UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

MASTER 2 MOSEF

Gestion des risques

Auteurs:
Mehdi FERHAT
Mathieu REINE
Yasmine OUYAHYA

Sous la direction de
M. Matthieu GARCIN

Résumé:

The aim of this project consists in quantifying model risk, in particular in the framework of market risk measures. You are going to implement simple market risk measures such as VaR or ES (expected shortfall), and to study different kinds of model risk.

Table des matières

1 Dataset description **1**

2 About various calculation methods for the VaR **1**

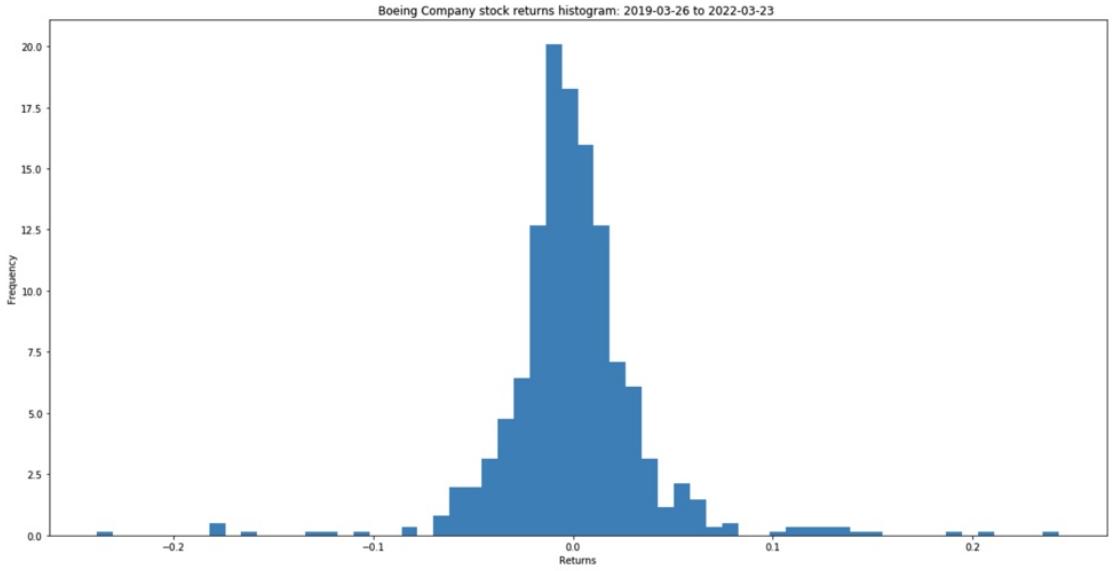
3 From VaR to ES **13**

4 Model risk as estimation uncertainty **15**

5 Model risk as specification uncertainty **19**

1 Dataset description

Les données collectées et utilisées pour cette étude proviennent de Yahoo! Finance et concernent le cours quotidien de l'action Boeing (The Boeing Company) sur 3 ans du 25-03-2019 ou 25-03-2022 (758 observations). Nous avons ensuite à partir de l'Adjusted Close calculé les rendements journaliers.



2 About various calculation methods for the VaR

In this section, you will calculate VaRs following various methods

Question 1. *Determine your VaR at 99% at a one-day horizon using alternatively the following methods:*

- *empirical quantile*
- *parametric distribution*
- *non parametric distribution*

La VaR est la perte maximale attendue pour un niveau de confiance donné, ici 99%.

On note pour les gains au niveau de confiance α : $VaR_\alpha(X) = -F_X^{-1}(1-\alpha)$. Nos rendements étant calculés à un jour la VaR correspond au quantile de niveau $\alpha\%$ de la distribution de pertes sur la série de rendement de l'action Boeing (BA). Ainsi la Value at

Risk obtenue serait telle que 99% des rendements (gains) de la série lui sont supérieurs (ou les pertes inférieures).

Les 3 méthodes de détermination de la VaR sont définies comme suit :

Premièrement la VaR-Historique, qui est la façon la plus simple de calculer la VaR. Cette méthode réorganise simplement les rendements historiques, en les classant de manière croissante. Elle part ensuite du principe que l'historique se répétera du point de vue du risque :

```
def VaR_Histo(returns, quantile):

    '''
    Cette fonction permet de calculer la VaR historique

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    quantile: float
        Le Quantile de la distribution

    Returns
    -------
    VaR : Float
        Renvoie la VaR estimée avec la méthode du quantile empirique
    '''
    returns_sorted = returns.sort_values()
    rank = quantile * (len(returns) + 1) - 1
    rank_l = int(rank)
    VaR = returns_sorted.iat[rank_l] + (returns_sorted.iat[rank_l + 1] -
    returns_sorted.iat[rank_l]) * (rank - rank_l)

    return VaR

var_histo = VaR_Histo(returns, 0.01)
var_histo*100

OUT : -11.13108803984891
```

La VaR paramétrique (variance-covariance), cette méthode part du principe que les gains et les pertes sont répartis de manière égale. De cette façon, les pertes potentielles peuvent être définies en termes d'écarts types par rapport à la moyenne. Les rendements estimés et l'écart-type peuvent ainsi être combinés pour créer une courbe de distribution normale.

```
def VaR_Parametric(returns, quantile):
    '''
```

```
    Cette fonction permet de calculer la VaR paramétrique

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    quantile: float
        Le Quantile de la distribution

    Returns
    -------
    VaR : Float
         Renvoie la VaR estimée avec la méthode de la distribution paramétrique

    '''
    returns_sorted = returns.dropna().sort_values()
    returns_mean = returns_sorted.mean()
    return_std = returns_sorted.std()
    z = norm.ppf(quantile)

    VaR = returns_mean + return_std * z

    return VaR

var_param = VaR_Parametric(returns, 0.01)
var_param*100

OUT : -8.456332115292815
```

Enfin la VaR non paramétrique (Monte-Carlo). Les simulations de Monte Carlo font référence à toute méthode qui génère des tirages de manière aléatoire. Comme la méthode paramétrique, la méthode de Monte Carlo doit d'abord calculer la moyenne et la covariance. Nous allons d'abord simuler les rendements avec la structure de la matrice de variance-covariance, moyenne et variance des rendements historiques, calculer et ordonner les rendements simulés et enfin calculer la VaR à partir de l'échantillon simulé pour le quantile associé :

```
def VaR_MC(X):
      '''
    Cette fonction permet de calculer la VaR non paramétrique

    Parameters
    ----------
    X :  Series
         La série des rendements returns

```

```
    Returns
    -------
    VaR : Float
        Renvoie la VaR estimée avec la méthode de la distribution non
paramétrique (Monte Carlo)

    '''
    X_sorted = X.sort_values()
    X_mean = X_sorted.mean()
    X_std = X_sorted.std()

    mc_X = np.random.normal(X_mean, X_std,10000)
    var_MC = np.percentile(mc_X,0.01)
    return var_MC

var_MC = VaR_MC(returns)
var_MC*100

OUT : -13.329919287571776
```

Question 2. *Using the estimator of Pickands, determine the parameter of the GEV function of losses. Comment its value.*

Afin de pouvoir estimer l'indice de queue de la loi GEV et de décrire les queues de distribution de nos rendements, on utilise l'**estimateur de Pickands**. Dans notre cas, il s'agit de la modélisation des queues de distributions des **pertes de nos rendements.**

La loi GEV s'écrit sous la forme suivante:

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

Cette de distribution s'apparente à des lois paramétriques en fonction des valeurs de ξ.

$$\text{si} \quad \xi > 0, \quad \text{la loi est de type Fréchet;}$$

$$\text{si} \quad \xi = 0, \quad \text{la loi est de type Gumbel;}$$

$$\text{si} \quad \xi < 0, \quad \text{la loi est de type Weibull;}$$

Estimateur de Pickands:

On commence d'abord par ordonner les rendements de notre actif et on utilise donc la formule suivante :

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

On choisit k une fonction de $\mathbb{N}$ dans $\mathbb{N}$ tel que :

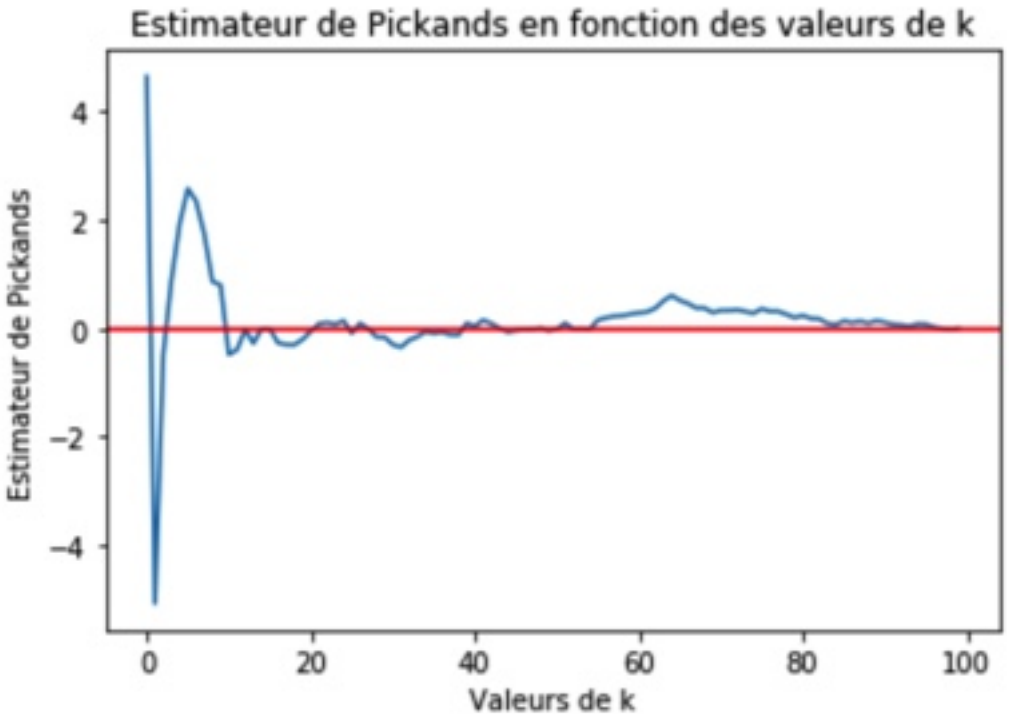
$$\lim_{n\to\infty} k(n) = \infty$$
$$\lim_{n\to\infty} \frac{k(n)}{n} = 0$$

L'estimateur de Pickands converge en probabilité vers ξ.
De plus, si

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

alors la convergence de $\xi^P_{k(n),n}$ vers ξ se fait presque sûrement et non pas seulement en probabilité.

```python
def Estimator_of_Pickands(X):

    '''
    Cette fonction permet de calculer l'estimateur de Pickands pour les pertes
des rendements

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    Returns
    -------
    estimator: Array
               Liste avec les valeurs de l'estimateur de Pickands pour les
pertes
    '''

    X = X.where(X.values<0).dropna().sort_values().reset_index(drop=True)
    n = len(X)

    a = np.array([X[i] for i in range(0,int(n/4))])
    b = np.array([X[i] for i in range(1,int(n/2),2)])
    c = np.array([X[i] for i in range(3,int(n),4)])

    estimator = (1 / np.log(2)) * np.log((a-b) / (b-c))

    return estimator

pickands_path = Estimator_of_Pickands(returns)
```



Dans notre cas, $\xi = 0$ donc il s'agit donc d'une **loi de Gumbel** qui est une loi à queue fine où il y a peu d'évènements extrêmes.

Question 3. *Determine the VaR at 99% at a one-day horizon using the EVT approach.*

L'approche de l'EVT est utilisée pour prendre en compte le problème des queues de distributions fines (ou queues de Poisson). Les queues de poisson contiennent des informations précieuses, notamment sur les rendements des actifs.

La VaR paramétrique par exemple, considère le rendement d'une action comme une variable aléatoire décrite par une distribution normale. Or, la distribution normale est une distribution centrée qui se caractérise par des queues fines aux extrémités, ce qui ne permet pas au modèle de considérer les cas exceptionnels mais souvent dramatiques avec une fréquence plus élevée. Pour palier à ce problème l'approche de l'EVT va considérer ces queues comme une distribution à part entière.

Ci-après la VaR de l'EVT en considérant l'estimateur de Pickands :

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

où ξ^pest l'estimateur de Pickands du paramètre de la GEV

```
def VAR_EVT(X,k,p):

    '''
    Cette fonction permet de calculer la VaR avec l'estimateur de Pickands
pour les pertes des rendements selon l'approche EVT.

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    k :  Integer
        Paramètre vérifiant les conditions ennoncées dans la question
précédente (estimateur de Pickands)

    p :  Float
         Seuil de confiance

    Returns
    -------
    var : Float
          VaR estimée avec l'approche EVT
    '''
    X = X.dropna().sort_values()
    ksi = Estimator_of_Pickands(X)[-1]
    n = len(X)-1
    var = ((np.power(k/(n*(1-p)) , ksi) - 1)/(1 - np.power(2 , -ksi))) *
    (X[n-k+1] - X[n-2*k+1]) + X[n-k+1]
    return var

var_evt = VAR_EVT(returns,1,0.01)
var_evt*100

OUT : -7.622046909976726
```

La VaR à 99% pour l'horizon un an en utilisant l'approche de l'EVT est donc ≈ -7.6.

Question 4. *What is Leadbetter's extremal index for the dataset? Comment its value and explain how you should change your VaR to take this into account (only provide an explanation, with equations, do not calculate a new VaR).*

L'inférence pour les clusters de valeurs extrêmes d'une série temporelle nécessite généralement l'identification de groupes indépendants de dépassements d'un seuil élevé.

L'indice extrémal de Leadbetter est un paramètre qui mesure le degré de clustering des valeurs extrêmes dans un processus stationnaire. Le paramètre θ est connu comme l'indice extrémal, et quantifie l'étendue de la dépendance extrémale.

- $\theta = 1$ signifie qu'il s'agit d'un processus complètement indépendant
- $\theta = 0$ signifie qu'on a des niveaux croissants de dépendance extême

On peut estimer le paramètre θ en utilisant deux méthodes : blocks et runs declustering.

Nous avons utilisé la méthode **Blocks declustering** pour ce faire. Cette méthode consiste à d'abord choisir la taille arbitrairement d'un bloc b, et partitionner la séquence $\{1, \ldots, n\}$ en $k = \lfloor n/b \rfloor$ blocs, $\{(i-1)b+1, \ldots, ib\}$ pour $1 \leq i \leq k$. Le schéma stipule que toutes les observations extrêmes situées dans le même bloc appartiennent au même cluster.

Soit

$$M_{i,j} = \max\{X_{i+1}, \ldots, X_j\}$$

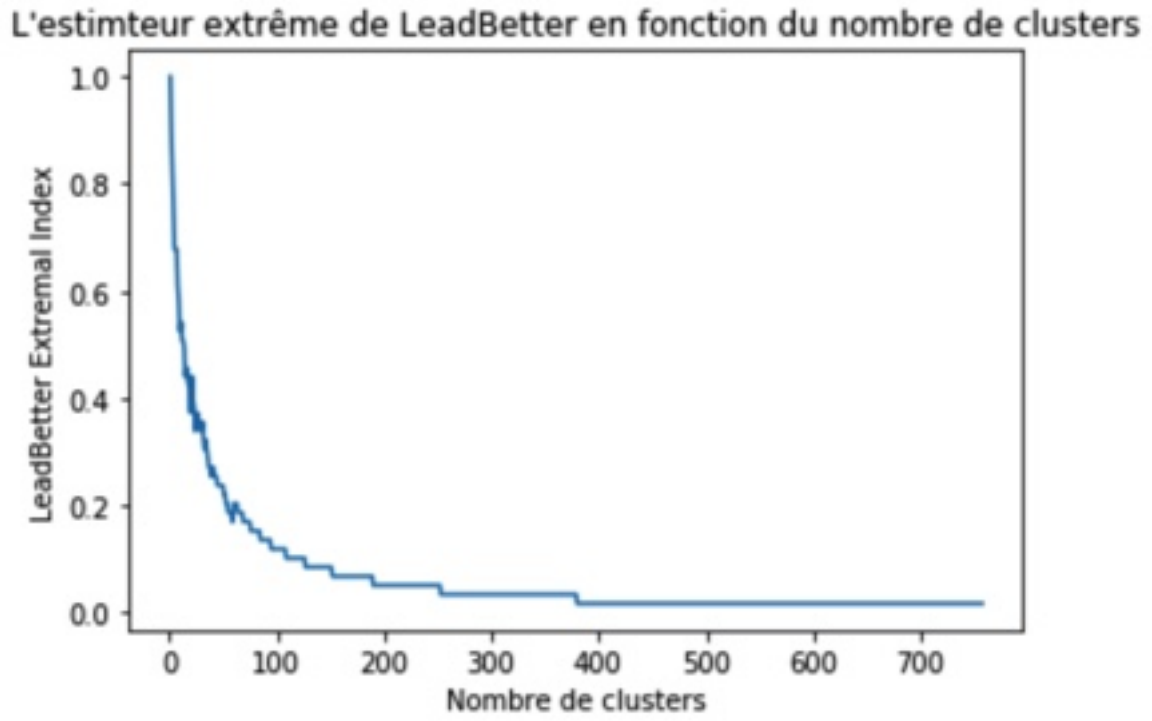
L'estimateur de l'indice extrémal qui correspond au schéma de déclustering par blocs est le suivant:

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

```python
def binarisation(X, threshold):
    """
    Cette fonction permet de binariser la série temporelle X en fonction du seuil.

    Parameters
    ----------
    X : Series
        La série de rendements returns

    threshold : int
                Le seuil définissant la condition de binarisation
    Returns
    -------
    binary_X : Array
               Matrice contenant des 0 là ou les rendements sont inférieurs au seuil et des 1 là où les rendements lui sont supérieurs
    """
    binary_X = np.where(X.values>threshold,1,0)
    return binary_X

```

```
def blocks_declustering(X,b,threshold):
    """
    Cette fonction permet d'estimer le Leadbetter extremal index par la méthode de blocks declustering.

    Parameters
    ----------
    X : Series
        La série de rendements returns

    b : int
        Le nombre de cluster à partir desquels seront estimé le Leabetter external index

    threshold : float
                Le seuil définissant les valeurs extrêmes des rendements
    Returns
    -------
        float
        Estimation du Leadbetter extremal index
    """
    n = len(X)
    k = int(n/b)
    binary_X = binarisation(X,threshold)
    sum_clusters = 0
    for i in range(1,k+1):
        if X[(i-1)*b:i*b].max()>threshold:
            sum_clusters+=1
    return sum_clusters/sum(binary_X)

threshold = returns.std()/0.99**(1/2)-returns.mean()
blocks_declustering(returns,480,threshold)

OUT : 0.01694915254237288 (LeadBetter Index)
```

On en conclut que la série est dépendante.



Question 5. *What is a Hurst exponent and how can one interpret its value?*

L'exposant de Hurst est une mesure de la tendance des séries temporelles à revenir ou à se regrouper vers un équilibre à long terme. Sa valeur est comprise entre [0,1].

- $H = 0$ implique une série de retour à la moyenne ; tout mouvement positif est immédiatement suivi d'un mouvement négatif et vice versa de sorte que la moyenne à long terme reste constante sur une période de temps.
- $H = 0.5$ représente un mouvement brownien, dans lequel il n'y a pas de corrélation entre la variable et sa valeur passée.
- $H = 1$ est une série à tendance ; les mouvements positifs ou négatifs élevés dans le présent sont suivis par des mouvements positifs ou négatifs élevés dans le futur respectivement.

Question 6. *Propose a risk measure taking into account the Hurst exponent and calculate its value for the price series used in this project. The Hurst exponent must be estimated with the absolutemoment estimator. This risk measure must take into account the autocovariance of a fBm (like in the Nuzman-Poor approach for predictions).*

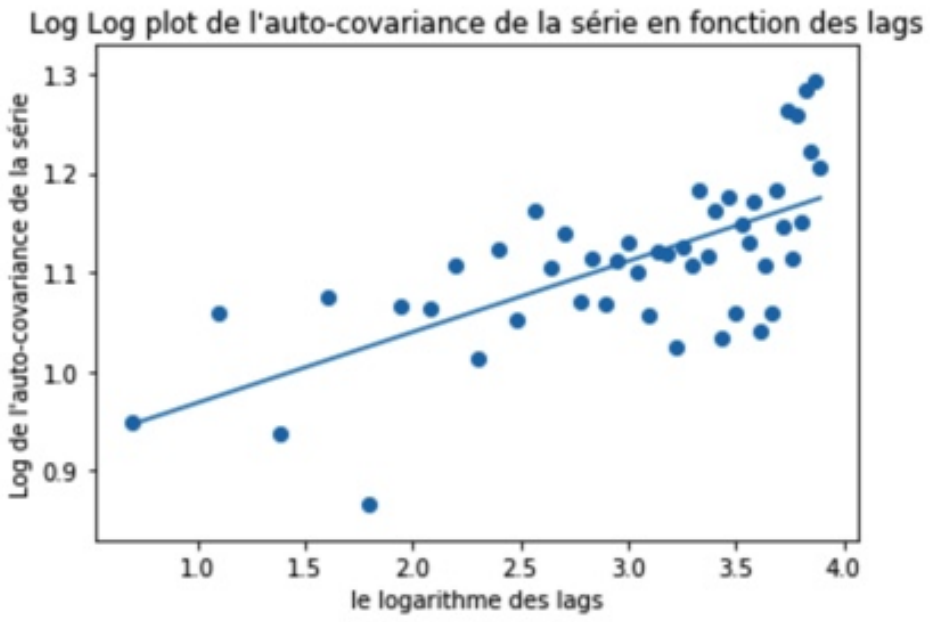
Si X est un fBm de paramètres H et σ^2, on peut utiliser la stationnarité de ses incréments pour estimer l'exposant de Hurst. En effet, la variance empirique des incréments d'une durée donnée $(t-s)$ converge vers $(t-s)^{2H}\sigma^2$. Ainsi, la pente de la courbe représentative du log de la durée par rapport au log de la variance empirique est égale à 2H.

$$M_{2,N,t_a,t_b}(S) = \frac{1}{N}\sum_{i=1}^{N}\left|S_{t_a+(t_b-t_a)\times i/N} - S_{t_a+(t_b-t_a)\times (i-1)/N}\right|^2 \text{[1]}$$

```python
def linear_regression(X, Y):
    """
    Cette fonction permet d'estimer les coefficients d'une régression linéaire univariée.

    Parameters
    ----------
    X : Array
        La variable explicative

    Y : Array
        La variable cible

    Returns
    -------
    B0 : int
         L'intercept de la régression
    B1 : int
         Le coefficient de la variable explicative
    """
    N = len(X)
    x_mean = np.mean(X)
    y_mean = np.mean(Y)

    B1_num = np.sum((X - x_mean) * (Y - y_mean))
    B1_den = np.sum((X - x_mean)**2)
    B1 = B1_num / B1_den
    B0 = y_mean - (B1*x_mean)

    return (B0, B1)

def hurst_exponent_estimator(X, max_lag=20):
    """
    Cette fonction renvoie l'exposant de Hurst de la série temporelle.

    Parameters
    ----------
    X : Series
        La série temporelle des rendements.
        Attention /!\ Cette série doit correspondre aux log rendements.
```

```
40
41      max_lag : int
42                  Le nombre de lag maximum sur lequel est estimé l'exposant de
    hurst
43
44      Returns
45      -------
46          float
47          Exposant de Hurst
48      """
49      """Returns the Hurst Exponent of the time series"""
50      lags = range(2, max_lag)
51      tau = [np.var(np.subtract(X[lag:], X[:-lag])) for lag in lags]
52      reg = linear_regression(np.log(lags), np.log(tau))
53      return reg[1]/2
54
55  log_returns = np.log(returns.dropna().values)
56  log_returns = log_returns[np.logical_not(np.logical_or(np.isnan(log_returns),
57  np.isinf(log_returns)))]
58  hurst_exponent_estimator(log_returns)
59
60  OUT : 0.03635049862235718
```

Ici l'exposant de Hurst est ≈ 0 on a donc une série de retour à la moyenne.



3 From VaR to ES

Question 7. *For each of the methods exposed above for calculating VaR, expose a method to extend them to ES. Calculate the 99% ES on the same sample.*

Supposons que X soit une variable aléatoire représentant la perte d'un portefeuille donné et que $\text{VaR}_\alpha(X)$ soit le VaR au niveau de confiance de $100(1-\alpha)$. On a l'$ES_\alpha(X)$ qui est défini par l'équation suivante.

$$ES_\alpha(X) = E\left[X \mid X \geq VaR_\alpha(X)\right].$$

L'Expected shortfall mesure combien on peut perdre en moyenne dans les états au-delà du niveau de la VaR. Lorsque la distribution des pertes n'est pas normale, la VaR ne tient pas compte de la perte au-delà du niveau de la VaR quand l'Expected shortfall au contraire la considère bien.

```
def ES(X,var):
    """
    Cette fonction renvoie l'Expected Shortfall de la série des rendements.

    Parameters
    ----------
    X : Series
        La série temporelle des rendements

    var : float
          La value at risk de la série des rendements

    Returns
    -------
    float
    L'Expected Shortfall de la série des rendements associée à la Value at
Risk.
    """

    X = X.dropna()
    return X[X.lt(var)].mean()

es_param = ES(returns,var_param)
es_histo = ES(returns,var_histo)
es_MC = ES(returns,var_MC)
es_evt = ES(returns,var_evt)

print(var_param*100,"\n",var_histo*100,"\n",var_MC*100,"\n",var_evt*100)
print(es_param*100,"\n",es_histo*100,"\n",es_MC*100,"\n",es_evt*100)

VaR Paramétrique : -8.456332115292815 - ES Paramétrique : -16.31218701636733
VaR Historique : -11.13108803984891 - ES Historique : -17.17476176544959
```

```
VaR Monte-Carlo : -12.466941047886317 - ES Monte-Carlo : -17.976518232901785
VaR EVT : -7.622046909976726 - ES EVT : -14.65751498842074
```

Question 8. *Backtest the ESs and the corresponding VaRs on your sample. Pay attention to the strict separation of estimation sample and test sample for this question (for the other questions, simply estimate the risk measures on the whole sample). Comment the result about the relative accuracy of ES and VaR.*

Nous avons donc procédé au backtesting de nos différentes mesures de risque. La procédure de backtesting sert à évaluer la qualité de la prévision d'un modèle de risque dans la mesure où elle compare les résultats réels à ceux générés par le modèle VaR. Nous allons donc scinder nos returns en une base de train sur laquelle seront estimées nos mesures de risque, et une base de test sur laquelle nous évaluerons leurs performances.

```
train = returns[:int(0.7*len(returns))]
test = returns[int(0.7*len(returns)):]

train_sorted = train.sort_values()
train_mean = train_sorted.mean()
train_std = train_sorted.std()

var_hist_train = VaR_Histo(train,0.01)
var_param_train = VaR_Parametric(train,0.01)
var_MC_train = VaR_MC(train)
var_EVT_train = VAR_EVT(train,1,0.01)

test_var_hist = test[test.lt(var_hist_train)]
test_var_param = test[test.lt(var_param_train)]
test_var_MC = test[test.lt(var_MC_train)]
test_var_EVT = test[test.lt(var_MC_train)]

print("Pourcentage de valeurs de test en-deçà de la VaR historique
:",test_var_hist.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de la VaR paramétrique
:",test_var_param.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de la VaR de mon Monte-Carlo
:",test_var_MC.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de la VaR EVT
:",test_var_EVT.count()/len(test),"\n")

es_param_train = ES(train,var_param_train)
es_hist_train = ES(train,var_hist_train)
es_MC_train = ES(train,var_MC_train)
es_EVT_train = ES(train,var_EVT_train)

test_es_hist = test[test.lt(es_hist_train)]
test_es_param = test[test.lt(es_param_train)]
```

```
test_es_MC = test[test.lt(es_MC_train)]
test_es_EVT = test[test.lt(es_EVT_train)]

print("Pourcentage de valeurs de test en-deçà de l'ES historique
:",test_es_hist.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de l'ES paramétrique
:",test_es_param.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de l'ES de Monte-Carlo
:",test_es_MC.count()/len(test))
print("Pourcentage de valeurs de test en-deçà de l'ES EVT
:",test_es_EVT.count()/len(test))

Pourcentage de valeurs de test en-deçà de la VaR historique : 0.0
Pourcentage de valeurs de test en-deçà de la VaR paramétrique : 0.0
Pourcentage de valeurs de test en-deçà de la VaR de mon Monte-Carlo : 0.0
Pourcentage de valeurs de test en-deçà de la VaR EVT : 0.0

Pourcentage de valeurs de test en-deçà de l'ES historique : 0.0
Pourcentage de valeurs de test en-deçà de l'ES paramétrique : 0.0
Pourcentage de valeurs de test en-deçà de l'ES de Monte-Carlo : 0.0
Pourcentage de valeurs de test en-deçà de l'ES EVT : 0.0
```

4 Model risk as estimation uncertainty

Question 9. *What is model risk and what are the different natures of model risk?*

Le risque de modèle est un type de risque qui se produit lorsqu'un modèle financier est utilisé pour mesurer des informations telles que les risques de marché ou les transactions d'une banque, et que le modèle échoue ou fonctionne de manière inadéquate et entraîne des résultats défavorables.

Le risque de modèle représente quelles pertes peuvent occasionner l'utilisation de modèles de finance . Ce risque peut provenir de différentes sources:

- D'abord, **la mauvaise spécification du modèle**. Ceci signifie le fait d'utiliser un modèle qui représente mal la réalité, ce qui peut entraîner un risque de modèle car les estimations seront forcément faussées. Les modèles qui sont mal spécifiés peuvent avoir des coefficients et des termes d'erreur biaisés. Par exemple, supposer un mouvement brownien au lieu d'un mouvement brownien fractionnaire.

- Dans un second temps, même si un modèle est bien spécifié il peut y avoir des **incertitudes dans l'estimation**. Ainsi, les paramètres peuvent être mal estimés si les données comportent beaucoup de bruit ou si on ne dispose pas d'un nombre important de données.

- Enfin, le risque de modèle peut provenir de **l'incertitude des résultats**. Il est, en effet, primordial de pouvoir créer des indicateurs sur les résultats comme les inter-

valles de confiances. Il y'a aussi une **incertitude liée aux méthodes numériques**. Par exemple, pour une simulation de Monte-Carlo, il faut penser à bien préciser la moyenne mais aussi l'écart-type.

Question 10. *For the parametric VaR, determine the distribution of your parameters and determine, either theoretically or by simulations, the corresponding distribution of VaR (each parameter is a random variable; if we draw a vector of parameters, we have one value for the VaR which is different from the VaR obtained for another drawn vector of parameters).*

On se propose de déterminer la distribution de la Value at Risk paramétrique. Pour ce faire, nous allons la calculer en se basant sur des échantillons simulés aléatoirement à partir de nos returns.

```
sample_means = []
sample_stds = []
sample_vars = []
for i in range(20000):
    random_returns = random.sample(returns.dropna().to_list(),50)
    random_returns.sort()
    sample_mean = np.mean(random_returns)
    sample_std = np.std(random_returns)
    sample_var = sample_mean + sample_std * norm.ppf(0.01)
    sample_means.append(sample_mean)
    sample_stds.append(sample_std)
    sample_vars.append(sample_var)ns)
```

Ci-dessous les distributions correspondantes :

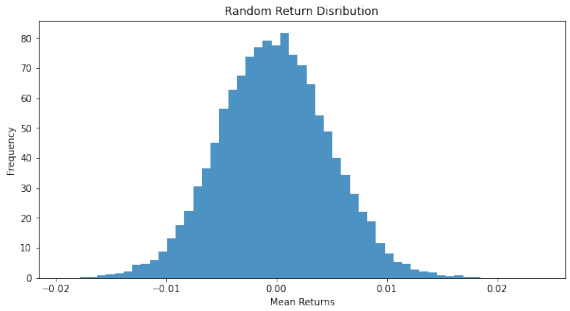


Figure 1:
Distribution de la moyenne des rendements

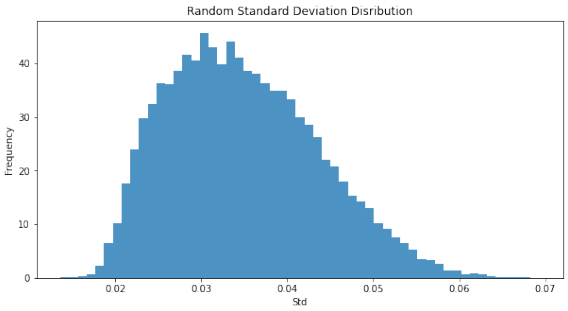
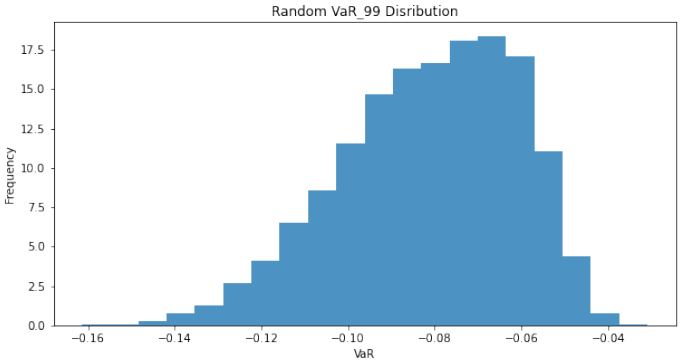


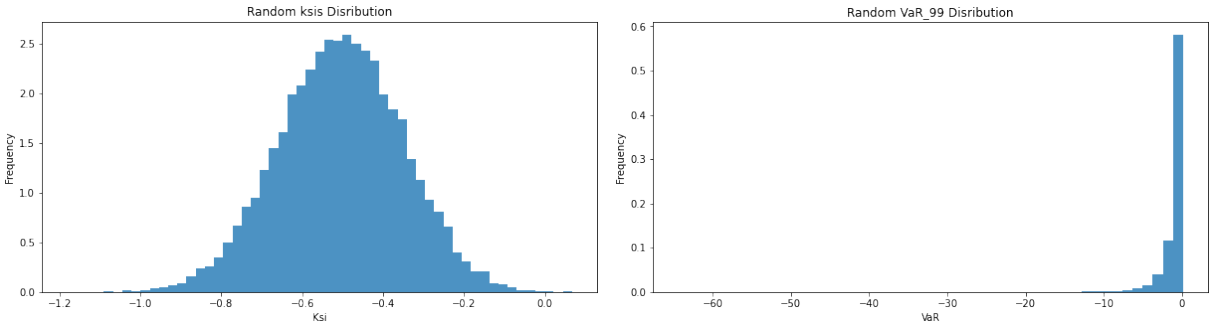
Figure 2:
Distribution de l'écart-type des rendements



Question 11. *If we suppose that price returns are iid Gaussian variables, determine, with the help of simulations, the distribution of the EVT parameters as well as the corresponding distribution of VaR.*

```
sample_means = []
sample_stds = []
sample_ksis = []
sample_vars = []
for i in range(20000):
    random_returns =
    np.random.normal(returns.dropna().mean(),returns.dropna().std(),1000)
    random_returns.sort()
    sample_mean = np.mean(random_returns)
    sample_std = np.std(random_returns)
    random_returns = pd.Series(random_returns)
    sample_ksi = Estimator_of_Pickands(random_returns)[-1]
    sample_var = VAR_EVT(random_returns,1,0.01)
    sample_means.append(sample_mean)
    sample_stds.append(sample_std)
    sample_ksis.append(sample_ksi)
    sample_vars.append(sample_var)
```

Les distributions des paramètres de l'EVT ont donc été dessinées :



Question 12. *Apply the same method than the one exposed in question 11 to determine the distribution of VaR in the fBm case.*

Question 13. *Represent in a graph the nonparametric VaR as a function of the bandwidth parameter of the kernel. Explain the method used to get this result.*

Le KDE est une méthode d'estimation de la densité de probabilité d'une variable aléatoire. Elle est obtenue en pondérant les distances entre les observations et un point quelconque à l'aide d'un kernel. Le résultat final est la somme des contributions de chacune des observations. Ainsi, plus on a d'observations dans une zone donnée, plus la probabilité d'observer un point dans cette zone est élevée.

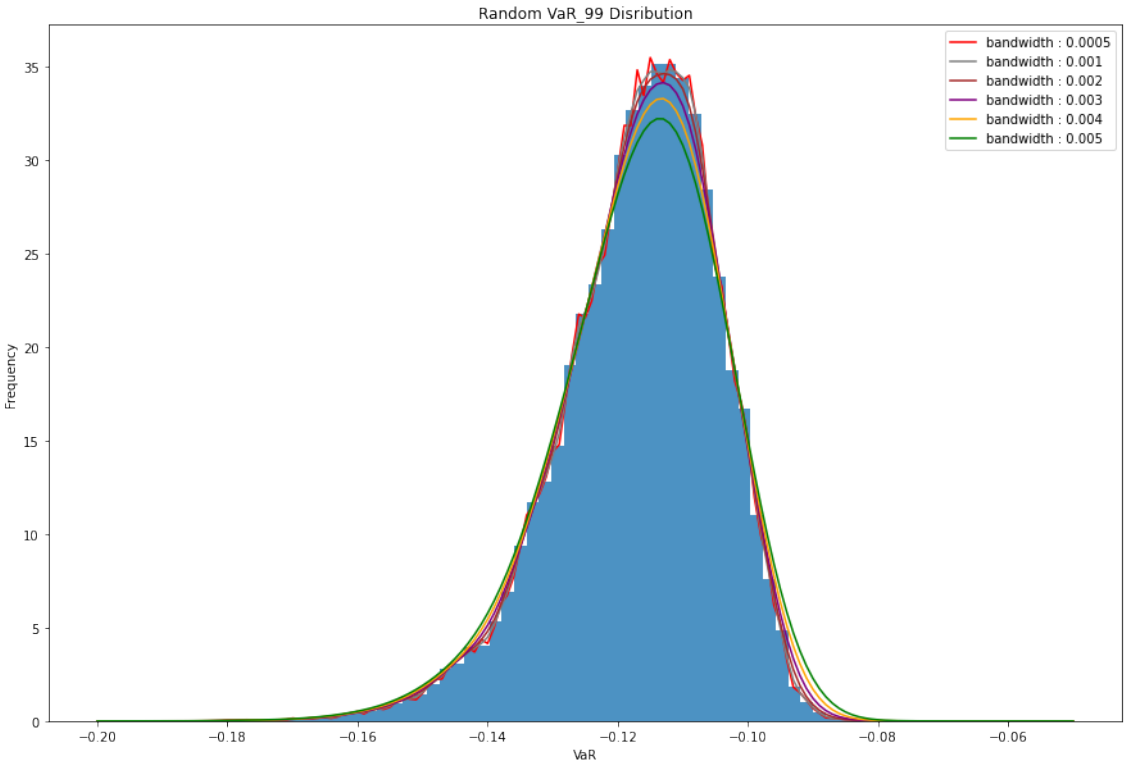
La pondération appliquée aux points de l'espace considéré est pilotée par la bandwidth. Plus celle-ci est faible, plus la contribution des observations éloignées du point considéré est faible. Dans ce cas, la courbe représentative de l'estimation de la distribution des observations présente de nombreuses ondulations. Cette pondération de=épend également du type de kernel appliqué (Triangulaire, Uniforme, Gaussien, etc.). Dans notre cas, nous utiliserons un kernel Gaussien.

L'estimation de la densité de probabilité de notre variable aléatoire peut s'exprimer mathématiquement comme suit :

$$\hat{f}(x) = \sum_{\text{observations}} K\left(\frac{x - \text{observation}}{\text{bandwidth}}\right)$$

```
sample_means = []
sample_stds = []
sample_vars = []
for i in range(20000):
    random_returns =
    np.random.normal(returns.dropna().mean(),returns.dropna().std(),1000)
    random_returns.sort()
    sample_mean = np.mean(random_returns)
    sample_std = np.std(random_returns)
    sample_var = np.percentile(random_returns,0.01)
    sample_means.append(sample_mean)
    sample_stds.append(sample_std)
    sample_vars.append(sample_var)

# Cette lambda expression renvoie la densité de probabilité d'une loi nourmale
d'espérance nulle et de variance égale à 1 évaluée en x.
# Il s'agit d'un kernel Gaussien.

kgauss = lambda x : 1/np.sqrt(2*np.pi) * np.exp(-1/2 * x**2)

# Cette lambda expression renvoie l'estimation de la densité de probabilité
des observations évaluée en x avec une bandwidth h et un kernel K.
```

```
20
21 D = lambda x, h, xi, K : 1/(len(xi) * h) * sum(K((x - xi) / h))
22
23
24 bandwidths = [0.0005, 0.001, 0.002, 0.003, 0.004, 0.005]
25 definition = np.arange(-0.2,-0.05,0.001)
```



5 Model risk as specification uncertainty

Question 14. *Using the VaRs and ESs implemented in the first two sections, determine the diameter for VaRs as well as the diameter for ESs. Comment the result with respect to model risk: is it more relevant to use ES or VaR?*

```
1 def VaRs_diameter(X, p):
2     """
3     Cette fonction renvoie le diamètre de la Value at Risk
4
5     Parameters
6     ----------
```

```
    X : Array
        La série temporelle des rendements
    p : float
        Le seuil de confiance

    Returns
    -------
        float
        Le diamètre de la Value at Risk
    """
    var_hist = VaR_Histo(X,p)
    var_para = VaR_Parametric(X,p)
    var_MC = VaR_MC(X)
    var_EVT = VAR_EVT(X,1,p)
    VaRs = [var_hist,var_para,var_MC,var_EVT]
    var_diameter =  np.max(VaRs)-np.min(VaRs)
    return var_diameter

def ESs_diameter(X, p):
    '''
    Cette fonction retourne le diamètre des ESs

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    p :  Float
         Seuil de confiance
    '''
    es_hist = ES(X,VaR_Histo(X,p))
    es_para = ES(X,VaR_Parametric(X,p))
    es_MC = ES(X,VaR_MC(X))
    es_EVT = ES(X,VAR_EVT(X,1,p))
    ESs = [es_hist,es_para,es_MC,es_EVT]
    es_diameter =  np.max(ESs)-np.min(ESs)
    return es_diameter

VaRs_diameter(returns,0.01)
ESs_diameter(returns,0.01)

OUT : 0.05625023898213771 (VaRS Diameter)
OUT : 0.04233714450823145 (ESs Diameter)
```

Le diamètre de l'Expected Shortfall est plus resseré que celui de la Value at Risk. Ainsi, l'incertitude sur l'estimation de l'Expected Shortfall est inférieure à celle de la Value at Risk. On privilégira donc l'Espected Shortfall pour l'évaluation des risques au niveau de confiance de 99

Question 15. *Is your conclusion at Question 14 the same if you change the confidence*

level from 99% to 90%, 95%, 98%, 99.5%, 99.9%, and 99.99%?

On réitère le procédé ci-dessus afin d'évaluer la mesure de risque à privilégier aux niveaux de confiance de 90%, 95%, 98%, 99.5%, 99.9%, et 99.99%.

```
def safest_risk_measure(X,p):
    '''
    Cette fonction retourne la mesure du risque avec le diamètre le plus
    faible

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    p :  Float
         Seuil de confiance
    '''
    var_diameter = VaRs_diameter(X,p)
    es_diameter = ESs_diameter(X,p)
    if var_diameter < es_diameter:
        safest_measure = {"risk_measure":"Value at Risk",
        "diameter":var_diameter}
    elif var_diameter > es_diameter:
        safest_measure = {"risk_measure":"Expected Shortfall",
        "diameter":es_diameter}
    else:
        safest_measure = {"risk_measure":"both","diameter":var_diameter}
    return safest_measure

list_conf = [0.1,0.05,0.02,0.01,0.005,0.001,0.0001]
for p in list_conf:
    safest_measure = safest_risk_measure(returns,p)
    print("confidence level:", 1-p,safest_measure)

confidence level: 0.9 {'risk_measure': 'Value at Risk', 'diameter':
0.10068881142423494}
confidence level: 0.95 {'risk_measure': 'Value at Risk', 'diameter':
0.08012219860570668}
confidence level: 0.98 {'risk_measure': 'Expected Shortfall', 'diameter':
0.05192367378848696}
confidence level: 0.99 {'risk_measure': 'Expected Shortfall', 'diameter':
0.033190032444810474}
confidence level: 0.995 {'risk_measure': 'Expected Shortfall', 'diameter':
0.053784445095056505}
confidence level: 0.999 {'risk_measure': 'both', 'diameter':
0.1755707885030155}
confidence level: 0.9999 {'risk_measure': 'both', 'diameter':
0.21444945852026637}
```

Comme à la question précédente on observe qu'à mesure que l'intervalle de confiance augmente, l'Expected Shortfall semble être la mesure du risque la plus intéressante.

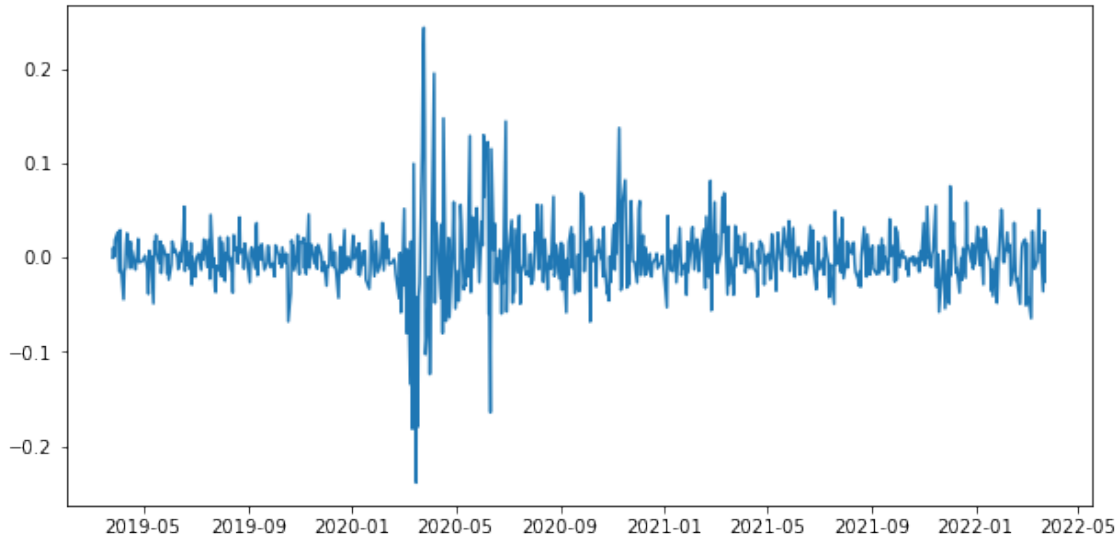
Question 16. *Add a noise process (say a Gaussian white noise) to the price return process and calculate the average impact on the VaR for each model. Which VaR method is the most robust? Display your results for various amplitudes of noise.*

Nous allons maintenant nous intéresser à l'impact de l'ajout d'un bruit blanc sur nos mesures de risques afin de déterminer laquelle est la plus robuste.

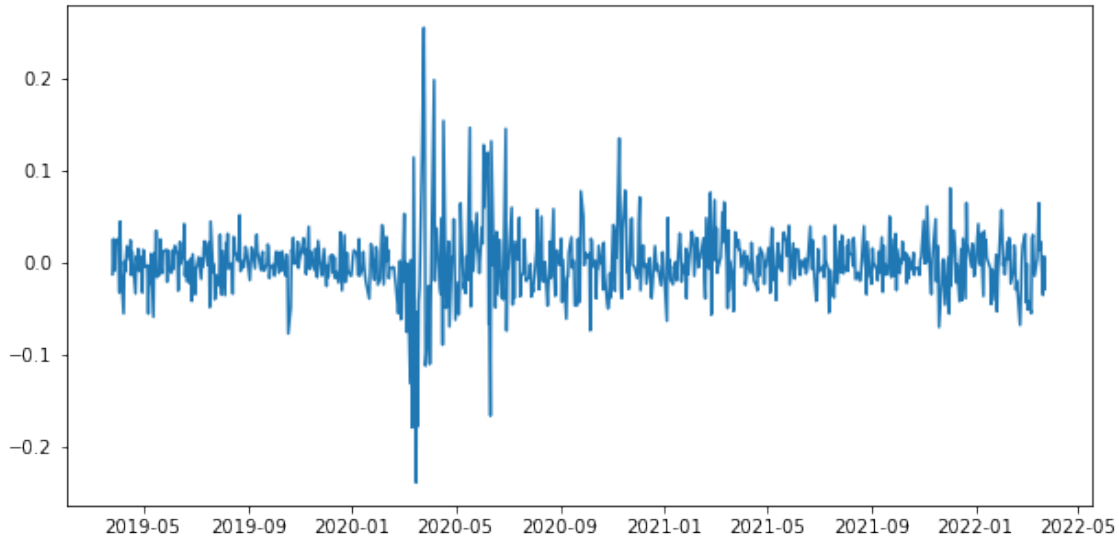
```
def compute_VaRs(X,p):
    '''
    Cette fonction permet de calculer les différentes méthodes de VaR à la
    série d'entrée

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    p :  float
         Seuil de confiance
    '''
    var_hist = VaR_Histo(X,p)
    var_para = VaR_Parametric(X,p)
    var_MC = VaR_MC(X)
    var_EVT = VAR_EVT(X,1,p)
    VaRs = {"Hist":var_hist,"Para":var_para,"MC":var_MC,"EVT":var_EVT}
    return VaRs

def compute_ESs(X,p):
    '''
    Cette fonction permet de calculer l'extension des différentes méthodes
    de VaR à l'ES sur la série

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    p : float
         Seuil de confiance
    '''

    es_hist = ES(X,VaR_Histo(X,p))
    es_para = ES(X,VaR_Parametric(X,p))
    es_MC = ES(X,VaR_MC(X))
    es_EVT = ES(X,VAR_EVT(X,1,p))
```

```
    ESs = {"Hist":es_hist,"Para":es_para,"MC":es_MC,"EVT":es_EVT}
    return np.array(ESs)

def white_noise_imputation(X,std):
    '''
    Cette fonction permet d'ajouter un bruit blanc à la série d'entrée

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    std: float
         Standard Deviation - Ecart type
    '''
    X = X.dropna()
    np.random.seed(30)
    white_noise = np.random.normal(0,std,len(X))
    X += white_noise
    return X

returns_and_wn = white_noise_imputation(returns,0.01)
```

Série



Série + bruit



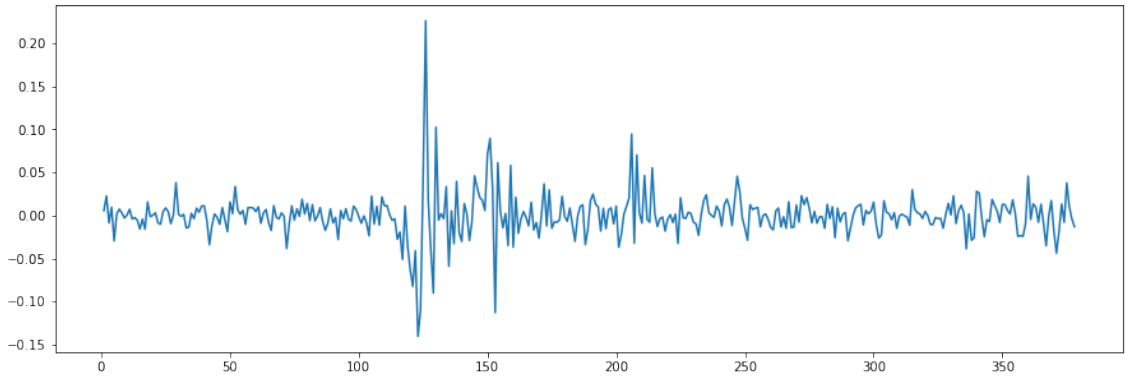

```python
def most_robust_with_wn_var(X,p):
    '''
    Cette fonction renvoie la méthode de calcul de VaR la plus robuste

    Parameters
    ----------
    X :  Series
         La série des rendements returns

    p: float
       Le Quantile de la distribution
    '''

    diff = {"Hist" : [],"Para" : [],"MC" : [],"EVT" : []}
    for std in np.arange(0,0.1,0.005):
        X_and_wn = white_noise_imputation(X,std)
        VaRs = compute_VaRs(X,p)
        VaRs_wn = compute_VaRs(X_and_wn,p)
        for key in diff.keys():
            diff[key].append(abs(VaRs[key]-VaRs_wn[key]))
    average_impacts = {"Hist" : 0,"Para" : 0,"MC" : 0,"EVT" : 0}
    for key in diff.keys():
        average_impacts[key] = np.mean(diff[key])
    method = min(average_impacts, key=average_impacts.get)
    average_impact = min(average_impacts.values())
    most_robust_var = {"method":method, "average_impact":average_impact}
    return most_robust_var
print(most_robust_with_wn_var(returns,0.01))

OUT : {'method': 'Hist', 'average_impact': 0.03929362503033088}
```

La Value at Risk la plus robuste à l'ajout d'un bruit blanc est la VaR historique. Il s'agit de la méthode enregistrant l'impact moyen le plus faible au rajout de simulations d'amplitudes variées.

Question 17. *Remove the noise of the price return process using the projection of your signal at one scale. Do this with scaling functions (also called father wavelet).*

Nous utilisons la fonction père de l'ondelette de Haar (ou scaling function) pour enlever le bruit :

```
def haar_denoising(X):
    '''
    Cette fonction permet de débruiter une série à l'aide de la méthode
    de Haar

    Parameters
    ----------
    X :  Series
         La série des rendements returns bruités
    '''
    def lowFreq(n , a):
        return (a[n] + a[n-1]) / 2

    if len(X) % 2 :
        X = pd.concat([X,pd.Series([0])])

    father = np.array([])
    for n in range(1 , len(X),2):
        father = np.append(father,[lowFreq(n , X)])

    return pd.Series(father,index=np.arange(0,len(father),1))
returns_denoised = haar_denoising(returns)
print(returns_denoised)
returns_denoised[2]
```



Série dénoisée

Question 18. *How do your VaR measures vary if they are applied to the denoised series? Display your results for various projection scales. Compare qualitatively your results with the ones of Question 16*

```
def most_robust_without_wn_var(X,p):

    diff = {"Hist" : [],"Para" : [],"MC" : [],"EVT" : []}
    X_no_wn = haar_denoising(X)
    VaRs = compute_VaRs(X,p)
    VaRs_no_wn = compute_VaRs(X_no_wn,p)
    for key in diff.keys():
        diff[key].append(abs(VaRs[key]-VaRs_no_wn[key]))
    average_impacts = {"Hist" : 0,"Para" : 0,"MC" : 0,"EVT" : 0}
    for key in diff.keys():
        average_impacts[key] = np.mean(diff[key])
    method = min(average_impacts, key=average_impacts.get)
    average_impact = min(average_impacts.values())
    most_robust_var = {"method":method, "average_impact":average_impact}
    return most_robust_var

print(most_robust_without_wn_var(returns,0.01))

OUT : {'method': 'Hist', 'average_impact': 0.017089114849668977}
```