Efficient Portfolio Risk Management using np.einsum

Peter Decrem

03/23/25

1 Summary

The below notions have been implemented in a very fast and efficient fashion. We think that the simplicity (you just have to agree on the PnL and risk formula) and efficiency are the strongest points of the approach. They should allow for testing and tracking of many strategies within seconds. The grouping approach (of which the current UMAP and K-means is a special case) allows for a top down approach, and can be expanded to the notion of a stack of grouping matrices.

1.1 Portfolio Exposures as a Matrix

Portfolio exposures are expressed as a matrix where each row corresponds to a country (n rows) and each column represents a tenor (k columns). In our implementation, we extend this notion by adding futures as additional tenors.

1.2 Time Series of Rate Changes

We maintain a time series of changes in rates, which represent the fluctuations in risk factors over time. These changes are essential for understanding the dynamics of market risk and form the basis for computing daily profit and loss (PnL) as well as risk measures.

1.3 Daily Expected PnL Computation

For a fixed portfolio, the daily expected PnL—ignoring carry effects—is calculated as the sum of the product of the daily rate changes and the corresponding exposures. Formally:

$$\mathrm{PnL}_t = \sum i = 1^n \sum_{j=1}^{k} W_{ij}\, \Delta X_{ij}(t).$$

In matrix form, this is equivalent to taking the inner product of W and $\Delta X(t)$.

```
Python Implementation (using np.einsum):
np.einsum('ij,ij->', W, dX)
```

1.4 Risk Measurement via Covariance

The risk associated with a portfolio is quantified as the square root of a quadratic form: the exposures transposed multiplied by a covariance matrix (computed over a moving window) and then multiplied by the exposures. In summation form:

$$\sigma = \sqrt{w^\top \Sigma\, w} \quad \text{where} \quad w = \text{vec}(W).$$

```
Python Implementation (using np.einsum):
np.einsum('i,ij,j->', w, Sigma, w)  # then take sqrt
```

1.5 Grouping of Risk Exposures

We define a mechanism to group the risk matrix, whereby every individual risk can be allocated to a specific group. In summation form:

$$W_{ij} = \sum_{g=1}^{G} \mathcal{D}_{gij}.$$

Our implementation uses UMAP clustering combined with K-means to create this grouping matrix, ensuring that all risks are properly covered and categorized. Other groupings can be contemplated and implemented based on specific user requirements.

1.6 Portfolio Strategies and Custom Exposures

Portfolios are subdivided into different strategies, managed by individual traders. Each strategy is represented as a custom portfolio exposure, which allows for tracking the performance of strategies—enabling one to see what worked and what did not. This segmentation facilitates a detailed analysis of risk and return at the strategy level. The implementation allows for this to scale to 10s of strategies per trader in a compute efficient fashion.

1.7 Hedge Portfolios

Hedges are treated as specialized portfolios designed to mitigate risk in a strategy or a group of strategies. Users can create these hedge portfolios, and our implementation supports techniques such as orthogonal projection onto the closest futures and ridge regression. This allows for selective hedging of particular risk subsets with chosen securities, thereby enhancing risk management precision.

1.8 Backtesting and Regime Analysis

The framework enables a set of strategies to be run backward in time to identify periods when the largest PnL and risk occurred. This retrospective analysis aids in understanding under which market regimes a portfolio has significant exposure. Once identified, these covariance scenarios can be saved for future use as stress testing for other portfolios. The same approach can also be used to train and test advanced analytical methods, such as the ridge regression described in Section 7. Future implementations will integrate visualization tools to better illustrate these regimes.

1.9 Hedge Efficiency Evaluation

Hedge efficiency is defined as the ratio of risk with and without the hedging strategies in place. By tracking this ratio over time, risk managers can identify periods during which the hedges were effective and when they were not, providing a vital metric for evaluating hedging performance.

1.10 Vectorized Implementation for Fast Performance

All of these computations are implemented in a fully vectorized manner using NumPy's np.einsum. This approach leverages highly efficient array operations to perform all calculations—whether computing PnL, measuring risk, or performing groupings—in a single call. The result is a high-performance system capable of rapid risk allocation in multiple dimensions.

1.11 Scenario Analysis

Any scenario, represented as a shock matrix applied to the country and tenor dimensions, can be incorporated to assess its impact on PnL for a given strategy or hedge. In this framework, a scenario is defined by a shock matrix that captures hypothetical rate changes. For further details on the mathematical formulations and implementation, please refer to the Appendix.

Modern technologies were used to assist in generating attached documentation.

2 Appendix

The following sections provide the full technical details, including all computations and their vectorized implementation using NumPy's np.einsum.

2.1 The Risk Matrix and Time Series

- W be the DV01 risk matrix for a single portfolio, where each element represents the risk for country i and tenor j.
- dX_t be the matrix of rate changes at time t.
- For multiple portfolios, exposures are represented as W_p, where P is the number of portfolios.

2.2 Daily PnL Computation

For a fixed portfolio, the daily PnL (ignoring carry) is calculated as:

$$\text{PnL}t = \sum i = 1^n \sum_{j=1}^{k} W_{ij}\, \Delta X_{ij}(t).$$

Python Implementation (using np.einsum):

```
np.einsum('ij,ij->', W, dX)
```

2.3 Portfolio Risk Computation

Portfolio risk is computed by first vectorizing W into w, then:

$$\sigma = \sqrt{w^\top \Sigma\, w}.$$

Python Implementation (using np.einsum):

```
sigma_sq = np.einsum('i,ij,j->', w, Sigma, w)

sigma = np.sqrt(sigma_sq)
```

2.4 Risk Decomposition

If

$$\mathcal{D} \in \mathbb{R}^{G\times n\times k}$$

is the risk decomposition tensor, then:

$$W_{ij} = \sum_{g=1}^{G} \mathcal{D}_{gij}.$$

Python Implementation (using np.einsum):

```
# Summation over groups to reconstruct W

W_reconstructed = np.einsum('gij->ij', D)

# Compute grouped risk if needed

grouped_risk = np.einsum('gij->g', D)
```

2.5 Python Implementation Using np.einsum

```
Below is the complete Python code implementing all computations:

import numpy as np


# Dimensions:
```

```
# n = number of countries, k = number of tenors, P = number of portfolios, G = number of risk groups

n, k, P, G = 5, 3, 3, 2


# ----------------------------------------

# Base computations for a single portfolio

# ----------------------------------------

W = np.random.rand(n, k)       # DV01 risk matrix

dX = np.random.randn(n, k)     # Risk factor changes


# Daily PnL

pnl_single = np.einsum('ij,ij->', W, dX)

print("Daily PnL (single portfolio):", pnl_single)


# Portfolio risk

w = W.flatten()

nk = n * k

Sigma = np.random.rand(nk, nk)

Sigma = 0.5 * (Sigma + Sigma.T)  # Make it symmetric

sigma_sq = np.einsum('i,ij,j->', w, Sigma, w)

sigma_single = np.sqrt(sigma_sq)
```

```
print("Portfolio Risk (sigma, single portfolio):", sigma_single)

# ---------------------------

# Risk decomposition

# ---------------------------

D = np.random.rand(G, n, k)  # Decomposition tensor

W_reconstructed = np.einsum('gij->ij', D)

grouped_risk = np.einsum('gij->g', D)

print("Reconstructed W from risk groups:\n", W_reconstructed)

print("Grouped Risk (per group):", grouped_risk)

# -----------------------------------------------

# Extended computations for multiple portfolios

# -----------------------------------------------

W_all = np.random.rand(P, n, k)

dX = np.random.randn(n, k)  # Common changes for all portfolios

# PnL per portfolio, per country

pnl_portfolio_country = np.einsum('pnj,nj->pn', W_all, dX)

print("\nPnL per portfolio, per country:\n", pnl_portfolio_country)
```

```
# Summation over portfolios -> total PnL per country

pnl_total_country = np.sum(pnl_portfolio_country, axis=0)

print("Total PnL per country (all portfolios):\n", pnl_total_country)


# ------------------------------------------

# Risk per country computations

# ------------------------------------------

Sigma_all = np.empty((n, k, k))

for i in range(n):

    A = np.random.rand(k, k)

    Sigma_all[i] = 0.5 * (A + A.T)  # Symmetric block


# Risk per portfolio, per country

risk_portfolio_country_sq = np.einsum('pnj,ijk,pnk->pn', W_all, Sigma_all, W_all)

risk_portfolio_country = np.sqrt(risk_portfolio_country_sq)

print("\nRisk per portfolio, per country:\n", risk_portfolio_country)


# Aggregated risk across portfolios

W_total = np.sum(W_all, axis=0)
```

```
risk_total_country_sq = np.einsum('nj,ijk,nk->n', W_total, Sigma_all, W_total)

risk_total_country = np.sqrt(risk_total_country_sq)

print("Total risk per country (all portfolios):\n", risk_total_country)
```

2.6 Summary Table of Notations and Formulas (Python Implementations)

Notation	Description	Summation Formula	Python Implementation
W_{ij}	DV01 risk for country i and tenor j	W_{ij}	W (shape: (n, k))
$\Delta X_{ij}(t)$	Change in risk factor at cell (i, j) at time t	$\Delta X_{ij}(t)$	dX (shape: (n, k))
PnL (single portfolio)	Daily profit and loss	$\sum_{i=1}^{n}\sum_{j=1}^{k} W_{ij}\Delta X_{ij}(t)$	$np.einsum('ij,ij->', W, dX)$
Σ	Covariance matrix (flattened, size (nk)x(nk))	$\Sigma \in \mathbb{R}^{(nk)\times(nk)}$	Sigma (shape: (nk, nk))
Portfolio Risk	Risk (volatility) measure	$\sigma = \sqrt{w^T \Sigma w}$	$np.einsum('i,ij,j->', w, Sigma, w)$ (then take sqrt)
$\mathcal{D}_{gij}$	Risk decomposition contribution for group g	$W_{ij} = \sum_{g=1}^{G} \mathcal{D}_{gij}$	$np.einsum('gij->ij', D)$ (to reconstruct W)
Group PnL	PnL for each risk group (using the decomposition tensor)	$\Delta \text{PnL}^{(g)} = \sum_{ij} \mathcal{D}_{gij}\Delta X_{ij}$	$np.einsum('gij,ij->g', D, dX)$
Group Risk	Risk for each risk group (assuming identity covariance for simplicity)	$\sigma^{(g)} = \sqrt{\sum_{ij} \mathcal{D}_{gij}^2}$	$np.sqrt(np.einsum('gij,gij->g', D, D))$
PnL per Country, per Portfolio	PnL aggregated by country for each portfolio	$\text{PnL}_i^p = \sum_j W_{ij}^p \Delta X_{ij}$	$np.einsum('pnj,nj->pn', W\_all, dX)$
Risk per Country, per Portfolio	Risk for each portfolio and country (block-diagonal covariance)	$\sigma_{i,p}^2 = W_{ij}^p \Sigma_{i,jk} W_{ik}^p$	$np.einsum('pnj,ijk,pnk->pn', W\_all, Sigma\_all, W\_all)$
Total Risk per Country	Aggregated risk for each country across portfolios	$\sigma_i^2 = \left(\sum_p W_{ij}^p\right) \Sigma_{i,jk} \left(\sum_p W_{ik}^p\right)$	$np.einsum('nj,ijk,nk->n', W\_total, Sigma\_all, W\_total)$

2.7 Conclusion

This integrated approach demonstrates that by fully vectorizing risk exposures, daily PnL, and risk measures, all computations can be efficiently executed using a single np.einsum call. The framework supports comprehensive, fast risk allocation across multiple dimensions—including country-level analysis, custom strategies, hedging, and scenario analysis—thus enabling rapid performance tracking and informed decision-making.

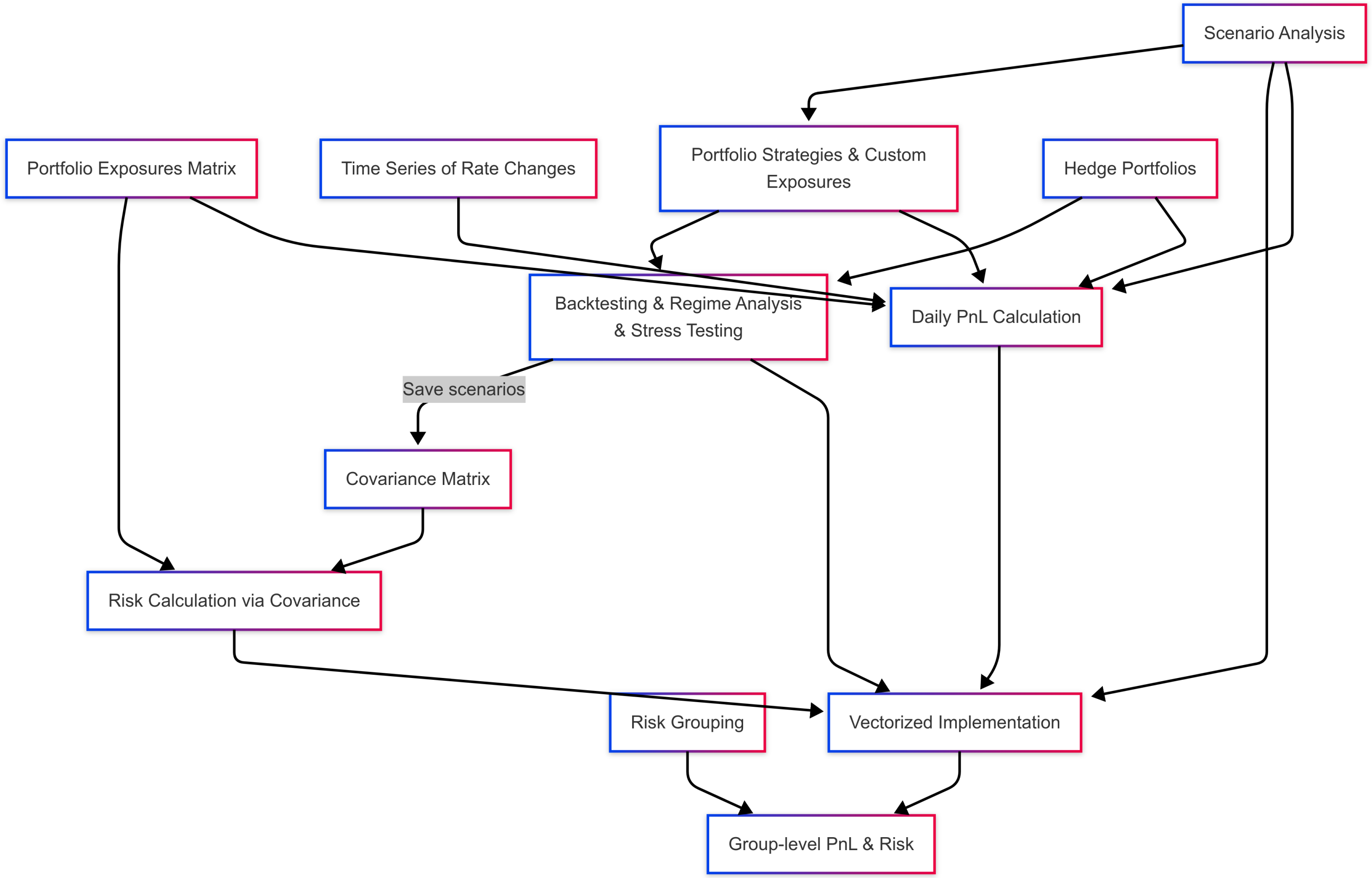
2.8 Scenario Analysis

Python Implementation (using np.einsum):

```
# Define a scenario shock matrix S (n x k)

S = np.random.randn(n, k)


# Scenario impact on PnL for a single portfolio

scenario_pnl_single = np.einsum('ij,ij->', W, S)


# Scenario impact on PnL per portfolio (W_all of shape (P, n, k))

scenario_pnl_portfolio = np.einsum('pnj,nj->pn', W_all, S)


# Aggregate exposures over portfolios, then compute scenario impact per country

W_total = np.sum(W_all, axis=0)

scenario_pnl_country = np.einsum('ij,ij->i', W_total, S)
```

2.9 Flow Diagram

Below is a conceptual flow diagram illustrating the overall structure of the approach:


Scenario Analysis
Portfolio Exposures Matrix
Time Series of Rate Changes
Portfolio Strategies & Custom Exposures
Hedge Portfolios
Backtesting & Regime Analysis & Stress Testing
Daily PnL Calculation
Save scenarios
Covariance Matrix
Risk Calculation via Covariance
Risk Grouping
Vectorized Implementation
Group-level PnL & Risk

In this diagram, you can see how the Portfolio Exposures Matrix, Time Series of Rate Changes, Portfolio Strategies & Custom Exposures, and Hedge Po