NTNU
Norwegian University of
Science and Technology

The use of extreme value statistics for calculating risk measures in finance

Thibaut Malafosse

Master of Science in Mathematical Sciences
Submission date: May 2018
Supervisor: Arvid Næss, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

First of all, I wish to thank the Norwegian University of Science and Technology (NTNU) and more precisely two of their professors: Madam Mette Langaas and Sir Arvid Nss. Arvid was my supervisor and help me all along my master's thesis but more importantly he took of his time to talk about statistics with me making me more and more passionate by this topic. Mette made me discover how interesting it can be to deal with data and apply statistics on them.

Then, I know that I do not use to tell them this kind of things but I would like to thank my parents and my sister for all their support during my studies. I would not be where I am without them. Even at more than 2,000 kilometres from them, they continue to give me precious advice and to believe in me.

Finally, I have a special thought for all my friends from Castelnaudary, Toulouse, Lille and Trondheim for their presence and all the energy they gave to me.

Summary

Whatever his strategy is, an investor has to know the risk he will deal with in taking a short or long position on an asset or a derivative. On the financial market, the Value at Risk is one of the values used to evaluate the risk. The main goal of this value is to know up to which amount we can invest without risking a shortfall. Currently, the methods to calculate this rate are developed in using the classical statistics. The three most famous methods are the Variance/Covariance, the historical and the Monte Carlo methods.
Nevertheless, the shortfall is usually caused by an unpredicted event, sometimes coming with an unpredicted cost. Consequently, it could be more coherent to study the Value at Risk inside the Extreme Values Theory. In this paper, we are dealing with the Average Conditional Exceedance Rate (ACER) functions built to permit us to work on the extreme values. However, this method seems to be accurate to predict values in a short interval but presents some limits concerning the calculation of the Value at Risk.
The goal of this Master's thesis is to compare this extreme value technique with the classical ones. We will detail as much as possible the different steps needed to create this method and to implement it.

Table of Contents

Summary i

Table of Contents iii

List of Figures v

Abbreviations vi

1 Introduction 1

2 Value at Risk and classical methods 3
- 2.1 A famous financial notion : the Value at Risk 3
- 2.2 Classical methods 4
  - 2.2.1 Time series model 4
  - 2.2.2 The historical method 8
  - 2.2.3 Monte Carlo method 9
  - 2.2.4 Comparison of the three methods 15

3 The ACER method 17
- 3.1 Extreme values theory 17
  - 3.1.1 Description of the model 17
  - 3.1.2 The extremal types theorems 18
- 3.2 ACER method 25
  - 3.2.1 Approximations to reach the ACER functions 25
  - 3.2.2 The ACER functions 28
  - 3.2.3 The confidence intervals of the ACER functions 30
  - 3.2.4 Fitting to the Asymptotic Gumbel distribution 33
  - 3.2.5 Fitting to a GEV distribution 37

3.2.6 How to choose η_1 and get the VaR 39
3.3 The R package 41
3.3.1 Calculation of Epsilon 41
3.3.2 Approximation of the confidence interval of the Epsilon . 43
3.3.3 Choice of k and η_1, η_f 44
3.3.4 The optimization part and the calculation of the VaR . . . 46

4 Analyses of results **51**
4.1 Execution time and accuracy 51
4.2 Tests on the Pareto distribution 61
4.2.1 The Pareto distribution 61
4.2.2 Test 10% error on the Pareto distribution of type 1 63
4.3 The sensibility of this method and the accuracy for the forecast of the VaR 66
4.3.1 VaR test on a financial data set 66
4.3.2 Limits 68

5 Conclusion **69**

Bibliography **71**

A Stochastic process **75**
A.1 Definition 75
A.2 The Poisson process 76

B Time Series Models **77**
B.1 The ARMA model 77
B.2 the GARCH model 78

C The R codes **79**
C.1 The R main code 79
C.2 The tests' code 86

List of Figures

3.1 Limiting distributions 19
3.2 b as a location parameter 23
3.3 a as a scale parameter 24
3.4 alpha as a shape parameter 24
3.5 Scheme of a bootstrapping with replacement 33
3.6 Logarithm function 34
3.7 Epsilon in function of the threshold 39
3.8 Choice of eta 1 and evolution for different k 45
3.9 ACER functions for different k 46

4.1 Execution times of the three methods in function of the number of data points 51
4.2 Execution time of the last method in function of the number of data points 52
4.3 Time executions of the three methods in function of the number of data points 53
4.4 Epsilon values obtained with the two methods 54
4.5 CI intervals in function of the number of data points for both methods. 54
4.6 Time executions and accuracy of the bootstrapping method in function of the number of samples 55
4.7 Error for the four optimization algorithms in function of the level . 56
4.8 Execution time for the LM algorithm on the simplified and complete model 57
4.9 Calculated and approximate epsilons 57
4.10 Accuracy of the VaR in function of n 58
4.11 Accuracy and execution time in function of the number of levels . 59

4.12 Accuracy and execution time in function of the number of data points 60
4.13 ACER functions for different k 61
4.14 Pareto cumulative function with parameter alpha = 1.16 62
4.15 5,000 Pareto random data with beta = 3 63
4.16 Error in percentage for each method 65
4.17 Values and approximative Values at Risk of the asset 67
4.18 Values and approximative Values at Risk of the asset every 5 days 67

Abbreviations

i.i.d =	Independent and Identically Distributed
GEV =	Generalized Extreme Value
VaR =	Value at Risk
ACER =	Average Conditional Exceedance Rate
MC =	Monte Carlo
CDF =	Cumulative Distribution Function
ARCH =	AutoRegressive Conditional Heteroskedasticity Process
GARCH =	Generalized AutoRegressive Conditional Heteroskedasticity Process
IGARCH =	Integrated Generalized Autoregressive Conditional heteroskedasticity model
ARMA =	AutoRegressive Moving Average model
AR =	AutoRegressive
MA =	Moving Average model
ARIMA =	AutoRegressive Integrated Moving Average
PRNG =	PseudoRandom Number Generator
CI =	Confidence Interval
LM =	Levenberg-Marquardt

Chapter 1

Introduction

In finance, analysts sometimes work on temporally close data. This configuration makes it difficult to assume the independent distribution. Thus, the assumption of identically independent distribution cannot be chosen.

In this paper, we will have a look on unexpected event and more precisely on the value at risk that means the maximum value we can invest up to a certain risk. To deal with low probability events, the extreme value theory seems to be the most efficient. Indeed, the studied distributions will deviate from the median one and consequently the classical statistics may be used with some troubles. Moreover, in extreme value theory, some important results are already documented especially concerning the asymptotic behaviour of a distribution. We know that the extreme distribution will converge to one of the three following model: the Gumbel, the Weidel or the Frechet distribution. To avoid the choice of the model, one generalized them towards the Generalized Extreme Value (GEV) distribution.

Our mission is to fit the distribution of the maxima of our data. To do that, we will study the peaks over a threshold, giving us a technique to extract the large values and how we reach them. Indeed, more than the peaks over a threshold we will develop a function giving us the behaviour of the distribution before being higher than this value. Finally, in choosing a threshold large enough, we will get a distribution whose the behaviour can be managed by the extreme value theory. In reversing the fitted GEV distribution, we will be able to get the VaR which will be a certain value of threshold.

This paper will present how to build these Average Conditional Exceedance Rate functions, model them and then getting the VaR by reversing them. But more important, it will compare this ACER method to the classical covariance/variance, historical and Monte Carlo (MC) methods. Finally, we will show the limits of this technique on a practical case, the asset of the bank Societe Generale.

Chapter 2

Value at Risk and classical methods

2.1 A famous financial notion : the Value at Risk

In financial markets, the art of evaluating the risk is a key to have a control on the safety of an investment [13]. The Value at Risk (VaR) is a notion essentially used in market risk. It estimates the evolution of the amount of a position during a time period in function of the general market movements. In ”Options, futures and other derivatives” of John Hull, they define the VaR by: ”I am X percent certain there will not be a loss of more than VaR dollars in the next N days.” [1]. One famous used of this notion is to ensure that after an event the financial institution will still be in business. In this document, we focus on the use of the VaR for a financial institution that means we use the VaR as the maximal loss during a given time for a given probability.
Now, we define the VaR through a probabilistic framework. Let t be the time index, l be the number of periods and ΔV be the change in value of the asset. Then we have :

$$p = Pr(\Delta V(l) \leq VaR) = F_l(VaR) \tag{2.1}$$

If the cumulative distribution function (CDF) of the distribution, $F_l(x)$, is known, then the VaR is the p^{th} quantile, that means :

$$VaR = \inf\{x|F_l(x) \geq p\} \tag{2.2}$$

So we essentially need to know the CDF of the distribution. In practice, it is unknown and we have to estimate, predict it.
To sum up, the element we need to estimate is the CDF and we should know the probability interest p, the time horizon l, the frequency of data and the amount of the financial position.

2.2 Classical methods

2.2.1 Time series model

Riskmetrics or Variance-Covariance method

This method was developed by J.P. Morgan in 1992 [16]. To make the calculation of the VaR easier, it assumes that the continuously compounded daily return follows a conditional normal distribution. We denote the daily log return by r_t and the information available at $t-1$ by F_{t-1}. The assumption is so :

$$r_t|F_{t-1} \sim N(\mu_t, \sigma_t^2)$$

where μ_t is the conditional mean and σ_t^2 is the conditional variance of r_t. This method also assumes that :

$$\mu_t = 0 \tag{2.3}$$

$$\sigma_t^2 = \alpha\sigma_{t-1}^2 + (1-\alpha)r_{t-1}^2, \text{ with } 0 < \alpha < 1 \tag{2.4}$$

Finally, the logarithm of the daily price, $p_t = ln(P_t)$ satisfies the difference equation $p_t - p_{t-1} = a_t$.
$a_t = \sigma_t\epsilon_t$ follows an IGARCH(1,1) process without a drift.

For a k-return period, denoted by [k], we got the following property for the log return:

$$r_t[k] = r_{t+1} + ... + r_{t+k-1} + r_{t+k}$$

Under the IGARCH(1,1) model assumption, the conditional distribution $r_t[k]|F_t$ is normal with mean zero and variance $\sigma_t^2[k]$. Therefore, we use the equation Eq (2.4) and the independence assumption of ϵ_t, and we get:

$$\sigma_t^2[k] = Var(r_t[k]|F_t) = \sum_{i=1}^{k} Var(a_{t+i}|F_t) = \sum_{i=1}^{k} E(\sigma_{t+i}^2|F_t)$$

In using Eq.(2.4), we have $\sigma_{t+i}^2 = \sigma_{t+i-1}^2 + (1-\alpha)\sigma_{t+i-1}^2(\epsilon_{t+i-1}^2 - 1)$.
We know that $E(\epsilon_{t+i-1}^2 - 1|F_t) = 0$ by definition of the IGARCH model. So we get

$$E(\sigma_{t+i}^2|F_t) = E(\sigma_{t+i-1}^2|F_t), \text{ for } i = 2, ..., k$$

Now, we have $Var(r_{t+1}|F_t) = \sigma_{t+1}^2$ so $\sigma_t[k] = k\sigma_{t+1}^2$, and $r_t[k]|F_t \sim N(0, k\sigma_{t+1}^2)$. Consequently, according to the IGARCH(1,1) model, the conditional standard deviation of a k-period horizon log return is $\sqrt{k}\sigma_{t+1}$.

Now, we want to use these results to find an expression of the VaR. First we need to set the probability to a given level (5% usually). Then according to the normal distribution statistics table, we get the coefficient needed to apply the Risk-Metrics. For example, for a probability of 5%, we obtain the coefficient 1.645, that means $P(Z > 1.645) = 0.05$ where Z follows a standard normal distribution. Then, in this example, the Riskmetrics will use $1.645\sigma_{t+1}$ to measure the risk of the portfolio because σ_{t+1} represents the volatility of the asset at the time t+1. In fact, it is the one-sided 5% quantile of a normal distribution with mean zero and standard deviation σ_{t+1}. Then according to the property of the variance and an affine transformation, we get:

$$\begin{aligned}
\text{VaR} = \text{Amount of a position } &\times 1.645\sigma_{t+1} \\
\text{VaR(k)} = \text{Amount of a position } &\times 1.645\sqrt{k}\sigma_{t+1} \\
\text{And so VaR(k) } &= \sqrt{k}\ \text{VaR}
\end{aligned}$$

Thus the VaR of a k-day horizon can be deduced from the VaR of the day t. So to get the VaR with this method, the recipe is:

- Collect price data
- Create return series by differencing
- Estimate variance of return series
- Take the square root of the variance to get volatility
- Multiply the volatility by 1.645 times position size to get the estimate of 95% worst case loss. This is the VaR.

If the investor has several positions, we use the definition of the variance:

$$\sigma^2(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \sigma^2(X_i) + 2 \sum_{1 \leq i < j \leq n} \text{cov}(X_i, X_j)$$

and we get the generalized formula:

$$\text{VaR} = \sqrt{\sum_{i=1}^{m} \text{VaR}_i^2 + 2\sum_{i<j}^{m} \rho_{ij} \text{VaR}_i \text{VaR}_j}$$

where $\rho_{ij} = \frac{\text{Cov}(r_{it}, r_{jt})}{\sqrt{\text{var}(r_{it})\text{var}(r_{jt})}}$ is the cross-correlation coefficient between the returns of the i^{th} and j^{th} positions.

Econometric approach

The previous method is simple but requires some strong assumptions like the normal one. It is sometimes difficult to check them like for fat tails and then the method returns an underestimated VaR. In the following method, we will still use a GARCH model but we will make different hypotheses in order to be able to reach a correct VaR in more cases.
We consider again the log return r_t of an asset [8].
By choosing to fit our problem with a GARCH model, we get the following system:

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + a_t - \sum_{j=1}^{q} \theta_j a_{t-j} \tag{2.5}$$

$$a_t = \sigma_t \epsilon_t$$

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{u} \alpha_i a_{t-i}^2 + \sum_{j=1}^{v} \beta_j \sigma_{t-j}^2 \tag{2.6}$$

The equations (2.5) and (2.6) represent the mean and volatility equations for r_t. The k-step ahead forecasts of the conditional mean and variance of r_t can be done using these two equations if we already know the parameters of the GARCH model by estimating them first. We get :

$$\hat{r}_t(1) = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t+1-i} - \sum_{j=1}^{q} \theta_j a_{t+1-j}$$

$$\hat{\sigma}_t^2(1) = \alpha_0 + \sum_{i=1}^{u} \alpha_i a_{t+1-i}^2 - \sum_{j=1}^{u} \beta_j \sigma_{t+1-j}^2$$

where u and v are smaller than t.
Now, we need to guess the distribution of ϵ_t. Two cases seem to present the best ratio complexity/efficiency. First, we can assume that ϵ_t is Gaussian, then the conditional distribution of r_{t+1} follows $N[\hat{r}_t(1), \hat{\sigma}_t^2(1)]$. Quantiles can easily be obtained and so the VaR. For example, the 5% quantile is $\hat{r}_t(1) - 1.645\hat{\sigma}_t^2(1)$.
Secondly, one can assume that ϵ_t follows a standardized Student-t distribution with v degrees of freedom. Then the quantile used to get the VaR will be $\hat{r}_t(1) - t_v^*(p)\hat{\sigma}_t^2(1)$ where $t_v^*(p)$ is the p^{th} quantile of a standardized Student-t distribution with v degrees of freedom. We only need to determine the relation between quantiles of a Student-t distribution with v degrees of freedom, t_v, and those of its standardized distribution, t_v^*. We know that:

$$P(X \leq q) = P(X^* \leq \frac{q}{\sqrt{v(v-2)}}$$

where $\upsilon > 2$, X follows a Student-t distribution and X^* a standardized one.
So if q is the p^{th} quantile of a Student-t distribution with υ degrees of freedom, then $\frac{q}{\sqrt{\upsilon(\upsilon-2)}}$ is the p^{th} quantile of a standardized Student-t distribution with υ degrees of freedom.
To conclude, the quantile used to calculate the 1-period horizon VaR at time t is:

$$\hat{r}_t(1) - \frac{t_\upsilon(p)\hat{\sigma}_t^2(1)}{\sqrt{\upsilon(\upsilon-2)}}$$

where $t_\upsilon(p)$ is the p^{th} quantile of a student-t distribution with υ degrees of freedom.
In this part, we want to extend the previous results for multiple periods forecasting. Let F_h be the set of data known at time h. We provide more explanations about the Time Series models in appendix.
We want to find the conditional mean $E(r_h[k]|F_h) = r_h(1) + r_h(2) + ... + r_h(k)$. To obtain the forecast of r_t we will transform the ARMA model of the Eq.(2.5) in an infinite MA representation:

$$r_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + ...$$

Moreover, we should find the forecast error to know how accurate is the result. We can write the l-step ahead forecast error at the time origin h as:

$$\begin{aligned} e_h(l) =& r_{h+l} - r_h \\ =& a_{h+l} + \psi_1 a_{h+l-1} + ... + \psi_{l-1} a_{h+1} \end{aligned}$$

And so the forecast error of the expected k-period return $\hat{r}_h[k]$ is the sum of the forecast errors of r_t at the origin h up to $h + k$. We have:

$$\begin{aligned} e_h[k] =& e_h(1) + ... + e_h(k) \\ =& a_{h+1} + (a_{h+2} + \psi_1 a_{h+1}) + ... + \sum_{i=0}^{k-1} \psi_i a_{h+k-i} \\ =& a_{h+k} + \sum_{j=1}^{k-1} (\sum_{i=0}^{j} \psi_i) a_{h+k-j} \end{aligned}$$

where $\psi_0 = 1$
We can try to guess the expected volatility. The volatility forecast of the k-period return at origin h is the conditional variance of $r_h[k]$ given F_h. We will use the independent assumption of ϵ_{t+i} with i = 1,...,k and the pseudo linear property of

the variance.
We have:

$$\begin{aligned}\text{var}(e_h[k]|F_h) =& \text{var}(a_{h+k}|F_h) + (1+\psi_1)^2\text{var}(a_{h+k-1}|F_h) + ... \\ &+ (\sum_{i=0}^{k-1}\psi_i)^2\text{var}(a_{h+1}|F_h) \\ =& \sigma_h^2(k) + (1+\psi_1)^2\sigma_h^2(k-1) + ... + (\sum_{i=0}^{k-1}\psi_i)^2\sigma_h^2(1)\end{aligned}$$

where $\psi_0 = 1$ and $\sigma_h^2(l)$ is the l-step ahead volatility forecast at the forecast origin h.
If the volatility model is the GARCH model in Eq (2.6) then this volatility forecast can be computed recursively. Indeed, we can directly get $\sigma_h(1)$ from the equation Eq (2.6).
If ϵ_t is Gaussian, then the conditional distribution of $r_h[k]$ given F_h is normal with mean $k\mu$ and variance var($e_h[k]|F_h$). We can then get the quantiles needed in VaR calculation.

2.2.2 The historical method

The historical method uses the past performance, data to forecast the new ones. In fact, this method assumes that "the past is a good indicator of the near-future." We can split this method of Full Valuation in 4 steps [8].

First, we have to calculate the returns, also called price changes, of all the assets in the portfolio for each time interval. For example, if we have the past year of daily data, we can choose a time interval of one day. This requirement can show one weakness of this method. Indeed, Historical Simulations VaR requires a long history to get a meaningful VaR. One year of monthly returns should not be sufficient to get a VaR we can trust.

Then, we applied each calculated price changes to the current value of the asset. We assume that these returns can appear with the same likelihood than before. So for each time interval we get a new value of our asset and then we can re-value our portfolio. For example, for a daily return with 365 data points, we will get 365 simulations.

Finally, we have to sort our results from the lowest to the highest value. In function of the wanted confidence level, we can now "read" the value of the VaR. Indeed the VaR at $(1-\alpha)\%$ confidence level is the mean of the simulated values minus the $\alpha\%$ lowest value because we are working with the returns. We have:

$$\text{VaR}_{1-\alpha} = \mu(R) - R_\alpha$$

where

- $VaR_{1-\alpha}$ is the estimated VaR at the confidence level $(1-\alpha)$
- $\mu(R)$ is the mean of the simulated returns
- R_α is the return of the simulated series that corresponds to the level of significance α.

Another implementation of the historical method is to get again all the simulations, sort them and then take the one corresponding to the $1-\alpha$ percentile.
But now the i^{th} simulation will be:

$$v_{n+1}^{i} = v_n \frac{v_i}{v_{i-1}}$$

where v_n corresponds to the actual value of the asset, and $i < n$.

2.2.3 Monte Carlo method

In this subsection, we will explain one of the most famous and used method to calculate the VaR : the Monte Carlo (MC) simulation. This method can be applied on all portfolios and so we can calculate all VaR with it. Indeed, it uses no assumption about the shape of the volatility or the response of the market about an event. MC method will revalue the VaR in each scenario.

Work on the data

First to use the MC simulation [7], we have to identify our assets and to transform their values in percentages of change. The horizon time, k, will be required to know the data we will use. Indeed, the quantity of data is directly related to the accuracy and the length of the forecast. If we want a solution more accurate or a longer forecasting, we will have to use more data points. We can then calculate the new data vector for each asset in using the following formula :

$$V_{new,\ asset\ i}(t) = \frac{V_{old,\ asset\ i}(t+k) - V_{old,\ asset\ i}(t)}{V_{old,\ asset\ i}(t)}$$

Secondly, we have to get the covariance matrix Σ of the data where each column is a data vector. The classical formula using the mean can be used and so we get :

$$\Sigma(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1} = \Sigma(Y,X)$$

where

- X and Y are two different assets
- n is the number of data points.
- $\bar{X}$ is the mean of X : $\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$

We can also use a more statistical definition of the covariance matrix:

$$\Sigma(X,Y) = E[(X - E(X))(Y - E(Y))^T]$$

where E() is the expected value.

We know, by definition, that $E(AX + a) = AE(X) + a$ where A is a matrix $q \times p$, X a vector $p \times 1$ and a a vector $q \times 1$.

Now we will discuss some important properties of the covariance matrix and so of Σ. Obviously, we can notice that the matrix is symmetric by construction. Another nice property for us is the possibility to reduce the number of studied assets in using:

$$\begin{cases} Cov(X+Z,Y) = Cov(X,Y) + Cov(Z,Y) \\ Cov(X,Y+Z) = Cov(X,Y) + Cov(X,Z) \end{cases} \tag{2.7}$$

The covariance matrix can be directly modified in case of linear change of the market. Indeed, we have:

$$Cov(AX + a, BY + b) = ACov(X,Y)B^T. \tag{2.8}$$

Now, we are giving a proof of the two previous results:

$$\begin{aligned} Cov(X+Z,Y) &= \frac{\sum_{i=1}^{n}(X_i + Z_i - \bar{X} - \bar{Z})(Y_i - \bar{Y})}{n-1} \\ &= \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) + \sum_{i=1}^{n}(Z_i - \bar{Z})(Y_i - \bar{Y})}{n-1} \\ &= \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1} + \frac{\sum_{i=1}^{n}(Z_i - \bar{Z})(Y_i - \bar{Y})}{n-1} \\ &= Cov(X,Y) + Cov(Z,Y) \end{aligned}$$

$$\begin{aligned} Cov(AX + a, BY + b) &= E[(AX + a - E(AX + a))(BY + b - E(BY + b))^T] \\ &= E[(AX + a - AE(X) - a)(BY + b - BE(Y) - b)^T] \\ &= E[A(X - E(X))(B(Y - E(Y)))^T] \\ &= E[A(X - E(X))(Y - E(Y))^T B^T] \\ &= AE[(X - E(X))(Y - E(Y))^T]B^T \\ &= ACov(X,Y)B^T \end{aligned}$$

Finally, with the previous properties, we will prove the semi definite positiveness of the covariance matrix. According to our linear algebra classes, Σ is positive definite if and only if for all vector u, we have $u\Sigma u^T \geq 0$. Let u be a vector.

$$u^T\Sigma u = \sum_{i=1}^{n}\sum_{j=1}^{n} u_i Cov(X_i, X_j)u_j$$

in using (2.8) because u_i is a number so equal to his transpose,

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} Cov(u_i X_i, u_j X_j)$$

according to (2.7),

$$= Cov(\sum_{i=1}^{n} u_i X_i, \sum_{j=1}^{n} u_j X_j)$$
$$= Var(\sum_{i=1}^{n} u_i X_i) \geq 0$$

, by definition of the variance.

Thus, the covariance matrix is semi definite positive.

Nevertheless, we can face some difficulties if some data points are missing like 800 values for the asset 1 and 650 for the asset 2. Indeed, the covariance matrix will not be diagonalizable as required for the next step. The best thing to do is to approximate the missing values by an interpolation, a Brownian bridge or some regressions.

Diagonalization of the covariance matrix

We know that Σ is diagonalizable and symmetric and so we want to write it as: $\Sigma = VDV^T$ where D is a diagonal matrix with the eigenvalues of Σ on the diagonal and V is the matrix containing the eigenvectors of Σ. We can write V^T and notV^{-1} because Σ is symmetric:

$$\Sigma = \Sigma^T \Leftrightarrow VDV^{-1} = (V^{-1})^T DV^T$$

by identification, we have: $V^T = V^{-1}$

Then to build this matrix, we need the eigenvalues of Σ and the eigenvectors associated. The function eigen(x, symmetric) on R will permit us to get them. [12]
How works this function?
First, Σ, a real symmetric matrix, is reduced to real tridiagonal matrix T. Then we have: $\Sigma = QTQ^T$ where Q is orthogonal and T tridiagonal. The goal of this reduction is to make the research of the eigenvalues easier. Indeed, one can easily prove that the eigenvalues of Σ are the same as the ones of T (much easier to analyse). To get the matrix T, the function eigen() uses the Householder algorithm. The Householder reflection theorem is fundamental in this algorithm.

Theorem 2.2.1. *If X and Y are vectors with the same norm, there exists an orthogonal symmetric matrix P such that:*

$$Y = PX$$
$$\textit{where, } P = I - 2WW^T,\ W = \frac{X - Y}{\|X - Y\|}$$

P is called a Householder matrix, and is symmetric and orthogonal. That means we can, for each step k, find a Householder matrix such that: $P_k x_k = y_k = (0, ..., 0, v, x_k(k), v, 0, ..., 0)^T$ where $x_k(k)$ is the k^{th} element of the vector x_k and v is chosen to validate the requirement of the Householder reflection theorem. We can use this process for the n column vectors of Σ and so get a sequence of Householder matrices. We complete the previous matrices by some 1 and 0 in order to not change the values of the other columns. Finally we get T by multiplying on both sides Σ by the product of the Householder matrices: $T = P_n P_{n-1} ... P_1 \Sigma P_1 ... P_{n-1} P_n$ where Σ is a n×n matrix.
Now, we will apply the QR algorithm on the matrix T. The iterations are quite simple, we will decompose the matrix A as a product of an orthogonal matrix Q and an upper triangular one R, $A = QR$. Then A will get RQ. And we will iterate that until all values of A below the diagonal will converge to 0. The values on the diagonal of A will be equal to the eigenvalues of the original matrix. At the last iteration, Q will contain the eigenvectors of A.

Finally, the eigenvectors' matrix of Σ will be the multiplication of $P_n P_{n-1} ... P_1$ by the matrix Q. The main advantage of the tridiagonalization is to reduce the complexity of each step of the QR-algorithm from $O(n^3)$ to $O(n)$.

Creation of the random vector

In parallel with the building of the covariance matrix, we need to generate a random vector of length n. Of course, it is impossible to create an algorithm generating random numbers for the simple fact that is an algorithm. As we are working with the software R, we will develop the algorithm used for MC simulation with R. Obviously, producing a vector of n pseudo random numbers between 0 and 1 is done by R with the function: runif(n, min = 0, max = 1) But now the relevant question is: which algorithm is used by this function? This function uses the algorithm of Mersenne Twister [26] designed by Makoto Matsumoto and Takuji Nishimura in 1997. It is a pseudorandom number generator (PRNG). Its period length is equal to the Mersenne prime, $2^{19937} - 1$ (explaining its name). It is the most common PRNG, efficient for a lot of simulations like MC simulation but not enough secure to be used in cryptography. To understand this algorithm, we need some background about bits' operations and how a computer works:

- First of all, we need to know that computers write integers with bits that means an integer X will be saved as $(x_w, ..., x_0)$ where X is written as $\sum_{i=0}^{w} x_i 2^i$ where n depends also on the memory size of the device, X is decomposed on the 2 basis.

- Secondly, the shift, "XOR" and "AND" operators are essential to deal with computers and numbers. The left (right) shift operator can be symbolized by $\ll$ ($\gg$) and we have for a v bits shift:

$$X \ll v = (x_{w-v}, ..., x_0, 0, ..., 0)$$
$$X \gg v = (0, ..., 0, x_w, ..., x_{v+1})$$

  The "XOR" and "AND" operators are well known, symbolized by $\oplus$ and $\otimes$. Let X be equal to $\sum_{i=0}^{w} x_i 2^i$ and Y to $\sum_{i=0}^{w} y_i 2^i$. Then we have :

$$X \oplus Y = \sum_{i=0}^{w} (x_i \oplus y_i) 2^i, \text{ where } x_i \oplus y_i = x_i + y_i \text{ modulo } 2$$
$$X \otimes Y = \sum_{i=0}^{w} (x_i \otimes y_i) 2^i, \text{ where } x_i \otimes y_i = x_i \times y_i \text{ modulo } 2$$

- Moreover, we have to introduce a new function A, key of the algorithm. Let x be a number such as $x = (x_w, ..., x_0)$ in bits' notation.

$$A(x) = \begin{cases} (x \gg 1) \oplus 0 \text{ if x is even} \\ (x \gg 1) \oplus a \text{ otherwise} \end{cases}$$

  where a is a given constant.

- Finally, we introduce to key values $\underline{M}_r$ where the r first bits are equal to 1 and the others to 0 and $\overline{M}_r$ where the r last bits are equal to 1 and the others to 0.

The algorithm of Mersenne Twister can be decomposed in two essential steps. The recurrence operation is followed by a tempering one whose the goal is to create more entropy. The recurrence operation can be written as :

$$X_{k+n} = X_{k+m} \oplus A[(X_{k+1} \otimes \underline{M}_r) \oplus (X_k \otimes \overline{M}_r)]$$

where n, r, k are positive constant and $0 \leq m \leq n$. The tempering process is defined by:

$$\begin{aligned}
&Y \leftarrow X_{k+n}\\
&Y \leftarrow Y \oplus (Y \gg u)\\
&Y \leftarrow Y \oplus ((Y \ll s) \otimes b)\\
&Y \leftarrow Y \oplus ((Y \ll t) \otimes c)\\
&Y \leftarrow Y \oplus (Y \gg l)
\end{aligned}$$

where X_{k+n} is the result of the previous step and u, s, b, t, c, l are some given constants. The main goal of this step is to mix the bits of X_{k+n} and so create some entropy.

Now, we just need to apply a function to get a value between 0 and 1 :

$$U_k = \frac{Y_k + 0.5}{2^w}$$

The most famous Mersenne Twister generator is the MT19937 generating pseudorandom numbers U_k with the following parameters:

- ω = 32, word size
- n = 624, degree of recurrence
- r = 31, separation point of one word
- m = 397, middle word
- a = 2,567,483,615, coefficients of the rational normal form twist matrix
- u = 11 and l = 18, additional Mersenne Twister tempering bit shifts/masks
- s = 7 and t = 15, TGFSR(R) tempering bit shifts
- b = 2,636,928,640 and c = 4,022,730,752, TGFSR(R) tempering bitmasks

This choice of parameters permits us to maximize the period, equals to $2^{w \times n - r} - 1$ which is a Mersenne prime.

Get the VaR

As Σ, the covariance matrix, is symmetric, positive semi definite, we know that it is diagonalizable with positive eigenvalues according to the linear algebra. Thus, we can write : $\Sigma = V^T \Lambda V$ where Λ is a diagonal matrix with the eigenvalues of Σ on its diagonal, and V is the eigenvectors' matrix. As the eigenvalues are positive, we can write: $\Sigma = V^T \Lambda^{\frac{T}{2}} \Lambda^{\frac{1}{2}} V$ and notice that $\Sigma = B^T B$ where $B = \Lambda^{\frac{1}{2}} V$.

And now comes the interesting and surprising part. Let X be a vector of length n and with values randomly distributed between 0 and 1: $X \sim N(0, 1)$.
We can first notice that $XB \sim N(0, \Sigma)$,

$$E(XB) = E(X)B = 0, \text{ because B is a constant matrix and E(X) = 0}$$
$$Var(XB) = B^T Var(X) B = B^T B = \Sigma, \text{ because Var(X) = 1}$$

We manage to generate for each asset, a vector of length n as big as we want, with the same characteristics than the data vector. Or the Value at Risk is contained in the tail of the curse. In increasing the number of values we are improving the accuracy and so we are doing like a zoom on the tail. The value of the VaR will be better.

For example with 100 values of the asset 1, we can generate a vector with 10,000 values with the same characteristics. And now the VaR at level 5% will not be the 95^{th} value but the 9500^{th}.

2.2.4 Comparison of the three methods

The historical method and the Variance/Covariance method are limited by their first assumption. Indeed, they use a well known distribution (normal one or t-student one) to approximate the data or one statistics of the data.
The historical simulation method requires an important amount of data. The accuracy of this method is directly related to this amount.
The first problem that faced the MC method was the speed of the method. To deal with that, one needed to speeding up each revaluation or sampling fewer scenario or both. Indeed, a diagonalization of a matrix has a huge cost in time. Nevertheless, the biggest asset of this method is the accuracy we can get with a normal sized set of data. Moreover, it allows for any distribution and securities (we do not need to assume the distribution to be normal for example).

Chapter 3

The ACER method

3.1 Extreme values theory

3.1.1 Description of the model

In extreme value theory, we focus on the minimal and maximal values of the data set and in our study we will be more interested by the maximum to get the VaR. In all the theoritical part, we used the followinf references: [3] [4] [15] [20] [11]. So, the model's goal is essentially to represent the statistical behaviour of $M_n = \max\{X_1, ..., X_n\}$ where $X_1, ..., X_n$ is a sequence of independent random variables coming from our data and following a common distribution function F. For example, if the X_is represent the hourly value of an asset, M_n could be the daily return of this asset and so n would be equal to 12.
Theoretically, the distribution of M_n can be derived directly from all values of X_i because:

$$\begin{aligned} Pr(M_n \leq z) &= Pr(X_1 \leq z, ..., X_n \leq z) \\ &= Pr(X_1 \leq z) \times ... \times Pr(X_n \leq z) \end{aligned}$$

because the X_i's are independent,

$$Pr(M_n \leq z) = [F(z)]^n$$

But usually we do not know the distribution F. Indeed, when the data is collected, we get just some values, points. Moreover, estimating F from the observed data even with accurate techniques seems to not be efficient especially concerning

the accuracy.Indeed, a small error about F can be a big one for the distribution of M_n because of the power function.
The only available solution is to study directly the distribution of M_n, which is $[F(z)]^n$, to avoid the use of the power function. It is for this purpose that the theory of extreme value was developed. This theory will be really close to the usual practice of approximating the distribution of sample means by the normal distribution, that means the use of the central limit theorem.

In extreme value theory, we are looking the behaviour of F^n as n tending to infinity. This raw analysis faces a problem of degeneration of F^n. For all $z < z^*$ we have $F^n(z) \to 0$ as $n \to \infty$, z^* is defined by: $z^* = \inf\{z|F(z) = 1\}$.
To avoid this problem, we use a linear re-normalization of the variable M_n:

$$M_n^* = \frac{M_n - b_n}{a_n} \tag{3.1}$$

where a_n is a sequence of strictly positive constants and b_n is a sequence of constants.Thanks to these two sequences, the location and scale of M_n^* are stabilized. Thus, now we will try to approximate the distribution of M_n^* with good stabilizers instead of working with M_n.

3.1.2 The extremal types theorems

This theorem, also called the Fisher-Tippett-Gnedenko theorem, gives us the entire range of possible limit distributions for M_n^*.
The most important theorem in this paper can be stated as:

Theorem 3.1.1. *If there exist sequences of constants $a_n > 0$ and b_n such that:*

$$Pr(\frac{M_n - b_n}{a_n} \leq z) \to G(z), \text{ as } n \to \infty$$

, where G is a non-degenerate distribution function, then G belongs to one of the following families:

1. $G(z) = \exp[-\exp(-\frac{z-b}{a})], \; -\infty < z < \infty$;

2. $G(z) = \begin{cases} 0, & z \leq b \\ \exp[-\frac{z-b}{a}^{-\alpha}], & z > b \end{cases}$

3. $G(z) = \begin{cases} \exp[-(-\frac{z-b}{a}^{\alpha})], & z < b \\ 1, & z \geq b \end{cases}$

for parameters $a > 0$, b and $\alpha > 0$.

This theorem gives us three possible models for the distribution of $F^n(z)$ for a number of points big enough.
The three distributions are called the extreme value distributions. The first one is the Gumber extreme value distribution, the second one is the Fréchet one and the last one is the Weibull one.
a is the scale parameter, b the location one and α is the shape parameter when it exists. The strength of this result is that when M_n is stabilized with suitable sequences a_n and b_n, then the normalized variable M_n^* can only have one of the three limiting distributions seen previously.

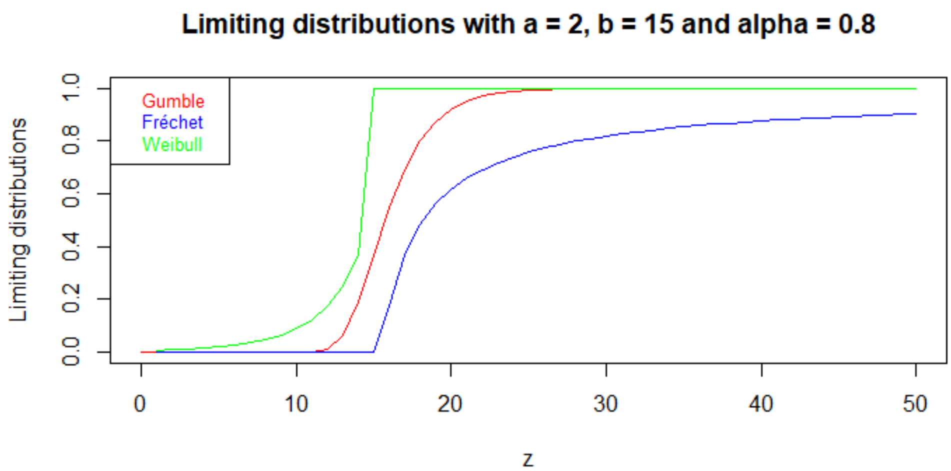

Figure 3.1: Limiting distributions

However, the choice of the model can become a big issue. As we can see on the chart, the major difference between the three models takes place in the tail of the plot, more precisely for $z > z^*$, defined previously. The plot represents the three distributions applied on a data composed of values following an uniform law between 0 and 50.
If z^* is finite, we should choose the Weibull distribution, if z^* is infinite we can fit the data by the Fréchet or the Gumbel model. This remark can easily be verified with the formula of the three models.
Then to differentiate the Fréchet and the Gumbel model, we have to analyse the decrease of the density of G, specially on the tail of the plot. If it decays exponentially, the best fit will be the Gumbel distribution, if it is polynomially the best will be the Fréchet one.
Even if it seems possible to choose rationally one of the three models, the need to

use a technique to fit the data with a distribution brings at least one more step into the calculation and consequently some inferences due to a tiny difference between the models for a data set. Consequently, it is easier to reformulate the problem and give a general formulation of the limiting distribution. So the Fisher-Tippett-Gnedenko theorem becomes :

Theorem 3.1.2. *If there exist sequences of constants $a_n > 0$ and b_n such that:*

$$Pr(\frac{M_n - b_n}{a_n} \leq z) \to G(z), \; asn \to \infty$$

for a non-degenerate distribution function G, then G is a member of the Generalized Extreme Value (GEV) family:

$$G(z) = \exp[-(1 + \xi\frac{z-\mu}{\sigma})^{-\frac{1}{\xi}}] \tag{3.2}$$

defined on $\{z|1 + \xi\frac{z-\mu}{\sigma} > 0\}$, where $-\infty < \mu < \infty$, $\sigma > 0$ and $-\infty < \xi < \infty$.

Now, we will show the equivalence between the two theorems.

- $\xi < 0$,
  The defined set becomes $\{z|z < \frac{-\sigma}{\xi} + \mu\}$ which is well of the form $z < b$. Moreover the general form with $\xi < 0$ is close to the Weibull distribution on the varying part (that means for $z < b$).

- $\xi > 0$,
  The defined set becomes $\{z|z > \frac{\sigma}{\xi} + \mu\}$ which is well of the form $z > b$. Furthermore, the general form with $\xi < 0$ is close to the Fréchet distribution on the varying part (that means for $z > b$).

- $\xi \to 0$,
  To show the similarity between the GEV distribution with $\xi \to 0$ and the Gumbel distribution we can work on the GEV distribution and show that the Taylor expansion of order 1 is equal to the Gumbel distribution.

$$\begin{aligned} G(z) &= \exp[-(1 + \xi\frac{z-\mu}{\sigma})^{-\frac{1}{\xi}}] \\ &= \exp[-\exp(-\frac{1}{\xi}\ln(1 + \xi\frac{z-\mu}{\sigma}))] \\ &= \exp[-\exp(-\frac{1}{\xi}\ln(1 + x)] \end{aligned}$$

, where $x = \xi\frac{z-\mu}{\sigma}$, $x \to 0$ when $\xi \to 0$

$$
\begin{aligned}
&= \exp[-\exp(-\frac{1}{\xi}(x + O(x^2))] \\
&\underset{x\to 0}{\sim} \exp[-\exp(-\frac{1}{\xi}x)] \\
&\underset{x\to 0}{\sim} \exp[-\exp(-\frac{1}{\xi}\xi\frac{z-\mu}{\sigma})] \\
&\underset{x\to 0}{\sim} \exp[-\exp(-\frac{z-\mu}{\sigma})]
\end{aligned}
$$

Thus, according to the defined set is $\{z|1 > 0\}$ and it is the same as for the Gumbel distribution: $-\infty < z < \infty$, we proved that the GEV distribution for $\xi = 0$ is in fact the Gumbel distribution.

The equivalence between the two theorems is true and with the GEV family we avoid the mistakes and the inferences bring by the choice of the model. Indeed, the data itself determines directly the type of tail behaviour through the value of ξ. Finally with ξ and more precisely the lack of certainty in the value of ξ, we can measure the accuracy for the data to fit one of the three models.

The normalizing constants a_n and b_n, unknown, seems to be a difficulty to face. Nevertheless, for large value of n, if $Pr(\frac{M_n-b_n}{a_n} \leq z) \approx G(z)$ then we have:

$$Pr(M_n \leq z) \approx G(\frac{z-b_n}{a_n}) = G^*(z)$$

Obviously, if G is a member of the GEV family, then G^* is also a member of this family.
We will give a sketch of the proof of the Fisher-Tippett-Gnedenko theorem. First we need the definition of max-stable. A distribution G is max-stable if, for every n=2, 3, ..., there are constants $\alpha_n > 0$ and β_n such that $G^n(\alpha_n z + \beta_n) = G(z)$. The following lemma will be assumed true because the proof requires complex analysis knowledge:

Lemma 3.1.3. *A distribution is max-stable if and only if it is a GEV distribution.*

Thanks to this theorem the proof of the Fisher-Tippett-Gnedenko theorem is much easier.

Proof. Let M_{nk} be the maximum random variable in a sequence of $n \times k$ identically distributed variables, n is large enough.
We can see M_{nk} as the maximum of a sequence of nk values or as the maximum

of a sequence of k values where each value is the maximum of a sequence of n observations.
Let G be the limit distribution of $\frac{M_n - b_n}{a_n}$.
As n is large enough, by the Fisher-Tippett-Gnedenko theorem, we have:

$$Pr(\frac{M_n - b_n}{a_n} \leq z) \approx G(z) \tag{3.3}$$

So for any integer k, as nk is large, we get: $Pr(\frac{M_{nk} - b_{nk}}{a_{nk}} \leq z) \approx G(z)$ that means $Pr(M_{nk} \leq z) \approx G(\frac{z - b_{nk}}{a_{nk}})$.
On the other hand, M_{kn} is the maximum of k variables, having the same distribution as M_n, $Pr(\frac{M_{kn} - b_n}{a_n} \leq z) = [Pr(\frac{M_n - b_n}{a_n} \leq z)]^k$. So $Pr(M_{nk} \leq z) \approx G^k(\frac{z - b_n}{a_n}$. Now we are doing the change of variable : $z* = \frac{z - b_{nk}}{a_{nk}}$ and we get:

$$G(z*) \approx G^k(\frac{a_{nk} z * + b_{nk} - b_n}{a_n}) \approx G^k(\alpha_k z * + \beta_k)$$

where $\alpha_k = \frac{a_{nk}}{a_n}$ and $\beta_k = \frac{b_{nk} - b_n}{a_n}$.
Thus, G is max-stable and therefore a member of the GEV family by the previous theorem. □

Finally, in this last part, we will talk a bit about one classical estimation method, the maximum likelihood estimation. This well-known technique requires some regularity conditions satisfies when $\xi > -0.5$. In this case the maximum likelihood estimators are regular and we have the usual asymptotic results. For the following part, we will work under this condition.
We are still assuming that the variables $Z_1, ..., Z_n$ are independent ad follow a GEV distribution. We have two cases to treat in function of the value of ξ:

- $\xi \neq 0$,
  The log-likelihood for the GEV parameters is:

$$\begin{aligned} l(\mu, \sigma, \xi) = &- n \log(\sigma) - (1 + \frac{1}{\xi} \qquad \sum_{i=1}^{n} \log(1 + \xi(\frac{z_i - \mu}{\sigma})) \\ &- \sum_{i=1}^{n} (1 + \xi(\frac{z_i - \mu}{\sigma})^{-\frac{1}{\xi}}) \end{aligned}$$

  provided that,

$$1 + \xi(\frac{z_i - \mu}{\sigma}) \qquad > 0, \text{ for i = 1, ..., n.}$$

When the last condition is not respected, that means at least one of the observed data falls beyond an end-point of the distribution, the log-likelihood equals $-\infty$.

- $\xi = 0$,
  In using the Gumbel limit of the GEV distribution, we get the following log-likelihood:

$$l(\mu,\sigma) = -n\log(\sigma) - \sum_{i=1}^{n} \frac{z_i - \mu}{\sigma} - \sum_{i=1}^{n} \exp(-\frac{z_i - \mu}{\sigma})$$

Thanks to this result, we have a technique to estimate the value of the parameters in maximising the log-likelihood.

To illustrate the role of each parameters we will use a data vector X uniformly distributed between 0 and 50: $X = (0, 1, 2, ..., 50)$. As said previously, these distributions have a behaviour inside the extreme value field quite similar to the one of the normal distribution in the usual statistical field.
First the parameter b is a position parameter. When the value of b is modified, we just translate the chart to the right or the left: it is a translation on the x axis.
This result is coherent with the following plot but also with the formula of the limiting distribution. The role of b is similar to the one of the mean μ in the normal distribution.

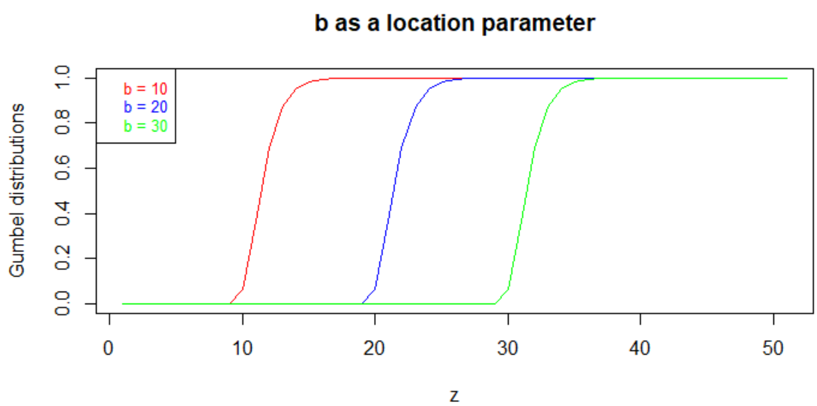

Figure 3.2: b as a location parameter

Secondly, the parameter a will have an influence on the scale of the curve. Similarly to the variance for the cumulative distribution function of the normal law, it will modify the angle of the change of curve explaining the name of scale parameter. Again, this can be seen on the following plot but also in the formula of the limiting distributions (quite similar to the normal one).

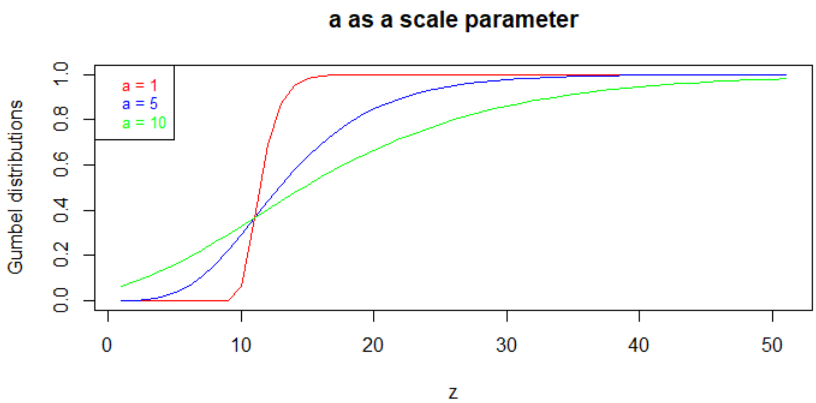

Figure 3.3: a as a scale parameter

Finally, the parameters α or ξ, depending on if we use the GEV distribution or the Fréchet/ Weibull distributions, will modify the shape of the graph, especially on the tail. We can see that onto the following graphs.

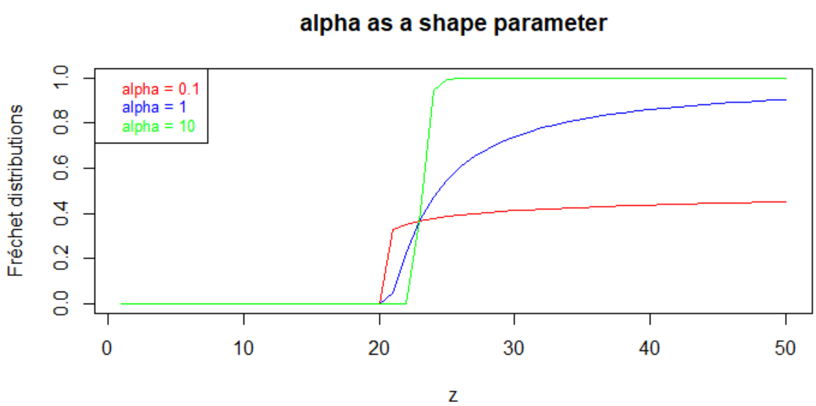

Figure 3.4: alpha as a shape parameter

3.2 ACER method

Now, we will develop the core of this paper, the estimation of extreme values by the Average Conditional Exceedance Rate (ACER) method and more precisely how to get the VaR from this method. The studied ACER method was explained in the research article published in 2013 by A. Naess, O. Gaidai and O. Karpa [18].
The strength of this method is to take in consideration possible statistical dependence between the data and to avoid to ensure the independence of the data, a condition often needed in methods to estimate extreme values. The main goal of this method is to offer more flexibility than the others like asymptotic extreme value distributions.

3.2.1 Approximations to reach the ACER functions

We consider a stochastic process Z(t) (definition given in annexe) observed over a time interval (0,T). From this process, values allocated to the discrete times in (0,T) are derived to give us $X_1, ..., X_n$. These values can be either Z(t) evaluated in times $t_1, ..., t_n$ inside the time interval (0,T), or a statistics of values of Z(t) in N different points (like average values on interval of length $\frac{T}{N+1}$).

In this paper, we are studying the VaR and so we are working with the extreme value theory. Consequently, we will focus on how to determine the distribution of the extreme value $M_n = \max(X_1, .., X_n)$ and more precisely how to estimate $P(\eta) = Pr(M_n \leq \eta)$ for large values of η.
So by definition and some calculations, we get:

$$\begin{aligned} P(\eta) &= Pr(M_n \leq \eta) \\ &= Pr(X_N \leq \eta, ..., X_1 \leq \eta) \\ &= Pr(X_N \leq \eta | X_{N-1} \leq \eta, ..., X_1 \leq \eta) \times Pr(X_{N-1} \leq \eta, ..., X_1 \leq \eta) \\ &= \prod_{j=2}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta) \times Pr(X_1 \leq \eta) \end{aligned}$$

So,

$$P(\eta) = \prod_{j=2}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta) \times Pr(X_1 \leq \eta) \tag{3.4}$$

In this method, we are assuming that the variables X_j are statistically dependent what is true most of the time. If we were under the assumption that all the X_j are statistically independent we could get the classical approximation (often used in the other methods):

$$P(\eta) \approx P_1(\eta) = \prod_{j=1}^{N} Pr(X_j \leq \eta).$$

For the ACER method, we will build a sequence of probabilities where the value k will correspond to the number of steps memory approximation we will account that means the number of points we will tale for the dependence between the X_j's. For example, if k =8, we will assume that the value of X_j is dependent only of the values of X_{j-i} where i = 1,...,8. Consequently, X_j will be independent of the values anterior to X_{j-8}.We will show how to build recursively this sequence. For $k = 2$ we have :

$$Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta)$$
$$\approx Pr(X_j \leq \eta | X_{j-1} \leq \eta) \text{ , for } 2 \leq j \leq N$$

Then we get: $P(\eta) \approx P_2(\eta) = \prod_{j=2}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta) Pr(X_1 \leq \eta)$

We assume now that for a general k, $2 \leq k \leq N-1$, we have:

$$\begin{aligned} P(\eta) \approx & P_k(\eta) \\ = & \prod_{j=k}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_{j-k+1} \leq \eta) \\ & \prod_{j=2}^{k-1} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta) Pr(X_1 \leq \eta) \end{aligned}$$

We want now show that we can get a similar formula for $\tilde{k} = k + 1$. By conditioning on one more data point, the one-step memory approximation is extended to:

$$\prod_{j=k}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_{j-k+1} \leq \eta)$$
$$\approx \prod_{j=k+1}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_{j-k} \leq \eta) Pr(X_k \leq \eta | X_{k-1} \leq \eta, ..., X_1 \leq \eta)$$

So,

$$
\begin{aligned}
P(\eta) \approx & P_{\tilde{k}} \\
=& P_{k+1}(\eta) \\
=& \prod_{j=k+1}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_{j-k} \leq \eta) Pr(X_k \leq \eta | X_{k-1} \leq \eta, ..., X_1 \leq \eta) \\
& \prod_{j=2}^{k-1} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta) Pr(X_1 \leq \eta) \\
=& \prod_{j=k+1}^{N} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_{j-k} \leq \eta) \\
& \times \prod_{j=2}^{k} Pr(X_j \leq \eta | X_{j-1} \leq \eta, ..., X_1 \leq \eta) Pr(X_1 \leq \eta)
\end{aligned}
$$

With this construction, we can easily see that $P(\eta) = P_N(\eta)$.

Now, we will have a look on this sequence to find a relation between them and so a technique to get them.
Let $\alpha_{kj}(\eta) = Pr(X_j > \eta | X_{j-1} \leq \eta, ..., X_{j-k+1} \leq \eta)$ for $2 \leq k \leq j$.
α_{kj} denotes the exceedance probability conditional on k - 1 previous nonexceedances that means the probability that the j^{th} is bigger than η given that all the previous values are smaller.
$\sum_{j=k}^{N} \alpha_{kj}(\eta)$ is the expected effective number of independent exceedance events provided by conditioning on k - 1 previous observations.
With this notation, we have $\alpha_{1j}(\eta) = Pr(X_j > \eta), j = 1, ..., N$ and so $P_1(\eta) = \prod_{j=1}^{N}(1 - \alpha_{1j}(\eta))$.
$\sum_{j=1}^{N} \alpha_{1j}(\eta)$ represents the expected number of exceedances of the threshold η during the time interval (0,T).
Moreover we know that the α_{kj}'s are small enough when η is big to approximate $1 - \alpha_{kj}$ by $\exp(-\alpha_{kj})$ in using a Taylor expansion of the exponential function. So we have:

$$
P(\eta) \underset{\eta \to \infty}{\approx} F_1(\eta) = \exp(-\sum_{j=1}^{N} \alpha_{1j}(\eta))
$$

With the same process, we can approximate each member of the sequence $P_k(\eta)$

by:

$$P(\eta) \underset{\eta\to\infty}{\approx} F_k(\eta) = \exp(-\sum_{j=k}^{N} \alpha_{kj}(\eta) - -\sum_{j=1}^{k-1} \alpha_{jj}(\eta))$$

$$\text{and } F_k(\eta) \to P(\eta) \text{ as } k \to N$$

Then we reach the wanted result $F_N(\eta) = P(\eta)$ for $\eta \to \infty$.
This result is true under the simple assumption that : there is a value $\hat{k}$ strictly inferior to N such that $F_{\hat{k}}(\eta) = F_N(\eta)$. That means there exists a value of k such that the fact to add one step memory to the approximation will not really change the result of the function F.
According to the previous assumption about a cut-off value, for k big enough, $\sum_{j=1}^{k-1} \alpha_{jj}(\eta)$ is negligible compared to $\sum_{j=k}^{N} \alpha_{kj}(\eta)$.
Consequently, we will simplify our function and get for stationary and nonstationary data:

$$F_k(\eta) = \exp(-\sum_{j=k}^{N} \alpha_{kj}(\eta)),\ k \leq 1 \tag{3.5}$$

3.2.2 The ACER functions

Now, we can introduce the average conditional exceedance rate (ACER) functions of order k as follows:

$$\varepsilon_k(\eta) = \frac{1}{N-k+1} \sum_{j=k}^{N} \alpha_{kj}(\eta),\ k = 1, 2, ... \tag{3.6}$$

where N is the number of data points. In practice, the process Z(t) can be either stationary or ergodic. If it is stationary, the unconditional joint probability distribution does not change when shifted in time. If it is ergodic, then its statistical properties can be deduced from a single, sufficiently long, random sample of the process. In fact we can see Z(t) as a process whose the variation in time of its parameters can me modelled as an ergodic process.
For the following part, we will assume that Z(t) follows an ergodic, to model long-term statistics.
Nevertheless, for both scenarios, the empirical estimation of the ACER function $\varepsilon_k(\eta)$ follows the same scheme. We start by counting the number of events satisfying the condition, that means a value larger than η followed by the at least the good number of values smaller than the threshold. Then, we just have to divide this amount by $N - k + 1 \approx N$ when $k \ll N$. One can show that this technique is working for the long-term situation.

We will develop a bit more the numerical estimation of the ACER functions. First, we introduce two functions which will make the description of our problem easier.

$$A_{kj}(\eta) = \mathbf{1}\{X_j > \eta, X_{j-1} \leq \eta, ..., X_{j-k+1} \leq \eta\}$$
$$B_{kj}(\eta) = \mathbf{1}\{X_{j-1} \leq \eta, ..., X_{j-k+1} \leq \eta\}$$

where $j = k, ..., N$ and $k = 2, ...$
$\mathbf{1}$ is the indicator function and can be defined by:

$$\mathbf{1}(C) = \begin{cases} 1, \text{ if } C \text{ is true} \\ 0, \text{ otherwise} \end{cases}$$

Then,

$$\alpha_{kj}(\eta) = \frac{\mathbb{E}[A_{kj}(\eta)]}{\mathbb{E}[B_{kj}(\eta)]}, j = k, ..., N, k = 2, ... \tag{3.7}$$

where $\mathbb{E}$ denotes the expected value.
If we assume that the process is ergodic, then by definition $\varepsilon_k(\eta) = \alpha_{kk}(\eta) = ... = \alpha_{kN}(\eta)$ and so we get:

$$\begin{aligned} \varepsilon_k(\eta) &= \frac{\mathbb{E}[A_{kj}]}{\mathbb{E}[B_{kj}]} \\ &= \lim_{N\to\infty} \frac{\sum_{j=k}^{N} a_{kj}(\eta)}{\sum_{j=k}^{N} b_{kj}(\eta)} \end{aligned}$$

where $j > k$, and $a_{kj}(\eta)$ (respectively $b_{kj}(\eta)$) are the realized values of $A_{kj}(\eta)$ (respectively $B_{kj}(\eta)$).
Clearly, by definition, we have: $\lim_{\eta\to\infty} \mathbb{E}[B_{kj}(\eta)] = 1$.
Let

$$\tilde{\varepsilon}_k(\eta) = \frac{\sum_{j=k}^{N} \mathbb{E}[A_{kj}(\eta)]}{N - k + 1} \tag{3.8}$$

Then we have: $\lim_{\eta\to\infty} \frac{\tilde{\varepsilon}_k(\eta)}{\varepsilon_k(\eta)} = 1$.
These new functions approximate well the ACER functions at the extreme levels and are easier to use. Thus, as we focus on extreme levels (η), we can use these functions.

We can recognize that:

$$\begin{aligned} P(\eta) &\approx \exp(-\sum_{j=k}^{N} \alpha_{kj}(\eta)) \\ &= \exp(-\sum_{j=k}^{N} \frac{\mathbb{E}[A_{kj}(\eta)]}{\mathbb{E}[B_{kj}(\eta)]} \\ &\underset{\eta \to \infty}{\simeq} \exp(-\sum_{j=k}^{N} \mathbb{E}[A_{kj}(\eta)]) \end{aligned}$$

We can draw a last approximation if the time series can be segmented into K blocks where $\mathbb{E}[A_{kj}(\eta)]$ remains constant within each block and $\sum_{j \in C_i} \mathbb{E}[A_{kj}(\eta)] \approx \sum_{j \in C_i} a_{kj}(\eta)$ for a sufficient range of η-values. C_i denotes the set of indices for the block number i.
Consequently, we have $\sum_{j=k}^{N} \mathbb{E}[A_{kj}(\eta)] \approx \sum_{j=k}^{N} a_{kj}(\eta)$ because this approximation hold on each block and so on the union of the blocks. Finally, we reach this final relation:

$$P(\eta) \approx \exp(-(N-k+1)\hat{\varepsilon}_k(\eta)) \text{ for large enough } \eta \tag{3.9}$$

where $\hat{\varepsilon}_k(\eta) = \frac{1}{N-k+1} \sum_{j=k}^{N} a_{kj}(\eta)$.

3.2.3 The confidence intervals of the ACER functions

In this part, we will show three different techniques to get the 95% confidence interval of $\tilde{\varepsilon}_k(\eta)$ assuming a stationary time series. For all the parts contening information about time series, we used the following references : [24], [9].

Segmentation and normal distribution

Here, we assume that we can get R sets of realizations directly if it is available or by segmenting our long realization into R subseries. Then we can estimate the standard deviation by a classical technique:

$$\hat{s}_k(\eta)^2 = \frac{1}{R-1} \sum_{r=1}^{R} (\hat{\varepsilon}_k^{(r)}(\eta) - \hat{\varepsilon}_k(\eta))^2 \tag{3.10}$$

where $\hat{\varepsilon}_k^{(r)}(\eta)$ denotes the ACER function estimate from realization number r and $\hat{\varepsilon}_k(\eta) = \sum_{r=1}^{R} \hat{\varepsilon}_k(\eta)/R$ is the mean of the ACER function estimates.
If we assume also that for R big enough the realizations are independent, then

thanks to Eq.(3.10), we get the following 95% confidence interval $CI = (C^-(\eta), C^+(\eta))$ for the value $\varepsilon_k(\eta)$ where:

$$C^{\pm} = \hat{\varepsilon}_k(\eta) \pm \frac{1.96\hat{s}_k(\eta)}{\sqrt{R}} \tag{3.11}$$

where 1.96 corresponds to the value of the normal distribution to get 95% of the curve. Indeed, we have because of the symmetry of the normal distribution:

$$Pr(-1.96 \leq z \leq 1.96) = 0.95$$

This technique requires strong assumptions with stationary of the serie and the independence of the realizations. It will be used if the data set can be easily segmented into independent realizations.

The idea of a Poisson process

The main idea of this technique, applicable to non-stationary series, is to assume that the conditional exceedances over a threshold η constitute a Poisson process, which can be non homogeneous. This process is a counting process characterised by:

- The number of occurrences in non-joint time intervals are independent
- On a small enough time interval, the number of occurrence is negligible.
- On a small time interval, the probability of an occurrence is directly proportional to the length of this interval.

Let $\hat{E}_k(\eta) = \frac{\sum_{j=k}^{N} A_{kj}(\eta)}{N-k+1}$ be an estimator of $\tilde{\varepsilon}_k(\eta)$. Then as $A_{kj}(\eta)$ follows a Poisson process, we have: $Var(A_{kj}(\eta)) = \mathbb{E}(A_{kj}(\eta))$. Thus,

$$\begin{aligned} Var(\hat{E}_k(\eta)) &= \frac{Var(\sum_{j=k}^{N} A_{kj}(\eta))}{N-k+1} \\ &= \frac{\mathbb{E}[\sum_{j=k}^{N} A_{kj}(\eta)]}{N-k+1} \\ &= \tilde{\varepsilon}_k(\eta) \end{aligned}$$

Finally, if we take η large enough, we have $\sum_{j=k}^{N} \mathbb{E}[A_{kj}(\eta)] \approx \sum_{j=k}^{N} a_{kj}(\eta)$ and so the following 95% confidence interval:

$$
\begin{aligned}
C^{\pm}(\eta) &= \hat{\varepsilon}_k(\eta)\left(1 \pm \frac{1.96}{\sqrt{N-k+1}\sqrt{Var[\hat{E}_k(\eta)]}}\right) \\
&= \hat{\varepsilon}_k(\eta)\left(1 \pm \frac{1.96}{\sqrt{N-k+1}\sqrt{\tilde{\varepsilon}_k(\eta)}}\right) \\
&\approx \hat{\varepsilon}_k(\eta)\left(1 \pm \frac{1.96}{\sqrt{N-k+1}\sqrt{\hat{\varepsilon}_k(\eta)}}\right)
\end{aligned}
$$

Again, we have a quite strong assumption with the exceedances following a Poisson process.

Bootstrapping

The bootstrapping technique was published by Bradley Efron in ”Bootstrap methods: another look at the jackknife” in 1979. The main idea is to build a random sampling with replacement from the empirical distribution function of the observed data. [10]
This method is simple, stable (because a modification of the initial data will bring the same change on the bootstrap vector), asymptotically accurate and really efficient for unknown distributions. The major issue we can face with this method is the need to assume that the samples are independent.

Now, we will show simple bootstrap algorithm used to get a confidence interval for a statistics.
The first task is to create a data sample big enough and take the interesting interval. In fact, we will take a random numbers of realizations and create the sampling with them. In our case, we are working with replacement so the same value can appear more than once. Then we have to sort the sample and only keep the wanted values. For example, for a 95% confidence interval and a sorted sample X of length 1,000, the confidence interval will be the 25^{th} value of X and the 975^{th} one: $CI_{95\%}^{1,000} = (X[25], X[975])$.

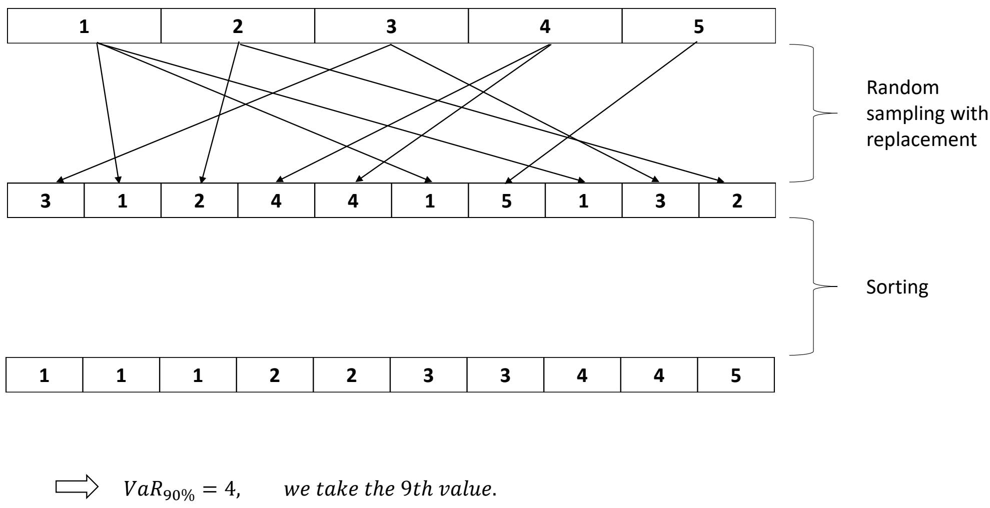

Figure 3.5: Scheme of a bootstrapping with replacement

3.2.4 Fitting to the Asymptotic Gumbel distribution

Up to now, we have seen that the ACER functions allow us to use all the data and not just the asymptotic ones. This should be a true strength to improve the accuracy of the result and to avoid us to prove that the observed extremes are asymptotic.
In this part, we will assume that the behaviour of the mean exceedance rate in the tail and more precisely the ACER functions can be modelled by a Gumbel type distribution that means:

$$\varepsilon_k(\eta) = q_k(\eta)\exp(-a_k(\eta - b_k)^{c_k})\text{ , with } \eta \geq \eta_1 \tag{3.12}$$

where the function q_k is slowly varying compared with the exponential function and a_k, b_k and c_k are suitable constants. In general all these constants are depending on k. η_1 is a tail marker and permits us to work inside the extreme value theory. Equation (3.12) becomes :

$$-\log|\log(\frac{\varepsilon_k(\eta)}{q_k(\eta)})| = -c_k\log(\eta - b_k) - \log(a_k)$$

And so we should have a perfectly linear tail behaviour for the plot $-\log|\log(\frac{\varepsilon_k(\eta)}{q_k(\eta)})|$ versus $\log(\eta - b_k)$.

In general, $q_k(\eta)$ is not a constant but it varies really slowly in the tail region and so may be replace by a constant. Sometimes to bring more credit to this assumption, we will just modify η_1 the tail marker to reach a better area. Then, we get our final equation:

$$\varepsilon_k(\eta) = q_k \exp(-a_k(\eta - b_k)^{c_k}) \text{ with } \eta \geq \eta_1 \tag{3.13}$$

Now, we will work for a chosen k so we will not write the index for each function or constant. We want to find the values of a, b, c and q.
To do that the easiest way is to minimize a mean square error function, with respect to all four arguments. We add a weight factor w_j to give more or less importance to our values. In our case, we want to give more importance to an reliable $\varepsilon(\eta)$ that means one which has a small confidence interval. So, the weights will permit us to fit with more accuracy the first values of the regular part of the curve.
The function to minimize is:

$$F(a, b, c, q) = \sum_{j=1}^{J} w_j |\log(\hat{\varepsilon}(\eta_j) - \log(q) + a(\eta_j - b)^c|^2 \tag{3.14}$$

, where $\eta_1 < ... < \eta_J$ denotes the levels where the ACER function has been estimated.
The new question should be how to find the weight factors w_j. This choice is arbitrary. For this paper, we took $w_j = [\log(C^+(\eta_j) - \log(C^-(\eta_j))]^{-2}$. First, the function logarithm permits us to create larger difference of value between the estimates because these values are positive and inferior to 1.

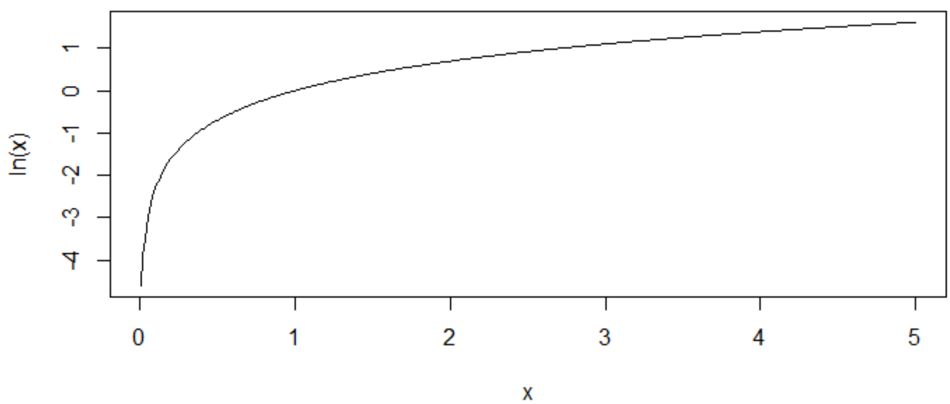

Figure 3.6: Logarithm function

Then, the reverse function creates the behaviour we want: if the length of the confidence interval is large, then the weight is small. Finally the square function makes the value positive.
Nevertheless, the use of the logarithm function can add some issues. Indeed, we cannot have C^- negative. One solution can be to stop the summation just before having a negative C^-. Another one can be to use the bootstrapping. This method works only on the realizations themselves. In finance, we cannot have an asset with a negative value. Thus, with the bootstrapping method, if the data values are positive, we can only get:

$$C^- \geq 0$$

Finally, we need to give a set of constraints for the optimization problem. We know that $0 < \hat{\varepsilon}_k(\eta_i) < 1$ then according to the approximation of $\hat{\varepsilon}(\eta_i) = q\exp(-a(\eta_i - b)^c)$, we can get the first two constraints: $q > 0$ and $\log(q) - a(\eta_i - b)^c \leq 0$. The second one will not be used in our problem because it is always true.
Then, b should be inferior to η_1 to be able to apply the function power by c and superior to $\min_j X_j$ to stay inside our set of realizations. Moreover, the function $\hat{\varepsilon}_k(\eta_i)$ should be decreasing in function of η (for η big enough). Indeed, it will be harder to find an exceedance when we increase the threshold. Thus, $a > 0$. For c we will give an arbitrary constraint : $0 < c < 5$.
To conclude, we have the following set of constraints:

$$\begin{aligned} 0 < q &< \infty \\ \min_j X_j < b &\leq \eta_1 \\ 0 < a &< \infty \\ 0 < c &< 5 \end{aligned}$$

We will then use an optimization algorithm. To improve the robustness of the results, we may apply a non-linearly constrained optimization. We choose to work with the weighted Levenberg-Marquardt (LM) algorithm discovered by Kenneth Levenberg,then published by Donald Marquardt.
The LM algorithm is an optimization method between the gradient descent algorithm and the Gauß-Newton one. Indeed, a weak value of the parameter makes the algorithm looking like the Gauß-Newton one, and so a large one will transform the algorithm to one similar to the gradient method. This algorithm is an iterative one.

First, we should rewrite the problem to make the writing of the routine easier.

$$\begin{aligned} F(a,b,c,q) &= \sum_{j=1}^{J} w_j |\log(\hat{\varepsilon}(\eta_j) - \log(q) + a(\eta_j - b)^c|^2 \\ &= \sum_i w_i |Y_i - f(\theta, X_i)|^2 \end{aligned}$$

where $Y_i = \log(\hat{\varepsilon}(\eta_i), \theta = (a,b,c,q)$ and $X_i = \eta_i$.
Now we can describe the routine:

$$\theta_{j+1} = \theta_j - (J^T J + D_\lambda)^{-1} J^T (Y - f(\theta, X_i)) \tag{3.15}$$

where J is the Jacobian matrix of f respecting to θ and D_{lambda} is a diagonal matrix permitting to adjust the importance of each iteration.

In the end of this part, we will show one technique to simplify the optimization problem.
When c is close to 1 we have approximately $F(a,b,q) = \sum_{j=1}^{J} w_j |\log(\hat{\varepsilon}(\eta_j) - \log(q) + a(\eta_j - b)|^2$, that is $F(a,b,c) = UU^T$ where $U_j = \sqrt{w_j}|\log(\hat{\varepsilon}(\eta_j) - \log(q) + a(\eta_j - b)|$. If we take small differences between the levels, we can see that the columns of U are correlated, so our optimization problem is ill-conditioned or close to. There is so an infinity of (b, q) values that gives the same value of F. We have at least one useless parameter. We can solve that by choosing $q = 1$. In our following studies, the values will not be so close so we can find an optimal value for q.
To make the optimization problem easier, we can fix first b and c. The problem is then reduced to a standard weighted linear regression problem.
Let $x_j = (\eta_j - b)^c$, $y_j = \log(\hat{\varepsilon}_j(\eta_j)$ and b and c be fixed
Then,

$$F(a,b,c,q) = \sum_j w_j |y_j - \log(q) + a x_j|^2 \tag{3.16}$$

The function F is regular enough to allow us to deal with the gradient to find the minima. So we want to find a^* and $\log(q))^*$ such as

$$\begin{cases} \frac{\partial F(a^*, \log(q)^*)}{\partial a^*} & = 2\sum_j w_j x_j |y_j - \log(q)^* + a^* x_j| = 0 \\ \frac{\partial F(a^*, \log(q)^*)}{\partial \log(q)^*} & = -2\sum_j w_j |y_j - \log(q)^* + a^* x_j| = 0 \end{cases}$$

As $0 < q < \infty$, the second equation becomes,

$$\begin{aligned} \sum_i w_i y_i + a^* \sum_i w_i x_i &= \log(q)^* \sum_i w_i \\ \log(q)^* &= \bar{y} + a^* \bar{x} \end{aligned}$$

where $\bar{x} = \frac{\sum_i w_i x_i}{\sum_i w_i}$, $\bar{y} = \frac{\sum_i w_i y_i}{\sum_i w_i}$.
Then we introduce $\log(q)^*$ inside the first equation and get:

$$\sum_j w_j x_j |y_j - \bar{y} - a^*\bar{x} + a^* x_j| = 0$$
$$\implies \sum_j w_j x_j |y_j - \bar{y} - a^*\bar{x} + a^* x_j| - \sum_j w_j \bar{x} |y_j - \bar{y} - a^*\bar{x} + a^* x_j|$$
$$+ \sum_j w_j \bar{x} |y_j - \bar{y} - a^*\bar{x} + a^* x_j| = 0$$
$$\implies \sum_j w_j (x_j - \bar{x}) |y_j - \bar{y} - a^*\bar{x} + a^* x_j| + \bar{x} \sum_j w_j |y_j - \bar{y} - a^*\bar{x} + a^* x_j| = 0$$

$\sum_j w_j (x_j - \bar{x}) |y_j - \bar{y} - a^*\bar{x} + a^* x_j| = 0$, because of the second equation.

$$\implies a^* = \frac{\sum_i w_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i (x_i - \hat{x})^2}$$

To conclude we have:

$$a^*(b, c) = \frac{\sum_i w_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i (x_i - \hat{x})^2} \tag{3.17}$$
$$\log(q)^* = \bar{y} + a^* \bar{x} \tag{3.18}$$

Finally, knowing a^* and $\log(q)^*$, we have reduced our optimization problem from four variables to two variables: b and c. Thus, we will apply an optimization algorithm, the Leverberg-Marquardt algorithm for example, on the function $\tilde{F}(b, c) = F(a^*(b, c), b, c, \log(q)^*)$ to get the optimal values b^* and c^*. Then, we can calculate $a^*(b^*, c^*)$ and $\log(q(b^*, c^*))^*$.

3.2.5 Fitting to a GEV distribution

To introduce this part, we remind the GEV distribution formula:

$$G(z) = \exp[-(1 + \xi \frac{z - \mu}{\sigma})^{\frac{-1}{\xi}}], \text{ defined on } \{z | 1 + \xi \frac{z - \mu}{\sigma} > 0\}. \tag{3.19}$$

Then the ACER function $\varepsilon_1(\eta)$ can be expressed asymptotically as

$$\varepsilon_1(\eta) \underset{\eta \to \infty}{\simeq} [1 + \xi(a(\eta - b))]^{\frac{-1}{\xi}} \tag{3.20}$$

because we are assuming that for a level large enough and independent data, ε_1 follows a GEV distribution.

Now, we want to extrapolate this result to the whole set of ACER functions. Thus, we use a similar type of approximation for the behaviour of the mean exceedance rate in the tail. This part of the curve is assumed to follow a function of the form $[1+\xi a(\eta-b)^c]^{\frac{-1}{\xi}}$ where $a>0$, $\eta \geq \eta_1 \geq b$, $c>0$ and $\xi>0$ are suitable constants and η_1 is again a chosen tail level.
Then, with these assumptions we get:

$$\varepsilon_k(\eta) = q_k(\eta)[1+\xi_k(a_k(\eta-b_k)^{c_k})]^{\frac{-1}{\xi_k}}, \eta \geq \eta_1 \tag{3.21}$$

Again, in this case, $q_k(\eta)$ is weakly varying compared with the function $[1+\xi_k(a_k(\eta-b_k)^{c_k})]^{\frac{-1}{\xi_k}}$ and a_k, b_k, ξ_k are suitable constants depending on k. In general, $q_k(\eta)$ is not a constant but it is possible to find a marker tail η_1 such that for bigger levels, its variations will be negligible.
From now on, we will work with a specific value of k and so we will write ξ, a, b, c, q instead of $\xi_k, a_k, b_k, c_k, q_k$. The studied formula is so:

$$\varepsilon(\eta) = q[1+\tilde{a}(\eta-b)^c]^{-\gamma} \tag{3.22}$$

where $\tilde{a} = a\xi$ and $\gamma = \frac{1}{\xi}$
We have so a new expression for the mean square error function that we want to minimize:

$$F(a,b,c,q,\xi) = \sum_i w_i|\log(\hat{\varepsilon}(\eta_i)) - \log(q) + \frac{1}{\xi}\log(1+\xi a(\eta_i-b)^c))|^2 \tag{3.23}$$

where w_i is a weighted factor previously defined. The set of constraints is:

$$\begin{aligned} 0 < q &< \infty \\ \min_j X_j < b &\leq \eta_1 \\ 0 < a &< \infty \\ 0 < c < 5\xi &> 0 \end{aligned}$$

Again, we can directly minimized this function in using the Levenberg-Marquardt lest squares optimization method. Nevertheless, we can reduce the number of variables up to 3 in using a trick similar to the one use for the Gumbel case.
Let $x_i = \log(1+\tilde{a}(\eta_i-b)^c)$ and $y_i = \log(\varepsilon(\eta_i))$.
The function F becomes:

$$F(\tilde{a},b,c,q,\gamma) = \sum_i |y_i - \log(q) + \gamma x_i|^2 \tag{3.24}$$

One can notice that, in fact, it is the equation (3.14) with γ instead of a. So we can find γ^* and $\log(q)^*$ when $\tilde{a}$, b and c are fixed.

$$\log(q)^*(\tilde{a}, b, c) = \bar{y} + \gamma^*(\tilde{a}, b, c)\bar{x} \tag{3.25}$$

$$\gamma^*(\tilde{a}, b, c) = -\frac{\sum_i w_i(x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i(x_i - \bar{x})^2} \tag{3.26}$$

where $\bar{x} = \frac{\sum_i w_i x_i}{\sum_i w_i}$ and $\bar{y} = \frac{\sum_i w_i y_i}{\sum_i w_i}$.
Then we will use the Levenberg-Marquardt algorithm on the function $\tilde{F}(\tilde{a}, b, c)$ to get $\tilde{a}^*, b^*, c^*$. With these values we can calculate the two last variables: γ^* and $\log(q)^*$.

3.2.6 How to choose η_1 and get the VaR

First, we will deal with the choice of η_1. This value can be directed read on the log plot $(\eta, \log(\hat{\varepsilon}_k(\eta)))$. The tail marker corresponds to the beginning of the regular tail behaviour. To illustrate that, we use the Pareto distribution with the parameter $\beta = 3$. With this example, we can see that a tail marker of 3.5 permits us to be in

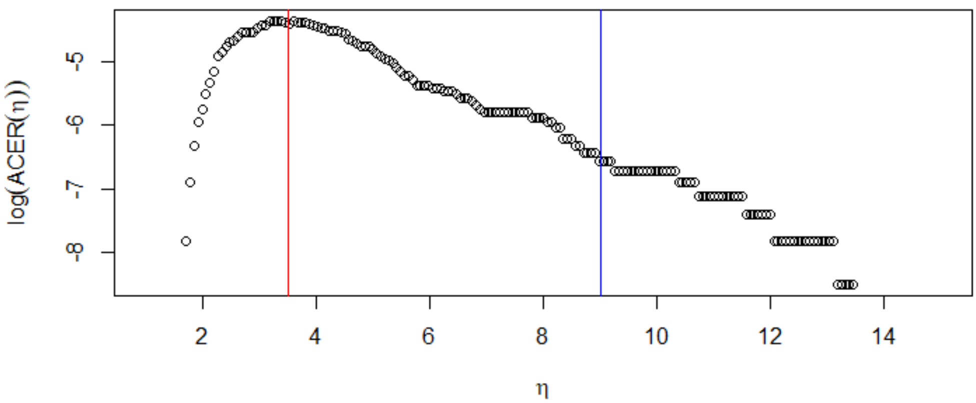

Figure 3.7: Epsilon in function of the threshold

regular tail behaviour. In fact a value of 4 can be acceptable. Moreover we can see that to stay on the regular tail, we must choose a end marker. Indeed, because of the value of k, the ACER function will be a strong piecewise function and so this shape can make our fit less accurate. Here we could take 9 for end marker.

Secondly, we want to rely this approximation of ε to the calculation of the VaR. For the Gumbel case, we have:

$$\varepsilon(\eta) = q\exp(-a(\eta-b)^c), \eta \geq \eta_1$$

and,

$$P(\eta) = P(M_n \leq \eta) \approx \exp(-N\hat{\varepsilon}_k(\eta))$$

The Value at Risk at $\alpha\%$ will be equal to η such that: $P(\eta) = 1-\alpha$.
Then,

$$\begin{aligned}
P(\eta) &\approx \exp(-N\hat{\varepsilon}_k(\eta)) = 1-\alpha \\
&\Rightarrow \hat{\varepsilon}_k(\eta) \approx \frac{-\log(1-\alpha)}{N} \\
&\Rightarrow q\exp^{-a(\eta-b)^c} \approx \frac{-\log(1-\alpha)}{N} \\
&\Rightarrow a(\eta-b)^c \approx -\log(\frac{-\log(1-\alpha)}{Nq}) \\
&\Rightarrow \eta \approx b + (-\frac{1}{a}\log(\frac{-\log(01-\alpha)}{Nq}))^{\frac{1}{c}}
\end{aligned}$$

One can notice that $\log(1-\alpha) \approx \alpha$ for α small enough by Taylor expansion.

For the general case, we have:

$$\varepsilon(\eta) = q[1+\tilde{a}(\eta-b)^c]^{-\gamma}, \eta \geq \eta_1$$

and,

$$P(\eta) = P(M_n \leq \eta) \approx \exp(-N\hat{\varepsilon}_k(\eta))$$

The Value at Risk at $\alpha\%$ will be equal to η such that: $P(\eta) = 1-\alpha$.
Then,

$$\begin{aligned}
P(\eta) &\approx \exp(-N\hat{\varepsilon}_k(\eta)) = 1-\alpha \\
&\Rightarrow \hat{\varepsilon}_k(\eta) \approx \frac{-\log(1-\alpha)}{N} \\
&\Rightarrow q[1+\tilde{a}(\eta-b)^c]^{-\gamma} \approx \frac{-\log(1-\alpha)}{N} \\
&\Rightarrow \tilde{a}(\eta-b)^c \approx (\frac{-\log(1-\alpha)}{Nq})^{-\frac{1}{\gamma}} - 1 \\
&\Rightarrow \eta \approx b + (-\frac{1}{\tilde{a}}(\frac{-\log(1-\alpha)}{Nq})^{\frac{-1}{\gamma}} - 1)^{\frac{1}{c}}
\end{aligned}$$

To sum up, we have:

- For the Gumbel case, $\mathrm{Va}R_\alpha \approx b + (-\frac{1}{a}\log(\frac{\alpha}{Nq}))^{\frac{1}{c}}$.
- For the general case, $\mathrm{Va}R_\alpha \approx b + (-\frac{1}{\tilde{a}}(\frac{\alpha}{Nq})^{\frac{-1}{\gamma}} - 1)^{\frac{1}{c}}$.

Here $\log$ represents the natural logarithm that means the inverse of the exponential function.

3.3 The R package

Now, in this section, we will describe all the functions used to create our R package. [2] [5] The explanations about how to create a R package can be found in appendix. According to the last section, the sketch of our main function should be:

Get the peaks over threshold and calculate the ε

$\downarrow$

Get the confidence interval

$\downarrow$

Optimize the problem and return the VaR

3.3.1 Calculation of Epsilon

Let X be the data vector.
First, we have to count the number of exceedances the means the matrix A to get the value $\sum a_{ij}$, fundamental to calculate the ε.
The initial idea was to build directly the matrix A. Thus, we creates
$building\_matrices$. This function has the data and the level to test as inputs and returns the matrix A. To proceed, we choose a k and we try to test for each j such that $k \leq j \leq N$, where N is the length of the data vector. The matrix A is initialized to $0_{N\times N}$ and each time we have $X[j] >$ level and $X[i] \leq$ level for $i = j-k+1, ..., j-1$ we modify the value of $A[k-1, j-1]$ to 1. We can remark

that we increment $A[k-1, j-1]$ and not $A[k, j]$ because in the true case k will start at 2 and it is easier to start the matrix to 1 (small gain of space too).
The major problem of this method is the need of two loops with several comparisons. Moreover, to get the expected value we have to take the sum of each lines of the matrix. All these defaults make our function really slow.

The first idea to improve this calculation time was to avoid to do the last sum and doing that directly inside the loop. Besides, we don't need to create and use a matrix but just the final vector. Instead of modifying the value of $A[k-1, j-1]$ we will just increment of 1 the value of $a[k-1]$ where $a[k-1]$ represents the sum of the $k-1^{th}$ line of A.
This method sum_peak works because in fact we do not need the full matrix but only the sum of its lines.

Even with this trick, the counting of the peak over threshold was kind of slow. The last thing to do to improve that was to try to avoid as much as possible the number of loops. Indeed, especially in finance, we will work with data vector of at least 300/400 values. Having a calculation time of order N^2 like the two previous functions is not an option.
Consequently, our last chance to make our project doable was to reduce this calculation time to an order of N that means a linear calculation time. To do that, we needed to change completely our way of thinking the problem. Instead of proceeding directly with calculation of the vector a, we will now work on the data vector X itself.
First, we use the R function $which(X > level)$ on the data. This function will return a vector with all the positions of X where the value is strictly bigger than level. Already implemented on R, this function is really fast.
Then we just have to work with a vector a lot smaller. For each value of the vector, we will subtract the previous one (or nothing for the first value) and increment all the values of the vector a, the counting value, up to the value of this difference. Let call the vector resulting from the which function by W. So for example, if $W[i] - W[i-1] = 7$, we will increase by 1 all the values of a up to 7 : $a[1:7] = a[1:7] + 1$. Indeed, if $W[i] - W[i-1] = 7$, that means that between $W[i-1]$ and $W[i]$, there are 6 values inferior to level. We can illustrate that with the following example where level = 4.
Let X be :

1	8	3	4	1	9	10	2	5

Then after the use of the function which with level equal to 4, we get:
W =

2	6	7	9

And we have directly:
a =

4	3	1	1	0	...

Consequently, our function responsible of that, denoted $sum\_validate\_data$, is a lot faster than the two previous ones. In the following chapter, we will show on one chart, that the convergence seems to be linear. It is difficult to know the complexity of this function because of the unknown one of $which$.

Finally, now we have the number of exceedances for each k and we just have to divide by $N - k + 2$, again because k starts at 2 but it is easier to put the value k=2 in the case one of our vector ε and so on. The function $epsilon\_app$ takes the result of $sum\_validate\_data$, the data vector to get its length and the maximal value of k. The output of this method is so the vector epsilon.

3.3.2 Approximation of the confidence interval of the Epsilon

The goal of this subsection is to get a dataframe (equivalent to a dynamic table in R) with two columns: one for the lower boundary and one for the upper one.

First, we implement the function a segmentation of the data and a normal distribution. This function, denoted CI_block, uses the data vector and the threshold as inputs.
To find a way to split our initial data vector, we try to fit the serie with an ARIMA model (described in annexe). Moreover, we use the function $auto.arima$ to get directly the values of the parameter and we force this function to find a seasonality in initializing D by 1. Then, we just save the value s of this seasonality to use it to get our segmentation. Each block of data will be of length s.
After that, we just apply for k inferior to s the function $epsilon\_app$. Finally, we calculate the variance of the distribution of ε modelled by a normal law:

$$\hat{s}_k(\eta)^2 = \frac{1}{R-1}\sum_{r=1}^{R}(\hat{\varepsilon}_k^{(r)}(\eta) - \hat{\varepsilon}_k(\eta))^2 \tag{3.27}$$

We just have to apply the following formula to get the confidence interval:

$$C^{\pm} = \hat{\varepsilon}_k(\eta) \pm \frac{1.96\hat{s}_k(\eta)}{\sqrt{R}} \tag{3.28}$$

The second used approach to get these confidence intervals, $CI\_Poisson$, is to consider the ε's distribution follows a Poisson process. Here, as seen in the previous section, the computation is really easy and we just have to get the ε thanks to the function $epsilon\_app$ and apply the following formula:

$$C^{\pm}(\eta) \approx \hat{\varepsilon}_k(\eta)\left(1 \pm \frac{1.96}{\sqrt{N-k+1}\sqrt{\hat{\varepsilon}_k(\eta)}}\right) \tag{3.29}$$

where $\hat{\varepsilon}_k(\eta)$ is the value of ε calculated with $epsilon\_app$ for a k and a level equal to η.

The third and last implemented method will be the most used and is based on the bootstrapping method.
This function, CI_boot, requires the data vector X, the level to test, the maximal number of steps back we test and the number of samplings to do as inputs.
The idea behind this technique is kind of simple. We will create a vector of dimension the number of samplings. Then we will apply the function $epsilon\_app$ on this vector obtained by using the sample R function with replacement. All the values of this vector are from the initial data but some initial values can not appear in this vector.
Then we will take the 5% percentile, that means :

$$
\begin{aligned}
C^- &= M[\text{floor}(0.025 * \text{number of samplings})] \\
C^+ &= M[\text{floor}(0.975 * \text{number of samplings}) + 1]
\end{aligned}
$$

We use the function floor in order to validate the following conditions.
If $0.025*$number of samplings is not an integer, we take the biggest integer smaller than it.
If $0.975*$number of samplings is not an integer, we take the smallest integer larger than it it.

3.3.3 Choice of k and η_1, η_f

These two constants are highly dependent on the data. More precisely, η_1 depends also on k but the opposite is not true. Nevertheless, we have to fix η_1 before choosing a good k.
First, when we increase the value of k, we translate the curve to the right. The safest configuration to choose η_1 is for a large k that is a curve translate to the right. Indeed, we may be less accurate but we will take no point outside the regular tail causing big fitting mistakes.

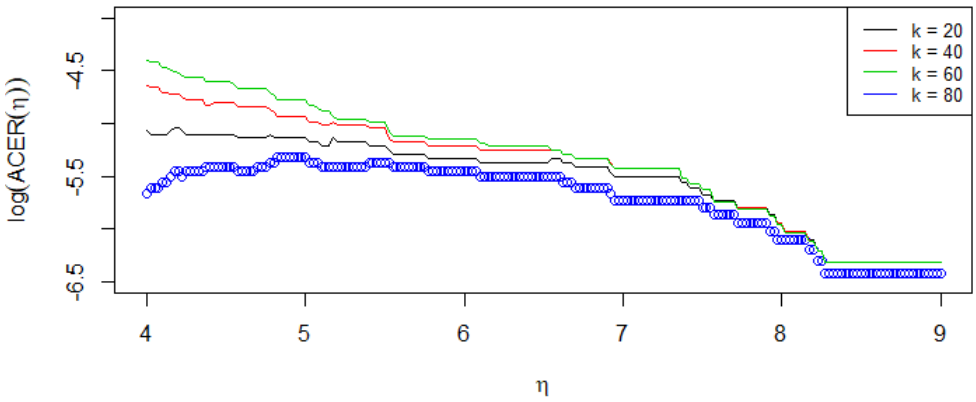

Figure 3.8: Choice of eta 1 and evolution for different k

We can see that the choice of mu1 is arbitrary. For example, with this data, η_1 equals to 4 seems to be a good choice.
At this point we can also decide to the value of η_f. Here, we have two choices:

- $\eta_f = 9$, This value permits us to take a maximum of data points without having a too piecewise function.

- $\eta_f = 7$, This value is the safest one. On the one hand with have a smaller level scale than the one with $\eta_f = 9$. On the other hand we have a function close to be fully regular.

To choose k, we need to plot the ACER function and take the smallest k such that the tail of the curve is the same (to a certain accuracy) as the one of a large k.

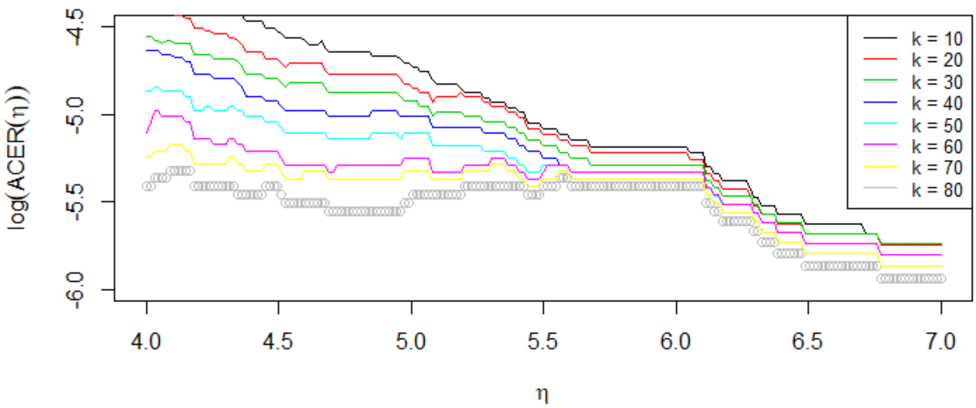

Figure 3.9: ACER functions for different k

As expected, all charts have a similar shape. Nevertheless, we can divide the results into four groups:

- k = 80, The plot for k = 80 is the reference one for this case (because it is the biggest value of k). But we will try to find a correct smaller k. Indeed, the smaller k is, the more data points we use.

- k = 70, The plot is really close to the one for k = 80. Nevertheless, the value of k remains big.

- k = 40, 50, 60, The tails of the curves for these k are the same. Consequently, if this accuracy is the one we choose we will take k = 40.

- k =10, 20, 30, The tails of the curves for these k are the same. But the curves seem to be too different from the one with k=80.

In this case, we should choose k = 40.

In this part, we saw that the choice of the three main parameters (k, η_1 and η_f) is arbitrary. However, we will try to build a function doing this human process but these three parameters will make our solution sensible. Indeed the fitting will be different and so the VaR too.

3.3.4 The optimization part and the calculation of the VaR

As seen in the previous section, we have two different ways of working on the optimization problem: the direct one by applying an algorithm on the complete

function and one where we simplify the problem before. [21]
Moreover, we want to model the tail of the ε functions in the Gumbel case and the general one.

The Gumbel case

We will build one function with 4 different possibilities for the optimization. First, this function will find η_1, η_f and k. Then we will decide how many levels we want to study for which k. For each level, we will calculate with our function the values of ε and the weights : $w_j = [\log(C^+(\eta_j)-\log(C^-(\eta_j)]^{-2}$. We remove from these vectors the values for the indexes where ε is equal to zero.
At this part we have two vectors of same length.

First, we will deal with the direct method. For this method, we can use two different optimization R functions: $constrOptim$ and $nls.lm$. The first function works well but can sometimes not be enough robust.
The implementation of the functions to optimize will be different for the two methods: $constrOptim$ requires a function returning a scalar and $nls.lm$ a vector. So for the scalar function, we use the formula:

$$result = \sum_i w_i |\log(\varepsilon_i) - \log(q) + a(\text{level}[i] - b)^c|^2$$

where a, b, c and q are the variables to optimize. Then we get the values of these variables with:

$$\theta \leftarrow constrOptim(c(1, 1, minit, 1.1), function, NULL, U, C)$$

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$C = (0, 0, \min(X), -\eta_1, 0, -5)$$

where $U \times \theta \leq C$ represents the constraints and $\theta = (q, a, b, c)$.
For the vector function, we use:

$$result[i] = \sqrt{w_i}|\log(\varepsilon_i) - \log(q) + a(\text{level}[i] - b)^c|$$

$$\theta \leftarrow nls.lm(c(1, 1, \frac{\min(X) + \eta_1}{2}, 1), lower = c(0, 0, \min(X), 0),$$
$$upper = c(Inf, Inf, \eta_1, 5), function)$$

The second approach to this problem is the use of some simplification before the optimization. For both of the optimization algorithms, we have to calculate outside the function to optimize $y = \log(\text{eps})$, where eps is the vector composed of the ε's values, and $\bar{y} = \frac{\sum_i w_i * y_i}{\sum_i w_i}$. Then inside the function to optimize we will proceed to the following computations:

$$\begin{aligned}
x &\leftarrow (\text{level} - b)^c \\
\bar{x} &\leftarrow \frac{\sum_i w_i * x_i}{\sum_i w_i} \\
a &\leftarrow -\frac{\sum_i w_i(x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i(x_i - \bar{x})^2} \\
q &\leftarrow \exp(\bar{y} + a\bar{x})
\end{aligned}$$

For the scalar function,

$$result \leftarrow \sum_i w_i(y - \log(q) + ax)^2$$

For the vector function,

$$result[i] \leftarrow \sqrt{(}w_i)|y[i] - \log(q) + a * x[i]|$$

Finally, according to the wanted optimization, we use $constrOptim$ or $nls.lm$ to get b and c. Now, we just have to calculate a and q:

$$\begin{aligned}
x &\leftarrow (\text{level} - b)^c \\
\bar{x} &\leftarrow \frac{\sum_i w_i * x_i}{\sum_i w_i} \\
a &\leftarrow -\frac{\sum_i w_i(x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i(x_i - \bar{x})^2} \\
q &\leftarrow \exp(\bar{y} + a\bar{x})
\end{aligned}$$

At this point, we model the tail of the $\log(\varepsilon)$ functions and we can get the VaR with $\text{Va}R_\alpha = b + (-\frac{1}{a}\log(\frac{\alpha}{Nq}))^{\frac{1}{c}}$ for α small enough.

The general case

The computation of the general case is similar to the one of the Gumbel case with the add of one variable ξ. The only modifications are on the functions to optimize and the inputs of the optimization functions.
For the direct case we will have now:

$$result \leftarrow \sum_i w_i |\log(\varepsilon_i) - \log(q) + \gamma \log(1 + \tilde{a}(\text{level}[i] - b)^c)|^2$$

$$\theta \leftarrow constrOptim(c(1, 1, \frac{\min(X) + \eta_1}{2}, 1, 1), function, NULL, U, C)$$

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$C = (0, 0, \min(X), -\eta_1, 0, -5, 0)$$

and

$$result[i] \leftarrow w[i] |\log(\varepsilon[i]) - \log(q) + \gamma \log(1 + \tilde{a}(\text{level}[i] - b)^c)|$$

$$\theta \leftarrow nls.lm(c(1, 1, \frac{\min(X) + \eta_1}{2}, 1, 1), lower = c(0.1, 1, \min(X), 0.1, 1),$$
$$upper = c(Inf, Inf, \eta_1, 5, Inf), function))$$

where $\theta = (q, \tilde{a}, b, c, \gamma)$ with $\gamma = 1/\xi$ and $\tilde{a} = \xi a$.

For the simplified case, we calculate first $y \leftarrow \log(eps)$ and $\hat{y} \leftarrow \frac{\sum_i w_i y_i}{\sum_i w_i}$. Then inside the function we have:

$$x \leftarrow 1 + \tilde{a}(\text{level} - b)^c$$
$$\bar{x} \leftarrow \frac{\sum_i w_i * x_i}{\sum_i w_i}$$
$$\gamma \leftarrow -\frac{\sum_i w_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i (x_i - \bar{x})^2}$$
$$q \leftarrow \exp(\bar{y} + \gamma \bar{x})$$

For the scalar function,

$$result \leftarrow \sum_i w_i (y - \log(q) + \gamma \log(x))^2$$

For the vector function,

$$result[i] \leftarrow \sqrt{(}w_i)|y[i] - \log(q) + \gamma \log(x[i])|$$

Then we use *constrOptim* or *nls.lm* to find b, c, $\tilde{a}$ and we calculate the value of γ and q:

$$\begin{aligned}
x &\leftarrow 1 + \tilde{a}(\text{level} - b)^c \\
\bar{x} &\leftarrow \frac{\sum_i w_i * x_i}{\sum_i w_i} \\
\gamma &\leftarrow -\frac{\sum_i w_i(x_i - \bar{x})(y_i - \bar{y})}{\sum_i w_i(x_i - \bar{x})^2} \\
q &\leftarrow \exp(\bar{y} + \gamma\bar{x})
\end{aligned}$$

So we have modelled the tail of the ε functions and we can get the VaR with $\text{Va}R_\alpha = b + (-\frac{1}{\tilde{a}}(\frac{\alpha}{Nq})^{\frac{-1}{\gamma}} - 1)^{\frac{1}{c}}$.

Remark. *One can notice that, in fact, we need only one function for each case. Indeed, if we sum the square of the values of the vector function's return we get the result of the scalar function.*

Chapter 4

Analyses of results

4.1 Execution time and accuracy

Comparison of the three functions to get the exceedances rate

We create three different functions in order to count the number of exceedances over a threshold. The first one should be the slowest one due to the building of a full matrix and then the sum of each line. The second one should be in the middle in term of execution time. We can see an illustration of this thought in the following plot:

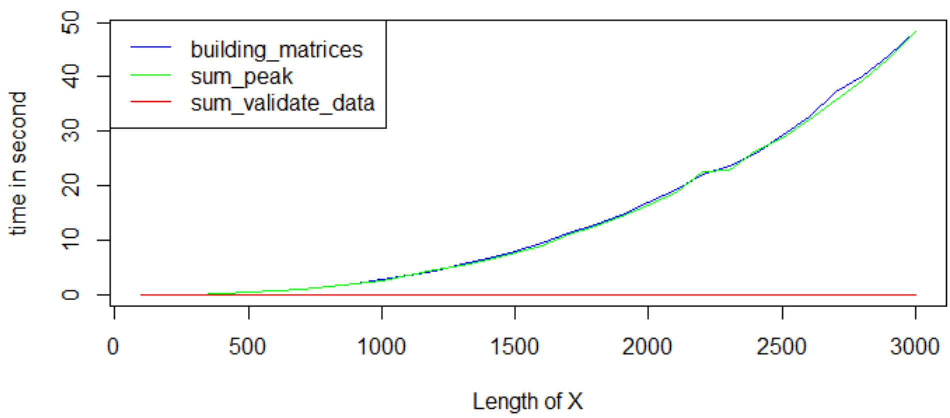

Figure 4.1: Execution times of the three methods in function of the number of data points

A bit surprising, the fact of avoiding the final sum is not really important to save some time. This may be a consequence of the use of the R function sum. Indeed, already implemented, this function is really fast.
However, we can see that working directly on the data and not on the result is a big improvement in term of execution time. The two first methods have an execution time following an exponential tend whereas the last one seems to be linear or even constant. In fact, as shown in the following chart, the execution time of the last method follows a linear evolution but its values are too small to be seen on the previous graph.

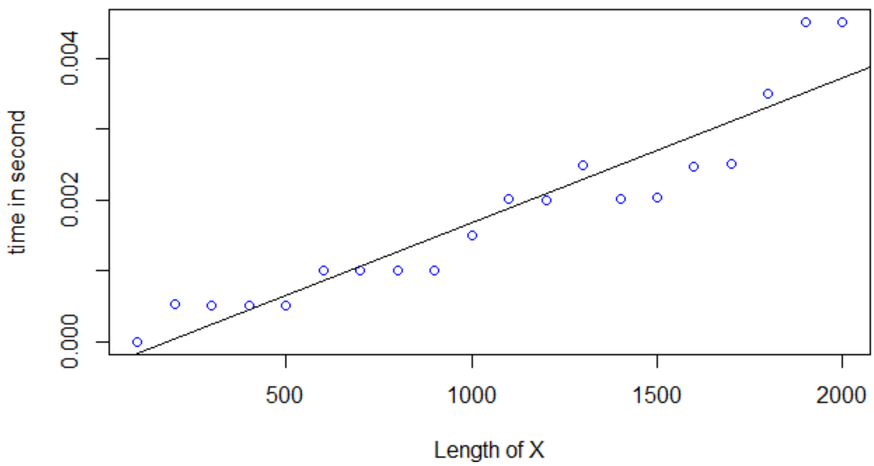

Figure 4.2: Execution time of the last method in function of the number of data points

To conclude this part, we can say that the implementation and the obtained results are coherent with our expectation in the sense that the last method is really more efficient and should be used.

Comparison of the CI functions

In this part, we have three functions to study in terms of rapidity and accuracy. To get the accuracy, we will specially focus on the size of the CI. To be accurate we need a CI interval as small as possible.
First, we plot the speed in function of the number of data points for the three methods.

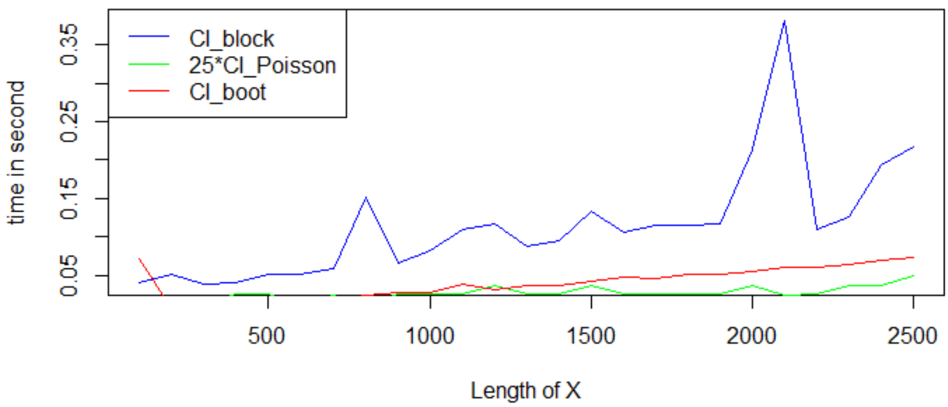

Figure 4.3: Time executions of the three methods in function of the number of data points

We can see that the first method using the segmentation and the normal distribution is less stable and slower than the other. Moreover the difficulty to get a segmentation value can be a big weakness.
Secondly, the second method is a lot faster than the two others. Nevertheless, the function using a percentile bootstrapping is relatively fast and could be used. Indeed, this method presents a big advantage with no negative value. We did a bootstrapping with 100 samples, this number can be reduced.
Now, in a second step, we will compare the accuracy of the Poisson and bootstrapping methods. To do that we will first plot the obtained values of ϵ.

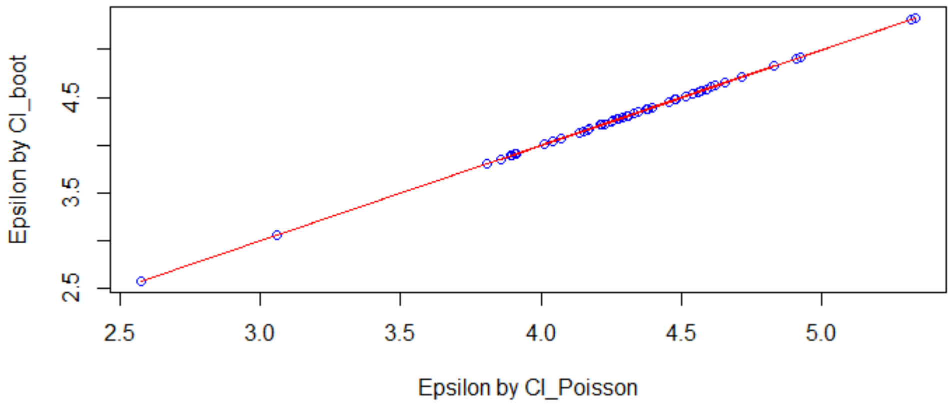

Figure 4.4: Epsilon values obtained with the two methods

The ε values obtained by the two methods are really close to each other. Consequently, we cannot make a choice based on that. So we need to compare the size of the CI.

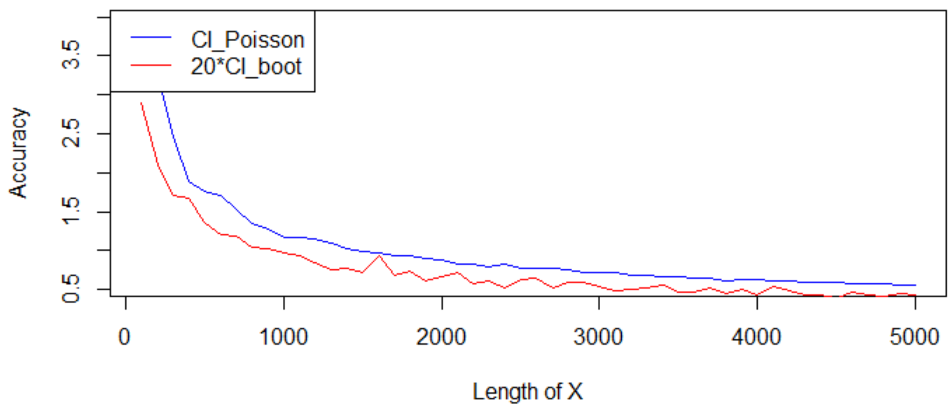

Figure 4.5: CI intervals in function of the number of data points for both methods.

And now we can notice that the bootstrapping method with the use of no negative values gives us a CI approximately twenty times smaller.

Now we need to choose a reasonable number of samples to get a stable CI and a certain rapidity.

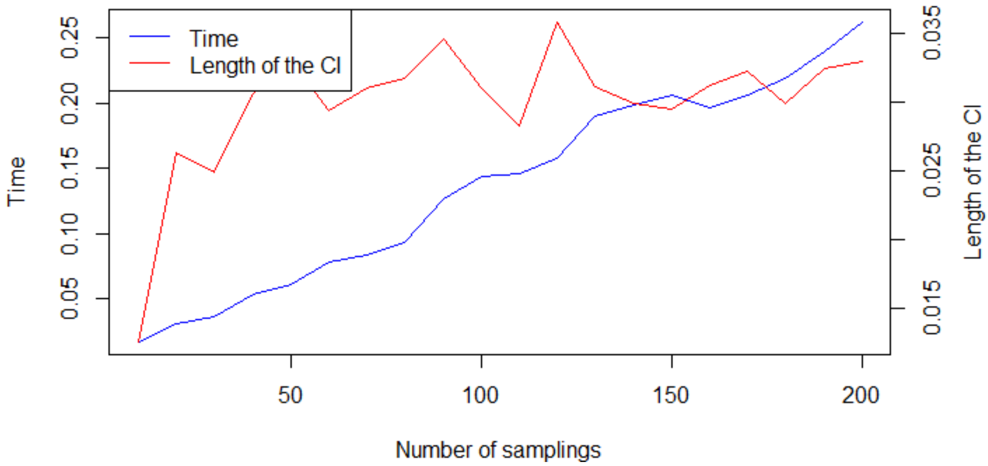

Figure 4.6: Time executions and accuracy of the bootstrapping method in function of the number of samples

A number of samples around 50 gives us a good stability for the size of the CI. In the following part, we will use 50 samples in order to improve the speed convergence. Indeed, this function will be called a lot of times.

Nevertheless, for our program, we will decide to use the $CI\_Poisson$ because we really need to accelerate the process. The weakness of this function is the possibility to get negative value for the lower bound of the confidence interval. But it can happen only for large levels and thanks to η_f we will not work in this part (the function is already too piecewise).

In the following parts, we will work with a Pareto distribution with $\beta = 3$ and a VaR at 1%. We will in the next section study this distribution with our optimal set up that the VaR should be equal to 10. The following parameters to set up have a similar behaviour for the Gumbel or the general case. Consequently we will only work with the Gumbel one.

The best optimization algorithm

First, we have to choose between the four different kinds of optimization : the use of a linear optimization algorithm with constraints or the Levenberg-Marquardt algorithm applied on the complete problem or a simplified version. [25]

So for each case, we will get the average execution time and accuracy. For the accuracy, we will fix the number of data points to 5000, the value of step back (k) to 20 and the precision to 0.05. Finally, we will calculate the sum square difference between the obtained $\log(\varepsilon)$ and the ones approximate by $\log(q)+(a(b-level)^c)$.

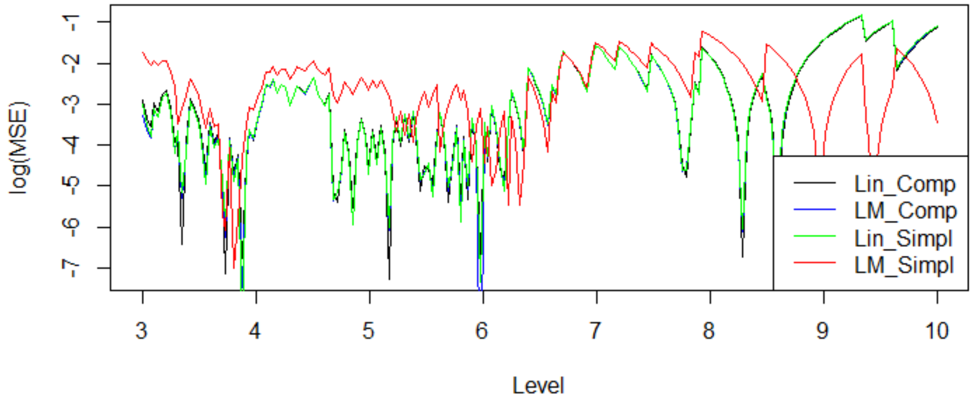

Figure 4.7: Error for the four optimization algorithms in function of the level

To get this plot, we choose to take $\eta_1 = 3$, $\eta_f = 10$, $k = 30$ and we calculate for the Gumbel case the natural logarithm of the Mean Square Error that means:

$$log(\text{MSE}(\eta)) = \sqrt{((\log(\varepsilon(\eta)) - \varepsilon_{\text{app}}(\eta))(\log(\varepsilon(\eta)) - \varepsilon_{\text{app}}(\eta)))} \tag{4.1}$$

where ε_{app} is the one calculated with the parameters found by the optimization. We can see that the shape of error is close to be the same for the three algorithms. Moreover these error are really small (order of 1%) for all algorithms. However, the Levenberg-Marquardt seems to be better for large level that means when we are at the end of the tail (a major part for our study). Consequently, we will choose tis algorithm. We get the following chart:

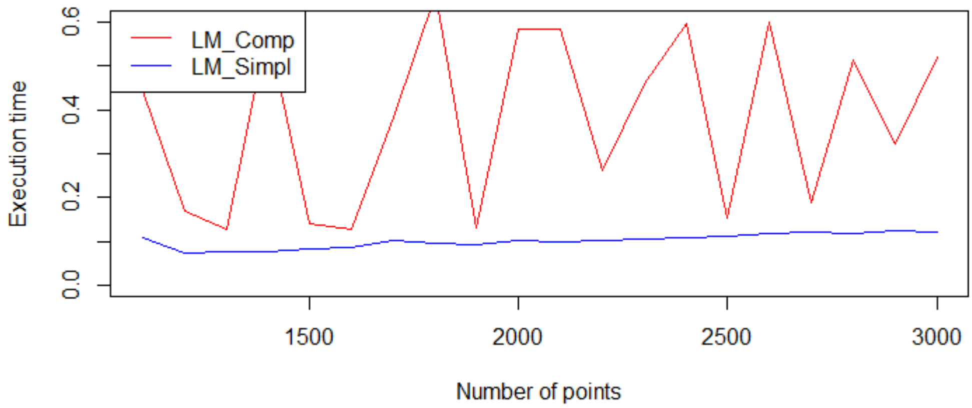

Figure 4.8: Execution time for the LM algorithm on the simplified and complete model

On this chart, we can clearly see that the LM algorithm is a lot more stable for the simplified case. As recommended by the paper published by A. Naess, O. Gaidai and O. Karpa, we will choose to work with the LM algorithm applied on this case.

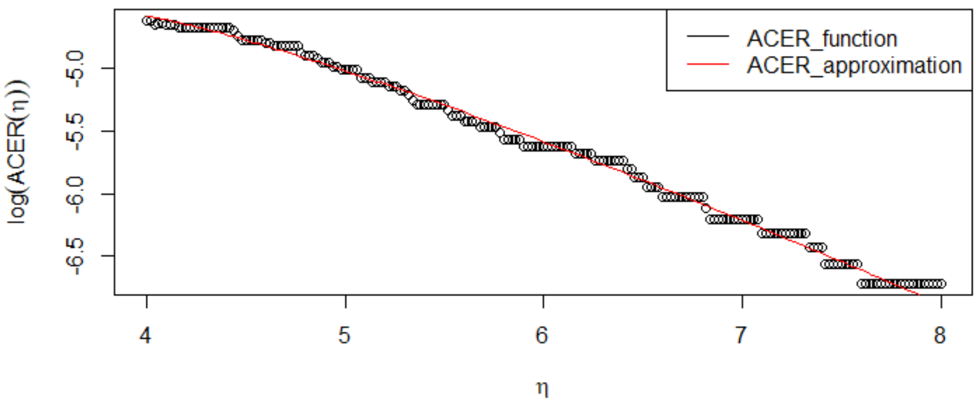

Figure 4.9: Calculated and approximate epsilons

We can see in the previous plot the efficiency of the fit on the tail. Indeed the red line is really close to the calculated ε in black. Besides, we can see that the fit is close to be perfect for the lowest level and become less good on the end of the tail. This behaviour is the one we wanted to create with the use of weights. Indeed, for lower levels, the values of ε are more accurate and have to be fitted as good as possible.

Parameters of the ACER optimized functions

In this part, we will try to find the best parameters. Here, we will study the influence of the amount of data points, the value of the tail marker η_1 and the number of levels to test.

First, we will focus on the influence of η_1. The function $mu1\_detect$ calculates the values of ε but for some levels. We know that the curve of ε reach a peak before converging quickly to its tail. So, we detect the level where we are at the top and we take for η_1 this level's value plus n steps. Then we get the following chart where we were working with a Pareto distribution of parameter 3 and a precision of 0.01 (so a VaR of 10):

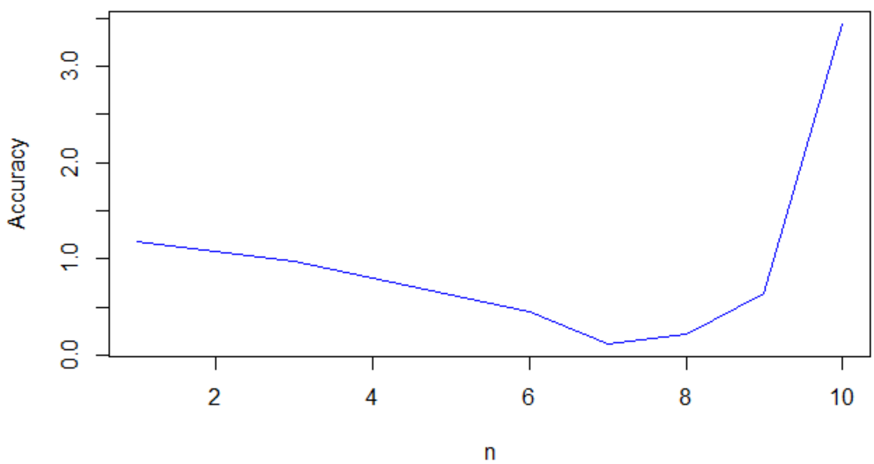

Figure 4.10: Accuracy of the VaR in function of n

Two parts are clearly identifiable on the plot. First, when we increase n, the result is becoming more and more accurate. Indeed, we are moving our tail marker to a position close to the beginning of the tail. Consequently, we are selecting really the efficient level and so get a better fit. Then after some n, our accuracy grows up quickly. We took a tail marker too big and so we are working on a level's scale too small: the used data do not permit us to reach an acceptable result. Then for the following studies, we will work with n = 6.

To optimize the number of levels to test, we will use again a Pareto distribution with 3 as parameter and a precision of 0.01 and we will obtain the accuracy in comparing the VaR to 10 the theoretical one.

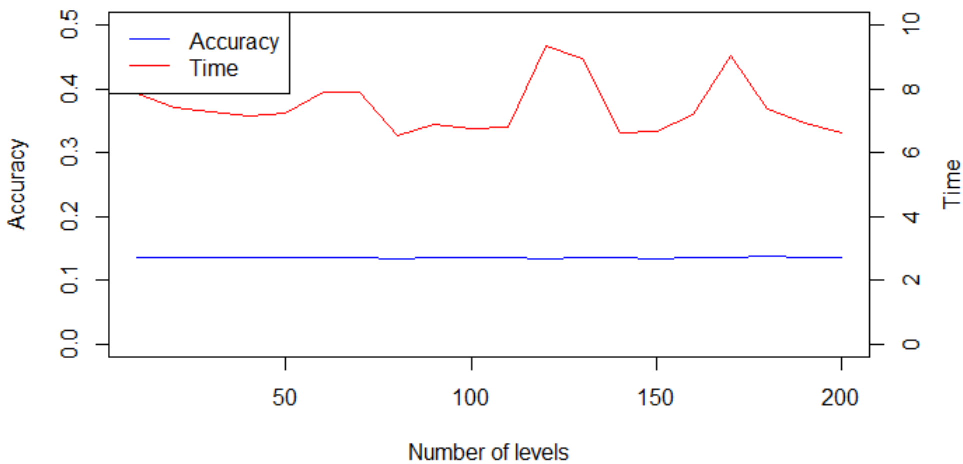

Figure 4.11: Accuracy and execution time in function of the number of levels

We can see that the number of levels does not have a big influence on the accuracy or the execution time of our function. So we will choose a 60 as number of tested levels.

Finally, we will plot the accuracy and the execution time for different amount of points. We will use again a Pareto distribution with parameter $\beta = 3$ and a precision of 0.01 (so an expected VaR of 10).

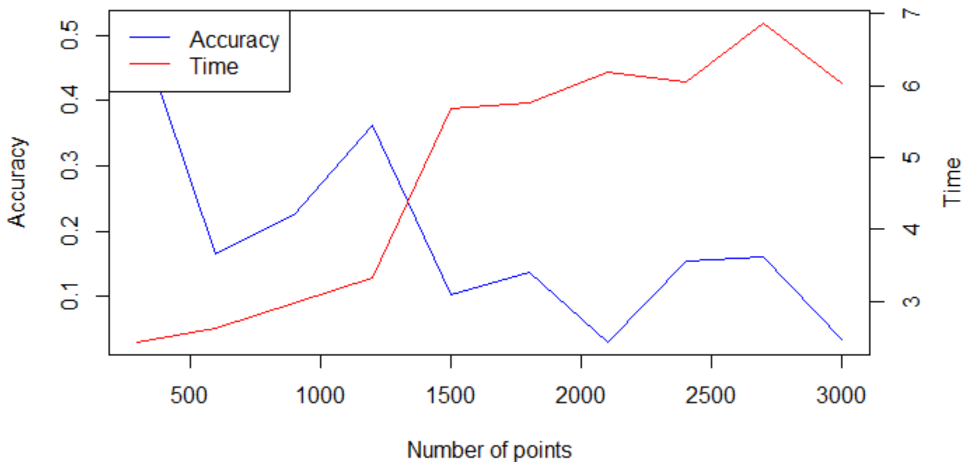

Figure 4.12: Accuracy and execution time in function of the number of data points

Then we can see that a number of data points equal to 20^{th} the inverse of the precision seems kind of coherent. Indeed it seems to be the best execution time/ accuracy ratio. So:

$$\text{number of data points} = \frac{20}{\alpha^2}$$

where α is the precision of the VaR.

Choice of k

To choose k, we need to deal between a lack of values to be really significant for large k and a lack of accuracy for small values of k. Indeed, when k is large, we require the values for k steps back inferior to the level. Consequently, a smaller value of k will permit us to deal with more data points. The simplest solution seems to take k as small as possible respecting a correct tail shape. In the following grpah, we could choose k = 40 for example.

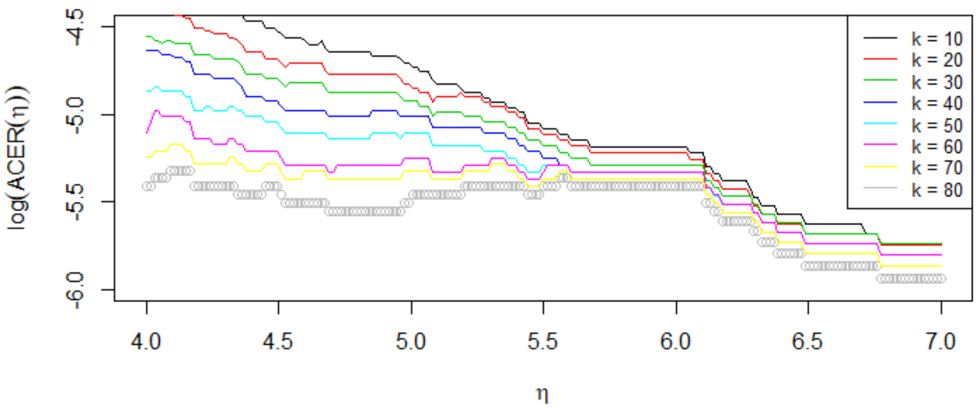

Figure 4.13: ACER functions for different k

4.2 Tests on the Pareto distribution

Now, we will compare the accuracy of the ACER method on a Pareto Distribution with the Monte Carlo method, the Variance/Covariance method and the historical method.

4.2.1 The Pareto distribution

The Pareto distribution is essentially famous because of the ”Pareto principle” or ”80/20”. [17] [27] The first goal of this law was to bring a good model to describe the distribution of wealth in a society. Its creator was an italian civil engineer, economist and sociologist called Vilfredo Pareto. We will see later that the relation between the Pareto distribution and the ”80/20” rule is just a good choice of parameter for the distribution.
In this paper, we will just use and so describe the Pareto distribution of type 1.
Let X be a random variable with a Pareto distribution of type 1.
Then we have the following function to define this distribution:

$$F(X) = Pr(X > x) \tag{4.2}$$

$$= \begin{cases} (\frac{x_m}{x})^{\alpha}, x \geq x_m \\ 1, otherwise \end{cases} \tag{4.3}$$

where x_m is the minimum possible value of X (positive), and is a positive parameter.

This function is called the tail function. The Pareto distribution is really interesting for us to test our model because of its characteristics of fat tail. Indeed, we can see that asymptotically the function F is equivalent to a negative power function. The advantage for our example of this kind of distribution is to have a consequent tail permitting us to work on it.
The cumulative distribution function of a Pareto distribution is given by:

$$F_X(x) = \begin{cases} 1 - (\frac{x}{x_m})^\alpha, x \geq x_m \\ 0, x < x_m \end{cases} \tag{4.4}$$

So to get the plot of the cumulative distribution function, we use the R function *ppareto* and we get:

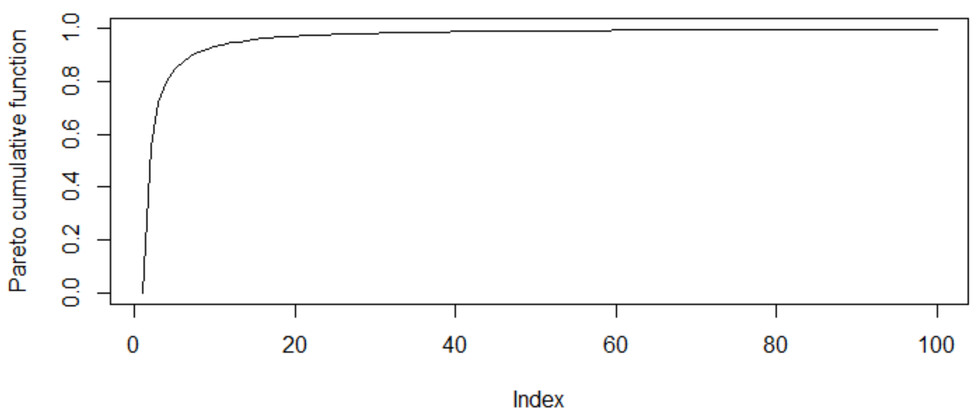

Figure 4.14: Pareto cumulative function with parameter alpha = 1.16

Finally, to create our set of data points, we will use *rpareto* and can get some graphs similar to:

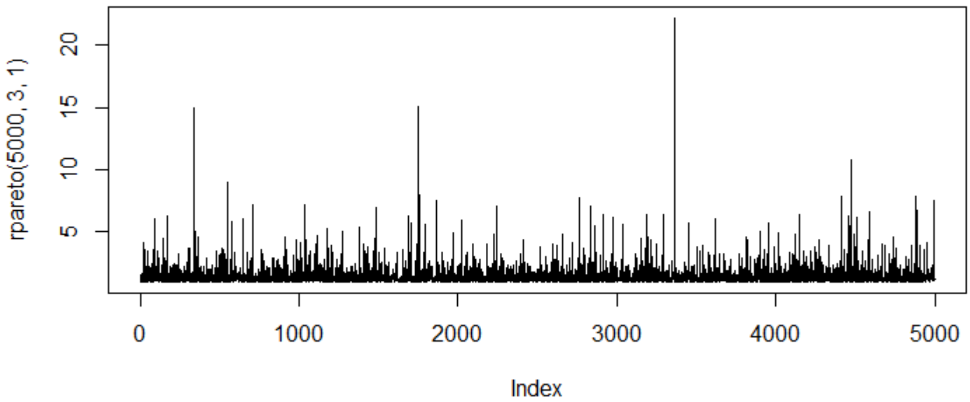

Figure 4.15: 5,000 Pareto random data with beta = 3

Last step, we need to calculate theoretically the VaR of a Pareto distribution. Let V be the VaR, X a random variable following a Pareto distribution with parameters β and x_m and α the accuracy.
We will first use the definition of the VaR and then the cumulative distribution function of a Pareto distribution.

$$\begin{aligned} & P(X \leq V) = \alpha \\ \Rightarrow & (\frac{x_m}{V})^{\beta} = \alpha \\ \Rightarrow & \frac{x_m}{V} = (\alpha)^{\frac{1}{\beta}} \\ \Rightarrow & V = \frac{x_m}{(\alpha)^{\frac{1}{\beta}}} \end{aligned}$$

In the following parts, we will take $x_m = 1$. So we will have: VaR = $(\frac{1}{\alpha})^{\frac{1}{\beta}}$.

4.2.2 Test 10% error on the Pareto distribution of type 1

In this section we will use a Pareto distribution with $\beta = 3$, $x_m = 1$ and $\alpha = 0.01$. According to the last part, we should get a VaR equal to 10.

Moreover, we will take the parameters in agreement with the previous section:

$$\begin{aligned} k &= 40 \\ \text{Amount of data points} &= 2,000 \\ \text{Number of levels to test} &= 60 \end{aligned}$$

The goal is to test several times the ACER function for Pareto distributions with the same parameters and get the VaR to study the accuracy of this method.
We will apply these tests directly on the general case. Indeed, working first with the Gumbel one is not really relevant here. To get sensible values of VaR with the Gumbel case, we need first a distribution we can fit with it.

Econometric	Historical	MC	ACER
7.505949	-0.2351880	-12.190136	-9.027734
7.159335	3.1276093	9.339418	-15.117095
9.233138	15.9544339	10.341797	-7.195085
6.432598	12.4951935	-4.839931	-7.956153
5.163524	6.3381826	-19.419115	-13.623851
6.679253	6.1494003	-8.407815	-14.973722
5.709627	10.2055667	-14.437933	-8.121825
5.595895	4.9289990	-2.145528	-10.814206
5.948782	-1.0481238	-3.620458	-13.600873
10.238323	13.0640472	43.246556	-6.805136
9.259485	4.8304549	2.796427	-13.635974
5.603998	0.2122486	-28.914783	-13.859061
11.173345	9.3848449	-18.457542	-12.149786
9.287330	5.8023602	-9.584246	-12.904584
5.135650	6.8091541	10.480409	-10.395875
11.792332	17.2899524	111.507327	-12.213839
6.237432	1.1305833	-2.035769	-11.531499
6.745641	9.5314288	41.571887	-6.747482
7.937711	3.9064610	-7.710967	-14.970340
6.834449	9.3696491	-17.705047	-12.466970
21.038023	7.5097929	-23.435499	-12.016909
11.535418	3.0190129	-24.135729	-11.212458
6.534582	9.9574133	-14.624497	-6.129999
7.467767	6.4553842	60.989683	-11.208432
5.152897	6.9428139	-21.888150	-11.383310
22.489610	11.1207957	18.196300	-5.142224
8.809668	8.8513421	-19.372116	-6.623456
24.934696	16.0585956	45.395806	-8.820480
13.988804	7.1045940	59.910879	-8.457491
11.977433	11.5935108	68.553153	-8.562538

Figure 4.16: Error in percentage for each method

On this table, the values represent the error for each method. To calculate them in percent, we use the theoritical value of the VaR for the Pareto distribution with $\alpha = 0.05$ and $\beta = 3$ as paramaters, we should get approximately 2.714. Then, the error follows the formula:

$$e = \frac{VaR - 2.714}{2.714} \times 100 \tag{4.5}$$

We used a data set of size 1,000 and an accuracy of 0.05. For the ACER method, we have as parameters $k = 10$, $\eta_1 = 3$ and $\eta_f = 8$.
The green values correspond to an error (in percent) between -10% and 10%, so less than 10% in absolute value.
The orange values are the errors inferior to -10% and the red ones the errors bigger than 10%.
These colors were chosen to differentiate the two kinds of mistakes. Indeed during an investment, the VaR is used to control the taken. However, with a bigger VaR than the correct one, the banker can invest more money than he should and so conduct the company to big troubles. A smaller VaR will "just" bring an investment smaller than the best one. So the banker will win less money but he will respect the safety policy of the bank.

We can see that the ACER method is less accurate than the Econometric one but safer. In a following section, we will try to explain why we have this lost of accuracy. Nevertheless, the fact to get no red value over 30 samples is a really interesting result to use this method in real life. Indeed, we can get with this method a limit for an investment lower than the best doable investment but at least safe. The main goal of the VaR is to check if the investment is safe enough to not bring catastrophic damage.

4.3 The sensibility of this method and the accuracy for the forecast of the VaR

4.3.1 VaR test on a financial data set

Finally, we will check on the asset's values of the Société Générale (SG) if our forecast of maximum is correct.
To do that we will test for each group of 1,000 data points, the next point is smaller than the VaR and how close it can. Indeed, a value cannot be bigger than the VaR otherwise the company's risk is too important. Moreover, to make as much money as possible, the value of the VaR should as close as possible of the value of the VaR. This configuration permits to the banker to invest the biggest amount of money without taking excessive risk and so to have the opportunity to create wealth.
We choose to work on the asset of SG because the values of this asset were directly available on the website of the french bank as an Excel spreadsheet.

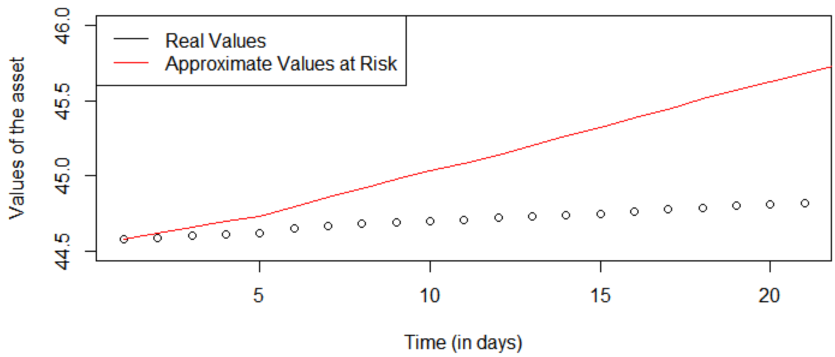

Figure 4.17: Values and approximative Values at Risk of the asset

We can see that the forecast is close to be perfect for up to 5 days. Afterwards, it becomes less and less accurate due to the use of the approximate previous VaR. A solution to keep having a correct VaR could be to do short prediction and use the updated dataset.
Another solution to get a good VaR at one month for example should be to work with a frequency of data seven times bigger than usual that means we will work with data point every seven days in our case.

On the following example, we work with data taken every five days that mean every weeks because we are working with working days.

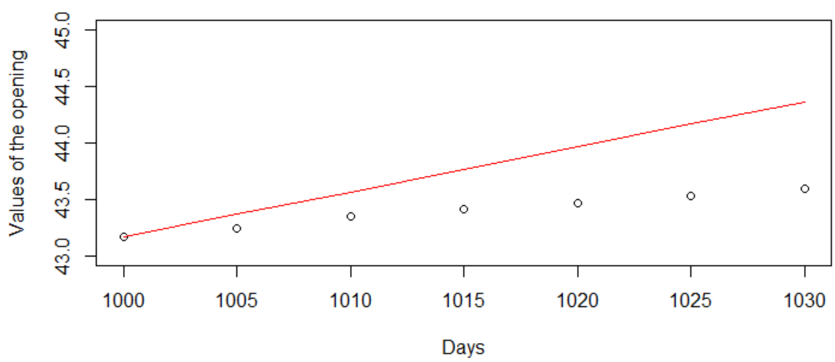

Figure 4.18: Values and approximative Values at Risk of the asset every 5 days

Again, we can see that the values become less and less accurate according to the time of prediction. Nevertheless, we get a correct VaR for a prediction of one, two weeks. Thus we achieve to our goal and the degeneration of our results is essentially due to the use of more approximative values at each step. Indeed, similarly to the forecast of a time series by an ARIMA model, the error will be multiplied by itself or a constant for each step.
Consequently, to keep getting a useful VaR (because a VaR twice bigger than the maximum value is useless in the sense that it will allow all investment), we will need more and more data. For example, to predict a VaR at one month, we will need ten years of data. This need of data represents a limit. Nevertheless, in assuming that the evolution in a long period of time will be just a dezoom of one in a short period that means the asset will evolve quite regularly, the user will be able to do a long forecasting in using the same amount of data but in using recent data. In a nutshell, instead of using more than 10 years of data to predict monthly, he will be able to use the data of the last 125 days. The result will be less accurate but a lot better than the use of forecast data.

4.3.2 Limits

The ACER method seems to be quite efficient for the calculation of a 5% VaR. [22]
Nevertheless, for more accurate VaR, this function may be limited. Indeed, with the use of the natural logarithm, when we will reduce the taken risk in the calculation of the VaR, we will try to find the level for which the logarithm of the ACER function is smaller in negative value or bigger in absolute value. Or, the tail of the curve is really small, and the ACER functions are piecewise for large levels. Consequently, we cannot find an accurate level on a piecewise part and the VaR will not be close to the theoretical one. [6, 14]

Moreover, we can see that a small variation of the parameters k, η_1 and η_f can bring differences on the estimates of the parameters q, a, b, c and γ. This difference will be more important on the calculation of the VaR.

To improve this method, one thing to do should be to find an optimal way to get k, η_1 and η_f. A technique using some "machine learning" can be a good way to implement that. Indeed the function will have to represent an human behaviour. Finally, we can use a complement to this method in calculating the number of peaks over a threshold on a row. This new value could be use in the weights for example. Indeed, if the asset stays a long time over a value, that means the probability to be over this value should be more important. Or in our calculation we do not take this fact in consideration.

Chapter 5

Conclusion

Through this paper, we have compared the ACER method to some classical ones essentially to calculate the VaR.
After having defined the VaR, we have presented the three most famous method for VaR calculation : the historical method, the variance/covariance method and the Monte Carlo one. These methods are today used for their simplicity, accuracy and stability.

However, extreme value statistics is the part of statistics dealing with extremely large or small events. In our case, we are working on rare event happening with a probability of 5%, 1% etc. Thus, the calculation of VaR is a concrete case where we should use the extreme value theory.
To use this theory, after some assumptions, we have built some probabilistic functions calculating the number of peaks over a level, threshold. We called them ACER functions. Then we have fitted these functions with a GEV distribution. Once fitted thanks to a non linear algorithm, the Levenberg-Marquart one, we have reversed the cumulative distribution function to get the VaR.
In this paper, we have justified each step to get the ACER functions but also our program and the made choices like the optimization algorithm.

Finally, we have tested the fitting of the ACER function but also how good could be the ACER method. Comparing to the used one, the ACER methods can sometimes be short of accuracy. Nevertheless, the need of few data and assumptions present a big advantage. Moreover, we can work with dependant data without specified that. Because of these advantages, it should be interesting to go further with the use of extreme value theory and so develop more complex functions to fit.

Bibliography

[1] *Options, futures, and other derivatives*. John, C.Hull, .

[2] *R documentation*.

[3] *An Introduciton to Statistical Modeling of Extreme Values*. 2001.

[4] *Statistics of Extremes, Theory and Applications*. 2005.

[5] *The R book*. 2007.

[6] *Measurmeent error and uncertainty in data: a fascinating statistical challenge*, NTNU, 2017.

[7] Kenneth Abbott. Mathematics with applications in finance - value at risk (var) models, https://www.youtube.com/watch?v=92wanz9mpey, 2013.

[8] Glyn A.Holton. *Value-at-Risk*. , .

[9] Peter Bartlett. Introduction to time series analysis. In .

[10] Patrick Breheny. Bootstrap confidence interval. In .

[11] Chris Egeland Busuttil. Analysis of bivariate extreme values. Master's thesis, NTNU, 2015.

[12] George Casella and Roger L.Berger. *Statistical Inference*.

[13] Learning Curve. An introduction to value-at-risk. In .

[14] Patrick D. T. OConnor and Andre Kleyner. *Practical Reliability Engineering*. 2012.

[15] Manfred Gilli and Evis Kllezi. An application of extreme value theory for measuring financial risk. *Computational Economics*, 2007.

[16] Morgan J.P. *RiskMetrics - Technical Document*. New York, United States of America, 1996.

[17] Wo-Chiang Lee. Applying generalize pareto distribution to the risk management of commerce fire insurance. 2009.

[18] A Nss, O Gaidai, and O Karpa. Estimation of extreme values by the average conditionnal exceedances rate method. *Journal of Probability and Statistics*, 2013.

[19] The University of Regina. The definition of a stochastic process.

[20] Paddy Paddam. A sort introduction to extreme value theory. In .

[21] Computational Economics Practice. Optimization in r. In .

[22] Kristoffer Kofoed Rodvei. Coparing the acer and pot mcmc extreme value statistics methods through analysis of commodities data. Master's thesis, NTNU, 2016.

[23] Sheldon M Ross. *Introduction to probability models*.

[24] Ruey S. Tsay. *Analysis of Financial Time Series - Financial Econometrics*. University of Chicago.

[25] Wikipedia. Levenberg-marquardt.

[26] Wikipedia. Mersenne twister.

[27] Wikipedia. Pareto distribution.

[28] Wikipedia. The probability space.

Appendix A

Stochastic process

A.1 Definition

A stochastic process is simply a collection of random variables, indexed by the time. To give a mathematical definition, we first need to define a probability space.

Definition 1. *A probability space consists of three parts:*

- *A sample space, Ω , which is the set of all possible outcomes.*
- *A set of events $\mathcal{F}$, where each event is a set containing zero or more outcomes.*
- *The assignment of probabilities to the events; that is, a function P from events to probabilities.*

[28] Now, we can give a simple definition of a stochastic process.

Definition 2. *Suppose that (Ω, F, P) is a probability space, and that I $\sqsubset$ R is of infinite cardinality. Suppose further that for each $\alpha \in$ I, there is a random variable X_α : $\Omega \rightarrow$ R defined on (Ω, F, P). The function X : I $\times\Omega \rightarrow$ R defined by X(α, ω) = $X_\alpha(\omega)$ is called a stochastic process with indexing set I, and is written* $X = \{X_\alpha, \alpha \in I\}$.

[19]
Finally, we will define one last kind of stochastic, the counting processes. A counting process is a stochastic process $\{N(t), t \geq 0\}$ whose N(t) represents the number of events occurring by time t.
A counting process needs the following properties:

- N(t) ≥ 0
- N(t) is an integer
- If $s < t$, then $N(t) \geq N(s)$.
- For $s < t$, N(t) - N(s) equals the number of events that occurs in the interval (s,t].

A.2 The Poisson process

In this section, we will present the Poisson process used to get the CI of the ACER functions. This process is a stochastic process with some strong properties.

Definition 3. *A counting process $\{N(t), t \geq 0\}$ is said to be a Poisson process with rate $\lambda > 0$ if:*

- *N(0) = 0*
- $\{N(t), t \geq 0\}$ *has independent increments*
- *P(N(t+h) - N(t) = 1) = λ h + o(h)*
- *P(N(t+h) - N(t) $\geq$ 2) = o(h)*

[23]

We can see that the ACER functions are a counting process of conditional exceedances and consequently it can be reasonable to calculate the CI in using a Poisson process.

Appendix B

Time Series Models

In this chapter of the appendix, we will briefly talk about some time series models. A time serie is simply a data set where the points are indexed in time order. Consequently, time series are discrete-time data sets. In finance for example (but also in weather forecast ...), the main goal is to predict the following values in time. To do that, the process is to fit the data with a known model and then use this model to predict the following values.

We have to define two notions before presenting the ARMA and GARCH models.

Definition 4. *A discrete time stochastic process a_t is called white noise if its mean and its variance are finite. Moreover if a_t is normally distributed with zero mean then a_t is a Gaussian white noise.*

In the following parts, we will use B as index left shift that means: $BX_n = X_{n-1}$.

B.1 The ARMA model

An ARMA model gives us a description of a stationary stochastic process thanks to two polynomials: one for the AR part and one for the MA part. This model was discovered in 1951 by Peter Whittle nad popularized 19 years after by George E. P. Box and Jenkins.

Let X be the data points.
Then the ARMA model follows:

$$\Phi_p(B)X = \Theta_q(B)a_t$$

where $\Phi_p(B) = 1 - \sum_{i=1}^{p} \phi_i B^i$, $\Theta_q(B) = 1 - \sum_{i=1}^{q} \theta_i B^i$ and a$_t isaGaussianwhitenoise.$

If $\Phi_p(B) = 1$ then we have a MA model. Moreover if $\Phi_p(B)$ has no zero root (it is invertible) then we can transform the ARMA model into a infinite MA model. If $\Theta_q(B) = 1$ then we have a AR model.

Finally, the error forecast for the ARMA model can easily be obtained after the transform to the infinite MA process.
We assume that Φ_p has no zero root, then we can write:

$$X_t = \mu + a_t + \sum_{i=1}^{p-1} \psi_i a_{t-i}$$

Then the variance of the error forecast will be: $Var(a_t) * (1 + \sum_{i=1}^{t} \psi_i^2)$.

B.2 the GARCH model

The ARCH model is used to describe, in time series analysis, the error variance. Besides if an ARMA model is assumed for the error variance then the model is a GARCH model.

A GARCH(p,q) model follows:

$$\begin{aligned}
&Y_t = \mu + \varepsilon_t = \mu + \sigma_t z_t \\
&\varepsilon_t | \psi_t \tilde{N}(0, \sigma_t^2) \\
&\sigma_t^2 = w + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{p} \alpha_i \sigma_{t-i}^2
\end{aligned}$$

where ε is the mean-corrected strictly stationary time series, z_t an independent identically distributed random variable ψ_t the set of all informations up to time t-1 and w is a positive constant.

The α_i measures the short-term impact of ε_t on conditional variance and the β_i the long one.

Appendix C

The R codes

C.1 The R main code

```
library(PtProcess)
library(minpack.lm)

#### Calulation of k ####

#This function f_k will count the number of different values
# of the ACER functions between eta_1 and eta_f
f_k<- function(X,k,eta1, etaf, alpha)
{

  # aux <- CI_boot(X,mean(X),k, 50, alpha)
  aux <- CI_Poisson(X, mean(X),k)
  aux1 <- aux$epsilon
  aux_eps <- aux1
  nb_points <- 100
  if(etaf != eta1){
    pas <- abs(etaf - eta1)/(nb_points)
  }
  else{
    pas <- 0
  }

  test_level <- seq(eta1, etaf, pas)
  eps <- rep(0,nb_points)
  for(level in test_level)
  {
```

```
    # aux <- CI_boot(X, level,k, 100)
    aux <- CI_Poisson(X, level,k)

    aux1 <- aux$epsilon
    eps <- c(eps, aux1)
  }

  v_dif <- length(unique(eps[eps!=0]))

  return(v_dif)
}

#This function detect_k will choose a k to
# get a value of k as small as possible without
# having a lack of precision with long sequences
#of constant values
detect_k <- function(X, eta1, etaf, alpha)
{
  N <- length(X)
  M <- 105
  m <- 5
  bi <- f_k(X,m, eta1, etaf, alpha)

  q <- floor(0.8*bi)
  while((M-m) > 5){
    m_aux <- floor((M+m)/2)
    v <- f_k(X,m_aux, eta1, etaf, alpha)
    if(v < q){
      M <- m_aux
    }
    else{
      m <- m_aux
    }
  }
  return(m)
}



#### Calculation of etaf1, eta_f ####
#eta1 will represent the lowest level and eta_f the
#highest of our study
eta_detect <- function(X, k, alpha)
{
  nb_values <- 200
```

```
  pas <- (max(X)-min(X))/nb_values
  test_level <- seq(min(X),max(X),pas)
  i <- 1
  k_i <- k
  aux_end <- rep(0,k_i)
  for(level in test_level)
  {
    # aux <- CI_boot(X,level,k_i,100, alpha)
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    if(aux1>0){
      aux_end[i] <- (aux1)

    }
    else{
      aux_end[i] <- 0
    }
    i = i+1
  }
  m <- max(aux_end)
  ind <- max(which(aux_end == m))
  v <- aux_end[ind:length(aux_end)]
  mu1 <- test_level[ind]
  w <- diff(log(v))
  l <- which(w != 0)
  r <- diff(l)

  m <- min(which(r >8))
  p <- l[m]
  muf <- test_level[p]

  return(c(mu1, muf))
}



#### Detection of exceedances ####
sum_validate_data <- function(X, level)
{
  N <- length(X)
  a <- rep(0, N-1)
  # Calculation of a, number of exceedance respecting
  # the condition
  index <- which(X > level)
  l <- length(index)
```

```
  if(l==0)
  {
    return(a)
  }
  ### firt index
  aux <- index[1]
  if(aux > 0)
  {
    a[1:(aux)] <- a[1:(aux)] + rep(1,aux)
  }
  if(l >1)
  {
    for (i in 1:(l-1))
    {

      aux <- index[i+1] - index[i] - 1
      if(aux >= 1)
      {
        a[1:(aux)] <- a[1:(aux)] + rep(1,aux)
      }
    }
  }

  return(a)
}


#### ACER values ####
epsilon_app <- function(X,aux_sum, k)
{
  # Here we just divide the number of exceedance
  # by approximately the number of points to get a probability
  epsilon <- aux_sum[k]/(length(X)+1-k+1)
  return(epsilon)
  # }
}


#### Confidence intervals ####
CI_Poisson <- function(X, level, k)
{
  #First we calculate the value of epsilon
  x <- sum_validate_data(X, level)
  N <- length(X)
  eps_aux <- epsilon_app(X,x, k)
```

```
  # Then we apply the classical formula of a CI for a Poisson
  #distribution
  if(eps_aux == 0){
    C_plus <- 1.96*sqrt((N-k+1))
    C_minus <- -1.96*sqrt((N-k+1))
  }
  else{
    C_plus <- eps_aux*(1 + 1.96/sqrt((N-k+1)*eps_aux))
    C_minus <- eps_aux*(1 - 1.96/sqrt((N-k+1)*eps_aux))
  }


  C <- data.frame(C_plus, C_minus)
  return(list("CI" = C, "epsilon" = eps_aux))
}

CI_boot <- function(X, level, k, nb, alpha)
{
  nb_sampling <- nb
  L <- rep(0, nb_sampling)
  for(j in 1:nb_sampling){
    # We calculate the value of epsilon for sample based on
    # the initial data set.
    x <- sum_validate_data(sample(X, replace = T), level)
    L[j] <- epsilon_app(X,x, k)
  }

  # we take the alpha % confidence interval
  aux <- sort(L)
  C_minus <- aux[floor(alpha/2*length(aux))]
  C_plus <- aux[floor((1-alpha/2)*length(aux))+1]

  C <- list(C_plus, C_minus)
  x <- sum_validate_data(X, level)
  eps_aux <- epsilon_app(X,x, k)

  return(list("CI" = C, "epsilon" = eps_aux))
}



#### ACER method ####
ACER_General <- function(X, alpha)
{
```

```
#First we fixe k at a possible value
# and we find eta1 and eta_f. These two values
#do not vary too much with k
eta <- eta_detect(X, 2, alpha)
eta1 <- eta[1]
etaf <- eta[2]

#With eta1 and eta_f, we find k.
k <- detect_k(X, eta1, etaf, alpha)

#We calculate then the true values of eta1 and eta_f
eta <- eta_detect(X, k, alpha)
eta1 <- eta[1]
etaf <- eta[2]

nb_points <- 200

pas <- (etaf-eta1)/(nb_points)
test_level <- seq(eta1, etaf, pas)
aux_eps <- 0
aux_w <- 0
for(level in test_level)
{
  # aux <- CI_boot(X, level,k, 100)
  aux <- CI_Poisson(X, level, k)
  aux1 <- aux$epsilon
  aux2 <- unlist(aux$CI)
  v <- 1/(log(abs(aux2[1])) -log(abs(aux2[2])))^2

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}

eps <- aux_eps[2:length(aux_eps)]
w <- aux_w[2:length(aux_w)]

level <- test_level

y <- log(eps)
# plot(w)
y_m <- sum(w*y) / sum(w)

ACER_sim_vect <- function(param){
  #we are creating now the function for the given k
```

```
  b <- param[2]
  c <- param[3]
  a_t <- param[1]
  x <- log(abs(1 + a_t*abs(level - b)^c))
  x_m <- sum(w*x)/sum(w)
  gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + gam*x_m)
  result <- rep(0, length(w))
  for(i in 1:length(w)){
    result[i] <- as.double(sqrt(w[i])*abs(y[i] - log(q)
             + gam*log(abs(1 + a_t*abs(level[i] - b)^c))))
  }
  return(unlist(result))
}
#
#
m <- min(X)
# ## we need to find b, c
minit <- (m+eta1)/2


o <- nls.lm(c(1,minit, 1), lower = c(0,m, 0),
      upper = c(Inf,eta1, 5), ACER_sim_vect,  jac = NULL,
      nls.lm.control(ftol = 0.001,maxiter = 1000))


param <- o$par
##Now, we need to find a, q_ln
a_t <- param[1]
b <- param[2]
c <- param[3]
x <- log((1 + a_t*abs(level - b)^c))
x_m <- sum(w*x)/sum(w)

gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))

q <- exp((y_m + gam*x_m))


# print(c(a_t,b,c,q,gam))

lres <- rep(0, length(eps))

for(l in 1:length(eps)){
  lres[l] <- log(q) - gam*log(1+a_t*abs(level[l] - b)^c)
```

```
  }

  return(list(coef = c(a_t, b, c, q, gam), eps = eps,
              level = level))

}



#### Calculation VaR ####
VaR <- function(X, alpha){
  v <- ACER_General(X, alpha)
  coef <- v$coef
  coef <- unlist(coef)
  a_t <- coef[1]
  b <- coef[2]
  c <- coef[3]
  q <- coef[4]
  gam <- coef[5]
  # VaR <- b+abs(1/a_t*abs(alpha/q)^(1/gam) - 1)^(1/c)
  VaR_new <- b + abs(((alpha/q)^(-1/gam)-1)/a_t)^(1/c)
  while(VaR_new > max(X))
  {
    VaR_new <- VaR(X,alpha)
  }
  return(VaR_new)
}
```

C.2 The tests' code

```
#### Libraries ####
library(minpack.lm)
library(optimx)
library(forecast)
library(PtProcess)
library(tseries)
library(formattable)
library(htmltools)
library(webshot)

#### Tests of functions to get the exceedance ####
# These functions work elementwise on the matrices to build
# Building of A_kj
building_matrices <- function(X, level)
{
```

```
  # initialisation
  N <- length(x = X)
  init <- rep(0, times = (N-1)*(N-1))
  A = matrix(data = init ,nrow =  N-1, ncol = N-1)
  # N - 1 because k starts at 2

  # Calculations
  for(k in 2:N)
  {
    for(j in k:N)
    {
      if(max(X[(j-k+1):(j-1)]) <= level)
      {
        if(X[j] > level)
        {
          A[k-1,j-1] = 1
        }
      }
    }
  }
  return(A)
}


# Just calculation of the expected values
sum_peak <- function(X, level)
{
  # initialisation
  N <- length(x = X)
  a = rep(0,N-1)
  # Calculations
  for(k in 2:N)
  {
    for(j in k:N)
    {
      if(max(X[(j-k+1):(j-1)]) <= level)
      {
        if(X[j] > level)
        {
          a[k-1] <- a[k-1] + 1 #sum of a_kj for k fixed
        }
      }
    }
  }
  return(a)
```

```
}

# This function works on the data directly
sum_validate_data <- function(X, level)
{
  N <- length(X)
  a <- rep(0, N-1)
  # Calculation of a and b
  index <- which(X > level)
  l <- length(index)
  if(l==0)
  {
    return(a)
  }
  # firt index
  aux <- index[1]
  if(aux > 0)
  {
    a[1:(aux)] <- a[1:(aux)] + rep(1,aux)
  }
  if(l >1)
  {
    for (i in 1:(l-1))
    {

      aux <- index[i+1] - index[i] - 1
      if(aux > 1)
      {
        a[1:(aux)] <- a[1:(aux)] + rep(1,aux)
      }
    }
  }
  return(a)
}

test_exceedance_funct <- function()
{
    l <- 10
    t_1 <- rep(0, l)
    t_2 <- rep(0, l)
    t_3 <- rep(0, l)
    aux <- rep(0, l)
    level <- 2
    for(l in 1:l){
      X <- rpareto(l*100, 3, 1)
```

```
      t <- Sys.time()
      A <- building_matrices(X,level)
      aux <- sum(A)
      t_1[l] <- Sys.time()-t
      t <- Sys.time()
      a <- sum_peak(X,level)
      t_2[l] <- Sys.time()-t
      t <- Sys.time()
      a <- sum_validate_data(X,level)
      t_3[l] <- Sys.time()-t
    }
    x <- seq(100, l*100, 100)
    plot(x = x, y = t_1, type = "l", col = "blue",
         xlab = "Length␣of␣X", ylab = "Time␣in␣second")
    lines(x = x, y = t_2, type = "l", col = "green" )
    lines(x = x, y = t_3, type = "l", col = "red" )
    legend("topleft", legend = c("building_matrices",
    "sum_peak", "sum_validate_data"),
    col=c("blue", "green", "red"), lty = c(1,1,1))
}

#### Tests on CI functions and calculation of ACER function ####
# Calculation of Epsilon
epsilon_app <- function(X,aux_sum, k)
  {
    # Here we just divide the number of exceedance
    # by approximately the number of points to get
    # a probability
    epsilon <- aux_sum[k]/(length(X)+1-k+1)
    return(epsilon)
    # }
  }


# Confidence interval

# Block method
CI_block <- function(X, level, k)
{

  # First we test the seasonality best chance to get
  # a block structure
  model <- auto.arima(X, D=1)
  # to force the auto.arima to find a seasonability
  b <- model$arma
```

```
  s <- b[5] ### s get the seasonality
  s <- 3
  aux_mean <- rep(0, s-1)
  # we have to share the data into blocks of
  # length s and work on each blocks
  N <- length(X)
  R <- floor(N/s)
  list_aux <- list()
  x_aux <- rep(0, N)
  for(i in 1:R)
  {
    x <- X[((i-1)*s+1) : (i*s)]
    x_aux <- sum_validate_data(x, level)
    list_aux[[i]] <- epsilon_app(X, x_aux, k)
  }

  # last block

  x <- X[(R*s+1) : N]
  x_aux <- sum_validate_data(x, level)
  list_aux[[R+1]] <- epsilon_app(x_aux, length(x_aux), k)

  eps_aux <- data.frame(t(sapply(list_aux,c)))
  # Calculation of s_k
  std_dev <- rep(0, s-1)
  for(k in 1:(s-1))
  {
    l_aux <- list_aux[[k]]
    aux <- unlist(l_aux)
    aux[is.na(aux)] <- 0
    aux_mean[k] <- mean(aux)
    aux2 <- aux - rep(aux_mean[k], length(aux))
    std_dev[k] <- (sum(aux2*aux2))/(R-1)
  }

  # Creation of the confidence interval
  C_plus <- aux_mean + 1.96*sqrt(std_dev)/sqrt(R)
  C_minus <- aux_mean - 1.96*sqrt(std_dev)/sqrt(R)
  C <- data.frame(C_plus, C_minus)
  return(list("CI" = C, "epsilon" = eps_aux))
}

CI_Poisson <- function(X, level, k)
{
  #First we calculate the value of epsilon
```

```
  x <- sum_validate_data(X, level)
  N <- length(X)
  eps_aux <- epsilon_app(X,x, k)

  # Then we apply the classical formula of a CI for a Poisson
  #distribution
  if(eps_aux == 0){
    C_plus <- 1.96*sqrt((N-k+1))
    C_minus <- -1.96*sqrt((N-k+1))
  }
  else{
    C_plus <- eps_aux*(1 + 1.96/sqrt((N-k+1)*eps_aux))
    C_minus <- eps_aux*(1 - 1.96/sqrt((N-k+1)*eps_aux))
  }


  C <- data.frame(C_plus, C_minus)
  return(list("CI" = C, "epsilon" = eps_aux))
}

CI_boot <- function(X, level, k, nb)
{
  nb_sampling <- nb
  L <- rep(0, nb_sampling)
  for(j in 1:nb_sampling){
    #We calculate the value of epsilon for sample based on
    # the initial data set.
    x <- sum_validate_data(sample(X, replace = T), level)
    L[j] <- epsilon_app(X,x, k)
  }

  # we take the alpha % confidence interval
  aux <- sort(L)
  C_minus <- aux[floor(0.025*length(aux))]
  C_plus <- aux[floor(0.975*length(aux))+1]

  C <- list(C_plus, C_minus)
  x <- sum_validate_data(X, level)
  eps_aux <- epsilon_app(X,x, k)

  return(list("CI" = C, "epsilon" = eps_aux))
}


level <- 2
```

```
k <- 6
nb_samp <- 50
len <- 20
t_1 <- rep(0, len)
t_2 <- rep(0, len)
e_k_2 <- rep(0, len)
a_2 <- rep(0, len)
t_3 <- rep(0, len)
e_k_3 <- rep(0, len)
a_3 <- rep(0, len)
level <- 2

test_CI_time <- function(){
    # TIME
    for(l in 1:len){
      X <- rpareto(l*100, 3, 1)
      t <- Sys.time()
      b <- CI_block(X, level, k)
      t_1[l] <- Sys.time()-t
      t <- Sys.time()
      c <- CI_Poisson(X, level, k)
      t_2[l] <- Sys.time()-t
      t <- Sys.time()
      d <- CI_boot(X, level,k, nb_samp)
      t_3[l] <- Sys.time()-t
    }
    x <- seq(100, len*100, 100)


    plot(x = x, y = t_1, type = "l", col = "blue",
        xlab = "Length of X", ylab = "time in second")
    lines(x = x, y = 25*t_2, type = "l", col = "green" )
    lines(x = x, y = t_3, type = "l", col = "red" )
    legend("topleft", legend = c("CI_block", "25*CI_Poisson",
    "CI_boot"), col=c("blue", "green", "red"), lty = c(1,1,1))
}

test_CI_acc <- function(){
    # ACCURACY vs length
    for(l in 1:len){
      X <- rpareto(l*100, 3, 1)
      t <- Sys.time()
      a <- CI_Poisson(X, level, k)
      e_k_2[l] <- (a$epsilon)[k]
      diff_2 <- (a$CI)[1]- (a$CI)[2]
```

```
    a_2[l] <- sqrt(sum(diff_2*diff_2))
    t_2[l] <- Sys.time()-t
    t <- Sys.time()
    b <- CI_boot(X, level,k, nb_samp)
    e_k_3[l] <- (b$epsilon)[k]
    diff_3 <- (b$CI)[1]- (b$CI)[2]
    a_3[l] <- sqrt(sum(diff_3*diff_3))
    t_3[l] <- Sys.time()-t
  }
  x <- seq(100, len*100, 100)
  plot(x = x, y = a_2, type = "l", col = "blue",
       xlab = "Length␣of␣X", ylab = "Accuracy")
  lines(x = x, y = 20*a_3, type = "l", col = "red" )
  legend("topleft", legend = c("CI_Poisson", "20*CI_boot"),
         col=c("blue", "red"), lty = c(1,1,1))

  plot(x = e_k_2, y = e_k_3, col = "blue",
  xlab = "Epsilon␣by␣CI_Poisson", ylab = "Epsilon␣by␣CI_boot")
  lines(e_k_2,e_k_2, col = "red" )
}


test_CI_boot_nb_sample <- function(){
  # TIME and ACCURACY vs nb_sample
  for(l in 1:len){
    X <- rpareto(2000, 3, 1)
    t <- Sys.time()
    b <- CI_boot(X, level,k, l*10)
    e_k_3[l] <- (b$epsilon)[k]
    diff_3 <- (b$CI)[1]- (b$CI)[2]
    a_3[l] <- sqrt(sum(diff_3*diff_3))
    t_3[l] <- Sys.time()-t
  }
  x <- seq(10, len*10, 10)
  par(mar=c(4,4,3,5))
  plot(x = x, y = t_3, type = "l", col = "blue",
       xlab = "Number␣of␣samplings", ylab = "Time")

  par(new=T)
  plot(x = x, y = a_3, type = "l", col = "red", axes=F,
       xlab = "", ylab ="")
  mtext("Length␣of␣the␣CI",side=4, line = 2.5)
  axis(4)
  legend(x="topleft",legend=c("Time","Length␣of␣the␣CI"),
         col=c("blue","red"),lty = c(1,1))
```

```r
}

#### Tests optimization algorithms ####
# Detection of the tail marker eta1
eta_detect <- function(X, k, alpha)
{
  nb_values <- 200
  pas <- (max(X)-min(X))/nb_values
  test_level <- seq(min(X),max(X),pas)
  i <- 1
  k_i <- k
  aux_end <- rep(0,k_i)
  for(level in test_level)
  {
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    if(aux1>0){
      aux_end[i] <- (aux1)

    }
    else{
      aux_end[i] <- 0
    }
    i = i+1
  }
  m <- max(aux_end)
  ind <- max(which(aux_end == m))
  v <- aux_end[ind:length(aux_end)]
  mu1 <- test_level[ind]
  return(mu1)
}

# Start research model

ACER_Gumble_1 <- function(X, k, alpha)
{
  k_i <- k
  mu1 <- eta_detect(X, k)
  muf <- max(X)
  nb_points <- 100
  pas <- (muf - mu1)/(nb_points)
  test_level <- seq(mu1, muf, pas)

  aux <- CI_Poisson(X,mean(X),k_i)
  aux1 <- aux$epsilon
```

```r
aux2 <- aux$CI
aux_eps <- aux1
print(aux)
aux_w <- 1/(log(aux2[1]-aux2[2]))^2
for(level in test_level)
{
  aux <- CI_Poisson(X, level, k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  v <- 1/(log(aux2[1] -aux2[2])^2)

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}
aux_w <- aux_w[-1]
aux_eps <- aux_eps[-1]



# create the functions to optimize
eps <- aux_eps[aux_eps != 0]
w <- aux_w[aux_eps != 0]
test_level <- test_level[aux_eps != 0]
level <- test_level
ACER_funct_value <- function(param){

  q <- param[1]
  a <- param[2]
  b <- param[3]
  c <- param[4]
  result <- 0
  result <- sum(w*abs((log(eps) - log(q) + a*(test_level - b)^c))^2)

  return(result)
}


U <- matrix(data = c(1,0,0,0, 0, 1, 0, 0, 0, 0 ,1, 0, 0, 0 ,-1, 0,
      0, 0, 0, 1, 0, 0, 0, -1), nrow = 6, ncol = 4, byrow = TRUE)
m <- min(X)
M <- max(X)
C <- c(0 ,0,  m, -mu1, 0, -5)

minit <- (m+mu1)/2
```

```
  print(minit)
  print(m)
  print(mu1)
  o <- constrOptim(c(1,1, minit, 2), ACER_funct_value, NULL, U, C,
                   outer.eps = 0.0001 )
  param <- o$par
  q <- param[1]
  a <- param[2]
  b <- param[3]
  c <- param[4]

  VaR <- b+(1/a*(abs(log(alpha/q))))^(1/c)

  return(list(q = q, a = a, b = b, c = c, VaR = VaR,
              tl = test_level, eps = eps))
}

ACER_Gumble_2 <- function(X, k, alpha)
{
  k_i <- k
  mu1 <- eta_detect(X, k)
  muf <- max(X)
  nb_points <- 100
  pas <- (muf - mu1)/(nb_points)
  test_level <- seq(mu1, muf, pas)

  aux <- CI_Poisson(X,mean(X),k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  aux_eps <- aux1
  print(aux)
  aux_w <- 1/(log(aux2[1]-aux2[2]))^2
  for(level in test_level)
  {
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    aux2 <- aux$CI
    v <- 1/(log(aux2[1] -aux2[2])^2)

    aux_eps <- c(aux_eps, aux1)
    aux_w <- c(aux_w, v)

  }
  aux_w <- aux_w[-1]
  aux_eps <- aux_eps[-1]
```

```
  # create the functions to optimize
  eps <- aux_eps[aux_eps != 0]
  w <- aux_w[aux_eps != 0]
  test_level <- test_level[aux_eps != 0]
  level <- test_level
  ACER_funct_vect <- function(param){
    q <- param[1]
    a <- param[2]
    b <- param[3]
    c <- param[4]
    result <- rep(0, length(w))
    for(i in 1:length(w)){
      result[i] <- as.double(sqrt(w[i])*abs(log(eps[i]) - log(q)
                                  + a*abs(test_level[i] - b)^c))
    }
    return(result)
  }

  m <- min(X)
  M <- max(X)

  minit <- (m+mu1)/2
  o <- nls.lm(c(1,1, minit, 1), lower = c(0, 0, 0.99*m, 0),
              upper = c(Inf, Inf, 1.01*mu1, 5),ACER_funct_vect,
          jac = NULL,
          nls.lm.control(ftol = 0.0001,maxiter = 1000))
  param <- o$par
  q <- param[1]
  a <- param[2]
  b <- param[3]
  c <- param[4]
  VaR <- b+(1/a*(-log(alpha/q)))^(1/c)
  return(list(q = q, a = a, b = b, c = c, VaR = VaR,
              tl = test_level, eps = eps))
}

ACER_Gumble_3 <- function(X, k, alpha)
{
  k_i <- k
  mu1 <- eta_detect(X, k)
  muf <- max(X)
  nb_points <- 100
```

```
pas <- (muf - mu1)/(nb_points)
test_level <- seq(mu1, muf, pas)

aux <- CI_Poisson(X,mean(X),k_i)
aux1 <- aux$epsilon
aux2 <- aux$CI
aux_eps <- aux1
print(aux)
aux_w <- 1/(log(aux2[1]-aux2[2]))^2
for(level in test_level)
{
  aux <- CI_Poisson(X, level, k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  v <- 1/(log(aux2[1] -aux2[2])^2)

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}
aux_w <- aux_w[-1]
aux_eps <- aux_eps[-1]



# create the functions to optimize
eps <- aux_eps[aux_eps != 0]
w <- aux_w[aux_eps != 0]
test_level <- test_level[aux_eps != 0]
level <- test_level

y <- log(eps)
y_m <- sum(w*y) / sum(w)

ACER_sim <- function(param){
  b <- param[1]
  c <- param[2]

  x <- (level - b)^c
  x_m <- sum(w*x)/sum(w)
  a <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + a*x_m)
  result <- sum(w*(y-log(q)+a*x)^2)
  return(as.double(result))
}
```

```
  #
  #
  # ## we need to find b, c
  U <- matrix(data = c( 1, 0,-1, 0, 0, 1, 0, -1), nrow = 4,
              ncol = 2, byrow = TRUE)
  m <- min(X)
  C <- c(0.99*m, -1.01*mu1, 0, -5)
  minit <- (mu1+m)/2
  o <- constrOptim(c(minit, 1), ACER_sim, NULL,
                   method =  "Nelder-Mead",
                   U, C, outer.eps = 1e-07)

  param <- o$par
  b <- param[1]
  c <- param[2]
  x <- (level - b)^c
  x_m <- sum(w*x)/sum(w)
  a <- abs(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + a*x_m)


  VaR <- b+(1/a*(-log(alpha/q)))^(1/c)

  return(list(q = q, a = a, b = b, c = c, VaR = VaR,
              tl = test_level, eps = eps))
}

ACER_Gumble_4 <- function(X, k, alpha)
{
  k_i <- k
  mu1 <- eta_detect(X, k)
  muf <- max(X)
  nb_points <- 100
  pas <- (muf - mu1)/(nb_points)
  test_level <- seq(mu1, muf, pas)

  aux <- CI_Poisson(X,mean(X),k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  aux_eps <- aux1
  print(aux)
  aux_w <- 1/(log(aux2[1]-aux2[2]))^2
  for(level in test_level)
  {
```

```
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    aux2 <- aux$CI
    v <- 1/(log(aux2[1] -aux2[2])^2)

    aux_eps <- c(aux_eps, aux1)
    aux_w <- c(aux_w, v)

  }
  aux_w <- aux_w[-1]
  aux_eps <- aux_eps[-1]



  # create the functions to optimize
  eps <- aux_eps[aux_eps != 0]
  w <- aux_w[aux_eps != 0]
  test_level <- test_level[aux_eps != 0]
  level <- test_level
  y <- log(eps)
  y_m <- sum(w*y) / sum(w)



  ACER_sim_vect <- function(param){
    b <- param[1]
    c <- param[2]

    x <- abs(level - b)^c
    x_m <- sum(w*x)/sum(w)
    a <- abs((sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2))))
    q <- exp(y_m + a*x_m)
    result <- as.double(w*(y-q+a*x)^2)
    return(result)
  }


  # ## we need to find b, c
  m <- min(X)
  minit <- (mu1+m)/2

  o <- nls.lm(c(minit, 1), lower = c(m, 1), upper = c(mu1, 5),
      ACER_sim_vect,  jac = NULL,
      nls.lm.control(ftol = 0.001,  maxiter = 1000))
```

```
  param <- o$par
  #Now, we need to find a, q_ln
  b <- param[1]
  c <- param[2]
  x <- abs(level - b)^c
  x_m <- sum(w*x)/sum(w)
  a <- abs(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + a*x_m)


  VaR <- b+(1/a*(abs(log(alpha/q))))^(1/c)

  return(list(q = q, a = a, b = b, c = c, VaR = VaR))
}



X <- rpareto(2000, 3, 1)

## error optimization algo
test_error_opti_alg <- function(X){
    v <- ACER_Gumble_1(X,20, 0.05)
    ep <- v$eps
    ep <- log(ep)
    x <- v$tl
    q <- v$q
    a <- v$a
    b <- v$b
    c <- v$c
    a_1 <- log(q) -a*abs(x-b)^c
    e_1 <- sqrt((ep-a_1)*(ep-a_1))
    plot(x=x[x>2], y=0.5*e_1[x>2], type = "l", xlab = "Level",
         ylab = "Error", ylim = c(0,0.7))

    v <- ACER_Gumble_2(X, 20, 0.05)
    q <- v$q
    a <- v$a
    b <- v$b
    c <- v$c
    a_1 <- log(q) -a*abs(x-b)^c
    e_1 <- sqrt((ep-a_1)*(ep-a_1))
    lines(x[x>2], e_1[x>2], type = 'l', col = "blue")

    v <- ACER_Gumble_3(X, 20, 0.05)
    q <- v$q
```

```
    a <- v$a
    b <- v$b
    c <- v$c
    a_1 <- log(q) -a*abs(x-b)^c
    e_1 <- sqrt((ep-a_1)*(ep-a_1))
    lines(x[x>2], e_1[x>2], type = 'l', col = "green")

    v <- ACER_Gumble_4(X, 20, 0.05)
    q <- v$q
    a <- v$a
    b <- v$b
    c <- v$c
    a_1 <- log(q) -a*abs(x-b)^c
    e_1 <- sqrt((ep-a_1)*(ep-a_1))
    lines(x[x>2], e_1[x>2], type = 'l', col = "red")

    legend(x="topright",
      c("0.5*Lin_Comp","LM_Comp", "Lin_Simpl","LM_Simpl"),
      col=c("black","blue", "green","red"),lty = c(1,1,1,1))
}

test_time_opti_alg <- function(){
    # execution time algo
    len <- 20
    t_1 <- rep(0,len)
    t_2 <- rep(0,len)
    t_3 <- rep(0,len)
    t_4 <- rep(0,len)

    for(l in 1:len){
      X <- rpareto(1000+100*l, 3, 1)
      t <- Sys.time()
      v <- ACER_Gumble_1(X,8,0.05)
      t_1[l] <- Sys.time()-t
      t <- Sys.time()
      v <- ACER_Gumble_2(X,8,0.05)
      t_2[l] <- Sys.time()-t
      t <- Sys.time()
      v <- ACER_Gumble_3(X,8,0.05)
      t_3[l] <- Sys.time()-t
      t <- Sys.time()
      v <- ACER_Gumble_4(X,8,0.05)
      t_4[l] <- Sys.time()-t
    }
    x <- seq(1100, 3000, 100)
```

```
    plot(x, t_1, col = "red", type = "l", ylab = "Time",
         xlab = "Number␣of␣points")
    lines(x, t_2, type = "l",col= "blue")
    lines(x, t_3,type = "l", col= "green")
    lines(x, t_4, type = "l",col = "black")
    legend(x="topleft",
           c("Lin_Comp","LM_Comp", "Lin_Simpl", "LM_Simpl"),
           col=c("black","blue", "green","red"),lty = c(1,1,1,1))
}




## test_LM_algo_nbPoints
test_nb_points <- function(){
    #here we will use LM_simpl and LM_Compl
    len <- 20
    t_1 <- rep(0,len)
    t_2 <- rep(0,len)
    t_3 <- rep(0,len)
    t_4 <- rep(0,len)

    for(l in 1:len){
      X <- rpareto(500+200*l, 3, 1)
      t <- Sys.time()
      v <- ACER_Gumble_1(X,8,0.05)
      t_1[l] <- Sys.time()-t
      t <- Sys.time()
      v <- ACER_Gumble_4(X,8,0.05)
      t_2[l] <- Sys.time()-t
    }
    x <- seq(700, 4500, 200)
    plot(x, t_1, col = "red", type = "l", ylab = "Time",
         xlab = "Number␣of␣points")
    lines(x, t_2, type = "l",col = "black")
    legend(x="topleft",legend=c("Lin_Simpl", "LM_Simpl"),
           col=c("black","red"),lty = c(1,1))
}

#Now we choose LM_simpl (ACER_gumbel_4)
test_nb_pts_time_acc <- function(){
  #we want to test the influence of the number of points on
  # the time and the accuracy with VaR
    len <- 10
```

```
    t <- rep(0,len)
    ac <- rep(0,len)
    alpha <- 0.001

    # X <- rpareto(1000, 3, 1)
    # v <- ACER_Gumble_4(X,3,alpha)
    # print(v$VaR)

    for(l in 1:len){

      for(i in 1:3){
        X <- rpareto(300+300*l, 3, 1)
        t_aux <- Sys.time()
        v <- ACER_Gumble_4(X,3,alpha)
        print(v)
        t[l] <- t[l] + Sys.time()-t_aux
        ac[l] <- ac[l] + v$VaR
      }
      ac[l] <- ac[l]/3
      t[l] <- t[l]/3
    }
    ac <- abs(ac - 10)/10

    x <- seq(300, len*300, 300)
    par(mar=c(4,4,3,5))
    plot(x = x, y = ac, type = "l", col = "blue",
         xlab = "Number␣of␣points", ylab = "Accuracy")

    par(new=T)
    plot(x = x, y = t, type = "l", col = "red", axes=F,
         xlab = "", ylab ="")
    mtext("Time",side=4, line = 2.5)
    axis(4)
    legend(x="topleft",legend=c("Accuracy","Time"),
           col=c("blue","red"), lty = c(1,1))
}


## test precision mu1
test_accur_eta <- function(){
      eta_detect_test <- function(X,n){
          nb_values <- 200
          pas <- (max(X)-min(X))/nb_values
          test_level <- seq(min(X),max(X),pas)
          i <- 1
```

```
    k_i <- k
    aux_end <- rep(0,k_i)
    for(level in test_level)
    {
      aux <- CI_Poisson(X, level, k_i)
      aux1 <- aux$epsilon
      if(aux1>0){
        aux_end[i] <- (aux1)

      }
      else{
        aux_end[i] <- 0
      }
      i = i+1
    }
    m <- max(aux_end)
    ind <- max(which(aux_end == m))
    v <- aux_end[ind:length(aux_end)]
    mu1 <- test_level[ind+n]


  return(mu1)
}

ACER_Gumble_4_eta1 <- function(X, k, alpha,n)
{
  k_i <- k
  mu1 <- eta_detect_test(X,n)
  muf <- max(X)
  nb_points <- 100
  pas <- (muf - mu1)/(nb_points)
  test_level <- seq(mu1, muf, pas)

  aux <- CI_Poisson(X,mean(X),k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  aux_eps <- aux1
  print(aux)
  aux_w <- 1/(log(aux2[1]-aux2[2]))^2
  for(level in test_level)
  {
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    aux2 <- aux$CI
    v <- 1/(log(aux2[1] -aux2[2])^2)
```

```
    aux_eps <- c(aux_eps, aux1)
    aux_w <- c(aux_w, v)

  }
  aux_w <- aux_w[-1]
  aux_eps <- aux_eps[-1]



  # create the functions to optimize
  eps <- aux_eps[aux_eps != 0]
  w <- aux_w[aux_eps != 0]
  test_level <- test_level[aux_eps != 0]
  level <- test_level
  y <- log(eps)
  y_m <- sum(w*y) / sum(w)



  ACER_sim_vect <- function(param){
    b <- param[1]
    c <- param[2]

    x <- abs(level - b)^c
    x_m <- sum(w*x)/sum(w)
    a <- abs((sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2))))
    q <- exp(y_m + a*x_m)
    result <- as.double(w*(y-q+a*x)^2)
    return(result)
  }


  # ## we need to find b, c
  m <- min(X)
  minit <- (mu1+m)/2

  o <- nls.lm(c(minit, 1), lower = c(m, 1),
              upper = c(mu1, 5),ACER_sim_vect,  jac = NULL,
              nls.lm.control(ftol = 0.001, maxiter = 1000))

  param <- o$par
  #Now, we need to find a, q_ln
  b <- param[1]
  c <- param[2]
```

```
    x <- abs(level - b)^c
    x_m <- sum(w*x)/sum(w)
    a <- abs(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
    q <- exp(y_m + a*x_m)


    VaR <- b+(1/a*(abs(log(alpha/q))))^(1/c)

    return(list(q = q, a = a, b = b, c = c, VaR = VaR))
  }

  len <- 10
  ac <- rep(0,len)
  alpha <- 0.01

  # X <- rpareto(1000, 3, 1)
  # v <- ACER_Gumble_4(X,3,alpha)
  # print(v$VaR)
  X <- rpareto(1500, 3, 1)
  for(l in 1:len){

    for(i in 1:1){

      v <- ACER_Gumble_4_eta1(X,3,alpha,l)
      print(v)
      ac[l] <- ac[l] + v$VaR
    }
    ac[l] <- ac[l]/1
  }
  ac <- abs(ac - 10)/10

  x <- seq(1, len, 1)
  plot(x = x, y = ac, type = "l", col = "blue", xlab = "n",
       ylab = "Accuracy")
}


## test nb levels
test_nb_lvl <- function(){
    ACER_Gumble_4_lev <- function(X, k, alpha, nb_level)
    {
        k_i <- k
        mu1 <- eta_detect(X,k)
        muf <- max(X)
        nb_points <- nb_level
```

```
pas <- (muf - mu1)/(nb_points)
test_level <- seq(mu1, muf, pas)

aux <- CI_Poisson(X,mean(X),k_i)
aux1 <- aux$epsilon
aux2 <- aux$CI
aux_eps <- aux1
print(aux)
aux_w <- 1/(log(aux2[1]-aux2[2]))^2
for(level in test_level)
{
  aux <- CI_Poisson(X, level, k_i)
  aux1 <- aux$epsilon
  aux2 <- aux$CI
  v <- 1/(log(aux2[1] -aux2[2])^2)

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}
aux_w <- aux_w[-1]
aux_eps <- aux_eps[-1]



# create the functions to optimize
eps <- aux_eps[aux_eps != 0]
w <- aux_w[aux_eps != 0]
test_level <- test_level[aux_eps != 0]
level <- test_level
y <- log(eps)
y_m <- sum(w*y) / sum(w)



ACER_sim_vect <- function(param){
  b <- param[1]
  c <- param[2]

  x <- abs(level - b)^c
  x_m <- sum(w*x)/sum(w)
  a <- abs((sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2))))
  q <- exp(y_m + a*x_m)
  result <- as.double(w*(y-q+a*x)^2)
  return(result)
```

```
    }


    # ## we need to find b, c
    m <- min(X)
    minit <- (mu1+m)/2

    o <- nls.lm(c(minit, 1), lower = c(m, 1),
         upper = c(mu1, 5), ACER_sim_vect,  jac = NULL,
         nls.lm.control(ftol = 0.001, maxiter = 1000))

    param <- o$par
    #Now, we need to find a, q_ln
    b <- param[1]
    c <- param[2]
    x <- abs(level - b)^c
    x_m <- sum(w*x)/sum(w)
    a <- abs(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
    q <- exp(y_m + a*x_m)


    VaR <- b+(1/a*(abs(log(alpha/q))))^(1/c)

    return(list(q = q, a = a, b = b, c = c, VaR = VaR))
}

len <- 20
t <- rep(0, len)
ac <- rep(0,len)
X <- rpareto(2000, 3,1)
k <- 8
alpha <- 0.01
for(i in 1:len){
  t_aux <- Sys.time()
  v <- ACER_Gumble_4_lev(X,k,alpha, 50*len)
  t[i] <- Sys.time() - t_aux
  ac[i] <- abs(v$VaR - 10)/10
}
x <- seq(50, len*50, 50)
par(mar=c(4,4,3,5))
plot(x = x, y = ac, type = "l", col = "blue",
     xlab = "Number␣of␣levels", ylab = "Accuracy")

par(new=T)
plot(x = x, y = t, type = "l", col = "red", axes=F,
```

```
         xlab = "", ylab ="")
    mtext("Time",side=4, line = 2.5)
    axis(4)
    legend(x="topleft",legend=c("Accuracy","Time"),
           col=c("blue","red"), lty = c(1,1))
}



#### 10% test ####
test_ten_percentage <- function(){
      ## Calulation of k

      #This function f_k will count the number of different values
      # of the ACER functions between eta_1 and eta_f
      f_k<- function(X,k,eta1, etaf, alpha)
      {

        # aux <- CI_boot(X,mean(X),k, 50, alpha)
        aux <- CI_Poisson(X, mean(X),k)
        aux1 <- aux$epsilon
        aux_eps <- aux1
        nb_points <- 100
        pas <- abs(etaf - eta1)/(nb_points)

        test_level <- seq(eta1, etaf, pas)
        eps <- rep(0,nb_points)
        for(level in test_level)
        {
          # aux <- CI_boot(X, level,k, 100)
          aux <- CI_Poisson(X, level,k)

          aux1 <- aux$epsilon
          eps <- c(eps, aux1)
        }

        v_dif <- length(unique(eps[eps!=0]))

        return(v_dif)
      }

      #This function detect_k will choose a k to
      # get a value of k as small as possible without
      # having a lack of precision with long sequences
      #of constant values
```

```
detect_k <- function(X, eta1, etaf, alpha)
{
  N <- length(X)
  M <- 105
  m <- 5
  bi <- f_k(X,m, eta1, etaf, alpha)

  q <- floor(0.8*bi)
  while((M-m) > 5){
    m_aux <- floor((M+m)/2)
    v <- f_k(X,m_aux, eta1, etaf, alpha)
    if(v < q){
      M <- m_aux
    }
    else{
      m <- m_aux
    }
  }
  return(m)
}



## Calculation of etaf1, eta_f
#eta1 will represent the lowest level and eta_f the
#highest of our study
eta_detect <- function(X, k, alpha)
{
  nb_values <- 200
  pas <- (max(X)-min(X))/nb_values
  test_level <- seq(min(X),max(X),pas)
  i <- 1
  k_i <- k
  aux_end <- rep(0,k_i)
  for(level in test_level)
  {
    # aux <- CI_boot(X,level,k_i,100, alpha)
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    if(aux1>0){
      aux_end[i] <- (aux1)

    }
    else{
      aux_end[i] <- 0
```

```
      }
      i = i+1
    }
    m <- max(aux_end)
    ind <- max(which(aux_end == m))
    v <- aux_end[ind:length(aux_end)]
    mu1 <- test_level[ind]
    w <- diff(log(v))
    l <- which(w != 0)
    r <- diff(l)

    m <- min(which(r >8))
    p <- l[m]
    muf <- test_level[p]

    return(c(mu1, muf))
  }


  ## ACER method
  ACER_General <- function(X, alpha)
  {
    #First we fixe k at a possible value
    # and we find eta1 and eta_f. These two values
    #do not vary too much with k
    eta <- eta_detect(X, 3, alpha)
    eta1 <- eta[1]
    etaf <- eta[2]

    #With eta1 and eta_f, we find k.
    k <- detect_k(X, eta1, etaf, alpha)

    #We calculate then the true values of eta1 and eta_f
    eta <- eta_detect(X, k, alpha)
    eta1 <- eta[1]
    etaf <- eta[2]

    nb_points <- 200

    pas <- (etaf-eta1)/(nb_points)
    test_level <- seq(eta1, etaf, pas)
    aux_eps <- 0
    aux_w <- 0
    for(level in test_level)
    {
```

```
  # aux <- CI_boot(X, level,k, 100)
  aux <- CI_Poisson(X, level, k)
  aux1 <- aux$epsilon
  aux2 <- unlist(aux$CI)
  v <- 1/(log(abs(aux2[1])) -log(abs(aux2[2])))^2

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}

eps <- aux_eps[2:length(aux_eps)]
w <- aux_w[2:length(aux_w)]

level <- test_level

y <- log(eps)
# plot(w)
y_m <- sum(w*y) / sum(w)

ACER_sim_vect <- function(param){
  #we are creating now the function for the given k
  b <- param[2]
  c <- param[3]
  a_t <- param[1]
  x <- log(abs(1 + a_t*abs(level - b)^c))
  x_m <- sum(w*x)/sum(w)
  gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + gam*x_m)
  result <- rep(0, length(w))
  for(i in 1:length(w)){
    result[i] <- as.double(sqrt(w[i])*abs(y[i] - log(q)
                + gam*log(abs(1 + a_t*abs(level[i] - b)^c))))
  }
  return(unlist(result))
}
#
#
m <- min(X)
# we need to find b, c
minit <- (m+eta1)/2


o <- nls.lm(c(1,minit, 1), lower = c(0,m, 0),
            upper = c(Inf,eta1, 5),
```

```
                  ACER_sim_vect,  jac = NULL,
                  nls.lm.control(ftol = 0.001, maxiter = 1000))


  param <- o$par
  #Now, we need to find a, q_ln
  a_t <- param[1]
  b <- param[2]
  c <- param[3]
  x <- log((1 + a_t*abs(level - b)^c))
  x_m <- sum(w*x)/sum(w)

  gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))

  q <- exp((y_m + gam*x_m))



  lres <- rep(0, length(eps))

  for(l in 1:length(eps)){
    lres[l] <- log(q) - gam*log(1+a_t*abs(level[l] - b)^c)
  }
  return(list(coef = c(a_t, b, c, q, gam), eps = eps,
                  level = level))

}



## Calculation VaR
VaR <- function(X, alpha){
  v <- ACER_General(X, alpha)
  coef <- v$coef
  coef <- unlist(coef)
  a_t <- coef[1]
  b <- coef[2]
  c <- coef[3]
  q <- coef[4]
  gam <- coef[5]
  # VaR <- b+abs(1/a_t*abs(alpha/q)^(1/gam) - 1)^(1/c)
  VaR_new <- b + abs(((alpha/q)^(-1/gam)-1)/a_t)^(1/c)
  M <- max(X)
  if(is.na(VaR_new)){
    print("There is a problem with the optimization")
```

```
  }
  else{
    if(VaR_new > M){
      print("There␣is␣a␣problem␣with␣the␣optimization")
    }
  }
  return(VaR_new)
}

## DATA FRAME ##
l <- 20
VaR_E <- rep(0,l)
VaR_hist <- rep(0,l)
VaR_R <- rep(0,l)
VaR_MC <- rep(0,l)
VaR_ACER <- rep(0,l)
e_E <- rep(0,l)
e_hist <- rep(0,l)
e_R <- rep(0,l)
e_MC <- rep(0,l)
e_ACER <- rep(0,l)

for(i in 1:l){
  ## DATA  ##
  N <- 800
  X <- rpareto(N, 3, 1)
  D <- X[2:N] - X[1:(N-1)]


  ## Riskmetrics ##
  P <- log(X)
  D_R <- P[2:N] - P[1:(N-1)]
  VaR_R[i] <- mean(X)*1.645*sqrt(exp(var(D_R)))
  e_R[i] <- (VaR_R[i]-2.7)*100/2.7

  ## Econometric approach ##
  P <- log(X)
  G <- garch(P)
  coeff <- G$coef
  VaR_E[i] <- exp(sum(coeff * P[(N-length(coeff)+1) : N]))
              + 1.645*exp(var(P))
  e_E[i] <- (VaR_E[i]-2.7)*100/2.7

  ## Historical Method  ##
  D_hist <- sort(D)
```

```
  X_hist <- mean(X)+D_hist
  VaR_hist[i] <- X_hist[0.95*N]
  e_hist[i] <- (VaR_hist[i]-2.7)*100/2.7

  ## MC Method  ##
  D_MC <- D
  Sigm <- var(D_MC)

  l_r <- 10000
  r <- runif(l_r, 0, 1)

  X_MC <- X[N]  + Sigm * r
  X_MC <- sort(X_MC)
  VaR_MC[i] <- X_MC[0.95*l_r]
  e_MC[i] <- (VaR_MC[i]-2.7)*100/2.7

  ## ACER method ##
  VaR_ACER[i] <- VaR(X,0.05)
  e_ACER[i] <- (VaR_ACER[i]-2.7)*100/2.7

}


dt <- data.frame(Riskmetric = e_R, Econometric = e_E,
         Historical = e_hist, MC = e_MC, ACER = e_ACER)

w <- formattable(dt)
sign_formatter <- formatter("span",
      style = x ~ style(color = ifelse(x > 10, "red",
      ifelse(x < -10 | x == "NaN", "orange", "green"))))
w <- formattable(dt, list(Riskmetric = sign_formatter,
     Econometric =sign_formatter, Historical =sign_formatter,
     MC =sign_formatter, ACER =sign_formatter))

export_formattable <- function(f, file, width = "100%",
                     height = NULL,
                     background = "white", delay = 0.2)
{
  w <- as.htmlwidget(f, width = width, height = height)
  path <- html_print(w, background = background, viewer = NULL)
  url <- paste0("file:///", gsub("\\\\", "/",
                             normalizePath(path)))
  webshot(url,
          file = file,
          selector = ".formattable_widget",
```

```
                delay = delay)
      }
      export_formattable(w,
      "C:/Users/Mala/Desktop/Master's_thesis/Latex/fig/w2.png")
}


#### test soge ####

#This function f_k will count the number of different values
# of the ACER functions between eta_1 and eta_f
f_k<- function(X,k,eta1, etaf, alpha)
{

  # aux <- CI_boot(X,mean(X),k, 50, alpha)
  aux <- CI_Poisson(X, mean(X),k)
  aux1 <- aux$epsilon
  aux_eps <- aux1
  nb_points <- 100
  pas <- abs(etaf - eta1)/(nb_points)
  if(eta1 >= etaf){
    etaf <- 2*eta1
  }
  test_level <- seq(eta1, etaf, pas)
  eps <- rep(0,nb_points)
  for(level in test_level)
  {
    # aux <- CI_boot(X, level,k, 100)
    aux <- CI_Poisson(X, level,k)

    aux1 <- aux$epsilon
    eps <- c(eps, aux1)
  }

  v_dif <- length(unique(eps[eps!=0]))

  return(v_dif)
}

#This function detect_k will choose a k to
# get a value of k as small as possible without
# having a lack of precision with long sequences
#of constant values
detect_k <- function(X, eta1, etaf, alpha)
{
```

```
  N <- length(X)
  M <- 105
  m <- 5
  bi <- f_k(X,m, eta1, etaf, alpha)

  q <- floor(0.8*bi)
  while((M-m) > 5){
    m_aux <- floor((M+m)/2)
    v <- f_k(X,m_aux, eta1, etaf, alpha)
    if(v < q){
      M <- m_aux
    }
    else{
      m <- m_aux
    }
  }
  return(m)
}



## Calculation of etaf1, eta_f
#eta1 will represent the lowest level and eta_f the
#highest of our study
eta_detect <- function(X, k, alpha)
{
  nb_values <- 200
  pas <- (max(X)-min(X))/nb_values
  test_level <- seq(min(X),max(X),pas)
  i <- 1
  k_i <- k
  aux_end <- rep(0,k_i)
  for(level in test_level)
  {
    # aux <- CI_boot(X,level,k_i,100, alpha)
    aux <- CI_Poisson(X, level, k_i)
    aux1 <- aux$epsilon
    if(aux1>0){
      aux_end[i] <- (aux1)

    }
    else{
      aux_end[i] <- 0
    }
    i = i+1
```

```
  }
  m <- max(aux_end)
  ind <- max(which(aux_end == m))
  v <- aux_end[ind:length(aux_end)]
  mu1 <- test_level[ind]
  w <- diff(log(v))
  l <- which(w != 0)
  r <- diff(l)

  m <- min(which(r >8))
  p <- l[m]
  muf <- test_level[p]
  if(muf <= mu1){
    muf <- 2*mu1
  }
  return(c(mu1, muf))
}

## ACER method
ACER_General <- function(X, alpha)
{
  #First we fixe k at a possible value
  # and we find eta1 and eta_f. These two values
  #do not vary too much with k
  eta <- eta_detect(X, 3, alpha)
  eta1 <- eta[1]
  etaf <- eta[2]

  #With eta1 and eta_f, we find k.
  k <- detect_k(X, eta1, etaf, alpha)

  #We calculate then the true values of eta1 and eta_f
  eta <- eta_detect(X, k, alpha)
  eta1 <- eta[1]
  etaf <- eta[2]


  nb_points <- 200
  if(eta1 == etaf){
    etaf <- 2*eta1
  }
  pas <- (etaf-eta1)/(nb_points)
  test_level <- seq(eta1, etaf, pas)
  aux_eps <- 0
  aux_w <- 0
```

```
for(level in test_level)
{
  # aux <- CI_boot(X, level,k, 100)
  aux <- CI_Poisson(X, level, k)
  aux1 <- aux$epsilon
  aux2 <- unlist(aux$CI)
  v <- 1/(log(abs(aux2[1])) -log(abs(aux2[2])))^2

  aux_eps <- c(aux_eps, aux1)
  aux_w <- c(aux_w, v)

}

eps <- aux_eps[2:length(aux_eps)]
w <- aux_w[2:length(aux_w)]

level <- test_level

y <- log(eps)
y_m <- sum(w*y) / sum(w)

ACER_sim_vect <- function(param){
  #we are creating now the function for the given k
  b <- param[2]
  c <- param[3]
  a_t <- param[1]
  x <- log(abs(1 + a_t*abs(level - b)^c))
  x_m <- sum(w*x)/sum(w)
  gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))
  q <- exp(y_m + gam*x_m)
  result <- rep(0, length(w))
  for(i in 1:length(w)){
    result[i] <- as.double(sqrt(w[i])*abs(y[i] - log(q)
                  + gam*log(abs(1 + a_t*abs(level[i] - b)^c))))
  }
  return(unlist(result))
}
#
#
m <- min(X)
# we need to find b, c
minit <- (m+eta1)/2


o <- nls.lm(c(1,minit, 1), lower = c(0,m, 0),
```

```
              upper = c(Inf,eta1, 5), ACER_sim_vect,
              jac = NULL,
              nls.lm.control(ftol = 0.001, maxiter = 1000))


  param <- o$par
  #Now, we need to find a, q_ln
  a_t <- param[1]
  b <- param[2]
  c <- param[3]
  x <- log((1 + a_t*abs(level - b)^c))
  x_m <- sum(w*x)/sum(w)

  gam <- -(sum(w*(x-x_m)*(y-y_m))/sum((w*(x-x_m)^2)))

  q <- exp((y_m + gam*x_m))



  lres <- rep(0, length(eps))

  for(l in 1:length(eps)){
    lres[l] <- log(q) - gam*log(1+a_t*abs(level[l] - b)^c)
  }
  return(list(coef = c(a_t, b, c, q, gam), eps = eps,
              level = level))

}

## Calculation VaR
VaR <- function(X, alpha){
  v <- ACER_General(X, alpha)
  coef <- v$coef
  coef <- unlist(coef)
  a_t <- coef[1]
  b <- coef[2]
  c <- coef[3]
  q <- coef[4]
  gam <- coef[5]
  # VaR <- b+abs(1/a_t*abs(alpha/q)^(1/gam) - 1)^(1/c)
  VaR_new <- b + abs(((alpha/q)^(-1/gam)-1)/a_t)^(1/c)
  M <- max(X)
  if(is.na(VaR_new)){
    print("There␣is␣a␣problem␣with␣the␣optimization")
  }
```

```
  else{
    if(VaR_new > M){
      print("There␣is␣a␣problem␣with␣the␣optimization")
    }
  }
  return(VaR_new)
}

### Every day
options(digits = 6)
data<-read.table(
  "C:/Users/Mala/Desktop/Master's␣thesis/R/final/quotes.csv",
  sep=";")
op_a <- levels(data$V2)
op <- op_a[3:(length(op_a)-5)]
op <- sub(",", ".", op)
op <- as.numeric(op)
soge_test_everyday <- function(op){
  D <- diff(op)
  pts <- 1000
  nb <- 30
  frc <- rep(0,nb)
  start <- 1100
  frc[1] <- op[start+1]
  for(i in 2:nb){
    u <- VaR(D[(start+i-pts):(start+i)], 0.05)
    D[start+i+1] <- u
    frc[i] <- frc[i-1] + u
  }
  plot(op[(start+1):(start+nb)], ylim = c(44.6, 48),
       ylab = "Values␣of␣the␣opening", xlab = "Days")
  lines(frc, col = "red")
}

soge_test_week <- function(op){
  aux <- seq(5,1405,5) #281 values
  D <- diff(op[aux])
  nb <- 7
  frc <- rep(0,nb)
  frc[1] <- op[1000]
  for(i in 2:nb){
    u <- VaR(D[i:(150+i)], 0.05)
    D[start+i+1] <- u
    frc[i] <- frc[i-1] + u
  }
```

```
  aux <- seq(1000, 1000+(nb-1)*5,5)
  plot(x = aux, op[aux], ylim = c(43,45),
       ylab = "Values␣of␣the␣opening", xlab = "Days")
  lines(aux,frc, col = "red")

}
```