Optimization for Big Data Project - Portfolio Optimization

Chiara Cavalagli
chiara.cavalagli@studenti.unipd.it

Dacia Braca
dacia.braca@studenti.unipd.it

Pavel Ianko
pavel.ianko@studenti.unipd.it

Roberto Russo
roberto.russo.4@studenti.unipd.it

Abstract

1. Introduction to Constrained Problems

The *Portfolio Optimization* is a special case of the family of **Constrained Optimization Problems**. Before going into a deep understanding of the project, it is good to give a brief explanation of the mathematical context. From now on, the problem of interest is the following:

$$\min_{x \in C} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function and $C \subseteq \mathbb{R}^n$ is a convex compact set.

Unlike unconstrained optimization problems (where $C \equiv \mathbb{R}^n$), the minimization process needs to ensure that the points at each iteration x_k will still satisfy the constrain, i.e. $x_{k+1} \in C$. It is then necessary to define the set of *feasible directions* for any point $\overline{x} \in C$. If C is convex, the definition is then the following:

$$F(\overline{x}) = \{d \in \mathbb{R}^n \mid d = x - \overline{x},\ x \in C,\ x \neq \overline{x}\}. \tag{2}$$

Before giving the definition of the oracle, let us first enunciate the optimal condition for constrained convex optimization problems.

Definition 1.1. If $x^* \in C$ is a local minimum for the problem 1 with C convex set and $f \in C^1(\mathbb{R}^n)$, then

$$\nabla f(x^*)^T (x - x^*) \geq 0 \; \forall \, x \in C. \tag{3}$$

When dealing with convex functions, the condition (3) becomes necessary and sufficient. This optimal condition will give the idea for the search of the descent direction. Indeed, the work of finding the optimal candidate is done by the oracle, or more precisely, the *linear minimizer oracle* (LMO) that, as the name says, minimize a linearized version of the function instead of the actual one. The general equation is given by the following:

$$\text{LMO}(x) \doteq \underset{s \in C}{\operatorname{argmin}} \langle \nabla f(x), s \rangle. \tag{4}$$

Depending on the structure of C and the hypothesis on f, the oracle can be rewritten in different ways:

- If the set C is convex then we have that for every point $\overline{x}$ in C, the vector $d = \overline{x} - x$ still belongs to C and so we can substitute in the equation $s = d$;
- If C is convex and compact (i.e. closed and bounded in $\mathbb{R}^n$), then the oracle will always return a solution thanks to the Weierstrass Theorem;
- If C is a convex hull $C = conv(A)$, where A is the set of atoms, we obtain particular results which are the one of interest in this project.

Let us suppose for instance that $C = conv(v_1, \ldots, v_k)$ i.e. C is the convex hull of k extreme points, or vertices. Then the LMO becomes what follows:

$$LMO(x) = \underset{s \, \in \, C}{\operatorname{argmin}} \langle \nabla f(x), s \rangle = v_{i^*}, \tag{5}$$

that is one the vertices of the set A, and its index i^* is such that $i^* = \operatorname{argmin}_i \nabla_i f(x)$, with $i \in \{1, 2, \ldots, k\}$. This means that the optimal point will be one of the vertices. This is very powerful because we restrict the research of a minimum into a finite set of points.

Our work is based on a special convex hull, the *unit simplex*, whose equation is the following:

$$\begin{aligned} C &= \{x \in \mathbb{R}^n \mid e_1 x_1 + \ldots e_n x_n = 1,\ x_i \geq 0 \; \forall \, i\} \\ &= conv(\{e_1, \ldots, e_n\}) \,. \end{aligned} \tag{6}$$

Another great hack that is used in the research of the minimum in those family of problems, is defining the *gap function* of f:

$$gap\ (x) \doteq \max_{s \in C} \langle \nabla f(x), x - s \rangle; \tag{7}$$

this equation maximize, inside the set C, the gap between the function and the first order linearization of the function, both of them evaluated in the current point x. The gap function will be the main stopping condition of each algorithm implemented. That is because those values can be obtained at each iteration k without any computation, once we have $\hat{x}_k = \text{LMO}(x_k)$, we just have to check that:

$$\langle \nabla f(x_k), \hat{x}_k - x_k \rangle \geq -\varepsilon, \tag{8}$$

where ε is the tolerance level.
Also the mathematical structure and properties of the objective function f play a significant role in the resolution of the optimization problem (1). In order to obtain a theoretical guarantee of convergence for the solving algorithm, it is usually required convexity or convexity together with Lipschitz continuity of its gradient (LCG).

Definition 1.2. A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ has Lipschitz continuous gradient if there exists a constant $L > 0$ such that

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y||, \quad \forall x, y \in \mathbb{R}^n\,. \tag{9}$$

In several machine learning problems, the objective function (like in the case of our interest) falls into the category of *self-concordant* functions (SC), which unfortunately often are neither convex nor respect the LCG condition.

Definition 1.3. Given $f \in C^3(\mathbb{R}^n)$ a convex function with open domain ($\overset{\circ}{dom}\ f$), this is said to be Self-Concordant with parameter $M > 0$ if and only if:

$$|\phi^{'''}(x, 0)| \leq M\phi^{''}(x, 0)^{\frac{3}{2}}, \tag{10}$$

where ϕ is a function defined for each fixed $x \in dom\ f$ and for each direction $u \in \mathbb{R}^n$, that is:

$$\phi(x; t) = f(x + tu)\ \forall\ t \in\ dom\ \phi(x; \cdot),$$

Comment 1.1. *We can observe that the third order of continuity is necessary in order to define the condition (10).*

The prime example of a SC function is the logarithm, or more precisely like the following: $f(x) = -log(-g(x))$, where $g(\cdot)$ is a convex function satisfying some particular conditions. We will see later that the objective function of interest satisfies this structure.

1.1. The role of the curvature

Most of the convergence analysis of the FW algorithms, rely on the *curvature of the function*, whose definition is valid for convex functions and its formulation is the following:

$$C_f = \sup_{\substack{x,s \in domf \\ \gamma \in [0,1]}} \frac{2}{\gamma^2} D_f(x + \gamma(s - x), x); \tag{11}$$

where D_f is the *Bregman Divergence*:

$$D_f(x, y) = f(y) - f(x) - \langle \nabla f(x), y - x \rangle \tag{12}$$

defined for every point $x, y \in\ domf$. This (12) is the difference between the function evaluated in x and the linearization of the function around y, evaluated in x. Plugging this into the curvature definition, we have a measure about the *non linearity* of our function of interest.

In the classical framework of FW analysis, we have that $C_f < \infty$ i.e. the curvature is bounded, and this is crucial for the proof of the convergence of the algorithm and its variants. Moreover, the curvature is strictly linked to the property of Lip. continuous gradient: it has been shown by Nesterov [6] what follows.

Lemma 1.1. *Let f be a convex function with Lip. continuous gradient with constant $L > 0$. Then:*

$$C_f \leq diam_{||.||}(domf)^2 L. \tag{13}$$

Unfortunately, as anticipated before, SC functions usually do not have finite curvature (i.e. $C_d \nless \infty$) or neither Lip. continuous gradient. It is then obvious that another approach is needed in order to prove the convergence of FW algorithms.

What it will be the key tool for proving the convergence of the algorithms is the *Fenchel Conjugate* of f, that is:

$$f^*(u) \doteq \sup_{z \in domf} \{\langle z, u \rangle - f(z)\}, \tag{14}$$

and the *Fenchel-Young* inequality, that is:

$$f(x) + f^*(y) \geq \langle y, x \rangle, \tag{15}$$

for all $x \in domf$ and $y \in domf^*$. See Section 3.3 for more details.

2. Portfolio Optimization Problem

A *portfolio* is a collection of financial investments called *assets*, defined as resources with an economic value that an individual, corporation or country owns and controls with the expectation to reach a future benefit. There exists many examples of asset, each one characterized from an economic perspective by upsides and downsides which inevitably influence the success of investments [2]. It is clear that a large number of combinations are outlined, so much so that portfolio managing becomes a non-trivial task.

We then define the *portfolio optimization* as the process of selecting the best asset distribution within a set of possibilities, according to some objective. In this frame the wisdom of diversification represents a key concept, trying to reduce economical risks by allocating resources among various financial instruments, industries and other categories [1]. Moreover, it aims to maximize returns by investing in several areas that would each react differently to the same event [1]. Such objectives can be formalized in a mathematical optimization problem in which the goal traduces into the minimization of an opportune utility function $f(x)$ by choosing the weights of the assets in the portfolio [4].

Let us then consider a matrix of data $R \in \mathbb{R}^{T\times n}$, where T gives the number of periods of time and n the number of possible assets. Each row $r_t \in \mathbb{R}^n$ contains the economic returns associated to the entire n assets at period t in the investment horizon. The best portfolio x is the solution to the constrained problem:

$$\min_{x\in dom f} \left\{ f(x) = -\sum_{t=1}^{T} \ln\left(r_t x\right) : x_i \geq 0, \sum_{i=1}^{n} x_i = 1 \right\}. \tag{16}$$

It is easy to conclude that $f(x)$ satisfy the form of a logarithmic self-concordant function by recalling that a linear combination of self-concordant functions is still self-concordant. Each t-th argument is obtained as a product between the returns vector and the variable of interest x, which has dimension n. If the argument is less than one it will give negative contribution to the utility function, neutral if unitary and positive otherwise.

The domain set of function $f(x)$ is compact and convex, defined as follows:

$$dom\ f = \{x \in \mathbb{R}^n : r_t \cdot x > 0\}. \tag{17}$$

Each x_i coordinate quantifies the percentage of resources to invest for an asset i during a considered period t.

The need to give a physical and economic meaning to variable x justifies the definition of the constraint C: we ask that each element of this unknown vector be non-negative in order to not allow short selling and the unitary sum guarantees that the total amount of money invested does not exceed the available funds. We therefore note that the set C is the unit simplex 6, moreover the so defined portfolio problem could admit a solution since $dom f \cap C \neq \varnothing$.

3. Frank-Wolfe algorithm and variants

The *Frank-Wolfe method*, also known as conditional gradient method or reduced gradient method, is an iterative first-order optimization algorithm proposed by Marguerite Frank and Philip Wolfe in 1956 to solve linearly constrained programming problems [7, 6]. In order to solve large-scale problems in machine learning field, the first order methods are usually preferred and the Frank-Wolfe algorithm may represent a good choice, having nice and advantageous properties compared to projected gradient methods [4]. In each k-th iteration, the algorithm considers a linear approximation of the objective function and moves towards a minimizer of that, taken over the same domain, i.e. a feasible search atom $\hat{x}_k$ [7, 4]. Another reason for the recent increased popularity of Frank-Wolfe-type algorithms is the sparsity of their iterates: each iterate can be represented as a sparse convex combination of at most $k+1$ atoms $S_t \subseteq A$ (S_t *active set*) of the set C [7]. The next iterate x_{k+1} is then obtained by doing a line-search on f along the direction passing through x_k and $\hat{x}_k$.

The method admits simplifications in the search for the descending direction when the constraint has a special structure and in particular for the unit simplex case the LMO returns a kind of solution described in the previous section (5). This allows to streamline the algorithmic procedure (see Algorithm 1).

Algorithm 1 Frank-Wolfe method for unit simplex

```
1: Choose a point $x_0 = e_i \in dom f \cap C$, with $i = 1, \dots, n$
2: Choose a tolerance $\varepsilon > 0$
3: for $k = 0, 1, \dots, K$ do
4:     $\hat{x}_k = e_{i_k}$ with $i_k = \mathrm{argmin}_i \nabla_i f(x_k)$
5:     if $\langle \nabla f(x_k), x_k - \hat{x}_k \rangle \leq \varepsilon$ then STOP
6:     else $x_{k+1} = x_k + \alpha_k(\hat{x}_k - x_k)$, with $\alpha_k = \frac{2}{k+2}$
```

Unfortunately, in the more general cases, the traditional version of Frank-Wolfe method can have computational limitations in several optimization problems, such as a slow convergence rate. It is possible to prove that the method has a sublinear convergence rate $\mathcal{O}(1/k)$, when the function to be minimized is convex and has a bounded curvature. A further improvement is obtained in the special case in which the convexity becomes strong, leading to a linear rate $\mathcal{O}(c^k)$ [4, 7]. Same results could be reached in well-conditioned problems over polytopes under a-priori assumption that the optimum solution stays in the relative interior of the feasible set [4, 5]. In this particular case, the linearity of the convergence is not free from constraints as the constant depends on the distance between the solution and the boundary. On the contrary, when no preliminary information is available or when the solution lies on the boundary the convergence rate of the iterates becomes dramatically slow [7]. In general it is not always possible to correct the algorithm to have improvements but in the case of polyhedral constraint there are a couple of variants that guarantee a satisfactory convergence rate, avoiding the so called "zig-zag" effect gained thought the iterations [4, 7]. Indeed, when the algorithm gets closer to the optimum solution and the point belongs to the boundary, the moving directions become more and more orthogonal to the gradient.

A first approach which tries to avoid this effect is called *pairwise Frank-Wolfe method*, obtained by modifying a previous version called away-step. The main idea is to include directions d_k^{AS} pointing away from extreme points of constraint (away-step directions) and, instead of selecting one between this and the one calculated in the usual way d_k^{FW}, the two contributions are unified in a single direction (see Algorithm 2) [7].

Algorithm 2 Pairwise Frank-Wolfe method for unit simplex

```
1:  Choose a point $x_0 = e_i \in dom f \cap C$, with $i = 1, \dots, n$
2:  Choose a tolerance $\varepsilon > 0$
3:  Set $S_0 = \{x_0\}$
4:  for $k = 0, 1, \dots, K$ do
5:      $\hat{x}_k^{FW} = e_{i_k}$ with $i_k^{FW} = \text{argmin}_i \nabla_i f(x_k)$ in $A$
6:      if $\langle \nabla f(x_k), x_k - \hat{x}_k^{FW} \rangle \leq \varepsilon$ then STOP
7:      $\hat{x}_k^{AS} = e_{j_k}$ with $j_k^{AS} = \arg\max_j \nabla_j f(x_k)$ in $S_k$
8:      $d_k = d_k^{FW} + d_k^{AS} = \hat{x}_k^{FW} - \hat{x}_k^{AS}$
9:      Set $\gamma_{max} = x_k[j_k^{AS}]$
10:     Line search: $\gamma_k \in \text{argmin}_{\gamma \in [0, \gamma_{max}]} f(x_k + \gamma d_k)$
11:     $x_{k+1} = x_k + \gamma_k d_k$
12:     $S_{k+1} = \{e_i \mid x_k[i] > 0\}$
```

3.1. Convergence Analysis

As explained in the introduction, the main results of convergence of FW algorithms are based on the curvature of the function. In our case we cannot assume that $C_f < \infty$ as well as f with Lipschitz continuous gradient and so there is no theorem showing such result. However, it is clear from the numerical experiments that the two algorithms behave surprisingly good. The pairwise approach is the best one for gap reduction because it stops the algorithm after 100 iterations more or less; that means that the early stopping condition is satisfied (20). The classical FW, that is the poorer regarding the constrains on the problem, still manage to reach a proper minimum of the objective function after just 10 iterations.

3.2. Adaptive FW method

The structure of the constraint certainly plays a decisive role for the convergence of the method but also the nature of the function to be optimized does. The convergence of FW method, together with the modified version, relies on bounded curvature of the objective function and do not present any particular study on the Self-concordant ones [4]. A sufficient condition is that $f(x)$ presents a Lipschitz continuous gradient over the domain of interest but, unfortunately, as we already know, SC functions typically have neither unbounded curvature nor the strongly convexity (feature that would have led to a possible linear rate). Therefore, we need to constructs adaptive variants of the FW method by working on new step-size policies, ensuring sublinear global convergence. The main trick is to come up with step-sizes which ensure monotonicity of the method, leading to Lipschitz smoothness and strong convexity of the objective on a level set [4]. Asking this, the algorithms will not require knowledge of the Lipschitz constant but rather rely only on basic properties of SC functions. The operating principle follows the one described up to now with a substantial difference in the line search procedure and in the local Lipschitz estimate L_k [3, 4]. The so-called *backtracking subroutine* looks for the best stepsize while estimating a local Lipschitz constant and it does that by using a quadratic upper approximation of the objective function, that is:

$$Q(x_k, \alpha, \mu) = f(x_k) - \alpha \, \text{gap}\,(x_k) + \frac{\alpha^2 \mu}{2} ||d_k||_2^2 . \quad (18)$$

This result is a consequence of the following descent lemma:

Lemma 3.1. *Let f be a SC function and $(x_k)_k$ a series generated by the algorithm FW Variant 2. Then we have that, by assuming that $x_{k+1}(t) = x_k + t d_k \in S_k$ for every $t \in [0, \gamma_k]$ and for every iteration k, we have:*

$$||\nabla f(x_k + t d_k) - \nabla f(x_k)|| \leq L_{\nabla f} t ||d_k||_2. \quad (19)$$

Comment 3.1. *This lemma tells us that we have LCG for the iterates of the algorithm and so that we can obtain a localized version of the result for all t $\in [0, \gamma_k]$:*

$$f(x_k + t d_k) \leq f(x_k) + \langle \nabla f(x_k), t d_k \rangle + \frac{L_{\nabla f} t^2}{2} ||d_k||_2^2; \quad (20)$$

We can introduce then the function Q by taking the second member of the inequality.

It is easy to observe that each term of the new function is given *for free* during the computation. At each subiteration, the subtrack process looks for the right stepsize until it finds the first value that satisfy the stopping condition (20)[4, 3].

The hyperparameters $\gamma_g < 1$ and $\gamma_u > 1$ (the recommended choices for them are 0.9 and 2) are needed respectively to define the the extremes of the interval in which the search of the L_k constant will be made. The variable that approximates L_k is defined as:

$$\mu = \text{Clip}_{[\gamma_d L_{k-1}\,,\, L_{k-1}]} \left(\frac{\text{gap}\,(x_k)^2}{2\big(f(x_k) - f(x_{k-1})\big)||d_k||_2^2} \right). \quad (21)$$

Since this is a recursive algorithm (see input variables in Algorithm 3, the quality of the approximation depends on the first estimate L_{-1}, crucial for the performance of the entire algorithm. Starting from the definition of LCG condition, a heuristic procedure is provided ($\varepsilon = 10^{-3}$) [3]:

$$L_{-1} = \frac{||\nabla f(x_0) - \nabla f(x_0 + \varepsilon d_0)||}{\varepsilon ||d_0||}. \quad (22)$$

A final comment concerns the relationship between the local constant L_k and the relative estimate for the stepsize α_k. The latter depends on the first in a sort of inversely proportional way: the bigger is the constant, the smaller is the value of the stepsize so that in the various iterations the steps taken on the descent direction are minimal leading to bad performance of the method [4].

3.2.1 Addressing the problem of bad L_{-1} initialization

As it is shown in the results section, the heuristic to initialize L_{-1} leads to a much slower convergence compared to the other algorithms. This happens because the heuristic mimics the LCG that gives a lower bound for the Lipschitz constant. But this conditions does not apply to our objective function, so the heuristic leads to a poor initialization of the Lipschitz. That causes a imprecise quadratic approximation of the objective function, yielding very small step-sizes in the first iterations. So here we introduce an alternative procedure to initialize L_{-1} that leads to a much better performance. The logic behind this new methodology is that since the line search attempts to find the step-size $\gamma*$ that minimizes the quadratic approximation, given a fixed local approximation of L, instead of guessing this constant in the first iteration we fix the initial step-size α_{-1} and assume that it minimizes the quadratic approximation, given L_{-1}. Once done this, we calculate L_{-1} inverting the formula for the step size of the variant 2, fixing $\alpha_{-1} = 1$. Doing so, we get the correct Lipschitz constant if the guess for the step-size is right. Since at the first iteration it is very probable that we need to move far away from the starting point, initializing α_{-1} to 1 is a good guess. In our implementation we fix $\alpha_{-1} = 1$, but other choices are possible, for example by mean of a line search or just trying different values. In our case, the first iteration will be equal to the first iteration of the classic Frank-Wolfe algorithm. This heuristic yields the following formula:

$$L_{-1} = \frac{gap(x_0)}{||d_0||_2^2} \tag{23}$$

For the sake of simplicity we will refer to the variant 2 with this new initialization strategy as variant 3, keeping in mind that the only difference with the original variant 2 is the way we perform the first iteration.

Algorithm 3 Adaptive Frank-Wolfe method for SC functions

```
1:  Choose a point $x_0 = e_i \in domf \cap C$, with $i = 1, \dots, n$
2:  Choose a tolerance value $\varepsilon > 0$
3:  for $k = 1, 2, \dots, K$ do
4:      if gap $(x_k) > \varepsilon$ then
5:          $\hat{x}_k = e_{i_k}$ with $i_k = \text{argmin}_i \nabla_i f(x_k)$
6:          $d_k = \hat{x}_k - x_k$
7:          $\mu \in [\gamma_d L_{k-1}, L_{k-1}]$
8:          $\alpha = \min\left\{1, \frac{\text{gap }(x_k)}{L_{k-1}\,||d_k||_2^2}\right\}$
9:          if $f(x_k + \alpha d_k) > Q(x_k, \alpha, \mu)$ then
10:             $\mu = \gamma_u \mu$
11:             $\alpha = \min\left\{1, \frac{\text{gap }(x_k)}{\mu\,||d_k||_2^2}\right\}$
12:         $x_{k+1} = x_k + \alpha_k d_k$, where $\alpha_k = \alpha$
```

3.3. Convergence Analysis

The knowledge needed to prove the convergence of the algorithm involves the *dual objective function*, that is:

$$\psi(z) \doteq -f^*(z) - H_{domf}(-z);$$

where f^* is the Fenchel conjugate of f (14) and $H_{domf}(c) = sup_{x \in domf} \langle x, c \rangle$. Let us then enunciate the results.

Theorem 3.1. *Let f be a SC function and $(x_k)_k$ a series of iterates generated by the adaptive FW algorithm. Then, denoted with*

$$h_k = f(x_k) - f(x^*),$$

the approximation error at iteration k, we have that:

$$h_k \leq \frac{2gap(x_0)}{(k+1)(k+2)} + \frac{k\dot{d}iam(domf)^2}{(k+1)(k+2)}\overline{L}_k; \tag{24}$$

where $\overline{L}_k = \frac{1}{k}\sum_{i=0}^{k-1} L_i$, i.e. the arithmetic mean of all the estimates of the Lip. constant given as output by the algorithm 3.

What follows from that statement is that the convergence is $O(\frac{1}{k})$; that is impressive, given the fact the our function f is only convex and SC.
The proof of this theorem relies on a series of results. It is first proved that

$$gap(x_k) = f(x_k) - \psi(\nabla f(x_k));$$

then, by using the Fenchel-Young inequality (15), an upper bound for the approximation error h_{k+1} is proved, that will lead then to the thesis.

4. Results and Discussion

4.1. Experiment with synthesised data

To synthesize datasets for testing algorithms, we follow the procedure, described by Dvurechensky et al. and Sun et al. [3, 8].

The datasets represent three matrices of size (*T*, *n*), where *T* stands for number of periods, and *n* – number of assets.

We used three randomly generated matrices of sizes (1000, 800), (1000, 1200) and (1000, 1500). Each entry of the matrices is a random value, described as:

$$1 + N(0, 0.1), \tag{25}$$

where normally distributed component simulates varying closing price of *n* assets [3]. Thus, Figure 1 illustrates the oscillating returns of assets in the dataset.

For each dataset, we run three algorithms:

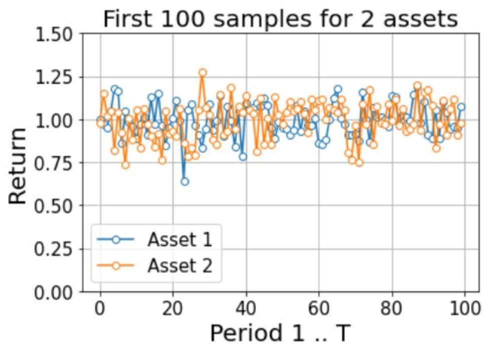

Figure 1. Returns of first two assets, modeled as random variable, during first 100 periods

- Frank Wolfe,
- Pairwise Frank Wolfe,
- Improved variant Frank Wolfe, [6]

and compare them, based on following characteristics:

- Objective function curve,
- Gap curve,
- Number of iterations,
- Computational time,
- Distribution of asset allocation.

4.2. Results with synthesised data

For all studied algorithms, we fixed number of iterations to 10000. As it is seen from figure 2, a significant difference in convergence is observed for pairwise Frank Wolfe method – similar value of objective function is achieved after around 100 iterations, while Pairwise FW reaches the lowest gap possible, with 100 times less iterations. However, a remarkable, 100 fold difference is attained for total number of iterations (Fig. 3), computational time (Fig. 4).

As summarized in tables 1 and 2, Pairwise Frank Wolfe method was found to be the most efficient in terms of computational time and number of iterations. Compared to standard Frank Wolfe and improved variant, the pairwise variation takes remarkably less iterations, reaching the stopping condition before iteration limit.

The early quit for pairwise Frank Wolfe is clearly seen on the gap curve (Fig. 2).

Data size	FW	Var 2	Var 3	Pairwise FW
800	10000	10000	2984	**111**
1200	10000	10000	374	**74**
1500	10000	10000	10000	**68**

Table 1. Comparison of number of iterations, necessary to solve portfolio optimization for three datasets

Dataset	FW	Var 2	Var 3	Pairwise FW
800	34.36	45.05	15.69	**1.23**
1200	55.23	68.11	2.51)	**1.26**
1500	76.36	86.54	92.26	**1.45**

Table 2. Comparison of CPU time (in seconds), necessary to solve portfolio optimization for three datasets

A heatmap on Figure 5 visualizes the distribution of investments, returned by each algorithm. One can see, all algorithms converged to the same distribution. According to the heatmap, the optimization algorithms mainly recommend splitting budget between several (5-6) key assets. The bad behavior of the variant 2 in terms of objective function and gap decreasing is described before. We can see the improvement of the performance watching at the fast decreasing of the gap of the variant 3. In all the cases it gets the fastest decrease in the first 40 iterations, but then it slows down because of the “zig-zag” phenomenon, that is addressed only by the Pairwise Frank-Wolfe, in fact is the only one that do not suffer from it. Still the fact that the variant 3 converges much faster than the variant 2 and the Classic Frank - Wolfe means that our new heuristic works good enough. The fast initial decreasing, suggests that this methodology could be used as line - search for Pairwise Frank - Wolfe, potentially yielding a faster convergence.

4.3. Results with real data

For this experiment we downloaded a list of all the tickers corresponding to all the common stocks traded in the NASDAQ market. Using the *YahooFinance API Python* library we downloaded the historical series of the adjusted closing price of all these assets, from the 2020-01-01 to the 2022-03-09. Cleaning the dataset we found that most of the historical price series have 21 missing values, all of them in the same days, so we removed all the historical series that do not. After removing the missing values we yielded a dataset composed of 550 days and 2214 assets. We evenly split this dataset in two group of assets and we perform the experiment with the three algorithms for the two groups. Then we compute the return as

$$r_{ij} = \frac{p_{ij}}{p_{i-1j}} \tag{26}$$

where i is the period, j the asset and p_{ij} the price of the asset j a time i. The performances are shown in the plots.

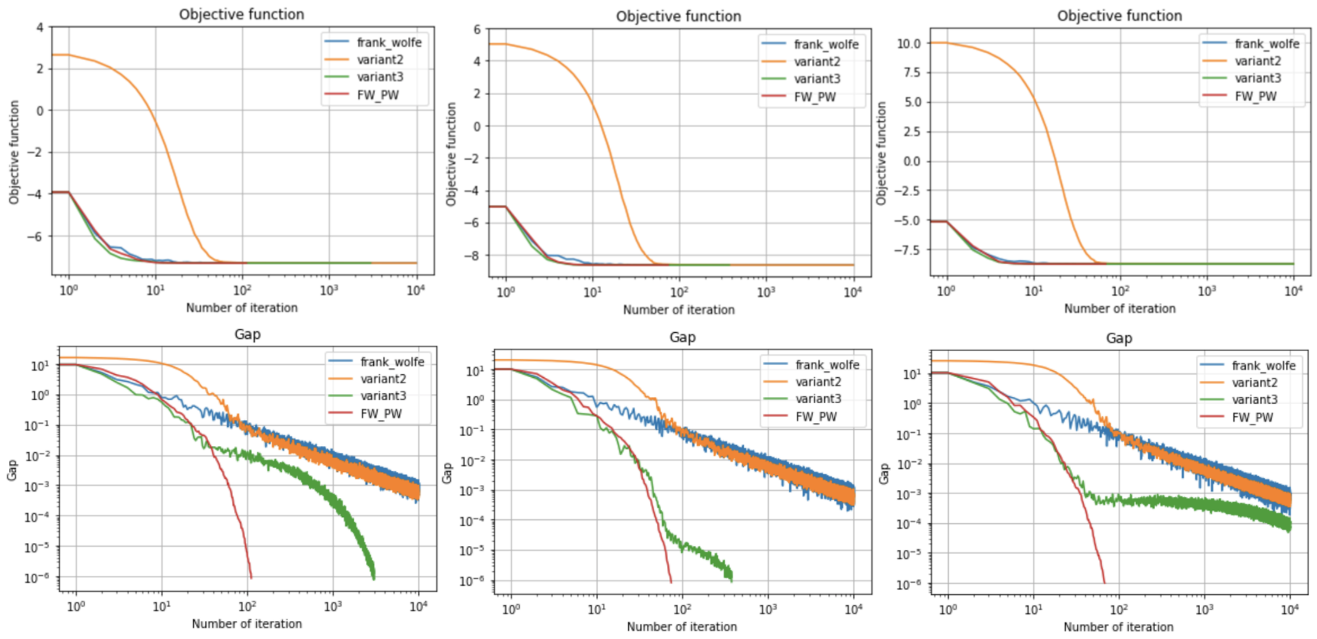

Figure 2. Objective and gap for number of periods = to 800, 1200, 1500

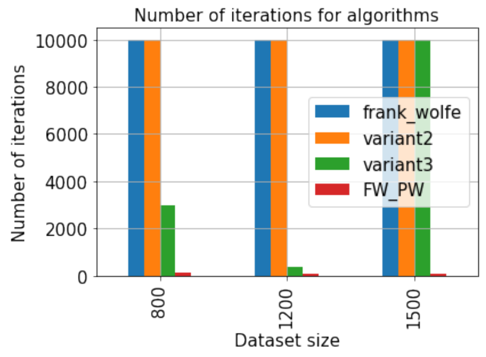

Figure 3. Number of iterations, used by algorithms to accomplish optimization for datasets with 800, 1200 and 1500 assets

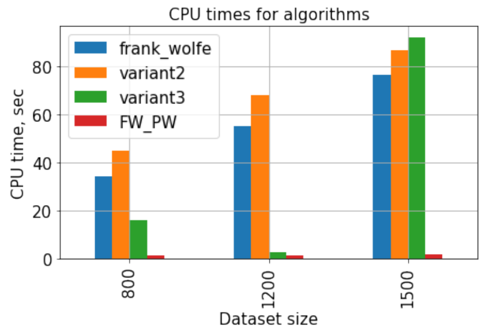

Figure 4. Computational time, used by algorithms to accomplish optimization for datasets with 800, 1200 and 1500 assets

Again, the variant 2 has a bad behavior in the first 100 iteration, due to the same reason explained before, despite a

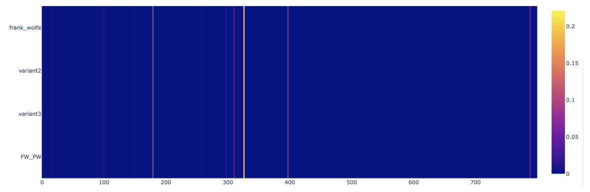

Figure 5. Distribution of investments over 800 assets. Vertical line represents optimization algorithms, versus 800 assets on horizontal axis

faster convergence in the first experiment. While the Pairwise Frank-Wolfe remains the best, and the variant 3 has the fastest decrease at the beginning (until around the 10th iteration), supporting our conclusion on our new heuristic. We notice also how also in this case all algorithms but Pairwise Frank-Wolfe suffer of “zig-zag” problem starting from some point.

5. Conclusion

In the experimental part, three datasets were randomly generated, following the procedure described in [3]. Three studied algorithms were leveraged to solve portfolio optimization problem with synthesised data, and compared based on number of iterations, computational time and objective function.

As a result, pairwise Frank Wolfe method showed best results, requiring approximately 500 times less iterations than the rest of algorithms, reaching early stop condition. The pairwise variation was the best, based on computational

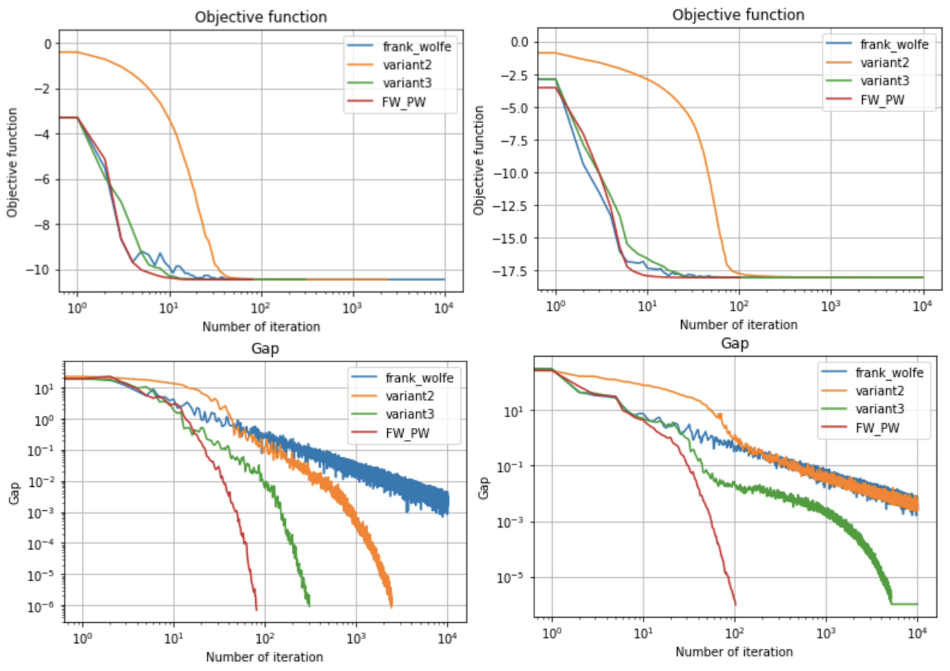

Figure 6. Objective function and gap of matrix r1 and matrix r2

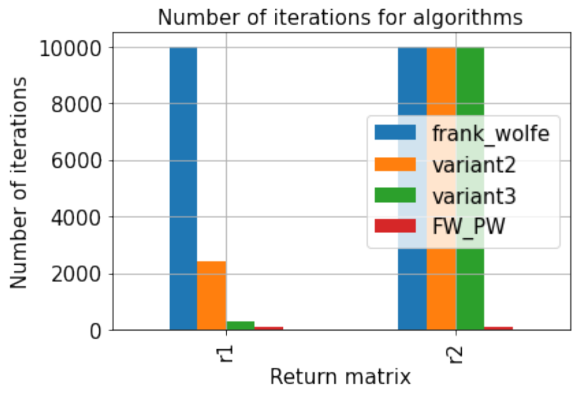

Figure 7. Comparison of iterations, performed by each algorithm on real dataset

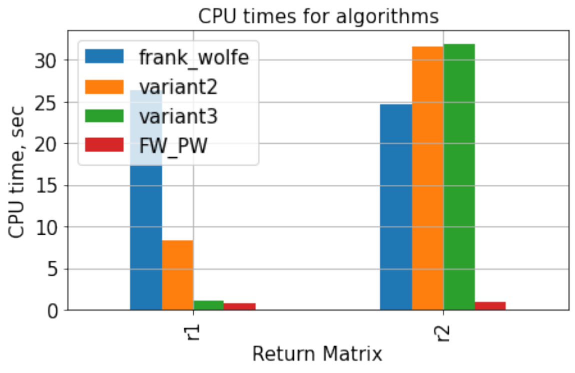

Figure 8. Comparison of computational time, necessary for solving the task for each algorithm

time, and reaching the smallest gap in least number of iterations. According to the asset allocation, all algorithms converged to recommending similar portfolio structure. Regarding the new procedure we introduced we remark how it could be used as effective line search to speed up the variants of Frank-Wolfe algorithms.

References

[1] David Kindness Adam Barone. Asset.

[2] Amanda Bellucco-Chatham Carla Tardi, Thomas Brock. What is a portfolio.

[3] Pavel Dvurechensky, Petr Ostroukhov, Kamil Safin, Shimrit Shtern, and Mathias Staudigl. Self-concordant analysis of frank-wolfe algorithms-supplementary material.

[4] Pavel Dvurechensky, Petr Ostroukhov, Kamil Safin, Shimrit Shtern, and Mathias Staudigl. Self-concordant analysis of frank-wolfe algorithms. In *International Conference on Machine Learning*, pages 2814–2824. PMLR, 2020.

[5] J. Guélat and Marcotte. Some comments on wolfe's 'away step'. 1986.

[6] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pages 427–435. PMLR, 2013.

[7] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. *Advances in neural information processing systems*, 28, 2015.

[8] Tianxiao Sun and Quoc Tran-Dinh. Generalized self-concordant functions: a recipe for newton-type methods.

Mathematical Programming, 178(1):145–213, 2019.