Market Risk Project Report

Financial Engineering

Authors :
Axel Portier
Gaspard Salluron–Besnard

Course Coordinators :
Julie Gamain
Mathieu Garcin

8 janvier 2026

Table des matières

1 Implemented tools and libraries

In accordance with the project guidelines, the use of specialized statistical or financial packages was strictly avoided. All estimation procedures, including kernel-based risk measures, extreme value estimators, and model calibrations, were implemented manually.

The following Python libraries were used exclusively for data manipulation, numerical computation, and visualization purposes. They do not provide built-in functions for the estimation methods studied in this project and were therefore necessary to handle the data and present the results.

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
```

Listing 1 – Tools necessary

- `pandas` : used for data loading, cleaning, time series handling, and return computation.
- `numpy` : used for numerical operations, vectorized computations, and matrix-based calculations required for manual implementation of estimators.
- `matplotlib` : used solely for data visualization, including histograms, kernel density plots, and comparisons of risk measures.

All these libraries are assumed to be imported whenever required in the code listings presented in the following sections.

Abstract

This project investigates several approaches to market risk modeling through both non-parametric methods and tools from Extreme Value Theory. We first estimate a historical Value at Risk (VaR) based on kernel density estimation and analyze the impact of bandwidth selection by comparing under-smoothing, Silverman's rule, Scott's rule, and over-smoothing. The sensitivity of risk measures to the choice of the bandwidth is illustrated through both density estimates and VaR comparisons.

We then study the tail behavior of asset returns using Extreme Value Theory, focusing on the Pickands estimator. Different functional forms are tested and the dependence of the estimator on the threshold parameter is analyzed. In a third step, we estimate the parameters of Bouchaud's price impact model and discuss its ability to describe market dynamics. Finally, we explore multiscale properties of financial time series using Haar wavelets and estimate the Hurst exponent to assess long-range dependence and volatility scaling. Overall, the project highlights how modeling choices directly influence risk estimation and financial interpretation.

2 Introduction

Market risk measurement plays a central role in portfolio management and financial regulation. Accurately modeling the distribution of asset returns is particularly challenging due to empirical features such as heavy tails, asymmetry, and complex dependence structures. These characteristics often limit the relevance of classical parametric models and motivate the use of alternative approaches.

In this project, we adopt a non-parametric and data-driven perspective to analyze financial risk. We begin by estimating a historical Value at Risk using kernel density estimation with a biweight kernel. Special attention is paid to the choice of the bandwidth parameter, as different levels of smoothing can lead to noticeably different density estimates and risk measures, especially in the tails of the distribution.

We then extend the analysis within an Extreme Value Theory framework in order to better capture extreme gains and losses. Using the Pickands estimator, we study the sensitivity of tail index estimates to the choice of the threshold parameter and compare the resulting implications for tail-based Value at Risk.

In a third part, we estimate the parameters of Bouchaud's price impact model to evaluate its capacity to describe the relationship between order flow and price movements. Finally, we adopt a multiscale approach using Haar wavelets to analyze correlation structures across different time scales and estimate the Hurst exponent as a measure of long-range dependence.

Through these different methodologies, this project illustrates how market risk assessment depends not only on the data, but also on the modeling assumptions and estimation choices made throughout the analysis.

3 Question A (Ex2, part of Q1 and of Q2 of TD1)

3.1 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 (biweight Kernel, that is the derivative of the logistic function.

The Biweight kernel offers is more sensitive to distribution's tails. Unlike the Gaussian kernel, it assigns zero weight to points beyond the bandwidth."

According to the question, the pdf of the Kernel is given by :

$$K(u) = \frac{15}{16}(1-u^2)^2 1_{\{|u|\leq 1\}} \tag{1}$$

Then, we can remind the general form of the cdf estimated in the non-paremtric Kernel method by :

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} G\left(\frac{x-R_i}{h}\right)$$

Where $G(u)$ is the primitive of the Kernel $K(u)$:

$$G(u) = \int_{-1}^{u} K(x)dx$$

$$\iff G(u) = \int_{-1}^{u} \frac{15}{16}(1-x^2)^2 dx \qquad \text{for } u \in [-1,1]$$

In developping the polynom $(1-x^2)^2 = 1 - 2x^2 + x^4$, we obtain for $u \in [-1,1]$:

$$G(u) = \frac{3}{16}u^5 - \frac{5}{8}u^3 + \frac{15}{16}u + \frac{1}{2}$$

Finally, we will determine the VaR numerically by dichotomie, using this relation :

$$\hat{F}(\text{VaR}_\alpha) = P(R_t <= \text{VaR}_\alpha) = \alpha$$

First of all we loaded the data, and made sure that all the columns were well encoded. With the code :

```
df=pd.read_csv('natixis_stock.txt', sep="\t", names=["date", "price"])
df["price"]=df["price"].str.replace(',','.').astype(float)
df["date"]=pd.to_datetime(df["date"], format="%d/%m/%Y")
df["price"] = pd.to_numeric(df["price"], errors="coerce")

print(df.head())
effectif_tot = len(df)
print("\nEffectif:", effectif_tot)
```

Listing 2 – Data Processing

Then, as it is mendatorate to work only between january 2015 and december 2017, we truncated the data. We also computed the arithmetic return of the prices with the formula :

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \tag{2}$$

Where :
— R_t is the arithmetic return at time t.
— P_t is the price at time t.
— P_{t-1} is the price at time $t-1$.

Moreover, as we highlighted, in the formula of the CDF estimated by the kernel method, a term h is introduced. This term, known as the bandwidth, acts as a smoothing parameter that controls the trade-off between bias and variance : it enables the kernel to either overfit the raw data or smooth the approximation to capture the general shape of the distribution. Effectively, the bandwidth determines how closely the estimated curve fits the initial data points. In the literature, we found several statistical methods to determine the optimal h, and we have tested the following ones.

Under Smoothing :

$$h_{under} = 0.79 \cdot \sigma \cdot n^{-1/5}$$

Silverman's Rule :

$$h_{silverman} = 1.06 \cdot \sigma \cdot n^{-1/5}$$

Over Smoothing :

$$h_{over} = 1.144 \cdot \sigma \cdot n^{-1/5}$$

Scott's Rule :

$$h_{scott} = 3.5 \cdot \sigma \cdot n^{-1/3}$$

where σ stands for the standard deviation of the serie.

```
df_qA = df[(df["date"] >= "2015-01-01") & (df["date"] < "2017-01-01")].
    copy()
df_qA["return"] = df_qA["price"].pct_change()
df_qA=df_qA.dropna()
df_qA.sort_values(by="return", ascending = True)
returns = df_qA["return"].values

effectif_A = len(returns)
print("\nEffectif:", effectif_A)

methods_bandwidths = {
    "Under Smoothing" : 0.79 * np.std(returns) * (effectif_A**(-1/5)),
    "Silverman": 1.06 * np.std(returns) * (effectif_A ** (-1/5)),
    "Over Smoothing" : 1.144 * np.std(returns) * (effectif_A**(-1/5)),
    "Scott": 3.5 * np.std(returns) * (effectif_A**(-1/3))
}

epsilon = 1e-10
```

Listing 3 – Data Preparation

Next, we implemented the estimated PDF and CDF using the kernel density function $K(u)$ and its corresponding integral $G(u)$. To compute the Value-at-Risk (VaR), we used a dichotomy method. This approach allows us to find the specific return value (the abscissa) where the estimated CDF matches our confidence level α. In other words, we find the threshold such that the probability of returns falling below this level satisfies our risk condition.

```
def K(u):
    abs_u = np.abs(u)
    mask = abs_u <= 1
    return (15/16) * (1 - u**2)**2 * mask
def kernel_density_estimate(x, R, h):
    n = len(R)
    u = (x - R) / h
    kde = np.sum(K(u)) / (n * h)
    return kde
def G(x):
    if x>=1 :
        return 1
    elif x< -1 :
        return 0
    else :
        return (3/16)*x**5+(5/8)*x**3-(15/16)*x+0.5
def cdf_estimate(x, R, h):
    total = 0
    for i in range(len(R)):
        total += G((x - R[i]) / h)
    return total/len(R)
def dich(R,eps,alph, h):
    a = min(R)
    b = max(R)
    m=(a+b)/2
    F = cdf_estimate(m, R, h)
    while abs(b-a) > eps:
        if F > alph:
            b=m
        else:
            a=m
        m=(a+b)/2
        F = cdf_estimate(m, R, h)
    return m
```

Listing 4 – Functions

```
alpha = float(input("alpha␣error␣(ex␣5%)")) * 0.01
VaR = {}
for name,h in methods_bandwidths.items() :
    VaR_npar_qA = dich(returns, epsilon, alpha, h)
    VaR[name] = VaR_npar_qA
    print(f"\n␣-VaR␣{int(100-alpha*100)}%␣1␣day␣horzion␣=␣{VaR_npar_qA
        :.4f}␣({VaR_npar_qA*100:.2f}%)␣with␣{name}␣bandwidth␣(h={h:.3f})"
        )
```

Listing 5 – Loop in order to determine the VaR using different bandwidth selection methods

The following VaR values were obtained using a **dichotomic method** :

```
-VaR 95% 1 day horizon = -0.0393 (-3.93%) with Under Smoothing bandwidth (h=0.005)
-VaR 95% 1 day horizon = -0.0399 (-3.99%) with Silverman bandwidth (h=0.007)
-VaR 95% 1 day horizon = -0.0405 (-4.05%) with Over Smoothing bandwidth (h=0.008)
-VaR 95% 1 day horizon = -0.0424 (-4.24%) with Scott bandwidth (h=0.010)
```

Intepretation :
We clearly see that the bandwidths produce different VaR values : some are more pessimistic than others, simply due to a different level of smoothing. Finally with h increasing, the smoothing is higher but the VaR is then more pessimistic.

We then wanted to see how the different methods were estimating the initial curve and how the bandwitdh was impacting it. We therefore ploted both the cdf and pdf estimated.

```
plt.figure(figsize=(15, 6))
x = np.linspace(min(returns), max(returns), 1000)
for name,h in methods_bandwidths.items() :
    y = [kernel_density(i, returns, h) for i in x]
    plt.plot(x,y, label = f"{name} {h:.3f}")
plt.hist(returns, bins=50, density=True, alpha=0.6, color='gray',
   edgecolor='black', label='Returns Histogram')

plt.axvline(x=VaR["Silverman"], color='red', linestyle='--', label='VaR
   Silverman (h=0.007)')
plt.axvline(x=VaR["Scott"], color='blue', linestyle='--', label='VaR
   Scott (h=0.01)')
plt.text(VaR["Scott"], 0, f'{VaR["Scott"]:.3f}', color='blue',
         ha='left', va='top', fontweight='bold')
plt.text(VaR["Silverman"], -1, f'{VaR["Silverman"]:.3f}', color='red',
         ha='right', va='top', fontweight='bold')
plt.legend()
```

Listing 6 – Plot of the pdf estimated with different bandwidth

```
plt.figure(figsize=(15, 6))
x = np.linspace(min(returns), max(returns), 1000)
for name,h in methods_bandwidths.items() :
    y = [cdf_estimate(i, returns, h) for i in x]
    plt.plot(x,y, label = f"{name} {h:.3f}")
plt.legend(title='Bandwidth Values (h)')
```

Listing 7 – Plot of the cdf estimated with different bandwidth

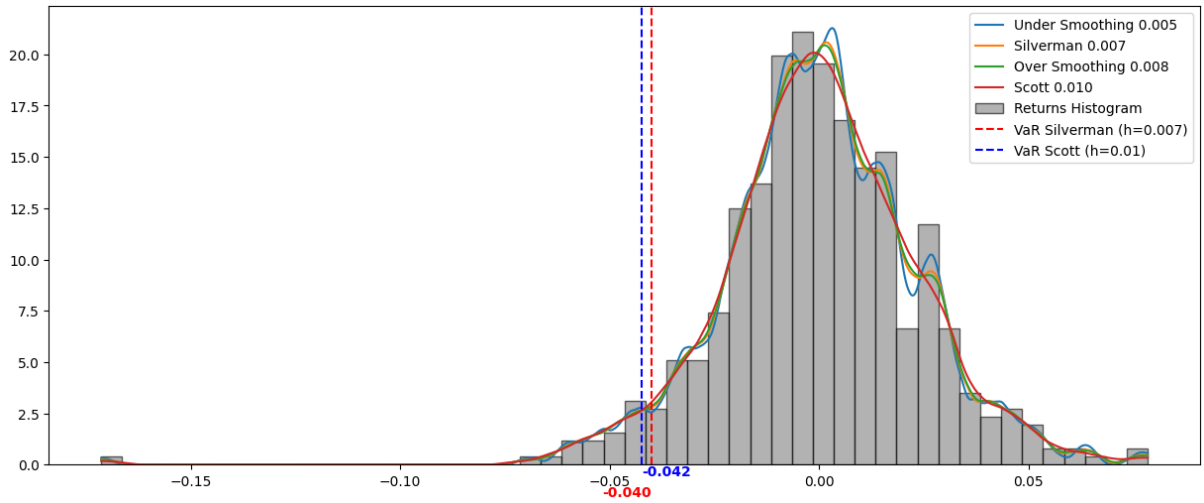

FIGURE 1 – Plot of the pdf estimated with different bandwidth

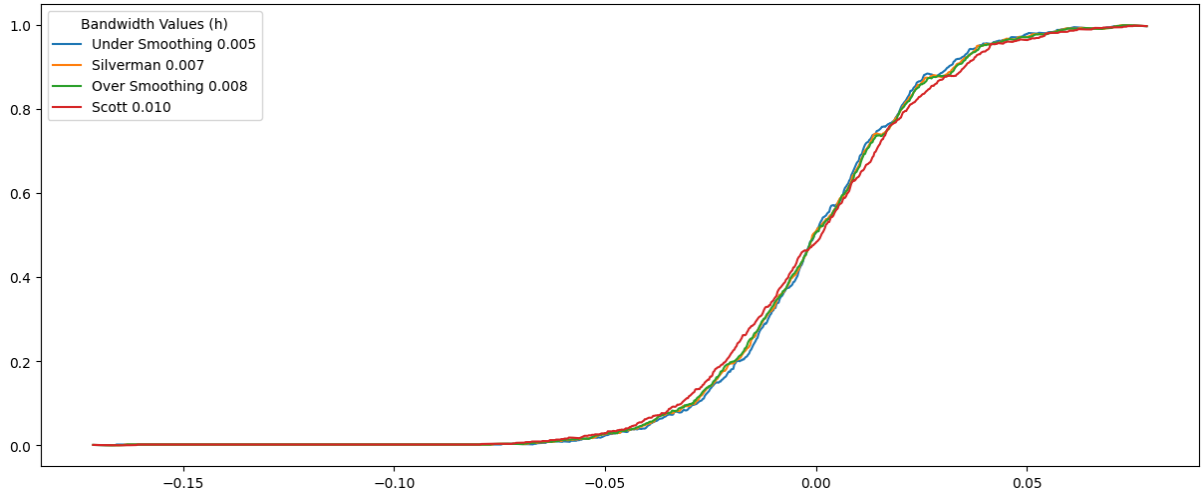

FIGURE 2 – Plot of the cdf estimaed with different bandwidth

Intepretation

From the plot, we can clearly see that the estimated density depends strongly on the choice of the bandwidth h. All curves are **centered** around zero and follow the general shape of the **returns histogram**, but they differ in how **smooth** they are and how much detail they capture.

- **With a smaller bandwidth (h = 0.005)**, the density is much more irregular. This suggests under-smoothing, where **noise is captured**.
- **As the bandwidth increases (Silverman and h = 0.008)**, the curves become **smoother** and more **stable**.
- **Using Scott's rule (h = 0.010)** results in the smoothest curve, which highlights the overall shape but tends to **flatten** local variations, especially in the **tails**.

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

```
df_new = df[(df["date"] >= "2017-01-01") & (df["date"] < "2019-01-01")].
    copy()
df_new["return"] = df_new["price"].pct_change()
df_new=df_new.dropna()
effectif_new = len(returns)
print("\nEffectif:", effectif_new)
for name in methods_bandwidths.keys() :
    print(f"\n -{name}'s bandwidth : {(len(df_new[df_new["return"]<=VaR[
        name]])/effectif_new)*100:.3f}% return exceed VaR (95% confidence
        )")
```

Listing 8 – Loop to obtain the proportion of return crossing the VaR threshold

```
-Under Smoothing's bandwidth : 1.572% return exceed VaR (95% confidence)
-Silverman's bandwidth : 1.572% return exceed VaR (95% confidence)
-Over Smoothing's bandwidth : 1.572% return exceed VaR (95% confidence)
-Scott's bandwidth : 1.375% return exceed VaR (95% confidence)
```

Intepretation

According to the results, the percentage of returns exceeding the VaR 95% threshold is below the 5% limit for all four methods. This **validates** that our non-parametric Kernel VaR approach is appropriate and sufficiently conservative for this dataset.

To go further, we should perform a **backtesting analysis** to verify that the observations crossing the threshold do not form **clusters**. It is crucial to ensure that these exceedances are **independent** over time to guarantee the model's reliability across different market regimes.

4 Question B (Ex2, Q5 of TD2)

4.1 Calculate the expected shortfall for the VaR calculated in question A. How is the result, compared to the VaR ?

```
ES = {}
for name in methods_bandwidths.keys() :
    expected_shortfall = df_qA[df_qA["return"]<=VaR[name]]["return"].
        mean()
    ES[name] = expected_shortfall
    print(f"\n -{name} bandwidth's method : {expected_shortfall*100:.2f
        }% expected shortfall (correspondant VaR: {VaR[name]*100:.2f}%)")
```

Listing 9 – Expected shortfall

-Under Smoothing's bandwidth : -5.60% ES (correspondant VaR: -3.93%)
-Silverman's bandwidth : -5.60% ES (correspondant VaR: -3.99%)
-Over Smoothing's bandwidth : -5.60% ES (correspondant VaR: -4.05%)
-Scott's bandwidth : -5.74% ES (correspondant VaR: -4.24%)

```
x = np.linspace(min(returns), max(returns), 1000)
y = [kernel_density(i, returns, h) for i in x]
plt.figure(figsize=(15, 6))
plt.plot(x,y, label = "Silvermann (h=0.007)")
plt.hist(returns, bins=50, density=True, alpha=0.6, color='gray',
    edgecolor='black', label='Returns Histogram')
plt.axvline(x=VaR["Silverman"], color='red', linestyle='--', label='VaR
    Silverman (h=0.007)')
plt.axvline(x=ES["Silverman"], color='blue', linestyle='--', label='ES
    Silverman (h=0.007)')
plt.text(ES["Silverman"], 0, f'{ES["Silverman"]:.3f}', color='blue',
         ha='left', va='top', fontweight='bold')
plt.text(VaR["Silverman"], -1, f'{VaR["Silverman"]:.3f}', color='red',
         ha='right', va='top', fontweight='bold')
plt.legend()
plt.title('Density kernel for VaR 95% and its corresponding expected
    shortfall with Silverman bandwidth')
```

Listing 10 – Code for ploting the pdf with the ES and VaR to compare

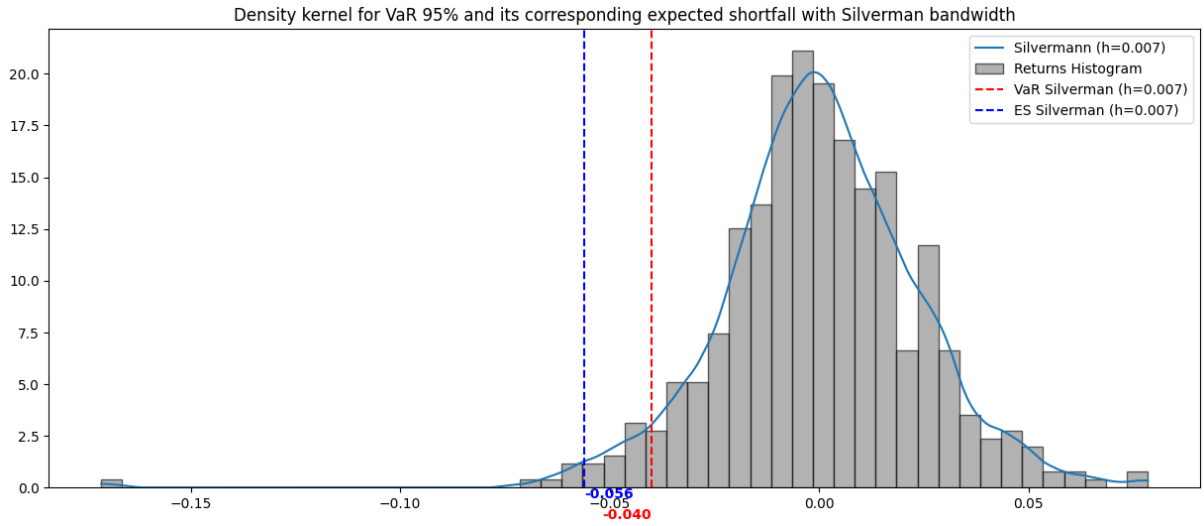

FIGURE 3 – Plot of the pdf with the ES and VaR to compare

Interpretation

We see clearly that our expected shortfalls, no matter the method of the bandwidth, are larger than the VaR. It is perfectly coherent as the expected shortfall is defined as the average occuring beyond the VaR threshold. So our results confirm the consistency of the non parametric model.

5 Question C (Ex2, Q1 and Q2 of TD3)

5.1 a – Estimate the GEV parameters for the two tails of the distribution of returns, using the estimator of Pickands. What can you conclude about the nature of the extreme gains and losses ?

While historical VaR is limited by the worst past events, Extreme Value Theory (EVT) allows us to model the asymptotic behavior of the distribution's tails. It provides a more reliable risk assessment for extreme market shocks that have not yet occurred in the sample.

```
X_losses = df[df["return"]<0]["return"]
X_losses = (-X_losses).copy()
n_losses = len(X_losses)
X_gain = df[df["return"]>=0]["return"].copy()
n_gain = len(X_gain)
print("Number of losses :",n_losses)
print("Number of gain :",n_gain)
```

Listing 11 – Code for separate losses and gain

Moreover, the core of EVT is based on the assumption of **i.i.d.** returns. We will analyze gains and losses separately to estimate the GEV parameters associated with the limit distribution of the maximum of the returns.

According to the Market Risk course of M. Garcin :

Let $(X_n)_{n\geq 1}$ be a sequence of **i.i.d.** random variables, whose cumulative distribution function F belongs to the **max-domain of attraction** of a Generalized Extreme Value (**GEV**) distribution with shape parameter $\xi \in \mathbb{R}$.

Let $k(n)$ be a function $\mathbb{N} \to \mathbb{N}$. If :

$$\lim_{n\to\infty} k(n) = \infty \quad \text{and} \quad \lim_{n\to\infty} \frac{k(n)}{n} = 0$$

Then, the **Pickands estimator**, defined by :

$$\hat{\xi}^P_{k(n),n} = \frac{1}{\log(2)} \log\left(\frac{X_{n-k(n)+1:n} - X_{n-2k(n)+1:n}}{X_{n-2k(n)+1:n} - X_{n-4k(n)+1:n}}\right)$$

converges in **probability** to ξ :

$$\hat{\xi}^P_{k(n),n} \xrightarrow{P} \xi \quad \text{as} \quad n \to \infty$$

Moreover, if :

$$\lim_{n\to\infty} \frac{k(n)}{\log(\log(n))} = \infty$$

then the convergence of $\hat{\xi}^P_{k(n),n}$ towards ξ is **almost sure** $(a.s.)$ and not only in probability.

```python
def root(x):
    return np.sqrt(x)
def log(x):
    return np.log(x)

def pickands_estimator(X, k):
    k=int(k) # need an integer as k will represent a list's index
    X = X.sort_values(ascending=True)
    n = len(X)

    if 4*k >= n:  # validity condtion as n-4*k>=0 and n-2*k>=0
        return np.nan

    num = X.iloc[n-k] - X.iloc[n-2*k]
    denom = X.iloc[n-2*k] - X.iloc[n-4*k]

    return (1/np.log(2)) * np.log(num/denom)
```

Listing 12 – Functions needed to determine the EVT parameter

To ensure the convergence of the Pickands estimator $\hat{\xi}^P_{k(n),n}$, the sequence $k(n)$ must satisfy specific growth conditions relative to the sample size n. We have chosen to test two classic functions, **Logarithmic** and **Square Root**, because they both respect the required asymptotic behavior :

— **Logarithmic Function** : $k(n) = \lfloor \ln(n) \rfloor$, satisfying :

$$\lim_{n\to\infty} \ln(n) = \infty \quad \text{and} \quad \lim_{n\to\infty} \frac{\ln(n)}{n} = 0$$

— **Square Root Function** : $k(n) = \lfloor \sqrt{n} \rfloor$, satisfying :

$$\lim_{n\to\infty} \sqrt{n} = \infty \quad \text{and} \quad \lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$$

```python
print(f"-Pickands estimator of losses (log) = {pickands_estimator(X_losses, log(n_losses)):.3f} for k = {int(log(n_losses))}")
print(f"-Pickands estimator of losses (root) = {pickands_estimator(X_losses, root(n_losses)):.3f} for k = {int(root(n_losses))}")

print(f"\n-Pickands estimator of gain (log) = {pickands_estimator(X_gain, log(n_gain)):.3f} for k = {int(log(n_gain))}")
print(f"-Pickands estimator of gain (root) = {pickands_estimator(X_gain, root(n_gain)):.3f} for k = {int(root(n_gain))}")
```

Listing 13 – Calculation of the pickands estimators for log and root function

```
-Pickands estimator of losses (log) = -0.509 for k = 6
-Pickands estimator of losses (root) = 0.408 for k = 22
-Pickands estimator of gain (log) = 0.577 for k = 6
-Pickands estimator of gain (root) = 0.720 for k = 22
```

Interpretation

The fact that these two valid functions return opposite signs for the shape parameter ξ of the losses highlights the **instability** of the estimator. While both $k(n)$ functions are theoretically "convergent" at infinity, they clearly fail to provide a stable estimate on our finite sample.

Therefore, we will **delve deeper** into the analysis by plotting the estimator for all $k \in [5, n/4]$ to identify a true **stability**.

```
K_losses = range(5, n_losses//4)  # k must respect < n/4
K_gain = range(5, n_gain//4)

values_loss = [pickands_estimator(X_losses,k) for k in K_losses]
values_gain = [pickands_estimator(X_gain,k) for k in K_gain]

fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharex=True)

axes[0].plot(K_losses, values_loss, marker="o")
axes[0].set_title("Losses")
axes[0].set_ylabel("Pickands estimator")
axes[0].set_ylim(-1, 1)

axes[1].plot(K_gain, values_gain, marker="o")
axes[1].set_title("Gain")
axes[1].set_ylim(-1, 1)

plt.suptitle("Pickands Estimator evolution in function of k", fontsize
    =14, fontweight="bold")
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Listing 14 – Code for plotting the evolution of pickands estimator in function of k

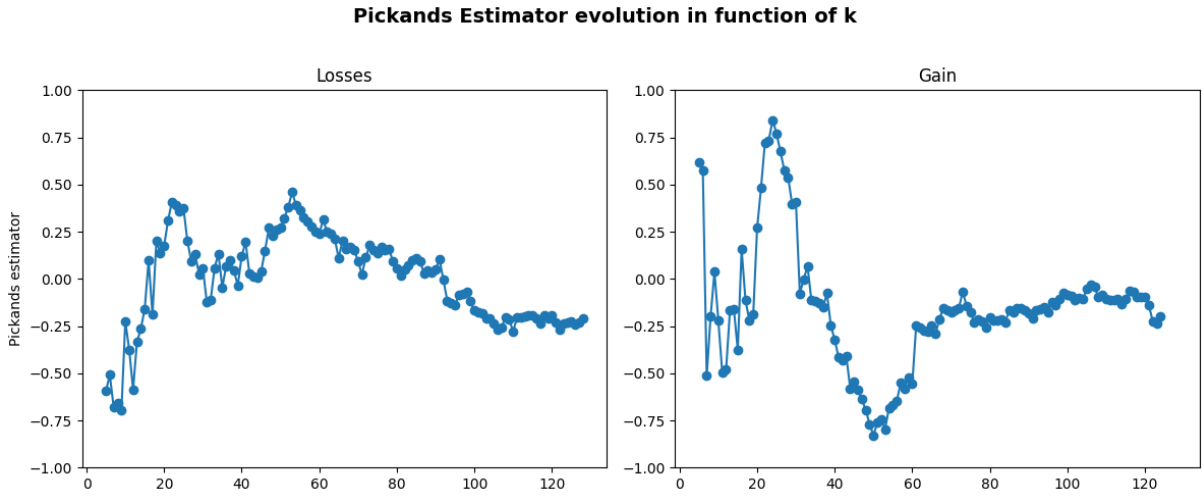

FIGURE 4 – Variation of $\hat{\xi}^P_{k(n),n}$ in function of k for losses and gain

Interpretation

Based on the generated Pickands plots, we can draw the following conclusions regarding the stability of the estimator $\hat{\xi}^P$:

- **Overall Instability** : The curves exhibit significant volatility, particularly for **small values of** k ($k < 20$). This confirms that the estimator is highly sensitive to the sample's noise in the extreme tail, leading to unreliable results for low thresholds.
- **Analysis of Losses** : For the loss distribution, the estimator appears relatively stable within the range $k \in [20, 80]$. However, we observe a notable **spike around** $k \approx 55$, suggesting a local irregularity in the data that could bias the estimation if that specific threshold is chosen.
- **Analysis of Gains** : The behavior for gains is more robust. A **stability plateau** emerges near $k \approx 60$ and extends smoothly without major spikes. Consequently, a threshold k selected between **60 and 120** appears consistent and reliable for estimating the tail index of the gains.

Conclusion for VaR Calculation :

The choice of k is a classic trade-off between **bias** (high k) and **variance** (low k). Given these observations, we will select an "efficient k" within these identified plateaus to perform the final **Value-at-Risk (VaR)** estimation.

k=25 for the losses and 70 for the gain

5.2 b – Calculate the value at risk based on EVT for various confidence levels, with the assumption of iid returns.

According to M. Garcin's course, the Pickands VaR is defined as :

$$VaR(p) = \frac{\left(\frac{k}{n(1-p)}\right)^{\xi^P} - 1}{1 - 2^{-\xi^P}}(X_{n-k+1:n} - X_{n-2k+1:n}) + X_{n-k+1:n}$$

Where :
— ξ^P : Is the Pickands estimator of the GEV parameter.
— n : Total number of observations.
— p : Confidence level (e.g., 0.99).
— k : The chosen threshold (number of upper order statistics).
— $X_{n-k+1:n}$: The (k)-th largest observation.

```python
def Pickands_VaR(X, k, alpha):
    X = X.sort_values(ascending = True)
    n = len(X)
    xi = pickands_estimator(X,k)
    num = (k/(n*(1-alpha)))**xi
    denom = 1-2**(-xi)
    return ((num-1)/denom)*(X.iloc[n-k]-X.iloc[n-2*k])+X.iloc[n-k]

alphas = [0.90, 0.95, 0.975, 0.99, 0.995, 0.999]
k_gain = 70
k_loss = 25

ksi_gain = pickands_estimator(X_gain,k_gain)
ksi_losses = pickands_estimator(X_losses,k_loss)
print(f"EVT Parameter (losses): {ksi_losses:.3f} | EVT Parameter (gains)
    : {ksi_gain:.3f}")
```

Listing 15 – Code for implementing the function to calculate the Pickands VaR

```
EVT Parameter (losses): 0.374 | EVT Parameter (gains): -0.175
```

Based on our stability analysis, we selected $k = 25$ for losses and $k = 70$ for gains, as these values correspond to the plateau where the estimator $\hat{\xi}^P_{k(n),n}$ remains stable. We can therefore conclude that, for the losses, we deal with a Frêchet (heavy) tail and for the gain, a Weibull (bounded support) tail.

We then computed the Pickands VaR for various confidence levels α.

```python
for alpha in alphas:
    var_pick_gain = Pickands_VaR(X_gain, k_gain, alpha)
    var_pick_loss = Pickands_VaR(X_losses, k_loss, alpha)
    print(f"\n-Pickands VaR at alpha={alpha:.3f} Loss = {var_pick_loss
        :.4%} and Gain = {var_pick_gain:.4%})")
```

Listing 16 – Calculation of Pickands VaR for different confidence levels

```
-Pickands VaR at alpha=0.900 Loss = 3.1332% and Gain = 3.2858%)
-Pickands VaR at alpha=0.950 Loss = 4.3796% and Gain = 4.0452%)
-Pickands VaR at alpha=0.975 Loss = 5.9953% and Gain = 4.7180%)
-Pickands VaR at alpha=0.990 Loss = 8.8880% and Gain = 5.4909%)
-Pickands VaR at alpha=0.995 Loss = 11.8394% and Gain = 5.9987%)
-Pickands VaR at alpha=0.999 Loss = 22.5151% and Gain = 6.9652%)
```

Interpretation

First, we observe that the VaR for losses is generally higher than for gains, especially as we move deeper into the tail. While the VaR for gains increases at a relatively moderate and steady pace, the VaR for losses grows much more aggressively for extreme quantiles. This divergence indicates a significant **asymmetry** in the distribution's tails, with losses exhibiting a "fat tail" behavior that is far more pronounced than that of gains.
To better visualize these different behaviors of extreme risks, we will plot the evolution of both VaR estimates as a function of the confidence level α.

```python
alpha_range = np.linspace(0.90, 0.999, 100)

var_losses = [Pickands_VaR(X_losses, k_loss, a) for a in alpha_range]
var_gains = [Pickands_VaR(X_gain, k_gain, a) for a in alpha_range]

plt.figure(figsize=(10, 6))
plt.plot(alpha_range, var_losses, label='VaR Losses (Pickands)', color='red')
plt.plot(alpha_range, var_gains, label='VaR Gains (Pickands)', color='green')

plt.title(f"Pickands VaR Evolution (k_loss={k_loss}, k_gain={k_gain})", fontsize=14)
plt.xlabel("Confidence Level (alpha)")
plt.ylabel("Estimated VaR")
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.legend()

for a in [0.95, 0.99, 0.995]:
    val = Pickands_VaR(X_losses, k_loss, a)
    plt.scatter(a, val, color='black', zorder=5)
    plt.annotate(f'{val:.2%}', (a, val), textcoords="offset points", xytext=(0,10), ha='center')
    val = Pickands_VaR(X_gain, k_gain, a)
    plt.scatter(a, val, color='black', zorder=5)
    plt.annotate(f'{val:.2%}', (a, val), textcoords="offset points", xytext=(0,10), ha='center')

plt.show()
```

Listing 17 – Code for plotting the evolution of pickands estimator in function of k

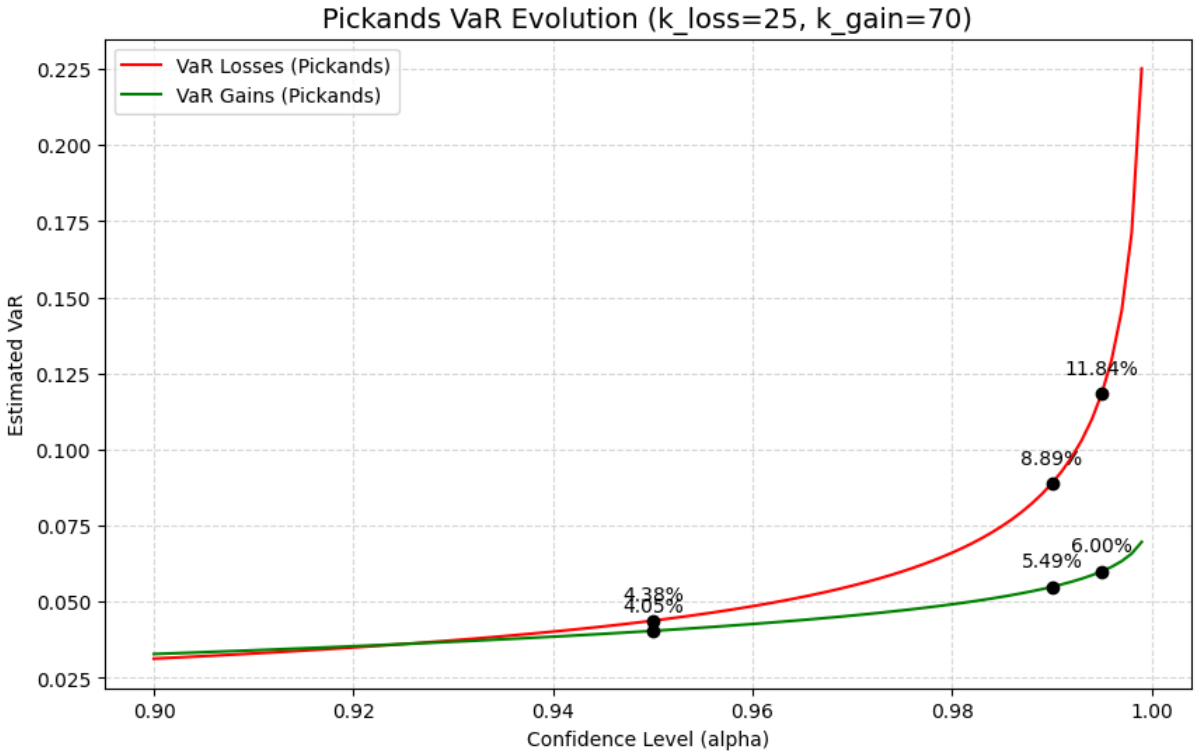

FIGURE 5 – Comparison of Tail VaR for Gains and Losses

Interpretation

We analyze the evolution of the **Pickands VaR** across different confidence levels α. This approach allows us to visualize the model's behavior in the **extreme tail** (from 90% up to 99.9%).

— **Extreme Tail Asymmetry** : While both curves follow a similar path for $\alpha < 0.95$, they diverge significantly in the extreme tail. The **Losses (red curve)** exhibit a much steeper, nearly exponential slope compared to the **Gains (green curve)**.

— **Quantitative Gap** : At the extreme $\alpha = 0.995$ level, the estimated VaR for losses reaches **11.84%**, which is nearly **double** the **6.00%** observed for gains. This gap quantifies the "asymmetry of fear" in the market.

6 Question D (Ex2, Q5 of TD4)

6.1 Bouchaud's Price Impact Model : With the dataset and the framework provided for TD4, estimate all the parameters of Bouchaud's price impact model. Comment the obtained values. Is this model well specified ?

The current price p_t is modeled as the sum of past transaction impacts :

$$p_t = p_{-\infty} + \sum_{s=-\infty}^{t-1} G(t-s)\varepsilon_s S_s V_s^r \tag{3}$$

Where :

— p_t : Current price at time t.
— $p_{-\infty}$: Fundamental price (initial price before impact).
— $G(t-s)$: The **Propagator** function. It is defined as 0 on $\mathbb{R}^-$ and represents the temporal decay of a single order's impact. In practice, it is often modeled as a power-law decay : $G(t) \sim t^{-\gamma}$.
— ε_s : The **sign** of the transaction, where $\varepsilon_s = 1$ for a buy and $\varepsilon_s = -1$ for a sell.
— S_s : The **spread** at the time of transaction s.
— V_s : The **volume** of the transaction.
— r : The exponent representing the **concavity** of the impact. It is typically close to 0 to ensure that marginal impact decreases with volume.

```
df = pd.read_excel("Dataset TD4.xlsx")

#We replace commas with dots in object columns and convert to float
for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].str.replace(',', '.', regex=False).astype(float,
       errors='ignore')

df.rename(columns={'transaction date (1=1day=24 hours)': 'Date',
                   'bid-ask spread': 'Spread',
                   'volume of the transaction (if known)': 'Volume',
                   'Sign of the transaction': 'Sign',
                   'Price (before transaction)': 'Price'}, inplace=True)

print(df.head())
print(df.isna().sum())
print(len(df))

results = []
```

Listing 18 – Code for implementing the Dataset

```
       Date  Spread  Volume  Sign    Price
0  0.000202  0.1100     8.0    -1  100.000
1  0.004074  0.1294    32.0     1  100.164
2  0.014393  0.1141     8.0    -1  100.048
3  0.022861  0.0978   141.0    -1   99.876
```

```
4  0.037864  0.1291    121.0    -1    99.608
Date         0
Spread       0
Volume     688
Sign         0
Price        0
dtype: int64
820
```

We observe that 688 volumes out of 820 transactions are missing. Before removing these rows, we verified whether their exclusion significantly alters the price curve over time.

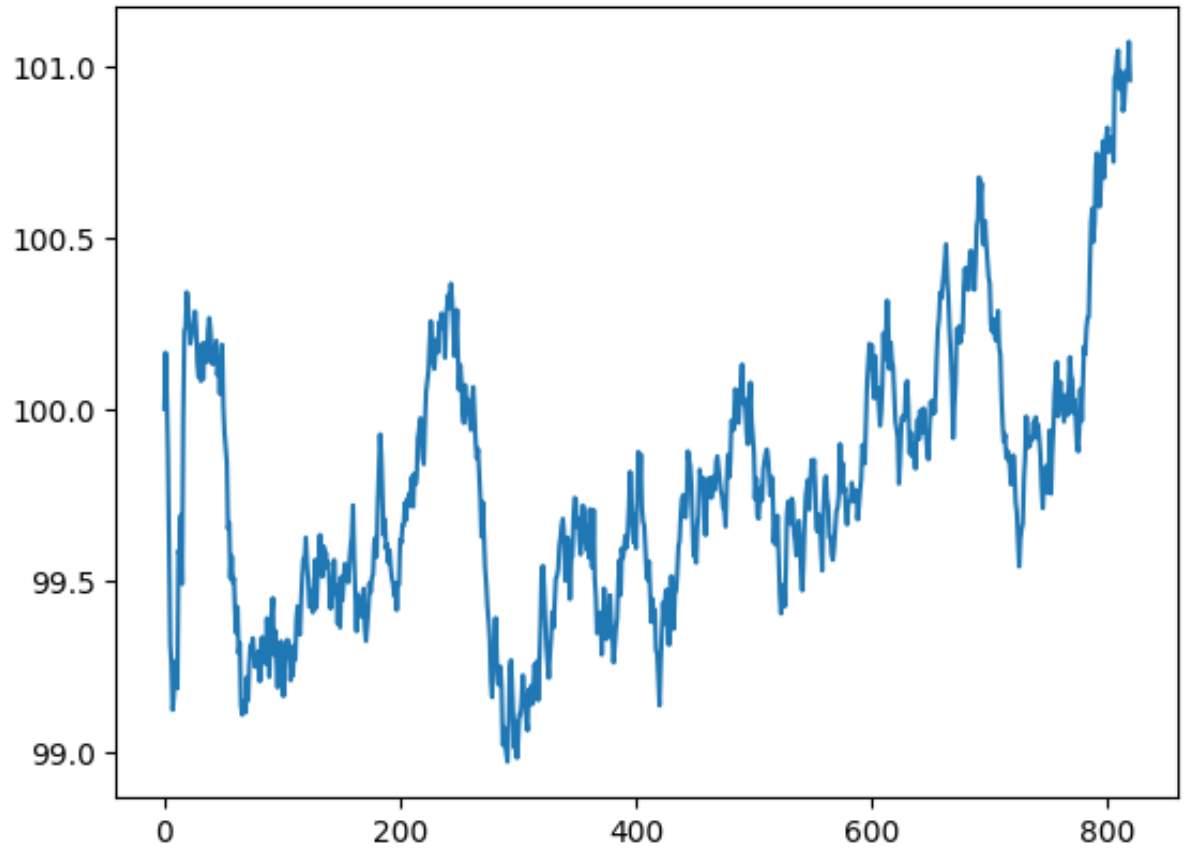

FIGURE 6 – Plot of the price in function of the time without dropping

```python
print(len(df))
df.dropna(inplace=True)
print(df.isna().sum())
print(len(df))

plt.plot(df.index,df["Price"])
```

Listing 19 – Code for dropping the NaN Values

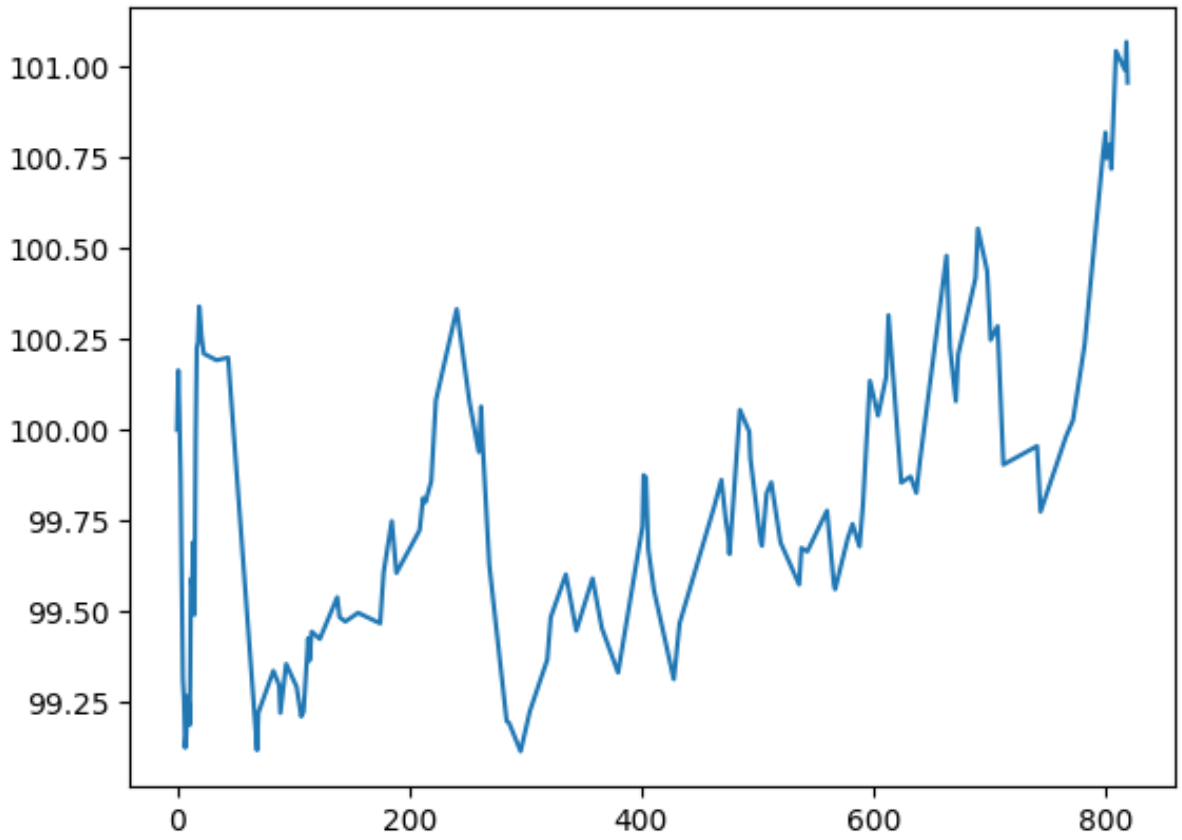

FIGURE 7 – Plot of the price in function of the time dropping volumes NaN

As shown in above, removing the observations with missing volumes does not significantly change the global trend of the price curve. Consequently, we will proceed with the filtered dataset, focusing only the 131 rows without NaN values.

Implementation and Parameter Estimation

First of all, looking at the data, we see that to implement the model, we need to estimate the propagator function G and the exponent r. Before delving into the estimation, we reviewed several papers to guide our approach. We looked at various papers, especially from J.-P. Bouchaud, who has written extensively on how to model the impact of meta-orders linked to their volume.As explained in M. Garcin's course and in paper [2], the model is non-linear due to the exponent r, and the transitory impact function is defined as $G(t) = t^{-\gamma}$ for $t \geq 0$. According to paper [3],[4], r is often approximated as 0.5 to satisfy the "square-root impact law". This ensures concavity, meaning that the marginal impact of volume decreases as the trade size increases.Regarding the temporal aspect, the function G decreases as time passes. This means that transactions closest to the present have a higher impact because $t - s$ is small. Since G is inversely proportional to time, the impact increases as $t-s$ decreases. According to [3], γ is often found to be around 0.8.

We decided to check quickly if the values advanced in the papers would fit correctly our data, therefore we plotted it.

```
best_r, best_gamma = (0.5, 0.8)
p0 = df["Price"].iloc[0]
T_max = len(df)
impacts = df["Sign"] * df["Spread"] * (df["Volume"]**best_r)
final_preds = []
for T in range(2, T_max):
    x_t = sum(((T-t)**-best_gamma) * impacts.iloc[t] for t in range(1, T
        ))
    final_preds.append(p0 + x_t)
plt.figure(figsize=(10, 5))
plt.plot(df.index[2:T_max], df["Price"].iloc[2:T_max], label="Actual␣
    Price", color="black", alpha=0.6)
plt.plot(df.index[2:T_max], final_preds, label="Predicted␣Price", color=
    "red")
plt.title(f"Price␣Prediction␣(r={best_r:.2f},␣gamma={best_gamma:.2f})")
plt.xlabel("Time␣(t)")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 20 – Code for dropping the NaN Values

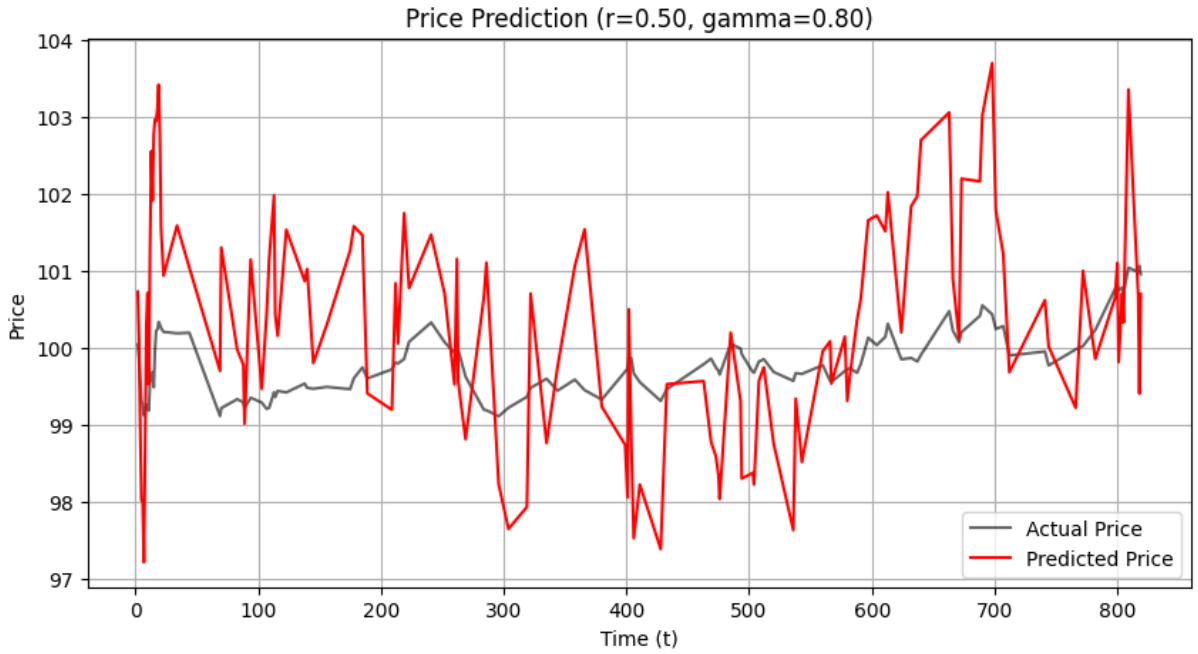

FIGURE 8 – Plot of both predicted price from Bouchaud's model and originaly price, using parameters from the paper

We clearly see that with those parameters, the model's predictions are not good at all. We then, decided to try a GridSearch method using several values for r and γ trying to find the lowest mse

```python
def grid_search(data, grid_r, grid_gamma):
    preds = []
    true = []
    p0 = data["Price"].iloc[0]
    best_mse = np.inf
    best_params = ()

    for r in grid_r:
        for gamma in grid_gamma:

            for T in range(2, len(data)):

                impacts = 0
                for t in range(1, T):
                    impacts += data["Sign"].iloc[t] * data["Spread"].
                       iloc[t] * (data["Volume"].iloc[t]**r) * (T - t)
                       **(-gamma)

                p_hat = p0 + impacts
                preds.append(p_hat)
                true.append(data["Price"].iloc[T])

            mse = np.mean((np.array(true) - np.array(preds))**2)

            if mse < best_mse:
                best_mse = mse
                best_params = (r, gamma, mse)
                #{"r": r, "gamma": gamma, "mse": mse}

    return best_params

grid_r = np.arange(0.1, 0.9, 0.05)
grid_g = np.arange(0.1, 0.9, 0.05)

best_param_grid = grid_search(df, grid_r, grid_g)
print(best_param_grid)

results.append({
    "Method": "␣Grid_Search",
    "r": best_param_grid[0],
    "gamma": best_param_grid[1],
    "MSE": best_param_grid[2]
})
```

Listing 21 – Code for the GridSearch

```
(0.1, 0.8500000000000002, 0.28560282181537294)
```

We then plotted the predictions

```
best_r, best_gamma, _ = best_param_grid
p0 = df["Price"].iloc[0]
T_max = len(df)

impacts = df["Sign"] * df["Spread"] * (df["Volume"]**best_r)

final_preds = []

for T in range(2, T_max):
    x_t = sum(((T - t)**-best_gamma) * impacts.iloc[t] for t in range(1,
        T))
    final_preds.append(p0 + x_t)

plt.figure(figsize=(12, 6))

plt.plot(df.index[2:T_max], df["Price"].iloc[2:T_max], label="Actual␣
    Price", color="black", alpha=0.4)
plt.plot(df.index[2:T_max], final_preds, label="Bouchaud␣Model", color="
    red", linewidth=1.5)

plt.title(fr"Fit␣of␣the␣GridSearch␣(r={best_r},␣$\gamma$={best_gamma}")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Listing 22 – Code for dropping the NaN Values

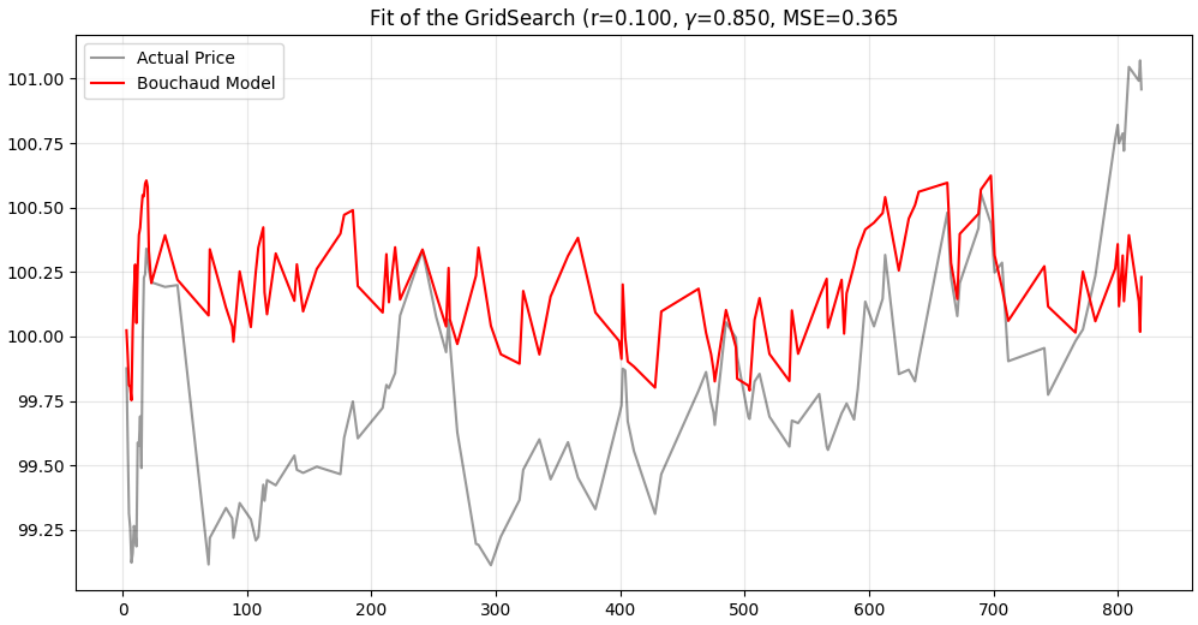

FIGURE 9 – Plot of both predicted price from Bouchaud's model and originaly price, using a Grid Search method

We see that the model has less noise than the previous one. It captures certain movement but is centered around P_0 and has a mean reverting behaviour. Thus, the grid search is a very heavy process. therefore we are going to try another method in order to deduce those parameters

Gradient Descent

Using the order of magnitude for γ and r suggested in the literature as a starting point, we tried to estimate these parameters via Gradient Descent by minimizing the MSE. Since the mean squared error is a **convex function**, we can find a minimum using this optimization method nevertheless as some terms are cuncave (especially the one with the exponent r) we will have to deal with local minimum.

$$MSE(r;\gamma) = \frac{1}{N}\sum_{i=1}^{N}(P_i - \hat{P}_i^B(r;\gamma))^2 \tag{4}$$

With $\hat{P}_i^B$ being the **Bouchaud Price** predicted by the model :

$$\hat{P}_i^B(r;\gamma) = P_0 + \sum_{s=1}^{i-1}(i-s)^{-\gamma}\varepsilon_s S_s V_s^r \tag{5}$$

The parameters are updated iteratively by subtracting the gradient of the MSE, scaled by a learning rate η :

$$\begin{cases} r_{n+1} = r_n - \eta\frac{\partial MSE}{\partial r} \\ \gamma_{n+1} = \gamma_n - \eta\frac{\partial MSE}{\partial \gamma} \end{cases} \tag{6}$$

The parameters r and γ are typically constrained to the interval $[0, 1[$. Specifically, r must be less than 1 to ensure a **concave impact** while a positive γ ensures the propagator $(i-s)^{-\gamma}$ to be a **decreasing function** as the time increases, reflecting the fact that the impact of older transactions are dissipating.

```
def get_mse(data, r, gamma):
    pred = []
    price_true = []
    p0 = data["Price"].iloc[0]
    for T in range(2,len(data)):
        impacts = 0
        for t in range(1,T):
            tau = T - t
            impacts += data["Sign"].iloc[t] * data["Spread"].iloc[t] * (
               data["Volume"].iloc[t]**r) * tau**(-gamma)
        pred.append(p0 + impacts)
        price_true.append(data["Price"].iloc[T])
    return np.mean((np.array(pred)-np.array(price_true))**2)
def get_gradient(data, r, gamma, eps=1e-5):
    current_mse = get_mse(data, r, gamma)
    mse_r_plus = get_mse(data, r + eps, gamma)
    grad_r = (mse_r_plus - current_mse) / eps
    mse_g_plus = get_mse(data, r, gamma + eps)
    grad_g = (mse_g_plus - current_mse) / eps
    return grad_r, grad_g
```

```python
def gradient_descent(data, learning_rate, max_iter,r,gamma):
    best_mse = np.inf
    iter = 0
    best_params = ()
    while iter < max_iter :
        grad_r, grad_gamma = get_gradient(data, r, gamma)
        r = r - learning_rate * grad_r
        gamma = gamma - learning_rate * grad_gamma
        r = max(0, min(0.7, r))
        gamma = max(0, gamma)
        mse = get_mse(data, r, gamma)
        if mse < best_mse:
            best_mse = mse
            best_params = (r, gamma, mse)
        iter += 1
        print(f"{iter} / {max_iter}")
    return best_params
best_param_no_scale = gradient_descent(df, 0.005, 50, 0.5,0.8)
print(best_param_no_scale)
# (data, learning_rate, max_iter, r, gamma)
results.append({
    "Method": "Gradient Descent ",
    "r": best_param_no_scale[0],
    "gamma": best_param_no_scale[1],
    "MSE": best_param_no_scale[2]
})
```

Listing 23 – Classical Gradient Descent without packaging (as it was prohibited)

Then we plotted the results (cf Notebook for code of the plot)

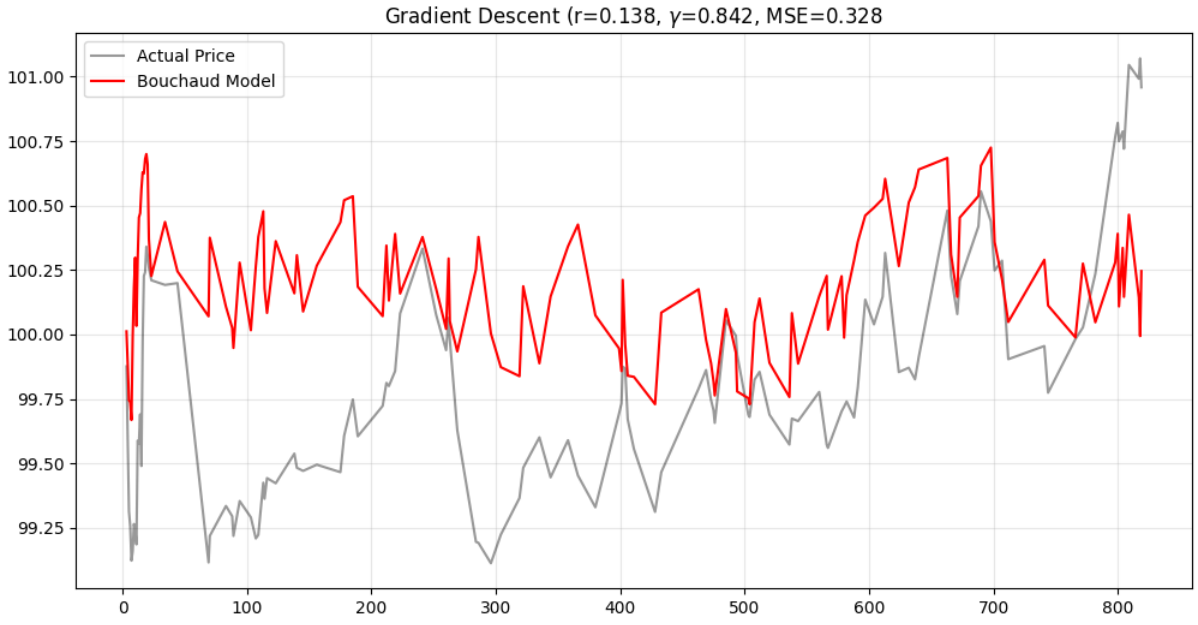

FIGURE 10 – Plot of both predicted price from Bouchaud's model and originaly price, using a Gradient Descent method

We see differences the model overreact to the variation. It is good that the model is less centered around P_0 but it still doesn't fit as we want the true price. Therefore, we will try several other method.

The next plots will be coded in the same way as the previous one for Gradient Descent with juste few changes in the formula that we will highlight here. To see all the codes, refer to the Notebook.

Gradient Descent with scaling factor to be optimized

$$\hat{P}_i^B(r;\gamma) = P_0 + \lambda \sum_{s=1}^{i-1} (i-s)^{-\gamma} \varepsilon_s S_s V_s^r \tag{7}$$

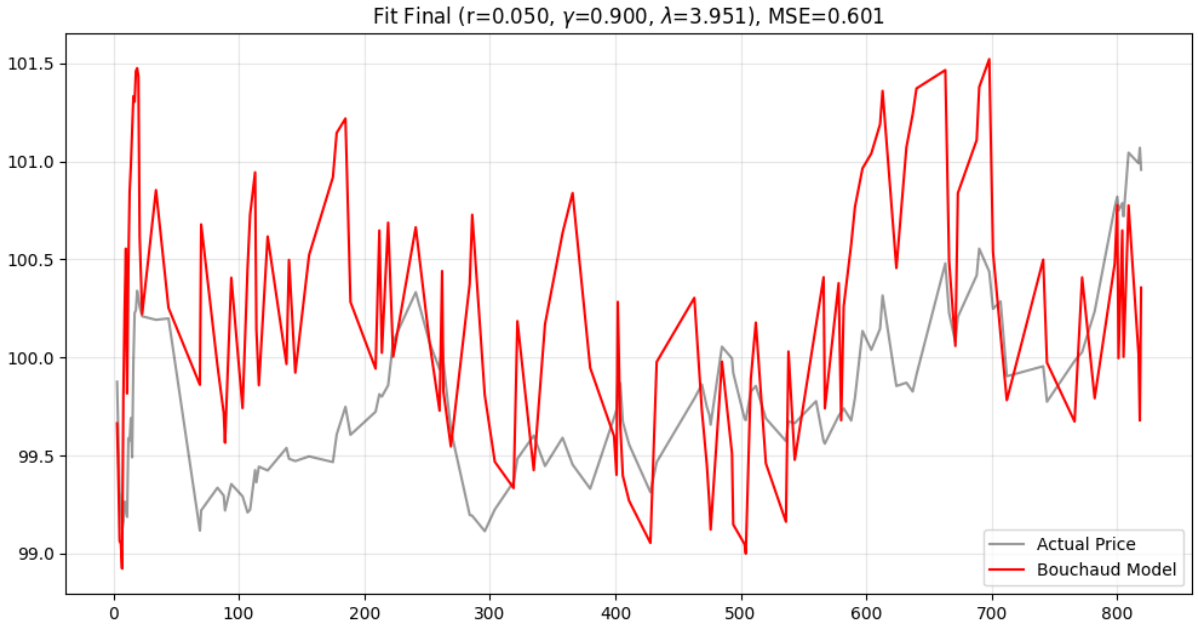

FIGURE 11 – Plot of both predicted price from Bouchaud's model and originally price, using a Gradient Descent method with a scaling factor optimized

We clearly see that multipying by a fixed factor is juste translating our curve with the same proportion every time. Therefore we need something that can by adaptive to follow the trend of the curve. We will introduce the volatility as it will capture the current trend of the curve.

Gradient Descent with volatility as scaling factor

$$\hat{P}_i^B(r;\gamma) = P_0 + \sum_{s=1}^{i-1} \sigma_{[s;i-1]} (i-s)^{-\gamma} \varepsilon_s S_s V_s^r \tag{8}$$

Where $\sigma_{[s;i-1]}$ is the expanding volatility of the price between the first instant s and and i-1 as it has to be in the same unit with P_0.

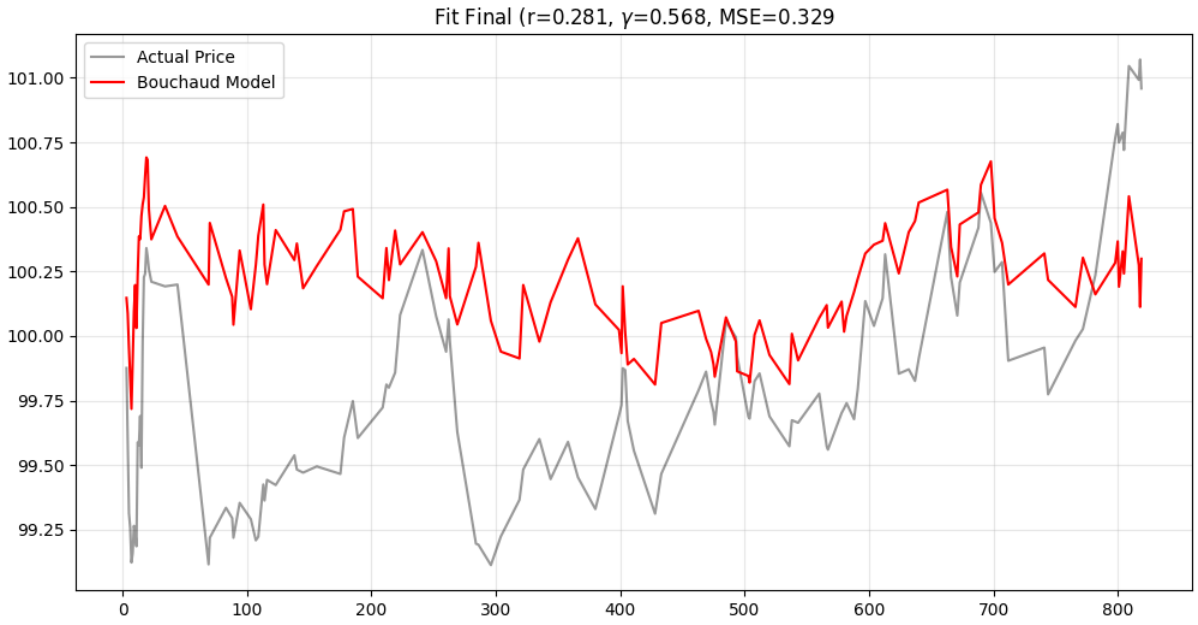

FIGURE 12 – Plot of both predicted price from Bouchaud's model and originally price, using a Gradient Descent method with the volatility of Price as scaling factor

Results comparison

	Method	r	gamma	MSE	lambda
0	Grid_Search	0.100000	0.850000	0.365018	NaN
1	Gradient Descent	0.138234	0.841573	0.328017	NaN
2	Gradient Descent + scaling factor	0.050000	0.900000	0.601315	3.951182
3	Gradient Descent + volatility	0.281006	0.568112	0.329469	NaN

Interpretation

The calibration results show a significant instability of the r and γ parameters across the different methods. While the model follows the general price trend on the charts, it fails to capture micro-volatility and actual market fluctuations.**The model is not well specified** for this dataset especially due to the fact that we try to modelize the impact of order needed for a high frequency aspects and we only have 131 rows which is very low. The parameter instability suggests that the MSE surface is not properly convex and contains many local minima. Moreover, the model is extremely sensitive to the scaling factor λ , the **learning rate** in the Gradient Descent and volatility, making predictions unreliable. However, the results are not entirely useless as the the model systematically find $r < 1$ (concave impact). Bouchaud's model correctly captures the theoretical "shape" of market impact.

7 Question E (Q2 and Q3 of TD5)

First, we imported the datasets for the three FX pairs. For each pair, we calculated a representative price as the arithmetic mean of the **High** and **Low** values provided. We then verified the integrity of the data by checking for missing values (NaN) in each row to ensure no incomplete observations were included in our analysis.

```
gbp_eur=pd.read_excel("Dataset_TD5.xlsx",sheet_name="GBPEUR Curncy")
sek_eur=pd.read_excel("Dataset_TD5.xlsx",sheet_name="SEKEUR Curncy")
cad_eur=pd.read_excel("Dataset_TD5.xlsx",sheet_name="CADEUR Curncy")

gbp_eur['PRICE'] = (gbp_eur["HIGH"] + gbp_eur["LOW"]) / 2
sek_eur['PRICE'] = (sek_eur["HIGH"] + sek_eur["LOW"]) / 2
cad_eur['PRICE'] = (cad_eur["HIGH"] + cad_eur["LOW"]) / 2

print(gbp_eur.isna().sum())
print(sek_eur.isna().sum())
print(cad_eur.isna().sum())
```

Listing 24 – Implementation of the datasets

7.1 a – With Haar wavelets and the dataset provided with TD5, determine the multiresolution correlation between all the pairs of FX rates, using GBPEUR, SEKEUR, and CADEUR (work with the average between the highest and the lowest price and transform this average price in returns on the smallest time step). Do you observe an Epps effect and how could you explain this ?

Next, we computed the arithmetic returns for each currency pair. To ensure the consistency of our time series, we removed the first row of each dataset, as the initial observation does not have a preceding price required for the return calculation. Finally, we checked for any missing values to drop them if necessary.

```
gbp_eur["RETURN"] = gbp_eur['PRICE'].pct_change()
sek_eur["RETURN"] = sek_eur['PRICE'].pct_change()
cad_eur["RETURN"] = cad_eur['PRICE'].pct_change()


gbp_eur = gbp_eur.dropna().reset_index(drop=True)
sek_eur = sek_eur.dropna().reset_index(drop=True)
cad_eur = cad_eur.dropna().reset_index(drop=True)

print(gbp_eur.isna().sum())
print(sek_eur.isna().sum())
print(cad_eur.isna().sum())
```

Listing 25 – Code for plotting the evolution of pickands estimator in function of k

First of all, in the wavelet multiresolution landscape, several conventions exist. We chose to work with the convention where j, the resolution level, is positive. Therefore, in the following code, we consider 2^j to be the number of observations per window. The final correlation results will not be impacted by this choice because, in the correlation formula, the scaling factors in the numerator and denominator simplify each other.

Hence, the purpose is to try different resolution levels j in order to observe how the correlation behaves between Haar wavelets of pairs of FX rates, and ultimately to determine whether the Epps effect is present.**The Epps effect** occurs when increasing the resolution (i.e., using smaller windows with fewer observations) emphasizes real market interactions and microstructure noise rather than the long-term correlation trends seen in larger windows.

This happens because at high resolutions (small windows), we look at very short periods of time. For example, if someone buys GBP, it might take a few minutes before the SEK market reacts and moves too. Because of this small delay, the two currencies don't seem to move together perfectly at first, which breaks the correlation. However, when we look at much larger time scales, this small delay of a few minutes doesn't matter anymore, so the correlation becomes stronger.

Step 1 : Truncating the time series

In our study, the data is sampled every 15 minutes. Therefore, we will work with windows of size $2^j \times 15$ minutes. We will range j from 1 to 9, meaning from $2^1 \times 15$ min ($\delta_t = 30$ min) to $2^9 \times 15$ min ($\delta_t = 128$ h ≈ 5.3 days). Thus, increasing j increases the length of the windows used to calculate the wavelet coefficients via the Haar transform.

Since we are working with different resolution levels ranging from $j = 1$ to $j = 9$, we must truncate our returns based on the maximum window size required for $j = 9$, which is the most restrictive scale. If we were to use different data lengths for each scale, our wavelet coefficients would be calculated on different underlying return samples, leading to inconsistencies.

For example, at $j = 9$, the window size is $2^9 = 512$ observations. With a total of 12,929 data points, we can form $12,929//512 = 25$ full windows (25 wavelet coefficients). This corresponds to a total of $25 \times 512 = 12,800$ observations.In contrast, at $j = 1$, the window size is $2^1 = 2$. The number of possible windows would be $12,929//2 = 6,464$, using $2 \times 6,464 = 12,928$ observations.

Consequently, at $j = 1$, we would be using returns that are not included in the $j = 9$ analysis, which could negatively impact the comparability of our model across scales. To avoid this, we fix the data length at $25 \times 512 = 12,800$ points for all scales, ensuring that we work with the exact same amount of data regardless of the resolution level.

```
n = len(gbp_eur)
size_max_window = 2**9
nb_max_window = n // size_max_window
n_truncated = nb_max_window * size_max_window

returns_gbp_eur = gbp_eur['RETURN'].iloc[-n_truncated:].values
returns_sek_eur = sek_eur['RETURN'].iloc[-n_truncated:].values
returns_cad_eur = cad_eur['RETURN'].iloc[-n_truncated:].values
```

Listing 26 – Code truncating the datasets

We sliced the data to focus on the latest observations rather than simply keeping values from the beginning of the series (from 0 to $n_{truncated}$). We decided to start from the end of the dataset because the most recent data is generally more relevant for current market analysis than older observations.

Step 2 : Correlation Function

Next, we defined the correlation function, which we will use to determine the relationships between the different FX pairs.

The correlation is defined by :

$$\rho(X,Y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

```
def correlation(list1, list2):
    if len(list1) != len(list2):
        return None

    n = len(list1)
    m1 = sum(list1) / n
    m2 = sum(list2) / n
    numerator = 0
    s_squared_1 = 0
    s_squared_2 = 0

    for i in range(n):
        diff1 = list1[i] - m1
        diff2 = list2[i] - m2

        numerator += diff1 * diff2
        s_squared_1 += diff1 ** 2
        s_squared_2 += diff2 ** 2

    denominator = (s_squared_1 * s_squared_2) ** 0.5

    if denominator == 0:
        return 0

    return numerator / denominator
```

Listing 27 – Code of the Correlation function

Step 3 : Wavelet Implementation

Then, it is important to recall the main function we used to determine the wavelet coefficients, which is taken from M. Garcin's course.

First of all, we had to apply the Haar transform to each of our windows. This is expressed by the formula :

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$

```
def haar_transfrom(window_returns, size_window):
    coef = 0
    for t in range(size_window):
        if t < size_window // 2:

            coef += window_returns[t]
        else:
            coef -= window_returns[t]

    return coef
```

Listing 28 – Code of the Correlation function

In our case, we took each window and applied this "door" function. In practice, this means that the coefficient for a window is obtained simply by adding the first half and subtracting the second half of the returns.

Next, it needed to be scaled to the new resolution using the scaling function :

$$\phi_{j,k}(t) = 2^{j/2}\psi(2^j t - k)$$

So, for each coefficient obtained by the Haar transform, we multiplied it by $2^{j/2}$ to put it on the correct scale.

Nevertheless, we faced a choice while obtaining the wavelet coefficient lists : should we work window by window or by incrementing the window one step at a time ? In other words, we tried two different methods : the non-overlapping window method and the overlapping method (incrementing by one). We noticed that for non-overlapping coefficients at scale j, we only had a few coefficients. Therefore, the overlapping method allows us to obtain more coefficients(not the same amount for all resolution levels j, but significantly more overall.)

For example, at scale $j = 9$, we had only 25 windows (and thus 25 coefficients) with the non-overlapping method. However, with the overlapping method, we had $len(returns) - size\_window + 1 = 12800 - 512 + 1 = 12289$ coefficients, which is significantly higher.

```python
def wavelet_coefficients_non_overlapping(returns, j):
    size_window = 2**j
    nb_window = len(returns) // size_window

    coefficients = []

    for i in range(nb_window):
        window = returns[i * size_window: (i+1)*size_window]
        coef = haar_transfrom(window, size_window)
        coefficients.append(coef)

    return np.array(coefficients) * size_window**0.5
```

Listing 29 – Code for the wavelet coefficients using the overlapping method

```python
def wavelet_coefficients_overlapping(returns, j):
    size_window = 2**j

    coefficients = []
    for i in range(len(returns) - size_window + 1):
        window = returns[i : i + size_window]
        coef = haar_transfrom(window, size_window)
        coefficients.append(coef)

    return np.array(coefficients) * size_window**0.5
```

Listing 30 – Code for the wavelet coefficients using the overlapping method

Step 4 : Matrix Implementation

Finally, we had to create a function to compile all the coefficients for each scale into a matrix. Since we were working with 3 FX rate pairs, the matrix is naturally 3x3.

```python
def correlation_matrix_at_level(list_wave_coef):
    matrix_size = len(list_wave_coef)
    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:
                corr = correlation(list_wave_coef[i], list_wave_coef[j])
                correlation_matrix[i, j] = correlation_matrix[j, i] =
                    corr

    return correlation_matrix
```

Listing 31 – Code for implementing the matrix

Step 5 : Analyzing the results

Finally, we plotted the correlation of the three FX rates across the different resolution levels.

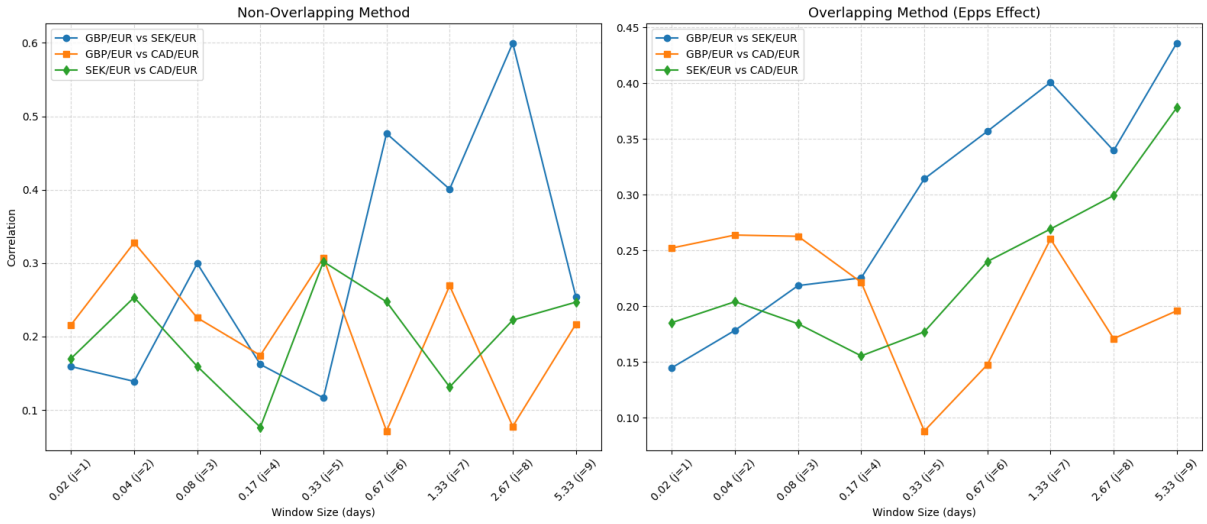

FIGURE 13 – Correlation evolution for different resolution windows

Interpretation

Finally, these curves emphasize several things.

First of all, for the non-overlapping method, we generally see a lot of instability ; the curves often cross each other, increasing then decreasing, and so on. This is due to the fact that as j increases, there are fewer wavelet coefficients, making the results subject to small changes. Moreover, we don't clearly see that augmenting the window size increases the correlation as the Epps effect would imply. Therefore, with this method, **we cannot conclude that an Epps effect exists**.

On the other hand, for the overlapping method, we can see that each curve has its own trend. Even though the orange curve shows some spikes at a few scales, the blue and green ones remain pretty constant, highlighting better stability because the number of wavelet coefficients is significant at every scale. Moreover, we clearly see(especially for the blue and green curves) that for bigger windows, the correlation clearly increases. **There is Epps effect**.

Finally, we could say that the CAD is not very stable compared to the others. The Canadian dollar is in a different timezone and is very correlated with the commodity market, especially oil. Therefore, the commodity market can be impacted while the currency market is not, which could explain why it is not very stable (even though the green curve (CADEUR vs SEKEUR) still seems better than CADEUR vs GBPEUR).

7.2 b – Calculate the Hurst exponent of GBPEUR, SEKEUR, and CADEUR. Determine their annualized volatility using the daily volatility and Hurst exponents.

According to M.Garcin course, we have those expressions for the Hurst exponent : We know that to approximate the Hurst exponent, we can use the following formula :

$$\hat{H} = \frac{1}{2}\log_2\left(\frac{M_2'}{M_2}\right)$$

where :

$$M_2' = \frac{2}{NT}\sum_{i=1}^{NT/2}\left|X\left(\frac{2i}{N}\right) - X\left(\frac{2(i-1)}{N}\right)\right|^2 \qquad M_2 = \frac{1}{NT}\sum_{i=1}^{NT}\left|X\left(\frac{i}{N}\right) - X\left(\frac{i-1}{N}\right)\right|^2$$

```python
def hurst_exponent_estimation(X):
    M_2 = 0
    M_2_prime = 0
    N = len(X)

    for i in range(1,N) :
        M_2 += (X.iloc[i] - X.iloc[i-1])**2

    M_2 = M_2 * (1/N)

    for i in range(1,N // 2) :
        M_2_prime += (X.iloc[2*i] - X.iloc[2*(i-1)])**2

    M_2_prime = M_2_prime * (2/N)

    if M_2_prime == 0:
            print("Error: The denominator M_2_prime is zero.")
            return np.nan

    return 0.5*np.log2(M_2_prime / M_2)
```

Listing 32 – Function for estimate the Hurst exponent

Then we calculated the three estimated Hurst exponent with the code :

```python
Hurst_GBPEUR = hurst_exponent_estimation(gbp_eur["PRICE"])
Hurst_SEKEUR = hurst_exponent_estimation(sek_eur["PRICE"])
Hurst_CADEUR = hurst_exponent_estimation(cad_eur["PRICE"])

print(f"Hurst exponent estimated for GBP/EUR :{Hurst_GBPEUR}")
print(f"Hurst exponent estimated for SEK/EUR :{Hurst_SEKEUR}")
print(f"Hurst exponent estimated for CAD/EUR :{Hurst_CADEUR}")
```

Listing 33 – Calulation of the Hurst exponents

```
-Hurst exponent estimated for GBP/EUR :0.6713731707231525
-Hurst exponent estimated for SEK/EUR :0.6545913434209861
-Hurst exponent estimated for CAD/EUR :0.6552402726566733
```

Interpretation

Based on our estimated Hurst exponents ($\hat{H}_{GBP/EUR} \approx 0.6714$, $\hat{H}_{SEK/EUR} \approx 0.6546$, and $\hat{H}_{CAD/EUR} \approx 0.6552$), we can conclude that the returns are not independent since $H \neq \frac{1}{2}$.

Specifically :

— **Persistence** : Since $H > \frac{1}{2}$, the returns exhibit **positive correlation**, which indicates a **long-memory** behavior in the time series.

— **Non-Linear Scaling** : Rather than using the standard square root of time rule (which assumes $H = 0.5$), we scale the volatility using the estimated Hurst exponent :

$$\sigma_{annual} = \sigma_{15min} \cdot (252 \times 8.5 \times 4)^{\hat{H}}$$

Where :

— H : Hurst exponent.

— 252 : Number of trading days in a year.

— 8.5 : Hours where the market is open in a trading day .

— 4 : Number of 15-minute periods in one hour.

```python
def annualized_volatility(returns, hurst_exponent=False):
    if hurst_exponent:
        return returns.std() *(252 * 8.5 * 4) ** hurst_exponent
    else:
        return returns.std() *(252 * 8.5 * 4) ** 0.5
annualy_vol_GBPEUR = annualized_volatility(gbp_eur["RETURN"],
    Hurst_GBPEUR)
print(f"Annualized volatility for GBPEUR: {annualy_vol_GBPEUR:.6f} with
    Hurst exponent")
annualy_vol_SEKPEUR = annualized_volatility(sek_eur["RETURN"],
    Hurst_SEKEUR)
print(f"Annualized volatility for SEKEUR: {annualy_vol_SEKPEUR:.6f} with
    Hurst exponent")
annualy_vol_CADPEUR = annualized_volatility(cad_eur["RETURN"],
    Hurst_CADEUR)
print(f"Annualized volatility for CADEUR: {annualy_vol_CADPEUR:.6f} with
    Hurst exponent")
annualy_vol_GBPEUR = annualized_volatility(gbp_eur["RETURN"])
print(f"\nAnnualized volatility for GBPEUR: {annualy_vol_GBPEUR:.6f}
    without Hurst exponent")
annualy_vol_SEKPEUR = annualized_volatility(sek_eur["RETURN"])
print(f"Annualized volatility for SEKEUR: {annualy_vol_SEKPEUR:.6f}
    without Hurst exponent")
annualy_vol_CADPEUR = annualized_volatility(cad_eur["RETURN"])
print(f"Annualized volatility for CADEUR: {annualy_vol_CADPEUR:.6f}
    without Hurst exponent")
```

Listing 34 – Code for the annualized volatility

```
Annualized volatility for GBPEUR: 0.272338 with Hurst exponent
Annualized volatility for SEKEUR: 0.122786 with Hurst exponent
Annualized volatility for CADEUR: 0.191148 with Hurst exponent

Annualized volatility for GBPEUR: 0.057692 without Hurst exponent
Annualized volatility for SEKEUR: 0.030280 without Hurst exponent
Annualized volatility for CADEUR: 0.046863 without Hurst exponent
```

Interpretation

The results highlight a real difference between the standard Square Root ($H = 0.5$) and the Hurst scaling. Since our estimated Hurst exponents are all approximately 0.67, we observe a strong **persistence** in FX returns, meaning that volatility scales much faster than a simple random walk would suggest.

— **Underestimation of Risk** : Using the standard $\sqrt{T}$ rule, the annualized volatility for GBPEUR is only **5.77%**. However, when accounting for the long-memory property, the volatility jumps to **27.2%**.

— **Persistence Effect** : This gap (a factor of nearly 5x) quantifies the impact of **positive autocorrelation**. In these markets, price movements tend to trend in the same direction, amplifying the total dispersion over a one-year horizon.

In conclusion, ignoring the Hurst exponent would lead to a severe **underestimation of market risk**. The standard model fails to capture the trend-following behavior identified in these currencies, whereas our multi-scale approach provides a much more realistic risk assessment.

8 References

-Bandwidth :
-https://openturns.github.io/openturns/latest/theory/data_analysis/kernel_smoothing.html
- https://bookdown.org/egarpor/NP-UC3M/kde-i-bwd.html

-Gradient Descent :
- https://www.youtube.com/watch?v=rcl_YRyoLIY
- https://datascientest.com/descente-de-gradient

-Bouchaud's model :
- https://ideas.repec.org/p/arx/papers/1602.02735.html[1]
- https://arxiv.org/pdf/1407.3390[2]
- https://www.cis.upenn.edu/~mkearns/finread/costestim.pdf[3]
- https://www.cfm.com/wp-content/uploads/2022/12/287-2016-The-square-root-impact-law-a
pdf[4]