign more weight to recent returns [2].

By testing HRP across diverse asset classes, we extend prior research that primarily focused on single asset classes [4]. We validate our empirical work through out-of-sample returns and weights analysis and comparisons of Sharpe ratios. Our results partially align with those in [4]. We find that HRP produces more stable portfolio weights than MV while maintaining competitive risk-adjusted returns. However, HRP does not consistently outperform all traditional portfolio optimization techniques across all asset classes: IVP generally provides the most stable weights, MV performs best in commodities, and uniform weighting performs best in highly diversified portfolios.

The remaining parts of the paper are structured in the following way: In the second section, we give more details about our data sources, how we preprocessed our data, and the machine learning methods we used (including a description of the HRP algorithm).

The third section presents our empirical results, including figures from our code. Finally, the fourth section discusses the implications of our results and directions for future research.

Methods

Data. To compare HRP with other portfolio optimization techniques, we tested the methods on several asset classes: USD to foreign currency exchange rates, US equities (the component stocks of the S&P 500), commodities, and US index funds. We obtained US equity, commodity, and US index fund data from Yahoo Finance, using a Python API [1]. For foreign exchange rates, we obtained data from the Federal Reserve Economic Data (FRED) database, using their Python API [11]. We did rudimentary data preprocessing, primarily getting rid of dates and assets that contained a lot of missing values. Lastly, we calculated the returns of each asset to feed into weight calculation methods. Both steps were performed using Pandas [7].

By testing HRP on multiple data sources, our paper goes above and beyond the testing that was performed in the original HRP paper[4], as well as follow up papers that performed small-scale tests on a single asset class [12, 5].

ML Methods. The three traditional portfolio optimization techniques we will compare HRP against are minimum variance optimization (MV), Inverse-Variance Portfolio (IVP), and uniform weighting. These methods are described in more detail in Appendix A. Here we present a brief description of the HRP algorithm from [4], adapted from the summary in [12]. The algorithm consists of three main stages: *Tree Clustering*, *Quasi-Diagonalization*, and *Recursive Bisection*.

We represent our input data as a $T \times N$ Pandas dataframe X, whose columns represent time series for N assets in the portfolio.

Stage 1: Tree Clustering.

1. Compute the $N \times N$ correlation matrix ρ with entries $\rho_{ij} = \rho(X_i, X_j)$.
2. Convert the correlation matrix ρ to a correlation-distance matrix d, where $d(X_i, X_j) = \sqrt{\frac{1}{2}(1 - \rho_{ij})}$.
3. Create another distance matrix $\tilde{d}$ by taking the Euclidean distance between the columns of d: $\tilde{d}[d_i, d_j] = \sqrt{\sum_{n=1}^{N}(d_{n,i} - d_{n,j})^2}$.
4. Recursively form clusters of assets using the linkage criterion described below:
   - The first cluster is defined as $u[1] := \underset{\substack{i \neq j \\ (i,j)}}{\operatorname{argmin}}\{\tilde{d}_{i,j}\}$.
   - Calculate the distances between the remaining items of $\tilde{d}$ and the newly formed cluster $u[1]$ via $\dot{d}_{i,u[1]} := \min\{\tilde{d}_{i,j} \mid j \in u[1]\}$. The result of this an $N \times 1$ vector $\dot{d}_{i,u[1]}$.
   - Update the matrix $\tilde{d}$ by appending $\dot{d}_{i,u[1]}$ and dropping the clustered rows and columns $j \in u[1]$:

$$\tilde{d} \longrightarrow \begin{pmatrix} \tilde{d} & \dot{d}_{i,u[1]} \\ (\dot{d}_{i,u[1]})^\top & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} \{\tilde{d}_{i,j}\}_{i,j \notin u[1]} & \dot{d}_{i,u[1]} \\ (\dot{d}_{i,u[1]})^\top & 0 \end{pmatrix}.$$

   - Repeat the previous three steps until a single cluster remains.

Stage 2: Quasi-Diagonalization.

This stage reorders the rows and columns of the covariance matrix of X so that similar investments appear together by recursively placing cluster constituents (from Stage 1) in order until no clusters remain. The result of this will be that the largest values of the covariance matrix lie along the diagonal. For a more detailed description of this step, see [4].

Stage 3: Recursive Bisection.

This is the final stage of the algorithm. It produces the actual weights assigned to the assets in our portfolio.

1. Initialize the list of items: $L = \{L_0\}$, where $L_0 = \{n_1, ..., n_N\}$.
2. Assign unit weight to all assets: $w_n = 1$ for $n = 1, ..., N$.
3. For each $L_i \in L$ with $|L_i| > 1$:
   - Bisect L_i into two subsets $L_i^{(1)} \cup L_i^{(2)} = L_i$.
   - Define the variance of $L_i^{(j)}, j = 1, 2$, via $\tilde{V}_i^{(j)} := \widetilde{w}_i^{(j)} V_i^{(j)} \widetilde{w}_i^{(j)}$, where $V_i^{(j)}$ is the covariance matrix of $L_i^{(j)}$, and $\widetilde{w}_i^{(j)} = \text{diag}\left[V_i^{(j)}\right]^{-1} \frac{1}{\text{tr}\left[\text{diag}\left[V_i^{(j)}\right]^{-1}\right]}$.
   - Compute the split factor $\alpha_i := 1 - \frac{\tilde{V}_i^{(1)}}{\tilde{V}_i^{(1)} + \tilde{V}_i^{(2)}}$.
   - Re-scale allocations w_n by a factor of α_i for all $n \in L_i^{(1)}$, and rescale by a factor of $1 - \alpha_i$ for all $n \in L_i^{(2)}$.
4. Repeat step 3 recursively until all clusters are allocated (i.e., $|L_i| = 1$ for all $L_i \in L$).

In vanilla HRP, we start with an initial covariance matrix Σ_0, which is computed over a specified time range with daily frequency. However, we can potentially improve upon this static covariance matrix using a covariance matrix of Exponentially Weighted Moving Average (EWMA) returns (see [2]), which allows us to assign more weight to more recent returns. The EWMA model adjusts dynamically by giving greater importance to recent data, allowing it to capture market fluctuations more effectively than a static covariance matrix.

Define r_t to be the vector of returns of the assets in our portfolio. Then, EWMA recursively computes

$$\text{EWMA}_{t+1} = \lambda r_t + (1 - \lambda)\text{EWMA}_t$$

where $\lambda = 0.94$ is a parameter for daily returns. This gives the following explicit equation for EWMA_{t+1} :

$$\text{EWMA}_{t+1} = (1 - \lambda)^{t+1}\text{EWMA}_0 + \lambda \sum_{k=0}^{t} (1 - \lambda)^k r_{t-k}.$$

Finally, we calculate the resulting covariance matrix using the EWMA returns. We compare the vanilla versions of the three non-uniform weighting algorithms with those involving covariance matrices computed from EWMA returns to see which performs better on our data.

It is also worth mentioning why tree-based clustering was chosen for Stage 1 of the HRP algorithm instead of another clustering technique such as k-means or a dimensionality reduction technique such as Principal Component Analysis (PCA). For example, k-means clustering partitions data into discrete groups, so it is unable to capture similarity between stocks in different groups [9]. Therefore, k-means may lead to abrupt allocation shifts. On the other hand, PCA is commonly used to identify latent factors that drive asset returns, but it assumes linear relationships between returns and does not directly identify dependencies among assets [10].

HRP improves upon both k-means clustering and PCA by recursively partitioning assets based on their correlations, ensuring that weights are allocated within groups of similar assets before diversifying across broader asset classes. It maintains the natural structure of

asset relationships, leading to more intuitive allocations.

Validation. We validated and compared the portfolio optimization methods by evaluating the variance of weights, overall returns, and the Sharpe ratios on out-of-sample (OOS) data. We performed cross-validation by calculating weights on a period 2 years of data and using the weights to create a portfolio for the subsequent month. Then for the following month, we recalculated the weights. This validation method allowed us to obtain 83 samples, which we consider sufficient to obtain conclusive results.

Our cross-validation method makes sense in the context of portfolio management because portfolio managers typically train models on recent data and use them in the short term before rebalancing. A typical rebalancing schedule is once a month, so our frequency for recalculating the weights makes sense.

Results

As we processed the asset data through HRP, we wanted to ensure that the algorithm was working as expected[1]. To confirm, we created a dendrogram of assets, found in Figure 1, and plotted the distance matrix in Figure 2. The dendrogram shows that HRP is able to hierarchically cluster similar assets together. As seen in Figure 1, the grain commodities are grouped together, as are the precious metals. The one asset that HRP seems to miss is Natural Gas, which doesn't seem to be similar to any other asset, in particular. Note that the distance matrix is roughly an inverse correlation matrix. Figure 2 demonstrates that the reordering process performs as expected, bringing similar assets to nearby indices of the matrix. With confidence that HRP was performing as expected, we moved on to testing it on out-of-sample data.

To understand HRP's performance, we ran it against industry-standard baseline models, namely uniform weighting, inverse-variance weighting (IVP), and minimum-variance optimization (MV). Processing the data to get 83-fold cross-validation took about two minutes for the stock data and less than a few seconds for other asset classes. Afterwards, we analyzed the variance of the portfolio weights, the return profiles, and the Sharpe ratios for each portfolio allocation strategy.

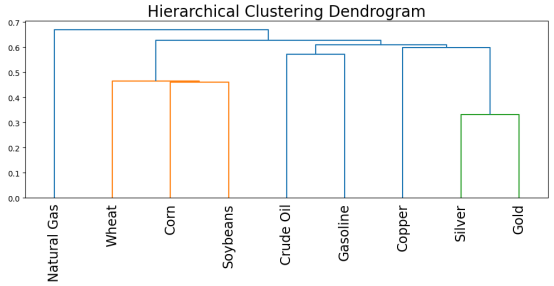

FIGURE 1. Dendrogram of commodities assets. Demonstrates similarity between similar commodities, such as gold and silver.

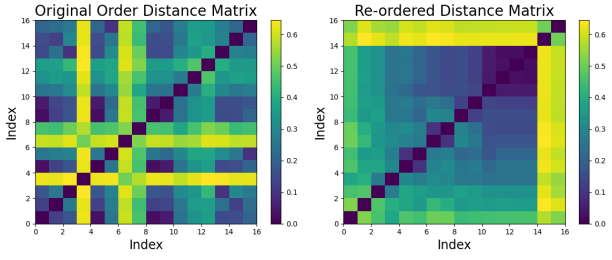

FIGURE 2. Distance matrix of USD to foreign exchange rates before and after hierarchical sorting.

[1]We used the Matplotlib Python package to generate all the figures found in this paper [3].

Overall, HRP performs at least as well as the baseline asset allocation strategies. In Figure 3, we can see that for the component stocks of the S&P 500 and foreign exchange rates, HRP dominates the other asset allocation strategies. In other cases, it is in the middle of the pack. Interestingly, MV performs significantly worse than the other strategies in all asset classes except commodities, where it performs the best. Another interesting observation is that the uniform weighting dominates in the portfolio of all assets. None of the other allocation strategies seems to focus on stocks, so their returns in the scenario are smaller. In other words, the non-uniform weighting strategies achieve significantly more returns in the S&P 500 or commodity cases as they do when employing all the assets.

Outside of pure returns, we also examine the approximate transaction costs incurred by the strategies. In Table 1, we can see the variance of the weights of the three non-uniform weighting strategies. IVP generally has the superior, lowest variance out of the three strategies, but HRP is not far behind in most cases. On the other hand, the minimum-variance portfolio has huge weight variances for all assets except FX, suggesting large transaction costs would need to be paid when implementing the strategy. Figure 4 visualizes these results in the form of a stacked bar chart, specifically for the commodities case. We can see that HRP and IVP have similar asset allocations, while the minimum-variance portfolio allocates more to gold at the expense of other commodities. In future work, we would like to create a more robust out-of-sample backtesting engine that would allow for more accurate transaction cost estimates.

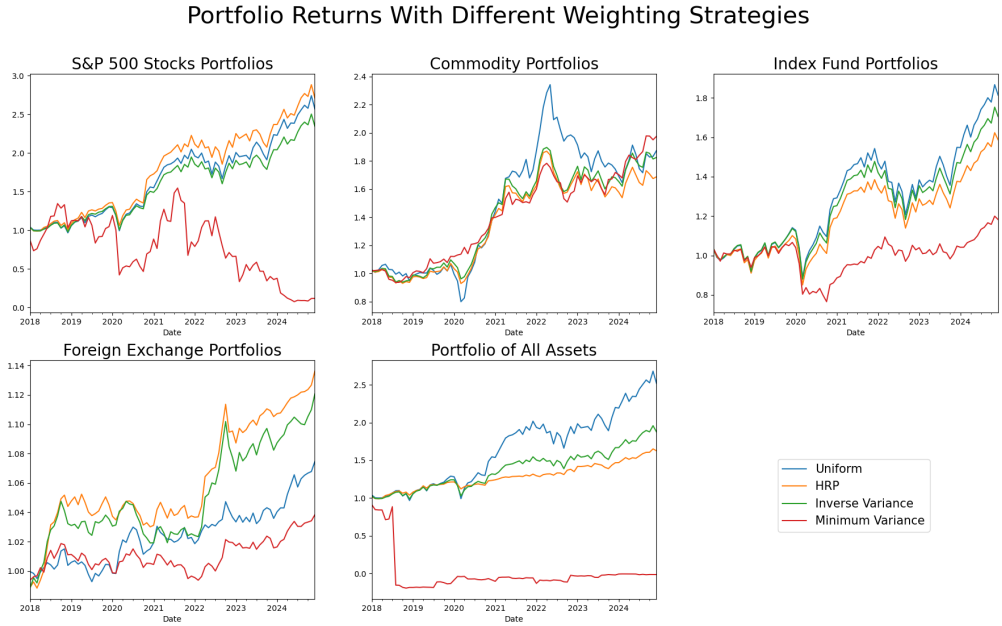

FIGURE 3. Portfolio returns using various weighting schemes on different asset categories. Note that the returns of the mean-variance portfolio drop to 0 in the portfolio of all assets because of matrix inversion instability caused by near linear dependence of stocks and index funds.

	Portfolio Weight Variance ($\times$ 1,000)				
	Stocks	Commodities	FX	Indices	All Assets
HRP Portfolio	0.026	**0.600**	5.320	0.444	0.036
Inverse-Variance Portfolio	**0.002**	0.908	1.107	**0.167**	**0.004**
Minimum-Variance Portfolio	29.729	3.476	**1.004**	810.221	979.782

TABLE 1. Table of variance of weights of portfolio allocation strategies. Smallest values are best (bolded) because they suggest fewer transaction costs needed to implement the portfolio.

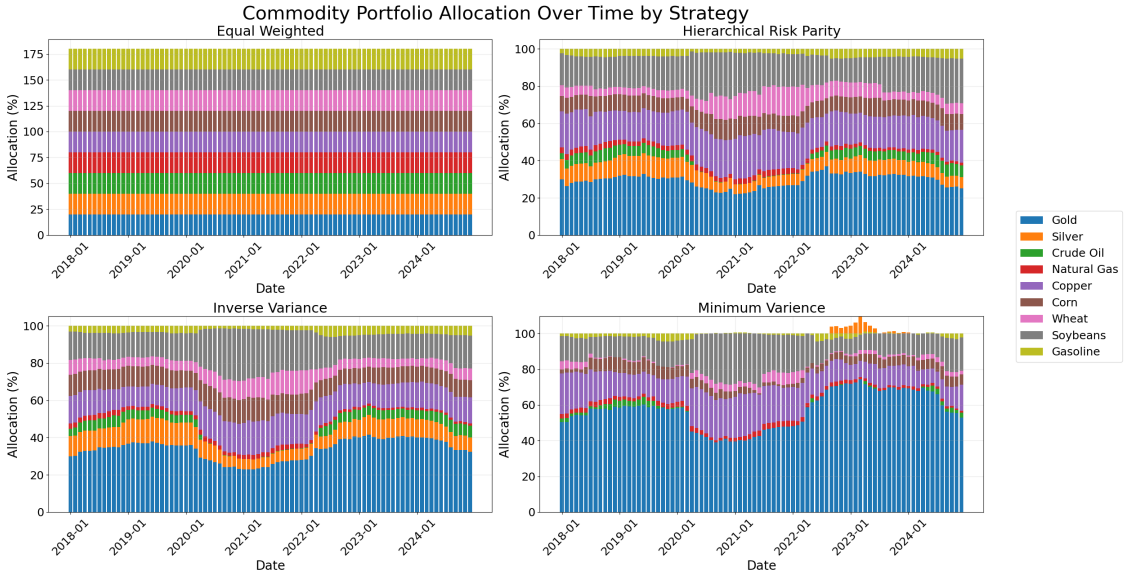

FIGURE 4. Distribution of commodity portfolio weights over time for the four asset allocation algorithms. We notice a spike above 100% weighting for the minimum-variance portfolio, due to a negative weight for a different asset.

In addition to transaction costs, the variance of the weights also quantifies the certainty and robustness of the portfolio optimization models in choosing weights. Likely due to matrix inverse instability, minimum-variance portfolio has higher weight variance.

While there are many metrics used to assess portfolio efficacy, we chose to focus on the Sharpe ratio in our analysis. Table 2 presents the Sharpe ratios for the four asset allocation strategies in each asset class. Additionally, for the strategies that utilize a covariance matrix to assign portfolio weights, we also try using a covariance matrix computed with Exponentially Weighted Moving Average (EWMA) returns. In several cases, the portfolios are unable to beat the assumed risk-free rate of 3%, so they achieve a negative Sharpe ratio.

In general, EWMA has a marginal, if any, effect on OOS returns. For example, it marginally improves the HRP portfolio in the stock, indices, Similar small effects are seen with the other two strategies. In the future, we would try tuning the exponential weighting parameter using cross validation.

In terms of performance, HRP performed the best in stocks and foreign exchange, uniform weighting performed the best in index funds and the all assets case, and minimum-variance performed the best in commodities. Even in the stock index and commodities cases, where HRP underperformed, it was still a close follower of the other portfolios. It was mainly in the all assets case where HRP lagged behind the winning strategy significantly. Without transaction costs, a Sharpe ratio of 2.455 in the S&P 500 stock case is a very solid performance, so we would consider it usable for investments.

Strategy	Model	Stocks	Commodities	Currencies	Indices	All Assets
Uniform Weighting	Regular	2.293	1.497	-3.991	**1.473**	**2.292**
HRP Portfolio	Regular	2.45	1.439	**-1.889**	1.117	2.161
	EWMA	**2.455**	1.431	-2.002	1.126	2.244
Inverse-Variance Portfolio	Regular	2.16	1.759	-2.093	1.318	1.943
	EWMA	2.154	1.76	-2.093	1.322	1.948
Minimum-Variance Portfolio	Regular	-0.32	2.437	-6.197	0.053	-0.001
	EWMA	-0.313	**2.445**	-6.219	0.037	-0.671

TABLE 2. Table showing Sharpe ratios for the portfolio allocation strategies across the asset classes, assuming a 3% risk-free rate. The returns for all portfolios in in the foreign exchange case were below 2%, hence leading to a negative Sharpe ratio. As seen in the Figure 3, there is inconsistency about which strategy performs the best.

CONCLUSION

In this study, we compared Hierarchical Risk Parity (HRP) with three traditional portfolio optimization techniques — minimum-variance optimization (MV), Inverse-Variance Portfolio (IVP), and uniform weighting — across multiple asset classes. Our results highlight HRP's ability to deliver competitive risk-adjusted returns while maintaining stable and diversified portfolio allocations.

HRP outperformed in S&P 500 stocks and foreign exchange portfolios, offering superior returns and Sharpe ratios. It also demonstrated much lower weight variance than MV in all asset classes except foreign exchange, suggesting lower transaction costs while maintaining more stable allocations. Interestingly, MV still had the highest returns and Sharpe ratio in commodities while HRP had the lowest. In broad multi-asset portfolios, the uniform weighting strategy performed surprisingly well, suggesting that in highly diversified settings, simpler allocation methods may be sufficient.

We also examined the effect of Exponentially Weighted Moving Average (EWMA) returns and found that its impact was marginal. While it slightly improved HRP's performance in stocks, it had little to no effect in most cases and even slightly reduced HRP's Sharpe ratio in commodities and currencies. These findings suggest that EWMA is not consistently beneficial without further tuning.

Many of our findings align with López de Prado [4], particularly HRP's ability to produce more stable portfolio weights than minimum-variance optimization by avoiding

the instability of inverting the covariance matrix. However, our results provide a more nuanced view. While HRP generally reduced weight variance compared to MV, IVP had the lowest variance across most asset classes, except in commodities and foreign exchange, where HRP and MV performed better, respectively. These results suggest that while HRP remains a robust choice, IVP may sometimes provide greater stability, leading to lower transaction costs.

Despite HRP's advantages, our study has limitations. We did not explicitly model real-world transaction costs, relying on weight variance as a proxy. Future work could incorporate trading simulations to better estimate turnover and slippage costs. Additionally, our results indicate that static EWMA estimation may be insufficient; a more adaptive approach, potentially using machine learning to dynamically adjust covariance estimation, could improve performance.

Overall, our findings confirm that HRP is an effective portfolio optimization technique, balancing risk control, stable weight allocation, and strong out-of-sample performance. However, its effectiveness depends on the asset class and market conditions. Future improvements in cost modeling, covariance estimation, and adaptive strategy selection could further enhance HRP's real-world applicability.[2]

[2]Enhanced this conclusion using GPT-4o.

References

[1] Ran Aroussi. *yfinance*. 2017. URL: https://github.com/ranaroussi/yfinance?tab=readme-ov-file.

[2] NYU Stern School of Business VLab. *EWMA Covariance*. URL: https://vlab.stern.nyu.edu/docs/correlation/EWMA-COV.

[3] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[4] Marcos López de Prado and Marcos López de Prado. "Building Diversified Portfolios that Outperform Out-of-Sample". In: *Journal of Portfolio Management* (2016). DOI: 10.3905/jpm.2016.42.4.059. URL: http://dx.doi.org/10.2139/ssrn.2708678.

[5] Gautier Marti. *Hierarchical Risk Parity - Implementation & Experiments (Part I)*. 2018. URL: https://gmarti.gitlab.io/qfin/2018/10/02/hierarchical-risk-parity-part-1.html (visited on 02/21/2025).

[6] Robert Andrew Martin. "PyPortfolioOpt: portfolio optimization in Python". In: *Journal of Open Source Software* 6.61 (2021), p. 3066. DOI: 10.21105/joss.03066. URL: https://doi.org/10.21105/joss.03066.

[7] Wes McKinney et al. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[8] Marat Molyboga. *A Modified Hierarchical Risk Parity Framework for Portfolio Management*. 2020. URL: https://www.pm-research.com/content/iijjfds/2/3/128.

[9] Chris Piech. *K Means*. 2013. URL: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html.

[10] Alex Reinhart. *Lecture 3: Matrix factorization & mixture models*. 2025. URL: https://canvas.cmu.edu/courses/45508/files/folder/Lecture%20notes/Lecture%203%20-%20matrix%2C%20mixtures?preview=12337115.

[11] Federal Reserve Bank of St. Louis. *U.S. Dollars to Euro Spot Exchange Rate [DEXUSEU]*. 2025. URL: https://fred.stlouisfed.org/series/DEXUSEU.

[12] Aditya Vyas. *The Hierarchical Risk Parity Algorithm: An Introduction*. 2020. URL: https://hudsonthames.org/an-introduction-to-the-hierarchical-risk-parity-algorithm/ (visited on 02/21/2025).

Appendix A. Traditional Portfolio Optimization Techniques

In this appendix, we provide mathematical descriptions of the three traditional portfolio optimization techniques that we compared against Hierarchical Risk Parity (HRP): minimum-variance optimization (MV), Inverse-Variance Portfolio (IVP), and uniform weighting.

A.1. **Minimum-Variance Optimization (MV).** Minimum-variance optimization seeks to construct a portfolio that minimizes risk, as measured by portfolio variance, without considering expected returns. Mathematically, it solves the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}^\top \Sigma \mathbf{w} \\ \text{s.t.} \quad & \mathbf{w}^\top \mathbf{1} = 1 \end{aligned}$$

where:

- $\mathbf{w} \in \mathbb{R}^N$ is the vector of portfolio weights,
- $\Sigma \in \mathbb{R}^{N \times N}$ is the estimated covariance matrix of asset returns,
- $\mathbf{1} \in \mathbb{R}^N$ is a vector of ones, ensuring full investment (i.e., weights sum to one).

This optimization problem has solution

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

and ensures the portfolio has the lowest possible variance given the available assets.

A.2. **Inverse-Variance Portfolio (IVP).** Inverse-Variance Portfolio is a heuristic approach that assigns portfolio weights inversely proportional to asset variances. It does not require estimating the full covariance matrix, making it a more robust alternative when dealing with noisy or unstable covariance estimates. The portfolio weights are given by

$$w_i = \frac{1/\sigma_i^2}{\sum_{j=1}^N 1/\sigma_j^2}, \quad i = 1, \ldots, N$$

where σ_i^2 is the variance of asset i. This method ensures that assets with lower individual risk receive higher allocations, thereby emphasizing diversification without explicitly using correlations.

A.3. **Uniform Weighting.** Uniform weighting is the simplest portfolio allocation strategy, where capital is distributed equally among all assets. The portfolio weights are

$$w_i = \frac{1}{N}, \quad i = 1, \ldots, N$$

where N is the total number of assets in the portfolio. Equal weighting is easy to implement, does not rely on parameter estimation, and often serves as a strong benchmark due to its built-in diversification.

Appendix B. Relevant Code

[1]:
```
import pandas as pd
pd.options.display.float_format = '{:.6g}'.format
import scipy.cluster.hierarchy
import scipy.spatial.distance
import numpy as np
import matplotlib.pyplot as plt
import collections
import random
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta
import time
```

[2]:
```
class HRP:
    """
    Calculates the weights of the portfolio for all methodologies.
    """
    def __init__(self, prior_data=None, file_path=None, start_date=None,␣
↪end_date=None):
        if prior_data is not None:
            self.returns =  self._generate_returns(data=prior_data,␣
↪start_date=start_date, end_date=end_date)
        else:
            self.returns =  self._generate_returns(file_path=file_path,␣
↪start_date=start_date, end_date=end_date)

        self.corr,self.cov = self.returns.corr(), self.returns.cov()
        self.clusters = None
        self.weights = None
        self.d = None

    def _generate_returns(self, data=None, file_path=None, start_date=None,␣
↪end_date=None):
        if data is not None:
            stock_df = data
        else:
```

```
            stock_df = pd.read_csv(file_path)

        stock_df['Date'] = pd.to_datetime(stock_df['Date'])

        stock_datedf = stock_df
        if start_date and end_date:
            stock_datedf = stock_df[(stock_df['Date'] >= start_date) &␣
↪(stock_df['Date'] <= end_date)]

        stock_datedf.set_index('Date', inplace=True)
        returns_df = stock_datedf.pct_change().dropna(how="all").dropna(axis=1,␣
↪how="all") ## Drop rows and column with all NA

        return returns_df.fillna(0) ## Fill Na values with 0 returns

    def _get_clusters(self, returns,␣
↪corr_metric='simple',linkage_method='single'):
        if corr_metric == 'simple':
            corr = returns.corr()
        elif corr_metric == 'exponential':
            ewma_returns = returns.ewm(alpha= 0.94 ,adjust=False).mean()
            corr = ewma_returns.corr()
        elif corr_metric == 'noise':
            noise = np.random.normal(loc=0, scale=0.01, size=returns.shape)
            noisy_returns = returns + noise
            corr = noisy_returns.corr()

        d = np.sqrt(((1.-corr)/2.))
        d_tilde = scipy.spatial.distance.squareform(d,checks=False)
        clusters = scipy.cluster.hierarchy.linkage(d_tilde,linkage_method)

        return clusters,d,d_tilde

    def _get_quasi_diagonal_matrix(self):
        return scipy.cluster.hierarchy.to_tree(self.clusters, rd=False).
↪pre_order()

    def plot_corr_matrix(self):

        sort_idx = self._get_quasi_diagonal_matrix()
        N = len(self.d)
        seriated_dist = np.zeros((N, N))
        a,b = np.triu_indices(N, k=1)
        seriated_dist[a,b] = self.d.values[[sort_idx[i] for i in a],␣
↪[sort_idx[j] for j in b]]
        seriated_dist[b,a] = seriated_dist[a,b]
```

```
        fig, ax = plt.subplots(1, 2, figsize=(12, 5))

        c1 = ax[0].pcolormesh(self.d, cmap='viridis', shading='auto')
        fig.colorbar(c1, ax=ax[0])
        ax[0].set_title('Original Order Distance Matrix', fontsize=20)
        ax[0].set_xlabel('Index', fontsize=17)
        ax[0].set_ylabel('Index', fontsize=17)

        c2 = ax[1].pcolormesh(seriated_dist, cmap='viridis', shading='auto')
        fig.colorbar(c2, ax=ax[1])
        ax[1].set_title('Re-ordered Distance Matrix', fontsize=20)
        ax[1].set_xlabel('Index', fontsize=17)
        ax[1].set_ylabel('Index', fontsize=17)

        plt.tight_layout()
        plt.show()

    def plot_weights_pie(self, hrp_w, n=10):

        hrp_series = pd.Series(hrp_w).sort_values(ascending=False)

        top_n = hrp_series.head(n)
        # top_n.plot.pie(figsize=(10, 10), autopct='%1.1f%%', cmap=f'tab{n}',␣
↪legend=True)
        top_n.plot.pie(figsize=(10, 10), autopct='%1.1f%%', cmap=f'prism',␣
↪legend=True)
        plt.legend().remove()

        plt.title(f'Top {n} HRP Portfolio Weights')
        plt.ylabel('')  # Hide y-axis label
        plt.show()

    def plot_dendogram(self,n = None ):
        fig, ax = plt.subplots(figsize=(10, 5))
        if n is None:
            scipy.cluster.hierarchy.dendrogram(self.clusters, \
                                        labels=list(self.returns.columns) , \
                                            ax=ax, orientation="top")
        else:
            rets = self.returns.loc[:,random.sample(list(self.returns.columns),␣
↪n)]
            clusters_ = self._get_clusters(rets)[0]
            scipy.cluster.hierarchy.dendrogram(clusters_, \
                                        labels= list(rets.columns) , \
                                            ax=ax, orientation="top")

        ax.tick_params(axis="x", rotation=90, labelsize=17)
```

```
        plt.tight_layout()
        ax.set_title("Hierarchical Clustering Dendrogram", fontsize=20)
        plt.show()


    def _computer_cluster_variance( self, cluster ):
        cov_cluster = self.cov.loc[cluster,cluster]
        weights = 1 / np.diag(cov_cluster)
        weights /= weights.sum()
        return np.linalg.multi_dot((weights, cov_cluster, weights))

    def recursive_bisection( self, sorted_index):

        ordered_tickers =  self.cov.index[sorted_index].tolist()
        w = pd.Series(1., index=ordered_tickers)
        clusters = [ordered_tickers]

        while len(clusters) > 0:
            clusters = [item[j:k] for item in clusters for j,k in ((0, len(item)␣
↪// 2), (len(item) // 2, len(item)))
                        if len(item) > 1 ]

            for k in range(0,len(clusters),2):
                cluster_k = clusters[k]
                cluster_l = clusters[k+1]

                #Defininining variance
                var_cluster_k = self._computer_cluster_variance(cluster_k)
                var_cluster_l = self._computer_cluster_variance(cluster_l)

                #Defining weights
                alpha = 1 - var_cluster_k / (var_cluster_k + var_cluster_l)
                w[cluster_k] *= alpha
                w[cluster_l] *= 1 - alpha

        w[np.abs(w) < 1e-4] = 0.
        w = np.round(w, 5)
        return collections.OrderedDict(w.sort_index())


    def calculate_hrp_weights(self, corr_metric='simple',␣
↪linkage_method='single'):

        ## Step1
        self.clusters, self.d,_ = self._get_clusters(self.
↪returns,corr_metric,linkage_method)
```

```
        ##Step2
        sorted_idx = self._get_quasi_diagonal_matrix()

        ##Step3
        w = self.recursive_bisection( sorted_idx )
        return w

    def calculate_MV_weights(self, corr_metric='simple'):
        if corr_metric == 'simple':
            inv_covar = np.linalg.inv(self.cov)
        elif corr_metric == 'exponential':
            ewma_returns = self.returns.ewm(alpha= 0.94 ,adjust=False).mean()
            inv_covar = np.linalg.inv(ewma_returns.cov())

        u = np.ones(len(self.cov))

        w = np.dot(inv_covar, u) / np.dot(u, np.dot(inv_covar, u))
        w[np.abs(w) < 1e-4] = 0.
        w = np.round(w, 5)
        return collections.OrderedDict(zip(list(self.returns.columns) , w))


    def calculate_RP_weights(self, corr_metric='simple'):
        if corr_metric == 'simple':
            weights = (1 / np.diag(self.cov))
        elif corr_metric == 'exponential':
            ewma_returns = self.returns.ewm(alpha= 0.94 ,adjust=False).mean()
            weights = (1 / np.diag(ewma_returns.cov()))

        w =  weights / sum(weights)
        w[np.abs(w) < 1e-4] = 0.
        w = np.round(w, 5)
        return collections.OrderedDict(zip(list(self.returns.columns) , w))
```

1 OOS PERFORMANCE

[3]:
```
def get_oos_ret_freq(data,start_date,end_date,freq="M"):
    """
    Get Out of Sample data in returns format for each frequency period.
    """
    filter_data = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]
    filter_data.set_index('Date', inplace=True)
    returns_df = filter_data.pct_change().dropna(how="all").dropna(axis=1,␣
↪how="all").fillna(0)
    adj_ret = returns_df + 1
```

```
    adj_ret_freq = adj_ret.groupby(adj_ret.index.to_period(freq)).cumprod()
    ret_freq = adj_ret_freq.resample(freq).last().fillna(0)
    ret_freq[['EQ_Ret','HRP_Ret','RP_Ret','MV_Ret']] = 0

    return ret_freq

def get_rolling_returns(main_data, data_oos, lookback_days,␣
↪weight_type='simple'):
    """
    Get Rolling returns with training only given lookback_days.
    Also, Get variance of weights
    """
    start_timer = time.time()

    #WEIGHTS dataframes
    HRP_WEIGHT = data_oos.copy()
    RP_WEIGHT = data_oos.copy()
    MV_WEIGHT = data_oos.copy()

    num_assets = data_oos.shape[1] - 4
    eq_weights = 1/num_assets
    data_oos['EQ_Ret'] = np.sum(data_oos.iloc[:,:num_assets] * eq_weights,␣
↪axis=1)

    for i, date in enumerate(data_oos.index):
        end_date_train = (date.replace(day=1) - timedelta(days=1))
        start_date_train = end_date_train - relativedelta(days=lookback_days)

        end_date_train = end_date_train.strftime("%Y-%m-%d")
        start_date_train = start_date_train.strftime("%Y-%m-%d")

        hrp_oos = HRP(prior_data=main_data, start_date=start_date_train,␣
↪end_date=end_date_train)

        hrp_weights = pd.Series(hrp_oos.
↪calculate_hrp_weights(corr_metric=weight_type))
        rp_weights = pd.Series(hrp_oos.
↪calculate_RP_weights(corr_metric=weight_type))
        mv_weights = pd.Series(hrp_oos.
↪calculate_MV_weights(corr_metric=weight_type))

        HRP_WEIGHT.loc[date] = hrp_weights
        RP_WEIGHT.loc[date] = rp_weights
        MV_WEIGHT.loc[date] = mv_weights
```

```
        data_oos.loc[date,'HRP_Ret'] = np.sum(data_oos.iloc[i,:num_assets] *␣
↪hrp_weights)
        data_oos.loc[date,'RP_Ret'] = np.sum(data_oos.iloc[i,:num_assets] *␣
↪rp_weights)
        data_oos.loc[date,'MV_Ret'] = np.sum(data_oos.iloc[i,:num_assets] *␣
↪mv_weights)

    HRP_WEIGHT.dropna(inplace=True, axis=1)
    RP_WEIGHT.dropna(inplace=True, axis=1)
    MV_WEIGHT.dropna(inplace=True, axis=1)

    HRP_WEIGHT_VAR = np.mean(HRP_WEIGHT.T.var())
    RP_WEIGHT_VAR = np.mean(RP_WEIGHT.T.var())
    MV_WEIGHT_VAR = np.mean(MV_WEIGHT.T.var())

    weight_var_df = pd.DataFrame({
        'HRP': HRP_WEIGHT_VAR,
        'RP': RP_WEIGHT_VAR,
        'MV': MV_WEIGHT_VAR
    },index=["Variance"]).T


    end_timer = time.time()
    print(f"Time Taken for training: {end_timer-start_timer:.4f} sec")

    return data_oos[['EQ_Ret','HRP_Ret','RP_Ret','MV_Ret']], weight_var_df,␣
↪{'hrp': HRP_WEIGHT, 'rp': RP_WEIGHT, 'mv': MV_WEIGHT}

def get_ret_stats(roll_df):
    """
    Get Mean and Variance of stats, IN PERCENTAGE
    """
    stats_df = pd.DataFrame({
        'Mean': (roll_df.mean() - 1) * 12,
        'Standard Deviation': roll_df.std()
    }).T
    return stats_df
```

1.1 EQUITIES

[4]:
```
stock_data = pd.read_csv('data/sp500v2.csv')
stock_data['Date'] = pd.to_datetime(stock_data['Date'])

start_stock_oos = '2018-01-01'
end_stock_oos= '2024-12-31'
```

```
oos_stock_data = get_oos_ret_freq(stock_data, start_stock_oos, end_stock_oos,"M")
rolling_returns_stock, weight_var_stock, weights_dict_stock =␣
 ↪get_rolling_returns(stock_data, oos_stock_data, 2*365, weight_type='simple')
equity_stats = get_ret_stats(rolling_returns_stock)
```

```
Time Taken for training: 74.5047 sec
```

[5]:
```
# exponentially weighted returns
rolling_returns_stock_exp, weight_var_stock_exp, weights_dict_stock_exp =␣
 ↪get_rolling_returns(stock_data, oos_stock_data, 2*365,␣
 ↪weight_type='exponential')
equity_stats_exp = get_ret_stats(rolling_returns_stock_exp)
```

```
Time Taken for training: 73.7877 sec
```

[21]:
```
equity_stats
```

[21]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.152599	0.159712	0.137138	-0.033719
Standard Deviation	0.0534557	0.0529468	0.0496063	0.199314

[22]:
```
weight_var_stock
```

[22]:

	Variance
HRP	1.7542e-06
RP	9.7391e-07
MV	0.0296774

[23]:
```
equity_stats_exp
```

[23]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.152599	0.159395	0.136799	-0.0323741
Standard Deviation	0.0534557	0.052715	0.0495885	0.199421

[24]:
```
weight_var_stock_exp
```

[24]:

	Variance
HRP	1.76159e-06
RP	9.7887e-07
MV	0.029769

1.2 COMMODITIES

[6]:
```
comm_data = pd.read_csv('data/commodities_data.csv')
comm_data['Date'] = pd.to_datetime(comm_data['Date'])

start_comm_oos = '2018-01-01'
end_comm_oos= '2024-12-31'
```

```
oos_comm_data = get_oos_ret_freq(comm_data,start_comm_oos,end_comm_oos,"M")
rolling_returns_comm, weight_var_comm, weights_dict_comm =␣
 ↪get_rolling_returns(comm_data, oos_comm_data, 2*365)
comm_stats = get_ret_stats(rolling_returns_comm)
```

```
Time Taken for training: 1.4211 sec
```

[7]:
```
# exponentially weighted returns
rolling_returns_comm_exp, weight_var_comm_exp, weights_dict_comm_exp =␣
 ↪get_rolling_returns(comm_data, oos_comm_data, 2*365, weight_type='exponential')
comm_stats_exp = get_ret_stats(rolling_returns_comm_exp)
```

```
Time Taken for training: 1.3624 sec
```

[25]:
```
comm_stats
```

[25]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.104574	0.0828134	0.0939712	0.102818
Standard Deviation	0.0498016	0.0367042	0.0363755	0.0298857

[26]:
```
weight_var_comm
```

[26]:

	Variance
HRP	0.00879453
RP	0.0106302
MV	0.0985554

[27]:
```
comm_stats_exp
```

[27]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.104574	0.0824061	0.0939781	0.1031
Standard Deviation	0.0498016	0.0366242	0.0363592	0.0298984

[29]:
```
weight_var_comm_exp
```

[29]:

	Variance
HRP	0.00875373
RP	0.0106608
MV	0.0988793

1.3 INDEXES

[8]:
```
ind_data = pd.read_csv('data/daily_index_data_last_5_years.csv')
ind_data['Date'] = pd.to_datetime(ind_data['Date'])

start_ind_oos = '2018-01-01'
end_ind_oos= '2024-12-31'
```

```
oos_ind_data = get_oos_ret_freq(ind_data,start_ind_oos,end_ind_oos,"M")
rolling_returns_ind, weight_var_ind, weights_dict_ind =␣
 ↪get_rolling_returns(ind_data,oos_ind_data, 2*365)
ind_stats = get_ret_stats(rolling_returns_ind)
```

```
Time Taken for training: 2.0014 sec
```

[9]:
```
# exponentially weighted returns
rolling_returns_ind_exp, weight_var_ind_exp, weights_dict_ind_exp =␣
 ↪get_rolling_returns(ind_data,oos_ind_data, 2*365, weight_type='exponential')
ind_stats_exp = get_ret_stats(rolling_returns_ind_exp)
```

```
Time Taken for training: 2.0364 sec
```

[30]:
```
ind_stats
```

[30]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.0981801	0.0766183	0.0881391	0.0319379
Standard Deviation	0.0462969	0.0417255	0.044118	0.0362482

[31]:
```
weight_var_ind
```

[31]:

	Variance
HRP	0.00202508
RP	0.000585605
MV	1.22966

[32]:
```
ind_stats_exp
```

[32]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.0981801	0.0769885	0.0883461	0.0313522
Standard Deviation	0.0462969	0.0417316	0.0441415	0.0362087

[33]:
```
weight_var_ind_exp
```

[33]:

	Variance
HRP	0.00202834
RP	0.000574817
MV	1.23434

1.4 CURRENCIES

[10]:
```
curr = pd.read_csv('data/usd_exchange_rates.csv')
curr['Date'] = pd.to_datetime(curr['Date'])
curr_data = curr.copy()
curr_data.iloc[:,1:] = curr_data.iloc[:,1:].replace(".",np.nan).apply(pd.
 ↪to_numeric)
```

```
start_curr_oos = '2018-01-01'
end_curr_oos= '2024-12-31'

oos_curr_data = get_oos_ret_freq(curr_data,start_curr_oos,end_curr_oos,"M")
rolling_returns_curr, weight_var_curr, weights_dict_curr =␣
 ↪get_rolling_returns(curr_data,oos_curr_data, 2*365)
curr_stats = get_ret_stats(rolling_returns_curr)
```

```
Time Taken for training: 1.8657 sec
```

[11]:
```
# exponentially weighted returns
rolling_returns_curr_exp, weight_var_curr_exp, weights_dict_curr_exp =␣
 ↪get_rolling_returns(curr_data,oos_curr_data, 2*365, weight_type='exponential')
curr_stats_exp = get_ret_stats(rolling_returns_curr_exp)
```

```
Time Taken for training: 1.9645 sec
```

[34]:
```
curr_stats
```

[34]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.0104233	0.0184743	0.0165296	0.00546294
Standard Deviation	0.00490579	0.00609935	0.00644265	0.00395909

[35]:
```
weight_var_curr
```

[35]:

	Variance
HRP	0.00987378
RP	0.0052498
MV	0.00587578

[36]:
```
curr_stats_exp
```

[36]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.0104233	0.0173047	0.0165016	0.00546311
Standard Deviation	0.00490579	0.00606932	0.00645544	0.00395024

[38]:
```
weight_var_curr_exp
```

[38]:

	Variance
HRP	0.0099172
RP	0.00528323
MV	0.00585635

1.5 ALL DATA

[ ]:
```
all_data = curr_data.merge(comm_data).merge(stock_data)#.merge(stock_data)

start_all_oos = '2018-01-01'
end_all_oos= '2024-12-31'

oos_all_data = get_oos_ret_freq(all_data, start_all_oos, end_all_oos,"M")
rolling_returns_all, weight_var_all, weights_dict_all =␣
 ↪get_rolling_returns(all_data, oos_all_data, 2*365)
all_stats = get_ret_stats(rolling_returns_all)
```

```
Time Taken for training: 74.5602 sec
```

[13]:
```
# exponentially weighted returns
rolling_returns_all_exp, weight_var_all_exp, weights_dict_all_exp =␣
 ↪get_rolling_returns(all_data, oos_all_data, 2*365, weight_type='exponential')
all_stats_exp = get_ret_stats(rolling_returns_all_exp)
```

```
Time Taken for training: 78.6244 sec
```

[39]:
```
all_stats
```

[39]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.148745	0.0717874	0.0974862	0.0295601
Standard Deviation	0.0518005	0.0193409	0.0347331	0.328231

[40]:
```
weight_var_all
```

[40]:

	Variance
HRP	0.000113738
RP	2.15009e-05
MV	0.203572

[41]:
```
all_stats_exp
```

[41]:

	EQ_Ret	HRP_Ret	RP_Ret	MV_Ret
Mean	0.148745	0.0720464	0.0979724	-0.937218
Standard Deviation	0.0518005	0.0194477	0.0347626	0.571847

[42]:
```
weight_var_all_exp
```

[42]:

	Variance
HRP	0.000111932
RP	2.13925e-05
MV	0.386485