Market Risk Project

Author:

Ihab Machmoum

ESILV

December 31, 2024

Contents

1 Introduction and Essential Libraries

The aim of this project is to apply and implement concepts studied during lectures and further explored in the directed working sessions.

First, we're going to implement some risk measures such as non-parametric Historical VaR (case of Question A for Natixis stocks) and Monte Carlo VaR (case of Question B for a call option on the Natixis stock). We will then try to show when and why we use the Historical and the Monte Carlo VaR as a comparison between the two questions.

In Question C, we will study the distribution of extreme events and show that it should be close to a known parametrized distribution, which will yield the VaR calculation (EVT VaR). We will then take care of the serial dependencies using the extremal index, which provides more accuracy.

In Question D, we explore Impact Models, such as Almgren and Chriss, and discuss their significance in understanding market dynamics.

Finally, in Question E, we focus on multi-frequency aspects of risk and their implications in financial analysis.

The primary goal is to estimate various risk measures using real-world financial data.

The code used to answer each question is provided in the "Appendix" section at the end of the report. Python has been chosen as the programming language for this project due to its extensive ecosystem of libraries that simplify financial analysis.

We primarily rely on the following Python libraries:

- **pandas:** Used for data preprocessing, manipulation, and cleaning. It allows efficient handling of large datasets.
- **numpy:** Essential for numerical computations, enabling the implementation of mathematical models and algorithms.
- **scipy.stats:** Provides statistical functions crucial for tasks like computing distributions and quantiles.
- **matplotlib.pyplot:** Extensively used for visualization, creating insightful charts and graphs to better understand the data and results.

By leveraging these tools, we aim to obtain accurate estimations of risk measures and validate the methodologies studied throughout the course.

2 Question A

a From the time series of the daily prices of the stock Natixis **between January 2015 and December 2016**, provided with TD1, estimate a historical VaR on price returns at a one-day horizon for a given probability level (this probability is a parameter which must be changed easily). You must base your VaR on a **non-parametric distribution (logistic Kernel)**, that is K is the derivative of the logistic function $x \mapsto \frac{1}{1+e^{-x}}$.

b Which proportion of price returns **between January 2017 and December 2018** exceed the VaR threshold defined in the previous question? Do you validate the choice of this non-parametric VaR?

Answer to Part a

We first compute the return using `pct.change(1)`, which calculates the percentage change between the current and a prior element. Mathematically, it is expressed as:

$$\text{Return} = \frac{P_j - P_i}{P_i}, \quad \text{where } i < j.$$

The second step is cleaning data. This includes handling missing values (e.g., removing `NaN` entries) to ensure the dataset is ready to be filtered. After cleaning, we filter the returns for the period **January 2015 to December 2016**.

In this question, our goal is to calculate the Value at Risk (VaR) of Natixis stock returns. Since the Natixis stock has no maturity, its probability distribution is stationary. Meaning that future returns will follow the same distribution as past returns.

To estimate VaR, we use the Historical VaR method based on a kernel density estimation (KDE), a non-parametric method that is better than parametric approaches. Non-parametric methods do not assume any specific form for the data distribution, making them more robust in capturing the characteristics of real-world financial data.

Estimating the whole distribution

Before we start our VaR calculation we recall some theorotical notions An estimator $\hat{f}(x)$ of a density should satisfy the following properties:

- $\hat{f}(x) \geq 0 \quad \forall x \in R$,
- $\int_{-\infty}^{+\infty} \hat{f}(x)\, dx = 1$.

From a series of observed price returns $X_1, \ldots, X_n$, the probability density can be estimated by:

A kernel density, where the kernel $K(x)$ is a density function derived from its cumulative distribution function (CDF) $\mathcal{K}(x)$. In our case, the kernel is based on the derivative of the logistic function.

Recall that the logistic function is given by:

$$\mathcal{K}(x) = \frac{1}{1+e^{-x}},$$

Taking the derivative of the logistic function with respect to x, we obtain the kernel:

$$K(x) = \frac{d}{dx}\mathcal{K}(x) = \frac{e^{-x}}{(1+e^{-x})^2}.$$

This kernel is positive for all x, as both the numerator e^{-x} and the denominator $(1+e^{-x})^2$ are strictly positive. Therefore, it satisfies the requirements for a valid kernel density function.

The derivative of the logistic function ensures smoothness and robustness, making it a suitable choice for non-parametric density estimation in our application.

with a smoothing parameter $h > 0$ controlling smoothness and robustness. The estimated density $\hat{f}(x)$ is given by:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right),$$

and the cumulative distribution function is:

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{K}\left(\frac{x - X_i}{h}\right).$$

This non-parametric method is flexible and does not rely on assumptions about the data distribution, making it robust for estimating the density of returns. However, the choice of the smoothing parameter h affects the resolution, with fixed-size h potentially causing spurious effects at the tails.

That's why in our implementation (see Appendix), we use a dynamically calculated bandwidth h based on Silverman's rule of thumb:

$$h = 1.06 \cdot \sigma \cdot n^{-1/5},$$

where σ represents the standard deviation of the data and n is the number of data points.

Methods used for VaR Calculation

To calculate the Value at Risk (VaR), we identify the empirical quantile of the estimated distribution at a given confidence level (e.g., 95%). The VaR is the maximum potential loss for a given portfolio value over a specified time horizon

In our implementation, we used two methods to compute the empirical quantile based on the logistic kernel density estimation that we had before.

Method 1: Dichotomy Search

- Initialize two bounds, a and b.
- Compute $m = \frac{a+b}{2}$ and evaluate the cumulative distribution function (CDF) at m, which is our case the logistic function.
- Adjust a or b based on the difference between the logistic function value and the target confidence level.
- Repeat until the desired precision (ϵ) is reached.

This method provides the empirical quantile of our logistic function.

Method 2: Direct Search In the second method, we directly stock our estimated CDF (kce logistic values in the Apprendix) traverse these estimated values one by one and check. In fact we :

- Iterate through the CDF values and find the first point where the cumulative probability exceeds $1 - \alpha$.
- Identify the corresponding return value as the VaR.

This approach is computationally simpler and faster, especially for smaller datasets.

Results

By applying these methods, we obtained the following results for the Natixis stock returns:

- Using **Method 1 (Dichotomy Search)**, the VaR at a 95% confidence level was calculated as -4.333%.
- Using **Method 2 (Direct Search)**, the VaR at a 95% confidence level was calculated as -4.33%.

Applying this method, we finally obtain a VaR of -4.33% at 95% confidence level. This indicates that, for a portfolio valued at €1000, there is a 5% probability of incurring a loss greater than €43.33.

Answer to Part b

To evaluate the effectiveness of the non-parametric Value at Risk (VaR) estimated in the previous question, we performed a backtesting procedure. This analysis focused on the data from January 2017 to December 2018. The goal was to answer the question:

> *Which proportion of price returns between January 2017 and December 2018 exceed the VaR threshold defined in the previous question? Do you validate the choice of this non-parametric VaR?*

Steps in the Analysis

1. We filtered the data to include only returns within the specific time frame of January 2017 to December 2018.
2. We compared these returns against the VaR threshold obtained previously (-4.33%).
3. We calculated the proportion of returns that fall below this threshold (exceedances).
4. Finally, we validated the VaR model by checking if the observed exceedance proportion is less than or equal to the expected exceedance proportion, which is the complement of the confidence level ($1 - 0.95 = 0.05$ or 5%).

Results

The backtesting analysis yielded the following results:

- **Observed exceedance proportion:** 1.37%
- **Expected exceedance proportion:** 5%
- **Validation Result:** The validation check confirmed that the observed proportion is less than the expected proportion, We can conclude that the non-parametric VaR model is effective for the given confidence level.

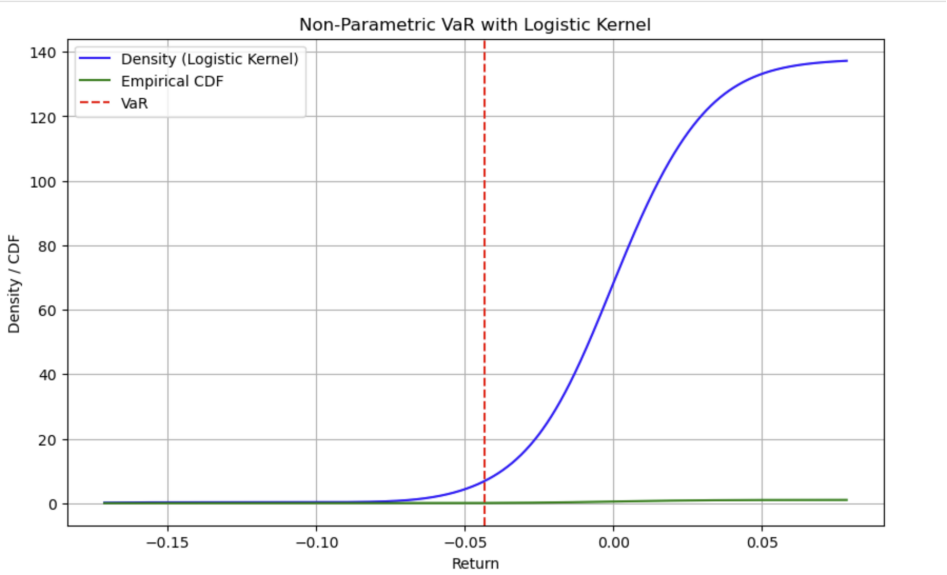

Figure 1: Non-Parametric VaR Validation with Logistic Kernel

3 Question B

We want to answer the question: calculate the VaR (on the arithmetic variation of price, at a one-day horizon) for a call option on the Natixis stock. You will implement a Monte-Carlo VaR since the call price is a non-linear function of the underlying price, that we are able to model thanks to historical data. Here is, in detail, how you must proceed:

- Estimate the parameters of a standard Brownian motion on the Natixis stock **between 2015 and 2018**, using an exponential weighting of the data.
- Simulate a number N (say $N = 1000$ or else, but justify your choice for this number) of prices of the stock in a one-day horizon (we are working at the last date of 2018).
- Transform each of these prices of underlying in prices of the corresponding call (say at the money, with one-month maturity and 0 risk-free rate and dividend).
- Pick the empirical quantile of these N call prices to build the VaR of the call.

Methodology

In this question we're trying to calculate the Value at Risk (VaR) for a call option on the Natixis stock, at a one-day horizon. Unlike in Question A, where the VaR was calculated based on the stationarity of the distribution of returns, in this case, we don't have a stationarity. We have to simulate the distribution of the underlying asset's price (stock prices) and then derive the derivative's price (which is the call price) to finally compute the empirical quantiles of these simulated prices.

The methodology to answer this question will be divided into three key parts:

1. **Simulate the distribution of the underlying asset:** Using the parameters of a standard Brownian motion on the Natixis stock (estimated using an exponential weighting of the data), we will generate a series of price trajectories for the underlying asset over a one-day horizon.
2. **Calculate the price of the derivative:** Transform each of the simulated prices of the underlying asset into prices of the corresponding call option (assuming one-month maturity, zero risk-free rate, and no dividends).
3. **Calculate the empirical quantiles of these simulated prices:** Use these transformed prices to determine the empirical quantile corresponding to the desired confidence level, thereby estimating the VaR.

In the next sections, we will delve deeper into each of these steps and provide the results obtained through this methodology.

3.1 Stimulating the Distribution of the Stock Price

The first step is to calculate the logarithmic returns of the Natixis stock prices to simulate the distribution of the stock price using a Brownian motion.

Calculating Logarithmic Returns Log-returns are calculated as follows:

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

where P_t is the price at time t and P_{t-1} is the price at the previous time step.

Exponential Smoothing of Log-Returns To give less weight to older data and emphasize recent observations, we apply an exponential smoothing technique seen in the practical working session:

$$\text{Returns\_lissés} = \text{Log\_Return} \times \lambda + \text{Log\_Return}_{\text{shifted}} \times (1 - \lambda),$$

where λ (e.g., 0.6 in our implementation) is the smoothing factor. This approach reduces the impact of older observations on the overall result, By doing this, we're more sure that more recent trends are taken into consideration.

Now that we have our log returns (Smoothed Returns), we will try to estimate the mean and the variance of the brownian motion in order to stimulate a number 1000 of

future stock prices starting from the last date of 2018. (See Apprendix). In fact we know that :

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t,$$

where:

- S_t: Stock price at time t,
- μ: Drift term (mean rate of return),
- σ: Volatility (standard deviation of the returns),
- W_t: Standard Brownian motion.

The solution to this SDE is given by:

$$S_t = S_0 \cdot \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t\right).$$

Now, consider the stock price at two consecutive time points, $t-1$ and t. Using the same relationship:

$$S_{t-1} = S_0 \cdot \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) (t-1) + \sigma W_{t-1}\right),$$

and

$$S_t = S_{t-1} \cdot \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) + \sigma (W_t - W_{t-1})\right).$$

Taking the natural logarithm of the ratio $\frac{S_t}{S_{t-1}}$, we get:

$$\ln\left(\frac{S_t}{S_{t-1}}\right) = \ln(S_t) - \ln(S_{t-1}) = \left(\mu - \frac{\sigma^2}{2}\right) + \sigma \Delta W_t,$$

where $\Delta W_t = W_t - W_{t-1} \sim \mathcal{N}(0,1)$, due to the properties of Brownian motion.

Taking the expectation of both sides:

$$E\left[\ln\left(\frac{S_t}{S_{t-1}}\right)\right] = E\left[\left(\mu - \frac{\sigma^2}{2}\right)\right] + E\left[\sigma \Delta W_t\right].$$

Since $E[\Delta W_t] = 0$, we find:

$$E\left[\ln\left(\frac{S_t}{S_{t-1}}\right)\right] = \mu - \frac{\sigma^2}{2}.$$

Thus, the expected logarithmic return $E[\ln(S_t/S_{t-1})]$ can be expressed as $\mu - \frac{\sigma^2}{2}$. This relationship allows us to calculate the drift term μ and volatility σ from historical logarithmic returns (log-return.mean() and log-return.std() see Apprendix) Using these parameters, we can simulate future stock prices using the standard Brownian motion, which will then be followed by estimating the price distribution of the derivative in subsequent steps.

Future Prices Using Brownian Motion Using the estimated parameters $\mu - \frac{\sigma^2}{2}$ and σ that we found before, we simulate future stock prices over a one-day horizon. In fact we:

1. Generate Brownian motion increments ΔW_t from a standard normal distribution.
2. Recursively compute future stock prices using:

$$S_t = S_{t-1} \cdot \exp\left(\mu - \frac{\sigma^2}{2} + \sigma \Delta W_t\right).$$

3. Organize the simulated prices in a DataFrame along with their corresponding future dates.

This process yields the simulated stock price paths required for subsequent derivative pricing and Value at Risk (VaR) calculations. The detailed implementation is shown in the code in the Apprendix.

3.2 Calculating the Price of the Call Option

Now that we have our simulated prices from the last date of 2018, we will calculate d_1 and d_2 using the following equations:

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + 0.5\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},$$
$$d_2 = d_1 - \sigma\sqrt{T-t},$$

where:

- S: Estimated price of the stock thatwe found before starting from last 2018.
- K: Strike price,
- r: Risk-free interest rate,
- σ: Volatility of the stock,
- $T - t$: Time to maturity.

In the code, we create a DataFrame called `future_df` that contains the simulated future estimated prices that we found from before. We then just compute the corresponding d1 and d2 for each price.

To calculate the price of a call option, we use the Black-Scholes formula:

$$C = S \cdot N(d_1) - K \cdot e^{-r(T-t)} \cdot N(d_2),$$

where $N(d_1)$ and $N(d_2)$ are the cumulative distribution functions of the standard normal distribution at d_1 and d_2, respectively. Note that in our implementation we import norm from scipy.stats library.

Implementation As shown in the code, the calculated d_1, d_2, and the resulting call prices are added as new columns to the `future_df` DataFrame. This allows us to easily analyze and extract the empirical quantiles required for the VaR computation.

For clarity, the following steps are carried out:

1. Calculate d_1 and d_2 for each row in `future_df`.
2. Use the Black-Scholes formula to compute the call price based on the simulated future prices.
3. Store the results in the DataFrame under the column `Call_Price`.

The results are presented in `future_df`, which now includes columns for d_1, d_2, and the calculated call prices. This will help us further in the calculation of the Value at Risk (VaR) for the call option.

3.3 Final Step: VaR Calculation

Now we can easily calculate the VaR using the empirical VaR, which is based on the empirical quantile of the simulated call prices.

To compute this, we:

1. Extract the simulated call prices from the `future_df` DataFrame.
2. Use the `empirical_var` function to compute the quantile at the specified confidence level (e.g., 95%).
3. Optionally, analyze the price changes (`Call_Price.diff()`) and compute the VaR for these changes as well.

Results From the code implementation:

- The VaR for the call prices at a 95% confidence level is approximately:

$$-1.329 \times 10^{-12}.$$

  Note that this value is extremely close to zero, indicating that, at this confidence level, there is virtually no significant loss expected in the call prices.

- The VaR for the changes in call prices (`Call_Price.diff()`) is approximately:

$$0.0601.$$

  This result reflects the expected maximum loss in the change of call prices over the specified time horizon.

Interpretation These results suggest that the risk associated with the simulated call prices is minimal at the given confidence level. However, the slight change in call prices (`Call_Price.diff()`) reflects some level of risk, which is quantified as a 6.01% potential maximum loss for the price changes. Let us consider a portfolio of call options with a total value of €1000. Using the computed VaR for the change in call prices:

$$\text{Maximum Loss} = \text{Portfolio Value} \times \text{VaR}.$$

Substituting the values:

$$\text{Maximum Loss} = 1000 \times 0.0601 = 60.10.$$

This means that, at a 95% confidence level, the portfolio can loose up to €60.10 due to changes in call prices.

Conclusion These results suggest that the risk associated with the simulated call prices is minimal at the given confidence level. However, when looking at the changes in call prices, the risk is more tangible, with a potential maximum loss of 6.01% or €60.10 for a €1000 portfolio. We conclude that we have to deal with both price levels and price changes when evaluating risk measures like VaR especially with assets that have a maturity.

4 Question C

4.1 Estimating the Shape Parameter with Pickands

The first step is to compute the daily returns from the dataframe provided. Extreme Value Theory (EVT) is much more accurate when it comes to tail behaviour and rare events. That's been said, we can have exreme gains and extreme losses. Here, we will divide our analysis in two parts, one with absolute extreme sorted losses and extreme sorted gains.

For the positive tail representing extreme gains, and the negative tail representing extreme losses, we calculate the shape parameters using the Pickands estimator where we take in our implementation k = log(n) as it goes to infinity but not as fast as n which satifies the condition from the Pickands estimator:

$$\text{Pickands Estimator} = \frac{1}{\log(2)} \log\left(\frac{X_{n-k+1:n} - X_{n-2k+1:n}}{X_{n-2k+1:n} - X_{n-4k+1:n}}\right)$$

The estimated shape parameters for the gains and losses are 0.7031 and -0.4822, respectively. Invoking the Fisher-Tippett Theorem, the positive shape parameter for gains implies a heavy-tailed distribution and the GEV is of Fréchet kind, reflecting the potential for unbounded extreme gains. Conversely, the negative shape parameter for losses suggests a distribution with a finite upper limit and the GEV is of a Weibull kind, indicating that losses, while potentially extreme, are not without bound; rather, they are contained within a certain threshold, beyond which the probability of occurrence is zero.

4.2 Taking care of the serial dependencies using the extremal index

The core EVT assumes that our returns are independent, which is not true in the majority of cases. That's where Leadbetter's theorem helps us in addressing these serial dependencies using the extremal index. In fact, if:

$$b_n^{-1}(M_n - a_n) \text{ converges to } G,$$

and there is a serial dependency, then the convergence will be toward G^θ instead of G, where θ is between 0 and 1 and θ is the extremal index.

The extremal index θ quantifies the degree of dependence between extreme events in a time series. A value of $\theta = 1$ indicates independence between events, while $\theta < 1$ signifies clustering of extremes, with lower values corresponding to stronger dependence.

To estimate θ, we employ the **de-clustering method**, which uses block-based or run-based techniques to account for serial dependencies in extreme events. For this implementation we chose to use block based method.

Block-Based De-Clustering In this method, we:

1. Partition the time series into $k = \lfloor n/b \rfloor$ blocks of size b. Here b is the gap which we chose carefully based on analyzing the most frequent gaps later in this document.

2. Count the number of blocks containing at least one extreme value above a threshold u, where u is chosen as the empirical quantile (e.g., 95%) of our returns (empirical VaR which we found equal to 0.031).

The block-based extremal index estimator is given by:

$$\hat{\theta}_n^B(u; b) = \frac{\sum_{i=1}^{k} 1\left(M_{(i-1)b,ib} > u\right)}{\sum_{i=1}^{kb} 1\left(X_i > u\right)},$$

where $M_{(i-1)b,ib}$ is the maximum of the observations in block i, and 1 is the indicator function that indicates if the return is above the empirical VaR.

Implementation in Python To compute θ, we implemented block-based method and run-based de-clustering method will not be used :

- **Block-Based De-Clustering:**
  - The time series was divided into blocks based on the observed gaps between exceedances.
  - The threshold u was set to the 95% empirical quantile.
  - Blocks containing at least one exceedance were identified, and the ratio was computed as shown in the formula.

In our case, the most frequent gap value was identified as 1 day (see the gap distribution figure), and the extremal index was calculated to be approximately 0.24, indicating moderate clustering of extreme events.

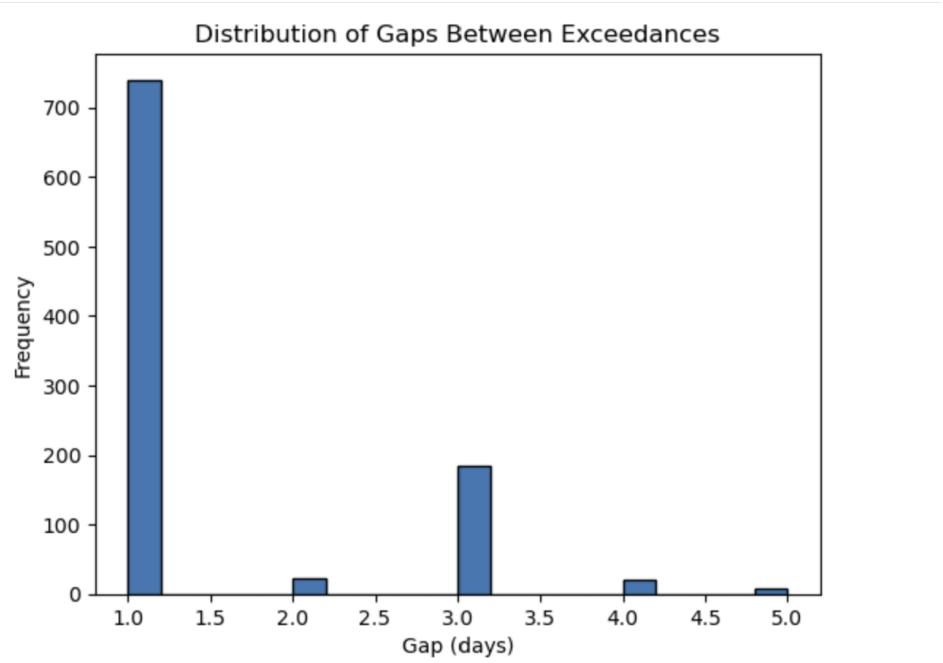

Figure 2: Distribution of Gaps Between Exceedances

By incorporating these methods, we have addressed the serial dependencies in the extreme events, providing a more accurate characterization of the risk associated with the extreme events. The EVT-based VaR can then be estimated using this extremal index.

5 Question D

Defining the model parameters

The Almgren and Chriss model relies on extracting key information from the provided data, including the volume of transactions, transaction prices, and the sign of each transaction (indicating buying or selling). Based on this data, the percentage change between consecutive transaction prices is calculated.

Annualizing volatility

A critical parameter in the Almgren and Chriss model is the asset's volatility, σ. This is estimated using the standard deviation of returns, adjusted to reflect trading hours per day (24) and trading days per year (252), yielding the annualized volatility. In this case, the estimated volatility is $\sigma = 10.7\%$.

γ Estimation

Another key parameter in the Almgren and Chriss model is γ. This parameter appears in the model's permanent impact function, expressed as:

$$g\left(\frac{nk}{\tau}\right) = \gamma\left(\frac{nk}{\tau}\right),$$

where:

- $g\left(\frac{nk}{\tau}\right)$ represents the permanent impact on price due to trading.
- γ is the proportionality constant indicating the magnitude of the permanent market impact.
- $\frac{nk}{\tau}$ is the rate of trading, with nk being the size of the trade and τ the trading interval.

We can see that there is a direct proportionality between the permanent impact and the rate of trading. To estimate γ, we perform a linear regression analysis. Here, the dependent variable is the change in the asset's price, calculated as $P(t+2) - P(t)$, while the independent variable is the rate of trading $\frac{nk}{\tau}$. The slope of the regression line provides an estimate of γ.

Linear Regression:

The model is given by:

$$P(t+2) - P(t) = \gamma \left(\frac{nk}{\tau}\right).$$

The estimated value of γ is 0.000502, with a mean squared error (MSE) of 0.00049 and a coefficient of determination (R^2) of 0.91. These results highlight a strong linear relationship between the signed volume and the price impact, as evidenced by the high R^2 value.

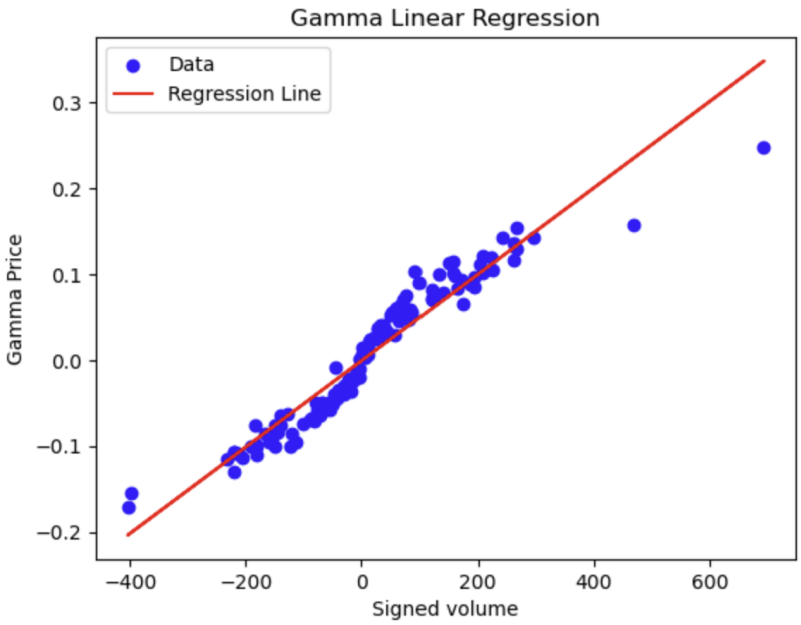

Figure 3: Linear Regression for Gamma Estimation

The positive slope of the regression line aligns with the theoretical expectations of the Almgren and Chriss model, affirming the positive relationship between the trade size and the price change. This reinforces the validity of our model.

Estimation of η

First Approach: Linear Regression The estimation of η is conducted using a linear regression model, defined by the following formula:

$$P(t+2) - P(t+1) = h\left(\frac{n_k}{\tau}\right) = \phi \cdot \text{sgn}(n_k) + \eta\left(\frac{n_k}{\tau}\right)$$

Here:

- ϕ represents the half spread.
- $\text{sgn}(n_k)$ is the sign of the trade.
- $\eta\left(\frac{n_k}{\tau}\right)$ accounts for the portion of the impact that scales with the rate of trading $\frac{n_k}{\tau}$.

Analyzing Figure 4, it becomes evident that the data exhibits non-linear behavior. Consequently, the linear regression model is not well-suited for this estimation. That's why we will try the multivariate regression.

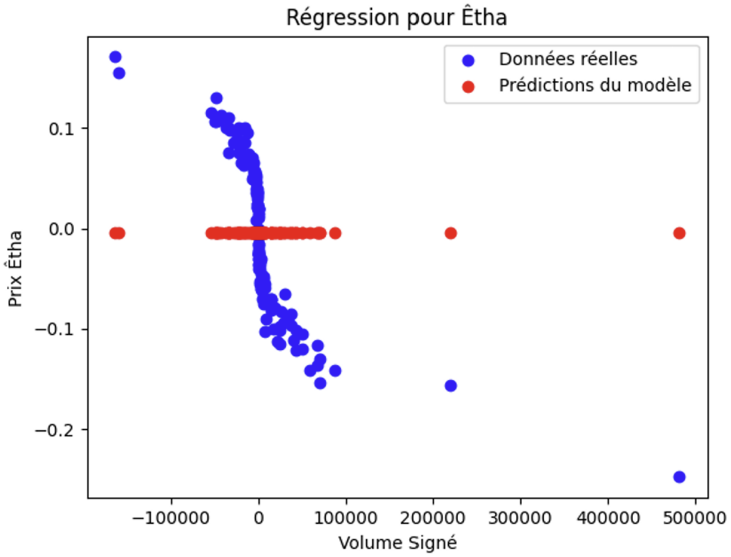

Figure 4: Linear Regression for Eta Estimation

Second approach: multivariate regression In an attempt to address the limitations of the linear regression approach, we explore a non-linear model by combining the quadratic cost from the first regression with the linear regression formula. This leads us to the following multivariate regression equation:

$$h\left(n_k^2 \cdot \text{Sign}(n_k)\right) = \eta\left(n_k^2 \cdot \text{Sign}(n_k)\right) = P(t+2) - P(t+1)$$

Here, $h(n_k^2 \cdot \text{Sign}(n_k))$ represents the price change due to the temporary market impact, and $n_k^2 \cdot \text{Sign}(n_k)$ is the squared signed volume of the trade.

Results of the regression analysis:

- **Estimated η:** $5.966990046987537 \times 10^{-8}$
- **Mean Squared Error (MSE):** 0.00052813
- **Coefficient of Determination (R^2):** 0.913

Conclusion: The substantial difference in the R^2 values and the visual divergence in the regression plots highlight potential shortcomings in the model's specification. While the γ regression appears well-specified, successfully capturing a significant proportion of the variance in price changes, the η regression suggests the need for additional factors to fully account for the temporary market impact.

The observed non-linear relationship in the first regression plot underscores the importance of exploring more sophisticated models. These models could include additional explanatory variables to better capture the nuances of market behavior.

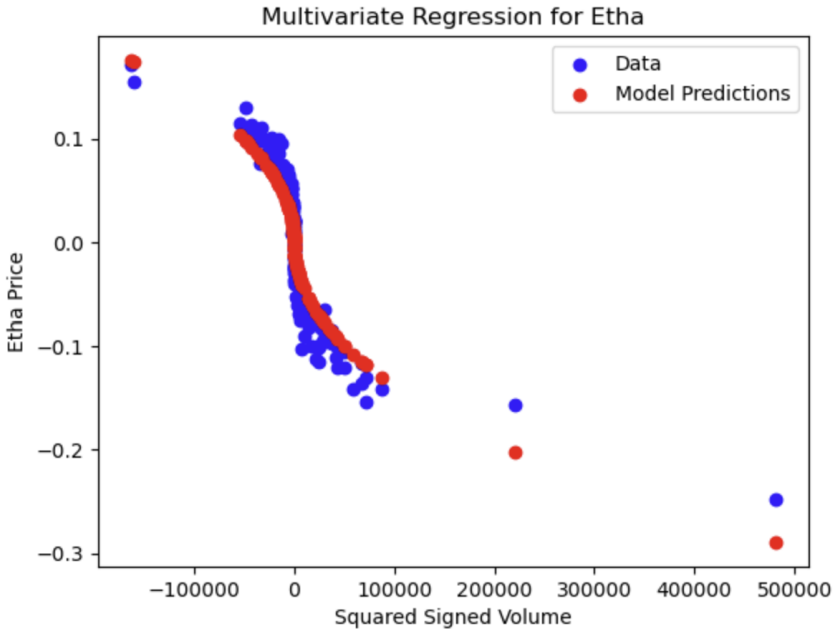

Figure 5: Multivariate Regression for Eta Estimation

What is the best liquidation strategy ?

Despite the anomalies observed in the specification of η within the Almgren and Chriss model, the framework suggests that the optimal liquidation trajectory is shaped by the trade-off between liquidity risk and market risk. This balance is influenced by the parameter λ, which represents the trader's level of risk aversion.

To explore this relationship, we analyze liquidation trajectories for various values of λ. Specifically, we consider λ values of 1×10^{-5}, 1×10^{-4}, and 1×10^{-3}. These trajectories are computed over a 24-hour period, with T denoting the fractional portion of the day.

The results highlight the influence of λ on liquidation strategies:

- A higher λ corresponds to a more aggressive liquidation strategy, where the trader prioritizes minimizing exposure to market risk, even at the cost of increased liquidity risk.
- A lower λ reflects a more gradual liquidation approach, prioritizing reduced liquidity risk at the expense of a higher market risk exposure.

This analysis demonstrates the critical role of λ in determining the pace and risk profile of optimal liquidation strategies.

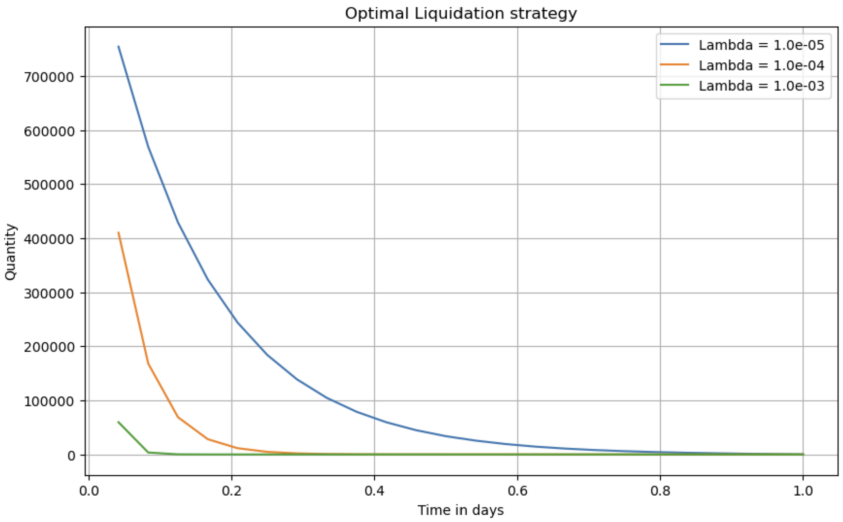

Figure 6: Risk aversion and Liquidation strategies

Question E

Multiresolution Correlation Matrices

To analyze the multiresolution correlation between the currency pairs **GBPEUR**, **SEKEUR**, and **CADEUR**, we use the **Haar wavelet transform**. This method is a more generalized Fourier transform. In fact, it allows for decomposition of the time series into components at varying resolutions. The key steps involved in this process are as follows:

1. **Calculate Returns:** Returns are computed as the percentage change of average prices for each currency pair.
2. **Haar Transform:** The returns are subjected to Haar wavelet transformation to generate coefficients for each resolution level.
3. **Correlation Matrices:** For each level of resolution, the correlation matrix is calculated using the transformed coefficients. These matrices provide insights into how correlations evolve across different time scales.

As an example, the correlation matrix at the finest resolution (level 9) is:

$$\begin{bmatrix} 1.00 & 0.03 & 0.24 \\ 0.03 & 1.00 & -0.05 \\ 0.24 & -0.05 & 1.00 \end{bmatrix}$$

This matrix shows the linear relationships between the currency pairs at this scale. The Epps effect can be observed because the correlation between currency pairs decreases as the time scale also decreases.

Hurst Exponent and Volatility Analysis

To estimate the **Hurst exponent** for each currency pair, the **absolute moments method** is applied:

1. Compute the **empirical absolute second moment** M_2 with spacing 1/N which is a consistent estimator of our real moment.
2. Compute the adjusted absolute second moment M_2' which the same as the first one but with a spacing 2/N.
3. The Hurst exponent is then then found by dividing the two absolute moments and applying the log function:

$$H = 0.5 \times \log_2 \left(\frac{M_2'}{M_2} \right)$$

The Hurst exponents for **GBPEUR**, **SEKEUR**, and **CADEUR** are found to be:

$$H_{GBPEUR} = 0.698, \quad H_{SEKEUR} = 0.65, \quad H_{CADEUR} = 0.697$$

The volatilities for each currency pair are scaled across different time frames using:

$$\text{Low-Frequency Volatility} = \text{High-Frequency Volatility} \times (2^j)^H$$

Where j represents the resolution level.

Portfolio Volatility

The portfolio volatility is calculated by constructing covariance matrices for each resolution level. The covariance matrix elements are computed as:

$$\text{Cov}_{i,k} = \text{Correlation}_{i,k} \times \text{Volatility}_i \times \text{Volatility}_k$$

Using these covariance matrices, the portfolio variance is computed assuming equal weights $W = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$. The portfolio volatility is then:

$$\text{Portfolio Volatility} = \sqrt{W^T \cdot \text{Covariance Matrix} \cdot W}$$

The annualized portfolio volatility, assuming 52 weeks in a year, is:

$$\text{Annualized Volatility} = \text{Weekly Scale Volatility} \times \sqrt{52}$$

Conclusion

This analysis highlights the dynamic relationships between currency pairs at varying time scales and provides a robust framework for estimating both individual and portfolio volatilities. The calculated Hurst exponents indicate persistent or mean-reverting behaviors, contributing to a nuanced understanding of market dynamics.

Appendix

Code Snippet 1: Question A

```
# Content from QuestionA.ipynb
#Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from scipy.stats import norm

#Returns and setting index equal to Date
df = pd.read_excel('Natixis_ihab.xlsx', header=None, names=['Date', 'Price'])
df.head()
df['Price'] = pd.to_numeric(df['Price'])
df['Return'] = df['Price'].pct_change(1)
df.set_index('Date', inplace = True)

# Here we're going to select data bztween January 2015 and December 2016
df1 = df.loc['2015-01-01':'2016-12-30', 'Return'].dropna()

#Step 1 : Calculating the logistic kernel
def logistic_kernel(l):
    return np.exp(-l) / (1 + np.exp(-l))**2

#Step 2 : Estimating the kernel density
def kernel_density_estimation(x, data, h):
    n = len(data)
    l = (x - data) / h #normalizing
    kde = np.sum(logistic_kernel(l)) / (n * h)
    return kde

# Caculating our parameters n and h
n = len(df1)
h = 1.06 * np.std(df1) * (n ** (-1/5))

range_of_x = np.linspace(min(df1), max(df1), num = 10000)

#Step 3 : Now we're going to group all the kde values gerated from the function
↪  above
kde_values_logistic = [kernel_density_estimation(x, df1, h) for x in range_of_x]

#We're going to reapeat the exact same steps for the cdf
#Step 1 : Calculating the logistic function
def cdf_logistic_kernel(l):
    return 1 / (1 + np.exp(-l))

#Step 2 : Estimating  the logistic kernel cumulative
def empirical_cdf(x, data, h):
    n = len(data)
    l = (x - data) / h
    kce = np.sum(cdf_logistic_kernel(l)) / n
    return kce

#Step 3 : Group all the kce values
kce_values_logistic = [empirical_cdf(x, df1, h) for x in range_of_x]
```

```python
# Method 1
# On utilise une dichotomie pour trouver la VaR
def quantile_empirique(fdr, alpha, a, b, epsilon, max_iter, h, returns):
    iter = 0
    alpha = 1 - alpha
    while b - a > epsilon and iter < max_iter:
        iter += 1
        m = (a + b) / 2
        F = fdr(m, returns, h)
        if abs(F - alpha) < epsilon:
            return -m
        elif F < alpha:
            a = m
        else :
            b = m
    return "Max iteration reached"

confidence_level = 0.95
non_param_VaR = quantile_empirique(empirical_cdf, alpha=0.95, a=-1, b=1,
↪  epsilon=0.0001, max_iter=1000000, h=h, returns=df1)
print(f"\nNon Parametric VaR at {confidence_level * 100}% confidence level:
↪  {non_param_VaR}")

# Method 2
confidence_level = 0.95
non_parametric_var_logistic = None

for i, cdf in enumerate(kce_values_logistic):
    if cdf >= (1 - confidence_level):
        non_parametric_var_logistic = range_of_x[i]
        break

non_parametric_var_logistic_message = (
    f"Non-Parametric_VaR (Logistic_Kernel) at {confidence_level * 100}% confidence
    ↪  level: "
    f"{non_parametric_var_logistic:.4f}"
)
print(non_parametric_var_logistic_message)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range_of_x, kde_values_logistic, label='Density (Logistic Kernel)',
↪  color='blue')
plt.plot(range_of_x, kce_values_logistic, label='Empirical CDF', color='green')
plt.axvline(x=non_parametric_var_logistic, color='red', linestyle='--', label='VaR')
plt.xlabel('Return')
plt.ylabel('Density / CDF')
plt.title('Non-Parametric VaR with Logistic Kernel')
plt.grid(True)
plt.legend()
plt.show()

#Part b
# data for the period from January 2017 to December 2018
data_2017_2018 = df.loc['2017-01-01':'2018-12-31', 'Return'].dropna()

# VaR threshold
var_threshold = -0.0433  # Example VaR value obtained earlier
```

```
#  exceedances (returns below the threshold)
exceedances = data_2017_2018 < var_threshold

# Calculate the proportion of exceedances
exceedance_proportion = exceedances.sum() / len(data_2017_2018)

# confidence level and expected exceedance proportion
confidence_level = 0.95
expected_proportion = 1 - confidence_level

# Validation: Check if the observed exceedance proportion matches the expected
validation = exceedance_proportion <= expected_proportion

# Print the results
print(f"Observed exceedance proportion: {exceedance_proportion:.4f}")
print(f"Expected exceedance proportion: {expected_proportion:.4f}")
print(f"Validation result: {'Valid' if validation else 'Not Valid'}")
```

Code Snippet 2: Question B

```
# Content from QuestionB.ipynb
#Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_excel('Natixis_ihab.xlsx', header=None, names=['Date', 'Price'])

#Computing the log return which is log(St/St-1)
df.set_index('Date', inplace=True)
df.loc[:,'Log_Return'] = np.log(df.loc[:,'Price']/df.loc[:,'Price'].shift(1))

#On va calculer le lissage car on va porter moins d'importance aux données
↪  lointaines
lissage = 0.6
df.loc["2015-01-05", "Returns_lissés"] = df.loc["2015-01-05", "Log_Return"]
df.loc["2015-01-06":, "Returns_lissés"] = df.loc["2015-01-06":,"Log_Return"]*lissage
↪  + df.loc["2015-01-06":,"Log_Return"].shift(-1)*(1-lissage)

#Simulate a number N
S0 = df["Price"].iloc[-1]  # The last observed price in your dataset
log_returns = df.loc[:,"Returns_lissés"].dropna()
mumoinssigmasur2 = log_returns.mean() #car l'esperance de Ln(St/St-1)=mu -
↪  sigma*sigma*1/2
sigma = log_returns.std()
mumoinssigmasur2,sigma

# Number of future days to simulate
n_days = 1000

# Generate Brownian motion: Random normal values for each day
brownian_motion = np.random.normal(0, 1, n_days)

# Calculate the future prices using the formula
future_prices = [S0]
```

```
for i in range(1, n_days):
    St = future_prices[i-1] * np.exp(mumoinssigmasur2 + sigma *
    ↪  brownian_motion[i-1])
    future_prices.append(St)

# Create a date range starting from the next day
future_dates = pd.date_range(start=df.index[-1] + pd.Timedelta(days=1),
↪  periods=n_days, freq='D')

# Create a DataFrame with the future prices
future_df = pd.DataFrame(data=future_prices, index=future_dates,
↪  columns=["Estimated_Price"])

future_df["Estimated_Price"].mean()

#Calculating the call price using the Black Scholes formula
K = S0
r = 0
T_minus_t =21
future_df['d1'] = (np.log(future_df['Estimated_Price'] / K) + (r + 0.5 * sigma**2) *
↪  T_minus_t) / (sigma * np.sqrt(T_minus_t))
future_df['d2'] = future_df['d1'] - sigma * np.sqrt(T_minus_t)

from scipy.stats import norm
future_df['Call_Price'] = future_df['Estimated_Price']*norm.cdf(future_df['d1']) -
↪  K*np.exp(-r * T_minus_t)*norm.cdf(future_df['d2'])

#VaR Calculation
confidence_level = 0.95
def empirical_var(returns, confidence_level):
    returns = returns.sort_values(ascending = False)
    return -np.percentile(returns, (1 - confidence_level) * 100)

VaR = empirical_var(future_df.loc[:,'Call_Price'],confidence_level)
print(f"\nHistorical VaR at {confidence_level * 100}% confidence level: {VaR}")

price_changes = future_df['Call_Price'].diff()
VaR2 = empirical_var(price_changes.dropna(), confidence_level)
print(f"\nHistorical VaR at {confidence_level * 100}% confidence level: {VaR2}")
```

Code Snippet 3: Question C

```
# Content from QuestionC.ipynb
#Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from scipy.stats import norm

data = pd.read_excel('Natixis_ihab.xlsx', header=None, names=['Date', 'Price'])

#Returns
df['Price'] = pd.to_numeric(df['Price'])
df['Return'] = df['Price'].pct_change(1)
df.set_index('Date', inplace = True)
returns = df['Return'].dropna()
```

```
#Gains
gains = returns[returns > 0]
sorted_gains = np.sort(gains)

#Losses
losses = returns[returns < 0]
sorted_losses = np.sort(abs(losses))

#Pickands
# k has to go to infinity but not as fast as n, we choose log(n)
def pickands_estimator(extreme_values):
    n = len(extreme_values)
    k = int(np.log(n))
    return ( 1 / np.log(2) ) * np.log( ( extreme_values[n-k+1]
                - extreme_values[n-2*k+1] ) / ( extreme_values[n-2*k+1]
                - extreme_values[n-4*k+1] ) )

#we then just apply this function to our extreme gains and losses
shape_gains = pickands_estimator(sorted_gains)
shape_losses = pickands_estimator(sorted_losses)
shape_gains, shape_losses

#Part b
# Let's find the threshhols (seuil) using the empirical VaR
confidence_level = 0.95
def empirical_var(returns, confidence_level):
    returns = returns.sort_values(ascending = False)
    return -np.percentile(returns, (1 - confidence_level) * 100)

VaR = empirical_var(returns,confidence_level)
print(f"\nHistorical VaR at {confidence_level * 100}% confidence level: {VaR}")

#The second step is finding the exceedances (returns below the seuil)
exceedances2 = returns[returns < VaR]
exceedances3 = returns[returns < VaR]

exceedances2['Gap'] = exceedances2.index.to_series().diff().dt.days

#Let's check the most frequent gaps
plt.hist(exceedances2['Gap'].dropna(), bins=20, edgecolor='black')
plt.title('Distribution of Gaps Between Exceedances')
plt.xlabel('Gap (days)')
plt.ylabel('Frequency')
plt.show()

gap = 1  # Define a gap (days) for de-clustering
exceedances3['Cluster'] = (exceedances3.index.to_series().diff().dt.days >
↪  gap).cumsum()

total_exceedances = len(exceedances3)
unique_clusters = exceedances3['Cluster'].nunique()

extremal_index = unique_clusters / total_exceedances
print(f"Extremal Index: {extremal_index}")
```

Code Snippet 4: Question D

```
# Content from QuestionD.ipynb
#Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from scipy.stats import norm

xlsx_file = "Dataset TD4.xlsx"
csv_file = "Dataset TD4.csv"


df = pd.read_excel(xlsx_file)


df.to_csv(csv_file, index=False, encoding='utf-8')

print(f"Le fichier {xlsx_file} a été converti en {csv_file}.")

csv_data = pd.read_csv("Dataset TD4.csv", delimiter=",")

#Defining the model parameters
volume = []
transaction_price = []
transaction_sign = []

for i in range(len(csv_data) - 1):  # Adjust to -1 to avoid index issues
    # Check if the 'volume of the transaction (if known)' is not NaN
    if pd.notna(csv_data['volume of the transaction (if known)'].iloc[i]):
        # Append the volume value to the list
        volume.append(csv_data['volume of the transaction (if known)'].iloc[i])

        # Extend the transaction_price list with current and next prices
        transaction_price.extend([
            csv_data['Price (before transaction)'].iloc[i],
            csv_data['Price (before transaction)'].iloc[i + 1]
        ])

        # Append the transaction sign to the list
        transaction_sign.append(csv_data['Sign of the transaction'].iloc[i])

# Returns
returns = [
    (transaction_price[i + 1] - transaction_price[i]) / transaction_price[i]
    for i in range(len(transaction_price) - 1)
]
returns = np.array(returns)

# sigma (standard deviation of returns) annualized
sigma = np.std(returns) * np.sqrt(24) * np.sqrt(252)  # 24 hours/day, 252 trading
↪  days/year
print("Sigma (annualized):", sigma)

# total volume
total_volume = np.sum(volume)
print("Total Volume:", total_volume)
```

```
# Time interval (1 hour in a day)
time_interval = 1 / 24
print("Time Interval:", time_interval)

# price difference
price_difference = np.array([
    transaction_price[i + 1] - transaction_price[i]
    for i in range(0, len(transaction_price) - 1, 2)
])

# Signed volume
signed_volume = np.array([
    volume[i] * transaction_sign[i]
    for i in range(len(volume))
])

volume_squared_signed = np.array([
    transaction_sign[i] * volume[i] ** 2
    for i in range(len(price_difference))
])

# modified prices
modified_price_gamma = price_difference
modified_price_tha = -price_difference

# Reshape signed volume for compatibility
X_gamma_mod = signed_volume.reshape(-1, 1)

# Stack columns to create X_tha_mod
X_tha_mod = np.column_stack((signed_volume, volume_squared_signed))

#Linear Regression
def linear_regression_np(X, y):

    X_mean = np.mean(X)
    y_mean = np.mean(y)

    # Calculate the slope (m)
    slope = np.sum((X - X_mean) * (y - y_mean)) / np.sum((X - X_mean) ** 2)

    # Calculate the intercept (c)
    intercept = y_mean - slope * X_mean

    return slope, intercept

X_gamma_array = np.array(X_gamma_mod[:, 0])
prix_gamma_array = np.array(modified_price_gamma)

slope_gamma, intercept_gamma = linear_regression_np(X_gamma_array, prix_gamma_array)


y_pred_gamma = slope_gamma * X_gamma_array + intercept_gamma

# MSE
mse_gamma = np.mean((prix_gamma_array - y_pred_gamma) ** 2)

# R^2 score
```

```
r2_gamma = 1 - (np.sum((prix_gamma_array - y_pred_gamma) ** 2) /
                np.sum((prix_gamma_array - np.mean(prix_gamma_array)) ** 2))

# Print regression results
print(f"\nEstimation gamma: {slope_gamma}")
print(f"MSE: {mse_gamma}; R^2: {r2_gamma}")

# Plot
plt.scatter(X_gamma_array, prix_gamma_array, color="blue", label="Data")
plt.plot(X_gamma_array, y_pred_gamma, color="red", label="Regression Line")
plt.xlabel("Signed volume")
plt.ylabel("Gamma Price")
plt.title("Gamma Linear Regression")
plt.legend()
plt.show()

#Multivariate Linear regression
def linear_regression_multivariate(X, y):
    X_mean = np.mean(X, axis=0)
    y_mean = np.mean(y)

    X_cov = np.dot(X.T, X) - np.sum(X, axis=0) * X_mean

    # Compute the inverse of the covariance matrix
    X_cov_inv = np.linalg.inv(X_cov)

    # Calculate the covariance between X and y
    X_X_mean_y_mean = np.dot(X.T, y) - np.sum(X, axis=0) * y_mean

    # Calculate coefficients (slopes)
    coefficients = np.dot(X_cov_inv, X_X_mean_y_mean)

    # Calculate the intercept
    intercept = y_mean - np.dot(coefficients.T, X_mean)

    return coefficients, intercept

X_tha_array = np.column_stack((X_gamma_mod, volume_squared_signed))
prix_tha_array = np.array(modified_price_tha)
coefficients_tha, intercept_tha = linear_regression_multivariate(X_tha_array,
↪  prix_tha_array)
y_pred_tha = np.dot(X_tha_array, coefficients_tha) + intercept_tha

# Calculating the Mean Squared Error (MSE) and R-squared (R²)
mse_tha = np.mean((prix_tha_array - y_pred_tha) ** 2)
r2_tha = 1 - sum((prix_tha_array - y_pred_tha) ** 2) / sum(
    (prix_tha_array - np.mean(prix_tha_array)) ** 2
)

# Calculating the theta coefficient
tau = time_interval
tha = coefficients_tha[1] * tau  # Etha is the second coefficient
print(f"\nEstimation tha: {tha}")
print(f"MSE: {mse_tha}; R²: {r2_tha}")

#Plot
plt.scatter(volume_squared_signed, prix_tha_array, color="blue", label="Data")
```

```
plt.scatter(volume_squared_signed, y_pred_tha, color="red", label="Model
↪  Predictions")
plt.xlabel("Squared Signed Volume")
plt.ylabel("Etha Price")
plt.title("Multivariate Regression for Etha")
plt.legend()
plt.show()

X = 1000000  # Initial number of actions
T = [(1 / 24) * i for i in range(1, 25)]  # Time intervals in days
Lambda_values = [1e-5, 1e-4, 1e-3]  # Different values of Lambda

#Plot Optimal liquidation Strategy
plt.figure(figsize=(10, 6))

for Lambda in Lambda_values:
    K = np.sqrt(Lambda * (sigma**2) / tha)  # Calculate K
    strat = [np.sinh(K * (1 - t)) * X / np.sinh(K * 1) for t in T]  # Strategy
    plt.plot(T, strat, label=f'Lambda = {Lambda:.1e}')

plt.title("Optimal Liquidation strategy")
plt.xlabel("Time in days")
plt.ylabel("Quantity")
plt.legend()
plt.grid(True)
plt.show()
```

Code Snippet 5: Question E

```
# Content from QuestionE.ipynb
#Libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from scipy.stats import norm

Currencies = pd.read_csv("Dataset-TD5-csv.csv")

# Split dataset and calculate averages
GBPEUR = Currencies.iloc[:, :3]
SEKEUR = Currencies.iloc[:, 3:6]
CADEUR = Currencies.iloc[:, 6:]

GBPEUR['GBPEUR_AVG'] = GBPEUR.iloc[:, [1, 2]].mean(axis=1)
SEKEUR['SEKEUR_AVG'] = SEKEUR.iloc[:, [1, 2]].mean(axis=1)
CADEUR['CADEUR_AVG'] = CADEUR.iloc[:, [1, 2]].mean(axis=1)

# Work with the last 513 data points
GBPEUR = GBPEUR.iloc[-513:]
SEKEUR = SEKEUR.iloc[-513:]
CADEUR = CADEUR.iloc[-513:]

# Calculate returns Step 1
GBPEUR_returns = GBPEUR['GBPEUR_AVG'].pct_change().dropna()
SEKEUR_returns = SEKEUR['SEKEUR_AVG'].pct_change().dropna()
```

```
CADEUR_returns = CADEUR['CADEUR_AVG'].pct_change().dropna()

# Haar Transform Function Step 2
def haar_transform(data):
    n = len(data)
    output = np.zeros(n)
    while n > 1:
        n = n // 2
        for i in range(n):
            output[i] = (data[2 * i] + data[2 * i + 1]) / 2
            output[n + i] = (data[2 * i] - data[2 * i + 1]) / 2
        data[:n] = output[:n]
    return output

#Correlation
def multiresolution_correlation(haar1, haar2, level):
    length = 2 ** level
    truncated_haar1 = haar1[:length]
    truncated_haar2 = haar2[:length]
    return np.corrcoef(truncated_haar1, truncated_haar2)[0, 1]

# Correlation Matrix at Level Step 3
def correlation_matrix_at_level(level, haar_transforms):
    matrix_size = len(haar_transforms)
    correlation_matrix = np.zeros((matrix_size, matrix_size))
    for i in range(matrix_size):
        for j in range(i, matrix_size):
            if i == j:
                correlation_matrix[i, j] = 1
            else:
                correlation = multiresolution_correlation(
                    haar_transforms[i], haar_transforms[j], level)
                correlation_matrix[i, j] = correlation_matrix[j, i] = correlation
    return correlation_matrix

# Hurst Exponent Calculation
def Hurst(serie):
    T = len(serie)
    M2 = np.mean((np.abs(serie[1:] - serie[:-1]))**2) # absolute moment scale 1/N
    M2_prime = np.mean(np.abs((serie[2:] - serie[:-2]))**2) # absolute moment scale
    ↪  2/N
    H_est = 0.5 * np.log2(M2_prime / M2)
    return H_est

# Applying our Haar Transform
gbpeur_haar = haar_transform(GBPEUR_returns.values)
sekeur_haar = haar_transform(SEKEUR_returns.values)
cadeur_haar = haar_transform(CADEUR_returns.values)

# Number of levels
num_levels = 9

haar_transforms = [gbpeur_haar, sekeur_haar, cadeur_haar]
correlation_matrices = [correlation_matrix_at_level(level, haar_transforms) for
↪  level in range(num_levels)]

# Hurst Exponents
hurst_gbpeur = Hurst(GBPEUR['GBPEUR_AVG'].values)
```

```
hurst_sekeur = Hurst(SEKEUR['SEKEUR_AVG'].values)
hurst_cadeur = Hurst(CADEUR['CADEUR_AVG'].values)

print("Hurst Exponents:", hurst_gbpeur, hurst_sekeur, hurst_cadeur)

# High-Frequency Volatilities
vol_gbpeur_hf = np.std(GBPEUR_returns)
vol_sekeur_hf = np.std(SEKEUR_returns)
vol_cadeur_hf = np.std(CADEUR_returns)

volatility_gbpeur = [vol_gbpeur_hf * (2**j)**hurst_gbpeur for j in
↪  range(num_levels)]
volatility_sekeur = [vol_sekeur_hf * (2**j)**hurst_sekeur for j in
↪  range(num_levels)]
volatility_cadeur = [vol_cadeur_hf * (2**j)**hurst_cadeur for j in
↪  range(num_levels)]

print("Volatilities (GBPEUR):", volatility_gbpeur)
print("Volatilities (SEKEUR):", volatility_sekeur)
print("Volatilities (CADEUR):", volatility_cadeur)
```