Market Risk Project

BONNEAU Nathan
CHARBONNIER Thibault

Summary

Import Data

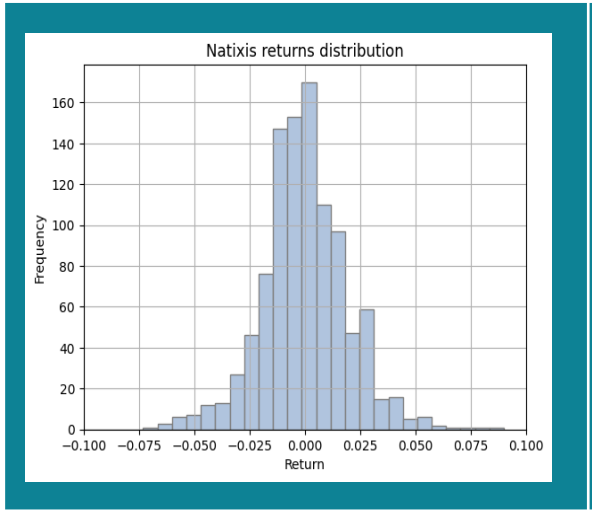
Preamble : We are working with the daily prices of Natixis stock between January 2015 and December 2016.
We used the *read_csv* function of *Pandas* to import them.

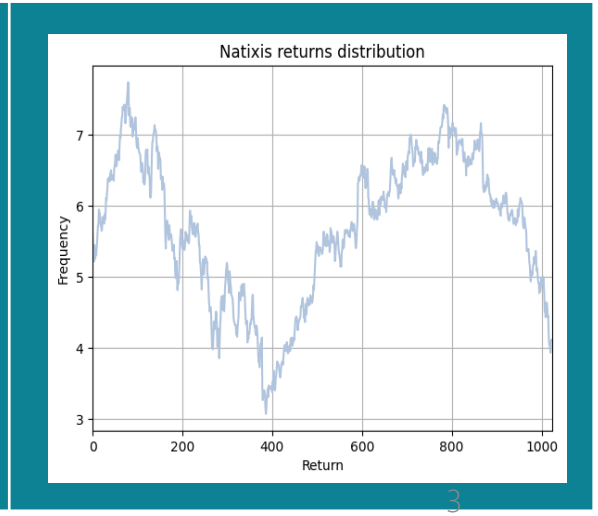
- Calculation of arithmetic returns for each day with the formula : $R_t = \frac{P_t - P_{t-h}}{P_{t-h}}$

- We add a column to our *data_natixis* that we call *returns* to store the returns :

```
data_natixis = data_natixis.rename(columns={0:"date",1:"stock_value"})
data_natixis['date'] = pd.to_datetime(data_natixis['date'])
data_natixis["stock_value"] = data_natixis["stock_value"].str.replace(',', '.').astype(float)
stock_values = data_natixis['stock_value']
returns = np.zeros(len(stock_values))
for i in range(1, len(stock_values)):
    returns[i] = (stock_values[i] - stock_values[i-1]) / stock_values[i-1]
data_natixis['returns'] = returns
data_natixis['returns'] = data_natixis['returns'].round(3)
```

- We now plot the returns and their distribution with the function *plot* from *matplotlib*. We can thus have a better idea of the representation of our returns and the historical distribution.





Question A

Preamble : The returns are stationary (Hypothesis) : they are distributed according to the same distribution on equal sized periods. Through this question, we will evaluate the historical Value at Risk (VaR) with a non-parametric distribution (biweight Kernel)

- The parameter h of the Kernel function represents the smoothing parameter. We can determine this with Scott's Rule for Bandwidth. We need to calculate the empirical variance.

$$h = \left(\frac{4 * \hat{\sigma}^5}{3 * n}\right)^{\wedge\left(\frac{1}{5}\right)} \approx 1{,}06\hat{\sigma} * n^{-0.2}$$

```
def get_scott_bandwidth(data, sigma):
    n = len(data)
    bandwidth = 1.06 * sigma * n**(-0.2)
    return bandwidth

rdt = data2015_2016['returns']
mean = round((np.sum(rdt))/len(rdt),5)
std = round(((np.sum((rdt - mean)**2))/(len(rdt)-1))**0.5,5)
scott_h = get_scott_bandwidth(rdt, std)
```

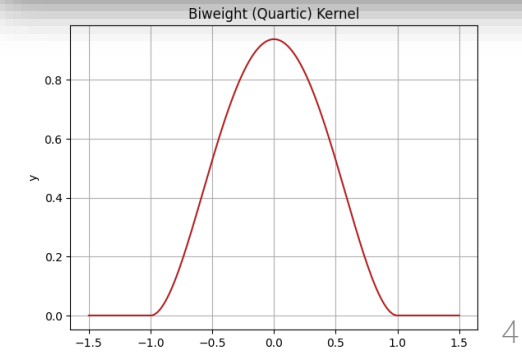
→ The smaller is h, the smaller is the bias and the higher is the variance

- Definition of biweight Kernel and Implementation :

$$K(u) = \frac{15}{16} * (1 - u^2)^2$$

Support : $|u| \leq 1$

```
def get_scott_bandwidth(data, sigma):
    n = len(data)
    bandwidth = 1.06 * sigma * n**(-0.2)
    return bandwidth
kernel_bi = biweight_kernel(np.linspace(-1.5, 1.5, 300))
```



Biweight Kernel representation

→ The biweight kernel assigns a lower weight to observations far from the median. It can be used when the data is relatively symmetrical.

Question A

- Estimation of the Kernel density function:

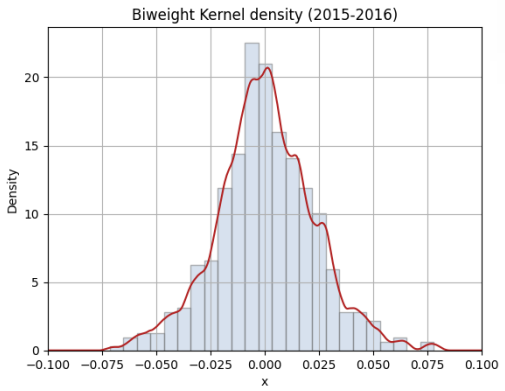
$$\hat{f}(x) = \frac{1}{n * h} \sum_{i=1}^{n} K(\frac{x - Xi}{h})$$

with Xi = the return ri

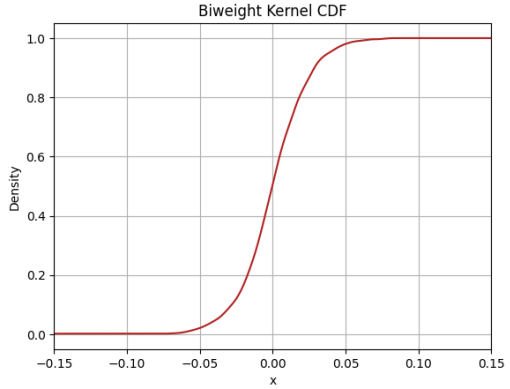
h = band width

We generate a sample of values between our min and max returns in order to estimate our kernel density function.

We notice that our density corresponds to the sum of small Gaussians around each observation.




```
def kernel_density_estimate(data, x, bandwidth):
    kernel_vals = 0
    for i in range(len(data)):
        kernel_vals += biweight((x - data.iloc[i]) / bandwidth)
    return kernel_vals / (len(data) * bandwidth)

x_values = np.linspace(np.min(rdt)-0.1, np.max(rdt)+0.1, 1000)
PDF = [kernel_density_estimate(rdt, x, scott_h) for x in x_values]
```



- From the cumulative sum at each moment, we estimate the cumulative distribution function.

```
integration_step = x_values[1] - x_values[0]
CDF = np.cumsum(PDF) * integration_step
```

We notice a strictly increasing cumulative distribution function between 0 and 1, this seems coherent.

Question A

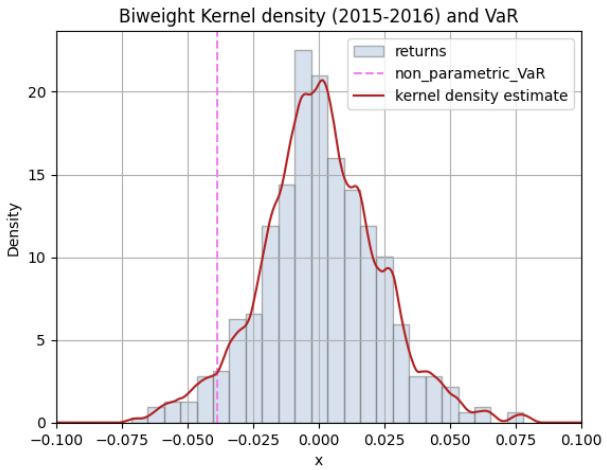
The VaR of a Gaussian Kernel is the maximum expected loss for a given confidence level.

- We have : $\widehat{Fn}(x) = \alpha$

We can solve the following Optimization problem to find the VaRα : $\underbrace{Argmin}_{x \in \mathbb{R}} \left(\widehat{Fn}(x) - \alpha\right)^2 = VaR_\alpha$

```
def get_Non_Param_VaR(data, alpha):
  min = np.argmin((CDF - alpha)**2)
  return round(x_values[min],5)

Non_parametric_VaR = get_Non_Param_VaR(rdt, 0.05)
print("Non-parametric Value at Risk :",Non_parametric_VaR)
```

```
Non-parametric Value at Risk : -0.03863
```



Now, we will evaluate the relevance of the VaR estimated both on sample data and out-sample data.

Question A

In sample (2015-2016) :

We count the % of our sample returns (*rdt*) that exceed our previous VaR estimate.

```
NpVaR_exceeding = (rdt <= Non_parametric_VaR).sum()
NpVaR_proportion = round(NpVaR_exceeding  / len(rdt),4)
print("With Non-parametric VaR:")
print(NpVaR_exceeding," returns on ", len(rdt), " exceed the VaR threshold")
print("Corresponding to",round(NpVaR_proportion*100,5),"%, for a risk level of 5%")
```

```
With Non-parametric VaR:
26  returns on  513  exceed the VaR threshold
Corresponding to 5.07 %, for a risk level of 5%
```

Out of sample (2017-2018):

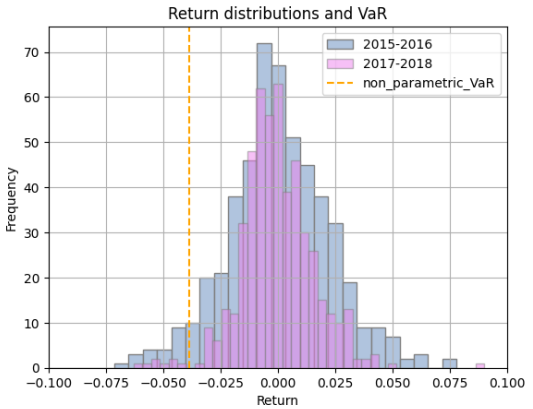
After only taking data between 2017-2018, we estimate the % of our *rtd2* returns that exceed our previous VaR estimate.

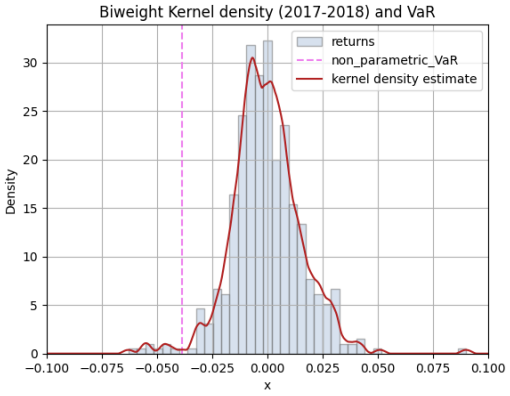
```
data2017_2018 = data_natixis[(data_natixis['date'] >=  datetime(2017, 1, 1)) & (data_natixis['date'] <= datetime(2018, 12, 31))]
rdt2 = data2017_2018['returns']

NpVaR_exceeding2 = (rdt2 <= Non_parametric_VaR).sum()
NpVaR_proportion2 = round(NpVaR_exceeding2 / len(rdt2),4)
print("With Non-parametric VaR:")
print(NpVaR_exceeding2," returns on ", len(rdt2), " exceed the VaR threshold")
print("Corresponding to",round(NpVaR_proportion2*100,5),"%, for a risk level of 5%")
```

```
With Non-parametric VaR:
8  returns on  510  exceed the VaR threshold
Corresponding to 1.57 %, for a risk level of 5%
```

For the “In Sample” part, we obtain a proportion of 5.07% which is a good result. However, for the “Out Sample” part, only 1.57% is above the estimated VaR: it is therefore a poor risk measure.

Historical VaR assumes that future returns follow the same distribution law as past returns. However, over the 2017-2018 period, the variance of returns is much lower (more centered), so there are fewer extreme values. This way, there will be fewer values that fall outside the VaR.





Question B

Preamble : Now, we will calculate the expected shortfall (ES) from the VaR calculated previously.

The Excepted Shortfall (ES) represents the average of losses above the VaR. We can therefore write (since the underlying distribution is a continuous distribution) : $ES_\alpha(L) = TCE_\alpha(L) = E(L\,|L \geq VaR_\alpha(L))$

- We need the empirical average of the number of returns located above the VaR.

```
rdt_over_VaR = data2015_2016['returns'][data2015_2016['returns']<= Non_parametric_VaR]
ES = round(sum(rdt_over_VaR) / len(rdt_over_VaR),5)
print("Non parametric VaR :",Non_parametric_VaR)
print("The expected shorfall for the non_parametric VaR in sample (2015-2016) is : ",ES)
```

```
Non parametric VaR : -0.03812
The expected shorfall for the non_parametric VaR in sample (2015-2016) is :  -0.05336
```

Naturally, we find that : $|ES_\alpha(L)| \geq |VaR_\alpha(L)|$

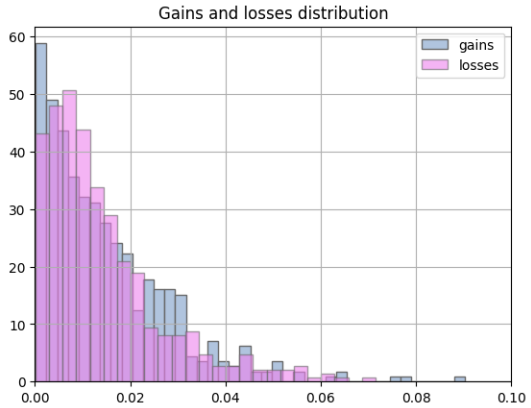
This equality is evident as the ES averages what lies below the VaR.

Question C

Preamble : In this question, we will focus on the estimation of extreme values (distribution tails).
We again use the dataset on Natixis share prices.

- We create two tables which will store the winning and losing returns (passing the losses as positive). Indeed, we must work on the two tails of the distribution

```
gains = data_natixis['returns'][data_natixis['returns']>0]
losses = -data_natixis['returns'][data_natixis['returns']<0]
gains_sorted = sorted(gains)
losses_sorted = sorted(losses)
```



- We calculate the Pickands estimator to estimate the distribution parameters of extreme values.

$$G_\varepsilon(x) = \begin{cases} \exp\left(-(1+\varepsilon x)_+^{-\frac{1}{\varepsilon}}\right) & si\ \varepsilon \neq 0 \\ \exp(-e^{-x}) & si\ \varepsilon = 0 \end{cases}$$

So, if :

- Si ε > 0, the GEV is of the Frechet type
- Si ε = 0, the GEV is of the Gumbel type
- Si ε < 0, the GEV is of the Weibull type

Question C

```python
def pickands_estimator(data):
    n = len(data)
    k_n = np.log(n)
    x1 = data[n-math.floor(k_n)]
    x2 = data[n-2*math.floor(k_n)]
    x3 = data[n-4*math.floor(k_n)]
    pickands = (1/np.log(2)) * np.log((x1-x2)/(x2-x3))
    return round(pickands,3)

pickands_gain = pickands_estimator(gains_sorted)
pickands_losses = pickands_estimator(losses_sorted)
```

```
Pickands estimator for extreme gain : 0.578
Pickands estimator for extreme losses : -0.512
```

- After calculating the Pickands estimator for losses and gains, we compare the value of the estimator with 0. And we can conclude that the GEV for extreme gains is of Frechet type (ε>0) and for extreme gains extreme Weibull type losses (ε<0)

- Calculation of the cumulative distribution of Frechet: $\Phi_\varepsilon(x) = \begin{cases} 0 \; si \; x \leq 0 \\ e^{-x^{-\varepsilon}}, otherwise \end{cases}$

```python
def frechet_cdf(x, param):
  if x <= 0 :
    frechet = 0
  else :
    frechet = math.exp(-x**(-param))
  return frechet
```

- Calculation of the cumulative distribution of Weibull : $\Psi_\varepsilon(x) = \begin{cases} e^{-(-x)^\varepsilon} \; si \; x \leq 0 \\ 1 \, , otherwise \end{cases}$

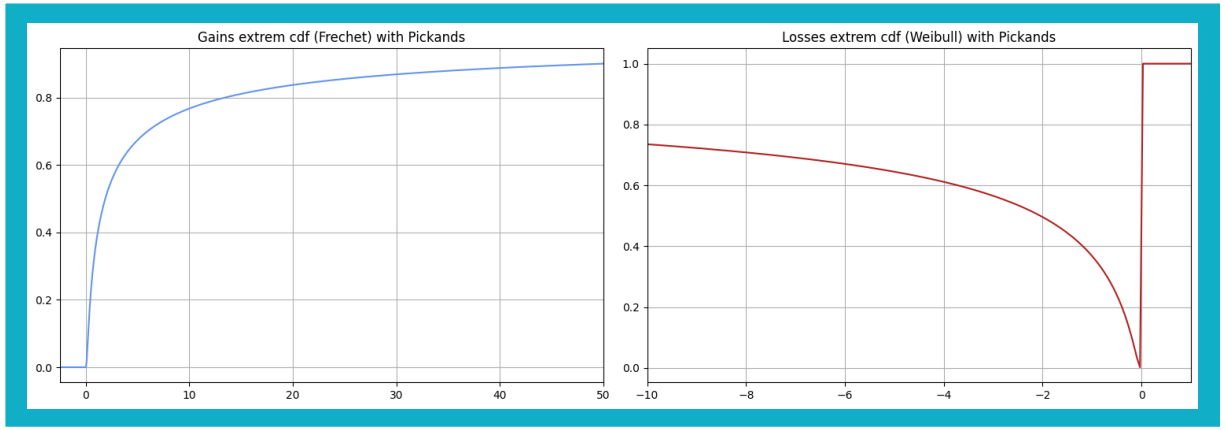
```python
def weibull_cdf(x, param):
  if x <= 0 :
    weibull = math.exp(-(-x)**(param))
  else :
    weibull = 1
  return weibull
```

Question C

Preamble : In this question, we assume that our returns are independent and identically distributed.

- After generating a sample set of values, we display our two cumulative distribution functions.

```
x_values = np.linspace(-10,50,1000)
cdf_frechet = [frechet_cdf(x, pickands_gain) for x in x_values]
cdf_weibull = [weibull_cdf(x, pickands_losses) for x in x_values]
```



- We create a function to calculate the Pickands VaR:

$$here\ k = 1 :\ VaR(p) = \frac{(\frac{1}{n(1-p)})^{\varepsilon} - 1}{1 - 2^{-\varepsilon}} (X_{n:n} - X_{n-1:n}) + X_{n-k+1:n}$$

with $X_{n:n} = data[n - \log(n)]$ and $X_{n-1:n} = data[n - 2\log(n)]$

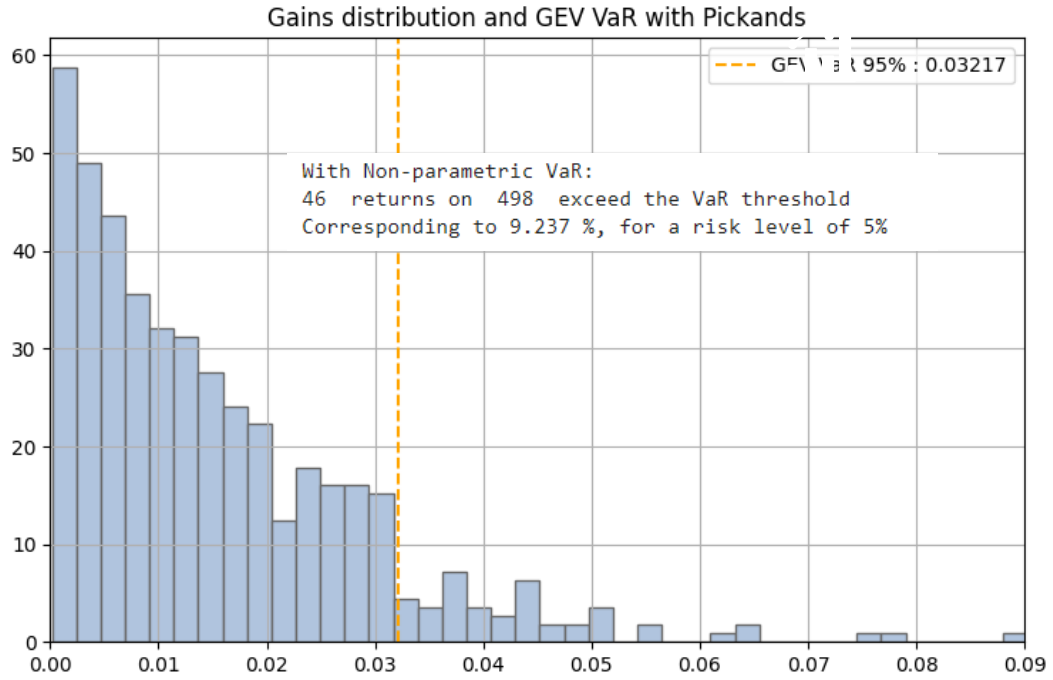
```
def VaR_GEV_Pickands(data, level):
    n=len(data)
    pickands=np.abs(pickands_estimator(data))
    k_n = np.log(n)
    x1=data[n-math.floor(k_n)]
    x2=data[n-2*math.floor(k_n)]
    num=((1/(n*(1-level)))**pickands) - 1
    den=1-2**(-pickands)
    return round((num/den)*(x1-x2) + x1,5)

VaR_GEV_P_gains = VaR_GEV_Pickands(gains_sorted, 0.95)
VaR_GEV_P_losses = VaR_GEV_Pickands(losses_sorted, 0.95)
print("GEV VaR at 95% for the gains :", VaR_GEV_P_gains)
print("GEV VaR at 95% for the losses :", VaR_GEV_P_losses)
```

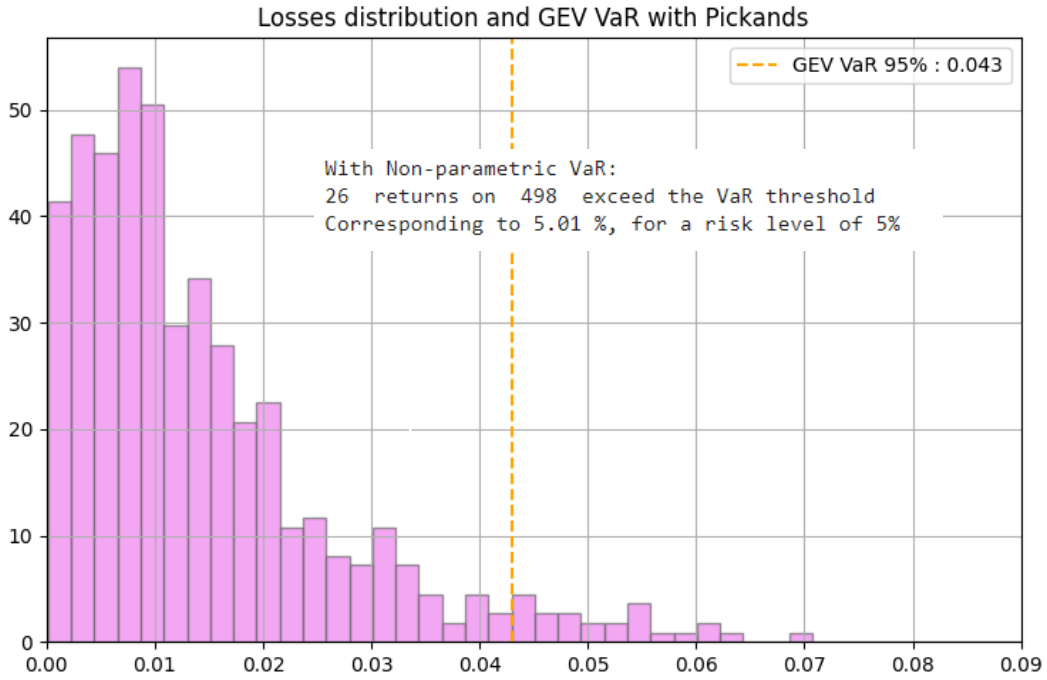
```
GEV VaR at 95% for the gains : 0.03217
GEV VaR at 95% for the losses : 0.043
```

NB: We noticed that we had to take the Absolute Value of the Pickands estimator to avoid having incoherences for the Pickands estimator for losses

Question C

- As for question A, we will evaluate the quality of rating VaR by counting the number of values exceeding our Pickands VaR :





The proportion that exceeds for gains is significant (9.237%), however for losses it corresponds to our level of risk (5.01%). As the gains follows a Fréchet distribution with heavy tails, it's not curious to get a high exceeding of the VaR threshold, the important proportion is the losses exceeding which is very close to the risk level.

Import Data

Preamble : We will now work on a new dataset which has columns on the bid-ask spread, volume, sign of the transaction and the price over a day. We used the *read_csv* function of *Pandas* to import them.

```
import pandas as pd
file_path = "Dataset TD4.xlsx"
data = pd.read_excel(file_path)
```

	Date	Bid-Ask Spread	Volume	Sign	Price
0	0.000202	0.1100	8.0	-1	100.000
1	0.001070	0.1030	NaN	1	99.984
2	0.001496	0.1015	NaN	-1	100.029
3	0.003336	0.0920	NaN	1	99.979
4	0.003952	0.1106	NaN	1	100.060

- We rename the columns of our dataset and then we set the value to "*nan*" in the Volume column for all transactions whose volume is unknown.

- We now create a dataset: *dataVolume* which only includes the rows whose volume we know.

```
data= data.dropna(subset=['Volume'])
dataVolume = data[data['Volume'] != 0].copy()
```

	Date	Bid-Ask Spread	Volume	Sign	Price
0	0.000202	0.1100	8.0	-1	100.000
6	0.004074	0.1294	32.0	1	100.164
16	0.014393	0.1141	8.0	-1	100.048
28	0.022861	0.0978	141.0	-1	99.876
51	0.037864	0.1291	121.0	-1	99.608

Question D : ŋ

Preamble : The goal of this question is to estimate the parameters ŋ, σ and γ of the Almgren and Chriss model

The transient impact formula is defined as : $P_{t+1} - P_{t+2} = h(v_k) = \epsilon * sgn(n_k) + \eta \frac{n_k}{\tau}$

- P_{t+1} price at time t+1
- $h(v_k)$ transient impact
- $\epsilon = \frac{Ask-Bid}{2}$
- $sgn(n_k) = 1\ if\ buy\ ou - 1\ if\ sale$
- n_k is the quantity liquidated at time k
- $\tau = t_k - t_{k-1}$

- Let's start by calculating the transient impact (*transientimpact*) : $P_{t+1} - P_{t+2} = h(v_k)$

```
transientimpact = []
t = []

for i in range(len(dataVolume['Volume']) - 2):
    if dataVolume['Volume'].iloc[i] != 0:
        TI = dataVolume['Price'].iloc[i + 1] - dataVolume['Price'].iloc[i + 2] #Pt+1 - Pt+2
        transientimpact.append(TI)
        t.append(dataVolume['Date'].iloc[i + 2] - dataVolume['Date'].iloc[i + 1]) #diffence between two dates of Pt+1 - Pt+2

print(transientimpact) #transient impact
```

- We define Y to then do the linear regression and find the ŋ : $Y = h(v_k) - \frac{Ask - Bid}{2} * sgn(n_k)$

```
Y = [] #NU * X = Y to solve the linear regression
for i in range(len(dataVolume['Volume']) - 2):
    Y.append(transientimpact[i] - (dataVolume['Bid-Ask Spread'].iloc[i]/2) * dataVolume['Sign'].iloc[i])
print(Y)
```

Question D : ŋ

- We define X as : $X = \frac{n_k}{\tau}$

```
X = []
for i in range(len(dataVolume['Volume']) - 2):
    temp = dataVolume['Volume'].iloc[i] / t[i]

    X.append(temp)
```

- The last step is to calculate the linear regression on ŋ : $Y = \eta X$

We implement the least squares method: the slope corresponds to the coefficient which minimizes the sum of the squares.

```
def LinearRegression(x,y):
  num = np.sum((x-np.mean(x))*(y-np.mean(y)))
  den = np.sum((x-np.mean(x))**2)
  slope = num/den
  return slope

print("ŋ =", LinearRegression(X,Y))
nu =  LinearRegression(X,Y)
```

→ Finally, we obtain: *η = -1.417e-09*

Question D : σ

- To find σ, we need to calculate our returns using this formula : $r_t = \frac{P_t - P_{t-1}}{P_{t-1}} * \frac{1}{\sqrt{timeStep}}$

```
dataVolume['return'] = pd.Series(index=dataVolume.index)
for i in range(1, len(dataVolume)):
  previous_price = dataVolume['Price'].iloc[i - 1]
  current_price = dataVolume['Price'].iloc[i]
  return_value = ((current_price - previous_price) / previous_price) * (1/((dataVolume['Date'].iloc[i] -dataVolume['Date'].iloc[i-1])**(1/2)))
  dataVolume['return'].iloc[i] = return_value
print(dataVolume)
```

- We estimate σ and the mean μ :

```
mean_return = sum(dataVolume['return'][1:]) / len(dataVolume['return']-1)
squaredSum = 0
for i in range(2, len(dataVolume['return'])):
    squaredSum += ( dataVolume['return'].iloc[i] - mean_return) ** 2

sigma =(squaredSum / (len( dataVolume['return']) - 1))**(1/2)
```

→ Finally, we obtain : *σ = 0.001991*

Question D : γ

The permanent impact formula is defined as : $g(v_k) = \gamma * v_k \; avec \; v_k = \frac{n_k}{\tau}$

- Let's start by calculating the permanent impact (*permanentimpact*) : $g(v_k) = P_{t+2} - P_t$

```
permanentimpact = []
t = []
for i in range(len(dataVolume['Volume']) - 2):
      TI = dataVolume['Price'].iloc[i +2] - dataVolume['Price'].iloc[i] #Pt+2 - Pt
      permanentimpact.append(TI)
      t.append(dataVolume['Date'].iloc[i + 2] - dataVolume['Date'].iloc[i]) #difference between two dates of Pt+2 - Pt

print(permanentimpact)
```

- We define X to then do the linear regression and find the γ : $X = \frac{n_k}{\tau}$

```
print("γ =", LinearRegression(X,permanentimpact))
gamma =  LinearRegression(X,permanentimpact)
```

→ Finally, we obtain: ***γ = 2.714e-07***

Question D : Strategy

Preamble : Now that we have estimated the parameters, we will calculate the optimal liquidation strategy following the Almgren and Chriss model.

Our utility function is given by : $U(\hat{\lambda}) = E(L) + \hat{\lambda}V(L)$

- We want to minimize this utility function and the solution to this optimization problem is :

$$\forall k \in [1; N], x_k = \frac{\sinh\left(M\left(T - \left(k - \frac{1}{2}\tau\right)\right)\right)}{\sinh(MT)}X$$

$$\text{où } K \underset{\tau \to 0}{\sim} \sqrt{\frac{\lambda * \sigma^2}{\eta}} + O(\tau)$$

With $M \underset{\tau \to 0}{\sim} \sqrt{\frac{\lambda * \sigma^2}{\eta}} + O(\tau)$

$T = 24$

$k = \frac{1}{24}$

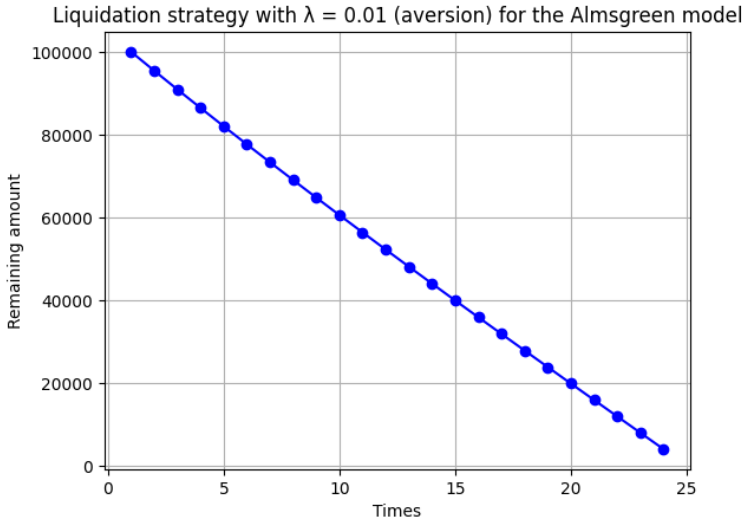
$\tau = \frac{1}{24}$

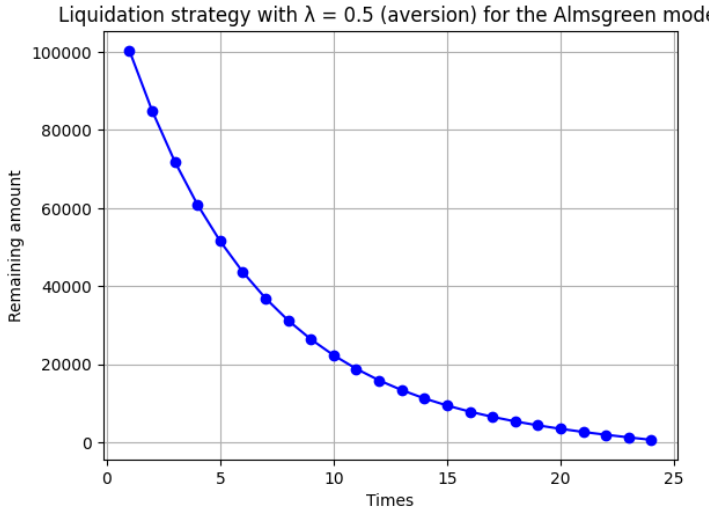
```
def calculate_xk(sigma, nu,lmbda):
    xk = np.zeros(24)
    M = np.sqrt(lmbda * (sigma**2) / abs(nu))
    if(lmbda==0):
      M = 0

    for i in range(24):
        xk[i] = (np.sinh(M*(24- (i - 1/2*1/24)))/np.sinh(M*24))*100000 # 100000 is the initial capital
    return xk

xk = calculate_xk(sigma, mean_return, 0.5) #we put mean_return instead of nu
```

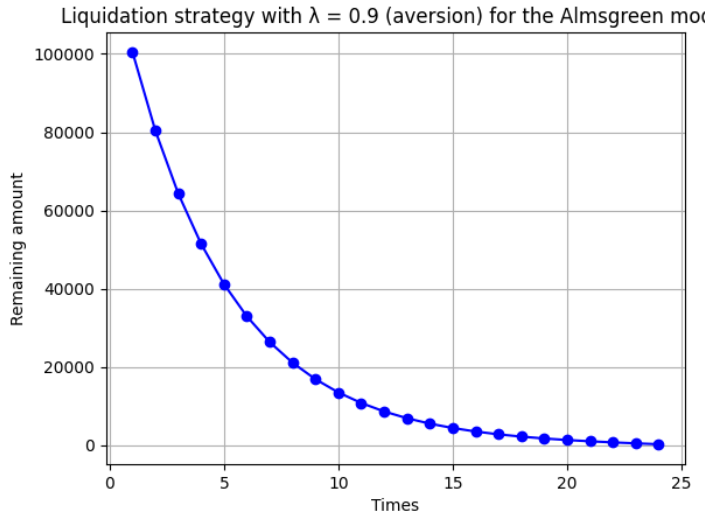
NB: We have incoherences when we put η in the formula for M, by putting *mean_return* we get better results.

Question D : Strategy

- Lambda corresponds to risk aversion; we vary it between [0;1] and display the result of the liquidation strategy :







These three graphs seem to us to be compatible with the formula in the sense that when the lambda is small, we give little importance to market risk and therefore our entire strategy will be focused on liquidity risk (the impact that we will have on the market). With a small lambda, we have a linear line, and we will liquidate uniformly. Conversely, when the lambda is high, we put a lot of weight on the variance (market risk), we are averse to the fluctuations that the asset may encounter. So, we liquidate a lot during the first slices.

Question E : FX rates

Preamble : We work with a dataset containing Forex rates for three currency pairs: GBPEUR, SEKEUR and CADEUR. The dataset gives us the HIGH and LOW prices every 15 minutes of these three currency pairs.

- We extract the dataset into three tables which each contain a currency pair :

```
GBPEUR = pd.read_excel('Dataset TD5.xlsx', header=2, usecols="A:C", names=["Date", "High", "Low"])
SEKEUR = pd.read_excel('Dataset TD5.xlsx', header=2, usecols="E:G", names=["Date", "High", "Low"])
CADEUR = pd.read_excel('Dataset TD5.xlsx', header=2, usecols="I:K", names=["Date", "High", "Low"])
```

- We calculate the mid prices for each data defined as: $P_{ti}^{Mid} = \frac{1}{2}(P_{ti}^{High} + P_{ti}^{Low})$

- Then, we calculate the returns (as for question A) and the geometric returns on the mid prices with the formula : $Rgeo_{ti}^{Mid} = \log(P_{ti}^{Mid}) - \log(P_{ti-\tau}^{Mid})$

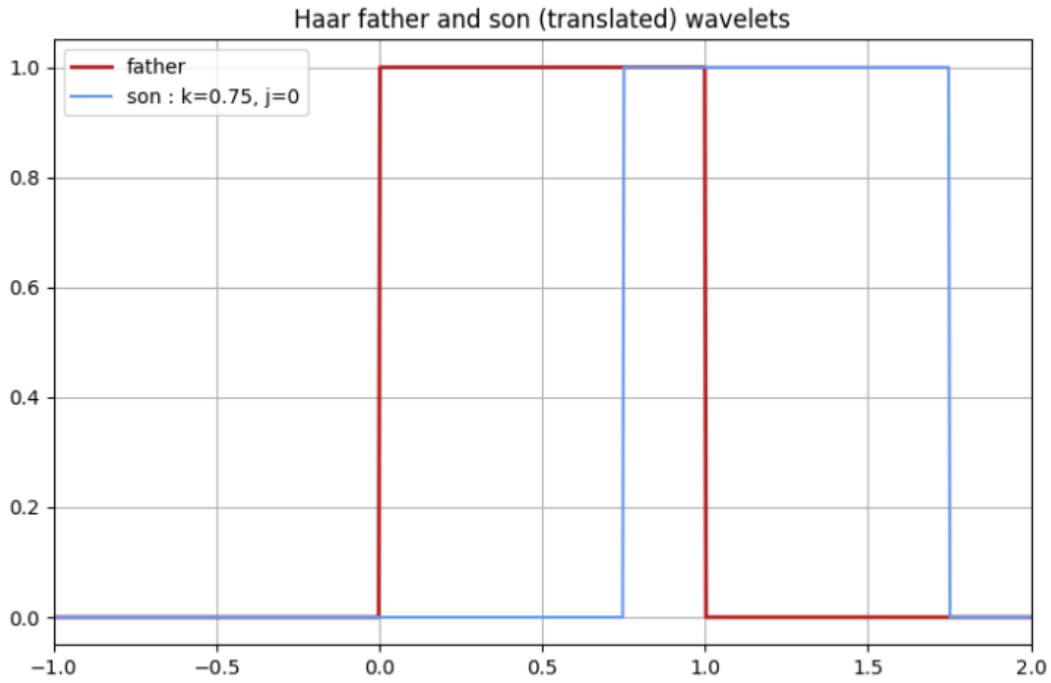
```
def mid_prices(list_data):
  for data in list_data :
    data['Mid'] = (data['High'] + data['Low'])/2
    returns = np.zeros(len(data['Mid']))
    log_returns = np.zeros(len(data['Mid']))
    for i in range(1, len(data['Mid'])):
      returns[i] = (data['Mid'].iloc[i] - data['Mid'].iloc[i-1]) / data['Mid'].iloc[i-1]
      log_returns[i] = np.log(data['Mid'].iloc[i]/data['Mid'].iloc[i-1])
    data['Returns'] = returns
    data['Log_returns'] = log_returns
```

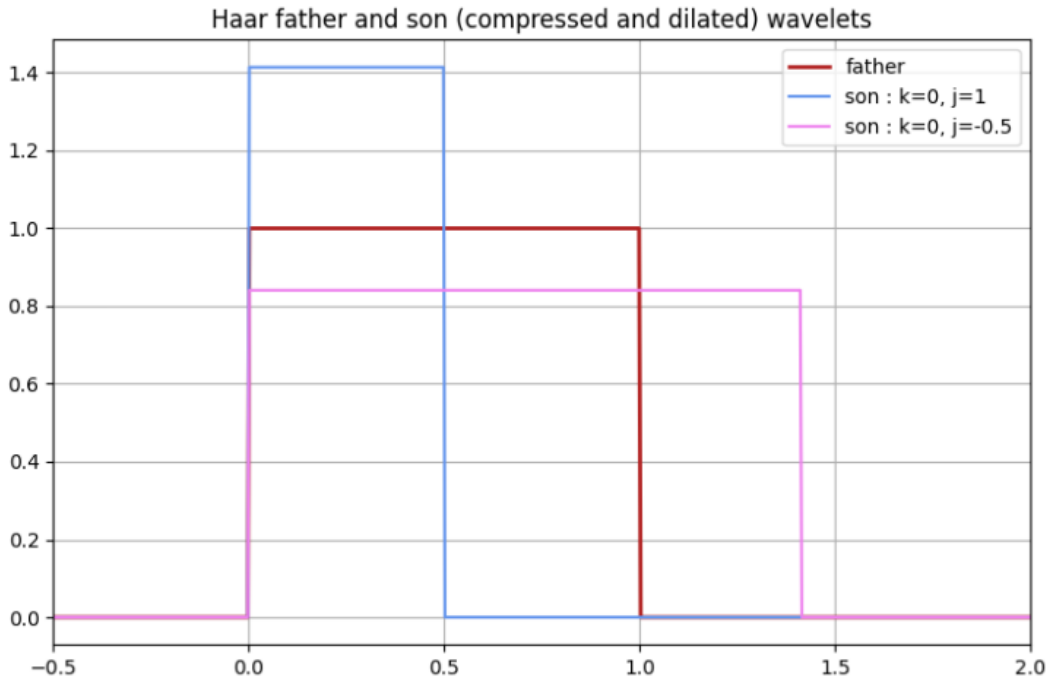
Question E : Haar wavelets

- We first define the Haar father and son wavelets as following :

Father wavelet : $\Phi(t) = 1_{[0;1[}(t)$

Son wavelet : $\phi_{j,k}(t) = 2^{j/2}\phi(2^j t - k)$





Question E : Scale coeff

- Now that we defined the basic wavelets, we build the scale coefficient : $c_{j_0,k} = \int z(t)\phi_{j_0,k}(t)$

```
def coefficient_c(signal, j, k):
    return sum(son_wavelet(i, k, j) * signal[::(2**j)][i]for i in range(int(len(signal)/(2**j))))
```

- Then we define the covariance (and correlation) at a given scale j : $Cov(j) = \frac{1}{T}\sum_{k=1}^{T}(c^1{}_{j,k} - \frac{1}{T}\sum_{l=1}^{T} c^1{}_{j,l})\,(c^2{}_{j,k} - \frac{1}{T}\sum_{l=1}^{T} c^2{}_{j,l})$

```
def cov(j, signal1, signal2):
  T = len(signal1)
  cov = 0
  division = int(T/(2**j))
  for i in range(division):
    m1 = sum(coefficient_c(signal1,j,k) for k in range(division))/division
    m2 = sum(coefficient_c(signal2,j,k) for k in range(division))/division
    cov += (coefficient_c(signal1,j,i) - m1) * (coefficient_c(signal2,j,i) - m2)
  return cov/division
```

Question E : Matrices

- Thanks to the covariance and correlation we can build the corresponding matrices at a given scale j for the 3 FX rates :

```
def cov_matrix(j, signaux):
  cov_matrix = np.zeros((3, 3))
  for i in range(3):
    for k in range(3):
      cov_matrix[i,k] = cov(j, signaux[i], signaux[k])
  return cov_matrix
```

```
def correl_matrix(j, signaux):
  correl_matrix = np.zeros((3, 3))
  for i in range(3):
    for k in range(3):
      correl_matrix[i,k] = correlation(j, signaux[i], signaux[k])
  return correl_matrix
```

NB: The scale factor j represents the dilatation : when j=0, there is therefore no dilatation, so we take every returns (one each 15min), when j=1 we take one return over 2 … for j=p we take one return over 2^p

Question E : Hurst exponent

- We want to calculate the Hurst exponent in order to go from a time scale to another :

Hurst exponent formula for second order moments : $$\widehat{H} = \frac{1}{2} log_2(\frac{M'_2}{M_2})$$

```
def hurst_exponent(signal):
  M1 = np.mean((abs(signal[1:]-signal[:-1]))**2)
  M2 = np.mean((abs(signal[2:]-signal[:-2]))**2)
  return round(0.5*math.log2(M2/M1),5)
```

GBPEUR : 0.67955
SEKEUR : 0.66729
CADEUR : 0.66774

As we defined it, we can now transpose the volatility vector of the portfolio from the smallest time internal (j=0) to another time scale by multiplying to 2^H*j

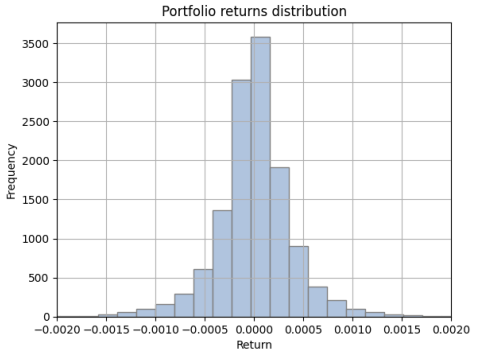
```
def transpose_vol(j, signal1, signal2, signal3):
  volatilities = (vol_vector(j, signal1, signal2, signal3))
  h1 = hurst_exponent(signal1)
  h2 = hurst_exponent(signal2)
  h3 = hurst_exponent(signal3)
  return [volatilities[1] * 2**(h1*j), volatilities[2] * 2**(h2*j), volatilities[3] * 2**(h3*j)]
```

Question E : Empirical volatility

- On this part we focus the study on the whole portfolio, we compute the price of the portfolio at each time and the returns :

```
portfolio.head()
```

	MidPrices	Returns
0	0.694408	0.000000
1	0.695075	0.000960
2	0.694842	-0.000336
3	0.694515	-0.000470
4	0.694523	0.000012



We can now calculate the empirical standard derivation (=volatility) of the portfolio at a given scale j :

```
def vol_j(j, returns):
  rdt = returns[::(2**j)].tolist()
  n = len(rdt)
  mean = sum(rdt)/len(rdt)
  std = (sum((rdt[i] - mean)**2 for i in range(n))/n) ** 0.5
  return std
```

Sources and acknowledgments

Most formulas and concepts come from :

- Market risk 2023-2024, by *Matthieu Garcin*

We would like to thank Ms.RIVOIRE Manon for her help in carrying out this project.