Student Name: Braun Arthur — Cabal Paul-Louis

ESILV ENGINEERING SCHOOL DE VINCI PARIS

Market Risk

Project

---

1. Question 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 (Epanechnikov Kernel).

   (a) First, let's set the data in the correct timeline and print the time series and the returns

```
# Data importation
data=pd.read_csv("Natixis_stock.txt",sep="\t",names=['date','value'])
data['date']=pd.to_datetime(data['date'],format="%d/%m/%Y")

#Data between early 2015 and the ned of 2016
temp=0
for i in range(len(data)):
    if data['date'].loc[i].date()>datetime.datetime(2016,12,31,0,0,0).date():
        temp=i
        break

data=data.drop([i for i in range(temp,len(data))])

data['value']=data['value'].str.replace(',','.').astype(float)

data['rendement']=[0 for i in range(data.shape[0])]

# Return computation
for i in range(1,data.shape[0]):
    data['rendement'].loc[i]=(data['value'].loc[i]-data['value'].loc[i-1])/data['value'].loc[i-1]
```

   (b) Now let's estimate the distribution with the Epanechnikov Kernel method, knowing that :

$$\hat{F}(x) = \int_{\infty}^{x} \hat{f}(x)\,dx \quad and \quad \hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K'(\frac{x - Xi}{h}) \tag{1}$$

   The Epanechnikov Kernel and it's antiderivative, $\forall x \in [-1, 1]$:

$$K'(x) = \frac{3}{4}(1 - x^2) \quad and \quad K(x) = \frac{3}{4}(x - \frac{x^3}{3}) \tag{2}$$

Thus, F(x) :

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} K(\frac{x - Xi}{h}) \tag{3}$$

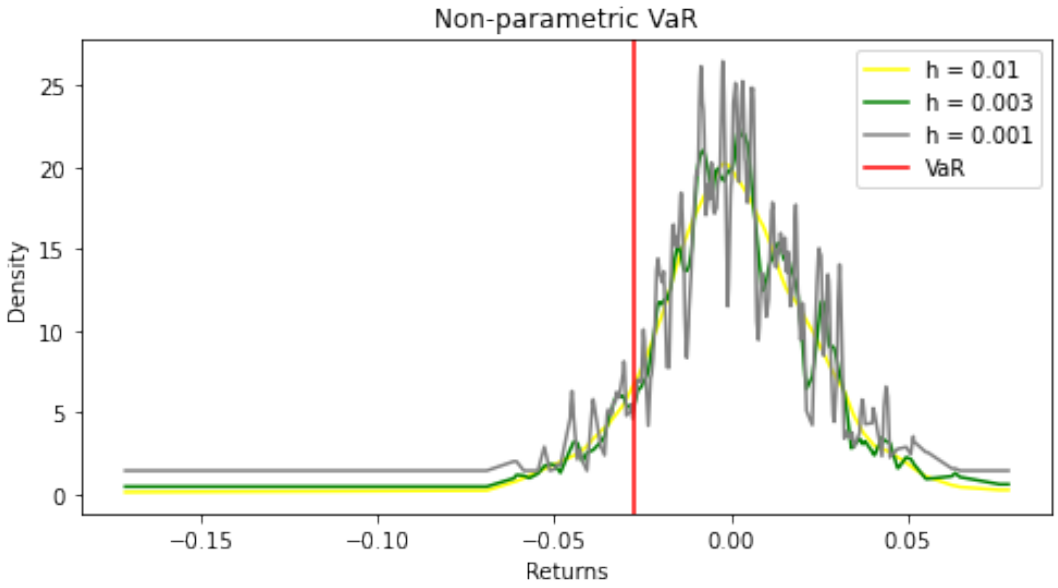
In Python, Edensity is f(x) and IEndensity is F(x) :

```
def Edensity (x) :
    Edensity = 0.0
    if -1 <= x <= 1 :
        Edensity = (3/4)*(1 - x**2)
    return Edensity

def IEdensity (x) :
    IEdensity=0.0
    if -1<= x <= 1 :
        IEdensity = (3/4)*(x - (x**3/3))
    return IEdensity
```

```
def Kernel_pdf(x,data,h): #Kernel density function based on a probability density
    cum_sum = 0.0
    n = len(data)

    for i in range(n):

        cum_sum += Edensity((x - data[i])/h)

    return (1.0 / (n * h)) *  cum_sum

def Kernel_cdf(x,h,data): #Kernel cumulative distribution function
    cum_sum = 0.0
    n = len(data)

    for i in range(n):
        y = (x - data[i])/h
        cum_sum += IEdensity(y)

    return (1.0 / n) *  cum_sum
```

(c) Now we chose between different smoothing parameter h

We obtain this graph by plotting the density function for different smoothing parameter : h1=0.01, h2=0.003, h3 = 0.001 :



In general, a larger smoothing parameter will give a smoother estimation of the distribution, while a smaller smoothing parameter will give a more accurate but noisier estimation. It is important to find a balance between precision and smoothing according to your modeling objectives.

Here we take h=0.01 because there is not much noise and it tend to an Epanechnikov density.

(d) Now we can estimate the non-parametric VaR
We know that :

$$F^{-1}\ (p) = inf\ \{\forall x \in R, F(x) \geq\ p\} \tag{4}$$

By taking the quantile at 1-p of the estimation of F(x) by the non-parametric Epachnenikov Kernel.

```
I = 1000 #Initial investment
p = 0.95 #Confidence level


non_param_VaR = np.quantile(fx1,1 - p)
print(f"The non-parametric VaR is equal to : {non_param_VaR}")
```

✓ 0.1s

```
The non-parametric VaR is equal to : -0.02725355857451945
```

Finally, If we invested I = 1000 € in the portfolio, we expect to lose less than 27.25€ at a 1 day horizon with a probability of 0.95

2. Question B

   (a) Let's calculate the expected shortfall for the VaR calculated in question A. How is the result, compared to the VaR ?
   The Expected shortfall with p confidence level, $\forall p \in [0, 1]$:

$$ES_p\ (x) = \frac{1}{1-p} \int_p^1 VaR_r\ dr \tag{5}$$

It can be translated as the mean of values taken below the Value at Risk

```
def expected_shortfall(data, var):
    # Identify the events beyond the VaR
    losses = []
    for x in data :
        if x < var :
            losses.append(x)



    # Calculate the average loss beyonf the VaR
    avg_loss = np.mean(losses)

    return avg_loss

# Calculate the expected shortfall
es = expected_shortfall(Oobs, non_param_VaR)
print("The Expected shortfall is : ",100*es,"%")
```

✓ 0.2s

```
The Expected shortfall is :  -4.270815918216243 %
```

Thus, the Expected shortfall with p=0.95 is : -0.043
The Expected shortfall is a coherent and spectral risk measure. It is the mean of values below the value at risk, so it's the average loss we can expect with confidence p. Then with the ES we tend to loose more rather than with the VaR

(b) Calculate the volatility, as well as the upper and lower semi-deviations for the Natixis stock for each of the four years in your dataset. What can you conclude about the riskiness of each of these years for this stock ?

We know that :

- volatility (standard deviation): $\sqrt{\mathbb{E}[(X - \mathbb{E}[X])^2]}$,
- lower and upper semi-deviation: $\sigma_-(X) = \sqrt{E[(X - EX)_-{}^2]}$ and $\sigma_+(X) = \sqrt{E[(X - EX)_+{}^2]}$, where $[X]_- = \max\{0, -X\}$ and $[X]_+ = \max\{0, X\}$.

	Volatility	Upper semi dev	Lower semi Dev
2015	0.317094	0.020067	-0.020067
2016	0.430961	0.026668	-0.026668
2017	0.252768	0.014983	-0.014983
2018	0.249340	0.017427	-0.017427

For each year, we can consider that Natixis' assets are not very volatile. Indeed, its volatility is between 24 % and 43 %. The level of risk is therefore quite low, the asset seems satble. We then calculated the semi-deviation which informs us about the dispersion of the Natixis asset. It differs from "simple" volatility, which gives the same importance to positive and negatives. Thus, for 2015, the potential losses are similar to the gain by semi-deviation. For the other years, the positive and negative semi-deviation differ slightly.
However, volatility does not take into account extreme events, or even semi-deviation. Volatility is therefore a very incomplete measure of risk. Consideration should be given to calculate the EVT VaR to complete our analysis.

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

   (a) Determine the extremal index using the block or run de-clustering, for the two tails of the distributions.

   The extremal_index function takes the return time series and the desired block size as input. It first calculates the mean and standard deviation of the return time series.

   Then, it divides the time series into blocks of bloc_size size using a comprehension list. Each block contains consecutive block_size returns of the time series.

   The function then initializes a counter of extremal blocks to 0.

   Then, it scans each block and checks if it contains an extremal return using the max function. If the block contains an extremal return, the counter is incremented by 1.

   Finally, the function calculates the extremal index by dividing the number of extremal blocks by the total number of blocks and returns the result.

Indeed, by choosing a block size b, we count the number of clusters for which at least one variable Xi is above a threshold u.
We have chosen the threshold u such that u represents the value for which 2.5% of the values of our series is strictly higher than u.

Thus, we have u = 0.04. We implemented the associated formula to block-declustering to find the optimal block parameter b to obtain a series as independent as possible
(close to 1).

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

```python
def extremal_index(data,u, block_size):
    # calculate mean and standard deviation of return time series
    mean = np.mean(data)
    std = np.std(data)

    # divide the time series into blocks of block_size size
    num_blocks = len(data) // block_size
    blocks = [data[i * block_size:(i + 1) * block_size] for i in range(num_blocks)]

    # initialize the counter of extremal blocks
    extreme_blocks = 0

    # go through each block and count the extremal blocks
    for block in blocks:
        if abs(np.max(block)) > u * std or abs(np.min(block)) > u * std:
            extreme_blocks += 1

    # compute the extremal index using the formula
    extremal_index = extreme_blocks / num_blocks

    return extremal_index

# Compute the extremal index for the upper tail of the return distribution
upper_tail_index = extremal_index(Oobs,0.04, block_size=6)

def Convert(lst): #
    return [ -i for i in lst ]

IOobs = Convert (Oobs)
 # compute the extremal index for the lower tail of the return distribution
 # Here IOobs = -Oobs
lower_tail_index = extremal_index(IOobs,0.04, block_size=6)

print(upper_tail_index)
```

Finally, we have : $\theta = 0.9765$.

(b) Propose an adaptation of the EVT VaR which takes into account the dependence of the returns.
The Extreme Value Theory (EVT VaR) Risk Value method is a method of calculating the Value at Risk (VaR) which uses the theory of extreme values to estimate the probability of an exceptional loss event.

The Pickands estimator ξ is a method used to estimate the dependence function between random variables. Define as :

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

And it's VaR associated :

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

Knowing that k(n) = log(n) is defined on the real positive and :

$$\lim_{n\to\infty} \frac{k(n)}{n} = 0 \quad , \quad p = F(VaR(p)) \quad , \quad G_\xi(x) = exp(-(1+\xi x)^{\frac{-1}{\xi}}) \tag{6}$$

Then we can deduce that :

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

In python :

```
def k(n) :
    return np.log(n)

def Est_pickands(data):
    n=len(data)
    res = np.log((data[n-int(np.log(n))] - data[n-2*int(np.log(n))]) / (data[n-2*int(np.log(n))] - data[n-4*int(np.log(n))]))
    return res / np.log(2)


def EVT_var_pickands(p, data) :
    n = len(data)
    e = Est_pickands(data)
    theta = extremal_index(data,0.04,6)
    numerator = ( (k(n) * theta) / (n * (1 - p)))**e - 1
    denominator = 1 - 2**(-e)
    return numerator/denominator * (data[n - int(theta * k(n))] - data[n - 2*int(theta * k(n))]) + data[n - int(theta * k(n))]
```

We obtain for different value of p :

```
The EVT-VaR with p = 0.9 is equal to :  0.025278044057997 24
The EVT-VaR with p = 0.95 is equal to :  0.03939383739883594
The EVT-VaR with p = 0.99 is equal to :  0.056783365565728224
```

Finally the Evt VaR is between the non-parametric VaR and the Expected shortfall

4. Question D

(a) Estimate all the parameters of the model of Almgren and Chriss. Is this model well specified?

The Almgren and Chriss model is a high-frequency trading model that determines the optimal amount of an asset to buy or sell based on its transaction cost and market volatility.
It is generally used to assess the transaction cost and risk associated with high-frequency trading strategies.

We know that the equation for the liquidation price of the model is the following :

$$\sum_{k=1}^{N} n_k \tilde{S}_k = XS_0 + \sum_{k=1}^{N} \left[\left(\sigma \sqrt{\tau} \varepsilon_k - \tau g\left(\frac{n_k}{\tau}\right) \right) x_k - n_k h\left(\frac{n_k}{\tau}\right) \right].$$

We want to estimate the variance σ^2 of our data assets at our disposal, as well estimate risk aversion λ and impact coefficient η.

In the following code, we define a cost function that measures the difference between the values predicted by the model and the values observed on the transaction data. We then use the minimize function of the scipy library to minimize the cost function using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. Finally, we display optimal lambda and variance values.

```
# Cost function
def cost_function(params, returns):
    lambda_, variance = params
    n = len(returns)
    cost = 0
    for i in range(n):
        cost += (lambda_ * returns[i] - variance / 2) ** 2
    return cost

# Estimation of the parameter model
params_init = [1, 1]
params_opt = minimize(cost_function, params_init, args=(Oobs,), method='BFGS')
lambda_opt, variance_opt = params_opt.x

print("Lambda optimal :", lambda_opt)
print("Variance optimal :", variance_opt)
```

✓ 0.8s

```
Lambda optimal : -5.808289130289158e-07
Variance optimal : 2.9972792613907576e-09
```

Now we have to estimate the impact coefficient η :

Estimating the impact coefficient seems more complex than the other parameters.
Indeed Almgren and Chriss submit the idea that it is estimated as for 1 % of the daily traded volume, we suffer an impact at the price equal to daily mean of the bid-ask value.
Thus we will estimate as is :

$$\eta = \frac{\xi}{0.01 * v * \lambda} \tag{7}$$

With : ξ the daily mean of the bid-ask value, v the daily traded volume, and λ the risk aversion
Right now, I didn't succeed to estimate η

(b) In the framework of Almgren and Chriss, what is your liquidation strategy (we recall that you can only make transactions once every hour).

The objective function of the Almgren and Chriss model is given by:

$J = V * (u - l) - C * V^2$

where:

J is the objective function
V is the execution rate
u is the market price at time t
l is the market price at time 0
C is the trading cost per share

To maximize the objective function, you need to find the values of V that maximize J. This can be done using an optimization algorithm, such as gradient descent or the Nelder-Mead method.

```python
# Define the trading cost per share and the total time horizon of the trade
trading_cost = 0.001
t = 1.0

# Calculate the initial market price and the final market price
l = returns[0]
u = returns[-1]

# Define the objective function
def objective(x):
    v = x[0]
    j = v * (u - l) - trading_cost * v**2
    return -j

# Initialize the execution rate
x0 = [1.0]

# Use the Nelder-Mead method to maximize the objective function
res = minimize(objective, x0, method='nelder-mead')

# Print the estimated parameters
print("Optimal execution rate:", res.x[0])
print("Optimal liquidation time:", t)
print("Optimal liquidation volume:", res.x[0] * t)
```

✓ 0.4s

```
Optimal execution rate: 21.315136718750047
Optimal liquidation time: 1.0
Optimal liquidation volume: 21.315136718750047
```

5. Question E

(a) With Haar wavelets and the dataset provided with this tutorial, 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?

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

In order to estimate the Hurst exponent of our series, we use the normalized range analysis. Considering a partial time series analysis, we have the formula :

$$\frac{R}{S} = C \times n^H \tag{8}$$

with :

- R : range of the time series
- S : the standard deviation
- C : a constant
- n : the length of the series
- H : the Hurst exponent.

That means that we also have the formula :

$$log(\frac{R}{S}) = log(C) + H \times log(n) \tag{9}$$

Thus, we can find an estimation of H by calculating $log(\frac{R}{S})$ of various partial datasets of our time series.

First we create a function split that will return a list of partial dataset of different lengths (from 0.2 to 0.8 of our original dataset length).

```
def split(data):
    splits=[]
    k=0
    for i in np.arange(0.2,0.8,0.02):
        splits.append(data.sample(frac=i, random_state=42))

    return splits
```

Then, there is the RS function, that returns the value of $log(\frac{R}{S})$ for every partial dataset.

```python
def RS(data):

    n=len(data)

    #Mean of the dataset
    mean=0
    for x in data:
        mean=mean + x
    mean=mean/n

    #Mean-adjusted data
    meanadjusted=data
    for x in meanadjusted:
        x=x-mean

    #Cumulative series
    cumulative=meanadjusted
    sum=0
    for x in cumulative:
        x=sum+x

    range=max(cumulative)-min(cumulative)


    std_dev = 0
    for x in data:
        std_dev=std_dev+(x-mean)**2
    std_dev=(std_dev/n)**0.5

    return np.log(range/std_dev)
```

Finally, to compute the estimation of the Hurst exponents, through the function HurstExponent, we do a linear regression on $log(\frac{R}{S})$ compared to $log(n)$ which gives us an estimation of C and an estimation of H, because H is the value of the slope. The function only returns H

```python
def HurstExponent(data):
    logRS=[]
    logn=[]
    for x in split(data):
        logRS.append(RS(x))
        logn.append(np.log(len(x)))


    param=np.polyfit(logn, logRS,1)
    return param[0]
```

Here are the results that we have for the linear regressions :

	Hurst Exponent
GBPEUR	0.368994
SEKEUR	0.303646
CADEUR	0.135847

Thus, for the GBPEUR series we have $H = 0.37$, for the SEKEUR series we have $H = 0.30$ and for the CADEUR series we have $H = 0.14$.

In order to determine the annualized volatility of the series, we use this formula :

$$V = \sigma \times \sqrt{252} \times H \tag{10}$$

with :

- V: annualized volatility
- σ : standard deviation
- H : Hurst exponent

We have the following results :

	Volatility
GBPEUR	0.365074
SEKEUR	0.157679
CADEUR	0.109175