Python for Finance and Optimisation

Mohamed Mrad

M2-IRFA 2026-2027

Table des matières

General Introduction

Context and the Quantitative Revolution

Since the pioneering work of Harry Markowitz in 1952 on portfolio selection, the financial industry has undergone an unprecedented technological and mathematical transformation. Modern finance is no longer merely a discipline based on accounting valuation or macroeconomic intuition ; it has become quantitative, algorithmic, and data-driven.

Today, asset managers, investment banks, and *hedge funds* operate in an ecosystem where information spreads at the speed of light. Faced with the proliferation of asset classes (equities, bonds, derivatives, commodities, crypto-assets), the ability to model risk, capture market comovements, and allocate capital optimally has become a critical competitive advantage.

Why Python for Quantitative Finance ?

For several decades, the financial computing landscape was divided :

- **C++** dominated the *Front Office* for high-performance computing (exotic option pricing, high-frequency trading) due to its execution speed.
- **MATLAB and R** were favored by researchers (*Quants*) for statistical prototyping and econometrics.

Python has established itself as the industry's de facto standard by enabling a unique convergence. Although it is an interpreted language (theoretically slower than C++), Python relies on low-level libraries written in C/C++ (`NumPy`, `SciPy`, `Pandas`). This allows financial engineers to benefit from an expressive, readable, and extremely fast-to-develop syntax, while maintaining matrix computation performance close to native code.

Furthermore, Python's native integration with the *Machine Learning* ecosystem (`Scikit-Learn`, `PyTorch`) and its ability to handle large-scale data make it an essential tool for modern financial engineering.

Course Philosophy and Learning Objectives

The objective of this 18-hour course is to bridge the gap between the rigor of financial mathematical models and the reality of their computational implementation. Theoretical optimization often suffers from an "ivory tower" syndrome : textbook models assume perfect markets, Gaussian returns, and infinite liquidity.

This course is built around a pragmatic philosophy : **confront theory with the frictions of the real world**. The curriculum is structured around three fundamental pillars :

1. ***Data Engineering* and Market Infrastructure (Part I) :** Learn how to collect, clean, and synchronize heterogeneous financial data. We will examine how to handle market anomalies (*bad ticks*), deal with asynchronous market closures across global exchanges, and model the time-varying nature of volatility (EWMA).
2. **Advanced Risk Modeling (Part II) :** Move beyond the restrictive assumption of the Normal distribution. Real financial markets are characterized by frequent crashes and asymmetric behavior. We will learn how to model these "fat tails" and implement regulatory-grade risk measures (Basel III / Solvency II), such as *Value-at-Risk* (VaR) and *Expected Shortfall* (CVaR).
3. **Constrained Optimization Algorithms (Part III) :** Translate asset allocation strategies into numerical optimization problems. Beyond the classical Markowitz model, we

will code portfolios that are robust to market noise, including the Black-Litterman model, Equal Risk Contribution (*ERC*) portfolios, and approaches based on unsupervised *Machine Learning*.

At the end of this course, students will be able to design, code from end to end, and *backtest* (test against historical data) an automated and robust investment strategy suitable for a fund's production environment.

Prerequisites

To get the most out of this material, students are expected to have :

— A solid understanding of the fundamentals of Python programming (data structures, loops, functions).

— Basic knowledge of matrix algebra (vector products, matrix inversion) and probability/statistics (expectation, variance, normal distribution).

Chapitre 1

Foundations of Quantitative Finance & Data Engineering

Chapter Introduction

Professional Context and Challenges

Data infrastructure is the nervous system of quantitative finance. Before applying complex optimization models or Machine Learning algorithms, the Quantitative Developer or Risk Analyst must face a fundamental reality : market data are asynchronous, incomplete, noisy, and subject to the microstructural rules of the order book.
Poor data acquisition management or a misunderstanding of the nature of returns systematically invalidates the results of downstream models, according to the fundamental computing principle : *"Garbage in, Garbage out"* (corrupted input data lead to abnormal output results).
This chapter establishes the technical foundations of the course. We will cover the mathematical modeling of financial time, time zone management, and the essential Data Engineering tools required to build a clean, robust, and production-ready data ingestion pipeline.

Learning Objectives

By the end of this chapter, students will be able to :

1. Master the mathematical and economic distinction between arithmetic and logarithmic returns.
2. Identify and correct geographical desynchronization biases in global markets.
3. Design a complete Python pipeline to filter price anomalies (*bad ticks*) and align heterogeneous time series.
4. Model the unstable dynamics of short-term volatility using a recursive EWMA-type estimator.

Keywords

Mid-Price, Epps Effect, Log-Returns, Forward Fill, EWMA, Frobenius Norm, Rolling Z-Score.

1.1 Microstructural Concepts : Nature of Market Data

Before executing optimization algorithms, the quantitative developer must understand the infrastructure of data originating from the order book. A single price does not exist in professional markets.

1.1.1 Bid, Ask, Mid-Price and Bid-Ask Spread

At any time t, the market presents two price streams :
— P_t^{bid} (Bid Price) : The highest price at which a buyer is willing to purchase the asset.
— P_t^{ask} (Ask Price) : The lowest price at which a seller is willing to sell the asset.
By definition, $P_t^{\text{ask}} > P_t^{\text{bid}}$. The difference between these two prices is called the **Bid-Ask Spread** :

$$S_t = P_t^{\text{ask}} - P_t^{\text{bid}} \tag{1.1}$$

The spread is a direct indicator of market illiquidity. For modeling purposes, the **Mid-Price** is frequently used :

$$P_t^{\text{mid}} = \frac{P_t^{\text{ask}} + P_t^{\text{bid}}}{2} \tag{1.2}$$

1.1.2 Execution Price and VWAP

Daily databases (such as Yahoo Finance) provide the transaction closing price (*Close*). For large order volumes, portfolio managers use the **VWAP** (*Volume Weighted Average Price*), which weights each execution price by the traded volume over the period :

$$P_{\text{VWAP}} = \frac{\sum_{i=1}^{M} P_i \times V_i}{\sum_{i=1}^{M} V_i} \tag{1.3}$$

1.2 Mathematical Formalization of Returns

The evolution of an asset price P_t is modeled by a non-stationary stochastic process. Moving to returns makes it possible to work with stationary series (statistical properties that remain invariant over time).

1.2.1 Non-Additivity of Arithmetic Returns vs. Additivity of Logarithmic Returns

The arithmetic return over the period $[0, T]$, composed of T sub-periods, is :

$$R_{0\to T} = \frac{P_T - P_0}{P_0} = \prod_{t=1}^{T}(1 + R_t) - 1 \tag{1.4}$$

The logarithmic return (or log-return) is defined as :

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln(P_t) - \ln(P_{t-1}) \tag{1.5}$$

Time additivity can be demonstrated by telescoping the sum :

$$r_{0\to T} = \ln\left(\frac{P_T}{P_0}\right) = \ln\left(\frac{P_1}{P_0} \times \frac{P_2}{P_1} \times \cdots \times \frac{P_T}{P_{T-1}}\right) = \sum_{t=1}^{T} r_t \tag{1.6}$$

1.2.2 The Asset Aggregation Trap

The arithmetic return of a portfolio (R_p) is strictly the linear combination of the arithmetic returns of its N components, weighted by the weights w_i :

$$R_p = \sum_{i=1}^{N} w_i R_i \tag{1.7}$$

Conversely, due to the non-linearity of the logarithmic function ($\ln(A+B) \neq \ln(A)+\ln(B)$), the logarithmic return of a portfolio is **not** equal to the weighted sum of log-returns :

$$r_p \neq \sum_{i=1}^{N} w_i r_i \tag{1.8}$$

1.3 Data Engineering : Ingestion, Synchronization and Cleaning of Noisy Data

1.3.1 Geographical Desynchronization and the Epps Effect

When optimizing assets traded across different time zones (e.g., Paris and New York), trading windows overlap only partially. If a raw daily correlation matrix is calculated without adjustment, the coefficients are subject to an artificial downward bias known as the **Epps Effect**.

1.3.2 Anomaly Cleaning : Rolling Z-Score Filter

Real-world data feeds contain *bad ticks* (data-entry errors or abnormal price jumps caused by API bugs (Software interface bugs)). We implement a filter based on a rolling Z-Score to remove returns that deviate by more than 3 standard deviations from the local mean.

Z-Score

The **Z-Score** (or *standard score*) is a fundamental statistical tool. It measures **how many standard deviations an observation lies away from the mean of a group**.
It is an essential standardization method for comparing variables that come from completely different distributions or scales.

Practical Use and Concrete Example Imagine that you take two different exams :

— In **Mathematics**, you obtain **15/20** (the class average is 13).

— In **French**, you obtain **13/20** (the class average is 10).

To determine in which subject you actually performed better relative to the rest of the students, simply comparing the deviation from the mean is not sufficient. The dispersion of the grades (the standard deviation) must also be taken into account. This is precisely what the Z-Score allows us to do.

Mathematical Formula To calculate the Z-Score of an observation, the following formula is used :

$$Z = \frac{x - \mu}{\sigma} \tag{1.9}$$

Where :
- x is the value of the observation being studied (e.g., your grade).
- μ represents the **mean** of the population or group.
- σ represents the **standard deviation**, which measures the dispersion of the data around the mean.

Interpretation of Results The result is a real number (positive, negative, or zero) and can be interpreted as follows :
- $Z = 0$: The value is exactly equal to the group mean.
- $Z > 0$ (e.g., $+1.5$) : The value is above the mean. A score of $+1.5$ means that the observation lies 1.5 standard deviations above the mean.
- $Z < 0$ (e.g., -2) : The value is below the mean.

The Empirical Rule (Normal Distribution) For a normal distribution (Gaussian curve) :
- Approximately **68%** of observations have a Z-Score between -1 and $+1$.
- Approximately **95%** of observations have a Z-Score between -2 and $+2$.
- A Z-Score greater than $+3$ or lower than -3 is extremely rare (less than 1% probability), making it very useful for detecting anomalies or outliers (*outliers*).

1.3.3 Python Toolkit for Data Ingestion

- `yf.download()` : Queries and downloads financial data.

Example

The following line of code makes it possible to download historical data for the different financial assets under study :

```
raw_prices = yf.download(tickers, start="2022-01-01", end="2026-01-01")["Adj Close"]
```

The function `yf.download()` belongs to the `yfinance` library. It retrieves historical market data available from Yahoo Finance.
The `tickers` variable contains the different assets being studied :

$$\texttt{tickers} = [\texttt{'AAPL'}, \texttt{'\^{}FCHI'}, \texttt{'\^{}N225'}, \texttt{'BTC-USD'}]$$

These codes correspond respectively to Apple, the CAC 40, the Nikkei 225, and Bitcoin expressed in U.S. dollars.
The argument

```
start="2022-01-01"
```

indicates the start date of the period under study, while

```
end="2026-01-01"
```

indicates the end date. Since the date specified in `end` is generally exclusive, the period considered is therefore :

$$\boxed{\text{2022-01-01} \leq t < \text{2026-01-01}}$$

The function `yf.download()` retrieves several pieces of information for each asset, including :
— `Open` : opening price ;
— `High` : highest price of the trading session ;
— `Low` : lowest price of the trading session ;
— `Close` : closing price ;
— `Adj Close` : adjusted closing price ;
— `Volume` : traded volume.
The expression

```
["Adj Close"]
```

then selects only the adjusted closing prices. The other information is therefore removed from the DataFrame.
The resulting structure is a dataset in which rows correspond to dates and columns correspond to the different assets :

Date	AAPL	CAC 40	Nikkei 225	BTC-USD
t_1	$P_{1,\text{AAPL}}$	$P_{1,\text{CAC}}$	$P_{1,\text{Nikkei}}$	$P_{1,\text{BTC}}$
t_2	$P_{2,\text{AAPL}}$	$P_{2,\text{CAC}}$	$P_{2,\text{Nikkei}}$	$P_{2,\text{BTC}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

The variable `raw_prices` therefore contains a `pandas DataFrame` grouping the adjusted closing prices of the different assets over the period under study.
The use of adjusted prices is particularly appropriate for return analysis, as they account for certain events affecting price series, notably dividends and stock splits (*splits*).
These data can then be used to calculate logarithmic returns :

$$\boxed{r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)}$$

where P_t denotes the adjusted price at date t.
— `df.rolling().mean()` and `df.rolling().std()` : Compute the local statistics required for the rolling Z-Score.
— `np.abs()` : Returns the absolute value to identify two-sided deviations.
— **Filling missing values with `ffill()` and `bfill()`**
The following expression, used with the `pandas` library,

```
raw_prices.ffill().bfill()
```

allows missing values (NaN, meaning “Not a Number”) in a data series to be filled. It combines two successive operations :

1. `ffill()` (*forward fill*) ;
2. `bfill()` (*backward fill*).

1. `ffill()` : Forward filling. The `ffill()` method replaces each missing value with

the last known value located before it.
For example, consider the following series :

$$[100,\ \text{NaN},\ \text{NaN},\ 105,\ \text{NaN},\ 110].$$

After applying `ffill()`, we obtain :

$$[100,\ 100,\ 100,\ 105,\ 105,\ 110].$$

Thus, for a position i containing a missing value, `ffill()` uses the last available observation before i.

2. `bfill()` : Backward filling.
The `bfill()` method replaces a missing value with the first known value located after it.
For example :

$$[\text{NaN},\ \text{NaN},\ 100,\ 105,\ \text{NaN}]$$

becomes, after applying `ffill()` :

$$[\text{NaN},\ \text{NaN},\ 100,\ 105,\ 105].$$

The first two values remain missing because no observation is available before them. Applying `bfill()` then makes it possible to complete them :

$$[100,\ 100,\ 100,\ 105,\ 105].$$

3. Combining the two methods
The expression

$$\boxed{\texttt{raw\_prices.ffill().bfill()}}$$

therefore performs the following operations :

$$\texttt{raw\_prices} \xrightarrow{\texttt{ffill()}} \text{forward filling} \xrightarrow{\texttt{bfill()}} \text{filling of remaining missing values.}$$

Its behavior can be summarized as follows :

$$\text{NaN} \longrightarrow \begin{cases} \text{last known value before,} & \text{if it exists,} \\ \text{first known value after,} & \text{otherwise.} \end{cases}$$

This method is particularly useful for time series, for example when some price observations are missing. It therefore makes it possible to obtain a series containing no missing values, provided that at least one observation is known in the series.

```python
import yfinance as yf # Yahoo Finance
import pandas as pd # manipulate, clean, analyze and organize data
import numpy as np # mainly used for numerical computation

# Ingestion of an international and multi-asset portfolio
tickers = ['AAPL', '^FCHI', '^N225', 'BTC-USD']
raw_prices = yf.download(tickers, start="2022-01-01", end="2026-01-01")['Adj Close']

# AAPL (Apple stock), '^FCHI' (CAC 40 index),
# '^N225' (Japanese Nikkei 225 index),
```

```python
# BTC-USD (Bitcoin in US dollars)

# Filling liquidity gaps (specific public holidays)
prices_cleaned = raw_prices.ffill().bfill()

# Calculation of log returns
log_rets = np.log(prices_cleaned / prices_cleaned.shift(1)).dropna()

# BAD TICK FILTER: 20-day rolling Z-Score
rolling_mean = log_rets.rolling(window=20).mean()
rolling_std = log_rets.rolling(window=20).std()
z_scores = np.abs((log_rets - rolling_mean) / rolling_std)

# Replace extreme returns (Z-Score > 3) with 0 (price stagnation)
log_rets_filtered = log_rets.mask(z_scores > 3, 0)

print(f"Number of anomalies filtered and corrected: {(z_scores > 3).sum().sum()}
    ")
```

Listing 1.1 – Market Data Ingestion and Anomaly Filtering Pipeline

Note : Counting Outliers with `sum()`
The following Python expression is used to count the total number of values whose Z-Score is greater than 3 :

$$\boxed{\texttt{(z\_scores > 3).sum().sum()}}$$

The presence of two calls to `sum()` is explained by the fact that `z_scores` is generally a `pandas DataFrame`, i.e., a two-dimensional data structure composed of rows and columns.
The condition

$$\texttt{z\_scores > 3}$$

compares each value in the DataFrame with 3. It produces a matrix of Boolean values :

$$\begin{pmatrix} 2.1 & 4.5 & 1.2 \\ 3.8 & 1.5 & 5.2 \end{pmatrix} \longrightarrow \begin{pmatrix} \texttt{False} & \texttt{True} & \texttt{False} \\ \texttt{True} & \texttt{False} & \texttt{True} \end{pmatrix}.$$

In Python, Boolean values are interpreted as numbers when performing a sum :

$$\texttt{True} = 1, \qquad \texttt{False} = 0.$$

The previous matrix can therefore be represented as :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

First `sum()`

The first call to `sum()` performs a column-wise sum :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \xrightarrow{\texttt{sum()}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

This gives the number of values greater than 3 in each column.

Second `sum()`

The second call to `sum()` then adds the results obtained for each column :

$$1 + 1 + 1 = 3.$$

We therefore obtain the total number of values greater than 3 in the entire DataFrame.
Thus,

$$\boxed{\texttt{(z\_scores > 3).sum().sum()} = \text{total number of values such that } z > 3}$$

The first `sum()` performs the sum across columns, while the second adds the results obtained for all columns.

Note. If `z_scores` is a `Series` rather than a `DataFrame`, a single call to `sum()` is generally sufficient :

$$\boxed{\texttt{(z\_scores > 3).sum()}}$$

Note. `dropna()` is a pandas function used to remove rows or columns containing missing values (NaN).

1.4 Modeling Time-Varying Volatility : EWMA

Return volatility is not constant ; it exhibits clusters of high and low instability (the *Volatility Clustering* phenomenon). To capture this dynamic, we formalize the **EWMA** model (*Exponentially Weighted Moving Average*), which is notably used by *RiskMetrics*.
The variance at time t, denoted by σ_t^2, is an exponentially weighted average of the past variance and the most recent squared return :

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda) r_{t-1}^2 \tag{1.10}$$

where λ is the persistence parameter (generally set to 0.94 for daily data).

1.4.1 Python Toolkit for EWMA

— `df.ewm(alpha, adjust=False)` : Directly implements the EWMA recurrence. The parameter `alpha` is related to λ through the relationship $\alpha = 1-\lambda$. The argument `adjust=False` ensures the strict application of the recursive formula without normalization of the initial weights.

```
import matplotlib.pyplot as plt

# RiskMetrics lambda parameter
lambda_param = 0.94
alpha_param = 1 - lambda_param

# Daily EWMA variance calculation
ewma_variance = log_rets_filtered['AAPL'].pow(2).ewm(
    alpha=alpha_param,
    adjust=False
).mean()

# Annualization of EWMA volatility
```

```
ewma_vol_annualized = np.sqrt(ewma_variance * 252)

plt.figure(figsize=(12, 5))
plt.plot(
    ewma_vol_annualized,
    color='darkblue',
    label='EWMA Volatility (RiskMetrics $\lambda=0.94$)'
)
plt.title("Time-Varying Volatility Dynamics of Apple")
plt.xlabel("Date")
plt.ylabel("Annualized Volatility")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1.2 – EWMA Volatility Calculation and Visualization

Explanations : Variance Estimation Using the EWMA Method

The following line of code estimates the variance of the logarithmic returns of Apple stock (`AAPL`) using the EWMA (*Exponentially Weighted Moving Average*) method :

```
ewma_variance = log_rets_filtered['AAPL'].pow(2).ewm(
    alpha=alpha_param,
    adjust=False
).mean()
```

This expression can be broken down into several steps.

Selecting AAPL Returns

The instruction `log_rets_filtered['AAPL']` selects the time series of logarithmic returns of Apple stock.
The logarithmic return at date t is defined as :

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right),$$

where P_t represents the asset price at date t.

Squaring the Returns

The instruction `.pow(2)` squares each return :

$$r_t \longrightarrow r_t^2.$$

For example :

$$[-0.02,\ 0.01,\ -0.03] \longrightarrow [0.0004,\ 0.0001,\ 0.0009].$$

Squaring returns makes it possible, in particular, to measure their dispersion around zero and constitutes the basis for variance estimation.

Exponentially Weighted Moving Average

The instruction

```
.ewm(alpha=alpha_param, adjust=False)
```

sets up an exponentially weighted moving average (*Exponentially Weighted Moving Average*, EWMA).
This method assigns greater weight to recent observations than to older observations.
The EWMA variance can be represented by the recursive relationship :

$$\boxed{\widehat{\sigma}_t^2 = \alpha r_{t-1}^2 + (1-\alpha)\widehat{\sigma}_{t-1}^2}$$

where :

- r_{t-1} represents the logarithmic return in the previous period ;
- $\widehat{\sigma}_{t-1}^2$ represents the estimated variance in the previous period ;
- α is the smoothing parameter ;
- $0 < \alpha < 1$.

When α is high, recent observations have a greater influence on the variance estimate. Conversely, when α is low, older observations retain greater influence.

Calculation of the Mean

The instruction `.mean()` calculates the exponentially weighted mean of the squared returns. The EWMA variance can therefore be written as :

$$\boxed{\widehat{\sigma}_t^2 = \alpha r_{t-1}^2 + \alpha(1-\alpha) r_{t-2}^2 + \alpha(1-\alpha)^2 r_{t-3}^2 + \cdots}$$

The most recent observations therefore have the greatest weights.
Thus, the entire instruction

```
log_rets_filtered['AAPL'].pow(2).ewm(
    alpha=alpha_param,
    adjust=False
).mean()
```

can be summarized as :

logarithmic returns → squared returns → exponentially weighted average → EWMA variance

The variable `ewma_variance` therefore contains a time series representing the estimated conditional variance of AAPL returns.
Finally, to obtain EWMA volatility, it is sufficient to take the square root of the variance :

```
ewma_volatility = np.sqrt(ewma_variance)
```

which gives :

$$\boxed{\widehat{\sigma}_t = \sqrt{\widehat{\sigma}_t^2}}$$

EWMA volatility therefore provides a measure of volatility that assigns greater importance to recent market movements.

1.5 Matrix Algebra and Co-Movement Spaces

Let $\mathbf{r}_t = [r_{1,t}, r_{2,t}, \ldots, r_{N,t}]^T$ be the vector of returns at time t. The annualized variance-covariance matrix $\mathbf{\Sigma}$ (of dimension $N \times N$) is expressed in matrix form as :

$$\mathbf{\Sigma} = 252 \times \frac{1}{T-1} \sum_{t=1}^{T} (\mathbf{r}_t - \mu)(\mathbf{r}_t - \mu)^T \tag{1.11}$$

where μ is the vector of the average returns of each asset.
The Pearson correlation matrix is then given by :

$$\boxed{\mathbf{R} = \mathbf{D}^{-\frac{1}{2}} \mathbf{\Sigma} \mathbf{D}^{-\frac{1}{2}}}$$

where $\mathbf{\Sigma}$ denotes the covariance matrix and $\mathbf{D}$ the diagonal matrix of variances :

$$\mathbf{D} = \text{diag}\left(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\right).$$

Thus, each element of the correlation matrix $\mathbf{R}$ is given by :

$$\boxed{\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}}$$

where $\sigma_{ij} = \text{Cov}(r_i, r_j)$.

```python
import seaborn as sns # statistical data visualization

# Calculation of the annualized covariance matrix
Sigma = log_rets_filtered.cov() * 252

# Calculation of the Pearson correlation matrix
R = log_rets_filtered.corr()

# Professional visualization using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(
    R,
    annot=True,
    cmap='bwr',
    vmin=-1,
    vmax=1,
    linewidths=0.5
)
plt.title("Co-Movement Matrix (Correlation $R$)")
plt.show()
```

Listing 1.3 – Matrix Calculation of Covariance and Correlation

1.6 Case Study : Temporal Sampling Bias (Graded Assignment)

Problem : An asset management model rebalances its portfolios every month. Is it preferable to estimate the risk parameters ($\mathbf{\Sigma}$) using daily or weekly returns ?

1.6.1 Validation Algorithm

The student must run the following script to measure the Frobenius norm of the difference matrix. The Frobenius norm measures the overall distance between two matrices :

$$\|\mathbf{\Sigma}_{\text{Daily}} - \mathbf{\Sigma}_{\text{Weekly}}\|_F = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}(\sigma_{ij,D} - \sigma_{ij,W})^2} \tag{1.12}$$

```python
# 1. Daily Estimation
Sigma_daily = log_rets_filtered.cov() * 252

# 2. Weekly Estimation using specific resampling
prices_weekly = prices_cleaned.resample('W').last()
log_rets_weekly = np.log(
    prices_weekly / prices_weekly.shift(1)
).dropna()

Sigma_weekly = log_rets_weekly.cov() * 52

# 3. Calculation of the Frobenius matrix distance
matrix_diff = Sigma_daily - Sigma_weekly
frobenius_norm = np.linalg.norm(matrix_diff, ord='fro')

print("--- Quantitative Audit Results ---")
print("Difference matrix :\n", matrix_diff)
print(f"\nGlobal Frobenius distance : {frobenius_norm:.6f}")
```

Listing 1.4 – Case Study Resolution Script

1.6.2 Reflection Questions for Students

1. Why is the Frobenius distance not strictly equal to 0 even though both methods measure risk over the same historical period ?
2. What is the impact of the Epps Effect on the weekly estimation compared with the daily estimation ?

Chapitre 2

Risk Modeling, Non-Gaussian Estimation

Chapter Introduction

Professional Context and Challenges

On October 19, 1987, during « Black Monday », the Dow Jones index collapsed by 22.6% in a single trading session. Under a classical normal distribution framework, such a movement represents an event more than 20 standard deviations (20σ) away from the mean, with a probability of occurrence so small that it should not happen even on the timescale of the age of the universe. Yet the crashes of 1998 (LTCM), 2008 (Lehman Brothers), 2020 (Covid-19), or liquidity crises in crypto-assets remind us of an unavoidable mathematical reality : **real financial markets are not Gaussian**.

For a portfolio manager or banking regulator, blindly using the normal distribution is a deadly danger. It systematically leads to an underestimation of extreme loss risks. The objective of this chapter is to learn how to measure risk where it actually hides : in asymmetries and fat tails. We will formalize and implement the key indicators of institutional risk management, required by the international Basel III and Solvency II frameworks.

Learning Objectives

At the end of this chapter, students will be able to :

1. Mathematically diagnose and reject the normality assumption for an asset using the Jarque-Bera statistical test.
2. Model and interpret higher-order statistical moments (Skewness and Kurtosis) for highly volatile assets.
3. Implement in Python the algorithms for calculating *Value-at-Risk* (VaR) and *Expected Shortfall* (CVaR) using historical and parametric approaches.
4. Adjust risk calculations using the Cornish-Fisher expansion to incorporate non-Gaussian effects into capital requirements.

Keywords

Leptokurtic Distribution, Higher-Order Moments, Loss Quantile, Value-at-Risk, Expected Shortfall, Cornish-Fisher, Jarque-Bera Test.

2.1 The Limitations of Gaussian Finance

Classical financial theory (Bachelier, Markowitz, Black-Scholes) relies on the fundamental assumption that the logarithmic returns of assets follow a normal distribution (Gaussian distribution). Although this assumption greatly simplifies analytical calculations, it is systematically contradicted by real market data.

2.1.1 The Stylized Facts of Mandelbrot and Fama

As early as the 1960s, Benoît Mandelbrot and Eugene Fama highlighted the **stylized facts** of financial time series :

— **Fat Tails :** Extreme events (crashes, euphoric rallies) have a much higher probability of occurrence than predicted by the normal distribution.

— **Asymmetry (Negative Skewness) :** The return distributions of risky assets (such as stocks) often exhibit left skewness, indicating sharper downward movements than upward movements.

— **Volatility Clustering** : Volatility is not constant and propagates in clusters.

2.1.2 Mathematical Formalization of Higher-Order Moments

To quantify the deviation from normality, we analyze the centered moments of order 3 (**Skewness**) and order 4 (**Kurtosis**). Let r denote the random variable representing returns, $\mu = E[r]$ the expected value, and $\sigma = \sqrt{E[(r-\mu)^2]}$ the standard deviation.
Fisher's skewness coefficient is defined as :

$$S = E\left[\left(\frac{r-\mu}{\sigma}\right)^3\right] \tag{2.1}$$

A perfect Gaussian distribution has $S = 0$.
The kurtosis coefficient is defined as :

$$K = E\left[\left(\frac{r-\mu}{\sigma}\right)^4\right] \tag{2.2}$$

A Gaussian distribution has $K = 3$. In quantitative finance, we use the **Excess Kurtosis** ($K_{\text{ex}} = K - 3$). If $K_{\text{ex}} > 0$, the distribution is said to be **leptokurtic** : it has fat tails and a higher central peak.

2.2 Statistical Analysis of the Distribution in Python

2.2.1 Python Toolbox : The `scipy.stats` Submodule

To audit the statistical shape of our time series, we use specialized functions :

— `stats.skew(array)` : Computes the empirical skewness coefficient. A negative value indicates a left-stretched tail (more extreme latent losses).

Explanation : In quantitative finance, this statistic is particularly useful for studying the asymmetry of return distributions.
Intuitively, skewness determines whether observations are distributed symmetrically around their mean or whether one of the two tails of the distribution is more significant than the other.

The important term in the formula is the cube $(r-\mu)^3$. Unlike the square used to calculate variance, the cube preserves the sign of the deviation from the mean.
Thus, an observation located above the mean contributes positively :

$$r-\mu>0 \quad \Longrightarrow \quad (r-\mu)^3>0$$

Whereas an observation located below the mean contributes negatively :

$$r-\mu<0 \quad \Longrightarrow \quad (r-\mu)^3<0$$

Interpretation of the Sign The sign of the skewness coefficient identifies the direction of the distribution's asymmetry.
— A positive skewness corresponds to right skewness :
— A zero skewness corresponds to an approximately symmetric distribution :
— A negative skewness corresponds to left skewness :
A negative skewness therefore means that the distribution has a **longer or heavier left tail** than its right tail.

Example Consider the following return series :

$$\{-2\%,-1\%,0\%,1\%,2\%,-10\%\} \tag{2.3}$$

Most returns are relatively close to zero, but one particularly large loss of -10% is observed. This observation contributes strongly and negatively to the skewness calculation because it appears in the term :

$$(-10\%)^3<0$$

The cube amplifies the influence of observations that are very far from the mean.
Consequently, a few strongly negative returns can lead to a significantly negative skewness.

Why Is Skewness Important in Finance ? Two assets can have similar means and volatilities while exhibiting very different return distributions.
For example, one asset may exhibit relatively symmetric gains and losses, while another may exhibit many small gains but also a few extremely large losses.
In the second case, the distribution may exhibit negative skewness : $\text{Skewness}(r)<0$.
This indicates that the distribution is asymmetric toward extreme negative returns.
Skewness therefore provides information complementary to volatility. Volatility mainly measures the dispersion of returns around their mean, while skewness provides information about the **asymmetry of the distribution**.

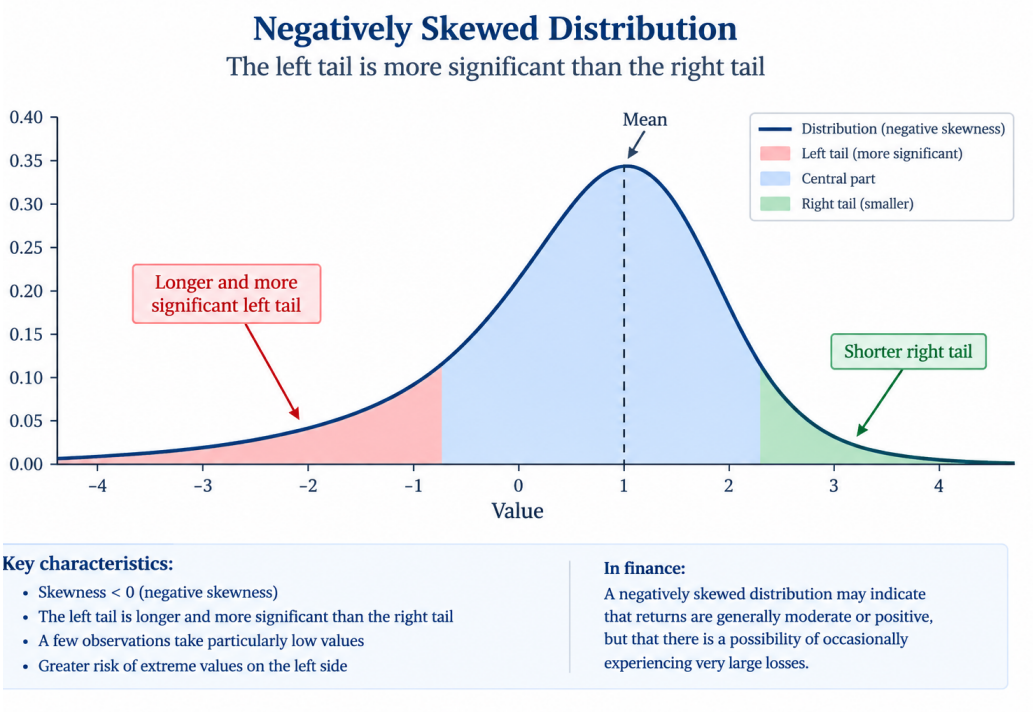

FIGURE 2.1 – Illustration of a negatively skewed distribution.

— `stats.kurtosis(array, fisher=True)` : allows the calculation of the kurtosis coefficient of a data series. In quantitative finance, kurtosis is particularly useful for studying the presence of extreme values in a return distribution. It can notably identify distributions exhibiting *fat tails*.

Example : The important term in the formula is :

$$(r - \mu)^4$$

The fourth power gives particularly strong importance to observations that are very far from the mean.
For example, consider two deviations from the mean :

$$r_1 - \mu = 1 \quad \text{and} \quad r_2 - \mu = 4.$$

With a power of two, the contributions would be :

$$1^2 = 1 \quad \text{and} \quad 4^2 = 16.$$

With a power of four, they become :

$$1^4 = 1 \quad \text{and} \quad 4^4 = 256 \tag{2.4}$$

The observation located four standard deviations from the mean therefore has a much greater influence on kurtosis. Kurtosis is thus particularly sensitive to extreme observations.

The Fisher Convention There are two conventions commonly used to define kurtosis. Pearson's kurtosis is given by :

$$K = \frac{\mathbb{E}\left[(r-\mu)^4\right]}{\sigma^4} \tag{2.5}$$

For a normal distribution, this quantity is $K = 3$. The Fisher convention consists of subtracting 3 from Pearson's kurtosis :

$$K_F = K_{ex} = \frac{\mathbb{E}\left[(r-\mu)^4\right]}{\sigma^4} - 3 \tag{2.6}$$

Thus, for a normal distribution :

$$K_F = 3 - 3 = 0 \tag{2.7}$$

This is the convention used when writing : `stats.kurtosis(array, fisher=True)`.
The parameter `fisher=True` therefore instructs the function to return the **excess kurtosis** rather than Pearson's kurtosis.

Interpretation of the Result When `fisher=True`, the reference value is therefore :

$$K_F = 0$$

The result can then be interpreted as follows :

$$K_F > 0 \quad \Longrightarrow \quad \text{fatter tails than those of a normal distribution}$$

$$K_F = 0 \quad \Longrightarrow \quad \text{kurtosis comparable to that of a normal distribution}$$

$$K_F < 0 \quad \Longrightarrow \quad \text{thinner tails than those of a normal distribution}$$

A distribution with positive Fisher kurtosis is generally described as **leptokurtic**.

What Does "Fat Tails" Mean ? A distribution has fat tails when it contains more observations very far from its mean than would be predicted by a normal distribution with the same mean and variance.
Consider, for example, a return series :

$$\{-10\%, -1\%, 0\%, 1\%, 12\%\}$$

The returns of -10% and 12% are very far from the mean.
Because of the fourth power present in the kurtosis formula, these observations have a significant influence :

$$(-10\%)^4 \quad \text{and} \quad (12\%)^4.$$

Thus, the presence of a few extremely large in absolute value returns can strongly increase kurtosis.

Financial Interpretation This property is particularly important when analyzing financial returns. In many financial markets, returns do not perfectly follow a normal distribution. They may exhibit **fat tails**, meaning that extreme events occur more frequently than predicted by a normal distribution.
For example, a model based on a normal distribution may assign a very low probability to a return of -8% whereas such an event may be less rare in actual financial data.
Positive kurtosis therefore highlights this characteristic :

$$K_F > 0 \implies \text{fat tails} \implies \text{more extreme observations}$$

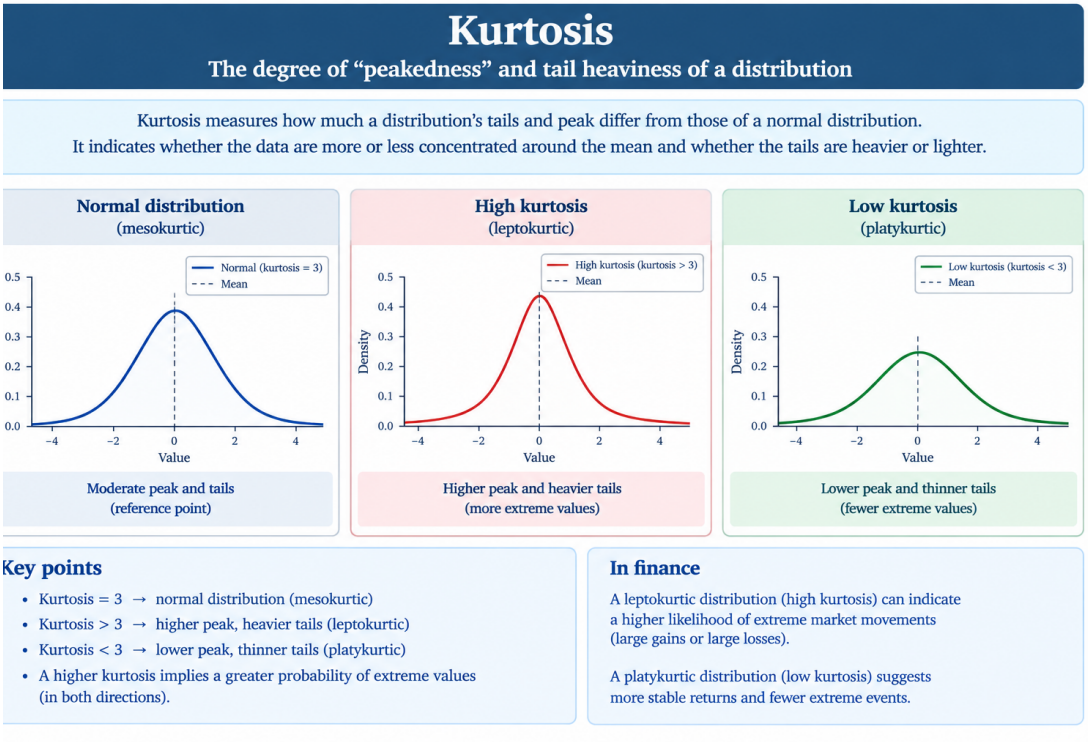

FIGURE 2.2 – Illustration of a negatively skewed distribution.

Difference Between Skewness and Kurtosis It is important to distinguish skewness from kurtosis.
Skewness primarily measures the asymmetry of the distribution :

$$\text{Skewness}(r) < 0 \implies \text{asymmetry toward negative returns} \tag{2.8}$$

Kurtosis primarily measures the importance of the distribution tails and the presence of extreme observations :

$$K_F > 0 \implies \text{fatter tails and more extreme values} \tag{2.9}$$

We can therefore remember :

$$\text{Skewness} \longrightarrow \text{distribution asymmetry} \tag{2.10}$$

and :

$$\text{Kurtosis} \quad \longrightarrow \quad \text{importance of extreme events} \tag{2.11}$$

An asset can therefore simultaneously exhibit negative skewness and positive kurtosis :

$$\text{Skewness}(r) < 0 \qquad \text{and} \qquad K_F > 0 \tag{2.12}$$

In this case, the return distribution exhibits both **asymmetry toward losses** and **fat tails**.
This combination is particularly important in risk management because it indicates that extreme losses can be both asymmetric and more frequent than suggested by a normal distribution.

2.3 The Jarque-Bera normality test :

Normality test based on the formula $JB = \frac{T}{6}\left(S^2 + \frac{K_{\text{ex}}^2}{4}\right)$, where :
— T represents the number of observations ;
— S represents skewness ;
— K represents excess kurtosis.

The statistic therefore combines the deviation from zero skewness and the deviation from zero excess kurtosis.
The function `stats.jarque_bera(.)` returns two values : the test statistic and the *p-value*. If the *p-value* < 0.05, the normality hypothesis is rejected at the 95% confidence level.

Explanation : The **Jarque-Bera** test determines whether a data series is compatible with a **normal distribution**. It is particularly useful in finance for checking whether returns can reasonably be modeled using a normal distribution.
The test is based on two characteristics of the distribution : **skewness** and **kurtosis**.
Recall that for a normal distribution : Skewness $= 0$ and and, when using Fisher's excess kurtosis : Kurtosis $= 0$, so that $JB = 0$.

Test Hypotheses The null hypothesis is :

$$H_0 : \text{the distribution of the data is normal} \tag{2.13}$$

The alternative hypothesis is :

$$H_1 : \text{the distribution of the data is not normal} \tag{2.14}$$

Interpretation of the p-value A significance level of 5% is generally used :

$$\alpha = 0.05 \tag{2.15}$$

If :

$$p\text{-value} < 0.05 \tag{2.16}$$

then H_0 is rejected and we conclude that the data exhibit characteristics significantly different from those of a normal distribution.
Conversely, if :

$$p\text{-value} > 0.05 \tag{2.17}$$

we do not reject H_0. This means that the test does not provide sufficient evidence to reject the normality hypothesis.

p-value of the Jarque–Bera test Under the null hypothesis of normality, and asymptotically,

$$\boxed{JB \overset{a}{\sim} \chi^2(2)}$$

that is, the Jarque–Bera statistic asymptotically follows a χ^2 distribution with two degrees of freedom. The p-value then corresponds to the probability, under the null hypothesis, of observing a test statistic at least as large as the one obtained from the sample :

$$P\left(\chi_2^2 \geq JB_{\text{obs}}\right)$$

or, equivalently, using the cumulative distribution function $F_{\chi_2^2}$ of the χ^2 distribution with two degrees of freedom :

$$1 - F_{\chi_2^2}(JB_{\text{obs}})$$

In Python, the test can be performed using :

```
jb_stat, p_value = stats.jarque_bera(log_rets_filtered['BTC-USD'])
```

```python
import yfinance as yf
import pandas as pd
import numpy as np
import scipy.stats as stats

# Data ingestion for a highly volatile asset (e.g., Bitcoin)
ticker = 'BTC-USD'
prices = yf.download(ticker, start="2021-01-01", end="2026-01-01")['Adj Close'].ffill()
log_returns = np.log(prices / prices.shift(1)).dropna()

# Calculation of distribution moments via scipy.stats
skewness_val = stats.skew(log_returns)
kurtosis_ex = stats.kurtosis(log_returns, fisher=True)

# Execution of the Jarque-Bera test
jb_stat, p_value = stats.jarque_bera(log_returns)

print(f"--- Statistical Analysis of {ticker} ---")
print(f"Skewness : {skewness_val:.4f} (Expected for a normal distribution: 0)")
print(f"Excess Kurtosis    : {kurtosis_ex:.4f}  (Expected for a normal distribution: 0)")
print(f"Jarque-Bera Statistic : {jb_stat:.2f}")
print(f"Test p-value       : {p_value:.6f}")

if p_value < 0.05:
    print("Conclusion: We reject H0. The distribution is non-Gaussian.")
else:
    print("Conclusion: We cannot reject H0.")
```

Listing 2.1 – Normality Test and Higher-Order Moment Analysis

2.4 The Shapiro–Wilk Normality Test

The Shapiro–Wilk test is another statistical test used to determine whether a sample is compatible with a normal distribution.
In the financial context, this test can be used to assess whether the observed returns of an asset or portfolio can reasonably be considered normally distributed.

Test hypotheses Consider a sample of returns :

$$X_1, X_2, \ldots, X_n.$$

The Shapiro–Wilk test is based on the following hypotheses :

$$\boxed{\begin{array}{l} H_0 : X \sim \mathcal{N}(\mu, \sigma^2), \\ H_1 : X \text{ does not follow a normal distribution.} \end{array}}$$

The null hypothesis H_0 therefore means that the observations are compatible with a normal distribution. The objective of the test is to determine whether the data provide sufficient evidence to reject this hypothesis.

General principle of the test The idea behind the Shapiro–Wilk test is to compare the observations in the sample with what we would expect if the observations actually came from a normal distribution.
The first step consists of ordering the observations in ascending order :

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}.$$

The $X_{(i)}$ are called the **order statistics**. For example, for the sample :

$$\{0.03, -0.01, 0.02, 0.05, -0.02\},$$

we obtain :

$$X_{(1)} = -0.02, \quad X_{(2)} = -0.01, \quad X_{(3)} = 0.02, \quad X_{(4)} = 0.03, \quad X_{(5)} = 0.05.$$

The test then compares this ordered structure with what would be observed in a sample drawn from a normal distribution.

The Shapiro–Wilk statistic The test statistic is given by :

$$\boxed{W = \frac{\left(\sum_{i=1}^{n} a_i X_{(i)}\right)^2}{\sum_{i=1}^{n} (X_i - \bar{X})^2}}$$

where :

- $X_{(i)}$ denotes the i-th observation after ordering ;
- $\bar{X}$ is the sample mean ;
- a_i are the coefficients of the Shapiro–Wilk test. They are chosen to efficiently measure the deviation between the observed distribution and a normal distribution.
- n is the sample size.

The statistic W generally lies between 0 and 1.

- A value of W close to 1 indicates that the structure of the data is compatible with what is expected under a normal distribution.
- Conversely, a lower value of W indicates a greater deviation from normality.
- However, the final decision of the test is generally based on the **p-value**, rather than simply on whether W is close to 1.

p-value of the Shapiro–Wilk test Once the observed statistic W_{obs} has been calculated, the p-value corresponds to the probability, under the null hypothesis, of obtaining a value of W at least as small as the one observed :

$$P\left(W \leq W_{\text{obs}} \mid H_0\right)$$

The direction of the inequality is important. In the Shapiro–Wilk test, values of W close to 1 are associated with data that are more consistent with a normal distribution, whereas low values of W indicate a deviation from normality. Unlike the Jarque–Bera test, the statistic W does not follow a simple standard distribution such as a normal distribution or a χ^2 distribution. The distribution of W under the null hypothesis depends notably on the sample size and is mathematically complex because it does not have a simple analytical form. Therefore, the p-value is obtained using numerical approximations implemented in statistical software.

Where do the coefficients a_i come from ? The coefficients a_i are the most technical part of the test. The idea is to construct coefficients that allow us to best compare the ordered observations in the sample with what we would expect to observe under a normal distribution. To understand their origin, consider a sample of independent random variables following a standard normal distribution :

$$Z_1, Z_2, \ldots, Z_n \sim \mathcal{N}(0,1).$$

These observations are also ordered :

$$Z_{(1)} \leq Z_{(2)} \leq \cdots \leq Z_{(n)}.$$

We then consider the expected values of the order statistics :

$$m_i = E[Z_{(i)}].$$

The vector of expected values is therefore :

$$m = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{pmatrix}.$$

This vector represents the average positions that the ordered observations should have if the data came from a standard normal distribution. For example : $m_1 < 0,\ m_n > 0$. Moreover, due to the symmetry of the normal distribution : $m_i = -m_{n+1-i}$.

The covariance matrix of the order statistics The order statistics $Z_{(1)}, Z_{(2)}, \ldots, Z_{(n)}$ are not independent. Therefore, we define their covariance matrix as :

$$V = \text{Cov} \begin{pmatrix} Z_{(1)} \\ Z_{(2)} \\ \vdots \\ Z_{(n)} \end{pmatrix}.$$

In other words : $V_{ij} = \text{Cov}\left(Z_{(i)}, Z_{(j)}\right)$.
Based on the vector m and the matrix V, the coefficients a_i are constructed in order to assign appropriate weights to the different ordered observations. A theoretical expression, depending on the convention used for the vectors, is :

$$a = \frac{m^\top V^{-1}}{\sqrt{m^\top V^{-1} V^{-1} m}}.$$

This construction determines the weights that make the comparison between the observations and the structure expected under normality as effective as possible.

Why are the a_i not calculated by hand ? In practice, the coefficients a_i are generally not calculated directly by hand. Indeed, one would first need to determine $E[Z_{(i)}]$ as well as the covariances $\mathrm{Cov}\left(Z_{(i)}, Z_{(j)}\right)$. These calculations quickly become complex as n increases.
Statistical software therefore uses numerical methods and specific approximations to obtain the coefficients required for the test.
For example, with Python and the `scipy` library, we can simply use : **scipy.stats.shapiro**().
The software automatically calculates the statistic W as well as the p-value.
Therefore, it is mainly important to understand the role of the coefficients a_i, rather than trying to calculate them manually. We can summarize this as follows :
The a_i weight the ordered observations in order to compare their structure with that expected under a normal distribution.
Intuitively :

$$W = \frac{\text{structure compatible with normality}}{\text{total variability}}.$$

— When the data are close to what is expected under normality, W is close to 1.
— When the data exhibit a substantial deviation from normality, W tends to decrease.

Statistical decision Once the statistic W has been calculated, the test provides a p-value. For a significance level of : $\alpha = 5\% = 0.05$, we apply the following rule :

$$\boxed{\begin{cases} p\text{-value} < 0.05 & \Rightarrow \text{reject } H_0, \\ p\text{-value} \geq 0.05 & \Rightarrow \text{do not reject } H_0. \end{cases}}$$

Thus, $p < 0.05$ means that the data provide sufficient evidence to conclude that they are not compatible with a normal distribution at the 5% significance level.
Conversely, $p \geq 0.05$ means that there is not sufficient evidence to reject normality.
Important : failing to reject H_0 does not mean that we have proved that the data are normally distributed. The correct formulation is :

"We do not reject the normality hypothesis."

and not :

"The data are normally distributed."

Python implementation The `scipy.stats` library allows us to directly perform the Shapiro–Wilk test.

Shapiro–Wilk test

```python
import numpy as np
from scipy.stats import shapiro

# Example of returns
returns = np.array([
    0.012, -0.008, 0.005, 0.003, -0.011,
    0.007, -0.004, 0.009, -0.002, 0.006
])

# Shapiro-Wilk test
W, p_value = shapiro(returns)

print("Shapiro-Wilk statistic:", W)
print("p-value:", p_value)

# Significance level
alpha = 0.05

if p_value < alpha:
    print("Reject H0: the returns are not normally distributed.")
else:
    print("Do not reject H0: the returns are consistent with normality.")
```

The result contains two main pieces of information :
— W : the test statistic ;
— the p-value : used to make the statistical decision.

Comparison between Shapiro–Wilk and Jarque–Bera Both tests have the same null hypothesis :

$$H_0 : \quad X \sim \mathcal{N}(\mu, \sigma^2).$$

However, they do not test normality in exactly the same way.

	Jarque–Bera	**Shapiro–Wilk**
Null hypothesis	The data follow a normal distribution	The data follow a normal distribution
Statistic	JB	W
Main idea	Uses skewness and kurtosis	Compares the structure of the ordered observations with that expected under normality
Characteristic value	$S = 0,\ K = 3$ under normality	W close to 1 under normality
Decision	Based on the p-value	Based on the p-value
If $p < 0.05$	Reject normality	Reject normality

TABLE 2.1 – Comparison between the Jarque–Bera and Shapiro–Wilk tests.

Joint interpretation of the two tests It can be useful to apply both tests to the same returns. If $p_{\mathrm{SW}} > 0.05$ and $p_{\mathrm{JB}} > 0.05$, neither test provides sufficient evidence to reject the normality hypothesis at the 5% significance level.
We can therefore conclude :

Normality is not rejected by either test at the 5% significance level.

Conversely, if $p_{\mathrm{SW}} < 0.05$ and $p_{\mathrm{JB}} < 0.05$, both tests reject the normality hypothesis. It is then reasonable to conclude that the returns exhibit a statistically significant deviation from a normal distribution.

When the two tests lead to different conclusions It is possible that the two tests do not lead to the same conclusion. For example, $p_{\mathrm{SW}} < 0.05$ but $p_{\mathrm{JB}} > 0.05$. This means that Shapiro–Wilk rejects normality whereas Jarque–Bera does not reject it.
This difference is not necessarily contradictory because the two tests use different information :

Shapiro–Wilk $\longrightarrow$ overall structure of the ordered observations

whereas :

Jarque–Bera $\longrightarrow$ skewness + kurtosis.

A difference in conclusions can therefore occur when the deviation from normality comes from a particular characteristic of the distribution, for example :

— substantial skewness ;
— heavier tails than those of a normal distribution ;
— extreme observations ;
— another form of deviation from normality.

$\Rightarrow$ The two tests are therefore complementary rather than competing.

Remark For financial returns, normality is an important assumption in many classical models. However, financial returns frequently exhibit skewness, kurtosis greater than 3, and fat tails. It is therefore particularly useful to combine several tools in order to obtain a more complete view of the distribution of returns.

2.5 Regulatory Risk Measures : VaR and CVaR

In financial risk management, knowing the average return of a portfolio is not sufficient : it is also essential to quantify the magnitude of potential losses, particularly under adverse and extreme scenarios. **Value-at-Risk (VaR)** and **Conditional Value-at-Risk (CVaR)**, also known as *Expected Shortfall (ES)*, are two fundamental measures used to quantify this risk.
VaR answers the following question :

> *What level of loss should the portfolio exceed only with a given probability, over a given horizon ?*

Concrete Example. Suppose a portfolio has a **95% VaR over a one-day horizon of** $50,000$.
The term "1 day" means that we study the portfolio's loss over a single day. The 95% confidence level means that we consider the quantile corresponding to the worst 5% of scenarios.
Mathematically, if $L_{t,1j}$ denotes the portfolio loss over one day, then :

$$\boxed{\mathbb{P}(L_{t,1j} > 50,000) = 5\%} \tag{2.18}$$

or equivalently :

$$\boxed{\mathbb{P}(L_{t,1j} \leq 50,000) = 95\%} \tag{2.19}$$

In other words, for a given day, the model estimates a 5% probability that the loss exceeds $50,000$. It is important to distinguish the different elements of this definition :

$$\boxed{\underbrace{1 \text{ day}}_{\text{horizon}} \qquad \underbrace{95\%}_{\text{confidence level}} \qquad \underbrace{5\%}_{\text{exceedance probability}}} \tag{2.20}$$

VaR therefore only provides a **loss threshold**. It does not specify the magnitude of the loss once this threshold is exceeded. This is precisely the role of **CVaR**. CVaR measures the **average loss in scenarios where VaR is exceeded**. It therefore answers the following question :

When the loss exceeds the VaR threshold, what is the average magnitude of that loss ?

For example, if the one-day 95% VaR is $50,000$ and the one-day 95% CVaR is $75,000$, this means that among the scenarios belonging to the worst 5% loss tail, the average loss is $75,000$.
CVaR therefore focuses on the portion of the distribution beyond the VaR threshold. Thus, the two measures provide complementary information :

$$\boxed{\text{VaR} \longrightarrow \text{threshold separating ordinary scenarios from extreme scenarios}} \tag{2.21}$$

$$\boxed{\text{CVaR} \longrightarrow \text{average loss in extreme scenarios beyond VaR}} \tag{2.22}$$

This distinction is particularly important in quantitative finance. Two portfolios may have the same VaR while having very different CVaRs. The portfolio with the higher CVaR is then exposed to potentially larger extreme losses.

Definition 2.5.1 *In general, consider a random variable L representing the **loss** of a portfolio over a given time horizon. For a confidence level $\alpha \in (0,1)$,*

— *the **Value-at-Risk** is defined as the α-quantile of the loss distribution :*

$$P(L > VaR_\alpha) = \alpha \tag{2.23}$$

— **Expected Shortfall (CVaR) :** *which measures the expected loss (absolute value), conditional on the loss exceeding the VaR threshold, is defined by*

$$CVaR_\alpha = E[L \mid L > VaR_\alpha] \tag{2.24}$$

Python Toolbox for Risk Calculation : To implement our three risk models (Historical, Parametric, Cornish-Fisher), we use the following key functions :

— `np.percentile(df, q)` : Extracts the q-th percentile of the distribution. For a historical VaR with a risk threshold $\alpha = 5\%$, we seek to isolate the breakpoint corresponding to the worst 5% of returns in the time series.

— `stats.norm.ppf(q)` : *Percent Point Function* (or quantile function). It is the inverse of the cumulative distribution function of the normal distribution. For $\alpha = 0.05$, it returns the standardized score $z_\alpha \approx -1.6448$.

— `df[df <= threshold].mean()` : Uses Pandas Boolean masking to isolate only returns less than or equal to the VaR quantile, and then computes their average. This is the direct implementation of the conditional expectation of historical Expected Shortfall.

```
# Definition of risk parameters
alpha = 0.05 # 5% threshold (95% confidence)
capital = 1000000 # $1,000,000 portfolio

# Calculation of empirical parameters
mu = log_returns.mean()
sigma = log_returns.std()

# --- METHOD 1: HISTORICAL (NON-PARAMETRIC) ---
# Extraction of the 5th empirical percentile
var_hist_ret = np.percentile(log_returns, alpha * 100)
var_historical = -var_hist_ret * capital

# Historical CVaR: average of returns below the VaR
cvar_hist_ret = log_returns[log_returns <= var_hist_ret].mean()
cvar_historical = -cvar_hist_ret * capital

# --- METHOD 2: GAUSSIAN PARAMETRIC ---
# Retrieval of the classical z-score via stats.norm.ppf
z_alpha = stats.norm.ppf(alpha)
var_parametric = -(mu + z_alpha * sigma) * capital

# Gaussian CVaR:
# Theoretical average of returns located in the left tail
# For alpha = 0.05, phi(z_alpha) / alpha corresponds to the expectation
# of a standard normal variable conditioned on Z <= z_alpha
cvar_parametric_ret = -(mu - sigma * stats.norm.pdf(z_alpha) / alpha)
cvar_parametric = cvar_parametric_ret * capital

# --- METHOD 3: CORNISH-FISHER (NON-GAUSSIAN ADJUSTMENT) ---
S = stats.skew(log_returns)
K = stats.kurtosis(log_returns, fisher=True)

# Polynomial extension of the z_alpha
z_cf = (z_alpha +
        (z_alpha**2 - 1) * S / 6 +
        (z_alpha**3 - 3 * z_alpha) * K / 24 -
        (2 * z_alpha**3 - 5 * z_alpha) * (S**2) / 36)

var_cornish_fisher = -(mu + z_cf * sigma) * capital

# Cornish-Fisher CVaR:
# Cornish-Fisher directly provides an approximation of the quantile.
# To obtain an approximation of CVaR, we integrate the Cornish-Fisher
# quantiles over the entire left tail of the distribution.
u_values = np.linspace(0.0001, alpha, 1000)

z_values = stats.norm.ppf(u_values)

z_cf_values = (z_values +
               (z_values**2 - 1) * S / 6 +
               (z_values**3 - 3 * z_values) * K / 24 -
               (2 * z_values**3 - 5 * z_values) * (S**2) / 36)
```

```
54
55 q_cf_values = mu + sigma * z_cf_values
56
57 # Numerical integration of the left-tail quantiles
58 cvar_cf_ret = np.trapezoid(q_cf_values, u_values) / alpha
59 cvar_cornish_fisher = -cvar_cf_ret * capital
60
61 print("\n--- Risk Analysis (1-Day Horizon, $1M Capital) ---")
62 print(f"Historical VaR (95%)          : {var_historical:.2f} $")
63 print(f"Historical CVaR (95%)         : {cvar_historical:.2f} $")
64 print(f"Gaussian VaR (95%)            : {var_parametric:.2f} $")
65 print(f"Gaussian CVaR (95%)           : {cvar_parametric:.2f} $")
66 print(f"Cornish-Fisher VaR (95%)      : {var_cornish_fisher:.2f} $")
67 print(f"Cornish-Fisher CVaR (95%)     : {cvar_cornish_fisher:.2f} $")
```

Listing 2.2 – Comparative Implementation of VaR and CVaR

Mathematical Origin of the Cornish-Fisher Expansion

The **Cornish-Fisher** formula is derived from the **Edgeworth expansion**, which approximates the cumulative distribution function of a non-normal distribution using the normal distribution. We seek the quantile q_α satisfying :

$$F(q_\alpha) = \alpha, \text{where F is the cumulative distribution function of a standard normal distribution}$$

We then express this quantile as a correction of the normal quantile z_α :

$$q_\alpha = z_\alpha + \delta,$$

where δ represents a correction related to the non-normal characteristics of the distribution.

By expanding the Edgeworth expansion around z_α and then approximately inverting the relationship, we obtain the Cornish-Fisher expansion :

$$\boxed{q_\alpha \approx z_\alpha + \frac{S}{6}(z_\alpha^2 - 1) + \frac{K}{24}(z_\alpha^3 - 3z_\alpha)\frac{S^2}{36}(2z_\alpha^3 - 5z_\alpha)} \tag{2.25}$$

where :

$$z_\alpha = \text{normal distribution quantile}, \qquad S = \text{skewness}, \qquad K = \text{excess kurtosis}. \tag{2.26}$$

The formula therefore corrects the normal quantile z_α to account for the **asymmetry** of the distribution, represented by S, and the **thickness of its tails**, represented by K.
When the distribution is normal :

$$S = 0 \qquad \text{and} \qquad K = 0. \tag{2.27}$$

The formula then reduces to :

$$\boxed{q_\alpha = z_\alpha} \tag{2.28}$$

The terms $z_\alpha^2 - 1$ and $z_\alpha^3 - 3z_\alpha$ naturally appear in the Edgeworth expansion and are related to the **Hermite polynomials** associated with the normal distribution.

2.5.1 Mathematical Interpretation of Cornish-Fisher CVaR

The important difference between VaR and CVaR is that the Cornish-Fisher formula directly provides an approximation of a **quantile**, and therefore of VaR, but not directly of CVaR.
The Cornish-Fisher VaR is approximately given by :

$$\boxed{\text{VaR}_\alpha^{CF} \approx -q_\alpha^{CF}} \tag{2.29}$$

where q_α^{CF} represents the return quantile obtained using the Cornish-Fisher expansion.
CVaR, however, requires considering all quantiles located in the left tail of the distribution. It corresponds to the average loss associated with the most unfavorable scenarios.
For a variable representing returns, CVaR can therefore be approximated by :

$$\boxed{\text{CVaR}_\alpha^{CF} \approx -\frac{1}{\alpha}\int_0^\alpha q_u^{CF}\,du} \tag{2.30}$$

where q_u^{CF} denotes the Cornish-Fisher quantile corresponding to the probability level u.
In our example, with $\alpha = 0.05$, we therefore consider the quantiles between 0 and 5% :

$$\text{CVaR}_{0.05}^{CF} \approx -\frac{1}{0.05}\int_0^{0.05} q_u^{CF}\,du. \tag{2.31}$$

The idea therefore differs from VaR. VaR uses a single quantile :

$$\text{VaR}_\alpha^{CF} \approx -q_\alpha^{CF}, \tag{2.32}$$

whereas CVaR uses all the quantiles in the tail :

$$\text{CVaR}_\alpha^{CF} \approx -\frac{1}{\alpha}\int_0^\alpha q_u^{CF}\,du. \tag{2.33}$$

In Python, the integral is approximated numerically by constructing a grid of probabilities :

$$u_1, u_2, \ldots, u_M \in [0, \alpha]. \tag{2.34}$$

We then calculate the corresponding Cornish-Fisher quantile for each value u_i :

$$q_{u_i}^{CF} = \mu + \sigma z_{CF}(u_i). \tag{2.35}$$

Finally, the integral is approximated numerically using the trapezoidal rule :

$$\int_0^\alpha q_u^{CF}\,du \approx \text{Trapz}\left(q_{u_1}^{CF}, \ldots, q_{u_M}^{CF}\right). \tag{2.36}$$

This provides an approximation of CVaR that accounts for the skewness and excess kurtosis of returns through the Cornish-Fisher expansion.
Thus :

$$\boxed{\text{Cornish-Fisher} \rightarrow \text{corrected quantiles} \rightarrow \text{tail integration} \rightarrow \text{CVaR}} \tag{2.37}$$