Portfolio Risk Calculator: Project Summary

Saxon Lee

29th of March 2025

Abstract

This is a summary of my portfolio risk calculation project, a Python-based tool designed to calculate Value-at-Risk (VaR) and Expected Shortfall (ES) for financial portfolios. The project implements two primary methodologies: the Parametric (Variance-Covariance) method and Monte Carlo simulation. It draws significantly from concepts outlined in Thierry Roncalli's Handbook of Financial Risk Management. This summary details the project's objectives, the risk management concepts applied, the structure of the Python codebase, and how each component contributes to the final risk metric calculations.

Contents

1 Introduction **2**

2 Core Risk Management Concepts Applied **2**
2.1 Value-at-Risk (VaR) . . . 2
2.2 Expected Shortfall (ES) . . . 2

3 Methodologies Implemented **2**
3.1 Parametric (Variance-Covariance) Method . . . 2
3.2 Monte Carlo Simulation Method . . . 3

4 Project Structure and Python File Walkthrough **3**
4.1 `src/main.py` . . . 3
4.2 `src/parametric_method.py` . . . 4
4.3 `src/monte_carlo_method.py` . . . 4
4.4 `src/utils.py` . . . 5
4.5 `config/portfolio_config.py` (Currently Embedded) . . . 5
4.6 `tests/test_parametric.py` and `tests/test_monte_carlo.py` . . . 5

5 Conclusion **6**

1 Introduction

The primary goal of the Portfolio Risk Calculator project is to provide a practical implementation of key financial risk measures. By developing functions to calculate Value-at-Risk (VaR) and Expected Shortfall (ES), the project aims to offer insights into potential portfolio losses under different market assumptions. The methodologies and theoretical underpinnings are largely based on established financial literature, with Thierry Roncalli's "Handbook of Financial Risk Management" (Chapman & Hall/CRC, 2020) serving as a key reference, particularly for the formulas and conceptual explanations found in its early chapters (e.g., Chapter 2).

This project emphasizes clarity in implementation, modularity in code structure, and adherence to sound financial theory, making it a valuable tool for understanding and applying risk management principles.

2 Core Risk Management Concepts Applied

The project focuses on two fundamental risk measures:

2.1 Value-at-Risk (VaR)

VaR is a widely used risk measure that estimates the maximum potential loss a portfolio could face over a specified time horizon, at a given confidence level, under normal market conditions. For example, a 10-day, 99% VaR of \$1 million signifies a 1% probability that the portfolio will lose more than \$1 million in the next 10 days. The project calculates VaR both as a monetary value and as a percentage return.

2.2 Expected Shortfall (ES)

Expected Shortfall (ES), also known as Conditional VaR (CVaR), quantifies the expected loss given that the loss exceeds the VaR threshold. It provides a more comprehensive view of tail risk than VaR by considering the magnitude of losses beyond the VaR point. ES is generally considered a more coherent risk measure. The project calculates ES in conjunction with VaR.

These concepts are central to modern risk management and are extensively discussed in Roncalli (2020), forming the theoretical basis for the calculations performed by this software.

3 Methodologies Implemented

Two distinct methodologies are implemented to calculate VaR and ES:

3.1 Parametric (Variance-Covariance) Method

This method assumes that asset returns are normally distributed, and consequently, the portfolio's return is also normally distributed.

- **Assumptions:** Normality of returns, Independent and Identically Distributed (IID) daily returns (allowing for time scaling where mean scales with T and volatility with $\sqrt{T}$).
- **Process:**
    1. Calculate the portfolio's expected daily return ($E[R_p]$) and daily volatility (σ_p) from individual asset weights (w_i), expected daily returns (μ_i), daily volatilities (σ_i), and their correlation matrix (ρ_{ij}). The portfolio daily variance is $\sigma^2_{p,daily} = \mathbf{w}^T \Sigma_{daily} \mathbf{w}$, where Σ_{daily} is the daily covariance matrix.
    2. Scale these daily figures to the target time horizon (T).

3. VaR is calculated using the Z-score (Z_α) from the standard normal distribution for a given confidence level ($1-\alpha$): $\text{VaR}_{\text{return}}(\alpha) = E[R_{p,T}] + \sigma_{p,T} \times Z_\alpha$.
4. ES is calculated using the formula: $ES_{\text{return}}(\alpha) = E[R_{p,T}] - \sigma_{p,T}\frac{\phi(Z_\alpha)}{\alpha}$, where $\phi(\cdot)$ is the PDF of the standard normal distribution.

These formulas align with those presented in Roncalli (2020, Chapter 2).

3.2 Monte Carlo Simulation Method

This method uses computational simulation to generate a distribution of potential portfolio returns, from which VaR and ES are derived.

- **Assumptions:** Asset returns are modeled using a multivariate normal distribution. Daily returns are simulated and compounded.
- **Process:**
  1. The daily covariance matrix of asset returns is used. Cholesky decomposition ($LL^T = \Sigma_{daily}$) is performed to generate correlated random asset returns.
  2. For a large number of simulations (N_{sim}):
     - For each day in the time horizon T:
       * Generate a vector of independent standard normal random numbers (Z_{day}).
       * Simulate correlated daily asset returns: $\Delta R_{assets,day} = \text{daily\_returns} + L \cdot Z_{day}$.
       * Calculate the portfolio's daily return and compound the portfolio value.
     - The final portfolio return over the horizon is recorded for each simulation.
  3. The N_{sim} simulated horizon returns are sorted.
  4. VaR is the return at the α-th percentile of this sorted distribution.
  5. ES is the average of all simulated returns that are worse than or equal to the VaR return.

4 Project Structure and Python File Walkthrough

The project is organized into several Python files within a structured directory layout, promoting modularity and maintainability.

4.1 `src/main.py`

This is the main orchestration script.

- **Purpose:** Serves as the entry point for the calculator. It loads portfolio configurations, calls the calculation functions for both parametric and Monte Carlo methods, and uses utility functions to display the results.
- **Functionality:**
  - Defines (currently embeds) the `DEFAULT_PORTFOLIO` configuration.
  - Prepares daily financial figures (returns, volatilities, covariance matrix) from annualized inputs.
  - Invokes `calculate_parametric_var_es` from `parametric_method.py`.
  - Invokes `calculate_monte_carlo_var_es` from `monte_carlo_method.py`.

- Uses `display_results` from `utils.py` to present the findings.
- Optionally, it can generate a histogram of Monte Carlo simulated returns using `matplotlib`.

- **Risk Concept Application:** This file directly applies the overall framework by calling the specific risk calculation modules and managing the flow of data (portfolio parameters, risk metrics).

4.2 `src/parametric_method.py`

This module contains the logic for the Parametric (Variance-Covariance) method.

- **Purpose:** To calculate VaR and ES assuming normally distributed returns.
- **Key Function:** `calculate_parametric_var_es(portfolio_config)`
  - Takes portfolio configuration as input.
  - Converts annual returns and volatilities to daily figures using `convert_annual_to_daily` from `utils.py`.
  - Computes portfolio daily mean return, daily variance (using matrix multiplication: $\mathbf{w}^T \Sigma_{daily} \mathbf{w}$), and daily volatility.
  - Scales these to the specified time horizon.
  - Uses `scipy.stats.norm.ppf` for the Z-score (Z_α) and `scipy.stats.norm.pdf` for the PDF value ($\phi(Z_\alpha)$) needed for ES.
  - Returns VaR and ES as both monetary values and percentage returns.
- **Risk Concept Application:** Directly implements the mathematical formulas for parametric VaR and ES based on the normality assumption, as discussed in Roncalli (2020, Chapter 2).

4.3 `src/monte_carlo_method.py`

This module implements the Monte Carlo simulation for VaR and ES.

- **Purpose:** To estimate VaR and ES by simulating a large number of possible portfolio return outcomes.
- **Key Function:** `calculate_monte_carlo_var_es(portfolio_config, daily_returns, daily_vols, daily_covariance_matrix)`
  - Takes portfolio configuration and pre-calculated daily figures as input.
  - Performs Cholesky decomposition (`np.linalg.cholesky`) on the daily covariance matrix to get the L factor for generating correlated returns.
  - Iterates for the specified number of simulations:
    - For each day in the time horizon, it generates correlated asset returns using L and random standard normal numbers (`np.random.normal`).
    - Calculates the daily portfolio return and compounds the portfolio value over the horizon.
  - Sorts the resulting distribution of simulated horizon portfolio returns.
  - VaR is determined as the appropriate percentile (e.g., 1st percentile for 99% VaR) from the sorted returns.

- ES is calculated as the mean of the returns in the tail beyond the VaR point.
- Returns VaR and ES values/returns, and the array of all simulated returns.

- **Risk Concept Application:** Implements the simulation-based approach to risk estimation. The derivation of VaR as a percentile and ES as a conditional tail mean from the empirical distribution of simulated outcomes are direct applications of their definitions.

4.4 `src/utils.py`

This utility module provides helper functions used across the project.

- **Purpose:** To house common, reusable functions, promoting cleaner code in the main calculation modules.
- **Key Functions:**
  - `display_results(...)`: Formats and prints the VaR and ES results along with key input parameters for clarity.
  - `convert_annual_to_daily(...)`: Converts annualized return and volatility figures to their daily equivalents, correctly handling the square root scaling for volatility.
- **Risk Concept Application:** While not directly calculating risk metrics, it supports the process by ensuring correct data transformations (e.g., annual to daily scaling, which is crucial for applying daily models over longer horizons) and clear presentation of results.

4.5 `config/portfolio_config.py` (Currently Embedded)

Originally intended as a separate file for portfolio definitions.

- **Purpose:** To define and store parameters for various portfolios (asset names, weights, expected returns, volatilities, correlation matrices, risk parameters like confidence level and time horizon).
- **Current Status:** For ease of execution in certain environments, the `DEFAULT_PORTFOLIO` dictionary is currently embedded directly within `src/main.py`. The structure remains the same, defining all necessary inputs for the risk calculations.
- **Risk Concept Application:** This component is critical as it provides all the necessary inputs (market views, portfolio composition) upon which the risk calculations are based. The accuracy of these inputs directly impacts the relevance of the VaR and ES outputs.

4.6 `tests/test_parametric.py` and `tests/test_monte_carlo.py`

These files contain unit tests for their respective calculation modules.

- **Purpose:** To verify the correctness of the implemented VaR and ES calculation logic. They use Python's `unittest` framework.
- **Functionality:**
  - `test_parametric.py`: Includes tests for basic calculation runs, checks if ES is generally a worse loss than VaR, and verifies calculations against known results for simple cases (e.g., single asset, zero mean).
  - `test_monte_carlo.py`: Includes tests for basic runs, ES/VaR relationship, output shapes, and error handling (e.g., for non-positive definite covariance matrices). Uses `np.random.seed(42)` for reproducibility of MC tests.

- **Risk Concept Application:** Ensures that the implemented code accurately reflects the mathematical formulas and logical processes of the VaR and ES methodologies. For instance, testing against a known Z-score for a simple parametric case directly validates the core formula application.

5 Conclusion

The Portfolio Risk Calculator project successfully implements two key methodologies for calculating Value-at-Risk and Expected Shortfall. By leveraging Python and its scientific libraries, and by grounding the calculations in established financial theory as detailed in Roncalli's Handbook of Financial Risk Management, my project provides a functional and understandable tool for financial risk assessment. The modular structure and inclusion of unit tests contribute to the robustness and reliability of the calculator. Future enhancements could include additional methodologies, more sophisticated data input options, and advanced simulation techniques.