`pyhrp`: Hierarchical Risk Parity and Schur Complementary Allocation on an Explicit Cluster Tree

Thomas Schmelzer*

September 15, 2026

Abstract

`pyhrp` is a small Python package that implements Hierarchical Risk Parity (HRP) and its Schur-complement extension by allocating weights on an *explicit*, inspectable binary tree rather than by returning a bare weight vector. This paper documents the construction the package actually performs — the correlation-to-distance map, the linkage and bisection trees, the recursive inverse-variance split, and the Schur augmentation of the two child blocks — and reports what that construction produces on the twenty-asset panel shipped with the test suite. Of three claims the package documents about its Schur allocator, one holds exactly and two do not. The one that holds is worth relying on: at $\gamma = 0$ the Schur allocator reproduces HRP bit for bit, so it is a safe drop-in. The two that fail are stated without qualification in the package's own documentation — that $\gamma = 1$ recovers the global minimum-variance portfolio, and that variance falls monotonically in γ — and we give a two-asset counterexample to the first and a 76/200 failure rate for the second. The gap is in the split rule, which stays inverse-variance at every γ; we make its consequences precise, since a user choosing γ is choosing between two behaviours that the documentation currently conflates.

1 Introduction

Mean-variance optimisation (Markowitz, 1952) inverts a covariance matrix. That inversion is the source of both its appeal and its notorious instability: the smallest eigenvalues of an estimated covariance matrix are the least well determined, and inversion weights them most heavily, so small changes in the input produce large changes in the output (Michaud, 1989). The standard responses are to improve the estimate — shrinkage (Ledoit and Wolf, 2004) — or to abandon the optimisation — equal weights (DeMiguel et al., 2009).

Hierarchical Risk Parity (López de Prado, 2016) takes a third route. It never inverts anything. Instead it uses the correlation matrix twice, and only in ways that are robust to estimation noise: once to build a hierarchy over the assets by agglomerative clustering, and once to compute sub-portfolio variances, which are quadratic forms rather than solves. Weights fall out of a recursive top-down split of a unit budget between the two children of every node, inversely proportional to each child's variance. The result is long-only by construction, sums to one without a constraint being imposed, and requires no expected returns.

The idea that a correlation matrix over financial assets carries hierarchical structure predates HRP (Mantegna, 1999); HRP's contribution is to make that structure the allocation mechanism rather than a visualisation.

*`https://github.com/tschm/pyhrp`

1.1 What this package adds

`pyhrp` makes the tree a first-class object. `hrp()` returns the root `Cluster` of the weighted tree, not a weight series, so the intermediate allocation at every node remains available for inspection: which cluster received what fraction of the budget, and how that fraction was subdivided. The weight vector is one attribute of the root, recovered as `root.portfolio.weights`.

Three allocators share that tree and the same input contract:

- `risk_parity` — HRP as published: split by the plain block variance of each child.
- `schur_risk_parity` — split by a Schur-augmented block variance (Cotton, 2024), interpolating with a parameter $\gamma \in [0, 1]$.
- `one_over_n` — a hierarchical equal-weight allocation, exposed as a generator that yields one complete portfolio per tree level, so the allocation can be watched as the hierarchy deepens.

Section 2 states the construction, Section 3 describes the implementation choices that are not implied by the mathematics, and Section 4 reports numerical results, including the two documentation claims that do not hold.

2 Method

Let $R \in \mathbb{R}^{T \times N}$ be a matrix of simple returns for N assets over T periods, with sample covariance $\Sigma \in \mathbb{R}^{N \times N}$ and sample correlation $C \in \mathbb{R}^{N \times N}$.

2.1 From correlation to a tree

Correlations are turned into distances by the standard map

$$d_{ij} = \sqrt{\frac{1 - C_{ij}}{2}}, \tag{1}$$

clipped to $[0, 1]$ before the square root and with an exactly zero diagonal. Perfectly correlated assets sit at distance 0, uncorrelated assets at $1/\sqrt{2}$, perfectly anticorrelated assets at 1.

The condensed form of d is passed to SciPy's agglomerative clustering (Virtanen et al., 2020) under one of four linkage methods — `single`, `complete`, `average` or `ward` (Ward, 1963), the default. The resulting linkage matrix is converted into a binary tree of `Cluster` nodes, each leaf carrying the column index of one asset.

The correlation matrix is validated before use. Fewer than two assets, or any non-finite entry, raises. A non-finite diagonal entry is reported with the offending asset named, because the cause is almost always a constant price series, whose zero variance makes its correlations undefined.

The bisection variant. Setting `bisection=True` keeps only the *leaf order* that the chosen linkage induces and discards the tree shape. The ordered list of leaves is then split in half, recursively, until singletons remain. This is closer to the construction described in the original HRP paper, and it produces a balanced tree by definition: depth is $\lceil \log_2 N \rceil$ regardless of how chain-like the linkage was.

A bisection tree has no linkage matrix of its own, so one is rebuilt for plotting. The merge height assigned to an internal node is its *cluster diameter*,

$$h(v) = \max_{i,j \in \text{leaves}(v)} d_{ij}, \tag{2}$$

which is monotone along the tree — a parent's leaf set contains each child's, so its maximum can only grow — and monotonicity is exactly what SciPy requires of that column.

2.2 The recursive split

Every node holds a `Portfolio`: a mapping from asset name to weight. A leaf holds its own asset at weight 1. An internal node combines its children.

For an internal node v with children ℓ and r, let w_ℓ and w_r denote the child portfolios — each already a unit-sum allocation over its own leaf set — and let Σ_ℓ, Σ_r be the corresponding diagonal blocks of Σ. Define the child variances

$$v_\ell = w_\ell^\top \Sigma_\ell w_\ell, \qquad v_r = w_r^\top \Sigma_r w_r. \tag{3}$$

The budget of v is split

$$\alpha_\ell = \frac{v_r}{v_\ell + v_r}, \qquad \alpha_r = 1 - \alpha_\ell = \frac{v_\ell}{v_\ell + v_r}, \tag{4}$$

so that $\alpha_\ell v_\ell = \alpha_r v_r$: the two children contribute equally to the parent, up to the cross term, which this rule ignores. If both variances vanish — riskless sub-portfolios — the split is $\alpha_\ell = \alpha_r = 1/2$ rather than undefined. The node's portfolio is

$$w_v = \alpha_\ell \, w_\ell \oplus \alpha_r \, w_r, \tag{5}$$

where $\oplus$ concatenates over disjoint asset sets. Since $\alpha_\ell + \alpha_r = 1$ and both are non-negative, the invariant that a node's weights are non-negative and sum to one propagates from the leaves to the root.

Algorithm 1. Allocation on a cluster tree

Input: root ρ, covariance Σ, per-node variance rule ν.

1: $\mathcal{O} \leftarrow$ nodes of ρ in pre-order, collected with an explicit stack so that every parent precedes its children.

2: For each $v \in \text{reverse}(\mathcal{O})$:

- if v is a leaf, set $w_v \leftarrow \{\, \text{asset}(v) : 1 \,\}$;
- otherwise compute $(v_\ell, v_r) \leftarrow \nu(\ell, r, \Sigma)$ — Eq. (3) for HRP, Eq. (8) for Schur — and set $w_v \leftarrow \alpha_\ell w_\ell \oplus \alpha_r w_r$ by Eq. (4).

3: Return ρ.

Algorithm 1 is the whole allocator. Reversing a pre-order walk guarantees that both children carry a portfolio before their parent needs one, which is what makes a single bottom-up pass sufficient.

2.3 Schur complementary allocation

Equation (3) uses only the diagonal blocks of Σ. The cross-block covariance between the two children is discarded — at every node, at every level. Schur Complementary Allocation (Cotton, 2024) restores part of it.

Write the covariance restricted to the union of the two children's assets in block form,

$$\Sigma_v = \begin{pmatrix} A & B \\ B^\top & D \end{pmatrix}, \tag{6}$$

with A the left block and D the right. The augmented blocks are

$$A_\gamma = A - \gamma \, B D^{-1} B^\top, \qquad D_\gamma = D - \gamma \, B^\top A^{-1} B, \tag{7}$$

and the split then uses

$$v_\ell = w_\ell^\top A_\gamma \, w_\ell, \qquad v_r = w_r^\top D_\gamma \, w_r. \tag{8}$$

At $\gamma = 0$ this is Eq. (3) unchanged. At $\gamma = 1$ the augmented blocks are the Schur complements of D and A in Σ_v, each group's covariance *conditioned* on the other. Intermediate γ interpolates linearly in the correction term.

Both augmentations require a solve against a covariance block. Collinear assets make those blocks singular, so the implementation falls back from `np.linalg.solve` to the minimum-norm least-squares solution when the solve raises, which keeps Eq. (7) defined rather than propagating an exception out of a recursion.

Note what is *not* changed: the split rule. Equation (4) remains inverse-variance for every γ. Section 4.3 shows why that matters.

2.4 Hierarchical $1/N$

The third allocator ignores Σ. At level n of the tree it distributes a budget 2^{-n} equally among the leaves of each node at that level, accumulates into a running portfolio, and yields a copy. A leaf that terminates early keeps the weight it was given, so each yielded portfolio is a complete allocation over all assets and the sequence shows how the equal-weight allocation refines as the hierarchy is descended.

3 Implementation

The package is about 1,200 lines of Python over eight modules, dependent only on NumPy (Harris et al., 2020), SciPy (Virtanen et al., 2020), Polars (Vink and Polars contributors, 2025) and Plotly. Covariance and correlation frames are Polars DataFrames whose columns name the assets; the allocators translate names to indices once and then work on NumPy arrays.

Iterative traversal, not recursion. Both the SciPy-tree conversion and the allocation walk are written as explicit two-pass stack loops. This is not stylistic. Single linkage chains: on noisy data its tree depth grows with N rather than with $\log N$, and the recursive forms of these two functions raised `RecursionError` on universes that the mathematics handles without complaint. The iterative form removes the asset count as a limit.

The allocator contract. `risk_parity` and `schur_risk_parity` write *into* the tree they are given and return the same root. Every node's portfolio is replaced rather than accumulated into, which makes them idempotent: re-running on an already weighted tree — with a different covariance matrix, or a different γ — gives the same answer as running on a fresh one. The cost is that the caller's tree is modified. `Dendrogram` is a frozen dataclass, but the `Cluster` it holds is not, so passing `dendrogram.root` to an allocator rewrites the portfolios inside that dendrogram; a caller who needs the unweighted tree must deep-copy it. `one_over_n` differs on both counts: it accumulates into a local buffer, leaving the input untouched, and yields rather than returns.

Numerical conventions. Returns are simple returns from prices, with leading all-null rows dropped and remaining nulls and NaNs filled with zero. Covariance and correlation are the plain sample estimators (`np.cov`, `np.corrcoef`). No shrinkage is applied anywhere: the package's robustness claim rests on avoiding the inversion, not on regularising the estimate.

Table 1: HRP allocations on the twenty-asset panel, by linkage method and tree construction. *Bisect* rebuilds the tree by halving the linkage leaf order.

Linkage	Bisect	Variance	Ann. vol.	N_{eff}	$\max_i w_i$
`single`	no	6.388×10^{-5}	0.1269	7.82	0.2245
`single`	yes	5.325×10^{-5}	0.1158	13.63	0.1308
`complete`	no	5.255×10^{-5}	0.1151	11.82	0.1591
`complete`	yes	5.131×10^{-5}	0.1137	13.00	0.1288
`average`	no	6.047×10^{-5}	0.1234	10.21	0.2042
`average`	yes	5.702×10^{-5}	0.1199	15.39	0.1003
`ward`	no	5.478×10^{-5}	0.1175	13.27	0.1377
`ward`	yes	5.397×10^{-5}	0.1166	14.05	0.1238

Table 2: Schur complementary allocation on the same panel, `ward` linkage, as γ varies. The last row is the global minimum-variance portfolio, which allows short positions, for reference.

	Variance	Ann. vol.	N_{eff}	$\max_i w_i$
$\gamma = 0.00$	5.478×10^{-5}	0.1175	13.27	0.1377
$\gamma = 0.25$	5.473×10^{-5}	0.1174	13.24	0.1381
$\gamma = 0.50$	5.467×10^{-5}	0.1174	13.20	0.1385
$\gamma = 0.75$	5.459×10^{-5}	0.1173	13.15	0.1390
$\gamma = 1.00$	5.449×10^{-5}	0.1172	13.08	0.1397
GMV (long/short)	4.137×10^{-5}	0.1021	4.37	0.2438

4 Numerical results

All numbers below come from the twenty-asset daily price panel shipped in the test suite as `tests/resources/stock_prices.csv` (320 observations, 20 assets). Variances are of daily returns; annualised volatility multiplies by $\sqrt{252}$. We report the effective number of positions $N_{\text{eff}} = 1/\sum_i w_i^2$ as a concentration measure — it equals N for equal weights and 1 for a single position.

4.1 Tree construction

Table 1 shows the choice of linkage mattering roughly as much as one would expect from the geometry. `single` linkage is the outlier: its chaining produces the most concentrated allocation of the eight ($N_{\text{eff}} = 7.82$ against 13.27 for `ward`) and the highest variance. This is the mechanism, not an accident — an unbalanced tree splits a budget unevenly by construction, because a deep leaf has been halved more often than a shallow one.

Bisection improves every method on both measures here, and improves `single` most (variance -17%, N_{eff} from 7.82 to 13.63), which is consistent with balance being what it supplies. We would not read the uniform improvement as general: this is one panel, and the ordering of two allocators on one covariance estimate is not evidence about their ordering out of sample.

4.2 The Schur parameter

Table 2 is the central empirical result of this paper, and it is mostly a null one. Moving γ from 0 to 1 — from no cross-block information to the full Schur complement — reduces the portfolio variance by 0.53% and the annualised volatility by 3 basis points. The allocation barely moves: the largest weight changes by 0.002, and N_{eff} falls from 13.27 to 13.08.

The reference row explains why that is worth stating. The global minimum-variance portfolio on this covariance matrix has variance 4.137×10^{-5}, a quarter below the $\gamma = 1$ allocation, and it is a visibly different object: $N_{\text{eff}} = 4.37$, and it holds short positions (its smallest weight is -0.132). Whatever γ interpolates towards in this implementation, on this panel it is not that portfolio.

4.3 Three documented claims, examined

The package documents three properties of the Schur allocator. We tested each.

$\gamma = 0$ **reproduces HRP exactly. Confirmed.** The two allocators agree to the last bit on this panel: the maximum absolute weight difference over the twenty assets is exactly 0, not merely below a tolerance. This is structural rather than fortunate — at $\gamma = 0$ Eq. (7) leaves A and D untouched, so the two code paths compute the same floating-point expression — and it makes `schur_hrp` a safe drop-in for `hrp`.

$\gamma = 1$ **recovers the minimum-variance portfolio. Not as stated.** Table 2 already shows the two portfolios differing substantially on the twenty-asset panel. The reason is visible in a case small enough to check by hand. Take

$$\Sigma = \begin{pmatrix} 4 & 1.5 \\ 1.5 & 1 \end{pmatrix}, \tag{9}$$

and the two-leaf tree. The global minimum-variance portfolio is $w^\star = (-0.25,\ 1.25)$; the Schur allocator at $\gamma = 1$ returns $(0.2,\ 0.8)$.

The mechanism is Eq. (4). Augmenting the blocks changes *which* variances are compared, but the comparison is still an inverse-variance split, and an inverse-variance split of two sub-portfolios is not the minimum-variance combination of them — the minimum-variance weight for two correlated blocks depends on their covariance, and no reweighting of the diagonal blocks alone can reproduce it. A further consequence: because both α's in Eq. (4) are non-negative, the output is long-only for every γ, while the minimum-variance portfolio generally is not. The two cannot coincide whenever $w^\star$ has a negative entry, whatever γ does.

We are describing a gap between this implementation and the sentence in its own documentation, not disputing Cotton (2024); recovering minimum variance requires the split itself to become a minimum-variance solve, which Eq. (4) never becomes.

Variance falls as γ rises. True here, not in general. It holds on the shipped panel, monotonically, as Table 2 shows. It is not a property of the algorithm. Over 200 synthetic eight-asset panels (60 observations each, independent Gaussian log-returns; see Appendix A), the variance failed to be non-increasing in γ in 76 cases — 38%. In the first such case the variance *rose* monotonically, from 1.0367×10^{-5} at $\gamma = 0$ to 1.0401×10^{-5} at $\gamma = 1$, an increase of 0.33%.

The effect is small in both directions, which is the practical point: γ is a weak knob on this construction. A user who reads "raising γ trades robustness for lower in-sample variance" should not expect the trade to be reliably available.

5 Limitations

The covariance estimator is the plain sample one. On a universe whose asset count approaches its observation count this estimator is badly conditioned, and while HRP never inverts it, the block variances of Eq. (3) are still computed from it and the Schur augmentation of Eq. (7)

does solve against it. Shrinkage (Ledoit and Wolf, 2004) is left to the caller, who may pass any covariance frame.

The allocation is unconditionally long-only and fully invested. There is no provision for weight caps, turnover limits, transaction costs, leverage, or a risk-free asset, and no rebalancing logic: the package maps one covariance estimate to one allocation. Nothing here evaluates HRP as a strategy. All variances reported are in-sample, computed against the same covariance matrix that produced the weights, and are properties of the construction rather than evidence about performance — the out-of-sample question that motivates HRP (López de Prado, 2016) is outside what this package measures.

6 Conclusion

Allocating on an explicit tree costs little and buys inspectability: the intermediate budget of every cluster survives in the returned object, which makes an HRP allocation auditable in a way a weight vector is not. The construction is otherwise faithful to the published algorithm, and the two variants the package adds behave as follows on the panel it ships. Bisection is a real improvement in balance, and most valuable exactly where the linkage tree is worst. The Schur parameter γ is a much weaker instrument than its documentation suggests: it moves the allocation by fractions of a percent, its variance reduction is not monotone across panels, and its $\gamma = 1$ endpoint is not the minimum-variance portfolio, because the split rule it feeds stays inverse-variance throughout. Users should treat γ as a small perturbation of HRP rather than as a dial between HRP and minimum variance, and the documentation should say so.

A Reproducibility

Every number in Section 4 was produced with `pyhrp` 2.3.3 and the panel in the repository. Tables 1 and 2:

```
import numpy as np, polars as pl
from pyhrp import hrp, schur_hrp, compute_returns, compute_cov

prices = pl.read_csv("tests/resources/stock_prices.csv",
                     try_parse_dates=True).drop("date")
cov = compute_cov(compute_returns(prices))

for method in ["single", "complete", "average", "ward"]:
    for bisection in [False, True]:
        root = hrp(prices=prices, method=method, bisection=bisection)
        w = np.array(list(root.portfolio.weights.values()))
        print(method, bisection, root.portfolio.variance(cov),
              1 / np.sum(w ** 2), w.max())

for gamma in [0.0, 0.25, 0.5, 0.75, 1.0]:
    root = schur_hrp(prices=prices, method="ward", gamma=gamma)
    print(gamma, root.portfolio.variance(cov))
```

The minimum-variance reference row is $w^\star \propto \Sigma^{-1}\mathbf{1}$, normalised to sum to one. The monotonicity experiment of Section 4.3:

```
rng = np.random.default_rng(0)
violations = 0
```

```
for _ in range(200):
    X = rng.standard_normal((60, 8))
    p = pl.DataFrame({f"A{i}": 100 * np.exp(np.cumsum(X[:, i] * 0.01))
                      for i in range(8)})
    c = compute_cov(compute_returns(p))
    v = [schur_hrp(prices=p, method="ward", gamma=g).portfolio.variance(c)
         for g in (0.0, 0.5, 1.0)]
    violations += not (v[0] >= v[1] >= v[2] - 1e-18)
print(violations, "/ 200")
```

References

Peter Cotton. Schur complementary allocation: A unification of hierarchical risk parity and minimum variance portfolios. *arXiv preprint arXiv:2411.05807*, 2024. URL `https://arxiv.org/abs/2411.05807`.

Victor DeMiguel, Lorenzo Garlappi, and Raman Uppal. Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The Review of Financial Studies*, 22(5):1915–1953, 2009.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. Array programming with numpy. *Nature*, 585:357–362, 2020.

Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411, 2004.

Marcos López de Prado. Building diversified portfolios that outperform out of sample. *The Journal of Portfolio Management*, 42(4):59–69, 2016. doi: 10.3905/jpm.2016.42.4.059.

Rosario N. Mantegna. Hierarchical structure in financial markets. *The European Physical Journal B*, 11(1):193–197, 1999.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

Richard O. Michaud. The markowitz optimization enigma: Is 'optimized' optimal? *Financial Analysts Journal*, 45(1):31–42, 1989.

Ritchie Vink and Polars contributors. Polars: Dataframes for the new era. `https://pola.rs`, 2025.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, et al. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.