University of Padova
Department of Mathematics

Frank-Wolfe: Optimization for Recommender Systems

Optimization for Data Science

Contributors: Max Hans-Jürgen Henry Horn
Mikhail Isakov
Lennart Niels Bredthauer

Supervisor: Prof. Dr. Francesco Rinaldi

Date: July 2025

Table of Contents

Abstract

This report investigates the application of Frank-Wolfe algorithms and their modern variants to the matrix completion problem in recommender systems. We explore classical Frank-Wolfe with diminishing step size and line-search, as well as enhanced in-face Away-Step and Pairwise Frank-Wolfe methods that leverage the geometry of minimal faces to accelerate convergence. Using three real-world datasets of increasing scale (MovieLens 100k, Jester Jokes, MovieLens 1M), we empirically evaluate the convergence rates. Our experiments confirm theoretical expectations, that the classical Frank-Wolfe exhibits sublinear convergence and pronounced zig-zagging, in-face variants achieve faster reductions in the duality gap, while maintaining more compact active sets. The findings highlight the practical benefits of in-face optimization for large-scale matrix completion tasks.

1 Introduction

Matrix completion [1] is a fundamental problem that arises in numerous fields, including collaborative filtering, machine learning, control, remote sensing, and computer vision. In the context of this work, we focus on its application to recommender systems and their optimization. Recommender systems aim to suggest relevant items (such as movies, books, or products) to users based on historical interaction data. These data are typically represented as a large, sparse matrix, where rows correspond to users, columns correspond to items, and the observed entries represent known preferences or ratings.
Our objective is to accurately infer the missing values in this partially observed matrix, to predict unknown user-item interactions, by recovering a low-rank approximation of the complete matrix. Such a low-rank structure reflects the assumption that user preferences and item characteristics can be captured by a small number of latent factors. In this work, we explore how variants of the Frank-Wolfe algorithm can be effectively applied to solve this matrix completion problem, taking advantage of their projection-free updates and scalability to large, high-dimensional datasets.

Matrix Completion Problem

The matrix completion problem [1] focuses on retrieving a low-rank matrix $X \in \mathbb{R}^{n_1 \times n_2}$ from a sparse set of observed matrix entries $\{U_{ij}\}_{(i,j)\in\mathcal{J}}$ with $\mathcal{J} \subset [1 : n_1] \times [1 : n_2]$, which fits the known entries.

The problem is formulated as:

$$\min_{X\in\mathbb{R}^{n_1\times n_2}} \quad f(X) := \sum_{(i,j)\in\mathcal{J}} (X_{ij} - U_{ij})^2$$

$$\text{s.t.} \quad rank(X) \leq \delta$$

The function f is defined as the squared loss over the observed entries of the matrix U and the recovered matrix X, where the rank is implicitly bounded with the constraining parameter $\|X\|_* \leq \delta$. This low-rank constraint is relaxed to a nuclear norm ball constraint. The nuclear norm $\|.\|_*$ of a matrix X is defined as the sum of its singular values.
This leads to the following convex optimization problem:

$$\min_{X\in\mathbb{R}^{n_1\times n_2}} \quad \sum_{(i,j)\in\mathcal{J}} (X_{ij} - U_{ij})^2$$

$$\text{s.t.} \quad \|X\|_* \leq \delta$$

The nuclear norm $\|.\|_*$ of a matrix X is defined as:

$$\|X\|_* := \sum_{t=1}^{\min(m,n)} \sigma_t(X)$$

where $\sigma_t(X)$ are the singular values of X. This formulation encourages low-rank solutions because the nuclear norm is the tightest convex relaxation of the matrix rank function:

$$\text{rank}(X) \approx \|X\|_* \quad \text{(convex surrogate)}$$

The constraint $\|X\|_* \leq \delta$ restricts the complexity of the solution and promotes generalization by preventing overfitting to the observed entries in $\mathcal{J}$.
The set of feasible low-rank matrices satisfying the above criteria is defined as follows: $\mathcal{D} = \{X \in \mathbb{R}^{n_1\times n_2} : \|X\|_* \leq \delta\} = \text{conv}\left\{\delta\, uv^\top : u \in \mathbb{R}^{n_1},\ v \in \mathbb{R}^{n_2},\ \|u\| = \|v\| = 1\right\}$.
As a result, the Frank-Wolfe algorithm incrementally reconstructs the target matrix as a sparse combination of rank-1 matrices. Moreover, since the gradient matrix $\nabla f(X)$ is sparse when it only contains $|\mathcal{J}|$ non-zero entries, then the computation of singular vectors is more efficient than for dense matrices [1].

2 Frank-Wolfe

We consider the problem of constrained convex optimization, where the objective function f is assumed to be convex and continuously differentiable, and the feasible domain $\mathcal{D}$ is a compact convex subset of a vector space. Among the earliest methods developed for solving such problems is the Frank-Wolfe (FW) algorithm [2], also known as the conditional gradient method. Due to its simplicity and projection-free updates, it has recently regained popularity, especially in the context of large-scale and sparse optimization tasks commonly encountered in machine learning [3].
The classical FW algorithm [2] iteratively approximates the solution by solving a linear minimization oracle (LMO) over the feasible domain,

rather than relying on orthogonal projections. This projection-free nature makes Frank-Wolfe methods especially appealing in scenarios where the constraint set has a complex or structured geometry. More generally, the family of Frank-Wolfe algorithms is characterized by their ability to maintain iterates as convex combinations of a small number of vertices, which is particularly advantageous for large-scale problems and settings that benefit from sparse or low-rank representations.

2.1 Classical Frank-Wolfe

The classical FW algorithm is designed to solve constrained convex optimization problems of the form:

$$\min_{x \in D} f(x)$$

where $f(x)$ is a convex and continuously differentiable function, and $\mathcal{D}$ is a compact convex set.

The algorithm iteratively refines the solution by solving a linear subproblem at each step to identify an improving direction:

$$s_t = \arg\min_{s \in D} \langle \nabla f(x_t), s \rangle$$

Then, the current iterate x_t is updated via a convex combination with the newly found direction s_t:

$$x_{t+1} = (1 - \gamma_k)x_t + \gamma_t s_t$$

where $\gamma_t \in [0, 1]$ is a step size parameter, typically determined by a line search or a predefined size.

The classical FW algorithm can be described in the following steps:

1. **Initialization:** Start with an initial point $x_0 \in \mathcal{D}$.
2. **Linear Subproblem:** Solve the linear minimization oracle to find the FW direction s_t.
3. **Step Size Selection:** Compute/Choose the step size γ_t.
4. **Update:** Perform the update to obtain x_{t+1}.
5. **Termination:** Check convergence criteria; if satisfied, terminate. Otherwise, repeat from step 2.

Algorithm 1 Classical Frank-Wolfe Algorithm

```
Require: Convex function $f$, compact convex set $\mathcal{D}$
1: Initialize: $x_0 \in \mathcal{D}$
2: for $t = 0 \dots T$ do
3:     Compute the linear subproblem:
           $s_t = \arg\min_{s \in D} \langle \nabla f(x_t), s \rangle$
4:     Update the solution:
           $x_{t+1} = (1 - \gamma_t)x_t + \gamma_t s_t$
5:     Choose step size $\gamma_t$
6: end for
```

2.2 Analysis of Convergence

For the classical Frank-Wolfe algorithm (Algorithm 1), it is well known that the iterates satisfy a sublinear convergence rate [2]. Specifically, the primal error decreases as:

$$f(x^{(t)}) - f(x^*) \leq \mathcal{O}\left(\frac{1}{t}\right),$$

where x^* is an optimal solution to the problem. This rate holds for general convex and smooth objectives without requiring strong convexity [3].

2.3 Convergence Guarantees

Convergence guarantees [2] for Frank-Wolfe algorithms provide upper bounds on the optimization error at each iteration t. Primal convergence rates characterize how fast the objective value approaches the optimum, while primal-dual convergence rates provide practical certificates of optimality through the duality gap, discussed in 2.4. Together, these results introduce a comprehensive understanding of efficiency and stopping criteria for all discussed algorithms (1, 2, 3).

Curvature Constant

The convergence analysis of Frank-Wolfe algorithms relies on the curvature constant C_f [2], which measures the non-linearity of the objective function f over the domain $\mathcal{D}$. For a convex and differentiable function f, the curvature constant is defined as:

$$C_f := \sup_{\substack{x, s \in \mathcal{D} \\ \gamma \in [0,1]}} \frac{2}{\gamma^2} \left[f(y) - f(x) - \langle y - x, \nabla f(x) \rangle \right], \quad \text{where} \quad y = x + \gamma(s - x).$$

The quantity $f(y) - f(x) - \langle \nabla f(x), y - x \rangle$ is known as the Bregman divergence, and measures how much the function deviates from its linear approximation at x. Intuitively, C_f captures the worst-case curvature of f over $\mathcal{D}$. We get a completely linear function, if $C_f = 0$. Furthermore, C_f is bounded if ∇f is Lipschitz continuous, and typically depends on the size of the domain $\mathcal{D}$, often scaling with the squared diameter of $\mathcal{D}$. Moreover, if the gradient ∇f is L-Lipschitz continuous with respect to a given norm, it holds that $C_f \leq \operatorname{diam}_{\|\cdot\|}(\mathcal{D})^2 \cdot L$, where $\operatorname{diam}(\mathcal{D})$ is the diameter of the domain. This relation highlights the influence of domain size on the convergence rate.
A larger curvature constant indicates stronger curvature and results in slower convergence of the algorithm, since the convergence rate depends directly on C_f.

Primal Convergence

Primal convergence [2] guarantees show that after $\mathcal{O}(1/\epsilon)$ iterations are ϵ-approximate solutions, meaning $f(x^{(t)}) \leq f(x^*) + \epsilon$.
Theorem 1 [2]: The primal error is bounded by:

$$f(x^{(t)}) - f(x^*) \leq \frac{2C_f}{t+2}(1+\delta), \tag{1}$$

where C_f is quantifying how strongly the function deviates from linearity over the feasible set. The parameter $\delta \geq 0$ measures the accuracy with which the internal linear subproblems (LMO) are solved, with $\delta = 0$ corresponding to exact solutions. The iterations are represented with t.

Primal-Dual Convergence

Obtaining a guaranteed small duality gap is crucial in practice, since both the optimum value $f(x^*)$ and the curvature constant C_f are typically unknown. The duality gap $g(x)$, as defined in Section 2.4, is an efficiently computable quantity that always upper bounds the primal error $f(x) - f(x^*)$, and thus serves as a useful certificate of optimality [2].

The following result shows that all variants of the Frank-Wolfe algorithm (Algorithms 1, 2, 3) obtain a guaranteed small duality gap after $\mathcal{O}(1/\epsilon)$ iterations, even when the linear subproblems are solved approximately.

Theorem 2 [2]: If any Frank-Wolfe variant is run for $T \geq 2$ iterations, then there exists an iterate $x^{(t)}$, with $1 \leq t \leq T$, satisfying:

$$g(x^{(t)}) \leq \frac{2\beta C_f}{T+2}(1+\delta), \tag{2}$$

where $\beta = \frac{27}{8} = 3.375$ and the iterations are represented with T.
The primal and primal-dual convergence guarantees ensure that all discussed Frank-Wolfe variants achieve at least an iteration complexity of $\mathcal{O}(1/\epsilon)$ and to reach the duality gap $g(x^{(t)}) \leq \epsilon$, independent from any assumptions on strong convexity.
As a result, they provide both theoretical insight and practical guidance for optimizing large-scale problems such as matrix completion.

Under additional conditions of strong convexity and favorable polytope geometry, Frank-Wolfe Away Step (AFW) and Frank-Wolfe Pairwise (PFW) can further improve this rate to global linear convergence, leading to significantly faster progress in practice [3], which is discussed later.

2.4 Variants and Methods

The Frank-Wolfe algorithms allow flexibility in choosing the step size and stopping criteria. In the following, key elements are described.

Standard Step Size

A fixed step size is often used in Frank-Wolfe algorithms when allowing approximate solutions of the linear subproblem with quality parameter $\delta \geq 0$ [1]. In this setting, inexact solutions are permitted to reduce computational cost. The condition for the approximate direction $\tilde{s}$ [2] is:

$$\langle \tilde{s}, \nabla f(x^{(t)}) \rangle \leq \min_{s' \in \mathcal{D}} \langle s', \nabla f(x^{(t)}) \rangle + \frac{1}{2}\delta\gamma C_f.$$

The step size is then chosen as:

$$\gamma_t := \frac{2}{t+2}.$$

Once the approximate direction $\tilde{s}$ is found, the current iterate is updated as in Algorithm 1.

Line Search

Instead of using a fixed step size, the convergence rate can be improved by dynamically selecting the optimal step size γ_t, chosen from the interval $[0, 1]$, at each iteration:

$$\gamma_t := \arg\min_{\gamma\in[0,1]} f\left(x^{(t)} + \gamma(s - x^{(t)})\right)$$

This is the exact line search [2], which is particularly useful when the objective function f allows for efficient exact minimization along the search direction. In other words, the best point on the line segment between the current iterate $x^{(t)}$ and s is picked.
In practice, the choice of step size depends on the structure of the objective function. For more complex objectives or when exact line search is too costly, Armijo-type line search offers a robust alternative, guaranteeing sufficient decrease at each iteration without requiring knowledge of C_f [1].

Duality Gap

As discussed in subsection 2.3, the true optimum $f(x^*)$ and the curvature constant C_f are typically unknown in practice. Therefore, the duality gap [2] is introduced as a practical certificate of optimality.

The duality gap $g(x)$ serves as an approximation to the optimal solution x^*, since computing the exact optimum $f(x^*)$ is often computationally expensive or infeasible [1, 2]. The duality gap measures the maximum possible linear improvement at the current iterate and is defined as:

$$g(x) := \max_{s\in\mathcal{D}} \langle x - s, \nabla f(x)\rangle \qquad (3)$$

Because $g(x) \geq f(x) - f(x^*)$, it provides an upper bound on the primal error (see Theorem 1). In practice, a target threshold ϵ is chosen, and the algorithm terminates once $g(x) \leq \epsilon$, ensuring that the current solution is sufficiently close to optimality, guaranteed by Theorem 2, which holds after $\mathcal{O}(1/\epsilon)$ iterations.

Linear Minimization Oracle

The Frank-Wolfe methods are projection-free algorithms, as introduced in Section 2. In our variants, the linear minimization is used to compute the steepest descent direction at each iteration (see Algorithm 1, line 3).

This central component of the Frank-Wolfe framework is the linear minimization oracle (LMO) [1, 2, 4], which replaces the projection steps commonly used in gradient-based methods. The LMO solves the following subproblem at each iteration:

$$s_t := \arg\min_{s\in\mathcal{D}} \langle s, \nabla f(x^{(t)})\rangle.$$

This is fundamentally different from projected gradient methods [4], which require solving a projection problem of the form:

$$\min_{x\in\mathcal{D}} \|x - y\|.$$

In projection-based methods, y typically corresponds to an intermediate gradient step, such as $y = x^{(t)} - \eta\nabla f(x^{(t)})$, which may fall outside the feasible set $\mathcal{D}$, while x is the projected point that restores feasibility. In contrast, the Frank-Wolfe algorithm maintains feasibility at each iteration by selecting a direction s_t within $\mathcal{D}$ via the LMO. This makes the algorithm more efficient.
In matrix completion problems with nuclear norm constraints, the solution to the LMO corresponds to a rank-1 matrix. This can be efficiently computed by performing a singular value decomposition (SVD) of the gradient $\nabla f(x^{(t)})$, where only the top singular vectors are required [1, 2]. As a result, each iteration involves a low-rank update direction.

3 Frank-Wolfe Away Step

One limitation of the classical Frank-Wolfe algorithm, discussed in Section 2.1, is the *zig-zagging* phenomenon [3]. When the optimal solution x^* lies on the boundary of the feasible set $\mathcal{D}$, the algorithm tends to alternate between vertices or atoms that define the face containing x^*, leading to slow progress and a sublinear convergence rate $\mathcal{O}(1/t)$ [2, 1]. The issue is particularly pronounced when $\mathcal{D}$ is a polyhedral, as iterates may become trapped in inefficient zig-zag movements along the boundary.

3.1 Algorithm Description

To address this, variants such as the Away-Step Frank-Wolfe (AFW) introduce an additional *away direction*, allowing the algorithm to remove or move away from poorly contributing atoms and correct the active set of atoms. This reduces zig-zagging and leads to faster convergence [3].

The Away-Step Frank-Wolfe [3] algorithm can be described in the following steps in Algorithm 2:

1. **Initialization:** Start with an initial point $x^{(0)} \in \mathcal{D}$, and initialize the active set $S^{(0)} = \{x^{(0)}\}$ (line 1).
2. **Compute Direction:** Compute the FW direction $s_t := \text{LMO}(\nabla f(x^{(t)}))$. Select the away atom $v_t := \arg\max_{v \in S^{(t)}} \langle \nabla f(x^{(t)}), v \rangle$. Set descent directions $d_t^{\text{FW}} := s_t - x^{(t)}$ and $d_t^{\text{A}} := x^{(t)} - v_t$ (line 3).
3. **Step Type and Step Size:** Decide between Frank-Wolfe or Away Step based on gradient information. Set $\gamma_{\max}$, and compute the step size γ_t, typically with line search (lines 7-12).
4. **Update:** Update the iterate $x^{(t+1)} := x^{(t)} + \gamma_t d_t$. Update the active set $S^{(t+1)} := \{v \in \mathcal{A} \mid \alpha_v^{(t+1)} > 0\}$ (line 13).
5. **Termination:** Stop if duality gap $g(x^{(t)}) \leq \epsilon$, otherwise repeat (lines 4 and 14).

The away direction d_t^{A} is computed by selecting the worst contributing atom $v_t \in S^{(t)}$, i.e., the atom maximizing $\langle \nabla f(x^{(t)}), v \rangle$. Since this search is performed only over the typically small active set $S^{(t)}$, it is computationally much cheaper than the classical FW algorithm. The conservative step size $\gamma_{\max}$ ensures that the updated iterate remains in the convex hull of $S^{(t)}$, i.e., $\text{conv}(S^{(t)}) \subseteq \mathcal{D}$. Maintaining the active set is required for AFW, in contrast to standard FW [3].

Algorithm 2 Away-step Frank-Wolfe Algorithm

```
Require: Initial point $x^{(0)}$, atom set $\mathcal{A}$, tolerance $\epsilon$ and set of active atoms $S^{(0)}$
 1: $S^{(0)} := \{x^{(0)}\}$
 2: for $t = 0 \dots T$ do
 3:     Compute:
            $s_t := \text{LMO}_{\mathcal{A}}(\nabla f(x^{(t)}))$
            $d_t^{\text{FW}} := s_t - x^{(t)}$
            $v_t := \arg\max_{v \in S^{(t)}} \langle \nabla f(x^{(t)}), v \rangle$
            $d_t^{\text{A}} := x^{(t)} - v_t$
 4:     if $\langle -\nabla f(x^{(t)}), d_t^{\text{FW}} \rangle \leq \epsilon$ then
 5:         return $x^{(t)}$
 6:     end if
 7:     if $\langle -\nabla f(x^{(t)}), d_t^{\text{FW}} \rangle \geq \langle -\nabla f(x^{(t)}), d_t^{\text{A}} \rangle$ then
 8:
            $d_t := d_t^{\text{FW}}, \quad \gamma_{\max} := 1$
 9:     else
10:
            $d_t := d_t^{\text{A}}, \quad \gamma_{\max} := \frac{\alpha_{v_t}}{1 - \alpha_{v_t}}$
11:     end if
12:     Line-search:
            $\gamma_t := \arg\min_{\gamma \in [0, \gamma_{\max}]} f(x^{(t)} + \gamma d_t)$
13:     Update:
            $x^{(t+1)} := x^{(t)} + \gamma_t d_t$
            $S^{(t+1)} := \{v \in \mathcal{A} \mid \alpha_v^{(t+1)} > 0\}$
14: end for
```

Weight updates:

The algorithm adjusts the weights $\alpha_v^{(t)}$ of the atoms in $S^{(t)}$ depending on whether a Frank-Wolfe Step or an Away Step is taken:

Frank-Wolfe step:

$$\alpha_{s_t}^{(t+1)} = (1-\gamma_t)\alpha_{s_t}^{(t)} + \gamma_t$$

$$\alpha_v^{(t+1)} = (1-\gamma_t)\alpha_v^{(t)} \quad \forall v \in S^{(t)} \setminus \{s_t\}$$

Away-step:

$$\alpha_{v_t}^{(t+1)} = (1+\gamma_t)\alpha_{v_t}^{(t)} - \gamma_t$$

$$\alpha_v^{(t+1)} = (1+\gamma_t)\alpha_v^{(t)} \quad \forall v \in S^{(t)} \setminus \{v_t\}$$

If $\gamma_t = \gamma_{\max}$, this is called a *drop step* and the atom v_t is removed from the active set $S^{(t)}$.

This mechanism enables AFW to adaptively correct the active set by reducing the influence of poorly aligned atoms and fully removing them when necessary (drop step) [3].

As a result, AFW avoids inefficient zig-zagging and accelerates convergence [3].

3.2 Analysis of Convergence

The global linear convergence of the AFW algorithm is derived from its ability to remove poorly contributing atoms from the active set through away steps [3]. The convergence depends on whether a so-called good step is performed, i.e., when the step size satisfies $\gamma_t < \gamma_{\max}$. When $\gamma_{\max}$ becomes small, typically due to a low weight of the away atom v_t, the algorithm performs a drop step ($\gamma_t = \gamma_{\max}$), which removes v_t from the active set. Drop steps do not guarantee sufficient progress but reduce the active set size. The number of drop steps can be bounded. They cannot occur in more than half of the iterations. The convergence proof balances the number of good steps and drop steps and establishes global linear convergence of AFW for strongly convex objectives and polytope constraints [3].

3.3 Away-Step Frank-Wolfe with In-Face Optimization

Classic Frank-Wolfe variants such as AFW aim to accelerate convergence by selecting descent directions not only toward new atoms (as in standard FW) but also away from previously chosen atoms in the active set. However, these directions are often limited to the current outer approximation of the feasible set. To enhance the effectiveness of AFW, we incorporate *in-face directions* [5] by leveraging the structure of the *minimal face* $\mathcal{F}(x^t)$, defined as the smallest face of the feasible set $\mathcal{D}$ that contains the current iterate x^t.

This design is theoretically motivated by the concept of the *pyramidal width* [3]. The pyramidal width $\mathrm{PWidth}(A)$ quantifies how well-aligned the best feasible descent direction $d = s - v$ is with the gradient $r := -\nabla f(x^t)$. Formally, it lower bounds the alignment:

$$\frac{\langle r, d\rangle}{\langle r, \hat{e}\rangle} \geq \mathrm{PWidth}(A),$$

where $\hat{e}$ is the normalized error direction. Since this bound becomes tighter when descent directions are restricted to a face of the domain with better geometry, we compute away directions that lie *within the minimal face.*

To compute the in-face direction D^t, we exploit a low-rank factorization of X_t using a thin SVD: $X_t = USV^\top$. We then define a small matrix G^t as:

$$G^t = U^\top \nabla f(X_t) V,$$

and symmetrize it to obtain $\mathrm{sym}(G^t) = \frac{1}{2}(G^t + G^{t\top})$. The optimal matrix $M^\star$ in the low-rank space is computed by maximizing the LMO:

$$M^t = \delta \cdot u_t u_t^\top,$$

where u_t is the eigenvector corresponding to the largest eigenvalue of $\mathrm{sym}(G^t)$. The updated in-face point is then:

$$Z_t = UM^tV^\top, \qquad D_{\text{in-face}} = X_t - Z_t.$$

We then compute the maximum allowable step-size along this direction, which ensures feasibility with respect to the nuclear norm constraint $\|X\|_* \leq \delta$. Based on the analysis in [5], we compute:

$$\alpha_t^{\text{stop}} = -\left(\lambda_{\min}\left(D^\top \Delta D\right)\right)^{-1},$$

where Δ represents the change in direction projected onto the low-rank tangent space of the face. This ensures that the update remains within the feasible region:

$$X_{t+1} = X_t + \alpha D^t, \quad \text{with } \alpha \leq \alpha_t^{\text{stop}}.$$

Thus, our algorithm selects between the standard FW direction and the in-face direction by comparing their respective duality gaps, and updates accordingly.

4 Pairwise Frank-Wolfe

While the AFW algorithm (Algorithm 2, Section 3.1) mitigates the zig-zagging effect and significantly improves over the classical FW method, its uniform rescaling of all active weights at each iteration is still suboptimal when the solution lies on a low-dimensional face of $\mathcal{D}$. In such cases, a more targeted update mechanism can further enhance sparsity and convergence efficiency [3].

4.1 Algorithm Description

The PFW algorithm inherits its structure from the AFW algorithm (Algorithm 2) but differs in the way weight mass is updated between atoms. Instead of applying a uniform rescaling to all active weights, PFW performs a more targeted update, where weight is transferred directly between two atoms at each iteration. In detail, mass is moved from a selected away atom v_t to the FW atom s_t, while all other weights remain unchanged. This selective mechanism is effective in promoting sparse solutions with smaller active sets and leads to better practical performance compared to AFW [3].
This is implemented by substituting the following:

Algorithm 3 Pairwise Frank-Wolfe algorithm: PFW$(x^{(0)}, \mathcal{A}, \epsilon)$

```
1: ... as in Algorithm 2, except replacing in lines 7 to 11 by:
       $d_t = d_t^{\mathrm{PFW}} := s_t - v_t, \quad \gamma_{\max} := \alpha_{v_t}$
```

Weight updates:

The weight updates in PFW follow, that at iteration t, weight is reduced on v_t and increased on s_t, with all other active weights remaining unchanged. This is referred to as a pairwise FW step [3]:

$$\alpha_{v_t}^{(t+1)} = \alpha_{v_t}^{(t)} - \gamma_t, \qquad \alpha_{s_t}^{(t+1)} = \alpha_{s_t}^{(t)} + \gamma_t,$$

with step size $\gamma_t \leq \gamma_{max} = \alpha_{v_t}^{(t)}$.

In contrast, AFW uniformly shrinks all active weights at each iteration, which can slow down the correction of suboptimal atoms. By focusing updates on the two most relevant atoms, PFW offers a more efficient mechanism.

4.2 Analysis of Convergence

The PFW algorithm achieves a similar global linear convergence guarantee [3]. The selective update rule allows PFW to prune the active set more effectively, especially when the optimal solution lies on a low-dimensional face of $\mathcal{D}$. However, PFW introduces the possibility of swap steps, where $\gamma_t = \gamma_{\max}$, but the atom v_t is not removed from the active set. Instead, its weight is fully swapped to s_t, leaving the size of $S^{(t)}$ unchanged. The number of swap steps is theoretically difficult to bound and is known to be loose. Nonetheless, in practice, PFW often outperforms AFW for sparse problems due to its more targeted weight updates [3].

4.3 Pairwise Frank-Wolfe with In-Face Optimization

Our in-face PFW builds upon the same geometric ideas and implementation as the in-face AFW described in Section 3.3. The main difference lies how the direction is used. Instead of choosing between $s_t - x_t$ and $x_t - v_t$, the pairwise method always uses the combined direction: $s_t - v_t$.

In our implementation, we compute Z_t using the same SVD-based in-face procedure and define the in-face direction as:

$$D_t = S_t - Z_t,$$

where S_t is the new FW atom and Z_t is the in-face point. As before, the feasible step size is bounded using the same formula for α_t^{stop} to ensure nuclear-norm feasibility. The duality gap is used to guide the step acceptance.

Since the key machinery is identical, we refer the reader to the in-face AFW in section 3.3 for a detailed description of the in-face direction computation and step-size control, and to the related literature [5].

5 Code Implementation

Our implementation is structured around an abstract `BaseOptimizer` class, which encapsulates the common operations required by all Frank-Wolfe variants. These include gradient computation, the linear minimization oracle (LMO), line search, and duality gap evaluation. This design promotes code reuse and modularity, making it straightforward to implement and extend different variants such as FW 1, AFW 2, and PFW 3. Simpler variants typically rely on a standard `step()` method, while more advanced approaches (e.g., `FWAwayStepInFace`) introduce additional helper functions to maintain clarity and separation of concerns.

The observed matrix U_{obs} is stored as a sparse matrix (`csr_matrix`) to avoid unnecessary computation or memory use on missing entries. The iterate X is projected onto the observed indices by direct indexing over the nonzero elements of U_{obs}, enabling efficient sparse updates and gradient computations restricted to the observed data. This substantially reduces computational overhead.

The LMO (subsection 2.4) is implemented using a rank-1 sparse SVD via `scipy.sparse.linalg.svds` with $k = 1$, efficiently extracting the top singular vectors of the gradient as required by the Frank-Wolfe framework. The line search exploits the quadratic structure of the objective, allowing it to be solved in closed form and thus avoiding iterative procedures. We implemented both a diminishing step size approach (subsection 2.4) and an exact line search strategy (subsection 2.4), providing flexibility to convergence behavior in different problem scenarios.

Convergence is monitored at each iteration using the duality gap (subsection 2.4), which also serves as the stopping criterion. The object-oriented design ensures the implementation remains modular, maintainable, and easily extensible to support algorithm variants such as AFW and PFW.

For each dataset and algorithm, we tracked the evolution of the duality gap and CPU time, with a convergence threshold set from $\epsilon = 10^0$ to $\epsilon = 10^{-1.5}$. To facilitate comparative analysis, we generated plots of the duality gap versus iteration count, marking the stopping threshold with a horizontal dashed red line to visually confirm convergence behavior.

6 Datasets

Three experiments were made to test the implementation of our algorithms. In the following sections, we will provide an overview of the tested data for the Matrix Completion Problem.

6.1 Dataset 1: MovieLens 100k

The *MovieLens ml-latest-small* dataset [6] contains approximately 100,000 ratings from 610 users on 9,742 movies. Each rating is an ordinal value in the range $[0.5, 5.0]$ in steps of 0.5. After mapping user and item IDs to consecutive indices, we form a sparse rating matrix $R \in \mathbb{R}^{610\times 9742}$.

This dataset is well-suited for low-rank matrix completion due to its sparse structure and the assumption that user preferences can be captured in a low-dimensional latent space. We use it to benchmark the performance of our optimization algorithms under realistic recommendation system settings.

6.2 Dataset 2: Jester Joke Recommender

The *Jester-1* dataset [7] provides continuous ratings collected from 24,983 users on 100 jokes, with values ranging from -10.00 to $+10.00$. The ratings were collected between 1999 and 2003, and each user rated at least 36 jokes. Ratings are stored in a matrix of dimension $24{,}983 \times 101$, where the first column indicates how many jokes were rated by the user, and the remaining columns contain the actual ratings. A value of 99 indicates a missing entry.

This dataset is especially valuable for nuclear norm minimization tasks because it includes real-valued feedback rather than integer scores, and certain joke subsets are densely rated. The continuous scale offers a more fine-grained signal, aligning well with assumptions made in convex relaxation methods.

6.3 Dataset 3: MovieLens 1M

The *MovieLens 1M ml-1m* dataset [8] consists of 1,000,209 ratings from 6,040 users on 3,706 movies, with each user having rated at least 20 movies. Ratings are integers in the range $[1, 5]$, forming a relatively dense matrix compared to other datasets.

We represent the data as a sparse matrix $R \in \mathbb{R}^{6040\times 3706}$. Due to its scale and density, the dataset serves as a more challenging benchmark for evaluating the convergence behavior and scalability of our Frank-Wolfe-based optimization algorithms. It enables us to analyze performance on realistic large-scale recommendation tasks.

7 Experiments

7.1 Setup

To systematically evaluate the convergence behavior and computational performance of the different Frank-Wolfe algorithms, we conducted experiments on three distinct matrix completion datasets of increasing scale and sparsity.

MovieLens 100k

We began with the MovieLens 100k dataset. This was downloaded from the GroupLens repository, parsed to extract user-movie interactions, and re-indexed into compact 0-based arrays to ensure right computation. We then performed an 80/20 split into training and testing sets using a fixed random seed for reproducibility. The resulting user-item rating matrix was stored in compressed sparse row (CSR) format for efficient gradient computations. For this dataset, we selected a nuclear norm ball with radius $\delta = 8000$ and limited the algorithms to a maximum of 2500 iterations.

Jester Jokes

The second dataset was the Jester joke recommendation dataset, which contains dense ratings of jokes by users. We read the data from its original *Excel* format and converted missing entries (indicated by a sentinel value of 99) to NaNs. The observed ratings were also stored as a sparse CSR matrix, with a separate mask used to track observed indices. Initialization was performed through a rank-one matrix scaled to match the nuclear norm constraint $\delta = 16000$. For consistency, the same four algorithmic variants were run up to 4000 iterations, with convergence monitored via the duality gap.

MovieLens 1M

Finally, we tested on the larger MovieLens 1M dataset. Like the smaller MovieLens version, the dataset was automatically downloaded and parsed. We applied the same 80/20 split and constructed training and test matrices in CSR format. Here we chose $\delta = 18000$ for the nuclear norm constraint, reflecting the larger scale of the data, and ran the algorithms up to 4000 iterations or until convergence.

Optimization Algorithms

Across all three datasets, we evaluated four variants of the Frank-Wolfe algorithm:

- Classical FW with diminishing step size $\gamma_t = \frac{2}{t+2}$.
- Classical FW with adaptive line-search.
- In-face AFW with line-search, which incorporates away steps to selectively reduce mass on previously chosen atoms.
- In-face PFW with line-search, transferring weight directly between two atoms, using a target rank of 5 for constructing the low-dimensional face.

7.2 Expectation

Based on the theoretical convergence analyses presented in Sections 2.2, 3.2 and 4.2, we expect the following regarding the relative performance of our tested algorithms. We anticipate observing the slowest convergence with the classical Frank-Wolfe algorithm employing a fixed diminishing step size, followed by the Frank-Wolfe variant using line-search. We further expect the in-face AFW to demonstrate accelerated convergence, with the in-face PFW achieving the fastest reduction on the optimality gap. In the next section, we will investigate whether our experimental findings align with these hypotheses.

7.3 Observations

MovieLens 100k Dataset

As shown in Table 1, Figure 1, and Figure 5, all methods achieve comparable reductions of the duality gap for moderate thresholds.

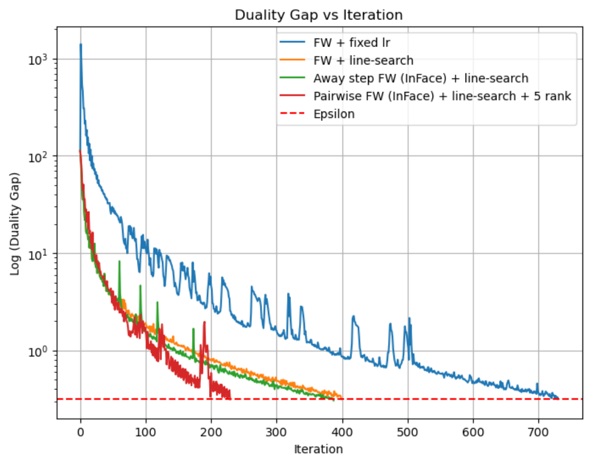

Figure 1: Running solvers on the MovieLens 100k dataset with $\epsilon = 10^{-0.5}$ related to table 1

However, the in-face PFW requires substantially fewer iterations to reach tighter optimality gaps (e.g., $10^{-1.5}$), although with increased CPU time. The duality gap trajectories illustrate that in-face PFW and in-face AFW reduce the gap much more rapidly compared to classical FW, reflecting their enhanced ability to adjust the active set by moving weight directly onto relevant atoms. This underscores how in-face strategies exploit the geometry of the feasible set to maintain low-rank iterates.

Jester Jokes Dataset

The results for the Jester Jokes dataset (Table 2, Figure 2, and Figure 6) further highlight the advantages of in-face variants.

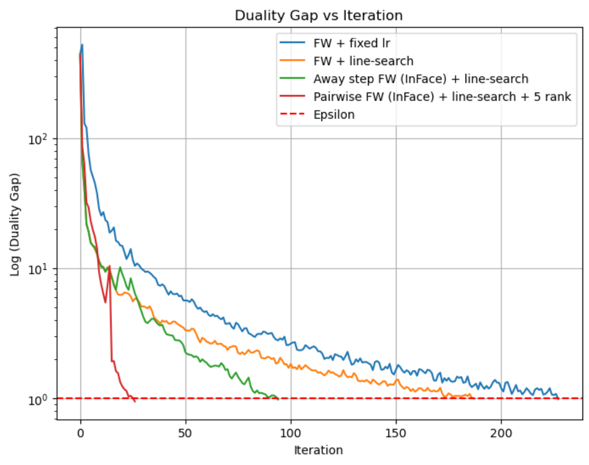

Figure 2: Running solvers on the Jester jokes dataset with $\epsilon = 10^{-0}$ related to table 2

The in-face PFW consistently converges in fewer iterations, due to its precise pairwise mass transfers between atoms. In contrast, classical FW struggles to sufficiently prune the active set, resulting in more iterations and a slower decrease of the duality gap. Meanwhile, the in-face AFW provides an effective balance, improving convergence by leveraging away steps to adjust the active set more globally.

MovieLens 1M Dataset

For the larger MovieLens 1M dataset (Table 3, Figure 3, and Figure 7), in-face PFW again achieves a steeper initial reduction of the duality gap compared to classical FW.

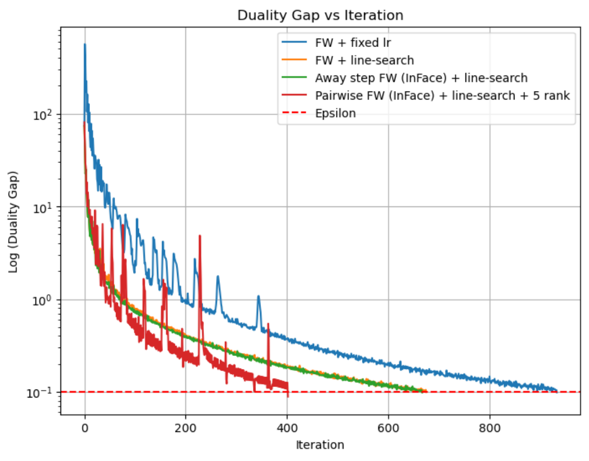

Figure 3: Running solvers on the MovieLens 1M dataset with $\epsilon = 10^{-1}$ related to table 3

We observe more pronounced oscillations, reflecting the aggressive weight shifts inherent in pairwise updates. The in-face AFW shows a more stable convergence pattern, with fewer oscillations but slightly slower pruning of the active set. This represents the trade-off between the corrections in our in-face PFW, that are more focused and local changes, and the global adjustments on the whole active set in the in-face AFW. In addition, in-face AFW is minimally better than the FW with line-search, which was frequently observed in our experiments.

Overall Comparison and Discussion

Across all datasets, the in-face variants (AFW and PFW) demonstrably outperform the classical Frank-Wolfe algorithm in reducing the duality gap and maintaining more compact active sets. The pairwise mechanism employed by in-face PFW, which redistributes weight exclusively between a selected away atom v_t and FW atom s_t, facilitates sharper corrections of the active set, making it particularly effective when the optimal solution resides on a low-dimensional face of the constraint set $\mathcal{D}$. In contrast, the in-face AFW applies a uniform rescaling across all active atoms, resulting in more globally balanced but comparatively slower corrections.

Additionally, as depicted in Figure 4, Figure 8 and summarized in Table 1 of Appendices B and A, the FW algorithm with line-search is observed to reach the prescribed relative optimality gap ϵ slightly earlier than the in-face AFW under the tested parameter configurations.

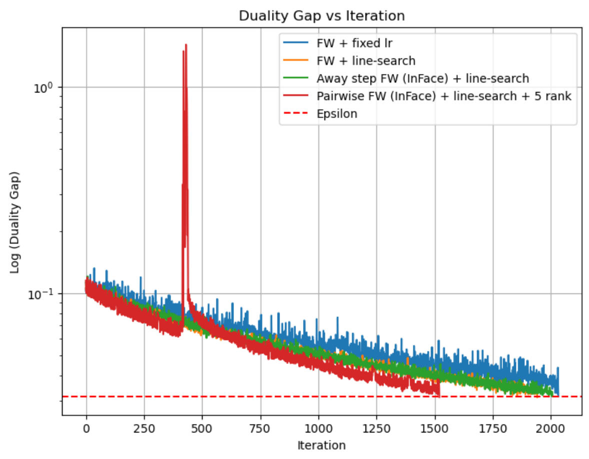

Figure 4: Plot of running solvers on the MovieLens 100k Dataset with $\epsilon = 10^{-1.5}$ related to table 1

This phenomenon is primarily attributed to transient oscillatory behavior, since both algorithms exhibit closely aligned trajectories over the trend of optimization. Consequently, the earlier attainment of ϵ by FW with line-search is likely a consequence of short-lived fluctuations rather than an indication of superior asymptotic convergence. Indeed, the in-face AFW ultimately demonstrates a more sustained reduction of the duality gap, underscoring its advantage in long-run convergence.

To cover all fluctuations, it is noteworthy that the classical FW algorithm exhibits a pronounced initial spike in the duality gap, as seen for instance in Figure 5 in Appendix A. This behavior could arise due to the random initialization of the starting matrix X_0, which typically lies far from the optimal low-rank solution. However, the first computed gradient reflects a large residual, prompting FW to take a substantial step toward a new atom in the feasible set. This aggressive correction manifests as a sharp drop in the duality gap during the first iterations. In contrast, the in-face variants adapt more gradually by searching minimal points in the local geometry of the minimal face, thereby avoiding such pronounced early jumps and leading to smoother initial convergence.

Limitations and Algorithmic Behavior

A key observation is the pronounced fluctuation in the duality gap plots for in-face PFW. This arises because each pairwise step reallocates weight mass only between two atoms, which can overshoot and require subsequent corrections. While this mechanism is powerful for quickly pruning irrelevant atoms and maintaining a small active set, it also introduces temporary instability in the convergence path. Furthermore, since our implementation of the LMO leverages top singular vector computations via SVD, the computational cost per iteration increases, particularly in PFW which relies heavily on these localized corrections. Thus, although in-face PFW achieves fewer overall iterations, it may still be outpaced in wall-clock time by more uniformly adjusting methods such as in-face AFW or classical FW with line search. This highlights the classical trade-off between iteration complexity and per-iteration computational cost in projection-free matrix completion.

8 Conclusion

Through experiments on datasets of varying sparsity and scale, we observed that in-face methods, particularly the Pairwise Frank-Wolfe, are able to achieve faster convergence in terms of iterations by selectively adjusting weights between atoms. However, this often comes at the expense of increased computational effort per iteration.
The in-face Away-Step Frank-Wolfe offered a robust compromise, balancing active set adjustments with smoother convergence trajectories, while the in-face Pairwise algorithm outperforms all the other algorithms tested, in terms of minimal duality gap.
Overall, these results underline the importance of adapting Frank-Wolfe variants to the present data geometry, especially in large-scale Recommender Systems, where maintaining low-rank solutions is critical for predictive accuracy and computational efficiency.

9 References

References

[1] Damiano Zeffiro Immanuel M. Bomze1, Francesco Rinaldi. Frank–wolfe and friends: a journey into projection-free first-order optimization methods. *Foundations and Trends in Machine Learning*, 13(1):314–345, 2021.

[2] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*. JMLR Workshop and Conference Proceedings, 2013.

[3] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. volume 28, 2015.

[4] Sebastian Pokutta Cyrille W. Combettes. Complexity of linear minimization and projection on some sets. *Foundations and Trends in Machine Learning*, 13(1):1–14, 2021.

[5] Rahul Mazumder Robert M. Freund, Paul Grigas. An extended frank-wolfe method with "in-face" directions, and its application to low-rank matrix completion. *SIAM Journal on Optimization*, 27(1):319–46, 2017.

[6] GroupLens Research. Movielens latest small dataset. `https://grouplens.org/datasets/movielens/`, 2023. Accessed: 2025-06-26.

[7] Ken Goldberg et al. The jester online joke recommender system dataset. `https://goldberg.berkeley.edu/jester-data/`, 2003. Accessed: 2025-06-26.

[8] GroupLens Research. Movielens 1m dataset. `https://grouplens.org/datasets/movielens/`, 2003. Accessed: 2025-06-26.

10 Appendix

A Empirical Convergence Analysis

In this section, we empirically analyze the convergence behavior of the proposed algorithms under varying relative optimality gap thresholds ϵ. The value ϵ serves as a stopping criterion and is visualized in the following plots as the red dashed line, indicating the targeted convergence boundary.

We compare the performance of the vanilla Frank-Wolfe algorithm (Algorithm 1), both with a fixed step size and with line search, against the in-face variants of Away-step Frank-Wolfe (Section 3.3) and Pairwise Frank-Wolfe (Section 4.3). The experiments are conducted across different datasets and values of δ, which controls the radius of the nuclear norm ball constraint.

The results highlight the impact of ϵ on the required CPU time (in minutes) and iteration counts until convergence. Each table reports these statistics for one dataset, allowing a detailed comparison of convergence rates and computational efficiency among the different algorithmic variants.

MovieLens 100k Dataset

Table 1: MovieLens100K Dataset ($m = 610$, $n = 9742$, $\delta = 8000$). CPU Time in min and maximal Iteration (Iter) threshold: 2500

Rel. Opt. Gap ϵ	FW fix		FW ls		AFW		PFW	
	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter
10^{0}	1.5	378	0.78	166	1.77	144	2.45	98
$10^{-0.5}$	3.35	731	2.1	400	5.29	388	6.97	230
10^{-1}	8.68	1631	6.38	1042	14.49	992	21.97	660
$10^{-1.5}$	14.28	2034	16.49	1943	34.34	2009	60.93	1522

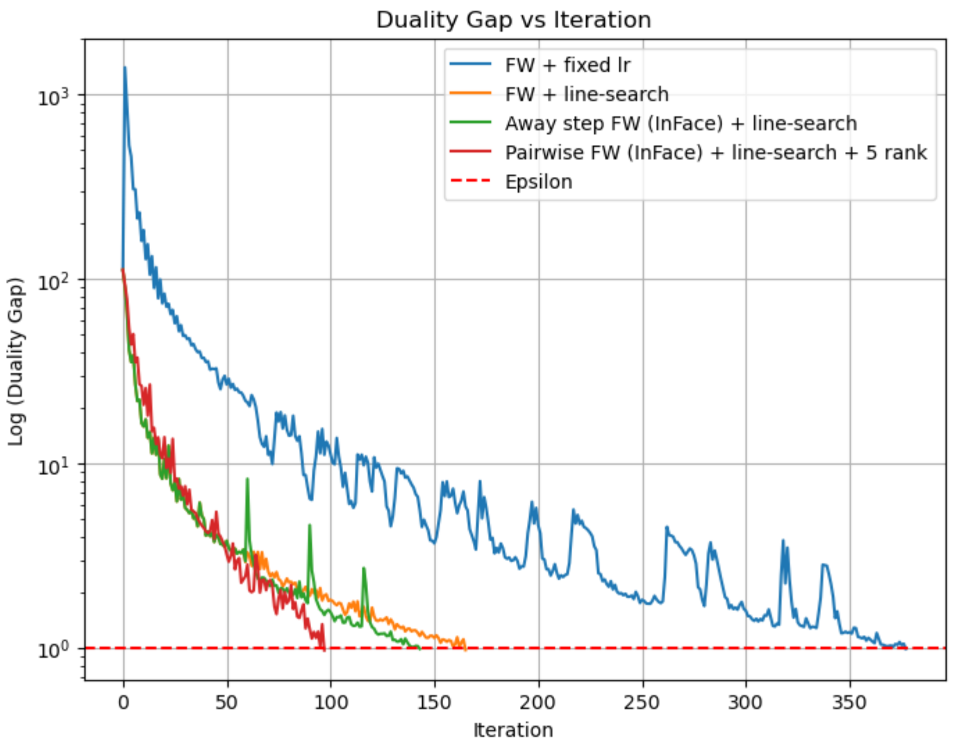

Figure 5: Running solvers on the MovieLens 100k dataset with $\epsilon = 10^{-0}$ related to table 1

Jester Jokes Dataset

Table 2: Jester Jokes Dataset ($m = 24.983$, $n = 100$, $\delta = 16000$). CPU Time in min and maximal Iteration (Iter) threshold: 4000.

Rel. Opt. Gap ϵ	FW fix		FW ls		AFW		PFW	
	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter
10^0	5.82	228	6.24	188	3.84	95	1.22	27
$10^{-0.5}$	21.49	676	26	633	24.52	458	4.87	98
10^{-1}	77.98	2033	96.10	2078	110.02	1862	21.65	384
$10^{-1.5}$	>156	>4000	>175	>4000	>228	>4000	76.59	1508

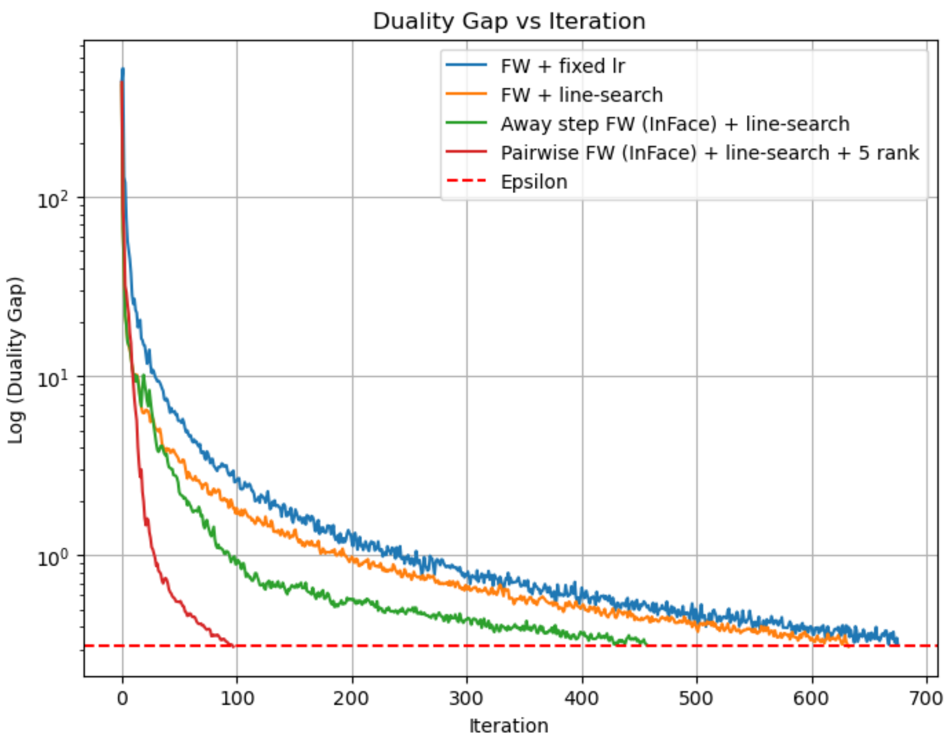

Figure 6: Plot of running solvers on the Jesker Jokes dataset with $\epsilon = 10^{-0.5}$ related to table 2

MovieLens 1M Dataset

Table 3: MovieLens1M Dataset ($m = 6040$, $n = 3706$, $\delta = 18000$). CPU Time in min and maximal Iteration (Iter) threshold: 2500.

Rel. Opt. Gap ϵ	FW fix		FW ls		AFW		PFW	
	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter	Time (min)	Iter
10^{0}	5.48	193	2.93	78	6.04	70	7.99	53
$10^{-0.5}$	16.99	434	10.94	250	23.39	244	21.19	141
10^{-1}	40.95	934	33.49	676	68.66	668	69.71	403
$10^{-1.5}$	107.77	2228	101.02	1922	198.34	1900	208.54	1237

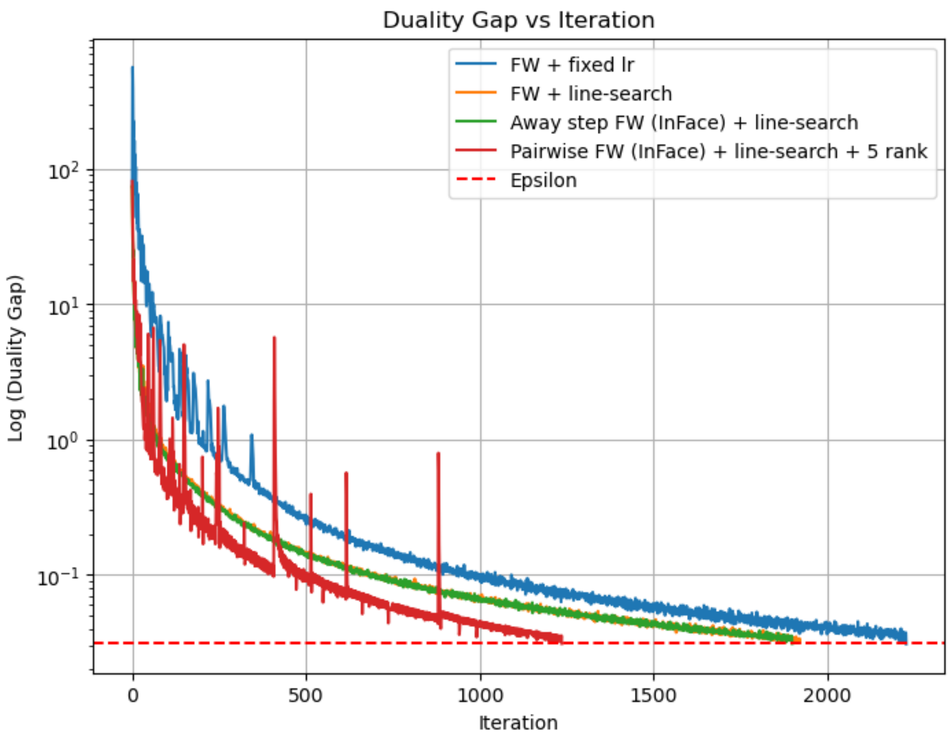

Figure 7: Plot of running solvers on the MovieLens 1M Dataset with $\epsilon = 10^{-1.5}$ related to table 3

B Observation of Fluctuations

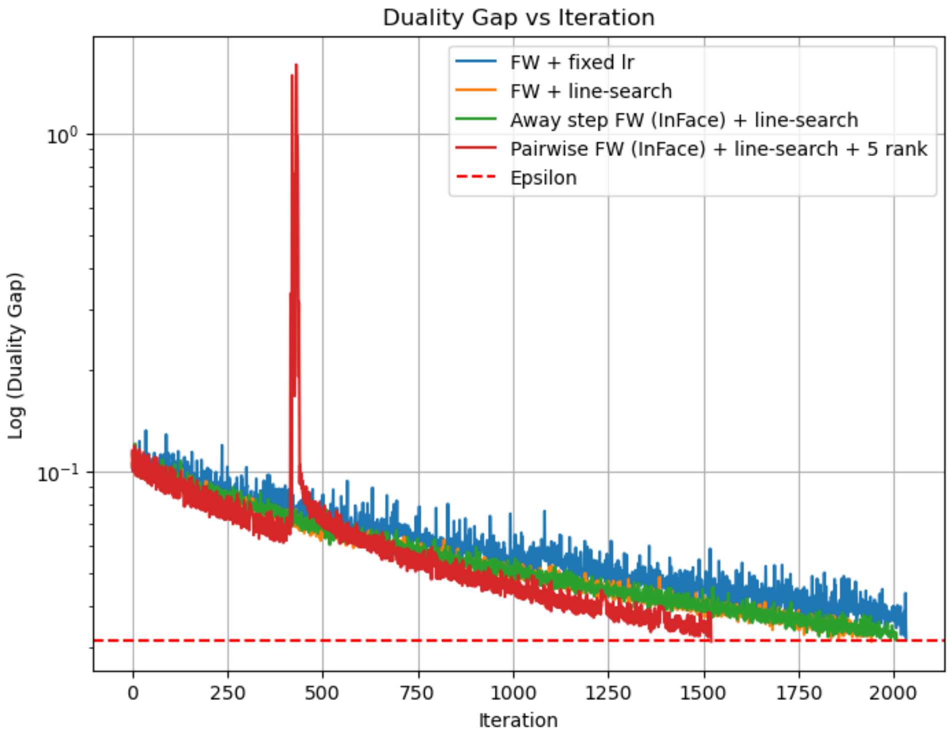

Figure 8: Plot of running solvers on the MovieLens 100k Dataset with $\epsilon = 10^{-1.5}$ related to table 1