Scenario-Based Portfolio Optimization

Edouard Berthe
The University of Queensland
4 PUBLICATIONS **2** CITATIONS

Advanced Topics in Operations Research
Scenario-Based Portfolio Optimization

Edouard BERTHE
Student 43966813
berthe.ed@gmail.com

Friday 21th October, 2016

Contents

Introduction

Portfolio management is certainly the most important part of the work of a Bank or a Hedge Fund. If we are given a set of tradable assets - it can be simple stocks but also bonds or any derivative product as Call / Put option, Swaps, etc. - what is the best way to spread the money we want to invest into these assets? The first question is to clearly define “best way”. There are really a lot of different entities playing in the financial sphere (of course Banks, but also Hedge Fund, Investment funds, ...) and certainly as much of different types of investment.

Some of them will expect a return by buying “information” which will give them an informative advantage on the other ones (for instance a new way to estimate a company’s market value, or a news about the close firing of a member of the executive commity, etc.). Other - the “high frequency trading” entities - make a return by developping high level algorithm and mathematical analysis on the stock price to be able to make a very small return, but with a very high probability.

Other again, mainly Hedge Funds, try to have a return on investment by choosing correlated stocks and this way lowering the volatility of their portfolio.

Chapter 1

Portfolio Management

1.1 General Scheme

We consider a set of assets $\{A_i\}_{i \in N}$ indexed by a discrete set N. These assets can be stocks, bonds, derivative products (call/put options, swaps, futures...).
Each asset A_i ($i \in N$) has a price at time t P_t^i, which is consider as a **random variable**, taking its value in $\mathbb{R}_+$. Therefore, the processes $\{P^i\}_{t \in T}$ are random processes. In this paper, we consider only **one price a day**, and we based our study on the **closing prices**. This choice is more meaningful than the opening price, because on the morning, before the opening of the market, a lot of orders are **stacked** in the orderbook, and are resolved at the exact time of opening, which can lead to high volatility during the first minutes. The closing prices, on the contrary, are the result of a stabilization process which occur during the last minutes.
Each asset A_i generates therefore a **stochastic daily return $\boldsymbol{R_i}$**, computes as:

$$R_t^i = \frac{P_t^i - P_{t-1}^i}{P_{t-1}^i} = \frac{P_t^i}{P_{t-1}^i} - 1$$

This definition makes sense, because in our case P_t^i is the **closing price on day $\boldsymbol{t}$**. Therefore, R_t^i represents the return between the previous closing price and the current closing price, which is actually exactly the performance of A_i during day t.
Of course, none can foresee the future price of an asset: none can now the **true mathematical model** that each asset's price (and so each asset's return) follows (if such a model exists!). Therefore, none can know the **true** mean return, or the **true** volatility. However, we can compute the **empirical mean and volatility** of each asset, from the historical data. We then have a set of different available assets, characterized by their **mean return** and their **volatility**. The case of the French CAC40 is shown on Figure 1.1.

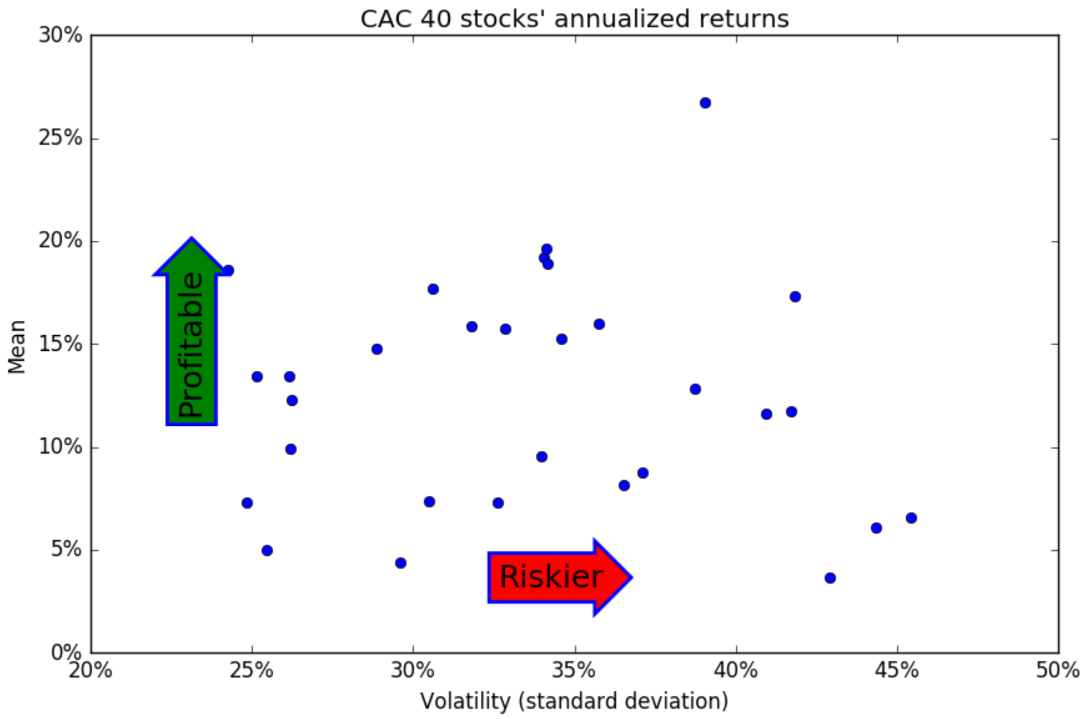

Figure 1.1: French CAC 40 returns

The goal of the investor is to **build a portfolio** θ which consists of money invested in some of the assets $\{A_i\}_{i \in N}$. For sake of simplicity, and without loss of generality, we does not take account of the total amount of money the investor has to invest, but only **the repartition** (in percentage) of this money invested in each asset. For each asset A_i, we then define $w_i \in [0, 1]$ as the **weight** of the portfolio invested in asset A_i.

Depending of his investment's strategy, the investor may want to:

- to **minimize his portfolio's risk**, subject to a **minimal expected return**

or

- to **maximize his portfolio's expected return**, subject to a **maximal risk**

In this paper, we focus on **the first scheme**, because for many financial actors, it makes more sense to be able to generate a minimum **required rate of return** asked by the investors, and then to be **as close as possible to this objective**.
Thus, the objective is to find the set of weights $\{w_i\}_{i \in N}$ (it will be our variables) to reach this objective.
In this section, we assume that we can only "be long" an asset, which means that we can only **buy** the asset. This is almost always the case for classical Investment Fund, or Venture Capital funds, which generate a return from the development of companies.
However, in the next section, we will add the possibility to "short" a stock, meaning **selling it without possessing it** (actually, this implies to deliver it later). In reality, this is almost always the case for Hedge Funds, which base their investment strategy by covering their long positions with negatively correlated short positions. In this case, the **leverage** L is the difference between the Long positions and the Short positions, and is defined by the investor's strategy.

1.2 Additional Constraints

Very often, the financial actors want to keep a control over:

- the **number of assets** they are **actually** invested in.
- the **minimum** investment in each assets. This constraint is explained by the fact that the market is not **perfect**, and the existence of **transaction taxes** can prevent the investor to invest less than a given percentage $W_{\min}$ in each asset.
- the **maximum** investment in each asset. This constraint may seem useless, because of the principle of **diversification** (the algorithm will naturally spread the portfolio to minimize its risk). However, it can be useful when the above constraint (on the maximum number of assets) is added, because that last tend to increase the weight in each asset.

It appears that any of these constraint can be reached by the addition of **binary variables**:

$$x_i \in \{0, 1\} \quad \forall i \in N$$

were, for any $i \in N$, X_i representing the fact that A_i is **actually** in the portfolio (i.e $W_i > 0$).
We obtain the following **General Portfolio Framework**:

General Portfolio Framework

$$\begin{aligned}
\mu_{\min} &\leq \mu && && (1.1)\\
w_i &\leq x_i && \forall i \in N && (1.2)\\
W_{\min} x_i &\leq w_i && \forall i \in N && (1.3)\\
\sum_{i \in N} x_i &\leq N_{\max} && \forall i \in N && (1.4)\\
\\
w_i &\in [0, W_{\max}] && \forall i \in N\\
x_i &\in \{0, 1\} && \forall i \in N
\end{aligned}$$

Were the constraint (1.2) assures the definition of x_i ($x_i = 1$ if $w_i > 0$), the constraint (1.3) assures the minimum investment in each asset actually in the portfolio, and the constraint (1.4) enforces the maximum number of assets.
It is important to stress that **the mean return of the portfolio** $\boldsymbol{\mu}$ is not defined here, but will be in the following models.
The `PortfolioModel` Python class, extending `gurobipy.Model`, is implemented in `entities/models.py`.
We now present the different model applying and extending this framework.

Chapter 2

Markowitz portofolio

2.1 Presentation of the model

The returns of each asset A_i have a **mean** (the mathematical expectation) and a **volatility** (the standard deviation). From a mathematical point of view, if θ is a portfolio composed of these N stocks, then the return of θ R^θ is itself a random variable, as a linear combination of the stocks' returns. By linearity of the expectation, we know that the mean return of the portfolio is the linear combination of the mean returns of the stocks :

$$\theta = \sum_{i=1}^{S} x_i S_i \Rightarrow \mathbb{E}\left[R^\theta\right] = \sum_{i=1}^{S} x_i \mathbb{E}\left[R^i\right]$$

However, **the variance is not linear**. Therefore, the volatility of the portfolio **is not** the linear combination of the volatility of the stocks:

$$\operatorname{Var}\left(R^\theta\right) = \sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j \sigma_{i,j} = \sum_{i=1}^{N} x_i^2 \sigma_i^2 + 2 \sum_{\substack{i,j=1 \\ i<j}}^{N} x_i x_j \sigma_{i,j}$$

where $\sigma_i^2 = \operatorname{Var}\left(R^i\right)$ and $\sigma_{i,j} = \operatorname{Cov}\left(R^i, R^j\right)$ are the elements of the variance-covariance matrix of the returns. The **Markowitz Model** is then an optimization based on the **minimization of this variance**. We obtain the following scheme:

Markowitz Model

$$\min \quad \sum_{i\in N} w_i^2 \sigma_i^2 + 2 \sum_{\substack{j\in N \\ i<j}} w_i w_j \sigma_{i,j}$$

$$\mu = \sum_{i\in N} \mu_i w_i$$

+ General Portfolio Framework

2.2 The efficient frontier

In optimization of Portfolio Management, the notion of **efficient frontier** is an essential concept, used a lot by all financial entities. Indeed, different investors will always adopt different investment strategies, in order to realize their investment objective. The financial actors will mostly be characterized by their **aversion**

to risk: will their require a higher return with high risk, or less important returns but less riskier? The **efficient frontier** represents the set of all the points representing **a possible optimal portfolio**, given the amount of risk the investor is ready to bear, or the minimal return he requires. In fact, it is **the optimal solution to the Markowitz Portfolio**.
To understand this notion, let us begin with an introductory example: let us suppose that we only have **2 stocks available** for our portfolio. In this case, our portfolio is characterized by a coefficient $\lambda \in [0,1]$ such that:

$$\theta = \lambda S^1 + (1-\lambda)S^2$$

In this case, the return of our portfolio is:

$$\mathbb{E}\left[R^\theta\right] = \lambda\mathbb{E}\left[R^1\right] + (1-\lambda)\mathbb{E}\left[S^2\right] \tag{2.1}$$

and the variance (the volatility squared) is:

$$(\sigma_\theta)^2 = \lambda^2\sigma_1^2 + (1-\lambda)^2\sigma_2^2 + \lambda(1-\lambda)\sigma_{1,2} \tag{2.2}$$

Using 2.1 and 2.2 we can obtain the variance in term of the return:

$$\sigma_\theta^2 = R_\theta^2\frac{\sigma_1^2+\sigma_2^2-\sigma_{1,2}}{(r_2-r_1)^2} + R_\theta\frac{\sigma_{1,2}(r_1+r_2)-2r_2\sigma_1^2-2r_1\sigma_2^2}{(r_2-r_1)^2} + \frac{r_1^2\sigma_2^2+r_2^2\sigma_1^2-r_1r_2\sigma_{1,2}}{(r_2-r_1)^2}$$

So **the variance of the portfolio is a parabola with respect to the return**, which can be illustrated on figure 2.1.

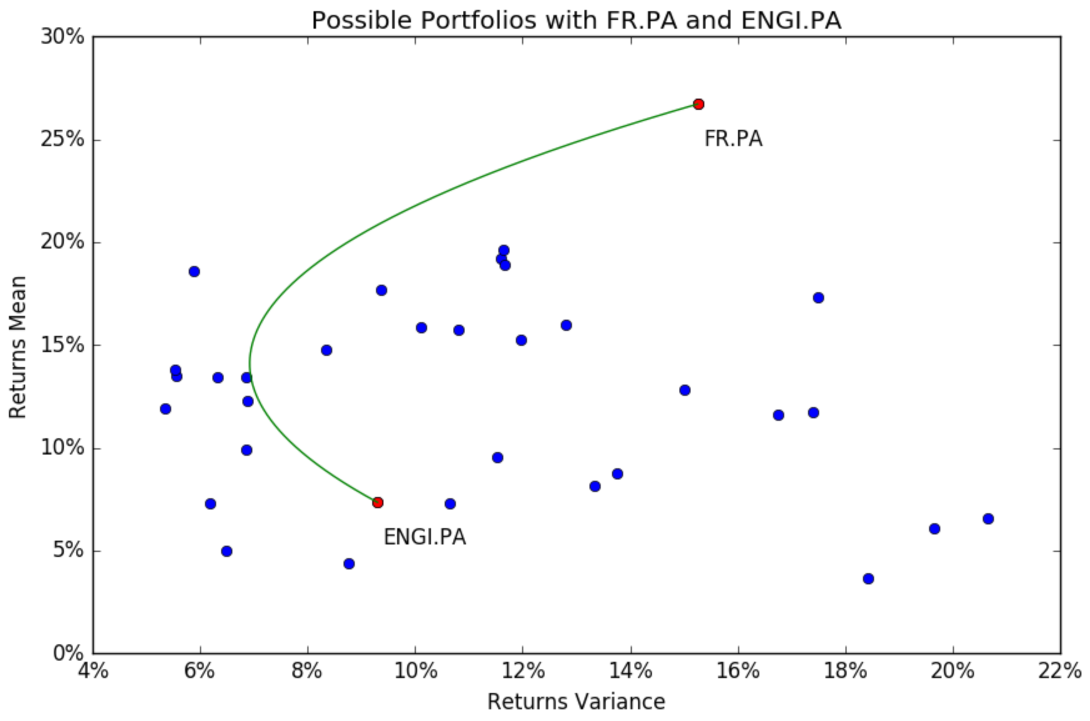

Figure 2.1: Efficient Frontier in closed-form - 2 Stocks

We can also extend this with 3 stocks. Now, we cannot find a closed-form solution for the optimal portfolio consisting of the 3 stocks. All we can do is computing the 3 possible "2-stocks" portfolios. Of course, the real efficient frontier can be computed (**really**) efficiently by Gurobi. The result can be seen in the Figure 2.2.

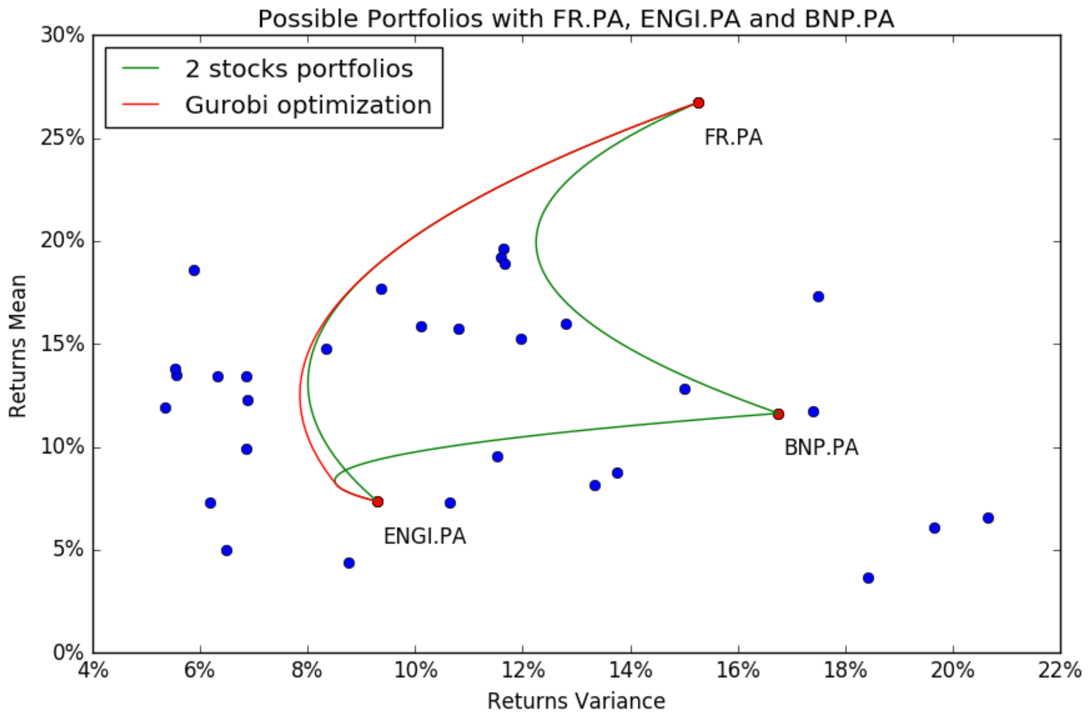

Figure 2.2: Efficient Frontier - 3 Stocks

Eventually, the efficient frontier of the french CAC4 is displayed in Figure 2.3.

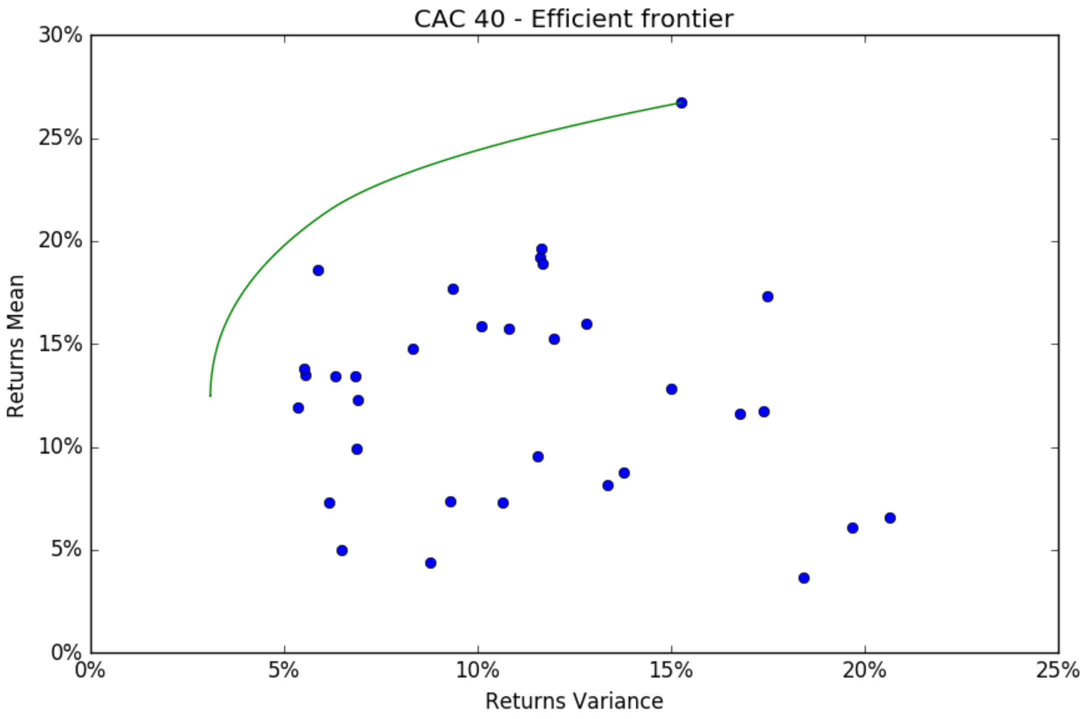

Figure 2.3: Efficient Frontier - CAC 40

Chapter 3

Scenarios-Based Linear Portfolio

It is important to stress that the Markowitz portfolio model has some issues:

- **Quadratic objective value**: if it used to be a real problem for the solver to handle a quadratic objective function, it is not really the case anymore, because of the very efficient improvements which has been brought to solvers.
- **Inclusion of external information**. There is not any investment strategy *only* based on a pure analysis of the historical data. A real investment strategy include a lot of other external information, the simplest example being an analysis of the news. Furthermore, some financial actors spend huge amounts of money to be subscribed to **financial mailing lists**, or buy **very** expensive reports from financial analysts and specialists of each sectors / companies. All this data cannot be easily included in optimization algorithms such as Markowitz Portfolio.
- **Handling of the extreme cases**: in the Markowitz model, we try to minimize the **variance** of the portfolio's return, which is in fact its second moment. The **extreme events**, as **big losses**, are not very rare in reality, but **they are swallowed by the computation of the variance-covariance matrix**. Therefore, the Markowitz portfolio **is not robust to outliers**, which is a **more and more primordial characteristic** we ask to portfolio model, mostly after the 2008 big crisis.

In this section, we present a totally different approach of portfolio management: **scenarios-based portfolio models**.

3.1 Introduction

We now suppose that there are **a finite number of possible scenarios** which can occur. Each scenario can occur with a given **probability**, and in each one of these scenarios, each stock has a possible return. An example of such a set of scenarios is presented in Table 3.1.

Asset	Scenario 1	Scenario 2	Scenario 3	Scenario 4	μ_i
	25 %	25 %	10 %	40 %	
A_1	1.2 %	4.5 %	- 0.4 %	2.3 %	2.3 %
A_2	2.4 %	5.3 %	- 3.2 %	0.4 %	1.7 %
A_3	0.5 %	0.1 %	0.9 %	1.8 %	2.1 %

Table 3.1: Example with 3 stocks and 4 scenarios

We then add to our **sets** the set of the scenarios $\{S_t\}_{t\in T}$ indexed by T.
Furthermore, we now have the following additional **data**:

Return of A_i in scenario S_t	$r_{i,t}$	$\forall i \in N \quad \forall t \in T$
Probability of S_t	p_t	$\forall t \in T$

So, in this framework, the mean return μ_i of each asset A_i becomes:

$$\mu_i = \sum_{t\in T} p_t r_{i,t} \quad \forall i \in N$$

And we will use the following **intermediate variables** ('LinExpr' in gurobipy) representing the return of the portfolio in scenario S_t, and depending on our variables $\boldsymbol{w} = \{w_i\}_{i\in N}$:

$$y_t(\boldsymbol{w}) = \sum_{i\in N} r_{i,t} w_i \tag{3.1}$$

The global mean return of the portfolio is therefore defined by:

$$\mu(\boldsymbol{w}) = \sum_{t\in T} p_t y_t(\boldsymbol{w}) = \sum_{t\in T}\sum_{i\in N} p_t r_{i,t} w_i = \sum_{i\in N} \mu_i w_i \tag{3.2}$$

where the last expression shows in a better way the link with the variables $\boldsymbol{w}$.
The **`LinearPortfolioModel`** Python class, extending **`PortfolioModel`**, can be found in **`linear/models/model.py`**.

3.2 Risk Measures

3.2.1 The Mean Absolution Deviation: MAD

The **Mean Absolute Deviation** is a dispersion measure which has the same role as the volatility of the portfolio in the last section (which can here be computed without the variance-covariance matrix). Actually, the *interaction* between the stocks, i.e the *covariance* of their returns, is **already computed in the different scenarios**.
Indeed, if we won the linear objective value, then we still need a way to compute the different scenarios (which will talk about later). The MAD is defined as:

$$\delta(\theta) = \mathbb{E}\left[\ \left|R^\theta - \mathbb{E}\left[R^\theta\right]\right|\ \right] = \mathbb{E}\left[\ \left|\sum_{i=1}^{N} R_i w_i - \mathbb{E}\left[\sum_{j=1}^{N} R_j w_j\right]\right|\ \right]$$

Contrary to the variance of the portfolio (and then contrary to the volatility, which is the square root of the variance) which considers the **square** of the distances to the mean, the MAD considers the **absolute distances** to the mean, which makes it linearly computable.
Indeed, even if we have an absolute value in our objective:

$$\min \sum_{t\in T} p_t \left|y_t - \mu(\boldsymbol{w})\right|$$

we know how to overcome this problem. Let us define the variables d_t ($t \in T$) representing the deviation in scenario t, that is:

$$\forall t \in T \quad d_t = |y_t - \mu(\boldsymbol{w})|$$

Then we can define the MAD Minimization Model as:

MAD Model

$$\min_{\mathbf{w},\ \mathbf{d}} \quad \sum_{t \in T} p_t d_t$$

$$d_t \geq y_t - \mu \qquad \forall t \in T \qquad (3.3)$$

$$d_t \geq -(y_t - \mu) \qquad \forall t \in T \qquad (3.4)$$

$$y_t = \sum_{i \in N} r_{j,t} w_j \qquad \forall t \in T$$

$$d_t \geq 0 \qquad \forall t \in T$$

+ General Portfolio Framework

The `MAD` Python class can be found in **`linear/models/risk/mad.py`**.

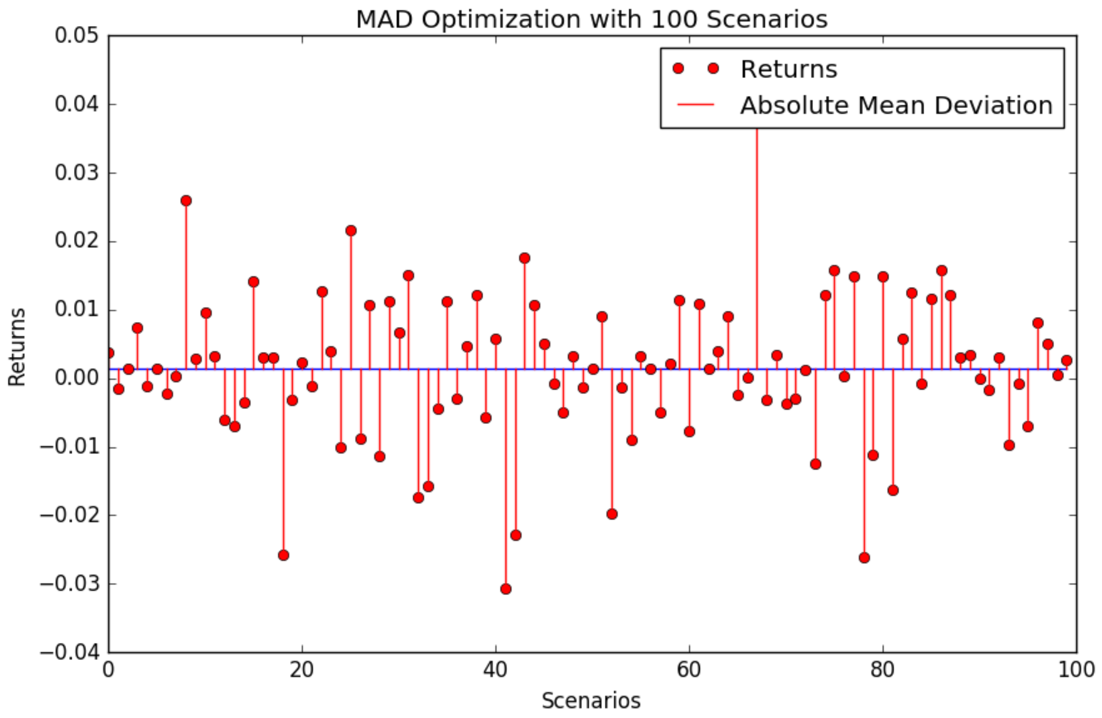

Figure 3.1: Mean Absolute Deviation Optimization

3.2.2 Semi-MAD

The MAD model measures the absolute differences of the returns in each scenario with the mean return. We could think that when the return of the scenario is "above" the mean return, then we do not want to minimize it, but we would only want to minimize the difference of the returns of the scenarios which are **below** the mean return.
Therefore, in this case, our problem would become:

Semi-MAD Model

$$\min_{\mathbf{w},\ \mathbf{d}} \quad \sum_{t \in T} p_t d_t$$

$$d_t \geq -(y_t - \mu) \qquad \forall t \in T \qquad (3.5)$$

$$y_t = \sum_{i \in N} r_{j,t} w_j \qquad \forall t \in T$$

$$d_t \geq 0 \qquad \forall t \in T$$

+ General Portfolio Framework

The `SemiMAD` Python class can be found in `linear/models/risk/semi_mad.py`.
However, as stated in [1], this will lead to **the exactly same portfolio**. Indeed, the mathematical formulation of the mean return implies that there is **as much absolute deviation below and above it** among the different scenarios. It is why the **Semi-MAD is half of the MAD**. Therefore, we will obtain the same weights, and half of the objective value of the MAD.
We can check this result with French CAC40.

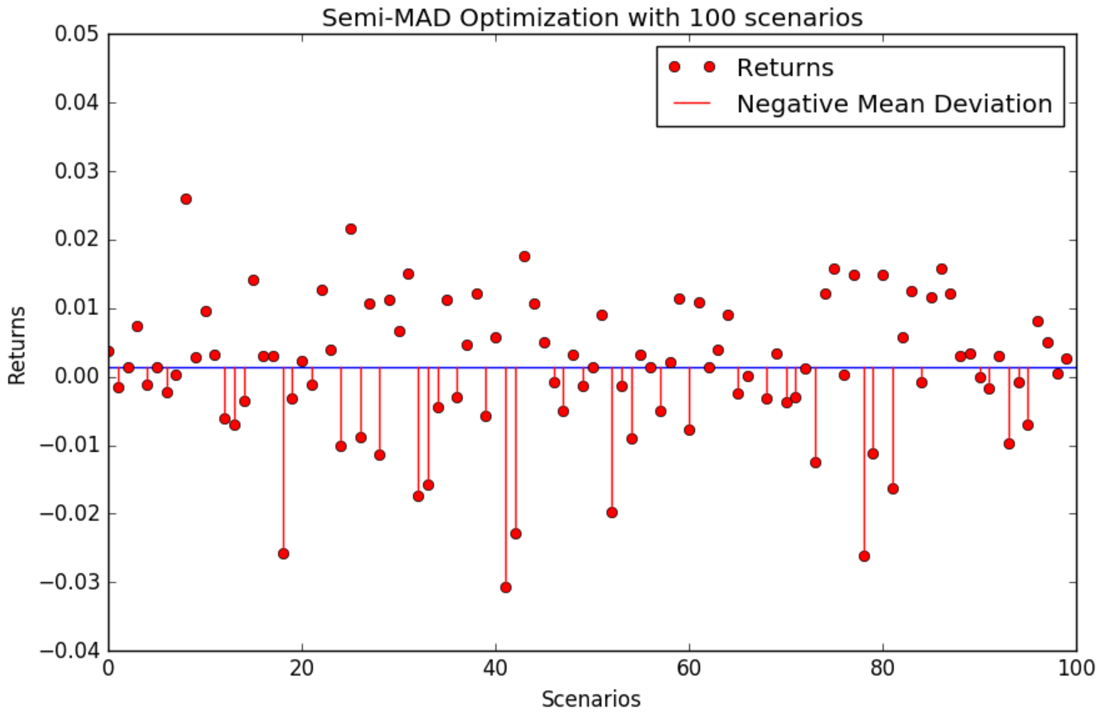

Figure 3.2: Mean Absolute Deviation Optimization

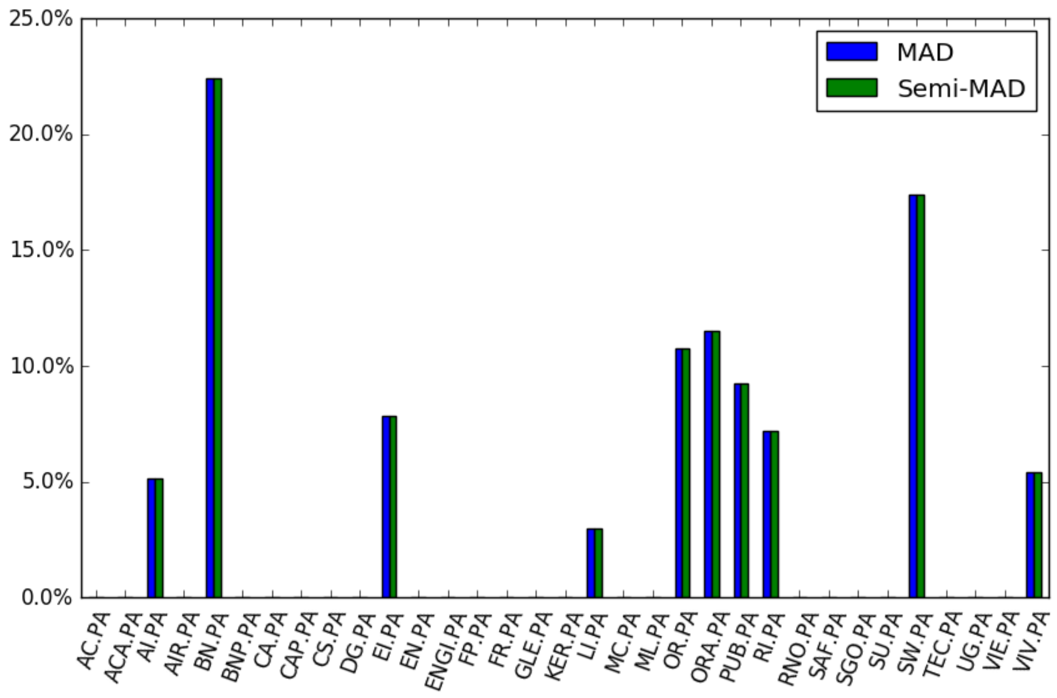

Figure 3.3: CAC40: MAD vs. SemiMAD

Furthermore, we can see on Figure 3.4 that the Semi-MAD is **faster to compute** than the MAD. Therefore, we will only make use of the Semi-MAD from now on.

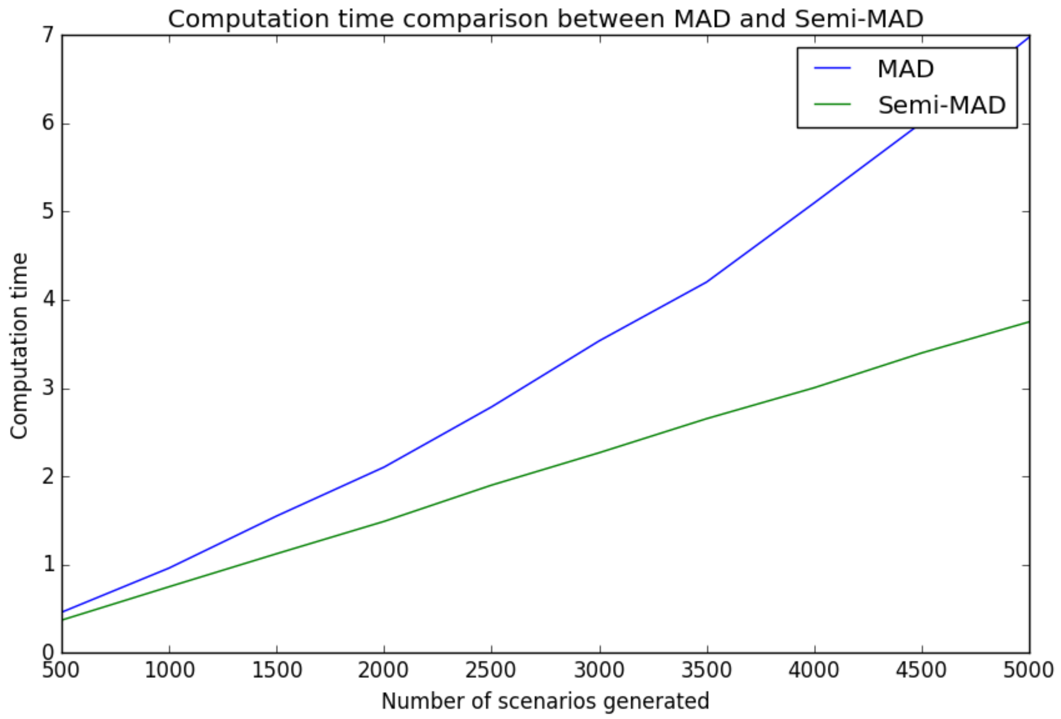

Figure 3.4: MAD and Semi-MAD computation time comparison

3.2.3 Gini's Difference

We have seen that the MAD and the Semi-MAD minimization led to the same portfolio. Another measure has been proposed more recently, called the **Gini's Mean Difference (GMD)**.
Instead of computing the absolute value of **the deviations from the mean return** of the portfolio and minimizing their sum, the idea is here to compute the absolute values of the **deviations between different scenarios**. More precisely, the absolute value of the deviation between the returns in two different scenarios t_1 and t_2 is:

$$|y_{t_1} - y_{t_2}|$$

By weighting these deviations by the probability of occurence of each scenarios, we obtain the following objective value:

$$\min \frac{1}{2} \sum_{t_1 \in T} \sum_{\substack{t_2 \in T \\ t_2 \neq t_1}} |y_{t_1} - y_{t_2}| \, p_{t_1} p_{t_2}$$

where the factor $\frac{1}{2}$ comes from the fact that we count each deviation twice in the double sum, or:

$$\min \sum_{t_1 \in T} \sum_{\substack{t_2 \in T \\ t_1 < t_2}} |y_{t_1} - y_{t_2}| \, p_{t_1} p_{t_2}$$

GMD Model

$$\min \quad \sum_{t_1 \in T} \sum_{\substack{t_2 \in T \\ t_1 < t_2}} d_{t_1,t_2} p_{t_1} p_{t_2}$$

$$d_{t_1,t_2} \geq y_{t_2} - y_{t_1} \quad \forall (t_1,t_2) \in T^2 \quad t_1 < t_2 \tag{3.6}$$

$$d_{t_1,t_2} \geq -(y_{t_2} - y_{t_1}) \quad \forall (t_1,t_2) \in T^2 \quad t_1 < t_2 \tag{3.7}$$

$$y_t = \sum_{i \in N} r_{i,t} w_i \quad \forall t \in T$$

$$d_{t_1,t_2} \geq 0 \quad \forall (t_1,t_2) \in T^2 \quad t_1 < t_2$$

+ General Portfolio Framework

The **GMD** Python class can be found in **`linear/models/risk/gmd.py`**.

Some computations and output comparisons of the three risk measures with different parameters are shown in the following Figures 3.5, 3.6 and 3.7.

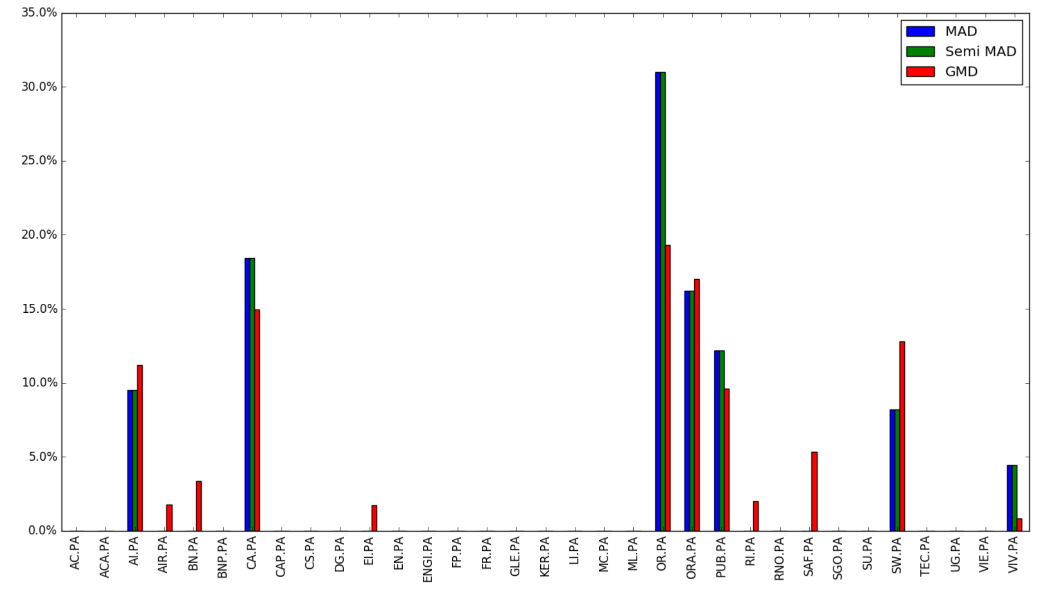

Figure 3.5: CAC40: MAD, SemiMAD and GMD, 100 scenarios

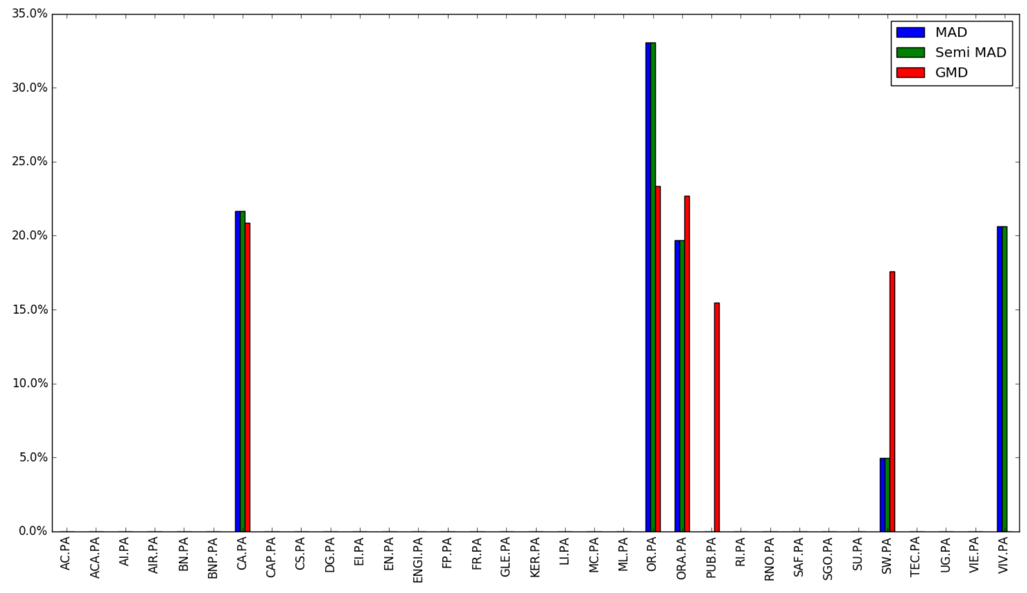

Figure 3.6: CAC40: MAD, SemiMAD and GMD, 100 scenarios, max 5 stocks

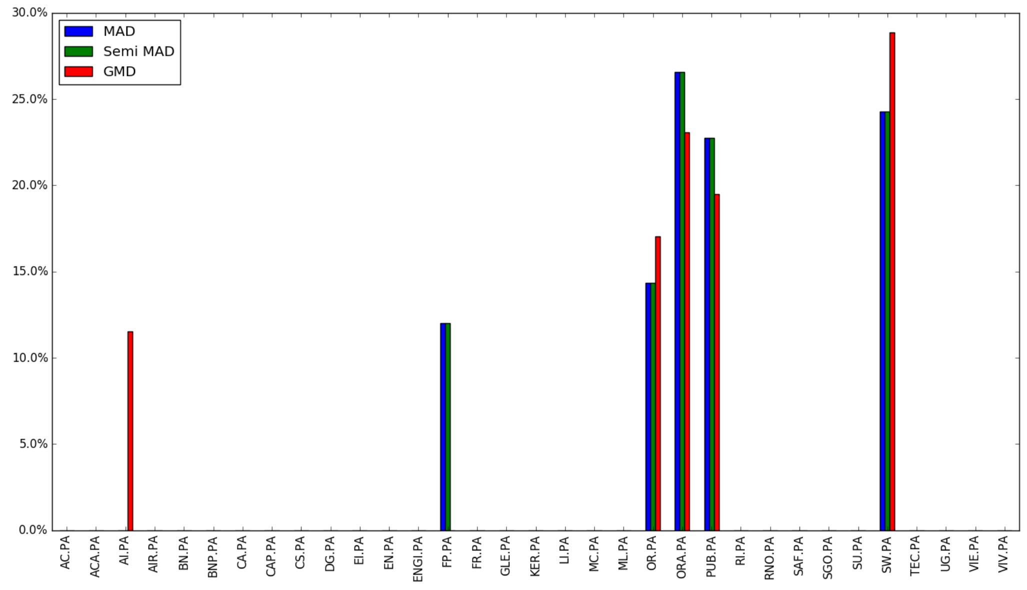

Figure 3.7: CAC40: MAD, SemiMAD and GMD, 200 scenarios, max 5 stocks

3.3 Safety Measures

In all of the risk measures we have seen in the last section, given a required (minimum) rate of return $\mu_{\min}$, we minimize the **dispersion** of the return around their mean value. The goal is then to have the most "stable" return as possible.
However, we could also want to be interested in another kind of measures: the control of the **worst cases**. This goal is **primordial** for financial entities, which are looking for more and more robust models against extreme scenarios, mostly since the last 2008 financial crisis.
In this section, we present different measures controlling the worst cases, or **safety measures**, from the simplest to the most general.

3.3.1 Minimax

To begin with the most simple possibility, the investor may want to have a control over **the** worst of the scenarios, and want to maximize it. This way, the investor is sure to have a final return which is **at least this minimum**. Given a portfolio $\boldsymbol{w}$, the minimum of the possible returns is:

$$m(\mathbf{w}) = \min_{t \in T} y_t = \min_{t \in T} \sum i \in N r_{j,t} w_j$$

This function is not linear with respect to the variable $\mathbf{w}$. However, we can use an artifical variable m_y, constrained to be **smaller than all the returns of the scenarios** and whose we want to maximize the value.
This model call **minimax** is therefore:

Minimax Model

$$\begin{aligned} \max \quad & M_y \\ & M_y \leq y_t & \forall t \in T \qquad (3.8) \\ & y_t = \sum_{i \in N} r_{j,t} w_j & \forall t \in T \end{aligned}$$

+ General Portfolio Framework

We can of course add the constraints concerning the max number of stocks and the minimum weight authorized.
The `Minimax` Python class can be found in `linear/models/safety/minimax.py`, and an example of Minimax for $N = 100$ scenarios is presented in Figure 3.8.

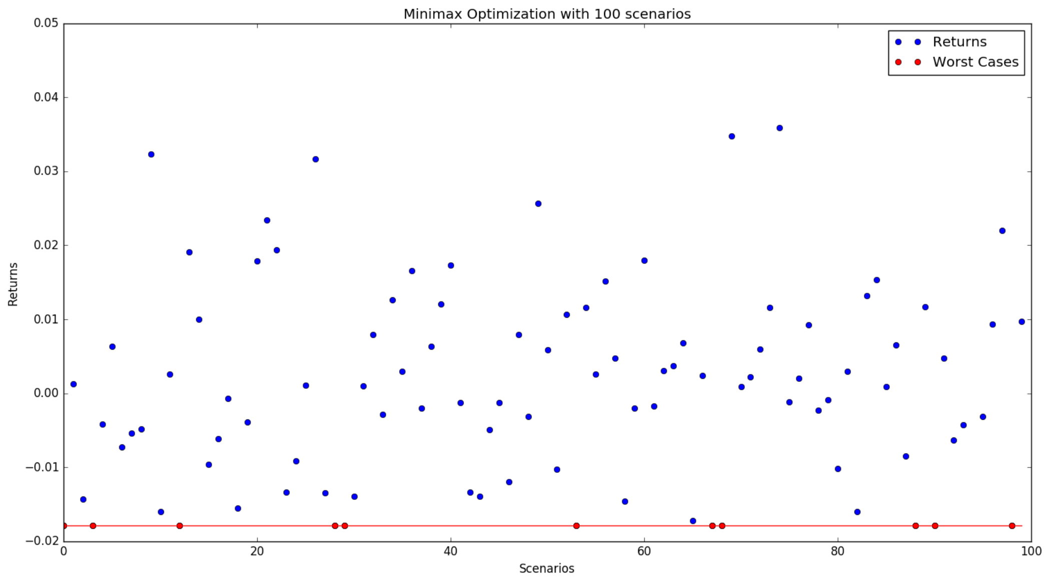

Figure 3.8: Mean Absolute Deviation Optimization

3.3.2 Value-at-Risk (VaR)

We have seen in the last section how to maximize **the worst return**. However, we could also be interested into minimizing not only the **worst cases**, but the **x %** worst cases. Indeed, instead of focusing on the minimum, defined as:

$$m(\mathbf{w}) = \min_{t \in T} y_t = \min_{t \in T} \sum i \in N r_{j,t} w_j$$

we could get interested into the **probability** that the return would be less or equal to some percentage. This problem is strongly linked to the notion of **quantile**.
Let us recall some definitions: if X is a real random variable, then the **cumulative distribution of X** (or **CDF**), is the real function F_X supported in $\mathbb{R}$ with values into $[0,1]$ such that:

$$\forall x \in \mathbb{R} \quad F_X(x) = \mathbb{P}\left(X \leq x\right)$$

Now, if X is a random variable whose CDF is F_X, then for any $\beta \in [0,1]$, the ***β*-quantile of *X*** is the real number $q_X(\beta)$ such that:

$$\lim_{\substack{x \to q_X(\beta) \\ x < q_X(\beta)}} F_X(x) \leq \beta \leq F_X(q_X(\beta))$$

In particular, if F_X is **continuous**, then F_X is a **bijection** from $\mathbb{R}$ to $[0,1]$, and:

$$q_X(\beta) = F_X^{-1}(\beta)$$

Now, if an investor wants to maximize the worst return within a given **confidence interval of** β **%**, then he would have to maximize the β-quantile. In Finance this is called the "Value at Risk" (or VaR). The CDF of the returns can have some discontinuities, which could lead to a non-linear objective function. However, the portfolio optimization problem can be formulated using Integer variables. For any scenario $t \in T$, let us define $z_t \in \{0,1\}$ as the boolean variable representing the fact that y is greater than the total return in t. We can write this constraint linearly:

$$y_t \geq m_y - M z_t \quad \forall t \in T$$

where M is greater than any possible return (we can take the max of the y_t for t in T plus one). Then, we can use the variables $\{z_t\}_{t \in T}$ to define the VaR condition:

$$\sum_{t \in T} p_t z_t \leq \beta - \epsilon \quad \forall t \in T$$

where ϵ as small as possible (useful in case of discontinuity of the CDF of the returns). We obtain the following **VaR Model**:

VaR MODEL

$$\begin{aligned}
&\max\ m_y \\
&\quad y_t \geq m_y - M z_t && \forall t \in T \\
&\quad \sum_{t \in T} p_t z_t \leq \beta - \epsilon && \forall t \in T \\
&\quad y_t = \sum_{j=1}^{N} r_{j,t} w_j && \forall t \in T \\
&\quad \mu = \sum_{i \in N} \mu_i w_i \\
&\quad \mu \geq \mu_0
\end{aligned}$$

The `VaR` Python class is implemented in **`linear/models/safety/var.py`**.

3.3.3 Conditional Value at Risk (CVaR)

One of the limits of the Value-at-Risk is that, **within the β % worst cases**, each case has **the same importance in the objective value**.
For instance, the worst case and the best case of the β % worst cases participate exactly the same to the objective value. It would make more sense to weight the worst case more than the best one.
If we **multiply each probability by the actual loss**, we come to the **definition of the mathematical expectation** of the loss, given that (or **conditionally to** the fact that) we are in the β % worst cases.:

$$\text{CVaR}_\beta(\theta) = \mathbb{E}\left[R_\theta \mid R_\theta < q_\beta\right]$$

This model is called **Tail mean** or **Conditional Value at Risk** (CVar). This is the name we adopt here. Within our framework, this becomes:

$$\text{CVaR}_{\frac{k}{T}}(\mathbf{w}) = \frac{1}{k/T} \sum_{s=1}^{k} p_{t_s} y_{t_s}$$

where $S_{t_1}, ..., S_{t_k}$ are the k worst scenarios (smaller y_{t_i}). The difficulty here is to represent the fact that a scenario is within the β % worst cases linearly. We can represent this using new variables:

$$u_t \in [0, 1] \quad \forall t \in T$$

which represent the **proportion of probability of scenario S_t within the β % worst cases**.
More specifically, we want that, at the optimum:

- $u_t = 0$ for all scenarios S_t which are not in the β % worst cases
- $u_t = p_t$ for all scenarios S_t which are in the β % worst cases
- $0 < u_t < p_t$ for at most **one** scenario (the best of the worst)

Therefore, the CVaR can be defined as:

$$\begin{aligned} \mathrm{CVaR}_\beta(\mathbf{w}) = \min_{\mathbf{u}} \quad & \frac{1}{\beta}\sum_{t\in T} y_t u_t \\ \text{s.t.} \quad & \sum_{t\in T} u_t = \beta && (3.9) \\ & 0 \le u_t \le p_t \quad \forall t \in T && (3.10) \end{aligned}$$

It is important to stress that it is **not** the CVaR global maximization problem, but only the **definition** of the CVaR, which means that we will have to take (after that) the max of the CVaR when the weights $\boldsymbol{w}$ vary. Furthermore, the Linear Expression y_t depend upon the weights $\boldsymbol{w}$. Therefore, **the objective value is not quadratic**.
However, it happens that, by taking the Dual problem of the definition problem of the CVaR, we can remove both problems. If we define ν as the dual variable of the constraint (3.9) and d_t the dual variables of the constraints (3.10), then:

$$\begin{aligned} \mathrm{CVaR}_\beta(\mathbf{w}) = \max_{\eta,\mathbf{d}} \quad & \eta - \frac{1}{\beta}\sum_{t\in T} p_t d_t \\ \text{s.t.} \quad & d_t \ge \eta - y_t && \forall t \in T \\ & d_t \ge 0 && \forall t \in T \end{aligned}$$

Therefore, we obtain the following model:

CVaR Model

$$\begin{aligned} \max \quad & \eta - \frac{1}{\beta}\sum_{t\in T} p_t d_t \\ & d_t \ge \eta - y_t && \forall t \in T \\ & d_t \ge 0 && \forall t \in T \end{aligned}$$

+ General Portfolio Framework

The **`CVaR`** Python class is implemented in **`linear/models/safety/cvar.py`**.
A representation of the CVaR optimization with 300 scenarios generated is shown in Figure 3.9.

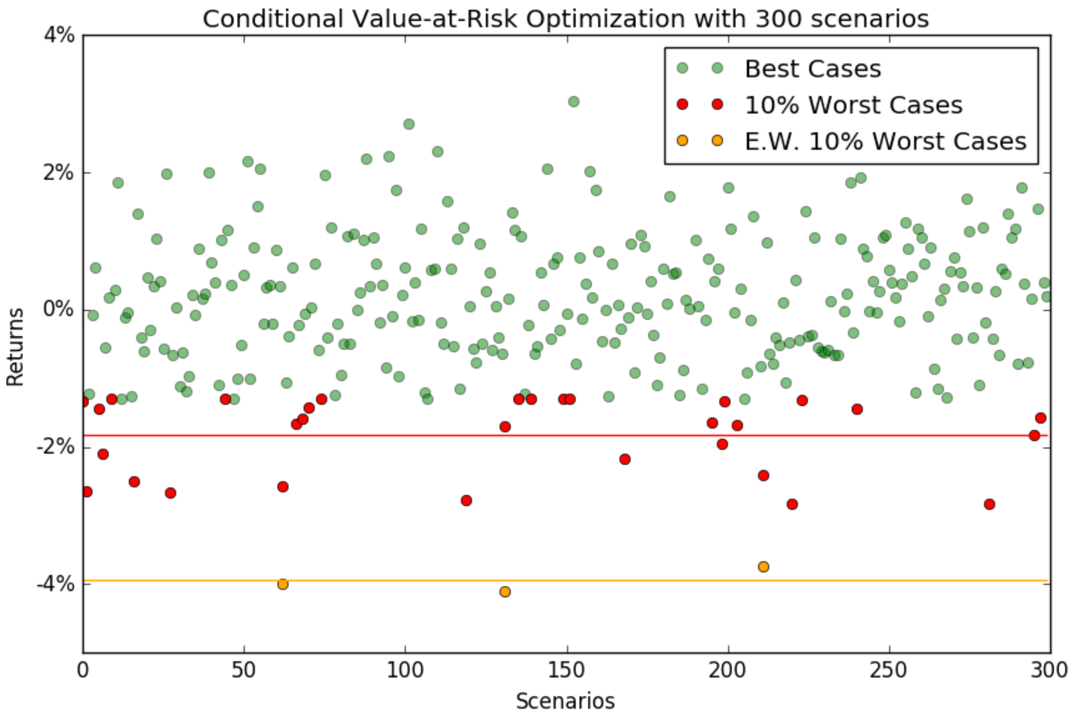

Figure 3.9: Conditional Value at Risk Optimization - $\beta = 10\%$ - Mean worst loss before and after optimization

3.4 Study of the variance

As the returns in the different scenarios **are randomly generated**, the output portfolio (the weights w_i), and thus the output objective value, of any of the measures we have seen **is also random**.
Indeed, if we launch the optimization three times on three different samples of 100 scenarios each, we will not have the same weights, as we can see on Figure 3.10. However, we can expect that, by increasing the number of scenarios generated, the different output portfolios gets closer and closer. It is indeed the case, as we can see on Figures 3.11 and 3.12.

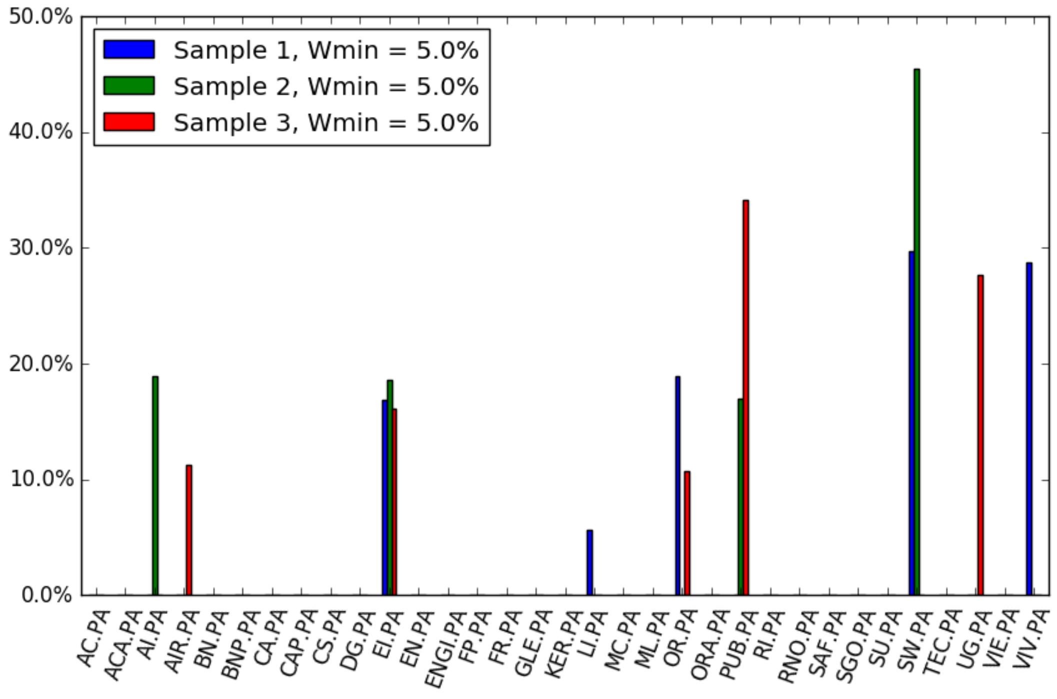

Figure 3.10: 3 optimizations - $W_{\min} = 5\%$ - 100 scenarios

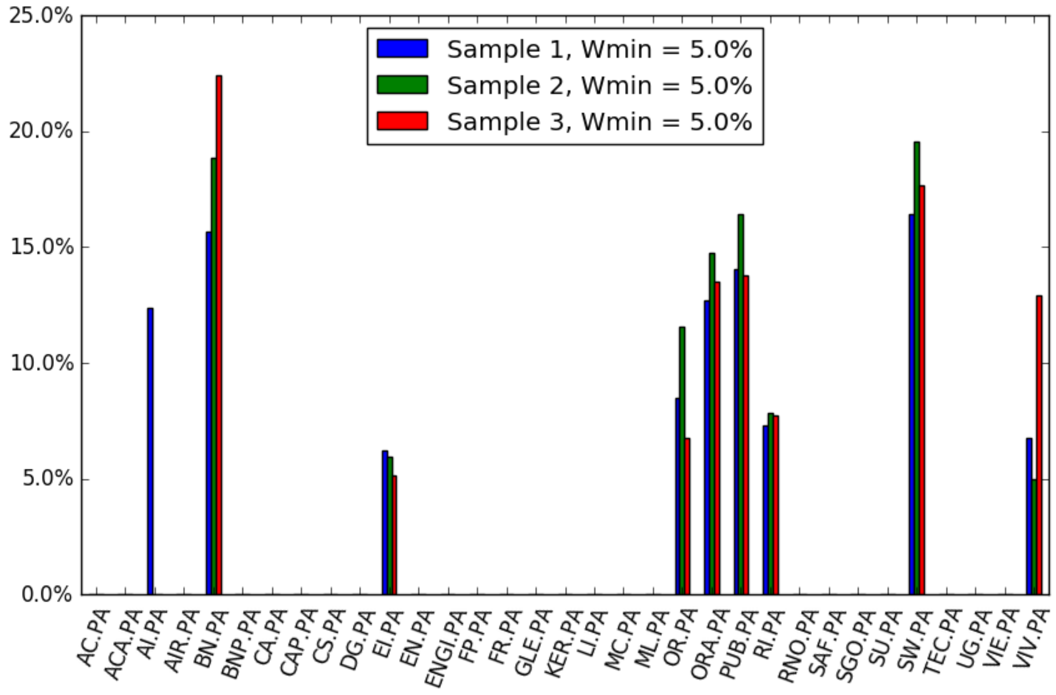

Figure 3.11: 3 optimizations - $W_{\min} = 5\%$ - 1,000 scenarios

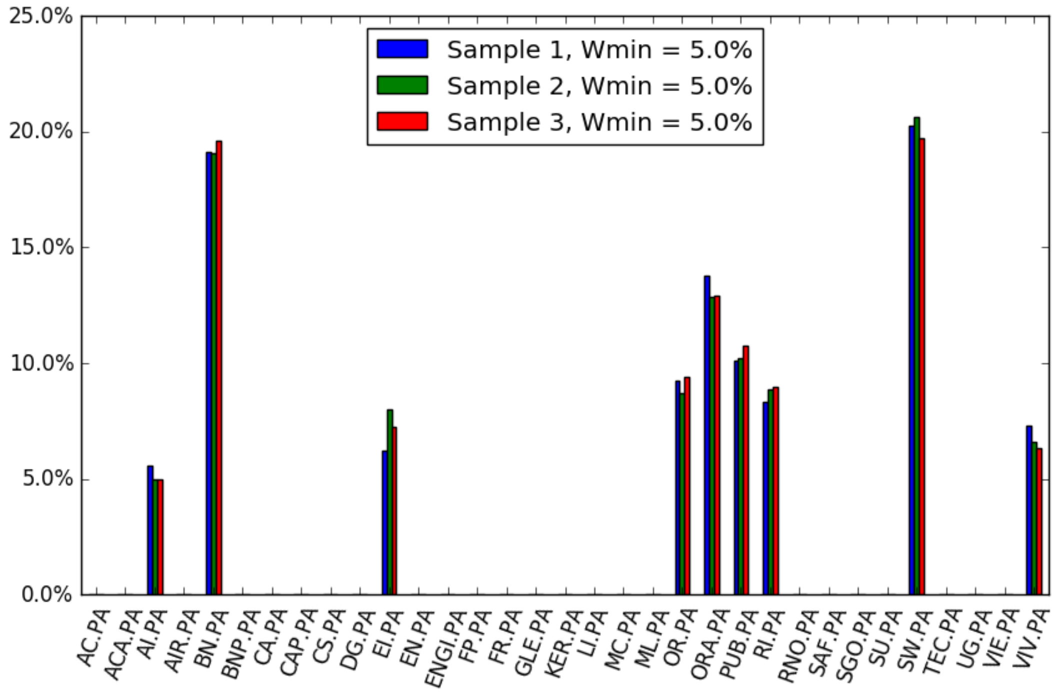

Figure 3.12: 3 optimizations - $W_{\min} = 5\%$ - 10,000 scenarios

Therefore, two very important question raises:

- **How can we measure the precision of the output?**
- **How many scenarios do we have to draw to obtain a given precision?**.
- **Does this number, and by how much, vary with the different Models?**

Concerning the first question, we could simply draw the **standard deviation of the objective value**. However, there are some issues doing that way:

- Firstly, the objective value is not **the same** for the different models, and, on top of that, it does not **represent** the same thing (it does not have the same physical dimension). It can be a volatility (MAD/GMD), a return (Minimax), an expectation (CVaR), a probability (VaR).
- Secondly, the objective value is not **useful** to actually compute the portfolio. The relevant output are the **weights**

Therefore, we chose to analyse the **standard deviation of the weights**. After having computed the standard deviations of each weights, we take the **maximum** of these standard deviations (which can be understood as the infinite norm $\|\cdot\|_\infty$). This is relevant, because we want a very **strong** control over the weights, and taking the mean would not be precise enough.
The results of the Standard Deviation computation is given in Figure 3.13.

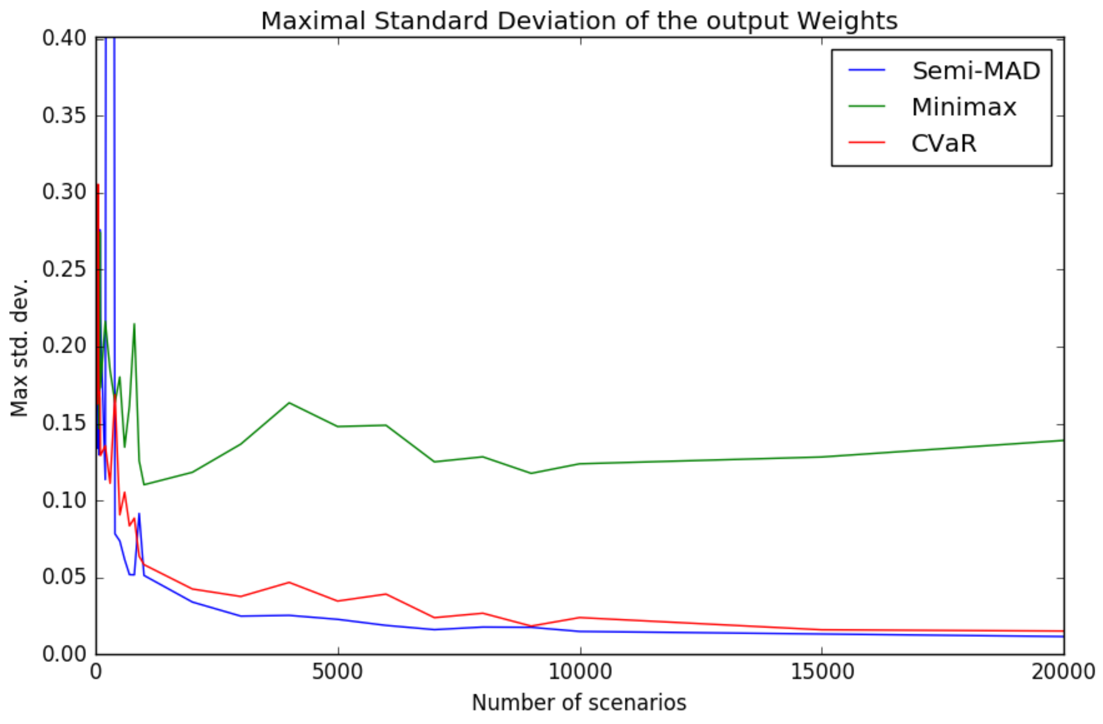

Figure 3.13: Maximal Standard Deviation of the output Weights

We can extract the following important results:

- For all three models, the asymptotic max standard deviation is reached for $T \approx 9,000$ scenarios.
- The *SemiMAD* and *CVaR* measures are **asymptotically more stable** than the *Minimax*. For the first ones, we can have an asymptotic max standard deviation of ≈ 0.03

Chapter 4

Output and Backtest

For any of the measures we have seen in the previous sections, the output is a **portfolio**.
To test the efficiency of these measures, what we could do is simply:

- Taking all the asseets' historical data from the beginning
- Computed N scenarios, with N big enough to have a low standard error on the output (determined in the last section "Variance")
- Analyse the performance of this portfolio, **if we would have hold it** since the beginning.

As we can see in Table 4.1, we obtain good results: This table has been created via the **`TableTime`** function in **`table.py`**.
We can see that:

- **Return**: Markowitz Portfolio managed to obtain the perfect mean return (by definition). The other ones manage to perform **better than the market** (first row)
- **Volatility**: all models manage to lower the volatility
- **Max Drow Down** (MDD): all models manage to limit the worst cases

However, **this is cheating**: indeed, it is impossible to have hold this portfolio from the beginning, because it was computed using the data until now, which is not available at the beginning.
Furthermore, as we can see on Figure 4.1, the different volatility can move a lot with the time.

Model	Mean Ret. (%)	Mean Cum. Ret. (%)	Vol. (%)	MDD(%)	Time (s)
Equally Weighted	12.55	10.27	23.55	- 9.01	
Semi-MAD	14.10	13.29	17.95	- 7.09	31.477
Minimax	13.78	12.56	19.72	- 7.44	16.757
CVaR	14.16	13.36	18.00	- 7.00	31.766
Markowitz	14.00	13.21	17.81	- 8.31	0.098

Table 4.1: Computation with $W_{\min} = 5\%$ - 20,000 scenarios - $\mu_{\min} = 14\%$

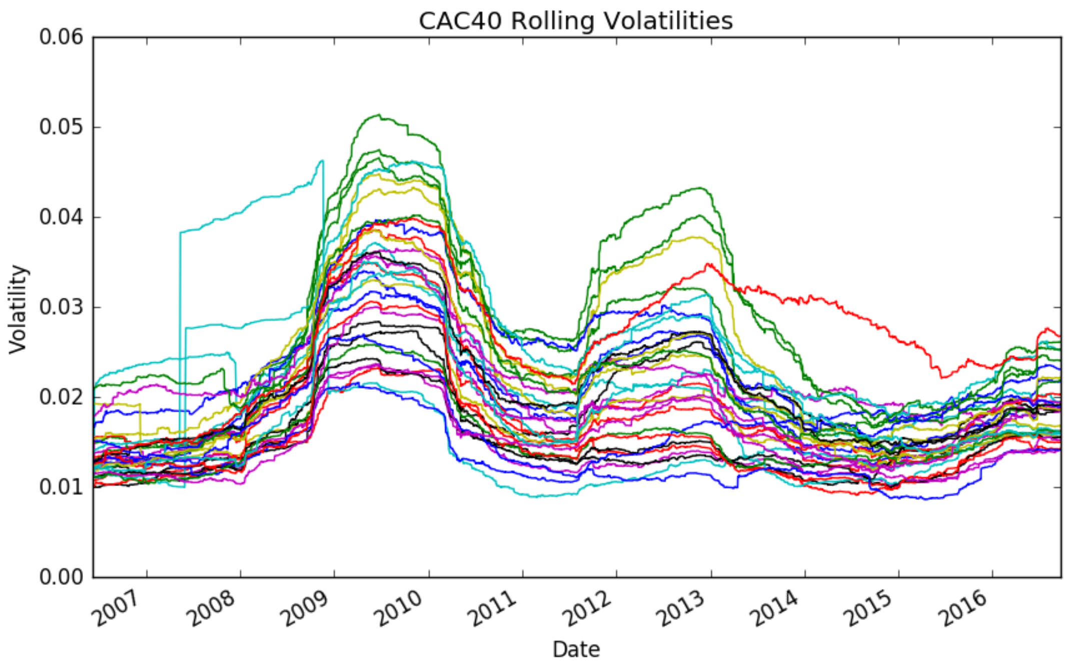

Figure 4.1: CAC 40 Rolling Volatilities (rolling window = 252 days)

Therefore, we need a way to take this "change" into account. What we can do to really represent the efficiency of our models, is **backtesting** the models:

- We choose a **rebalancing period** (weekly, fornightly, monthly, ...), and we suppose that we can rebalance our portfolio at each rebalancing date
- We choose a **rolling window** on which to compute the means and variance-covariance matrix, used to generate the scenarios

First of all, although we have already studied the standard deviation of the output, we can use a Backtest to show that the result of the backtest will not be **very** different, which stresses the robustness of our models. An example of 2 CVaR Models is presente in Figure 4.2.

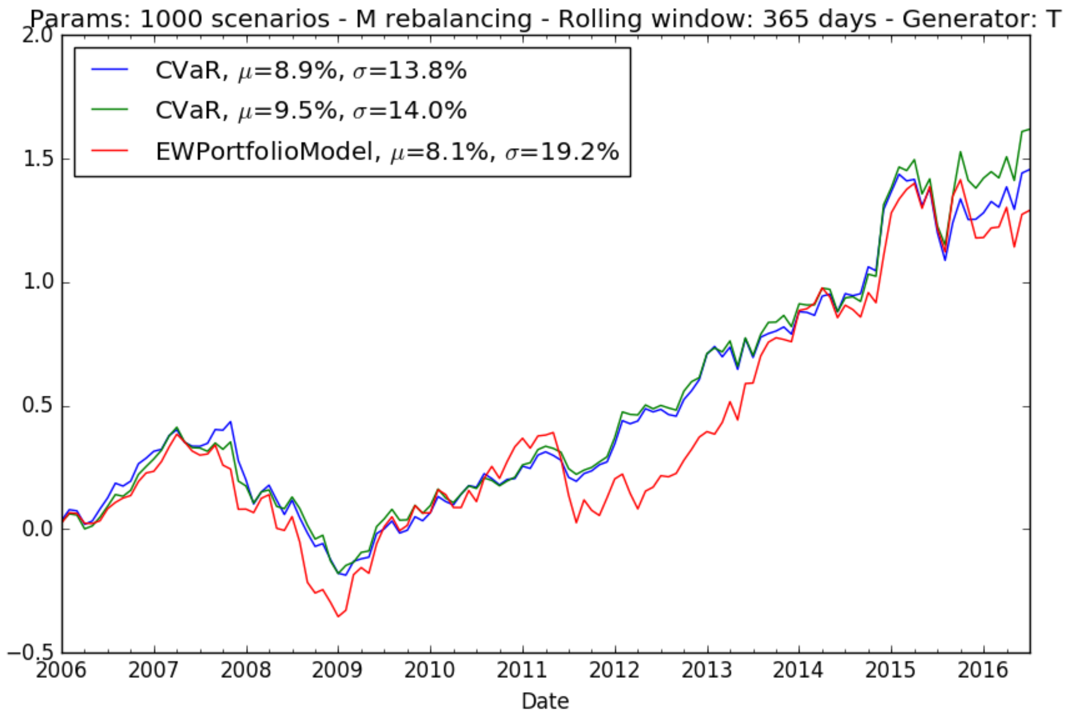

Figure 4.2: 2 BackTests of the same Model (CVaR)

We can see that the two CVaR are very close from each other. We also stress the fact that we only computed 1000 scenarios on each rebalancing date, which, as we saw, was not enough to reach the lowest standard deviation, but is far enough for such a BackTest.
Then, a final Backtest is shown in Figure 4.3.

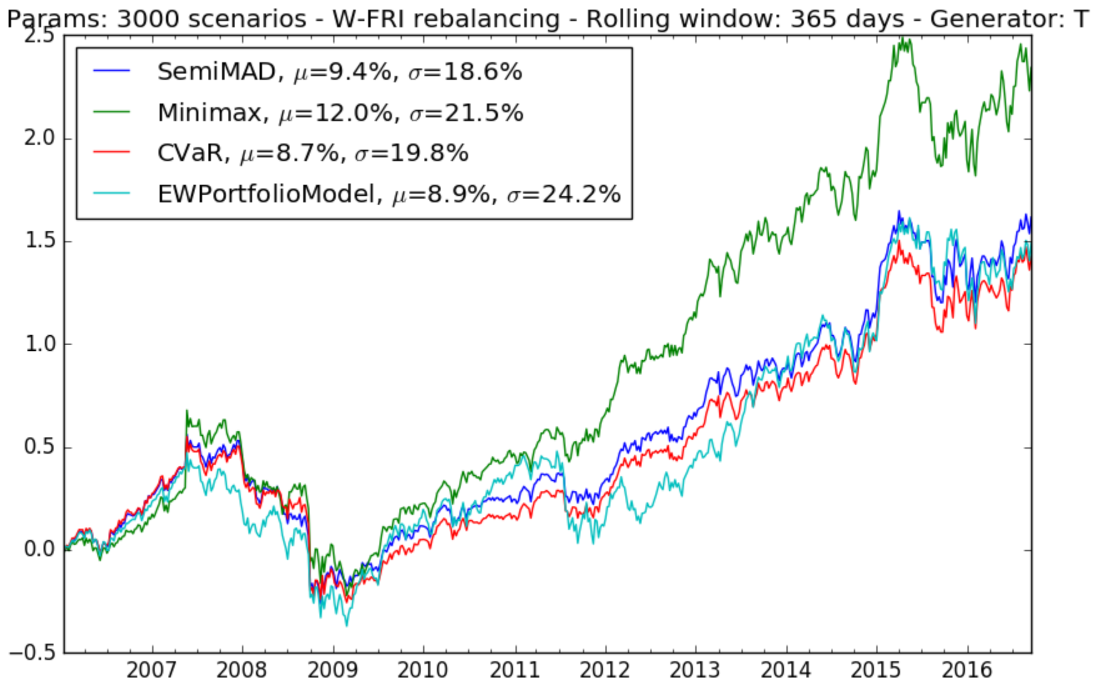

Figure 4.3: BackTest

We can see that:

- All three models have a **lower volatility that the market** (represented by the Equally Weighted portfolio).
- All models are **more robust to big losses**, as for example in July 2011, when the market incurred a big loss, contrary to our portfolios.

Chapter 5

Organization of the code

This short chapter is intended to provide a better understanding of the organization of the code.
First of all, the project requires some libraries which may not be installed by default:

- **`pandas`** (Time Series Analysis)
- **`tabulate`** (to draw results in Tables)
- **`pickle`** (Python object serialization)

All these three can be installed via **`pip`** or **`conda`**:

```
pip install pandas tabulate pickle
conda install pandas tabulate pickle
```

The user can simply launch the file **`main.py`** to have an illustration of the measures, and of a BackTest. However, an implicit requirement of this project was that it should be able to be used with an existing code of portfolio management, using Python classes (as "Security", "Bucket", etc.), and to be easily implemented in a web application already existing.
In order to reach this achievement, we have developped an "object-oriented" code. The goal of this chapter is to do a quick summary of the structural organization of the code, and to show the "sandbox" we have created so that the reader / user can easily test the code.
The code is organized in the following way:

`root/`				Root directory
	`entities/`			Base Python classes
		`portfolio.py`		Portfolio and PortfolioGroup classes
		`model.py`		Base Portfolio Model class
		`security.py`		
	`linear/`			All concerning Scenarios-based Portfolio
		`models/`		All Models classes
			`model.py`	Base class
			`risk/`	Risk measure classes
			`safety/`	Safety measure classes
		`plots.py`		Plot functions
		`scenarios.py`		Scenarios Generators
		`variance.py`		Variance analysis
	`markowitz/`			All concerning Markowitz Portfolio
	`backtest/`			BackTest functionnality
		`animation.py`		Animation functions to draw Backtest in live
		`backtest.py`		BackTest and BackTestGroup classes
	`data.py`			Data management: getting prices from Yahoo Finance API and **`pickling`** them
	`imports.py`			Run this file to import **everything**
	`table.py`			Draw computation results via tables
	`test.py`			Unit tests (via **`unittest`**) - 46 tests in $\approx$ 1m14s

An UML diagram of how are organized the different risk measures / safety measures extending **`gurobipy.Model`** is shown in Figure 5.1.

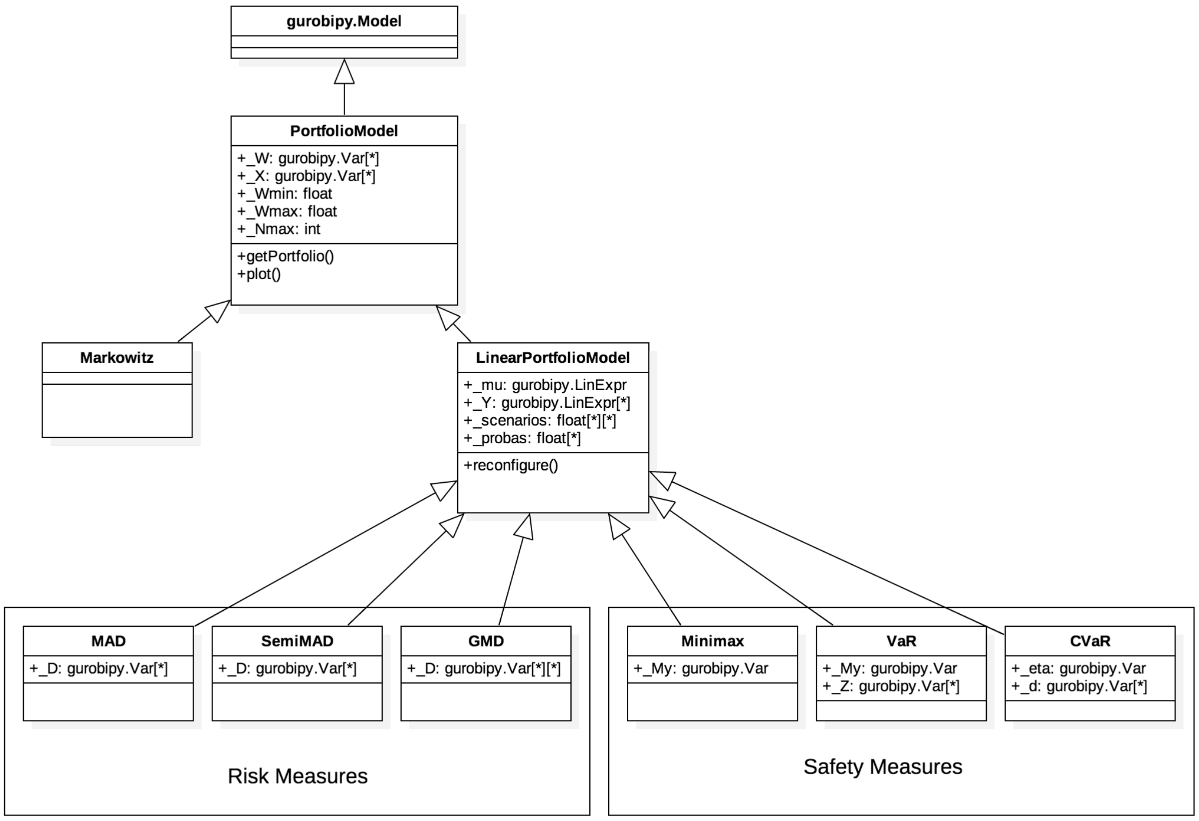

Figure 5.1: UML Diagram of the Portfolio Models hierarchy

To use the different elements of this project, the user can simply do:

```
from imports import *
```

or:

```
%run imports.py
```

When done, you can "play" we all the different measures, for instance creating a MAD Model:

```
s, p = generateGaussianScenarios(1000)
m = MAD(s ,p)
m.optimize()
 portfolio = m.getPortfolio()
 print portfolio.Return()      # Displays the annualized cumulative return
    of the portfolio
 portfolio.plot()
```

When a Model is instanciated, the following occurs:

- The relevant variables are created and added to the Model via **`addVar()`**
- **`update()`** is called
- The objective value is set

- The constraints and relevant Linear Expressions are created.

It is important to note that **`update()` is not called after that**, because it is computationally costly, and the user may prefer to use **`optimize()`** directly.
The user can also compare different measures via the **`PortfolioGroup`** object:

```
from imports import *

s, p = generateStudentTScenarios(100)
models = [MAD, SemiMAD, GMD, Minimax, CVaR]
group = PortfolioGroup([Model(s, p).optimize().getPortfolio() for Model in
    models])
group.plot()
```

The second important part easily available for the user is the **Backtesting**. The user can:

- Create a **`BackTestParamPool`**, which is a "pool" of parameters
- Create a Backtest
- Compute and plot it, or do both in the same time by animating it

```
from imports import *

pool = BackTestParamPool(freq='2W-FRI', window=365, generator=
    generateStudentTScenarios, N=1000)

g = BackTestGroup([SemiMAD, Minimax, CVaR, EWPortfolioModel], pool)

# Launches the animation
animateBackTestGroup(g)

# OR
# g.compute()
# g.plot()
```

Bibliography

[1] R. Mansini, W. Ogryczak, and M.G. Speranza. *Linear and Mixed Integer Programming for Portfolio Optimization*. Springer, 2015.