
École supérieure d'ingénieurs Léonard de Vinci

# Market Risk – Project – 2025-2026

***Student :***
Mathis Fourreau
Natacha Gaussin

***Class:***
IF3

# Table des matières

# 1 Introduction

This document presents the report of the Market Risk project. All formulas and concepts used in this report come from the course material or from the cited references.

The code was written in Python using a Jupyter Notebook. For more details on the implementation, please refer to the notebook.

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

## 2.1 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 $K$ is the derivative of the logistic function $x \mapsto \frac{15}{16}(1 - x^2)^2 \mathbf{1}_{|x|\leq 1}$.

First, recall the definition of the Value at Risk for a given small $\alpha$ from the course :

$$\mathrm{VaR}_\alpha(X) = -F_X^{-1}(\alpha)$$

where X denotes a return.

The aim is to estimate a historical VaR on a non-parametric distribution with Biweight kernel. To achieve this, we followed these steps :

### 2.1.1 Compute and Extract returns from January 2015 to December 2016

The return is calculated as such :

$$r_t = \frac{P_t}{P_{t-1}} - 1$$

where r is the return at time t, and P is the price of the stock.

```
df["return"] = df["value"] / df["value"].shift(1) - 1
```

FIGURE 1 – Computation of the returns in python

Moreover, the dataset references the prices of the stock Natixis from 2015 to 2018 so we extracted only the returns from the interval we are studying : 2015-2016.

```
returns_2015_2016 = df[df["date"] < "2017"].dropna().loc[:, "return"]
```

FIGURE 2 – Extraction of returns from 2015 to 2016

### 2.1.2 Estimate the kernel density

The kernel density function is given in the course by :

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

and from the wording, we have that

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

The aim of this step is to compute the kernel density function to then choose the appropriate value of the bandwidth, h.

To achieve this goal, we computed the kernel density function for multiple values of x. The values of x were chosen on the basis of the kernel value. Indeed, the kernel is not null when

$$|\frac{x - X_i}{h}| \leq 1$$

with $X_i$ being the return at time $i$. Then, we can deduce that for each $X_i$, the kernel is not null for

$$X_i - h \leq x \leq X_i + h$$

To ensure that the density is computed on a wide range of data and to better observe the global shape of the density, we chose a wide interval for values of x :

$$[\min_{1 \leq i \leq N} X_i - 5h; \max_{1 \leq i \leq N} X_i + 5h]$$

Finally, we chose to estimate the density for 1000 values of x in the interval.

```python
#def K(x)
def K(x):
    K_tab = []
    for i in x:
        if i <= 1 and i >= -1:
            K_tab.append((15/16) * (1 - i**2)**2)
        else:
            K_tab.append(0)
    return K_tab

# def f_hat(x, h, tab_returns)
def f_hat(x, h, tab_returns):
    return sum(K((x - tab_returns) / h)) / (len(tab_returns) * h)

# estimate 1000 times f_hat
def estimate_f_hat(tab_returns, h, nb_estimations = 1000):
    x_tab = np.linspace(min(tab_returns) - 5*h, max(tab_returns) + 5*h, nb_estimations)
    f_hat_tab = []
    for x in x_tab:
        f_hat_tab.append(f_hat(x, h, tab_returns))

    return f_hat_tab, x_tab
```

FIGURE 3 – Estimation of the kernel density

### 2.1.3 Choice of h

To estimate h, we used Scott's rule and Silverman's rule that are computed as follow :

$$h_{\text{Scott}} = \sigma\, n^{-1/5}$$

$$h_{\text{Silverman}} = 0.9\, \min\left(\sigma,\, \frac{\text{IQR}}{1.34}\right) n^{-1/5}$$

with $\sigma$ the standard deviation of the returns and IQR the interquantile.

The computation of each rule provided the following results :

— Scott's rule : h = 0.006853

— Silverman's rule : h = 0.005443

The bandwidth must be chosen so that the kernel density must display a good ratio between smoothness and noise. To do so, each kernel density is displayed.

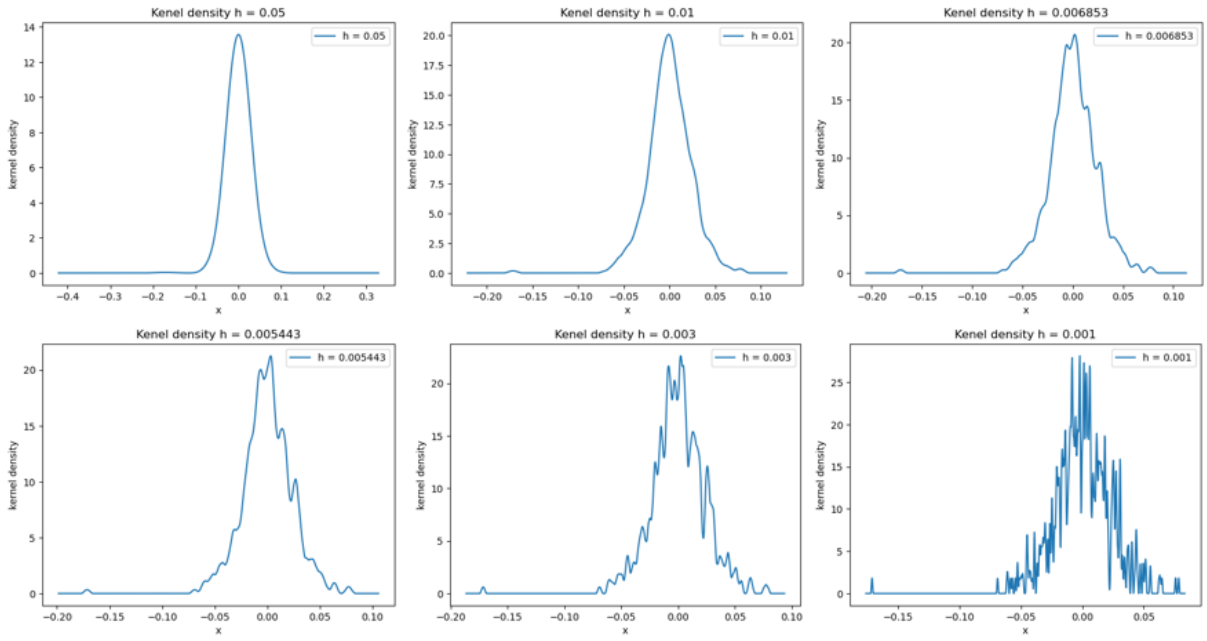



FIGURE 4 – Kernel density representation for each h

From these graphs, we observe that the kernel density with h = 0.001 is very noisy. On the contrary, for higher h, the kernel density is too smooth leading to loss of information. Some values of h seem to display a good ratio between noise and smoothness : h = 0.003, h = 0.005443 (Silverman's rule) and h = 0.006853 (Scott's rule).

Thus, we chose to continue the study with those three h.

### 2.1.4 Compute the Value at Risk with de CDF

The kernel cumulative distribution function is given in the course as follow :

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

by applying the primitive on the given kernel we define $\mathcal{K}$ as :

$$\mathcal{K}(x) = \begin{cases} 0 & \text{if } x < -1 \\ \frac{15}{16}\left(x - \frac{2x^3}{3} + \frac{x^5}{5} + \frac{8}{15}\right) & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$

*Démonstration.* Proof of the primitive : we know that :

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

we want to compute its primitive such that :

$$\mathcal{K}(x) = \int_{-\infty}^{x} K(t)\,dt$$

Study for $|x| \leq 1$ :

$$(1-x^2)^2 = 1 - 2x^2 + x^4$$

So its primitive is :

$$x - \frac{2x^3}{3} + \frac{x^5}{5}$$

By multiplying by $\frac{15}{16}$, we obtain on the interval $|x| \leq 1$ the primitive :

$$\mathcal{A}(x) = \frac{15}{16}(x - \frac{2x^3}{3} + \frac{x^5}{5}) + C$$

with $C$ the integration constant.

**case 1 :** $x < -1$ **:** On this interval, the kernel density is equal to 0, so

$$\mathcal{K}(x) = 0$$

**case 2 :** $x \in [-1, 1]$ **:**

$$\mathcal{K}(x) = \int_{-1}^{x} K(t)\,dt$$

So

$$\mathcal{K}(x) = \mathcal{A}(x) - \mathcal{A}(-1) = \frac{15}{16}(x - \frac{2x^3}{3} + \frac{x^5}{5}) + \frac{1}{2}$$

So on this interval $x \in [-1, 1]$, the cdf, $\mathcal{K}$, is

$$\frac{15}{16}\left(x - \frac{2x^3}{3} + \frac{x^5}{5} + \frac{8}{15}\right)$$

**case 3 :** $x > 1$ **:**

$$\mathcal{K}(x) = \int_{-1}^{1} K(t)\,dt = 1$$

Thus, we obtain the primitive of the kernel, $\mathcal{K}$. □

To compute the Value at Risk (whose formula is given on page 3), we first estimated the cdf of the kernel for multiple values of x from the interval $[\min_{1\leq i\leq N} X_i - 5h; \max_{1\leq i\leq N} X_i + 5h]$. Then, we computed the cumulative distribution function for each points of the x grid. Since, the value at risk corresponds to a quantile of the distribution, it was obtained by identifying the smallest value of $x$ such that the cumulative density function is above or equal to $\alpha$. Finally, the VaR is given by applying $-1$ to this quantile.

```
def K_cdf(x):
    if x <= 1 and x >= -1:
        return (15/16)*(x - 2*(x**3)/3 + (x**5)/5 + 8/15)
    elif x < -1:
        return 0.0
    else:
        return 1.0
```

FIGURE 5 – Function that computes the cdf of K for a given x

```
def F_hat(x, h, returns_tab):
    u_tab = (x-returns_tab)/h
    return sum(K_cdf_vector(u_tab)) / len(returns_tab)
```

FIGURE 6 – Function that computes the cumulative distribution function for a given x

Let's check the distribution of the cumulative distribution function :

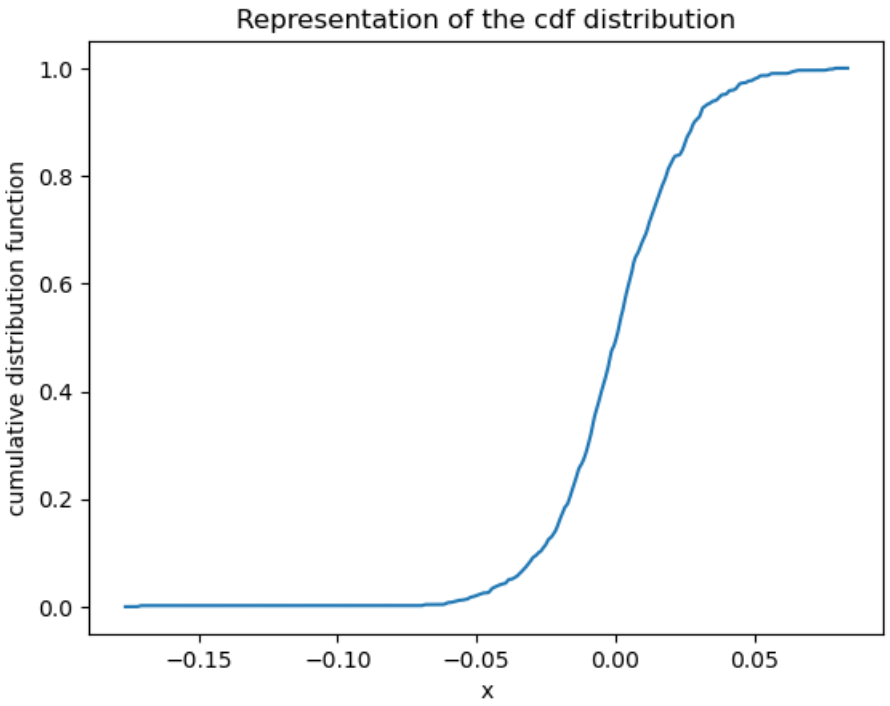



FIGURE 7 – Representation of the cdf distribution

The estimated cumulative distribution function shows the shape of a classic cdf. Thus, we can validate our function and the number of estimations point (F estimated with 1000 x) seems sufficient.

```
def ComputeVaR(returns_tab, alpha, h, nb_estimations=1000):
    x_tab = np.linspace(min(returns_tab) - 5*h, max(returns_tab) + 5*h, nb_estimations)
    F_hat_tab = Series_F_Hat(h, returns_tab, x_tab)
    i = 0
    while i < len(F_hat_tab):
        if F_hat_tab[i] >= alpha:
            return - x_tab[i]
        i+=1

    return - x_tab[len(x_tab) - 1]
```

FIGURE 8 – Function that computes the Value at Risk

```
def ComputeMultipleHVaR(returns_tab, alpha, h_tab):
    dico_var = {}
    for h in h_tab:
        dico_var[h] = ComputeVaR(returns_tab, alpha, h)
    return dico_var
```

FIGURE 9 – Function that computes the VaR for multiple h

We obtained the following results :
— for h = 0.003, VaR = 0.057508 with alpha = 0.01
— for h = 0.005443, VaR = 0.057256 with alpha = 0.01
— for h = 0.006853, VaR = 0.057439 with alpha = 0.01

The Value at Risk for a fixed alpha changes with respect to the value of the bandwidth h. Moreover, there is no clear relationship between h and VaR since h affects the whole density distribution.

In addition, we observe that the value at risk only varies a little based on h. This is due to the choice of the studied bandwidths whose values are close.

### 2.1.5 Choice of the VaR and Results

To choose the best bandwidth applied to our senario, we studied different research papers. The Silverman's rule focus on the density function global shape and cope with non-normal desities. In addition, when the data is unimodal and not symmetric, the silverman's rule well performs. In our case, the density verifies these conditions. Indeed, the histogram shows that the data is unimodal meaning that the distribution operates in one regime (the density has one peak and the data is mostly gathered into one cluster), the density in also non-symmetric and non-normal.

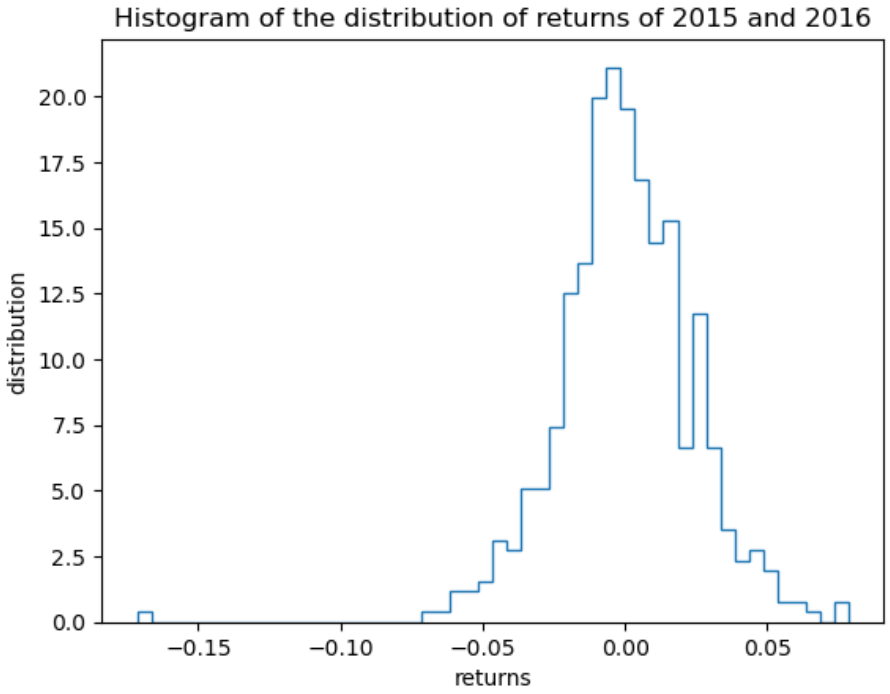



FIGURE 10 – Histogram of the distribution fo returns from 2015 to 2016

This Histogram shows that the distribution only has one visible peak. Thus the distribution is unimodal. In addition, it is clearly visible that the distribution is not symmetric around the central peak. Finally, we can observe heavy tails which validates the non-normal conditions.

Moreover, choosing the Value at Risk, based on Silverman's rule (h = 0.005443), we can verify the proportion of returns that exceed the Value at Risk. This proportion is equal to around 0.0097656 which is approximately equal to $\alpha$ ($\alpha = 0.01$) meaning that the VaR is consistent with the confidence level $\alpha = 0.01$.

Let's apply the backtesting method to validate the Value at Risk. First, let's check the coverage : the proportion of values that exceed the VaR must be close to alpha.

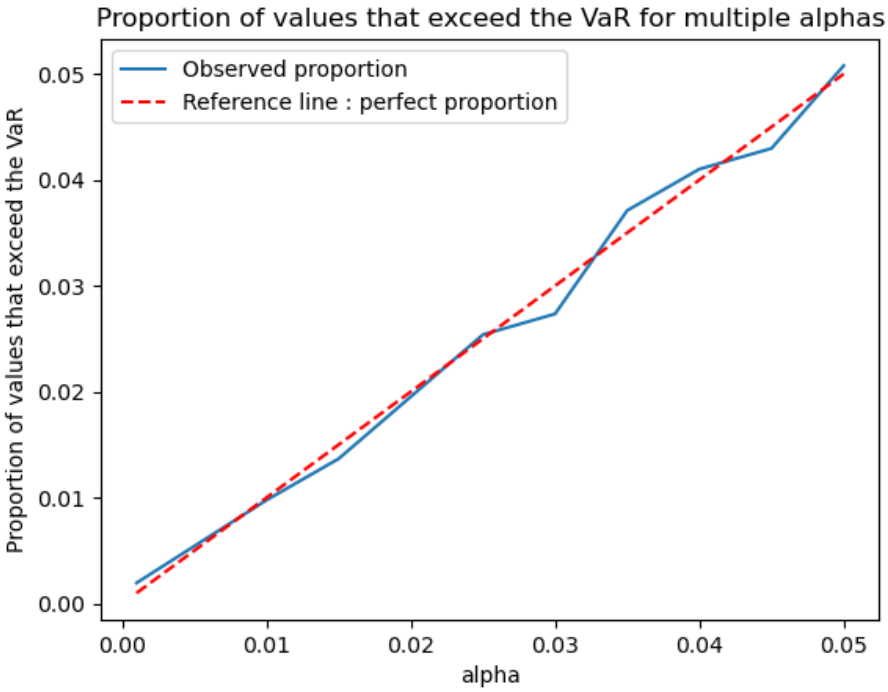



FIGURE 11 – Proportion of values that exceed the VaR for multiple alphas

The curve of the observed proportions as a function of alpha forms an almost perfectly linear line. We can therefore consider the VaR consistent for multiple values of $\alpha$.

Moreover, the independency (violations of VaR must not be in a cluster) is validated as only one value exceed the VaR.

To conclude, the distribution verifies the condition of the Silverman's rule and the VaR computed from the Silverman's bandwidth is consistent. That is why we chose the VaR associated with the bandwidth calculated from Silverman's rule. We thus obtain :

$$VaR = 0.057256$$

with

$$h = 0.005443$$

and $\alpha = 0.01$

# Références

[1] Netti Herawati, Khoirin Nisa, Eri Setiawan. (2017). *The optimal bandwidth for kernel density estimation of skewed distribution : a case study on survival data of cancer patients.*

[2] Jared K. Harpole (2013). *How Bandwidth Selection Algorithms Impact Exploratory Data Analysis Using Kernel Density Estimation.*

## 2.2 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

After extracting only the returns from January 2017 to December 2018, to find the proportion of returns of 2017 and 2018 that exceed the VaR threshold, we computed the frequency of the number of return whose value where smaller than the opposite of the VaR.

```
def Proportion(df, VaR):
    return df[df < -VaR].count() / df.count()
```

FIGURE 12 – Proportion of returns that exceed the VaR

The proportion can then be interpreted such that :
— if proportion $> \alpha$, then VaR underestimates the risk
— if proportion $= \alpha$, then VaR is correct
— if proportion $< \alpha$, then VaR overestimates the risk

| **h** | **VaR** | $\alpha$ | **proportion** | **interpretation** |
|---|---|---|---|---|
| 0.003 | 0.057508 | 0.01 | 0.001961 | VaR overestimates the risk |
| 0.005443 | 0.057256 | 0.01 | 0.001961 | VaR overestimates the risk |
| 0.006853 | 0.057439 | 0.01 | 0.001961 | VaR overestimates the risk |

TABLE 1 – Proportion results and interpretation

For each h, the Value at Risk **overestimates** the risk. Thus, the chosen value at risk ($VaR = 0.057256$ computed with Silverman's rule) overestimates the risk.

Moreover, the size of the data is relatively small (only 510 rows). Thus the number of values that exceed the VaR is small. That could explain why for small variation of the VaR the proportion of values that exceed the VaR stay the same. Backtesting does not allow us to validate the value at risk. In addition, the proportion of values that exceed the VaR is approximatly 0.001961, knowing that the size of the dataset is 510, this value corresponds to $1/510$. This indicates that the value at risk is exceeded exactly one time for each value at risk.

We can observe that statement on the graph bellow, the VaR threshold represented by the vertical line : there is one instance that is on the left side of the VaR threshold.

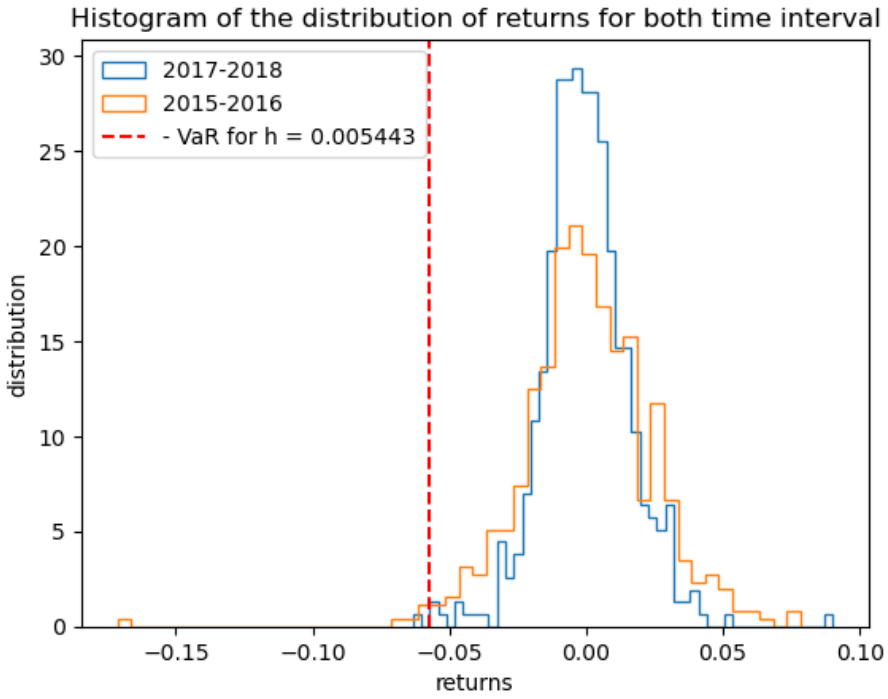



FIGURE 13 – Histogram of the distribution of returns for both time interval

This graph totally confirms our previous observations. Indeed, the 2015-2016 returns' distribution display a heavier tail and more spread distribution. So the Value at Risk calculated on the 2015-2016 dataset is based on a more spread distribution. That is why this Value at Risk overestimates the risk for the 2017-2018 dataset.

Thus, for the chosen value at risk (based on Silverman's rule), the VaR computed from January 2015 to December 2016 overestimates the risk of the returns from January 2017 to December 2018. The Value at Risk is thus not validated here.

# 3 Question B (Ex2, Q5 of TD2)

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

First, recall the definition of the Expected Shortfall from the course :

$$ES_\alpha(X) = \frac{1}{1-\alpha}\int_\alpha^1 \mathrm{VaR}_u(X)\,du,$$

where $X$ denotes a return.

The Expected Shortfall at level $\alpha$ represents the average loss exceeding the Value-at-Risk at level $\alpha$. Therefore, it can be estimated empirically by :

$$\widehat{ES}_\alpha = -\frac{1}{\mathrm{Card}\{X_i \le -\widehat{\mathrm{VaR}}_\alpha\}}\sum_{i=1}^n X_i \mathbf{1}_{\{X_i \le -\widehat{\mathrm{VaR}}_\alpha\}}.$$

Since the Expected Shortfall is defined as the average loss beyond the Value-at-Risk, it satisfies :

$$ES_\alpha(X) \ge \mathrm{VaR}_\alpha(X).$$

From Question A, we obtained a Value-at-Risk at the 99% level equal to around 0.0573. The corresponding Expected Shortfall provides additional information about what happens beyond this threshold in the tail of the loss distribution. An Expected Shortfall significantly larger than $\text{VaR}_\alpha$ would indicate the existence of losses much more extreme than the Value-at-Risk, meaning that the tail of the loss distribution does not vanish immediately beyond the VaR threshold but remains heavy.

### 3.1.1 Implementation of $ES(\alpha)$

In order to compute the Expected Shortfall at level $\alpha$, we retain only the returns that are lower than $-\text{VaR}_\alpha$ (since the Value-at-Risk is defined as the negative of the corresponding quantile), and we compute their empirical mean.

```
def expectedShortfall(returns_tab, VaR):
    return - np.mean([l for l in returns_tab if l <= - VaR])
es = expectedShortfall(returns_2015_2016, VaR)
print(f"For VaR = {VaR:.4f}, the expected shortfall is: {es:.4f}")
```

✓ 0.0s Python

```
For VaR = 0.0573, the expected shortfall is: 0.0842
```

**Discussions**

Since $0.0842 > 0.0573$, the Expected Shortfall is higher than the Value-at-Risk, which is a consistent and expected result.

# 4 Question C (Ex2, Q1 and Q2 of TD3)

With the dataset provided for TD1 on Natixis prices, first calculate daily returns. You will then analyse these returns using a specific method in the field of the EVT.

## 4.1 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 ?

From the course we have :

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 GEV distribution with parameter $\xi \in \mathbb{R}$. Let $k : \mathbb{N} \to \mathbb{N}$ be a function such that

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

and

$$\lim_{n\to\infty} \frac{k(n)}{n} = 0.$$

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 $\xi$.

### 4.1.1 Choice of $k(n)$

We choose :

$$k(n) = \lfloor \log n \rfloor$$

We clearly have

$$\lim_{n\to\infty} \lfloor \log n \rfloor = +\infty \quad \text{and} \quad \lim_{n\to\infty} \frac{\lfloor \log n \rfloor}{n} = 0.$$

### 4.1.2 Preparation of the datasets

In order to compute the tail distributions, we create two different lists : one for losses (returns $< 0$), which we multiply by -1 to work with positive values, and one for gains (returns $> 0$).

```
loss = list(df[df["return"] < 0]["return"] * -1)
gain = list(df[df["return"] > 0]["return"])
```

### 4.1.3 Estimation of the GEV parameters

We are now able to implement the Pickands estimator .

```
def pickandsEstimator(returns_tab):
    returns_tab.sort()
    N = len(returns_tab)
    k = np.floor(np.log(N))
    # starts from 0 so n is :
    n = N - 1
    e = np.log((returns_tab[int(n - k + 1)] - returns_tab[int(n - 2 * k + 1)]) /
               (returns_tab[int(n - 2 * k + 1)] - returns_tab[int(n - 4 * k + 1)]))
    e /= np.log(2)
    return e
```

We then apply the `pickandsEstimator` function to the two lists and obtain the following estimations.

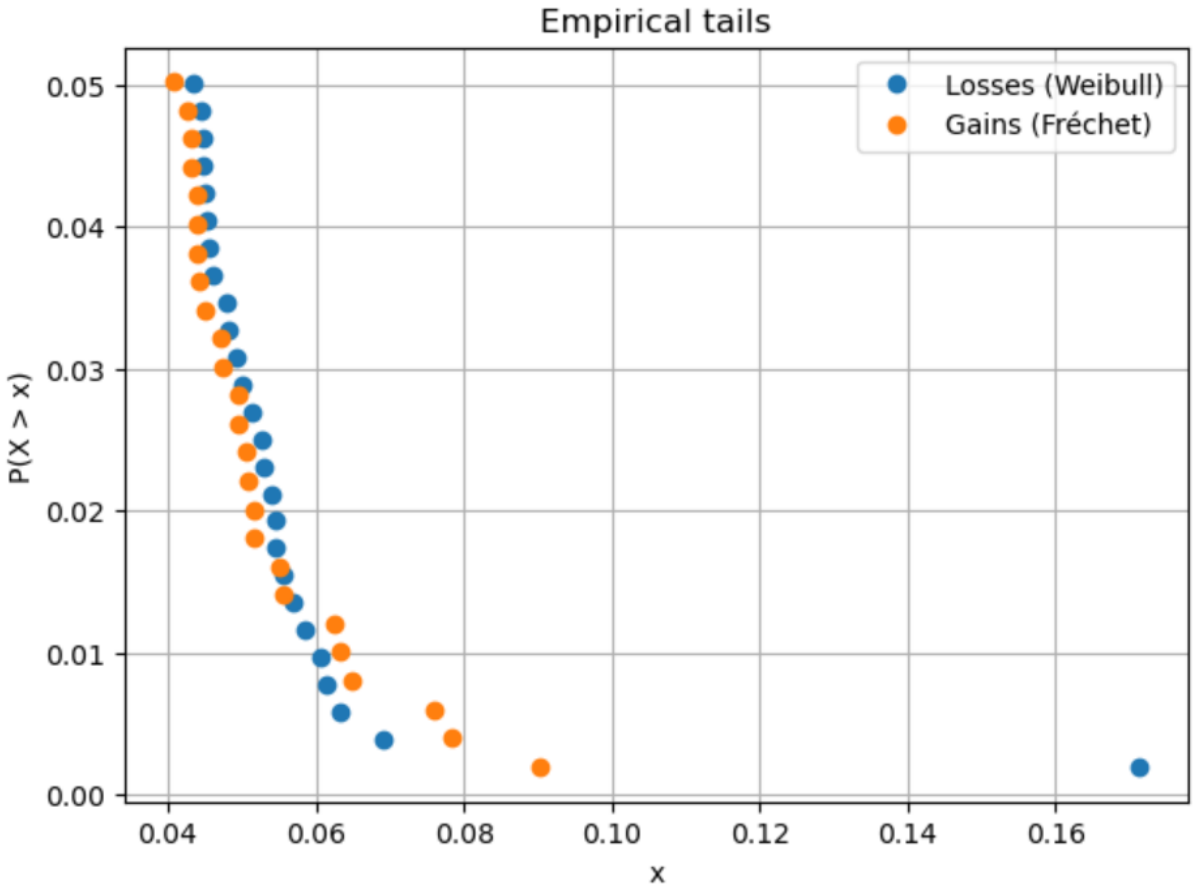
```python
def naturePickands(e):
    if e > 0:
        print("Heavy tails : GEV is of Fréchet kind")
    elif e == 0:
        print("Light tails : GEV is of Gumbel kind")
    else:
        print("Bounded support : GEV is of Weibull kind")

# loss
e_loss = pickandsEstimator(loss)
print("e_loss :", e_loss)
naturePickands(e_loss)

# gain
e_gain = pickandsEstimator(gain)
print("e_gain :", e_gain)
naturePickands(e_gain)
```

✓ 0.0s

```
e_loss : -0.5089715779341932
Bounded support : GEV is of Weibull kind
e_gain : 0.5772338569463368
Heavy tails : GEV is of Fréchet kind
```

**Interpretation of the estimations** The Pickands estimator yields a negative value for losses, $\xi_{loss} \approx -0.51$, and a positive value for gains, $\xi_{\text{gain}} \approx 0.58$. According to extreme value theory, a negative extreme value index indicates a Weibull-type distribution with a bounded upper tail, while a positive index corresponds to a Fréchet-type distribution with heavy tails.

These results therefore suggest an asymmetry in tail behavior : gains appear to exhibit a heavier tail than losses, with more extreme positive returns occurring with higher probability than extreme negative returns in the observed sample.




The visualization of the empirical tails confirms the Pickands estimates. Indeed, we observe that the gain distribution (in orange) decays more slowly, which is characteristic of fat tails, whereas the loss distribution decays much more rapidly, following an almost linear pattern that is closer to a Weibull-type behavior.

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

We work always in the classical extreme value theory framework. From the course :

Let $(X_i)_{i\geq 1}$ be a sequence of i.i.d. random variables with cumulative distribution function $F$ belonging to the max-domain of attraction of a GEV distribution with parameter $\xi$.

Under this setting, the Value-at-Risk at level $p$ can be estimated by

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

where $\hat{\xi}^P$ denotes the Pickands estimator of the extreme value index $\xi$.

### 4.2.1 Implementation of VaR(p)

**Setting :** Since we aim to estimate the Value-at-Risk, we focus on losses.
We have :
— $\hat{\xi}^P \approx -0.51$.
— $k(n) = \lfloor \log n \rfloor$, we keep the same choice of $k(n)$ as before.
— $X_{a:n}$ denotes the $a$-th order statistic of the sorted vector of losses.

```python
def ComputeVaRp(returns_tab, e, p):
    returns_tab.sort()
    N = len(returns_tab)
    k = np.floor(np.log(N))
    # starts from 0 so n is :
    n = N - 1
    multi = (k / (n * (1.0 - p))) ** e - 1.0
    multi /= (1.0 - 2.0**(-e))

    return (multi * (returns_tab[int(n - k + 1)] - returns_tab[int(n - 2 * k + 1)])
            + returns_tab[int(n - k + 1)])
```

## 4.3 Results and discussions

```python
p_tab = [0.995, 0.99, 0.98, 0.95]
for p in p_tab:
    print(f"For p = {p} and the estimator = {round(e_loss, 4)}, "
          f"we have VaR(p) = {round(ComputeVaRp(loss, e_loss, p), 4)}"
)
```

✓ 0.0s

```
For p = 0.995 and the estimator = -0.509, we have VaR(p) = 0.0633
For p = 0.99 and the estimator = -0.509, we have VaR(p) = 0.0596
For p = 0.98 and the estimator = -0.509, we have VaR(p) = 0.0543
For p = 0.95 and the estimator = -0.509, we have VaR(p) = 0.0436
```

We first compute the value at risk for four different confidence levels (95%, 98%, 99%, and 99.5%). The obtained values are positive, which is expected since we are working with losses. The Values at Risk increase with $p$ and are of the same order of those obtained using the same data in the previous question (with non parametric distribution). To ensure that this observation is general and not just for a few discrete observations, we then plot VaR($p$) as a function of $p$ and made the same observations.

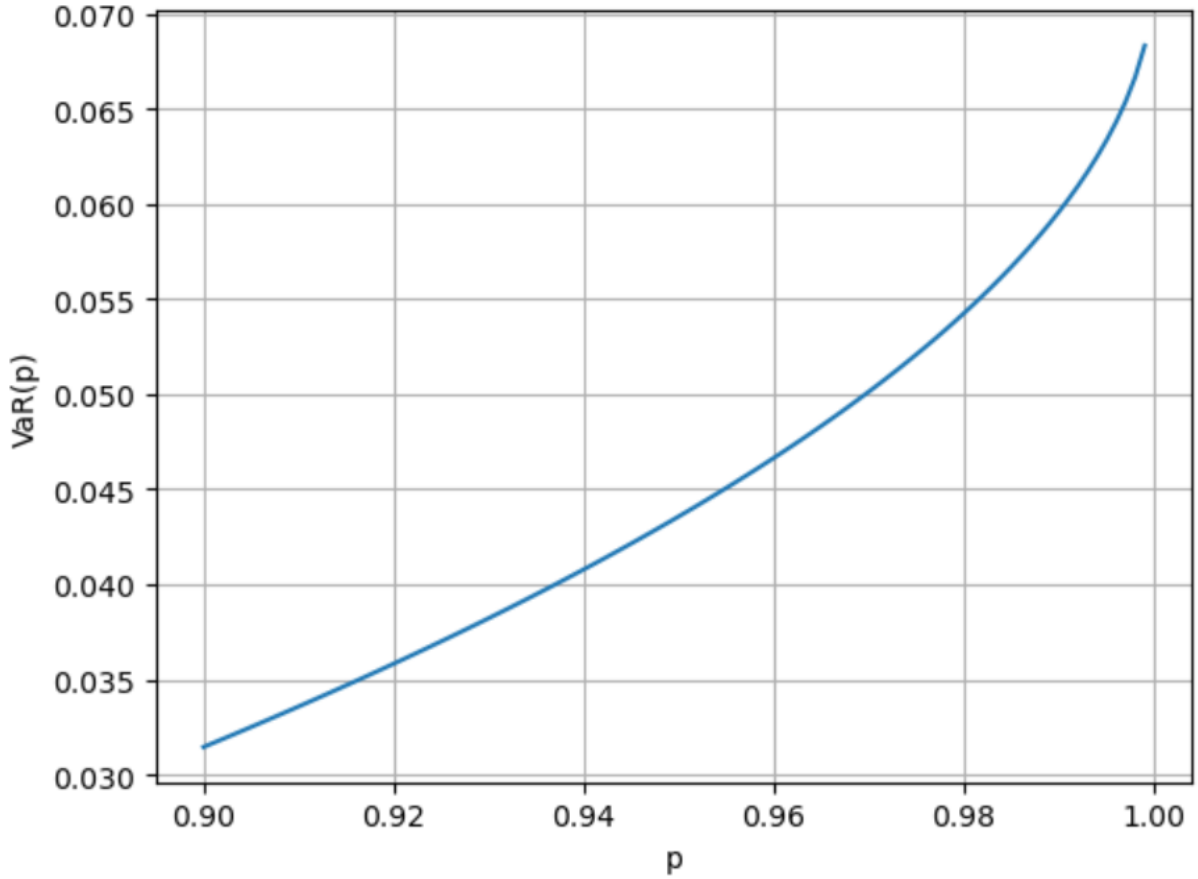



FIGURE 14 – Evolution of VaR($p$) as a function of $p$

# 5 Question D (Ex2, Q5 of TD4)

## 5.1 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 ?

We first recall the Bouchaud's price impact model from our course :

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

where :

— $S$ denotes the bid–ask spread ;

— $\varepsilon \in \{-1, 1\}$ indicates the sign of the trade, taking the value $+1$ for a buy order (price at the ask) and $-1$ for a sell order (price at the bid) ;

— $V$ is the transaction volume, with $r$ close to zero in order to obtain a concave impact function in volume ;

— $G$ is a function equal to 0 on $\mathbb{R}^-$, which can be interpreted as the impact of a single order. A statistical study of long-term correlations allows this function to be specified, for instance as a decreasing power-law function.

The parameters to be estimated are :

— the impact function $G$ ;
— the exponent $r$.

For this part every elements come from 3 differents articles writed by Jean-Philippe Bouchaud. In these articles we took methods, formulas, but also exemples of estimated parameters to check the coherence of our estimations.

# Références


[1] Bouchaud, J.-P. (2009). *Price Impact.*

[2] Taranto, D. E., Bormetti, G., Bouchaud, J.-P., Lillo, F., and Tóth, B. (2016). *Linear models for the impact of order flow on prices.*

[3] Bouchaud, J.-P., Gefen, Y., Potters, M., and Wyart, M. (2004). *Fluctuations and response in financial markets : the subtle.*


### 5.1.1 Estimation of $G(\ell)$

The model from the course is not really mentioned in the literature, or at least we do not find it explicitly. However, there exists an almost similar expression in (Bouchaud et al., 2009), where a non-linear model is introduced :

$$p_T = p_{-\infty} + \lambda \sum_{n=-\infty}^{N-1} G(N-n)\, \varepsilon_n\, v_n^{\psi}.$$

The only difference is the presence of the parameter $\lambda$, which is inversely proportional to market liquidity. In practice, $\lambda$ implicitly captures liquidity effects such as the bid–ask spread, which is explicitly present in the course formula but not in this expression. This formulation can be used to estimate the impact kernel $G$ and the exponent $\psi$ (denoted $r$ in the course notation).

First, from (2009),

$$G(\ell) = G(1) + \sum_{n=1}^{\ell} \mathcal{G}(n),$$

where

$$\mathcal{G}(\ell) = -(2\rho - 1)\, G(1)\, \lambda_\ell.$$

Here, $p$ is the probability that two successive transactions have the same sign ; it is usually taken in the interval $[0.6, 0.7]$. The quantity $\lambda_\ell$ can be obtained from another quantity that we are able to compute,

$$C_0(\ell) = \langle \varepsilon_{n+\ell} \varepsilon_n \rangle - \langle \varepsilon_n \rangle^2 \;\simeq\; \frac{C_0}{\ell^\gamma}, \qquad \text{for } \ell \gg 1,$$

which implies

$$\log C_0(\ell) \;\simeq\; \log C_0 - \gamma \log \ell.$$

If this approximation holds, we obtain

$$\lambda_\ell \sim \ell^{(\gamma-3)/2}.$$

In order to compute $\mathcal{G}(\ell)$ and so $G(\ell)$, we have to estimate $\lambda_\ell$ by using the approximation presented above. Therefore we also need an estimation of $\gamma$.

**Step 1 : Estimate $\gamma$ with $C_0$** We first verify whether the approximation

$$\log C_0(\ell) \simeq \log C_0 - \gamma \log \ell$$

holds. If this is the case, we directly estimate $\gamma$ using a linear regression of $\log C_0$ as a function of $\log \ell$ .

Recalling that

$$C_0(\ell) = \langle \varepsilon_{n+\ell} \varepsilon_n \rangle - \langle \varepsilon_n \rangle^2$$

we compute an empirical estimate of $C_0(\ell)$ from the data by averaging the products $\varepsilon_{n+\ell}\varepsilon_n$ and subtracting the squared mean of $\varepsilon_n$, as implemented in our code.

```python
#Compute C_0
def C_0(eps_tab, l_min, l_max):

    C0 = np.zeros(l_max - l_min)
    mu = np.mean(eps_tab)
    for l in range(l_min, l_max):
        C0[l-l_min] = np.mean(eps_tab[l:]*eps_tab[:-l]) - mu**2
    return C0
```

Then we compute $C_0(\ell)$ for different values of $\ell$ and estimate $\gamma$ using a log–log regression. Recall that the approximation we want to verify only holds for $\ell \gg 1$. Moreover, since we have a limited sample size (only 1000 observations), we cannot compute the autocovariance function $C_0(\ell)$ for very large values of $\ell$. We therefore choose to compute $C_0(\ell)$ and study the log–log relationship for $\ell \in [20, 50]$.

```python
l_min = 20
l_max = 50
eps = df_TD4["Sign of the transaction"].to_numpy()

C0 = C_0(eps, l_min, l_max)

mask = C0 > 0 #To be able to apply log() and find gamma
C0_reg = C0[mask]
l_vect = np.arange(l_min, l_max)[mask]

l_vect_log = np.log(l_vect)
C0_reg_log = np.log(C0_reg)

coeffs = np.polyfit(l_vect_log, C0_reg_log, 1)
gamma_hat = -coeffs[0]
intercept = coeffs[1]
print(f"Estimated C0 = {np.exp(intercept)}")
print(f"Estimated gamma = {gamma_hat}")
```

✓ 0.0s

```
Estimated C0 = 0.07464905158260039
Estimated gamma = 0.3652812433996198
```

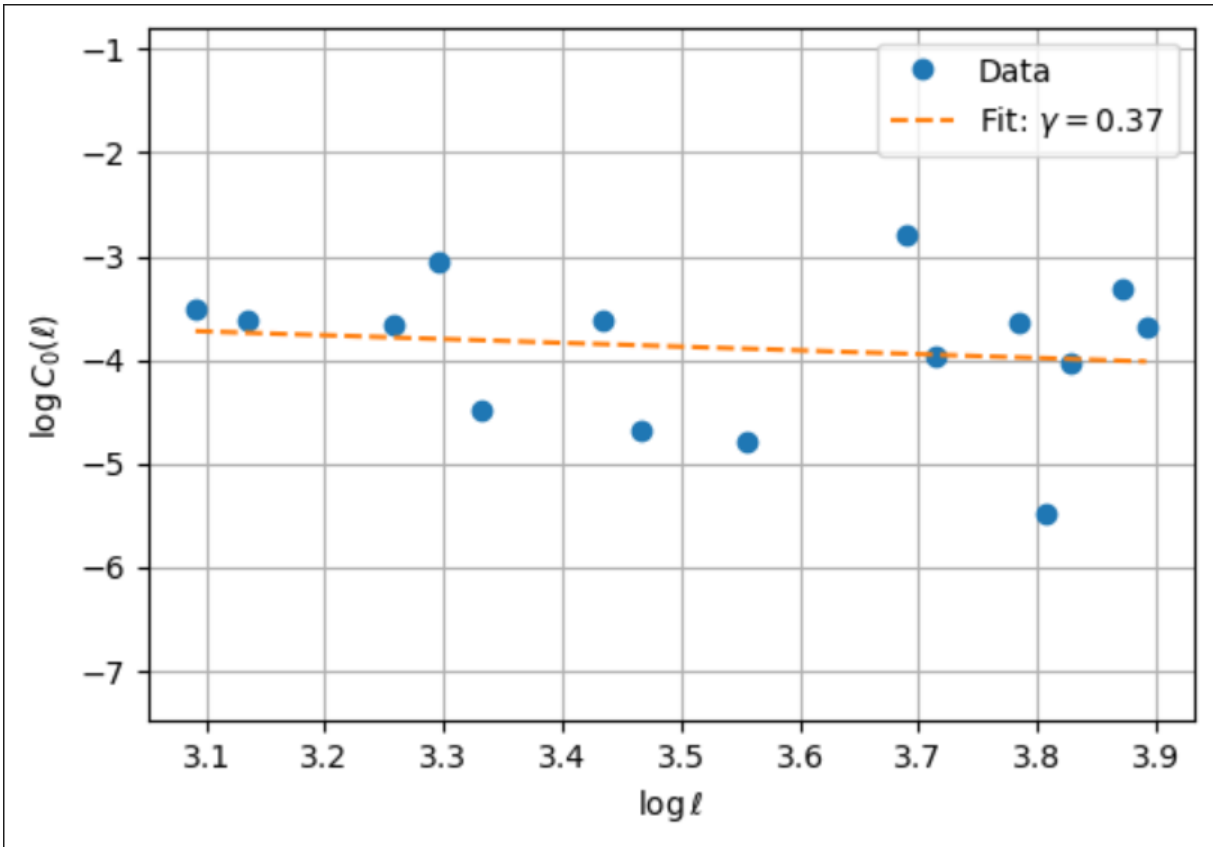



FIGURE 15 – Log-log plot of $C_0(\ell)$

**Discussion** This estimation does not have a high level of reliability, as we only use about 1,000 observations. In the papers by J.-P. Bouchaud, the estimations are based on millions of data points. However, they typically find values of $\gamma$ between 0.2 and 0.5. Our estimate, $\gamma \approx 0.37$, therefore is totally realistic.

**Second Step : Determine $\lambda_l$ and Compute $G(\ell)$** We obtained $\gamma$, therefore we can easely determine $\lambda_l$ with the opproximation presented above :

$$\lambda_\ell \sim \ell^{(\gamma-3)/2}$$

```
def lambda_l(gamma,l):
    lambda_l = l**((gamma-3)/2)
```

And thus we can also compute $\mathcal{G}(\ell)$ and so $G(\ell)$ using the formulas presented above.

```python
def Gs(l, p, G1, gamma):
    lambda_l = l**((gamma-3)/2)
    Gs_l = -(2*p-1)*G1*lambda_l
    return Gs_l

def G(G1, l, p, gamma):
    G_l = G1
    i = 1
    while i <= l:
        G_l += Gs(i, p, G1, gamma)
        i+=1
    return G_l
```

By convention we will take $G(1) = 1$ ,this choice has no impact on the shape of $G(\ell)$ and on the value of the other parameters. the choice of $\rho$ is much harder, in fact we choose the highest p such that $G(\ell)$ is well defined (such that we observe something close to a decreasing power function), we did that because when

$$C(\ell) \underset{\ell \gg 1}{\sim} \ell^{-\gamma},$$

already verified in Step 1. Then,

$$\rho \to 1^{-}.$$

```python
G1 = 1.0
gamma = gamma_hat
p = 0.63

L_max = 1000
lags = np.arange(1, L_max + 1)

G_vals = np.array([G(G1, l, p, gamma) for l in lags])

plt.figure(figsize=(7,4))
plt.plot(lags, G_vals, 'o-',markersize=1, label=r"$G(\ell)$")

plt.xlabel(r"Lag $\ell$")
plt.ylabel(r"$G(\ell)$")
plt.grid(True)
plt.legend()
plt.show()
```

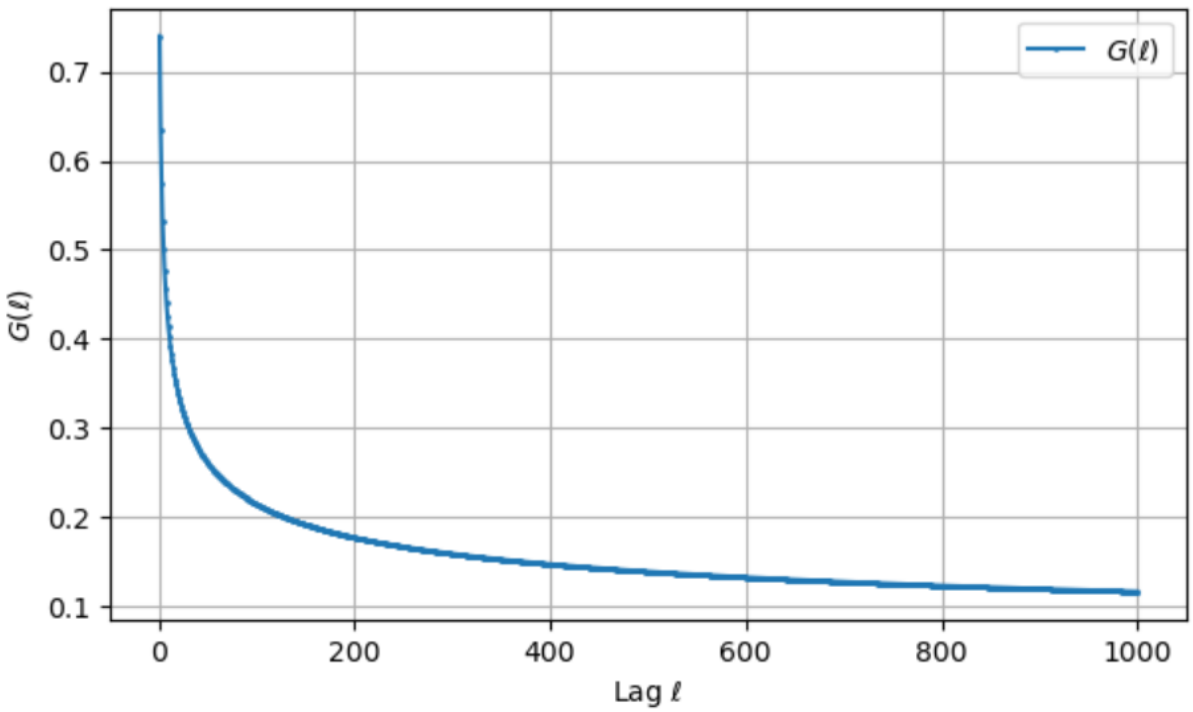



FIGURE 16 – $G(\ell)$ as a function of $\ell$.

**Approximate** $G(\ell)$ Now that we have computed $G(\ell)$, we aim to approximate it by a decreasing power-law function, as mentioned in the course. For this purpose, we use the result from (Taranto et all, 2016) :

$$C_0(\ell) \sim \ell^{-\gamma} \;\Rightarrow\; G(\ell) \sim \ell^{-\beta}, \qquad \beta = \frac{1-\gamma}{2}, \quad \ell \gg 1.$$

To obtain a more reliable estimation of $\beta$, we estimate it in two different ways :

— First, by determining the slope of the log–log plot of $G(\ell)$ as a function of $\ell$ (for large values of $\ell$).

— Second, using the approximation $\beta = \frac{1-\gamma}{2}$, which requires a Hurst exponent $H$ close to 0.5, a condition that we will verify.

**Determining the slope of the log–log plot of** $G(\ell)$ **as a function of** $\ell$ We have already computed $G(\ell)$ for a wide range of values of $\ell$. Recalling $G(\ell) \sim \ell^{-\beta}$, we plot $\log G(\ell)$ as a function of $\log \ell$.

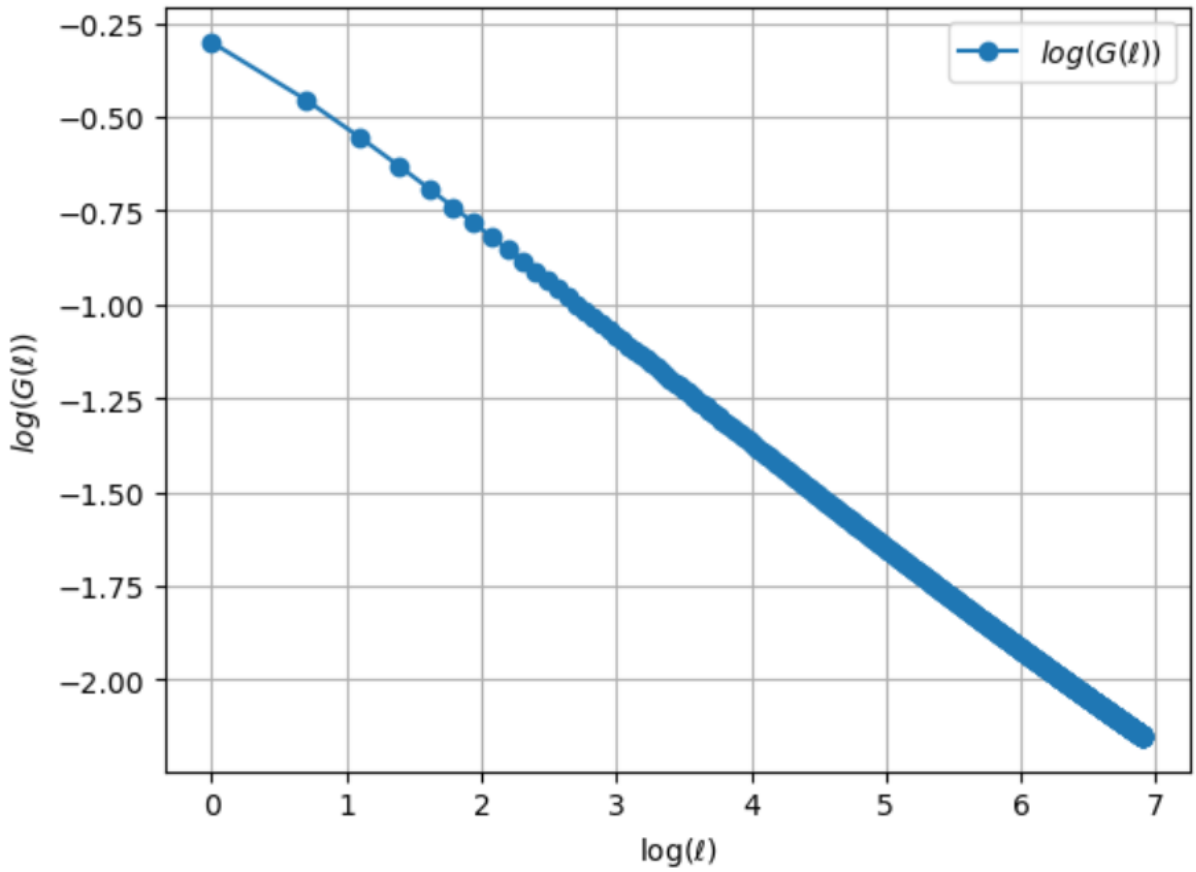



FIGURE 17 – Log–log plot of $G(\ell)$ as a function of $\ell$.

We can distinguish a clear linear relationship for large values of $\ell$. By restricting the plot to sufficiently large $\ell$, we can therefore estimate the slope and obtain an estimate of $\beta$, which is equal to 0.26 in this case.

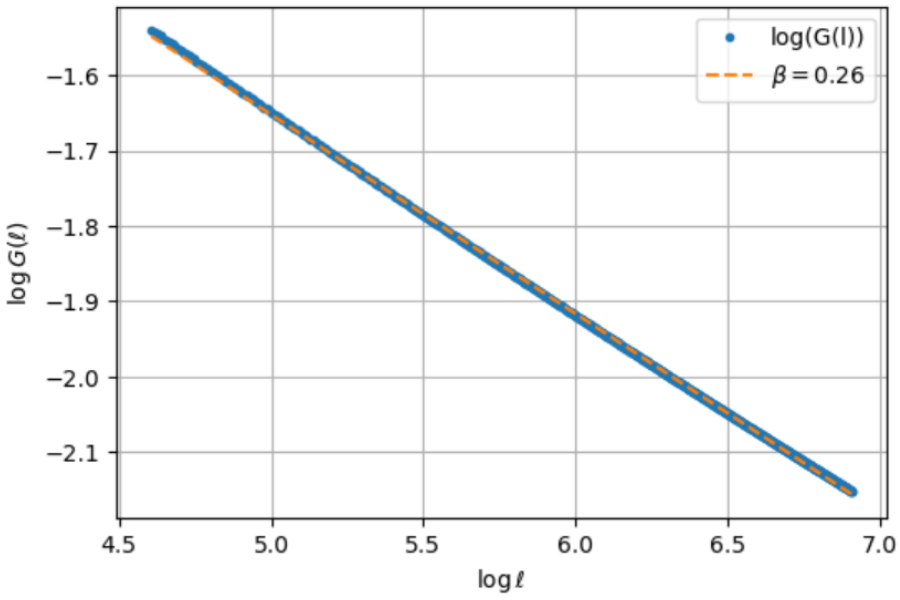



FIGURE 18 – Log–log plot of $G(\ell)$ for large values of $\ell$.

**Using now the approximation $\beta = \frac{1-\gamma}{2}$**

This approximation from (Taranto et al., 2016) is derived from the original relation in (Bouchaud et al., 2004) :

$$2H = 2 - 2\beta - \gamma,$$

where $H$ denotes the Hurst exponent.

Therefore, under the condition that fluctuations are diffusive at long times (i.e. $H = 1/2$), we obtain

$$\beta = \frac{1-\gamma}{2}.$$

We will estimate the Hurst exponent in order to verify whether this condition is satisfied, using the estimator

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

where $M_2$ denotes the second moment of price increments over one time step, and $M_2'$ the second moment of increments over two time steps.

```
#Use of df_TD4_grouped to have constant duration for the increments
M2 = ((df_TD4_grouped["price"]- df_TD4_grouped["price"].shift(1)) *
      (df_TD4_grouped["price"] - df_TD4_grouped["price"].shift(1))).mean()
def ComputeM2_prime(df):
    m2_prime = 0.0

    for i in range (1, (len(df)-1) // 2 + 1):
        m2_prime += ((df["price"].iloc[2*i] - df["price"].iloc[2*i-2]) *
                     (df["price"].iloc[2*i] - df["price"].iloc[2*i-2]))
    m2_prime /= (len(df)-1) // 2

    return m2_prime

M2_prime = ComputeM2_prime(df_TD4_grouped)
H_hat = np.log2(M2_prime/M2)/ 2
H_hat
✓ 0.0s
```

```
0.5451525909483125
```

The H estimate is pretty close to 0.5 so we can use the relation $\beta = \frac{1-\gamma}{2}$ but also the non simplified one, $2H = 2 - 2\beta - \gamma$, with the H estimated.

```
beta_hat2 = (1-gamma_hat)/2
print(beta_hat2)

beta_hat3 = (2 - 2*H_hat - gamma_hat)/2
print(beta_hat3)
```
✓ 0.0s
```
0.3173593783001901
0.27220678735187764
```

**Discussion about the estimation of $G(\ell)$**
These values are very close to the estimate of $\beta$ obtained with the first method. Therefore, for large $\ell$, we can approximate

$$G(\ell) \sim \ell^{-\beta},$$

with $\beta \approx 0.27$.

### 5.1.2 Estimation of $r$

In order to estimate $r$, we first express the price as follows :

$$p_t = p_0 + \sum_{s=0}^{t-1} G(t-s)\, \varepsilon_s\, S_s\, V_s^r.$$

In other words, the price at time $t$ is equal to the price at the beginning of the period plus the cumulative impact of all trades since the beginning of the period.

**Methodology**
We take :

- $p_0$ as the first observed price ;
- $G(\ell)$ as estimated above ;
- for the unknown volumes in our dataset, we use the mean volume. This approach is not perfect but remains coherent, since some papers by Bouchaud and co-authors consider constant volumes for their estimations.

In order to estimate $r$, we minimize the sum of squared errors (SE) between the true price and the model's price at each time $t$, from $t = 0$ to the last observation. The squared error is defined as

$$\mathrm{SE}(r) = \sum_{t=1}^{T} \left(p_t^{\text{model}}(r) - p_t^{\text{true}}\right)^2.$$

We first implement the function $p_t$.

```
def p_t(p_0, G_vect, eps_vect, S_vect, V_vect, r, t):
    return p_0 + (G_vect[:t] * eps_vect[:t] * S_vect[:t] * (V_vect[:t]**r)).sum()
```

We then define all the variables introduced above.

```
p_0 = df_TD4["price"].iloc[0]
p_t_true = df_TD4["price"].values
G_vect = G_vals[::-1]
S_vect = df_TD4["spread"].iloc[:-1]
V_vect = df_TD4["volume"].iloc[:-1]
eps_vect = eps[:-1]
```

Finally, we compute the squared error $\mathrm{SE}(r)$ for values of $r$ in the interval $[0, 1]$, which we later restrict to $[0, 0.3]$ in order to better observe the minimum. For each value of $r$, we iterate over time $t$ and sum the squared errors at each time step.

```
T = len(S_vect)
r_test = np.arange(0, 0.3, 0.01)
SE_test = []

for r in r_test:
    se_r = 0.0
    for t in range(1, T):
        p_model = p_t(p_0, G_vect, eps_vect, S_vect, V_vect, r, t)
        se_r += (p_model - p_t_true[t])**2
    SE_test.append(se_r)

SE_test = np.array(SE_test)
plt.plot(r_test, SE_test)
plt.xlabel("r")
plt.ylabel("Sum of squared errors")
plt.grid(True)
```

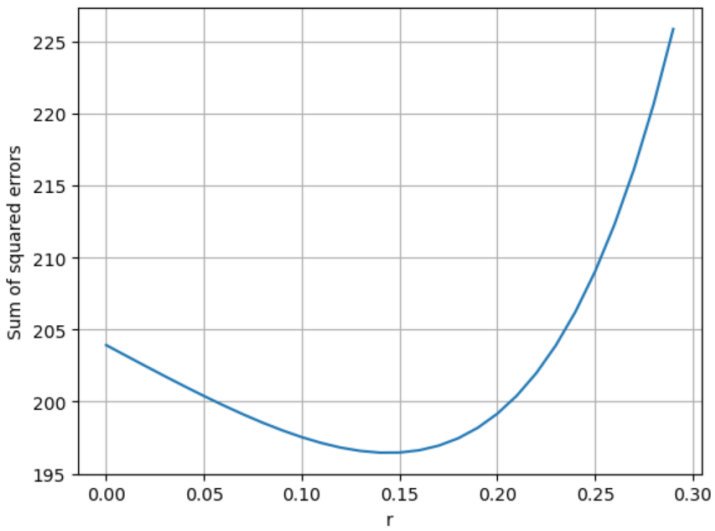



FIGURE 19 – Evolution of $\mathrm{SE}(r)$

**Discussion**
We find that the sum of squared errors is minimized for $r \approx 0.14$. This value is consistent with the existing literature ; for instance, Bouchaud et al. (2009) report values of $r$ in the range 0.1–0.3. However, this estimation relies on the assumption that the impact of transactions occurring before $t_0$ can be neglected.

### 5.1.3 Conclusion and discussion about the specification of the model

**Summary of the parameters estimations**

| Parameter | Estimated value | Method / Comment |
|---|---|---|
| $\gamma$ | $\approx 0.37$ | Estimated from the log–log regression of $C_0(\ell)$ |
| $\lambda_\ell$ | $\sim \ell^{(\gamma-3)/2}$ | Derived from $\gamma$ using the asymptotic approximation |
| $\beta$ | $\approx 0.27$ | Estimated from the slope of the log–log plot of $G(\ell)$ |
| $G(\ell)$ | $\sim \ell^{-\beta}$ | Derived from $\beta$ using the asymptotic approximation |
| $r$ | $\approx 0.14$ | Minimization of the sum of squared errors over the trajectory |

TABLE 2 – Summary of the parameters estimated for Bouchaud's price impact model.

**Visualization of the estimated price**

In this final visualization, we observe that the variations of the true price and the model-implied price exhibit a very similar behavior. However, when large price variations occur, the model does not capture them accurately. This limitation may be due to the lack of detailed volume information, which can lead the model to either overestimate or underestimate the impact of volume for certain transactions.

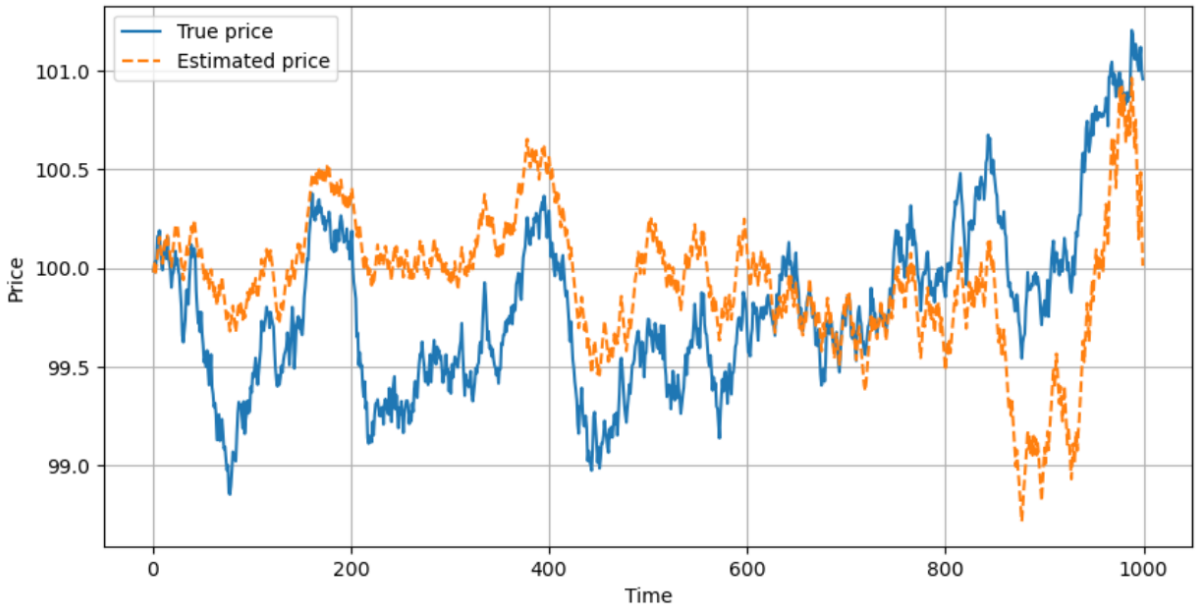



FIGURE 20 – Comparison between the observed price and the model's price

**Model specification**
The impact kernel $G(\ell)$ is assumed to be positive and decreasing, and seems to be well

approximated by a power-law function. The estimated values of $\gamma$ and $\beta$ are consistent with this specification and have classic behaviour comparing to the literature. However, the correct specification of $G(\ell)$ also depends on other parameters, such as $\rho$, which may affect the precise shape of the estimated impact kernel.

The dependence on volume is specified through a concave power-law function $V^r$. The estimated value $r \approx 0.14$ is in line with the values reported in the literature (and in accordance with the course that mentions a $r$ close to 0).

Several simplifying assumptions are nevertheless required. In particular, the impact of transactions occurring before the initial observation time is neglected, and missing volumes are replaced by their mean value. Furthermore our model suffers from a to small number of observations. Indeed this type of estimations are often done with millions of transactions, we have only 1000.

While these choices may affect the precision of the estimates, the overall consistency of the estimated parameters and the comparison between the observed price and the model's price suggest that the chosen specification provides a reasonable, though not highly accurate, approximation of price impact dynamics in the studied dataset.

# 6 Question E (Q2 and Q3 of TD5)

## 6.1 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 ?

To compute, the multiresolution correlation between the pairs and observe an Epps effect, we followed a series of steps.

### 6.1.1 Compute the average between the highest and lowest price

We will work with the mean of the highest and lowest price. That is why we created the following function.

```
def computeMean(df):
    df["mean"] = (df.iloc[:,1] + df.iloc[:,2])/2
```

FIGURE 21 – Function that computes the mean

### 6.1.2 Scale coefficients

We know from the course that scale coefficients are calculated for several adjacent discrete scales $j \in J$ for two series, by treating or eliminating the border effects : $(c^1_{j,k})_{j\in J, k\in[0,T]}$ and $(c^2_{j,k})_{j\in J, k\in[0,T]}$

To compute scales coefficients we started at level $j = 0$

$$C^{(0)} = (X_1, ..., X_N)$$

with X the mean between the highest and lowest price. Then for each level $j$, the scale coefficients are calculated from the previous ones :

$$C^{j+1} = \frac{C_{2k}^{(j)} + C_{2k+1}^{(j)}}{\sqrt{2}}$$

After executing all scales coefficient, we obtain a vector of vectors, each sub-vector representing the coefficients for each level j.

```
def Scale_coefficient(df):
    C = df["return"].to_numpy()

    scale_total = [C]
    while len(C) >= 2:
        if len(C) % 2 == 1:
            C = C[:-1]
        C = (C[0::2] + C[1::2]) / np.sqrt(2)
        scale_total.append(C)
    return scale_total

C_gbp = Scale_coefficient(df_gbp)
C_sek = Scale_coefficient(df_sek)
C_cad = Scale_coefficient(df_cad)
```

FIGURE 22 – Function that computes scale coefficients

### 6.1.3 Covariance at level j

$$\mathrm{Cov}(j) = \frac{1}{T}\sum_{k=1}^{T}\left(c_{j,k}^{1} - \frac{1}{T}\sum_{l=1}^{T} c_{j,l}^{1}\right)\left(c_{j,k}^{2} - \frac{1}{T}\sum_{l=1}^{T} c_{j,l}^{2}\right).$$

with $c_{j,k}^{1}$ and $c_{j,k}^{2}$, the coefficient at level $j, k$ for two scale coefficient vectors.

```
def covariance1D(c1, c2):
    return np.mean((c1 - np.mean(c1)) * (c2 - np.mean(c2)))
```

FIGURE 23 – Function that computes the covariance at level j

### 6.1.4 Correlation

The correlation is computed from :

$$\rho = \frac{\text{Cov}(X_1, X_2)}{\sqrt{\sigma_1^2 \times \sigma_2^2}}$$

```
def correlation1D(c1, c2):
    n = min(len(c1), len(c2))
    c1 = c1[:n]
    c2 = c2[:n]

    cov = covariance1D(c1, c2)
    var1 = np.mean((c1 - np.mean(c1))**2)
    var2 = np.mean((c2 - np.mean(c2))**2)
```

FIGURE 24 – Correlation at level j

We can then apply the correlation to each level and obtain a vector of correlations.

```
def correlationMultiScale(C1, C2):
    n = min(len(C1), len(C2))
    rho = []

    for j in range(n):
        rho.append(correlation1D(C1[j], C2[j]))

    return np.array(rho)

rho_gbp_sek = correlationMultiScale(C_gbp, C_sek)
rho_gbp_cad = correlationMultiScale(C_gbp, C_cad)
rho_cad_sek = correlationMultiScale(C_cad, C_sek)
```

FIGURE 25 – Function that computes the correlation for each level

### 6.1.5 Visualisation of the Wavelet multi-scale correlations and Discussion of the Epps effect

Here we refer to wavelet multi-scale correlation because the correlation is computed at each scale $j$ corresponding to different wavelet scale. The wavelet functions are defined as follows :

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

$$\psi_{j,k}(t) = 2^{j/2}\,\psi\left(2^j t - k\right), \qquad j \in \mathbb{Z},\ k \in \mathbb{Z}.$$

```python
# Mother wavelet
def Wavelet_Haar(t):
    return np.where((0 <= t) & (t < 0.5), 1,
        np.where((0.5 <= t) & (t < 1), -1,
                 0))

# Daughter wavelet
def daughter_wavelet(j, k, t):
    return 2**(j/2) * Wavelet_Haar(t * 2**j - k)
```

FIGURE 26 – Wavelet functions in python

From each vector of correlations, we can display the correlation as a function of the scale $j$. However, it is important to exclude from the graph scales composed of less than 20 coefficients. Indeed, when the number of coefficients is too small, the correlation becomes unstable and may not be relatable.

```python
valid_j_gbp = [j for j in range(len(C_gbp)) if len(C_gbp[j]) >= 20]
valid_j_cad = [j for j in range(len(C_cad)) if len(C_cad[j]) >= 20]
valid_j_sek = [j for j in range(len(C_sek)) if len(C_sek[j]) >= 20]

valid_j = list(set(valid_j_gbp) & set(valid_j_cad) & set(valid_j_sek))

rho_gbp_sek = rho_gbp_sek[valid_j]
rho_gbp_cad = rho_gbp_cad[valid_j]
rho_cad_sek = rho_cad_sek[valid_j]
```

FIGURE 27 – Select only levels with at least 20 elements

Finally, we obtain the following visualisation :

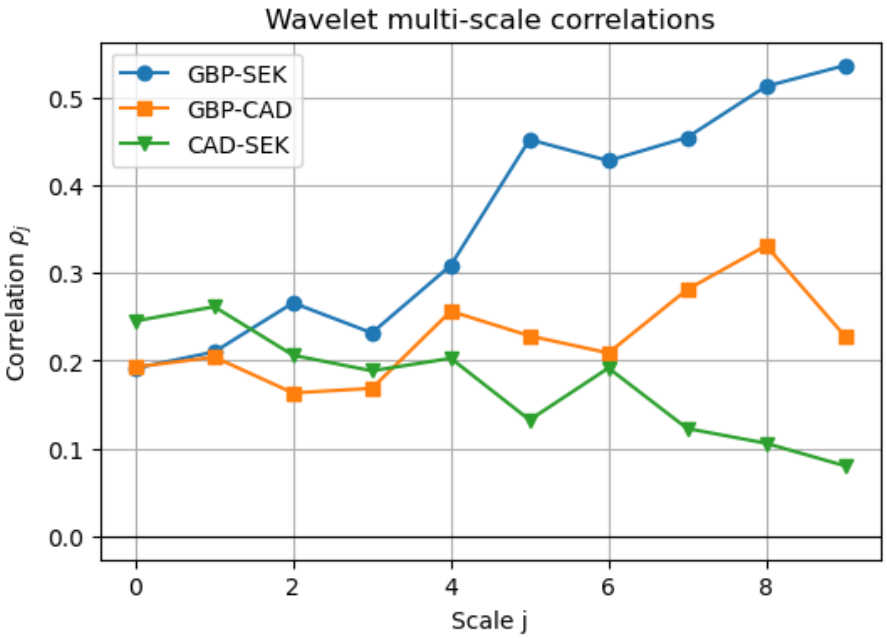



FIGURE 28 – Wavelet multi-scale correlations

**Discussion of the Epps effect**

Both articles provide a clear understanding of the Epps effect. "The Epps effect is the key phenomenology that couples the emergence of correlations to the choice of sampling time scales." The Epps effect can be described as follows : the empirical correlation between two assets is

— low at high frequency
— high at low frequency

At low frequency, correlations are stable and clear. Whereas, at high frequency, individual transactions, trading asynchrony and noise reduce the correlation.

The graph above represents the evolution of the correlation between three pairs of FX rates as a function of the scale $j$ (frequency). The higher $j$ is, the lower the frequency. Conversely, the lower $j$ is, the higher the frequency.

So, to observe the EPPS effect, the correlation should increase with the scale $j$

**Case 1 : Correlation GBPEUR - SEKEUR** The blue curve increases as the scale grows : at $j = 0$, the correlation is low and rises quickly as j increases.

Therefore, the EPPS is clearly demonstrated for GBPEUR - SEKEUR.

**Case 2 : Correlation GBPEUR - CADEUR**

The EPPS effect for GBPEUR and CADEUR is weakly visible. Indeed, the orange curve globally increases but shows many irregularities (the curve is not strictly growing).

Therefore, the EPPS effect is weekly present and the relationship between the two asset may be influenced by individual or market effet.

**Case 3 : Correlation CADEUR - SEKEUR**

The green curve decreases as $j$ increases. This behaviour is opposite to the EPPS effect : here at high frequency the correlation is higher than at low frequency.

According to the article "The Epps effect under alternative sampling schemes", when

two assets does not share a meaningful correlation, it becomes difficult to provide global measures of event synchronisation. As a result, even at low frequency, there may not be a clear correlation.

Thus, the absence of increasing correlations between CADEUR and SEKEUR at lower frequency suggests that the two assets do not share a strong common factor, correlation. The relationship between CADEUR and SEKEUR cannot be interpreted through the EPPS effect.

**Conclusion**

The wavelet multi-scale analysis is a relevant method to identify the Epps effect, which links frequency to correlation. Our study has shown that this effect is only observed for certain pairs. The absence of the Epps effect can be due to the lack of a strong common factor.

# Références


[1] Patrick Chang, Etienne Pienaar, Tim Gebbie (2020). *The Epps effect under alternative sampling schemes.*

[2] Jérôme Busca, Léon Thomir (2023). *EPPS EFFECT AND THE SIGNATURE OF SHORT-TERM MOMENTUM TRADERS.*


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

The definition of the Hurst exponent is given in the course as :

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

with

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

and

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

By applying these formulas on each dataset (GBPEUR, SEKEUR, CADEUR) with $N = 1$, we obtain the following Hurst exponent :

— Hurst exponent for GBPEUR : 0.671414
— Hurst exponent for SEKEUR : 0.654591
— Hurst exponent for CADEUR : 0.655244

We can remark that the hurst exponents are higher than $\frac{1}{2}$ thus the increments are positively correlated (erases singularities).

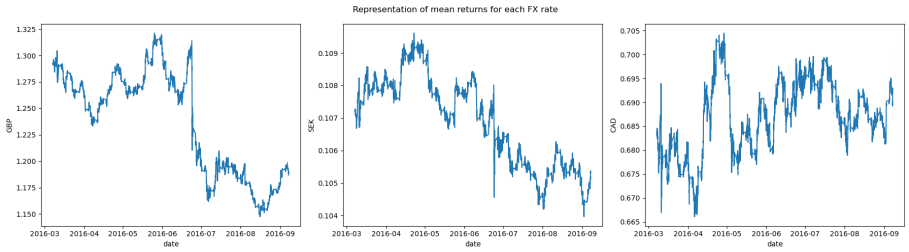



FIGURE 29 – Representation of mean returns from each FX rates

These representations are consistent with the interpretation of the Hurst exponent. Indeed, the Hurst exponent of each FX rate is higher than 0.5 and the representation of the mean rate is persistent (strong upward and downward trends) and suggest positive dependence between increments. Moreover, the GBPEUR Hurst exponent is slightly higher than SEKEUR and CADEUR and we can clearly see that GBPEUR representation displays less noise.

```python
def ComputeM2(df):
    return ((df["mean"]- df["mean"].shift(1)) * (df["mean"] - df["mean"].shift(1) )).mean()

def ComputeM2_prime(df):
    m2_prime = 0.0

    for i in range (1, (len(df)-1) // 2 + 1):
        m2_prime += (df["mean"].iloc[2*i] - df["mean"].iloc[2*i-2]) * (df["mean"].iloc[2*i] - df["mean"].iloc[2*i-2])
    m2_prime /= (len(df)-1) // 2

    return m2_prime

def ComputeH_hat(df):
    computeMean(df)
    m2_prime = ComputeM2_prime(df)
    m2 = ComputeM2(df)
    return np.log2(m2_prime/m2) / 2

print("Hurst exponent for GBPEUR :", ComputeH_hat(df_gbp_mean))
print("Hurst exponent for SEKEUR :", ComputeH_hat(df_sek_mean))
print("Hurst exponent for CADEUR :", ComputeH_hat(df_cad_mean))
```

FIGURE 30 – Python code that computes Hurst exponent

Then, we know from the course that the anualised volatility using the hurst exponent is :

$$vol_{annualised} = vol_{daily} \times 252^H$$

For this dataset, each row is given every 15 minutes. So the daily volatility can be written as :

$$vol_{daily} = vol_{15minutes} \times 96^H$$

By substituting this expression into the first equation, we obtain :

$$vol_{annualised} = vol_{15minutes} \times (252 \times 96)^H$$

```python
def ComputeVolAnnualised(df):
    H = ComputeH_hat(df)
    ComputeReturn(df)
    return np.std(df["return"], ddof=1) * ((252*96)**H)
```

FIGURE 31 – Annualised volatility with Hurst exponent

We obtain the following results :
— Annualised Volatility GBP : 0.546938
— Annualised Volatility SEK : 0.242234
— Annualised Volatility CAD : 0.377367