Market Risk Project

Clément Muret - Maxime Le Floch

December 26, 2023

Contents

1 Introduction

In today's financial landscape, understanding and managing market risk plays a vital role in ensuring the stability and success of financial enterprises. This project addresses the different facets of risk analysis, using empirical data and quantitative methodologies to understand, model and address risk exposures.

The project is structured to explore the key components of market risk. Divided into distinct sections corresponding to specific questions, it covers the estimation of value at risk (VaR) using a kernel, the calculation of expected loss on a stock, the application of extreme value theory (EVT), the evaluation of financial models such as the Almgren and Chriss model, and the analysis of portfolio volatility.

The presentation of the document follows a structured approach, with each section devoted to a particular question or task described in the project guidelines. The answers contained in these sections include code extracts (Python and Excel) and an interpretation of the results.

In the remainder of the project, we will use a dataset containing Natixis share prices from the beginning of 2015 to the end of 2018.

Happy reading!

2 Question A - Estimation of VaR

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 $K(u) = \frac{15}{16}(1-u^2)^2 \cdot 1_{|u|\leq 1}$)
b – Which proportion of price returns between January 2017 and December 2018 exceed the VaR threshold defined in the previous question? Do you validate the choice of this non-parametric VaR?

2.1 VaR Estimation

In our Python code we started by loading data from a specified text file, organizing it by date in a dictionary format for easy manipulation. The next process involves calculating daily returns for the years 2015 to 2018.
The main objective is to estimate the Value at Risk (VaR) on price returns between 2015 and 2016. Using a non-parametric approach with the kernel biweight function, the code calculates VaR at different confidence levels.
We've also created several small functions that we'll be using later (quantile mean and standard deviation calculation).
In addition to employing the non-parametric biweight kernel method to estimate VaR, a classical historical method was employed to calculate VaR for gains and losses at various confidence levels. This approach involves the direct computation of quantiles from the historical return data. The calculated quantiles represent the loss and gain thresholds at specified confidence levels, serving as a reference for subsequent analyses to assess the consistency of results obtained using different VaR estimation techniques.

Output:

```
Estimated VaR for 2015-2016 :

Confiance        VaR_loss                     VaR_gain

90.0%            -0.02780410742496053        0.02819865319865327

95.0%            -0.038933559035124875        0.03788316946211676

99.0%            -0.05704225352112663        0.055596196049743904
```

These values represent the VaR for losses and gains at specified confidence levels (90%, 95%, and 99%), indicating the potential loss or gain that could be exceeded with the given confidence level over the period of 2015-2016.
Here's the code to calculate kernel's distribution function. We already have the kernel density function with the given formula.

Kernel density is a method of estimating the probability distribution of a random variable from a sample of data. For a kernel function $K(u)$, the kernel density is calculated as follows:

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

where :

$\hat{f}(x)$ is the estimated kernel density for the value x,

n is the sample size,

h is the bandwidth,

x_i are the sample observations.

The kernel function $K(u)$ defines the relative weight assigned to each observation around the value x. It can take various forms, such as the Gaussian, the uniform, or in the specific case of the biweight kernel :

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

This function assigns a decreasing weight to observations farther away than x and a zero weight to observations farther than a certain distance, controlled by the condition $|u| \leq 1$ in the indicator $\mathbb{1}$.

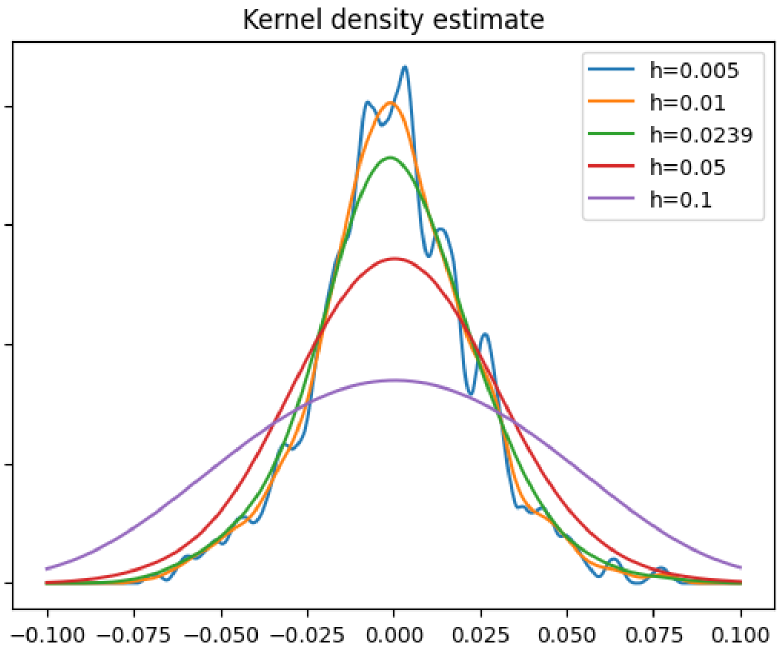
All we need to do is now integrate it to deduce its distribution function.

$\hat{F}(x) = \int_{-\infty}^{x} \hat{f}(x)\, dx$

Input:

```
def biweight_kernel(u):

    if abs(u) <= 1:

        return (15 / 16) * (1 - u ** 2) ** 2

    else:

        return 0



def kernel_density_estimate(x, data, h):

    n = len(data)

    total = 0

    for xi in data:

        total += biweight_kernel((x - xi) / h)

    return total / (n * h)



# Estimate an integral by trapzes method

def trapezes(f, a, b, n, data, h):

    step = (b - a) / n

    sum = 0

    for i in range(1, n):

        xi = a + i * step

        xj = a + (i + 1) * step

        sum += f(xi, data, h) + f(xj, data, h)

    return sum * step / 2
```

In our code, we decided to use the trapezoid method to numerically estimate the integral of a function by approximating the region under the curve of the function by a series of trapezoids. We also plotted the density as a function of several values of h to select the most relevant one. In the resulting graph, each curve represents the kernel density estimate for a specific value of h. By visually examining these curves, one can choose the value of h that appears to provide the best estimate of the kernel density. In this case, the value h=0.01 was selected, as it provides an adequate and consistent estimate of the kernel density for the specific data used in this analysis.



We then estimated the value at risk (VaR) for the period 2015-2016 using the kernel density estimation method. Here is an explanation of the code:

Input:

```
def estimate_var(data, h, probability):

    # Estimation of the distribution function using the trapezoid method

    def cumulative_distribution(x, data, h):

        return trapezes(kernel_density_estimate, min(data), x, 1000, data, h)



    # Find the value where the distribution function reaches the probability level

    lower_bound = min(data)

    upper_bound = max(data)



    # Dichotomy method to find the value corresponding to the probability level

    while upper_bound - lower_bound > 1e-5:

        middle = (lower_bound + upper_bound) / 2

        if cumulative_distribution(middle, data, h) < probability:

            lower_bound = middle

        else:

            upper_bound = middle



    return upper_bound
```

estimate-var(data, h, probability): This function aims to estimate the Value at Risk (VaR) for a given data series, a window width h for the kernel density, and a specific probability level. It uses the trapezoid method to estimate the cumulative distribution function, then finds the value of x corresponding to the given probability level.

cumulative-distribution(x, data, h): This is an internal function to estimate-var which estimates the cumulative distribution function using the trapezoid method for each x in the data.
We search for the value corresponding to the given probability level using a dichotomy method, this function iterates until the value of the cumulative distribution function reaches the specified probability level.
We then return the value of x corresponding to the given probability level, once the cumulative distribution function reaches the given probability level, the function returns the corresponding value of x.

Output:

```
Estimated VaR for 2015-2016 :

Confiance        VaR

90.0%            -0.027826494540893393

95.0%            -0.038123774394452845

99.0%            -0.05683570780710162
```

We can see that these results seem consistent with those found above with the simple historical method.

2.2 Proportion of Returns Exceeding VaR

Input:

```
var_2015_2016_kernel = estimate_var(returns_2015_2016, h, 0.05)


# Calculating returns between 2017-2018

returns_2017_2018 = calculate_returns(Prix_2017_2018)


# proportion of returns exceeding calculated VaR

rendements_sup_var = len([x for x in returns_2017_2018 if x <= var_2015_2016_kernel])

proportion = rendements_sup_var / len(returns_2017_2018)


# Display of the proportion of returns exceeding VaR

print(f"The proportion of returns exceeding VaR is : proportion*100%")

print(f"There are rendements_sup_var on len(returns_2017_2018)")
```

Output:

```
The proportion of returns exceeding VaR is : 1.5717092337917484%

There are 8 on 509
```

This code segment aims to answer the exercise question regarding the proportion of price returns between January 2017 and December 2018 that exceed the VaR threshold defined in the previous question.
The code simply counts the number of returns that exceed the estimated VaR threshold for 2015-2016.
The proportion of returns exceeding the VaR threshold is 1.57%. This means that over the period January 2017 to December 2018, approximately 1.57% of returns exceeded the VaR threshold estimated for the year 2015-2016 with a confidence level of 95%.
Concerning the validation of the choice of this non-parametric VaR, a low proportion of returns

exceeding the estimated threshold (< 5%) may indicate a possible underestimation of the risk by the model. This suggests that this VaR model may be slightly too conservative or less sensitive to extreme events. But we can also conclude that our VaR is rather consistent because the proportion of returns above it does not exceed 5%

3 Question B - Calculation of Expected Shortfall

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

Input:

```
# Identify returns less than VaR during 2015-2016

losses = [x for x in returns_2015_2016 if x <= var_2015_2016_kernel]



# Calculate Expected Shortfall (ES = average loss above VaR)

expected_shortfall = sum(losses) / len(losses)



print(f"The Expected Shortfall for VaR is : expected_shortfall")
```

Output:

```
The Expected Shortfall for VaR is : -0.05336181815045533
```

This piece of code calculates the Expected Shortfall (ES) for the VaR estimated in question A and comparing this result to the VaR itself.
The code works as follows:
Identification of losses below VaR: It identifies returns that are less than or equal to the VaR estimated for the year 2015-2016.
Calculation of the Expected Shortfall (ES): Once the returns corresponding to the losses have been identified, the code calculates the Expected Shortfall defined as the average of the losses beyond the VaR.
The Expected Shortfall result for the estimated VaR is -0.053. This means that, on average, losses beyond the VaR threshold are equivalent to -0.053. Compared to the VaR which was -0.038 (for a confidence level of 95%), we observe that the Expected Shortfall is slightly above the VaR.
This observation is consistent with expectations: the Expected Shortfall measures an average of losses beyond the VaR, so it is natural that it is generally higher than the value of the VaR itself.

4 Question C - Analysis using EVT

4.1 Estimate GEV Parameters

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?

Input:

```
# retrieves all available prices

Prix = []

for year in range(2015, 2019):

    for month in range(1, 13):

        Prix += price_data[year][month]



# Calculation of corresponding returns

returns = calculate_returns(Prix)

sorted_returns = sorted(returns)

def k(n):

    return np.log(n)

def Pickands_estim(data):

    n = len(data)

    num = data[n-int(k(n))]-data[n-2*int(k(n))]

    den = data[n-2*int(k(n))]-data[n-4*int(k(n))]

    res = np.log(num/den)

    return res/np.log(2)

# invert everything to be able to calculate the Pikands index on losses

sorted_returns_losses = [-x for x in sorted_returns]

sorted_returns_losses = sorted_returns_losses[::-1]

# Calculating the Pikands index

pikands_gain = Pickands_estim(sorted_returns)

pikands_losses = Pickands_estim(sorted_returns_losses)
```

Output:

```
Pikands Index :

Losses                    Gain

0.5089715779341741 0.5772338569463286
```

Let (X_n) be a sequence of independent and identically distributed random variables, with a distribution function F belonging to the maximum domain of attraction of a GEV distribution with parameter $\xi \in \mathbb{R}$. Let k be a function from $\mathbb{N}$ to $\mathbb{N}$ such that

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

and

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

then the Pickands estimator, defined as:

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

converges in probability to ξ. Moreover, if

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

then the convergence of $\hat{\xi}_{P,k(n),n}$ to ξ occurs almost surely rather than just in probability.
In our code, we've decided to use k(n) = log(n), as this function corresponds well to the excigence expressed above.
The Pickands indices obtained for the tails of the distribution of Natixis returns are 0.577 for gains and 0.509 for losses. These values indicate an asymmetry in the distribution of returns: a relatively higher concentration of extreme gains compared to extreme losses.
However, these results are sensitive to the choice of the function k(n) in the estimation and are based on theoretical assumptions.

4.2 Value at Risk Calculation using EVT

Calculate the value at risk based on EVT for various confidence levels, assuming iid returns.

Input:

```
def VaR_EVT(p, data, pickands):

    n = len(data)

    num = ((k(n)/(n*(1-p)))**pickands - 1) * (data[n-int(k(n))]-data[n-2*int(k(n))])

    den = 1 - 2**(-pickands)

    return num/den + data[n-int(k(n))]
```

Output:

```
VaR EVT :

Confiance          VaR loss                     VaR gain

90.0%             -0.018676315001953256         0.034136255033759964

95.0%             -0.03459467977312686         0.03787128303972365

99.0%             -0.05563667261604414         0.05522424941671052
```

In the course we have the formula for finding the VaR at a certain confidence level p for the pickands estimator:

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

We can see that the values for the VaR found are rather consistent with our previous results.

5 Question D - Almgren and Chriss model

5.1 Estimate all the parameters

In this part, we need to estimate the parameters of the model of Almgren adn Chriss. This model is used to find the best way to liquidate a portfolio. In our case, we need to liquidate a certain quantity of stock, to do that, we got an order book of the transaction in one day. This order book reflects trades over 24 hours segmented into 1001 trades whose time is weighted from 0 to 1 (1 = 24 hours).
The columns of our order book are:

- A time column
- A Bid-Ask spread column
- A volume column (if it's known)
- A sign column (1: buy; -1: sell)
- A price column (each price is before the transaction)

The first that we did was to import the dataset, change the names of the columns because it was more useful and create 3 new arrays : volume, price and sign.

Input:

```python
volume = []
price = []
sign = []
for i in range(len(data) - 1):
    if pd.notna(data['volume'].iloc[i]): #Recup the data when we know the volume
        volume.append(data['volume'].iloc[i])
        price.extend([data['price'].iloc[i], data['price'].iloc[i + 1]])
        sign.append(data['sign'].iloc[i])
```

Before begin, we need to know which parameters are required to use the model. On the, we got from page 132 to 136 the explanations of the model of Almgren ad Chriss. The purpose of the model is to find the good paramters to avoid on one side the impact of our liquidation and on another side the market risk of it.
The two parameters that we need to find are the γ and the η parameters.
These parameters can be obtain by using the LinearRegression function in python. First overall we need to have the returns and the volatility. After doing that, we need to know on which parameters do the linear regression.
If If we look at the slides 132, 133, and 134, we saw that to get the parameters η, we have the function

$$h\left(\frac{n_k}{\tau}\right) = \epsilon \cdot \text{sign}(n_k) + \eta \cdot \frac{n_k}{\tau}$$

where n_k represents the liquidation tranches. We also know that $x_k = X - \sum_{j=1}^{n_j} P_j$. Where x_k represents the quantity liquidate at the k-th tranche.

So to estimate eta we need to have the volume (and know if it's buy or sell), the difference between two prices (before the transaction and after) and volume squared (with - for selling and + for buying). The function h represents the transitory impact of an order on the book. To have that we do :

Input:

```
signed_volume = np.array([sign[i]*volume[i] for i in range(len(volume))])

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

signed_volume_squared = np.array([sign[i] * volume[i] * volume[i]
for i in range(len(price_diff))])
```

After that we need to specify the variables for the regression model and create the LinearRegression model.
The variables are defined like this :

Input:

```
gamma_prices = price_diff
eta_prices = -price_diff
X_gamma = signed_volume.reshape(-1, 1)
X_eta = np.column_stack((signed_volume, signed_volume_squared))
model = LinearRegression()
```

Now that we have our model, the next step is to estimate η. This involves fitting the model with X_η and eta_prices. We create our time step which represent when we can do a transaction (at each hour) and create η which is equal to the first coefficient after fitting the model multiply by τ :

Input:

```
model.fit(X_eta, eta_prices)
tau=1/24
eta = model.coef_[1] * tau
```

Now we got η and we can calculate the mean squared error like this :

Input:

```
y_pred_eta = model.predict(X_eta)
mse_eta = mean_squared_error(eta_prices, y_pred_eta)
r2_eta = r2_score(eta_prices, y_pred_eta)

print(f"Estimation of eta: eta")
print(f"Mean Squared Error: mse_eta ; R^2: r2_eta")
```

Our results are : Estimation of η:2.280706114725874 $\times 10^{-8}$ Mean Squared Error: 0.00023873091719548648; R^2: 0.9607480862831544

We got η; now we need to estimate γ. By using the same method, we fitted the model with X_γ and gamma_prices this time and γ represent the slope of the regression line, the function g represents the lasting impact on the order book. We recovered γ which is the first coefficient of the model. By the same method, we got for γ :

Input:

```
model.fit(X_gamma, gamma_prices)
gamma = model.coef_[0]

y_pred_gamma = model.predict(X_gamma)
mse_gamma = mean_squared_error(gamma_prices, y_pred_gamma)
r2_gamma = r2_score(gamma_prices, y_pred_gamma)

print(f"Estimation of gamma: gamma")
print(f"Mean Squared Error: mse_gamma ; R^2: r2_gamma")
```

Our results are : Estimation of γ:0.00050237805904092 Mean Squared Error: 0.0004907505222142958; R^2: 0.9193112589657632
So, we got our parameters estimated, we can apply the Almgren and Chriss model.

5.2 liquidation strategy

We take a quantity to liquidate arbitrary. Let's take 15000. We know that we can make only one transaction by hour. We choose our taste of risk (aversion) arbitrary, let's take $\lambda : 10^{-4.0}$. In the lecture, the page 136 states that an efficient frontier for several levels of risk aversion corresponding to the Lagrangian λ. Euler-Lagrange solution : This efficient frontier is defined by :

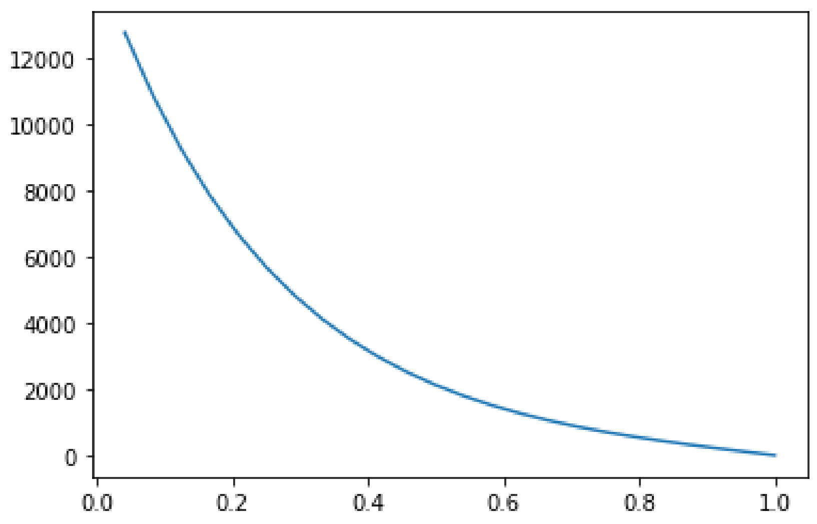
$$x_k = \frac{\sinh\left(K\left(T - \left(k - \frac{1}{2}\tau\right)\right)\right)}{\sinh(KT)}X, \quad \text{where } K\tau \to 0 \sim \sqrt{\frac{q\lambda\sigma^2}{\eta}} + O(\tau).$$

In this formula x_k represents the liquidation quantity that we need to liquidate after the k-th transaction. T represents the time of the last transaction in our case it's 1 ($\frac{24}{24}$ =1), and $k - \frac{1}{2}\tau$ represents the time it takes to complete a transaction ($\frac{1}{24}$,$\frac{2}{24}$,...,$\frac{24}{24}$)
We applied this on python :

Input:

```
X_to_liquidate=15000

possible_transaction=[(1/24)*i for i in range(1,25)]

Lambda_value = 10**(-4.0)

K_value = np.sqrt(Lambda_value * (sigma_total**2) / eta)

liquidation_quantity=[]

for t_value in possible_transaction :

    liquidation_quantity.append(np.sinh(K_value * (1 - t_value)) *
                X_to_liquidate / np.sinh(K_value * 1))
```

By doing that, the result of the optimal liquidation is :


12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

6 Question E - volatility analysis

6.1 correlation matrix

6.2 volatility directly from the series of prices

In this final part, we to determine the volatility of a porfolio which contains 3 FX rates.
These 3 FX rates are :

- GBP-EUR
- SEK-EUR
- CAD-EUR

We got the Date, the High and Low, that value are taken each 15minutes.
Firs overall, we created 3 different data sets with the returns of the bid-ask, to have the returns of it we need to have the mean between the bid and the ask price.
In python we do :

Input:

```
data=pd.read_excel("Dataset TD5.xlsx")
data_GBP=data.iloc[2:, :3]
data_SEK=data.iloc[2:, 4:7]
data_CAD=data.iloc[2:, 8:11]

half_bidask_GBP=[(data_GBP.iloc[i,1]+data_GBP.iloc[i,2])/2 for i in
range(0,len(data_GBP)-1)]
half_bidask_SEK=[(data_SEK.iloc[i,1]+data_SEK.iloc[i,2])/2 for i in
range(0,len(data_SEK)-1)]
half_bidask_CAD=[(data_CAD.iloc[i,1]+data_CAD.iloc[i,2])/2 for i in
range(0,len(data_CAD)-1)]

returns_gbp=[(half_bidask_GBP[i+1]/half_bidask_GBP[i])-1 for i in
range(0,len(data_GBP)-2)]
returns_sek=[(half_bidask_SEK[i+1]/half_bidask_SEK[i])-1 for i in
range(0,len(data_GBP)-2)]
returns_cad=[(half_bidask_CAD[i+1]/half_bidask_CAD[i])-1 for i in
range(0,len(data_GBP)-2)]
```

Now that we have our data sets, we will try to do the correlation matrix by using the wavelets.
In the lecture, from page 282 to 287, we have the part on the wavelets. We got the formula of the derived wavelet functions from a mother wavelet:

$$\psi_{j,k}(t) : t \in \mathbb{R} \rightarrow 2^{j/2}\Psi(2jt - k), \text{ where } j \in \mathbb{R} \text{ is the resolution level and } k \in \mathbb{R}.$$

For a discrete wavelet analysis, $j \in \mathbb{Z}$ and $k \in \mathbb{Z}$.

We got four other formulas. The first one is an example : the Haar wavelet :

$$\Psi_{\text{Haar}}(t) : t \in \mathbb{R} \rightarrow 1_{[0,1/2)}(t) - 1_{[1/2,1)}(t).$$

The second one is a scaling function ϕ construct by successive dyadic interpolations in a cascade algorithm :

$$\phi(x) = \sum_{k=0}^{2N-1} a_k \phi(2x-k)$$

The third is the function of the mother wavelet which is defined by :

$$\Psi(x) = \sum_{k=0}^{2N-1} (-1)^k a_{1-k} \phi(2x-k)$$

And the last one is the empirical wavelet coefficient :

$$\hat{z}_{\text{empirical},j,k} = \sum_{n=1}^{N} z(u_n) \psi_{j,k}(u_n)(u_n - u_{n-1})$$

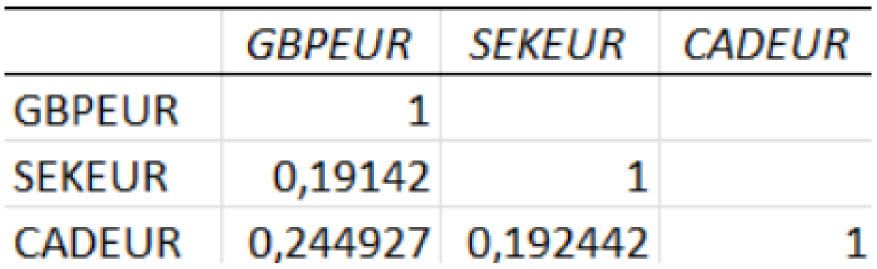
where u_n represents a discrete grid.

We tried to define functions in python to do this part but we didn't succeed. We did :

Input:

```
def calculate_empirical_moment(data, k, N):
    result = 0
    for i in range(1, len(data)*N):
        result += np.abs(data[int(i/N)] - data[int((i-1)/N)])**k
    return result * (1/N * len(data))

def calculate_empirical_moment_halved_resolution(data, k, N):
    result = 0
    for i in range(1, int((len(data)*N)/2)):
        result += np.abs(data[int(2*i/N)] - data[int(2*(i-1)/N)])**k
    return result * (2/N * len(data))
```

But after this we were blocked so we did the correlation matrix using Excel. The result is :



	GBPEUR	SEKEUR	CADEUR
GBPEUR	1		
SEKEUR	0,19142	1	
CADEUR	0,244927	0,192442	1

Now we have our correlation matrix. The next part is to find the Hurst exhibitor of our data sets.To do that we need 3 course formulas that are on slides 262 and 263.

The first one is the formula for the empirical absolute moment on a sample interval $[0, T]$ with $1/N$ spacings is:

$$M_k = \frac{1}{N}\sum_{i=1}^{N}\left|X\left(\frac{i \cdot T}{N}\right) - X\left(\frac{(i-1) \cdot T}{N}\right)\right|^k$$

After that, we need to have the empirical moment divided in two resolutions (halved resolution) :

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

The last one is the formula of the estimate Hurst exhibitor :

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

we know that $\hat{H}$ converges almost surely toward H

Now, we need to create functions in Python to do these formulas :

Input:

```
def calculate_empirical_moment(data, k, N):
    result = 0
    for i in range(1, len(data)*N):
        result += np.abs(data[int(i/N)] - data[int((i-1)/N)])**k
    return result * (1/N * len(data))

def calculate_empirical_moment_halved_resolution(data, k, N):
    result = 0
    for i in range(1, int((len(data)*N)/2)):
        result += np.abs(data[int(2*i/N)] - data[int(2*(i-1)/N)])**k
    return result * (2/N * len(data))

def estimate_Hurst(data, k, N):
    M2 = calculate_empirical_moment(data, k, N)
    M2_halved = calculate_empirical_moment_halved_resolution(data, k, N)
    return 0.5 * math.log2(M2_halved/M2)
```

With these formulas, we can find the Hurst exponents of the returns of our FX rates :

Input:

```
hurst_gbp= estimate_Hurst(returns_gbp, 2, 10)
hurst_sek=estimate_Hurst(returns_sek, 2, 10)
hurst_cad=estimate_Hurst(returns_cad, 2, 10)
```

The three Hurst exhibitor are equals to 0.5.
Now that we got the Hurst exhibitors and the correlation matrix, we can calculate the covariance matrix :

To do that, we need to have a method to get the scale volatility by using the Hurst exhibitors, after that we can diagonalized this result to have a matrix and do the multiplication between this matrix of scaled volatility and the correlation matrix.
To do that, we created two method, and do the multiplication between the two matrix :

Input:

```
scale_vol=[]
scale_vol.append(scale_volatility(calculate_std_dev(returns_gbp),hurst_gbp))
scale_vol.append(scale_volatility(calculate_std_dev(returns_sek),hurst_sek))
scale_vol.append(scale_volatility(calculate_std_dev(returns_cad),hurst_cad))

def calculate_covariance_matrix(correlation_matrix, volatility_vector):
    correlation_matrix = np.array(correlation_matrix)
    Sigma = np.diag(volatility_vector)
    covariance_matrix = np.dot(Sigma, np.dot(correlation_matrix, Sigma))
    return covariance_matrix
covariance_matrix = calculate_covariance_matrix(correlation_matrix, scale_vol)
```

The result of the scale volatility are :

$$\begin{array}{l} 0.0008814409953038501 \\ 0.00046263269044024267 \\ 0.0007159859130200779 \end{array}$$

The result of the covariance matrix is :

$$\begin{bmatrix} 7.76938228 \times 10^{-7} & 7.80579310 \times 10^{-8} & 1.54573173 \times 10^{-7} \\ 7.80579310 \times 10^{-8} & 2.14029006 \times 10^{-7} & 6.37442282 \times 10^{-8} \\ 1.54573173 \times 10^{-7} & 6.37442282 \times 10^{-8} & 5.12635828 \times 10^{-7} \end{bmatrix}$$

We can conclude that the variance of each FX rates is really weak so they doesn't experience large fluctuation.
The off-diagonal elements represent the co-variances between asset pairs. All off the covariance are positive this implies that the assets tend to move together but their values are too weak to conclude on this.
Overall, the portfolio seems to exhibit low individual asset volatility, and there's a suggestion of a positive relationship between assets, although the strength of this relationship is weak and should be interpreted with caution.

On the second part, we need to determine the volatility of the portfolio directly from the series of prices of the portfolio. To do that, we decided to try to create different data sets for different time scales :

Input:

```
column_names = ['Date', 'High', 'Low']

data_GBP.columns = column_names
data_SEK.columns = column_names
data_CAD.columns = column_names

data_GBP['Date'] = pd.to_datetime(data_GBP['Date'])
data_SEK['Date'] = pd.to_datetime(data_SEK['Date'])
data_CAD['Date'] = pd.to_datetime(data_CAD['Date'])

data_GBP.index = pd.to_datetime(data_GBP.index)
data_SEK.index = pd.to_datetime(data_SEK.index)
data_CAD.index = pd.to_datetime(data_CAD.index)

date_ranges = [
    ('2016-03-07 09:00:00', '2016-03-07 10:00:00'),
    ('2016-03-07 09:00:00', '2016-03-07 19:00:00'),
    ('2016-03-07 09:00:00', '2016-03-08 09:00:00'),
    ('2016-03-07 09:00:00', '2016-03-10 09:00:00'),
    ('2016-03-07 09:00:00', '2016-03-14 09:00:00'),
    ('2016-03-07 09:00:00', '2016-04-07 09:00:00'),
    ('2016-03-07 09:00:00', '2016-06-07 09:00:00')
]
```

But after created this samples, we didn't succeed to create the data set with the final date that are each values on the right, so we decided to apply the two method (overlapping and non-overlapping) on the whole data set.
With the Overlapping method, each 15's minutes (in this case) return is computed relative to the previous 15's minutes closing price. This is the classical method which is largely used in finance.
With the Non-Overlapping method, each 15's minutes return is computed relative to the initial 15's minutes closing price.

The choice between sliding (overlapping) and non-sliding (non-overlapping) windows depends on analysis goals. Use sliding windows for increased stability and more data points, beneficial for parameters estimation. Choosing non-sliding windows for independent observations, critical for certain statistical analyses.We need to consider the trade-off between stability and independence based on specific data characteristics and analysis objectives.

So, for the whole data set we did the two methods and the results are:

Result with Overlapping:

- Standard deviation of GBP: 0.0006232729049951721
- Standard deviation of GBP (Annualized): 0.002159080676865376
- Standard deviation of SEK: 0.00032713071260887243
- Standard deviation of SEK (Annualized): 0.0011332140299095596
- Standard deviation of CAD: 0.00032713071260887243
- Standard deviation of CAD (Annualized): 0.0011332140299095596

Result with Non-Overlapping:

- Standard deviation of GBP: 0.03799434923003744
- Standard deviation of GBP (Annualized): 0.13161628653388058
- Standard deviation of SEK: 0.012694382028994778
- Standard deviation of SEK (Annualized): 0.04397462928981649
- Standard deviation of CAD: 0.010946213754529869
- Standard deviation of CAD (Annualized): 0.037918796746710025

The interpretation is:

- The standard deviations for GBP, SEK, and CAD are significantly smaller when using overlapping data compared to non-overlapping data. This suggests that using overlapping data leads to lower estimates of volatility.
- The annualized standard deviations provide a measure of volatility over a one-year period. For each currency, the annualized volatility is higher when using non-overlapping data, indicating a higher perceived risk over a one-year horizon.
- The difference in results between the two methods may be attributed to the inclusion or exclusion of overlapping data points. We saw that the choice of method can significantly impact the calculated volatility and that the interpretation should consider the characteristics of the financial data and the objectives of the analysis.