

Dynamically Regularized MVP

Documentation

Mads Hebsgaard
*Copenhagen Business School,
Department of Finance, Center for Statistics*

February 5, 2026

Abstract

This is my second submission. My first submission did not remove stocks that were not in the test sample. That is, the minimum variance portfolios included assets that were not to be tested. This is obviously suboptimal, especially since shorting is allowed. This submission removes stocks that are not to be tested, by requiring `'chars['ctff_test'] == True'` for the weight month. Also, stocks need to be traded at least once within the last month to be included. Otherwise the code and the logic is completely unchanged from my previous submission. My previous submission's SR (through the standard deviation) very likely suffered greatly from this mistake.

Contents

1	Rationale for the Minimum Variance Objective	2
2	Robust Covariance Estimation	2
2.1	Local History Weighting	2
2.2	Global Dimensionality Adjustment	3
3	Construction and Constraints	3
4	Calibration and Robustness	4
A	Code Appendix	6
A.1	Project Repository	6
A.2	Python Source Files (.py)	6
A.2.1	main.py	6

1 Rationale for the Minimum Variance Objective

Standard Mean-Variance Optimization seeks to maximize the ex-ante Sharpe ratio via the Tangency Portfolio, $\mathbf{w}_{TP} \propto \Sigma^{-1}(\boldsymbol{\mu} - r_f \mathbf{1})$. However, Chopra and Ziemba (1993) demonstrate that estimation errors in expected returns ($\boldsymbol{\mu}$) degrade portfolio utility an order of magnitude more than errors in covariances. Given the low signal-to-noise ratio of daily returns, estimating distinct means quickly leads to poor out of sample performance. To mitigate this, we adopt a robust prior where all assets have identical expected returns ($\boldsymbol{\mu} = c\mathbf{1}$). Under this assumption, the Tangency solution simplifies:

$$\mathbf{w}_{TP} \propto \Sigma^{-1}(c\mathbf{1}) \propto \Sigma^{-1}\mathbf{1} = \mathbf{w}_{GMV} \quad (1)$$

Thus, absent reliable directional forecasts, the objective of maximizing the Sharpe ratio mathematically collapses to minimizing portfolio variance. We therefore focus exclusively on the robust estimation of $\hat{\Sigma}$ to maximize risk-adjusted returns by minimizing the denominator rather than speculating on the numerator.

2 Robust Covariance Estimation

In high-dimensional settings, the sample covariance matrix (S) is often ill-conditioned and prone to significant estimation error. Standard shrinkage methods, such as those proposed by Ledoit and Wolf (2004), mitigate this by pulling the sample covariance toward a structured, robust target (T), such as the identity matrix scaled by average variance. The classical linear shrinkage estimator is defined as:

$$\hat{\Sigma} = \delta T + (1 - \delta)S \quad (2)$$

where $\delta \in [0, 1]$ is a scalar shrinkage intensity. A primary limitation of this conventional approach is that δ is uniform; it shrinks every element of the covariance matrix with the identical intensity, disregarding the heterogeneous estimation uncertainty of specific covariance pairs or the time-varying dimensionality of the investment universe.

2.1 Local History Weighting

To address the limitations of uniform shrinkage in the presence of varying data availability (e.g., recent IPOs or irregular trading), we implement an observation-weighted shrinkage framework. Instead of a fixed scalar, we apply element-wise shrinkage based on the quantity of overlapping data points, scaled by a dynamic global confidence parameter.

We introduce an observation weight matrix Ω_{obs} , which penalizes covariance pairs with insufficient history. Let N_{ij} be the count of overlapping non-zero observations between assets i and j . We define a minimum history requirement N_{min} and a lower bound threshold N_{low} . The pairwise history weight ω_{ij} is calculated via a linear ramp function:

$$\omega_{ij} = \begin{cases} 0 & \text{if } N_{ij} \leq N_{low} \\ \frac{N_{ij} - N_{low}}{N_{min}} & \text{if } N_{low} < N_{ij} < N_{low} + N_{min} \\ 1 & \text{if } N_{ij} \geq N_{low} + N_{min} \end{cases} \quad (3)$$

This function ensures that when data is sparse ($N_{ij} \leq N_{low}$), the weight on the sample covariance is zero, forcing the estimator to default fully to the target prior.

2.2 Global Dimensionality Adjustment

The maximum weight applied to the sample covariance is not static. It is determined dynamically by the ratio of temporal observations (n) to the number of assets (p).

We define the dynamic sample confidence λ_{conf} as:

$$\lambda_{conf} = \min \left(\sqrt{\frac{n}{p}}, \lambda_{max} \right) \quad (4)$$

This formulation aligns with principles from Random Matrix Theory: as the number of observations n increases relative to the universe size p , estimation error decreases¹, and we permit a higher weight on S . Conversely, in the $p \gg n$ regime, λ_{conf} decreases, automatically forcing stronger shrinkage toward the target. We cap the confidence at λ_{max} to ensure regularization is never fully disabled².

The final estimator for Σ is expressed as:

$$\hat{\Sigma} = T + (\lambda_{conf} \cdot \Omega_{obs}) \odot (S - T) \quad (5)$$

where $\Omega_{obs} = [\omega_{ij}]$ and $\odot$ denotes the Hadamard (element-wise) product. In this framework, pairs with insufficient history are purely the target T , while pairs with robust history utilize the sample covariance scaled by the current regime's dimensionality confidence.

3 Construction and Constraints

The adjusted covariance matrix $\hat{\Sigma}$ is used to construct the Minimum Variance Portfolio (MVP). We first solve for the unconstrained optimal weight vector w_{mvp} :

$$w_{mvp} = \hat{\Sigma}^{-1} \mathbf{1} \quad (6)$$

where $\mathbf{1}$ is a vector of ones. In rare cases where $\hat{\Sigma}$ is singular, the Moore-Penrose pseudoinverse is utilized. As a numerical stability safeguard, if the sum of the vector elements ($\mathbf{1}' w_{mvp}$) is non-positive (indicating a collapsed or indefinite covariance structure), the portfolio defaults to an Equal Weight (EW) allocation. Otherwise, the base weights are normalized:

$$w_{base} = \frac{w_{mvp}}{\mathbf{1}' w_{mvp}} \quad (7)$$

To maintain economic plausibility and strictly control gross exposure, we impose a gross leverage constraint L_{max} . We calculate the gross leverage of the base portfolio as $L = \|\mathbf{1}' w_{base}\|_1$. If $L > L_{max}$, the weights are linearly shrunk toward the Equal Weight portfolio (w_{EW}) to satisfy the cap exactly:

$$w = \alpha w_{base} + (1 - \alpha) w_{EW} \quad (8)$$

where the mixing coefficient α is derived as:

$$\alpha = \frac{L_{max} - 1}{L - 1} \quad (9)$$

This formulation preserves the relative bets of the minimum variance solution w_{mvp} while strictly bounding the gross exposure to the defined limit (L_{max}).

¹The spectral distribution of the sample covariance stabilizes

²This almost ensures that $\hat{\Sigma}$ is positive definite, and in turn that $\hat{\Sigma}$ remains well-conditioned

4 Calibration and Robustness

The strategy relies on a set of hyperparameters governing the lookback windows, observation thresholds, and portfolio constraints, detailed in [Table 1](#).

The model configuration is designed to be as general as possible, avoiding overfitting through hyperparameter tuning. In our tests, the constraints (λ_{max} and L_{max}) were never binding, H_{min} and H_{max} are not special nor where they tuned, and neither was N_{min} or N_{low} . Changing these values does not effect the performance of the portfolio much. The shrinkage formula λ_{conf} adapts dynamically based on the signal-to-noise ratio $\sqrt{n/p}$; this is motivated by the Marchenko-Pastur law, which establishes that the spectral spread of noise in a random matrix scales proportional to $\sqrt{p/n}$.

We utilize the Scaled Identity matrix as the shrinkage target. Unlike more complex targets that preserve asset-specific variances such as (Ledoit and Wolf, 2004), our target assumes variance homogeneity. This further reduces estimation noise by simplifying the problem from estimating p distinct diagonal variances to estimating a single global mean variance. By shrinking toward this diagonal structure, we add regularization to the covariance matrix. This all but ensures positive definiteness and invertibility in practice, despite the theoretical loss of guarantees due to element-wise weighting³.

Parameter	Value	Description	Note
Target (T)	Identity	Structure of target matrix T	Average variance scaled
H_{min}	14 days	Minimum history length	Selection threshold
H_{max}	252 days	Maximum lookback window	1 year
N_{min}	42 days	Full weight threshold	$3 \times H_{min}$
N_{low}	14 days	Zero weight threshold	$1 \times H_{min}$
λ_{conf}	$\min(\sqrt{n/p}, \lambda_{max})$	Sample Confidence	Dynamic limit on S
λ_{max}	0.8	Maximum Sample Confidence	Constraint on λ_{conf}
L_{max}	3.0	Maximum Gross Leverage	Constraint on $\ w\ _1$

Table 1: Hyperparameter Configuration

Note: The target matrix T serves as a “naive” prior assuming zero correlation and homogeneous variance. This biases the optimizer toward an equal-weighted allocation in the absence of sufficient data history.

H_{min} is set to 14 days to minimize selection latency, allowing the strategy to incorporate new market entrants quickly. N_{min} (42 days) is set to triple the selection threshold to ensure an asset accumulates significant history before its sample covariance fully impacts the portfolio. N_{low} is set to 14 days to provide a smooth ramp-up for partial overlaps.

λ_{conf} adapts to the dimensionality of the available data. When the universe (p) is large relative to the history length (n), the model automatically reduces confidence in the sample covariance to prevent overfitting. The cap at $\lambda_{max} = 0.8$ preserves a minimum 20% shrinkage intensity even in data-rich regimes. Note that with $H_{max} = 252$, λ_{conf} never hits the cap in large universes (e.g., $p > 400$), making the formula purely driven by the n/p ratio since λ_{max} is not binding.

The leverage in the US sample is never higher than 3 either, and $\hat{\Sigma}$ is never singular given the hyperparameter specification in the table. Other hyperparameters do not change the results much, but setting lambda e.g. to a high constant, would sometimes lead to $\hat{\Sigma}$ being singular and the portfolio being extreme before the safeguards. Thus these measures are added to make the portfolio robust even in edge case scenarios, and are not binding the actual in-sample portfolio at any moment. While the leverage constraint $L_{max} = 3.0$ and the singular matrix checks are rarely binding in standard market regimes (such as this US sample, where they never where binding), they serve as critical safeguards.

³In the US sample, $\hat{\Sigma}$ was always invertible

References

- Chopra, Vijay K. and Ziemba, William T. (1993). “The Effect of Errors in Means, Variances, and Covariances on Optimal Portfolio Choice”. In: *The Journal of Portfolio Management* 19.2, pp. 6–11. DOI: [10.3905/jpm.1993.409440](https://doi.org/10.3905/jpm.1993.409440).
- Ledoit, O. and Wolf, M. (2004). “Honey, I Shrunk the Sample Covariance Matrix”. In: *The Journal of Portfolio Management* 30.4, pp. 110–119.

A Code Appendix

A.1 Project Repository

The python file (`main.py`) and weights file (`weights.csv`), is hosted on Google Drive:

[Google Drive Repository Folder](#)

A.2 Python Source Files (.py)

The python script below was used to produce the weights following the logic in the documentation above.

A.2.1 `main.py`

```
1 import pandas as pd
2 import numpy as np
3
4 def main(chars: pd.DataFrame, features: pd.DataFrame, daily_ret: pd.DataFrame) -> pd.
    DataFrame:
5     """
6     Constructs a Minimum Variance Portfolio (MVP) using dynamic
7     Ledoit-Wolf-style shrinkage and leverage constraints.
8
9     Args:
10        chars: Stock characteristics (ctff_chars.parquet)
11        features: Computed features (ctff_features.parquet)
12        daily_ret: Historical daily returns (ctff_daily_ret.parquet)
13
14    Returns:
15        DataFrame with columns: id, eom, w
16    """
17
18    # --- Configuration Parameters ---
19    H_min = 14          # Minimum lookback for covariance estimation
20    H_max = 252         # Maximum lookback (1 trading year)
21    N_min = 42          # Scaling factor for pairwise observation confidence
22    N_low = 14          # Minimum overlap threshold for pairwise observations
23    lambda_max = 0.8    # Upper bound for the shrinkage intensity
24    ctff_test_data = True # Only look at stocks to be tested
25
26    def mvp_weights(sigma, max_leverage=3.0):
27        """
28        Calculates weights for the Minimum Variance Portfolio.
29        Uses pseudoinverse as a fallback for singular matrices.
30        Includes a shrinkage-to-equal-weight mechanism to enforce leverage constraints.
31        """
32        n = sigma.shape[0]
33        ones = np.ones(n)
34
35        try:
36            # Efficiently solve for Sigma * x = 1 instead of explicit inversion
37            x = np.linalg.solve(sigma, ones)
38        except np.linalg.LinAlgError:
39            # Moore-Penrose pseudoinverse handles singular/ill-conditioned matrices
40            sigma_pinv = np.linalg.pinv(sigma)
41            x = np.dot(sigma_pinv, ones)
42
43        x_sum = np.sum(x)
44
45        # Handle edge case where the system yields a near-zero denominator
46        if x_sum < 1e-8:
47            ew = np.ones(n) / n
48            return ew
49
50        weights = x / x_sum
```

```
51     current_leverage = np.abs(weights).sum()
52
53     if current_leverage > max_leverage:
54         # Linear interpolation (shrinkage) between MVP and Equal Weight (EW)
55         # to satisfy the max_leverage constraint exactly.
56         alpha = (max_leverage - 1) / (current_leverage - 1)
57         ew = np.ones(n) / n
58         return alpha * weights + (1 - alpha) * ew
59
60     return weights
61
62 def identity_target(S):
63     """
64     Generates a target shrinkage matrix (T) using the Identity matrix
65     scaled by the average variance of assets.
66     """
67     vars_diag = np.diag(S)
68     avg_var = np.nanmean(vars_diag)
69
70     if np.isnan(avg_var):
71         avg_var = 1.0
72     return np.eye(S.shape[0]) * avg_var
73
74 def run_monthly_rebalance_dynamic(
75     log_returns: pd.DataFrame,
76     universe_map: dict,
77     H_min: int,
78     H_max: int,
79     N_min: int,
80     N_low: int,
81     lambda_max: float,
82     target_func
83 ) -> pd.DataFrame:
84     """
85     Iterates through monthly rebalance dates, selecting the valid universe
86     from universe_map and calculating MVP weights.
87     """
88     # Identify rebalance dates (Month Ends) based on available return data
89     rebalance_dates = log_returns.index[H_min:].to_series().resample('ME').max()
90     results = []
91
92     for date in rebalance_dates:
93         # --- Data Slicing & Filtering ---
94         # Look up valid universe for this specific month
95         lookup_key = (date + pd.offsets.MonthEnd(0)).date()
96         candidate_ids = universe_map.get(lookup_key, set())
97
98         if not candidate_ids:
99             continue
100
101         # Slice Historical Data
102         current_idx = log_returns.index.get_loc(date)
103         start_idx = max(0, current_idx - H_max)
104
105         # Intersection of "Stocks allowed this month" AND "Stocks with return data"
106         valid_cols = list(candidate_ids.intersection(log_returns.columns))
107         if len(valid_cols) < 2:
108             continue
109
110         # Slice history window
111         hist_data = log_returns.iloc[start_idx: current_idx][valid_cols]
112
113         # Filter for Minimum History (H_min)
114         valid_counts = hist_data.count()
115         valid_stocks = valid_counts[valid_counts >= H_min].index
116         if len(valid_stocks) < 2:
117             continue
118
119         sub_returns = hist_data[valid_stocks].copy()
120
```

```
121     # --- Data Cleaning ---
122     # Remove stale price sequences (consecutive zeros) which bias volatility
downwards
123     is_zero = abs(sub_returns) < 1e-8
124     is_streak = (is_zero & is_zero.shift(1)) | (is_zero & is_zero.shift(-1))
125     sub_returns[is_streak] = np.nan
126
127     # --- Dynamic Shrinkage Intensity (lambda) ---
128     n_obs, p_vars = sub_returns.shape
129     lambda_conf = min(np.sqrt(n_obs / p_vars), lambda_max)
130
131     # --- Observation Weighting Matrix ---
132     mask = (abs(sub_returns) > 1e-8).astype(int)
133     count_matrix = mask.T @ mask
134     obs_weight = ((count_matrix - N_low) / N_min).clip(0, 1)
135
136     # --- Covariance Estimation ---
137     # Compute Sample (S) and Target (T) matrices (monthly horizon)
138     sub_returns = sub_returns.fillna(sub_returns.mean())
139     S = sub_returns.cov().values * 21
140     T = target_func(S)
141
142     # Fallbacks for NaNs in Target or Sample
143     if np.isnan(T).any():
144         avg_var = np.nanmean(np.diag(S)) if not np.all(np.isnan(np.diag(S))) else
1.0
145         T = np.where(np.isnan(T), np.eye(len(valid_stocks)) * avg_var, T)
146     if np.isnan(S).any():
147         S = np.where(np.isnan(S), T, S)
148
149     # --- Shrink Covariance Matrix ---
150     eff_weight = lambda_conf * obs_weight
151     Sigma_shrunked = T + eff_weight * (S - T)
152
153     # --- Calculate MVP weights ---
154     try:
155         w = mvp_weights(Sigma_shrunked)
156     except np.linalg.LinAlgError:
157         print(f"\rError on {date.date()}: Fallback to equal weights")
158         w = np.ones(len(valid_stocks)) / len(valid_stocks)
159
160     # Store results
161     results.append(pd.DataFrame({
162         'eom': date,
163         'id': valid_stocks,
164         'w': w
165     }))
166
167     print(f"\rRebalanced {date.date()}: {len(valid_stocks)} stocks (Lambda: {
lambda_conf:.2f})", end="")
168
169     if results:
170         return pd.concat(results, ignore_index=True)
171     return pd.DataFrame(columns=['eom', 'id', 'w'])
172
173
174 # --- Main Execution Setup ---
175 chars['eom'] = pd.to_datetime(chars['eom'])
176 daily_ret['date'] = pd.to_datetime(daily_ret['date'])
177
178 # Prepare Universe Map
179 if ctff_test_data and 'ctff_test' in chars.columns:
180     valid_universe = chars[chars['ctff_test'] == True].copy()
181 else:
182     valid_universe = daily_ret[['id', 'date']].copy()
183     valid_universe['eom'] = valid_universe['date'] + pd.offsets.MonthEnd(0)
184     valid_universe = valid_universe[['id', 'eom']].drop_duplicates()
185     valid_universe['eom'] = (pd.to_datetime(valid_universe['eom']) + pd.offsets.MonthEnd
(0)).dt.date
186
```

```
187 # Lookup dictionary {Date -> Set(IDs)}
188 universe_map = valid_universe.groupby('eom')['id'].apply(set).to_dict()
189
190 # Prepare Returns Data
191 daily_ret['ret_exc'] = daily_ret['ret_exc'].clip(lower=-0.999)
192 daily_ret_wide = daily_ret.pivot(index="date", columns="id", values="ret_exc").
193   sort_index()
194 log_returns = np.log1p(daily_ret_wide)
195
196 # Run Rebalance
197 pf_weights = run_monthly_rebalance_dynamic(
198     log_returns=log_returns,
199     universe_map=universe_map,
200     H_min=H_min,
201     H_max=H_max,
202     N_min=N_min,
203     N_low=N_low,
204     lambda_max=lambda_max,
205     target_func=identity_target,
206 )
207
208 # Final Formatting
209 pf_weights['eom'] = pd.to_datetime(pf_weights['eom']) + pd.offsets.MonthEnd(0)
210 pf_weights['id'] = pf_weights['id'].astype(int)
211 pf_weights['w'] = pf_weights['w'].astype(float)
212
213 return pf_weights[['id', 'eom', 'w']]
```

Listing 1: main.py